Towards a More Intuitive Specification and Automatic Verification of System Properties with FBT: A Tool for Translating Interval Formulas into Büchi Automata

Miguel J. Hornos

Dpto. de Lenguajes y Sistemas Informáticos. University of Granada
E.T.S. Ingeniería Informática, Periodista Daniel Saucedo, s/n. 18071 Granada, Spain

Abstract. This paper presents the FBT (FIL to Büchi automaton Translator) tool which automatically translates any formula from FIL (Future Interval Logic) into its semantically equivalent Büchi automaton. There are two advantages of using this logic for specifying and verifying system properties instead of other more traditional and extended temporal logics, such as LTL (Linear Temporal Logic): firstly, it allows a succinct construction of specific temporal contexts, where certain properties must be evaluated, thanks to its key element, the *interval*; and secondly, it also permits a natural, intuitive, graphical representation. The underlying algorithm of the tool is based on the *tableau method* and is specially intended for application in *on-the-fly model checking*. In addition to a description of the design and implementation structure of FBT, we also present some experimental results obtained by our tool, and compare these results with the ones produced by another tool of similar characteristics (i.e. based on an *on-the-fly tableau* algorithm), but for LTL.

1 Introduction

The FBT (FIL to Büchi automaton Translator) tool presented in this paper automatically translates a formula from FIL (Future Interval Logic) [11] into its semantically equivalent Büchi automaton. The underlying algorithm [6] is based on the *tableau method* [13], and is specially intended to be applied to *on-the-fly model checking*. Until very recently, the integration of both approaches (*tableau* and *on-the-fly*) for an interval logic was considered unfeasible for this type of logic. For this reason, we consider FBT to be not only an innovation but also an important achievement.

Traditional temporal logics, such as LTL (Linear Temporal Logic) [10], allow reasoning about the relative ordering of events in a system. However, we must formulate quite intricate expressions with them in order to describe a temporal context in which certain requirements or properties must only be satisfied within it. This, together with the fact that these logics do not have an intuitive representation, such as a semantically equivalent graphical notation, has lead many system designers to believe that they are difficult to use as formal description languages for the requirement specification of systems, and that many of the specifications created with them are formulas which are too complicated to understand. All of this has hindered a more extended

J. Hornos M. (2004).
Towards a More Intuitive Specification and Automatic Verification of System Properties with FBT: A Tool for Translating Interval Formulas into Büchi Automata.
In *Proceedings of the 2nd International Workshop on Verification and Validation of Enterprise Information Systems*, pages 26-33
DOI: 10.5220/0002680600260033

use of the mentioned logics as specification languages during the analysis phase of the development cycle of industrial applications. Unlike these logics, the formal specification language FIL which our tool uses, allows the succinct construction of bounded temporal contexts, thanks to its key element, the *interval*, which defines such contexts clearly and concisely. In addition to the textual representation of its formulas, this logic also has a natural, intuitive, graphical representation, called GIL (Graphical Interval Logic) [1], and both the textual and graphical representation are semantically equivalent.

This paper is organized as follows. Section 2 introduces the textual syntax of FIL and the graphical syntax of GIL, and also the semantics associated with these. Section 3 describes the design structure and implementation of FBT, while Section 4 presents some of the experimental results obtained, and compares these with the results obtained with another similar tool (i.e. based on an *on-the-fly tableau* algorithm), but with LTL as the specification formalism. Finally, we present the conclusions and related lines of research which we would like to follow in the future.

2 Specification Formalism

Since our purpose is to specify the temporal properties of a system in a formalism that is very close to the way in which a human being reasons, we shall use a propositional, linear-time temporal logic, which uses an intuitive, graphical representation for its formulas. This logic, called GIL (Graphical Interval Logic) [1], allows us to carry out logical reasoning at the level of time intervals instead of instants. Nevertheless, its primitive elements are instants. An interval is therefore formed by identifying its end-points, which are instants satisfying certain properties. These points are searched for in the global context representing an infinite sequence of states corresponding to a system execution. Once the end-points of an interval have been located, the semantics of the nested formula (to the interval) is restricted to the subtrace delimited by these points. Each interval therefore represents a specific temporal context.

