

# An Optimized CLP-based Technique for Generating Propagation Sequences\*

F. Fummi V. Guarnieri C. Marconcini G. Pravadelli

Dipartimento di Informatica - Università di Verona, Strada le Grazie 15, 37134, Verona, Italy

{franco.fummi, valerio.guarnieri, cristina.marconcini, graziano.pravadelli}@univr.it

## Abstract

The paper presents a constraint logic programming-based methodology to generate propagation sequences for functional faults by traversing extended finite state machines. Moreover, different strategies are presented to deal with the state explosion problem arising when constraint logic programming is adopted for solving hard problems. Experimental results show the effectiveness of the proposed solutions.

## 1. Introduction

High-level automatic test pattern generators (ATPGs) based on simulation [1, 2] are fast. However, they are unable to cover corner cases, and they cannot prove untestability. On the contrary, high-level ATPGs exploiting formal methods [3, 4], being exhaustive, cover corner cases. On the other side, they tend to suffer of the state explosion problem when adopted for verifying large designs. In this context, the paper proposes a methodology for addressing hard-to-detect faults when a high-level ATPG is applied to verify functional descriptions of sequential circuits. In particular, the proposed approach is oriented to fault propagation that is, generally, the most critical phase of test generation.

The proposed fault propagation procedure relies on:

- the extended finite state machine (EFSM) model [5] to represent the design under verification (DVU), since such a model allows a more compact representation of the state space than traditional FSMs;
- constraint logic programming (CLP) [6] to deterministically propagate functional faults which are observed, but not detected by traversing the EFSM through the use of different ATPG-based approaches.

CLP is a form of constraint programming in which logic programming is extended to include concepts from constraint satisfaction. In the literature, there are some works which propose techniques based on CLP for generating test sequences for FSMs [7, 8, 9, 10]. However, the existent approaches differ in several aspects from the ones presented in this paper. Mainly, these works do not introduce strategies

\*This work has been partially supported by the European project VER-TIGO FP6-2005-IST-5-033709.

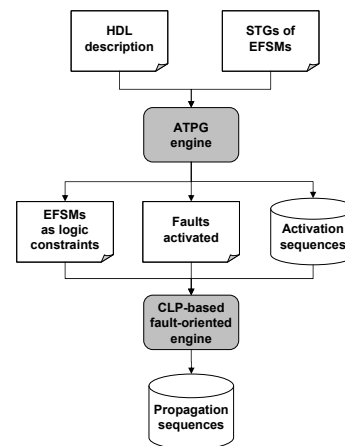


Figure 1. Methodology flow.

for avoiding the risk of state explosion when a CLP solver is asked to generate a test sequence to activate the target path. Also, they do not consider any approach, to deal with the problem of managing the complexity, that is the crucial aspect of adopting formal techniques.

The paper is organized as follows. Section 2 presents the overview of the proposed methodology. Section 3 introduces the EFSM model and it shows how CLP can be used to model an EFSM. Section 4 describes the techniques defined to deal with the CLP complexity. Finally, Section 5 reports experimental results, and Section 6 summarizes the concluding remarks.

## 2. Methodology Overview

Figure 1 presents the methodology flow. A high-level ATPG engine, like for example the one presented in [11], is first used to generate test sequences for functional faults injected into the DVU description. Faults that are activated, i.e., observed on internal registers, but not propagated to primary outputs (in the following, we will call them FTBP, i.e., “faults to be propagated”) are collected to be investigated by using the CLP-based approach. In particular, a fault is said to be activated, if it brings the faulty and fault free EFSMs to different configurations [12]. Then, for every FTBP, a CLP model, describing the faulty and fault-free EFSM configurations, is automatically generated and passed to the CLP-based fault-oriented ATPG engine, whose goal consists of generating propagation sequences for FTBPs. In particular, the CLP model describes a set of constraints which rep-

