New Qualitative Analysis Strategies in Metaprime

Olivier Coudert • Digital Paris Research Laboratory • Rueil Malmaison, France
Jean Christophe Madre • Digital Paris Research Laboratory • Rueil Malmaison, France
Henri Fraisse • Bull Research Center • Les Clayes sous Bois, France

Key Words: Non-coherent and multiple-fault tree analysis, Metaproducts, BDDs, prime covers

SUMMARY & CONCLUSIONS

The fault tree analysis tool Metaprime allows reliability engineers to make, in an interactive way, exact qualitative and quantitative analysis of very complex fault trees that could not be completely treated by previously available analysis technologies. The qualitative aspect of the analysis of a fault tree consists in computing all or part of the set of prime implicants (i.e., minimal cut sets) of this tree. Particularly interesting parts of this set are prime covers which are subsets of prime implicants that are sufficient to represent all the failure combinations of the system under analysis. When the tree is non-coherent, or/and when it is a multiple-fault tree, it is often possible, but very costly, to build covers with fewer prime implicants, which allows concise representation of the set of failure combinations.

This paper proposes original algorithms, now integrated in Metaprime, to generate efficiently prime covers, irredundant prime covers, and minimal prime covers of non-coherent fault trees. Experiments show that these algorithms are robust and provide reliability engineers with efficient strategies for the analysis of complex non-coherent multiple-fault trees.

1 INTRODUCTION

The set of prime implicants (i.e., minimal cut sets) of a fault tree represents all the minimal combinations of elementary failures that make the system fail. A prime cover of a fault tree is a subset of prime implicants that is sufficient to cover all the cases of failures of the system. When the single-fault tree in coherent, all its prime implicants are necessary to cover all the failure cases. But when the fault tree is non-coherent, or/and when it is a multiple-fault tree, it is often possible to build covers with fewer prime implicants, which simplifies the representation of the set of failure combinations, and so to dramatically simplify the analysis of the failure.

Metaprime is a fault tree analysis tool that allows reliability engineers to make exact qualitative and quantitative analysis of very complex fault trees that could not be completely analysed by previously available technologies [6]. This paper addresses the problem of computing prime covers, and proposes algorithms using the properties of Binary Decision Diagrams (BDDs) and metaproducts on which Metaprime is based, to generate efficiently prime covers, irredundant prime covers, and minimal prime covers of non-coherent multiple-fault trees.

The paper is divided into 6 parts. Section 2 gives the notations that will be used. Section 3 presents a prime cover computation algorithm of non-coherent fault trees. Section 4 presents the irredundant prime cover generation algorithm, and Section 5 describes the minimal prime cover computation procedure. Section 6 shows how the analysis of a multiple-fault tree reduces to the analysis of a non-coherent single-fault tree, and gives experimental results obtained with the different metaproduct based procedures presented in the paper.

2 DEFINITION AND NOTATIONS

2.1 FUNCTIONS AND PRIME COVERS

We consider Boolean functions from the set \{0,1\}^n into the set \{0,1,\ast\}. We note \(f^1\) and \(f^1x\) the sets \(f^{-1}(1)\) and \(f^{-1}(1) \cup f^{-1}(\ast)\) respectively. The function \(f\) is said to be completely specified if \(f^{-1}(\ast)\) is empty. An element \(x\) of \{0,1\}^n such that \(f^1(x) = 1\) is called a minterm of \(f\). We will often make no distinction between a function \(f^1\) and the set \(f\), it represents as characteristic function. The Shannon decomposition of a function \(f\) from \{0,1\}^n into \{0,1\} with respect to the variable \(x_k\) is the unique couple of functions that are the restriction of \(f\) to \(x_k = 0\) and \(x_k = 1\) respectively. This couple is noted \(f_{x_k},f_{x_k}^\ast\), and is such that \(f = (f_{x_k}^\ast \land f_{x_k}) \lor (f_{x_k} \land f_{x_k}^\ast)\) [1].