The first three graphical formulas in Figure 1 show the basic types of properties that can be expressed over an interval. Thus, (a) is an *initial property*, which states that the formula *f* expressing such a property holds at the first state of the interval, where *f* is drawn left-justified below its left end-point; (b) represents an *invariant property* over the interval, where *f* is placed below it and indented to the right of its left end-point to express that *f* holds at every state of the interval; and (c) give us an *eventuality property* stating that *f* eventually holds at some state within the interval, where a diamond is placed on it with *f* left-justified below the diamond.

In the formulas explained, *f* can be any GIL formula, even another interval formula, and the intervals in them can represent the global context or a subinterval extracted from a larger interval. Each end-point of a subinterval is defined by a *search pattern* represented by a horizontal concatenation of dashed search arrows, where search targets are left-justified below the arrowheads. Thus, the formula (d) in Figure 1 states that *i* is an initial property in the subinterval extracted from the global context by using a first search that locates the earliest state at which *f* holds (its left end-point), and from this, the right end-point is located by searching for the formula *h* from the state where *g* is found.

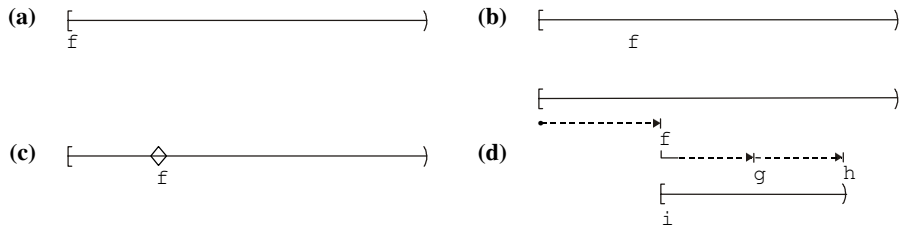

Fig. 1. GIL formulas expressing different types of properties

Every GIL formula is read from top to bottom and from left to right. The topmost interval represents the global context. Formulas can be composed using the usual infix Boolean operators laid out horizontally or vertically. Further details about the visual syntax of GIL can be found in [1].

FIL (Future Interval Logic) [11] is the textual representation used for defining the semantics of GIL. The equivalence between the textual formulas of FIL and the corresponding graphical formulas in GIL is established in [1]. The *syntax of FIL* for a finite set $\mathscr{P}$ of primitive propositions, where $p \in \mathscr{P}$, is defined as follows:

f ::== $p \mid \neg f \mid f_1 \vee f_2 \mid I f$ /* FIL formulas */

I ::== $[\theta_1|\theta_2) \mid [-|\theta_2) \mid [\theta_1|\rightarrow)$ /* Intervals */

θ ::== $\rightarrow f \mid \rightarrow f,\theta$ /* Search patterns */

A FIL formula is *purely propositional* when it does not contain any interval. Otherwise, it is an *interval formula*, with its structure given by $I f$, where I represents an interval and f represents any other FIL formula nested to it. All the intervals are half-open, including their left but not their right end-point. Each interval end-point is defined by a *search pattern*, which is either a sequence of one or more searches or a *trivial* pattern (represented by – or →). Each *search*, e.g. $\rightarrow f$, locates the first point in the reflexive future (which includes the current state) where the target formula f holds. When several searches are sequentially composed, such as in $\rightarrow g,\rightarrow h$, each subsequent search begins in the state located by the previous search; the last of these therefore locates the end-point of the interval that such a pattern defines. The *trivial search pattern* – leaves us at the point where we are, while → takes us to the end of the current context.

The other standard constructs of Propositional Logic are defined as abbreviations of certain expressions, i.e. $\mathbf{T}=p\vee\neg p$, $\mathbf{F}=\neg\mathbf{T}$, $\neg\neg f = f$, $f_1 \wedge f_2 = \neg(\neg f_1 \vee \neg f_2)$ and $f_1 \Rightarrow f_2 = \neg f_1 \vee f_2$. The *restricted* syntax presented above for FIL can be extended with several LTL temporal operators, defined as other abbreviations. Thus, since the logical constant **F** can only hold in the null context, the formula $[\rightarrow\neg f\,|\rightarrow)\mathbf{F}$ can never therefore be satisfied in a trace in which $\neg f$ holds at some state, with this formula being equivalent to $\Box f$. Its dual, $\neg[\rightarrow f\,|\rightarrow)\mathbf{F}$, states that there is some instant in the future where f holds, which is why it is equivalent to $\Diamond f$. The operator *strong until* is defined as $f_1 \mathrm{U} f_2 = \neg[\rightarrow(\neg f_1 \vee f_2)|\rightarrow)\neg f_2$. The complete FIL formal semantics can be found in [11]. The FIL formulas corresponding to the GIL ones in Figure 1 are: (a) f, (b) $\Box f$, (c) $\Diamond f$, and (d) $[\rightarrow f|\rightarrow f,\rightarrow g,\rightarrow h)\, i$.