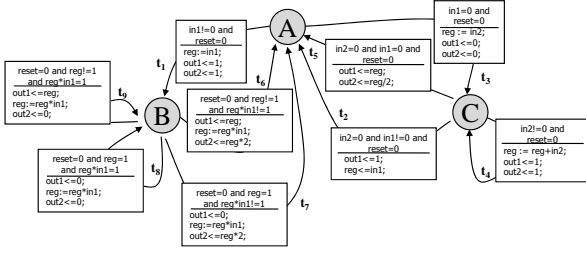


Figure 2. EFSM example.

reset the initial configurations of the faulty and fault-free EFSMs from which the solver has to try the generation of a propagation sequence. However, experimental results have shown that, in several cases, the problem submitted to the CLP solver is intractable by using a reasonable amount of time and memory resources, and the solver is unable to provide a propagation sequence as well as to prove that it does not exist within the given timeout. For this reason, Section 4 presents different approaches to reduce the constraints set, and thus to optimize the search and reduce the CLP solver's effort.

### 3. EFSM modeling in CLP

In this paper, we represent a sequential circuit as a set of concurrent EFSMs, one for each process of the DUV. In this way, according to the formal definition of EFSM given in [11], we capture the main characteristics of state-oriented, activity-oriented and structure-oriented models [13].

The EFSM is a labeled transition system composed of states and transitions. The EFSM differs from the classical FSM, since each transition does not present only a label in the classical form  $(i)/(o)$ , but it is extended with instructions that act also on the DUV registers. Thus, each transition has a source and a destination state, and it is labeled with an *enabling* function  $e$  and an *update* function  $u$ . Given a state  $s$  and a transition  $t$  out-going from  $s$ , the update function  $u$  of  $t$  is executed if its enabling function  $e$  is satisfied. In this case, we say that  $t$  is *fired* by applying the input values that satisfy  $e$ . Figure 2 shows the state transition graph of a simple EFSM. Many EFSMs can be generated starting from the same HDL description of a DUV. However, despite from their functional equivalence, they can be more or less easy to be traversed. Easiness of traversal is a mandatory feature for a computational model used in the test pattern generation, and it is a desirable condition to activate and propagate faults. Thus, in [5], a set of theory-based automatic transformations has been proposed to generate a particular kind of semi-stabilized EFSM ( $S^2$ EFSM). It allows an ATPG to easily explore the state space of the corresponding DUV reducing the risk of state explosion.

The CLP-based fault-oriented engine needs a CLP-based representation of the EFSM and some searching functions to generate the propagation sequences. The CLP-based

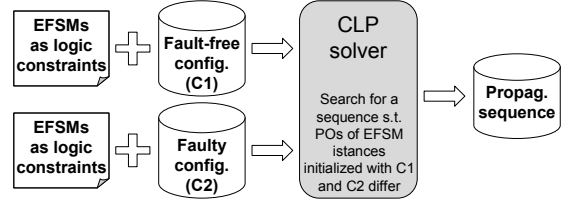


Figure 3. Use of the CLP solver for finding propagation sequences.

representation is automatically derived from the  $S^2$ EFSM model according to the rules reported in [12], which follows the syntax of the ECLiPSe CLP solver [14]. For example, the update function of transition  $t_2$  in Figure 2 is modeled as follows:

$$\begin{aligned}
 ((IN2[T]\# = 1) \text{ and } (\text{neg}(IN1[T]\# = 1)) \text{ and } (C[T])) & \Rightarrow T2[T], \\
 T2[T] & \Rightarrow ((IN2[T]\# = 1) \text{ and } (\text{neg}(IN1[T]\# = 1)) \text{ and } (C[T])), \\
 (T2[T] \text{ and } ((IN2[T]\# = 1) \text{ and } (\text{neg}(IN1[T]\# = 1)) \text{ and } (C[T]))) & \\
 \Rightarrow ((REG[Next.T]\# = REG[T]) \text{ and } (OUT1[Next.T]\# = 1) & \\
 \text{ and } (OUT2[Next.T]\# = IN1[T])) & .
 \end{aligned}$$