A product is a conjunction of literals, i.e., a propositional variable \(x_k\) or its negation \(\overline{x_k}\). We note \(P_n\) the set of all products built out of the variables \(x_1,\ldots,x_n\). Any set of products \(P\) can be partitioned in the canonical form \(P = P_1 \cup \{P_{x_k} \lor (P_{\overline{x_k}} \land (x_k \lor P_{x_k})\} \cup \{x_k \lor P_{x_k}\}\), where \(P_1\), \(P_{x_k}\), and \(P_{\overline{x_k}}\) are sets of products in which \(x_k\) does not occur. For instance, for \(P = \{x_1x_3, x_2, x_2x_4, \overline{x_2}x_3\}\), we have \(P_2 = \{x_1x_3\}, P_{x_2} = \{x_3\},\) and \(P_{\overline{x_2}} = \{1,x_4\}\).

A product \(p\) is an implicant of the function \(f\) iff \(p \subseteq f^1\),
and it is said to be prime if there is no other implicant of \( f \) that strictly contains it. A set of products \( P \) is a prime cover of \( f \) if it is made of prime implicants of \( f \), and \( f^1 \subseteq (\bigvee_{p \in P}) \subseteq f^{1*} \). This cover is irredundant if there does not exist any proper subset of \( P \) that is a prime cover of \( f \), and it is a minimal prime cover of \( f \) if there does not exist a prime cover of \( f \) that is made of strictly less products than \( P \).

2.2 BDDS AND METAPRODUCTS

The reader is invited to refer to [1, 3] and [4, 6] for detailed descriptions of the Binary Decision Diagram (BDD) and metaproduct representations.

Binary decision diagrams are a graph based canonical representation of Boolean functions. Binary Boolean operators can be evaluated with a quadratic worst case complexity [3], which makes BDDs very different from previously used representations of Boolean functions, for instance the disjunctive normal form for which the operators have different complexities, most of them being at least exponential.

The metaproduct representation is a graph based canonical representation of sets of products [4, 6]. Because there is a complete independence of the size of a metaproduct and of the number of products in the set it represents, metaproducts make possible to perform operations on sets of products with costs related to their sizes and not to the number of manipulated products. The metaproducts of the sets \( P_{1x} \), \( P_{1x} \), and \( P_{xk} \) resulting from the canonical decomposition of the set of products \( P \) with respect to the variable \( x_k \) can be obtained from the metaproduct of \( P \) in constant or linear time. Thanks to this very interesting property, it is very easy to implement with metaproducts the algorithms which manipulate sets of products that use a divide and conquer strategy based on the canonical decomposition defined in Section 2.1.

3 PRIME COVER

In this section we show that the metaproduct based prime implicant computation algorithm presented in [6] can be slightly transformed into an algorithm that produces a prime cover of \( f \). The proof of the following lemmas can be found in [4].

**Lemma 1** All prime implicants of \( f_{xk} \) (respectively of \( f_{xk} \)) that are implicants of \( f_{xk} \) (respectively of \( f_{xk} \)) are prime implicants of \( f_{xk} \), and thus prime implicants of \( f \).

**Lemma 2** \( \text{KeepImplicants}(P, f, 1) \) returns the set of products of \( P \) that are implicants of the Boolean function \( f \).

4 IRREDUNDANT PRIME COVER

This section presents the irredundant prime cover computation algorithm. Its cost is higher than the cost of the
prime cover computation procedure presented section 3, but the prime cover it generates is of better quality, since it is irredundant. This algorithm is based on the recursive computation scheme proposed by E. Morreale in [10] that, like the algorithm PrimeCover, makes use of a divide and conquer strategy based on Shannon decomposition to compute the 3 subsets composing the canonical decomposition of the computed irredundant prime cover with respect to the variable \( x_k \). This algorithm has a cost in \( O(2^{f/1}) \) because at each recursion it can be necessary to create vertices. However experiences show that it is quite robust since it has been able to produce irredundant prime covers for all the real life fault trees that have been analysed so far using Metaprime.

**Theorem 2** \( \text{IrrCover}(f^1, f^{1*}, 1) \) produces an irredundant prime cover of the Boolean function \( f \).

```plaintext
function \( \text{IrrCover}(f^1, f^{1*}, 1) \):
  if \( f^1 = 0 \) return \( \emptyset \);
  if \( f^{1*} = 1 \) return \( \{1\} \);
  let \( g0 = f^1_x \land \neg f^{1*}_x \)
      \( P0 = \text{IrrCover}(g0, f^1_x, k + 1) \)
      \( c0 = \bigcup_{p \in P0} p \)
    \( g1 = f^{1*}_x \land \neg f^1_x \)
    \( P1 = \text{IrrCover}(g1, f^{1*}_x, k + 1) \)
    \( c1 = \bigcup_{p \in P1} p \)
    \( h^1 = (f^1_x \land \neg c0) \lor (f^{1*}_x \land \neg c1) \)
    \( h^{1*} = f^1_x \land f^{1*}_x \)
  return \( P \cup \{(f^1_x) \times P0\} \cup \{(f^{1*}_x) \times P1\} \);  
```

Figure 4. The algorithm \( \text{IrrCover} \).