3 Design and Implementation of FBT

Figure 2 describes the structure of the different classes of FIL formulas considered in the design and implementation of FBT, using a class diagram in UML (Unified Modeling Language) [12]. *Fil* is an abstract class that defines the elements which are common to the different types of formulas, defined in its subclasses, which are: *FilAtom*, the class that represents the literals (atomic propositions, negated or not); *FilConstant* stands for the logical constants (**T** or **F**); *FilJunct* implements the conjunctions and disjunctions of (two) FIL formulas, while *FilIff* does something similar, but with equivalences and exclusive disjunctions; and *FilInterval*, the class that builds interval formulas, comprising two search patterns which attempt to locate each endpoint of the interval, and a FIL formula nested to it. An instance of the class *SearchPattern* is a sequence of zero (if it is a trivial pattern) or more FIL formulas, as many searches as comprise that pattern.

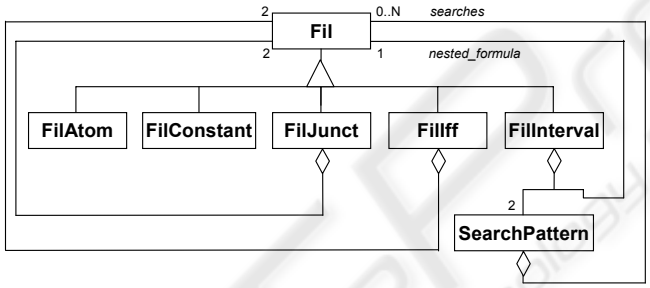

Fig. 2. UML class diagram representing the different types of FIL formulas

The UML class diagram shown in Figure 3 describes the structure of the graph that FBT generates from a FIL formula. *FilGraph* is therefore the class representing the graph, which is formed by the aggregation of nodes (i.e. objects of class *FilGraphNode*). Two nodes of the graph are related if a transition exists between them. This has been represented by means of an association with two role names: *predecessor* and *successor*. The structure of each node is constituted for the aggregation of the following sets of (zero or more) FIL formulas:

- *New*: Temporal properties that must hold in the node and have not yet been processed. When a node has been processed completely, this set is empty.
- *Old*: Formulas that must hold in the node and have already been analysed.
- *Next*: Temporal properties that must be satisfied in all the next nodes (i.e. states which are its immediate successors). This set can only contain interval formulas, since these are the only FIL formulas that can postpone their fulfilment.
- *Literals*: Literals stored in *Old*.

The implementation of FBT is based on the C++ code of LBT (LTL to Büchi automaton Translator) [8], a tool of similar characteristics, since it is based on an on-the-fly tableau algorithm [3], but with LTL as the specification formalism. Consequently, both tools share the same input and output interface, and the syntax and notation for all the formula components that are commonly accepted by both, i.e. the

logical constants, literals, propositional operators and temporal operators of LTL. Our initial intention was to integrate FBT into MARIA [9], a tool that performs on-the-fly model checking. This was the main reason for reusing part of the LBT code in order to make integration easier, since LBT is the translator used in MARIA. Obviously, we have had to incorporate a series of specific classes and functions for the analysis of interval formulas, which are not present in LTL. We have also implemented a series of heuristics in order to improve and optimise the code; we have therefore managed to generate automata with fewer nodes and edges and in less time, and to determine the accepting states more quickly. The FBT input and output interface as well as additional details about the tool design and implementation can be found in [5].