The propagation sequence for a fault is generated by providing the ATPG engine with two instances of the CLP-based representation of the DUV. The two DUV instances provided to the CLP solver are exactly equal, but their registers are initialized with different values for the faulty and the fault-free configurations. Such configurations consist of the values of the registers (included the value of the state register) in the faulty and fault-free DUVs at the moment the fault has been observed by using the activation sequence. As an example, consider the following constraints:

```

% ----- FAULT FREE -----
REG[1]\#=5941, A[1]\#=1, B[1]\#=0, C[1]\#=0,
% ----- FAULTY -----
REG.F[1]\#=5941, A.F[1]\#=0, B.F[1]\#=1, C.F[1]\#=0,

```

They state that, at time  $T = 1$ ,  $REG$  has the same value (i.e., 5941) in both the faulty and fault-free DUV instances, but the two  $S^2$ EFSMs are in different states (i.e., the fault-free DUV is in state A, while the faulty DUV is in state B).

After the the faulty and fault-free configurations set-up, the CLP-solver is asked to find a sequence starting from such configurations, propagates the effect of the fault towards the POs (see Figure 3). If the solver finds a solution, it consists of a propagation sequence that can be appended to the activation sequence which allows us to observe the target fault on the internal registers.

### 4. Optimization techniques

The typical problem of formal verification techniques is dealing with the issue of complexity. In fact, tools that exhaustively search for a solution of NP-hard problems, like searching for a propagation sequence in a sequential circuit, frequently run out of resources when the state space to be analyzed is too large. The same happens for the CLP solver used in the methodology presented in Figure 1, when it is asked to find a propagation sequence on large sequential designs. To limit such a problem, heuristics is generally used

for pruning the state space. However, this may prevent the solver from finding a solution (even if it exists), if the pruning is too restrictive. Thus, choosing a good heuristics is a very challenging task.

Therefore, in this section, we propose three strategies to manage the complexity of generating a propagation sequence on an EFSM by means of a CLP solver.

#### 4.1. Removal of useless transitions

The first optimization consists of a manipulation phase that removes constraints that could avoid the propagation of a fault towards POs, from the CLP-based EFSM model. For example, allowing the CLP solver the chance of traversing a transition, whose update function re-initialize a register where the fault was observed. This strategy prevents the generation of a propagation sequence for such a fault. Let us consider the EFSM represented in Figure 2 and a fault  $f$  observed on register  $reg$ . Moreover, let us suppose that in the faulty configuration  $reg$  is 0 and the EFSM is in the state A, while in the fault-free configuration  $reg$  is 2 and the EFSM is in the state A. Then, if transition  $t_3$  was traversed, a new value would be assigned to  $reg$ , losing the possibility of propagating the faulty configuration to the POs. In fact, if the update function of transition  $t_3$  was executed, the faulty and fault-free configurations would be equivalent. Therefore, given each FTBP, an algorithm has been defined to automatically remove this kind of transitions from the EFSM model, thus reducing the state space that the solver must analyze. The complexity of such a pruning is linear with respect to the number of transitions.

#### 4.2. Removal of isolated states

According to the removal of transition described in the previous subsection, some parts of the EFSM can remain isolated. Thus, the states of the EFSM that cannot be reached from the initial configuration are removed. Let us consider, for example, the EFSM of Figure 2, and let us assume that the activation sequence of a fault  $f$  lead the EFSM to state B. Starting from B, it is not possible to generate a sequence to reach state C. Therefore, state C and all its out-going transitions can be removed. Starting from Figure 2, the EFSM obtained after the reduction process described in Section 4.1 and the removal of isolated states proposed in this paragraph is depicted in Figure 4.