**Proof.** The proof is made using induction on the number of variables of \( f \). Figure 2.2 shows the different sets used in algorithm \( \text{IrrCover} \).

The function \( f^1 \) represents the set of elements of \( \{0, 1\}^n \) that have to be covered, and the function \( f^{1*} \) represents the set of elements that can be used to build the largest possible cubes. If the function \( f^1 \) is equal to 0, there are no elements to be covered so the irredundant prime cover is empty. If the function \( f^1 \) is not 0 and \( f^{1*} \) is the function 1, then the irredundant prime cover made of the product 1 is sufficient to cover \( f^1 \).

Every prime implicant of the irredundant prime cover \( P0 \) contains at least one element of \( f^1_x \) that is not in \( f^{1*}_x \).Lemma 1 states that none of these prime implicants can be a prime implicant of \( f \), so the literal \( f^1_x \land \neg c0 \) must be added to each of these prime implicants in order to produce prime implicants of the function \( f \). The same reasoning can be done for the prime implicants of the set \( P1 \).

The elements of \( \{0, 1\}^{n-1} \) that still have to be covered are in the set \( h^1 \). This set is the union of the set \( (f^1_x \land \neg c0) \), that contains all the elements of \( f^1_x \) that are not covered by any prime implicant of the set \( P0 \), and of the set \( (f^{1*}_x \land \neg c1) \), that contains all the elements of \( f^{1*}_x \) that are not covered by any prime implicant of the set \( P1 \). The set \( h^1 \) is included in the set \( h^{1*} \), which is the subset of \( \{0, 1\}^{n-1} \) that can be used to build the largest possible cubes that cover the elements of \( h^1 \). The products of the set \( P \) compose an irredundant prime cover of the function defined by \( h^1 \) and \( h^{1*} \), and this set is the last part of the canonical set decomposition of the irredundant prime cover.

\[ \square \]

## 5 MINIMAL PRIME COVER

The minimal prime cover computation problem consists in finding a minimal cardinality subset of prime implicants of a Boolean function \( f \) that is sufficient to cover it [11]. This section presents the basic ideas underlying a new minimal prime cover computation procedure based on BDDs and metaprocdures that overcomes the limitations of other algorithms because its cost is not related anymore to the numbers of minterm and of prime implicants of \( f \). A complete description of this new algorithm can be found in [7].

### 5.1 SET COVERING PROBLEMS

A set covering problem is a triple \( \langle X, Y, R \rangle \) made of two sets \( X \) and \( Y \), and of a relation \( R \) on \( X \times Y \). We say that \( y \) covers \( x \) when \( xRY \). The set covering problem \( \langle X, Y, R \rangle \) consists in finding a minimal cardinality subset of \( Y \) that...
covers all elements of \( X \). Finding a minimal prime cover of a Boolean function \( f \) consists in solving the set covering problem \((f, P, \in)\), where \( P \) is the set of prime implicants of \( f \). The Quine–McCluskey minimization procedure [9] widely used to find minimal prime covers iteratively reduces the sets \( X \) and \( Y \) using the concepts of dominance and essentiality.

### Dominance Relations

A quasi order \( \preceq \) on a set \( Z \) is a reflexive and transitive relation defined on \( Z \times Z \). The relation \( \equiv \) associated with a quasi order \( \preceq \) is defined by \((z \equiv z') \iff ((z \preceq z') \wedge (z' \preceq z))\). The relation \( \equiv \) is an equivalence relation on \( Z \), thus \( \preceq \) is a partial order on \( Z/\equiv \).