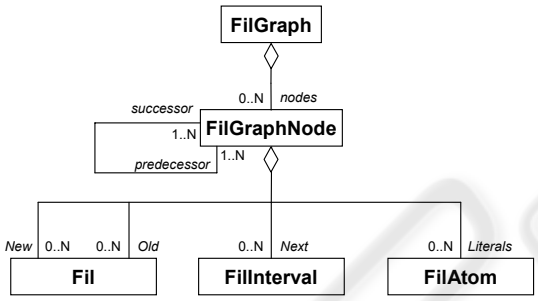

Fig. 3. Structure in UML of the graph generated from a FIL formula

4 Experimental Results

As FBT has been implemented from the LBT code, this section not only presents some experimental results obtained with our tool, but also those generated by LBT for the same or equivalent specifications, in order to compare both tools.

FBT recognizes not only the unique temporal operator of FIL, i.e. the interval, but also the following temporal operators of LTL: $\Box$ (*always* or *henceforth*) and its dual $\Diamond$ (*eventually*), and U (*until strong*) and its dual V (*release*), but as abbreviations of the corresponding FIL interval formulas. FBT therefore accepts LTL formulas, such as the one shown in the following example, but automatically transforms them into their equivalent ones in FIL, which are the ones that it really stores and processes.

Example 1: For the input specification $\Diamond p0 \wedge \Diamond p1$, FBT processes the FIL formula $\neg[\rightarrow p0|\rightarrow)_F \wedge \neg[\rightarrow p1|\rightarrow)_F$ in order to build its semantically equivalent automaton, which is graphically represented in Figure 4 using the tool GRAPHVIZ [2]. It should be noted that each edge is labelled with the conjunction of literals (in prefix notation) that enables that transition. The upper number labelling each node is its state identifier, while the lower numbers identify the acceptance conditions that it satisfies. The initial state is always shaded and numbered 0. In order to compare the automata generated by FBT and LBT, we count the number of nodes, edges and acceptance conditions, as well as the number of states that satisfy each condition. Since both tools produce *generalized Büchi automata*, the resulting automaton generally have k acceptance conditions; each one defines a set of accepting states, F_i (with i=1..k), which

contains those states that satisfy it. The automaton in Figure 4 has 9 nodes (the initial node is not considered), 20 edges (the ones leaving the initial node are counted), and two acceptance conditions (k=2), the first (identified by the number 0) is satisfied by six states (nodes 3, 4, 5, 7, 8 and 13), while the second one (identified by the number 1) is satisfied by another six (nodes 3, 4, 7, 11, 13 and 14). When k>1, if we want to obtain a *classic* Büchi automaton, i.e. one with only one set of accepting states, F, we only need to obtain the states that are in the intersection of the sets F_i (i.e. $F=\cap F_i$). There are therefore four accepting states in our example (nodes 3, 4, 7 and 13).

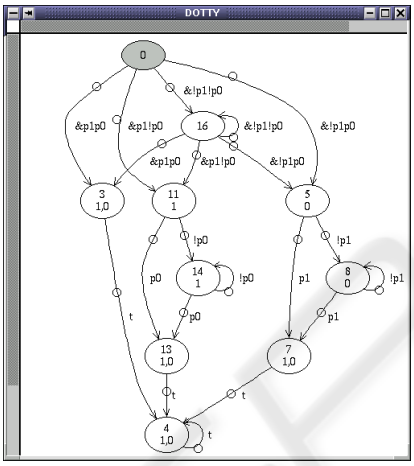

Fig. 4. Büchi automaton generated by FBT for the formula $\Diamond p0 \wedge \Diamond p1$

Table 1 gathers the values counted in the automata generated by FBT and LBT for various specifications. The one explained in Example 1 is shown in Case 5. Each case occupies two rows: the first row corresponds to the results obtained with LBT, and the second row is for those produced by FBT. It should be noted that in the second one, the input specification is represented in the extended syntax of FIL, while the processed formula is expressed in its restricted syntax. We can observe that for the simplest specifications (Cases 1 and 2), we obtained the same values in the automata generated by both tools. However, for slightly more complex specifications (Cases 4, 5 and 6), FBT generally produces smaller automata than those obtained with LBT. The formulas containing either the operator `U` or its dual `V` are usually the exception to this rule, since for these, LBT usually produces automata which are slightly simpler than those generated by FBT (Case 3). This is due to the fact that the corresponding interval formula which is analysed by FBT attempts to "simulate" the property that these operators represent more naturally. Something similar happens in the opposite way with interval formulas: since LBT does not admit interval formulas, it is necessary to resort to appreciably more complicated expressions (Case 7) in order to provide it with an equivalent LTL formula, as Example 2 explains.