#### 4.3. Pruning based on paths

The CLP solver try to find a propagation sequence starting from the configuration that activates the fault. Such effort can be reduced if the solver is provided with a hint about the paths that are more profitable to be traversed. In this case, no constraints are actually removed from the CLP-based representation of the EFSM, but we provide the solver with part of the solution, thus reducing the number of satisfiable constraints.

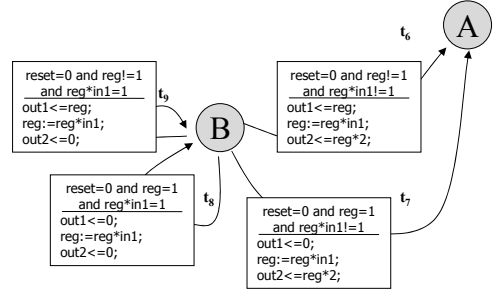


Figure 4. Optimized from configuration's state B.

Before invoking the solver to generate the propagation sequence, the ATPG applies an algorithm that searches for paths connecting the fault-free EFSM configuration with a state where at least one of update functions of its out-going transitions writes on the primary outputs. In fact, if a path allows to update the primary outputs, it is probable that the register value of the faulty configuration would be propagated on such outputs too. Then, all constraints related to the transition that are not involved in the generated paths are removed. These paths are generated such that they include transitions which allow the propagation of the values of registers involved in the faulty configuration towards the POs. These transitions are identified and marked by learning phases performed during the EFSM generation and the subsequent removal of useless transitions. In particular, for every transition  $t$  marked as useful, a path is generated from the current configuration state to the out-going state of transition  $t$ . The solver would try to check if the constraints are satisfiable and it would find a solution. If either it is not able to find a propagation sequence within a given timeout, or it returns that the problem is non satisfiable, a different path is generated and passed to the solver as initial configuration. Future works are related to associate a weight to each transition to generate paths maximizing the total weight of the involved transitions. Possible ideas for weighting transitions are: preferring transitions leading to a state whose out-going transitions update the primary outputs, or transitions whose update functions use a large number of faulty registers, or transitions whose enabling functions consist of “small” conditions, etc..

## 5. Experimental results

The effectiveness of the proposed methodology and of the related optimization strategies have been evaluated on a set of well known benchmarks, *b04*, *b11* and *b10*, selected from the ITC-99 suite [15], and *Prawn* that is an enhanced version of the Parwan RISC processor with the instruction set having been enhanced to include interrupt handling and conditional branches.

Experimental results are summarized in Table 1. Columns *FFs*, *Gates*, *State*, *Trans* report, respectively, the number of the flip-flops and gates of the considered benchmarks, and the number of states and transitions of the corresponding EFSMs. Column *FTPB* shows the number of

**Table 1. Experimental results.**

DUV	FFs	Gates	States	Trans	FTPb	TOut (sec.)	No Optimization			Optimized		
							PSEQ	Aborted	Time (sec.)	PSEQ	Aborted	Time (sec.)
b04	66	650	3	20	11	10	11	0	8.01	11	0	10.89
b10	17	264	11	42	23	10	23	0	26.08	23	0	7.43
b11	31	715	9	29	50	10	50	0	63.17	50	0	16.68
prawn	84	2064	61	160	66	14	0	66	998.12	66	0	105.89

faults to be propagated which have been activated by using the ATPG proposed in [11]. Column *TOut (sec.)* shows the timeout provided to the CLP solver for finding a propagation sequence. Columns *PSEQ*, *Aborted* and *Time (sec.)* under *No optimization* shows, respectively, the number of propagation sequences generated by the CLP solver, the number of faults aborted (i.e., the number of faults for which the solver was unable to provide a response, either positive or negative), and the total time required for generating the CLP constraints to model the EFSM and running the search. The same values have been computed after applying the optimization techniques presented in Section 4 (Columns under *Optimized*).