Given a set covering problem \((X, Y, R)\), the dominance relations \( \preceq_X \) and \( \preceq_Y \) defined on \( X \) and \( Y \) respectively are:

\[
x \preceq_X x' \iff (\forall y \in Y \ (x'Ry) \Rightarrow (xRy))
\]

\[
y \preceq_Y y' \iff (\forall x \in X \ (xRy) \Rightarrow (xRy'))
\]

Dominance relations are quasi orders that can be used to simplify the set covering problem \((X, Y, R)\) by removing elements from \( X \) and \( Y \). An element \( x' \) of \( X \) dominates \( x \) iff covering \( x' \) is sufficient to cover \( x \), so \( x \) can be removed from \( X \) without modifying the set of minimal solutions. Similarly, an element \( y' \) of \( Y \) dominates \( y \) iff \( y' \) covers at least all the elements of \( X \) covered by \( y \), and so \( y \) can be removed from \( Y \). Let us say that two set covering problems are equivalent if a minimal solution of one problem can be built with the other problem’s one. Then \((X, Y, R)\) is equivalent to \((\max_{\preceq_X} (X/\equiv_X), \max_{\preceq_Y} (Y/\equiv_Y), R)\), where the relation \( R \) is extended on the quotient sets.

### Essentiality

The second reduction rule is based on the essentiality concept. An element \( y \) of \( Y \) is essential iff it is the only one that covers an element \( x \) of \( X \). Since essential elements belong necessarily to any minimal solution, the problems \((X, Y, R)\) and \((X\setminus\{x \in X \mid \exists y \in E \ xRy\}, Y\setminus E, R)\), where \( E \) is the set of essential elements of \( Y \), are equivalent.

### Cyclic Core

The cyclic core of \((X, Y, R)\) is the fixpoint obtained after the two reduction processes presented above have been iteratively applied. In the case when this cyclic core is empty, the minimization is done, and the minimal prime cover is composed of the essential prime implicants that have been successively discovered. In the other case, the minimization can be terminated using a branch-and-bound algorithm [2, 9, 13].

### 5.2 Transposition Functions

In the Quine-McCluskey procedure and its optimizations [2, 9, 12, 13, 15], the classes of the partitions \( X/\equiv_X \) and of \( Y/\equiv_Y \) are manipulated through projections \( \tau \) that map each class onto one of its elements. This section shows that, instead of using these projections, one can use an isomorphism that maps the classes that must be manipulated on objects whose manipulation is less costly.

**Definition 1** Let \( \preceq \) be a quasi order on \( Z \). The function \( \tau \) defined on \( Z \) is a transposition function of \( \preceq \) iff \( z \preceq z' \) is equivalent to \( \tau(z) \subseteq \tau(z') \). In this case, \((Z/\equiv, \preceq)\) and \((\tau(Z), \subseteq)\) are isomorphic through \( \tau \circ \pi \).

\[
(Z/\equiv, \preceq) \quad \mapsto \quad \max_{\preceq}(Z/\equiv)
\]

\[
(Z, \preceq) \quad \mapsto \quad \max_{\subseteq} \tau(Z)
\]

Using the transposition function concept, one can reformulate set covering problems as follows.

**Theorem 3** Let \((X, Y, R)\) be a set covering problem, and \( \tau_X \) and \( \tau_Y \) be two transposition functions of \( \preceq_X \) and \( \preceq_Y \) respectively. Then \((X, Y, R)\) is equivalent to:

\[
(\max \tau_X(X), \max \tau_Y(Y), R')
\]

where \( R' \) is defined by \( \tau_X(x) R' \tau_Y(y) \iff xRy \). The relation \( R' \) is well defined thanks to the definition of transposition functions.

### 5.3 Minimal Prime Cover Computation in Metaprime

Usual minimization procedures aim at solving directly the set covering problem \((f, P, \in)\). The idea of the minimal prime cover computation procedure used in Metaprime consists in considering the equivalent set covering problem \((Q, P, \subseteq)\), where \( Q = \{q \in P_n \mid \exists x \in f^1, q = \{x\}\} \). This problem only involves sets of products and the subset relation. Then two transposition functions \( \tau_Q \) and \( \tau_P \) of the associated dominance relations \( \preceq_Q \) and \( \preceq_P \) are defined so that the reduction process described in Section 5.2 transposes into a set covering problem that still involves products and the subset relation. These transposition functions are fully described in [7]. Sets of products are represented and manipulated with metaproducts, which makes the cost of this new minimal prime cover computation algorithm independent of the number of minterms of \( f \) and its number of prime implicants.

When the cyclic core of the Boolean function \( f \) to be covered is empty, the exact minimal prime cover of \( f \) is found. When the cyclic core of \( f \) is not empty, a call to the algorithm IrrCover produces an irredundant prime cover of
the minterms that remain to be covered to compose the minimal prime cover of \( f \).