Example 2: Since LBT does not recognize the formula $[\rightarrow p0|\rightarrow p1)\Box\neg p2$, we must input an equivalent LTL formula. The semantics associated with this interval formula indicates that it holds whenever one of the following four conditions is fulfilled:

1. p0 does not hold in the future, i.e. the LTL formula $\Box\neg p0$ holds.
2. p1 never holds in the future, i.e. $\Box\neg p1$ is true.
3. p1 precedes p0, which is expressed as $p1 \mathbb{V} \neg p0$ in LTL. This formula asserts that in the first state where p1 holds as well as in all the previous ones to it $\neg p0$ holds.
4. Either p0 and p1 hold in the same state or p0 holds strictly before p1 and (in this case) $\neg p2$ is invariantly satisfied from the instant in which p0 holds until p1 is fulfilled. Both conditions are formulated in LTL as $\neg p1 \,\mathbb{U}\, (p0 \wedge \neg p2 \,\mathbb{U}\, p1)$.

Consequently, the equivalent formula to $[\rightarrow p0|\rightarrow p1)\Box\neg p2$ that must be inputted into LBT is the disjunction of the previous four LTL formulas, i.e. $\Box\neg p0 \vee \Box\neg p1 \vee p1\mathbb{V}\neg p0 \vee \neg p1 \,\mathbb{U}\, (p0 \wedge \neg p2 \,\mathbb{U}\, p1)$. The interval formula clearly expresses the explained property more concisely and elegantly. Moreover, the analysis carried out by FBT produces a simpler automaton (see Case 7 in Table 1).

Table 1. Comparison of results obtained with LBT and FBT

	Specification		Size		Accepting states		
	Input	Processed	Nodes	Edges	k	$\lvert F_i\rvert$	$\lvert\cap F_i\rvert$
1	$\Box\neg p0$		1	2	0		
	$\Box\neg p0$	$[\rightarrow p0\lvert\rightarrow)_{\mathbf{F}}$	1	2	0		
2	$\Diamond p0$		3	6	1	2	
	$\Diamond p0$	$\neg[\rightarrow p0\lvert\rightarrow)_{\mathbf{F}}$	3	6	1	2	
3	$p1 \,\mathbb{U}\, p2$		3	6	1	2	
	$p1 \,\mathbb{U}\, p2$	$\neg[\rightarrow(\neg p1 \vee p2)\lvert\rightarrow)\neg p2$	4	9	1	3	
4	$\Diamond\Diamond p1$		6	13	2	4, 5	3
	$\Diamond\Diamond p1$	$\neg[\rightarrow\neg[\rightarrow p1\lvert\rightarrow)_{\mathbf{F}}\lvert\rightarrow)_{\mathbf{F}}$	4	8	2	3, 3	2
5	$\Diamond p0 \wedge \Diamond p1$		13	29	2	8, 8	4
	$\Diamond p0 \wedge \Diamond p1$	$\neg[\rightarrow p0\lvert\rightarrow)_{\mathbf{F}} \wedge \neg[\rightarrow p1\lvert\rightarrow)_{\mathbf{F}}$	9	20	2	6, 6	4
6	$\Box\Diamond p1 \Rightarrow \Box\Diamond p2$		9	19	2	7, 7	5
	$\Box\Diamond p1 \Rightarrow \Box\Diamond p2$	$\neg[\rightarrow[\rightarrow p1\lvert\rightarrow)_{\mathbf{F}}\lvert\rightarrow)_{\mathbf{F}} \vee [\rightarrow[\rightarrow p2\lvert\rightarrow)_{\mathbf{F}}\lvert\rightarrow)_{\mathbf{F}}$	5	15	3	4, 3, 4	2
7	$\Box\neg p0 \vee \Box\neg p1 \vee p1\mathbb{V}\neg p0 \vee \neg p1 \,\mathbb{U}\, (p0 \wedge \neg p2 \,\mathbb{U}\, p1)$		17	33	2	15, 14	12
	$[\rightarrow p0\lvert\rightarrow p1)\Box\neg p2$	$[\rightarrow p0\lvert\rightarrow p1)[\rightarrow p2\lvert\rightarrow)_{\mathbf{F}}$	12	25	2	9, 9	7

5 Conclusions and Future Work

In this paper, we have presented the FBT tool, which is specially intended to be applied to the automatic verification of systems, using the on-the-fly model checking method and FIL formulas. We have adopted this logic as the specification formalism of our tool for two reasons: firstly, its ability to express succinctly limited temporal contexts in which certain properties must be satisfied; and secondly, its natural, intuitive, graphical representation, which makes the specifications easier to develop and understand.