Experimental results show that the proposed optimization techniques sensibly improve the effectiveness of the solver in searching for propagation sequences. The improvement is particularly evident in the case of *Prawn*, whose EFSM is very large. Without optimizations the solver always aborted, while after optimizations was applied, it succeeded in generating propagation sequences for all the FTPBs. Moreover, it can be observed that the sequence generation time was sensibly decreased, for all benchmarks, but *b04*. In the case of *b04*, optimizations did not provide benefits, since its EFSM is composed of very few states that cannot be further reduced by applying the proposed optimizations.

## 6. Concluding remarks

In this paper we have proposed a CLP-based methodology to generate propagation sequences for hard-to-detect faults. Moreover, we have defined different strategies to limit the explosion problem arising when a CLP solver is adopted for traversing an EFSM. Experimental results have highlighted the effectiveness of the proposed approaches.

## References

- [1] A. Fin and F. Fummi, "Genetic algorithms: the philosophers stone or an effective solution for high-level TPG?" in *Proc. of IEEE HLDVT*, 2003, pp. 163–168.
- [2] F. Corno, G. Cumani, M. S. Reorda, and G. Squillero, "Effective techniques for high-level atpg," in *Proc. of IEEE ATS*, 2001, pp. 225–230.
- [3] I. Ghosh and M. Fujita, "Automatic test pattern generation for functional register-transfer level circuits using assignment decision diagrams," *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems*, vol. 20, no. 3, pp. 402–415, 2001.
- [4] L. Zhang, I. Ghosh, and M. Hsiao, "Efficient Sequential ATPG for Functional RTL Circuits," in *Proc. of IEEE ITC*, 2003, pp. 290–298.
- [5] G. Di Guglielmo, F. Fummi, C. Marconcini, and G. Pravadelli, "EFSM Manipulation to Increase High-Level ATPG Efficiency," in *Proc. of IEEE ISQED*, 2006, pp. 57–62.
- [6] J. Jaffar and M. J. Maher, "Constraint logic programming: A survey," *Journal of Logic Programming*, vol. 19/20, pp. 503–581, 1994.
- [7] C. Pauli, M. L. Nivet, and J. F. Santucci, "Use of constraint solving in order to generate test vectors for behavioral validation," in *Proc. of IEEE HLDVT*, 2000, pp. 15–20.
- [8] R. Vemuri and R. Kalyanaraman, "Generation of design verification tests from behavioral vhd1 programs using path enumeration and constraint programming," *IEEE Trans. Very Large Scale Integr. Syst.*, vol. 3, no. 2, pp. 201–214, 1995.
- [9] F. Ferrandi, M. Rendine, and D. Sciuto, "Functional verification for systemc descriptions using constraint solving," in *Proc. of DATE*, 2002, pp. 744–751.
- [10] F. Xin and I. G. Harris, "Test generation for hardware-software covalidation using non-linear programming," in *Proc. of IEEE HLDVT*, 2002, pp. 175–180.
- [11] G. D. Guglielmo, F. Fummi, C. Marconcini, and G. Pravadelli, "Improving high-level and gate-level testing with FATE: a functional ATPG traversing unstabilized EFSMs," *IEE Computers and Digital Techniques*, vol. 1, no. 3, pp. 187–196, 2007.
- [12] F. Fummi, I. G. Harris, C. Marconcini, and G. Pravadelli, "A CLPbased Functional ATPG for Extended FSMs," in *Proc. of IEEE MTV*, 2007.
- [13] D. Gajski, J. Zhu, and R. Domer, "Essential issue in codesign," University of California, Irvine, Technical report ICS-97-26, 1997.
- [14] M. Wallace and A. Veron, "Two problems-two solutions: one system-ECLIPSE," in *IEE Colloquium on Advanced Software Technologies for Scheduling*, 1994, pp. 1–3.
- [15] "High time for high-level test generation," Panel at IEEE ITC, 1999.