6 EXPERIMENTAL RESULTS

A (non-coherent) multiple-fault tree is a logical network that models how \( n \) independent terminal events can combine to make \( m \) top events occur. A cover for a multiple-fault tree is a set of products, built from the \( n \) input variables of the network, that is sufficient to cover all the output Boolean functions \( f_1, \ldots, f_m \) associated with the \( m \) outputs of the network. The way we use here to represent the prime implicants of a multi-output Boolean function \([f_1 \ldots f_m]\) consists in introducing a vector of \( m \) variables \([y_1 \ldots y_m]\), and in defining a partially defined single-output Boolean function from \( \{0,1\}^{n+m} \) into \( \{0,1,\ast\} \), whose set of prime implicants can be trivially mapped onto the set of prime implicants of the multi-output function \([f_1 \ldots f_m]\) [8, 7]. This means that analysing a (non-coherent) multiple-fault tree comes down to analysing a non-coherent single-fault tree.

Table 1 presents the experimental results that have been obtained using the different prime cover computation algorithms presented in this paper. The first problems, namely from \texttt{plkall} to \texttt{bench4}, are non-coherent single-fault trees. The other problems, namely from \texttt{mish} to \texttt{jp}, are non-coherent multiple-fault trees. Column “\#Vars” gives, for each of the treated examples, the number of variables of the function to be covered. For the multiple-fault trees, the figure given in column “\#Vars” is \( n + m \). For each prime cover computation method \( <\text{M}\> \), column \( T<\text{M}\> \) gives the CPU time in seconds to compute on a DEC Station 3000 computer the metaproduct of the prime cover, and \#\(<\text{M}\> \) gives the number of prime implicants (i.e., minimal cut sets) the prime cover is made of. Method \texttt{AP} consists in computing all the prime implicants using the function described in [6]. Method \texttt{PC} consists in computing a prime cover using the function \texttt{PrimeCover}. Method \texttt{IPC} consists in computing an irredundant prime cover using the function \texttt{IrrCover}. Finally method \texttt{MPC} consists in computing a minimal prime cover using the algorithm presented in Section 5. A “\#” in the column “\#\text{MPC}\” indicates that the cyclic core of the Boolean function under treatment was empty so that it was not necessary to use the irredundant prime cover algorithm to terminate the generation of the minimal prime cover.

Experimental results show that in all cases the sizes of the prime covers generated using the algorithms presented here are much smaller than the numbers of prime implicants of the Boolean functions under treatment [6], and that in most of the cases they provide users with a very concise representation of the failure combinations of the system under analysis. Note that in the case when these covers are not very small, it is possible to use the browser of Metaprin [6] to further analyse the resulting sets of prime implicants. In all cases the CPU times needed to compute the minimal prime covers are larger than the CPU times needed to compute the irredundant prime covers, though these irredundant prime covers are in most cases nearly as good as the minimal prime covers.