We have also included class diagrams illustrating the design and implementation structure of FBT, and some experimental results which we have compared with the results obtained with LBT for the same or equivalent formulas. The automata generated by both translators are of a similar complexity, but those produced by FBT are slightly simpler in most of the analyzed cases. As a good specification formalism is

the one that describes the most frequently used properties in verification with specifications that are relatively short and not difficult to check in practice, we can conclude that FIL is a good specification formalism and that FBT is a good tool for the efficient translation of its formulas into Büchi automata.

In future work, we intend to supply FBT with a graphical editor for GIL formulas, so that the specifications can be pictorially provided, instead of in the textual syntax of FIL. GILED [7] is an editor of this type that automatically translates the graphical specifications created with it into the corresponding FIL formulas. The idea is to adapt this editor or to build one of similar characteristics for FBT. We also intend to integrate our translator into an on-the-fly model checking tool. Although FBT has been designed so that it may be easily incorporated into the model checker of MARIA [9], it can also be adapted to be integrated into other more popular finite-state verification tools such as SPIN [4].

References

1. Dillon, L.K., Kutty, G., Melliar-Smith, P.M., Moser, L.E. Ramakrishna, Y.S.: A Graphical Interval Logic for Specifying Concurrent Systems. ACM Transactions on Software Engineering and Methodology, 3, 2 (1994) 131–165
2. Gansner, E.R., Koutsofios, E., North, S.C., Vo, K.-P.: A Technique for Drawing Directed Graphs. IEEE Transactions on Software Engineering, 19, 13 (1993) 214–230
3. Gerth, R., Peled, D., Vardi, M.Y., Wolper, P.: Simple On-the-fly Automatic Verification of Linear Temporal Logic. Proceedings of the 15th International Symposium on Protocol Specification, Testing and Verification, Warsaw, Poland. Chapman & Hall (1995) 3–18
4. Holzmann, G.J.: The SPIN Model Checker: Primer and Reference Manual. Addison-Wesley, Boston (2003)
5. Hornos, M.J.: Tool Design and Implementation. In: From Interval Logic Specifications to Property Automata: A Tableau Construction for Application to On-the-fly Model Checking. Chapter 6, PhD. Thesis, University of Granada (2002) 153–182 (in Spanish)
6. Hornos, M.J., Capel, M.I.: On-the-fly Model Checking from Interval Logic Specifications. ACM SIGPLAN Notices, 37, 12 (2002) 108–119
7. Kutty, G., Dillon, L.K., Moser, L.E., Melliar-Smith, P.M., Ramakrishna, Y.S.: Visual Tools for Temporal Reasoning. Proceedings of the IEEE Symposium on Visual Languages, Bergen, Norway (1993) 152–159
8. Mäkelä, M.: LBT: LTL to Büchi Conversion. http://www.tcs.hut.fi/Software/maria/tools/lbt/
9. Mäkelä, M.: Maria: Modular Reachability Analyser for Algebraic System Nets. Proceedings of the 23rd International Conference on Application and Theory of Petri Nets, Adelaide, Australia. Lecture Notes in Computer Science, Vol. 2360, Springer-Verlag (2002) 434–444
10. Manna, Z., Pnueli, A.: The Temporal Logic of Reactive and Concurrent Systems: Specification. Springer-Verlag, New York (1992)
11. Ramakrishna, Y.S., Dillon, L.K., Moser, L.E., Melliar-Smith, P.M., Kutty, G.: Interval Logics and Their Decision Procedures. Part I: An Interval Logic. Theoretical Computer Science, 166, 1–2 (1996) 1–47
12. Rumbaugh, J., Jacobson, I., Booch, G.: The Unified Modeling Language Reference Manual. Addison-Wesley, Reading (1999)
13. Wolper, P.: The Tableau Method for Temporal Logic: An Overview. Logique et Analyse, 110–111 (1985) 119–136