References

<table>
<thead>
<tr>
<th>Tree</th>
<th>#Vars</th>
<th>#AP</th>
<th>TAP</th>
<th>#PC</th>
<th>TPC</th>
<th>#IPC</th>
<th>TIPC</th>
<th>#MPC</th>
<th>TMPC</th>
</tr>
</thead>
<tbody>
<tr>
<td>plall</td>
<td>42</td>
<td>23200</td>
<td>2.7s</td>
<td>1698</td>
<td>5.8s</td>
<td>145</td>
<td>2.9s</td>
<td>105</td>
<td>7.3s</td>
</tr>
<tr>
<td>s1423</td>
<td>36</td>
<td>81925</td>
<td>4.9s</td>
<td>1329</td>
<td>1.0s</td>
<td>676</td>
<td>2.2s</td>
<td>676</td>
<td>7.8s</td>
</tr>
<tr>
<td>s5378</td>
<td>56</td>
<td>19649</td>
<td>5.2s</td>
<td>392</td>
<td>0.8s</td>
<td>356</td>
<td>2.8s</td>
<td>355</td>
<td>7.3s</td>
</tr>
<tr>
<td>stage</td>
<td>21</td>
<td>191712</td>
<td>57.4s</td>
<td>2816</td>
<td>3.0s</td>
<td>2560</td>
<td>5.3s</td>
<td>2560</td>
<td>156s</td>
</tr>
<tr>
<td>s1196</td>
<td>21</td>
<td>2986</td>
<td>1.4s</td>
<td>485</td>
<td>0.7s</td>
<td>173</td>
<td>1.0s</td>
<td>153</td>
<td>7.9s</td>
</tr>
<tr>
<td>s9234</td>
<td>36</td>
<td>96462</td>
<td>0.3s</td>
<td>26694</td>
<td>0.4s</td>
<td>18450</td>
<td>0.4s</td>
<td>∗18450</td>
<td>1.0s</td>
</tr>
<tr>
<td>bench4</td>
<td>61</td>
<td>1205832</td>
<td>5.7s</td>
<td>39187</td>
<td>24s</td>
<td>24454</td>
<td>84.9s</td>
<td>∗24372</td>
<td>801s</td>
</tr>
<tr>
<td>mish</td>
<td>137</td>
<td>1.12 e+15</td>
<td>0.6s</td>
<td>1.12 e+15</td>
<td>1.7s</td>
<td>82</td>
<td>0.5s</td>
<td>∗82</td>
<td>1.4s</td>
</tr>
<tr>
<td>pdc</td>
<td>56</td>
<td>23232</td>
<td>2.0s</td>
<td>2885</td>
<td>1.3s</td>
<td>272</td>
<td>1.2s</td>
<td>151</td>
<td>46s</td>
</tr>
<tr>
<td>proml</td>
<td>49</td>
<td>9326</td>
<td>5.8s</td>
<td>558</td>
<td>1.8s</td>
<td>609</td>
<td>3.2s</td>
<td>∗472</td>
<td>41.5s</td>
</tr>
<tr>
<td>ibm</td>
<td>65</td>
<td>1047948792</td>
<td>0.7s</td>
<td>819466504</td>
<td>1.8s</td>
<td>173</td>
<td>0.5s</td>
<td>∗173</td>
<td>3.1s</td>
</tr>
<tr>
<td>signet</td>
<td>47</td>
<td>78735</td>
<td>32.2s</td>
<td>13067</td>
<td>72.3s</td>
<td>121</td>
<td>4.1s</td>
<td>∗119</td>
<td>124.1s</td>
</tr>
<tr>
<td>ti</td>
<td>115</td>
<td>836288</td>
<td>1.8s</td>
<td>284986</td>
<td>3.6s</td>
<td>377</td>
<td>1.9s</td>
<td>219</td>
<td>20.0s</td>
</tr>
<tr>
<td>jbp</td>
<td>93</td>
<td>2496810</td>
<td>6.1s</td>
<td>581823</td>
<td>13.0s</td>
<td>129</td>
<td>2.1s</td>
<td>122</td>
<td>38.1s</td>
</tr>
</tbody>
</table>

Table 1. All Primes, Prime Cover, Irredundant Prime Cover, and Minimal Prime Cover Computation.


BIOGRAPHIES

Dr. Olivier Coudert; Digital Paris Research Lab; 85 av. Victor Hugo, 92500 Rueil Malmaison, FRANCE
Olivier Coudert received his engineering degree from Ecole Centrale de Paris in 1987, and his Ph.D. in Computer Science from Ecole Nationale Supérieure des Télécommunications, Paris, France, in 1991. After 4 years at Bull Research Center, he joined Digital in 1993. He has been mainly active in the areas of automated reasoning, logics, and finite state machine verification. He is member of the IEEE.

Dr. Jean Christophe Madre; Digital Paris Research Lab; 85 av. Victor Hugo, 92500 Rueil Malmaison, FRANCE
Jean Christophe Madre received his engineering degree from Ecole Nationale Supérieure d’Informatique et Mathématiques Appliquées de Grenoble in 1984, and his Ph.D. in Computer Science from Ecole Nationale Supérieure des Télécommunications, Paris, France, in 1990. After 7 years at Bull, he joined Digital in 1993. His areas of interest include circuit design, formal verification of hardware, and automated reasoning. He is member of the IEEE.

Henri Fraisse; Bull Corporate Research Center; Rue Jean Jaurès; 78340 Les Clayes-sous-bois FRANCE
Henri Fraisse received his engineering degree from Ecole Supérieure d’Electricité in 1991, and is a Ph.D. student at Ecole Nationale Supérieure des Télécommunications, Paris, France, since 1992. He joined Bull in 1992, where he has been mainly active in the areas of automated reasoning and logics.