Universidad de los Andes
Facultad de Ingeniería
Departamento de Ingeniería de Sistemas y Computación

Andrea Navas Murcia
Laura Nicole Rojas
{a. navas380, ln.rojas1902}@uniandes.edu.co

Variability models in decision-making domains. Study case: Cerebrovascular accidents' etiology

Thesis Adviser: Oscar Gónzalez Rojas

Bogotá, Colombia
2017

Variability models in decision-making domains. Study case: Cerebrovascular accidents' etiology

Andrea Navas, Laura Nicole Rojas

Universidad de los Andes, Bogotá, Colombia

{a.navas380, ln.rojas1902}@uniandes.edu.co

ABSTRACT

Product lines' paradigm has proven to be of great utility in the context of decision making. Through the representation of product variance, and the selection of specific configurations the efficiency of this process has been greatly improved. Using methods like constraint programming as well as variability models, technology has supported a configuration process based on stakeholder's interests. Nevertheless, this approach's utility hasn´t been fully exploited as 1) it has been used, almost exclusively, in technologically-driven domains; and 2) performance issues have become a problem when dealing with large product lines and runtime environments. Therefore, this paper aims to elucidate how to extend the product's line paradigm impact. In the first place by illustrating its usage in a domain such as the medical one, and in the second place exploring an approach that can provide a response to the performance and scalability concerns. Thus, the employment of satisfiability modulo theories-based (SMT) algorithms is evaluated considering the evidence of their high efficiency and expressiveness.

KEYWORDS

Product line, configuration, variability and feature models, FaMa, CoCo, SMT, performance and scalability.

TABLE OF CONTENTS

1. INTRODUCTION 4

2. BASIC CONCEPTS 5

2.1 FEATURE MODELS 5

2.2 CoCo 9

2.3 CP-PROGRAMMING 11

3. SMT-LIB 11

4. SOLUTION PROPOSAL 36

5. VALIDATION 49

6. CONCLUSIONS 55

7. BIBLIOGRAPHY 59

1. INTRODUCTION

The relationship between technology and business has always been a topic of interest, especially with regards to information's management. The huge amount of data, as well as its speed of change has increased the importance of developing an efficient decision-making process. The aid technology can provide in this context can be crucial and thus the need to study approaches that support that process is not only relevant, but also indispensable.

The importance of managing information is indubitable as it allows making pertinent decisions are on time. Therefore, the need to establish the consequences of each choice and to analyze priorities and interests involved in every choice, has acquired an urgent label. Representing adequately the decision-making's domain, visualizing selection possibilities, risks and expected results has become mandatory.

Variability models, which follow product line paradigm, provide a representation of domain's particularities as well as the possibility to incorporate stakeholders', and organizations' interests in the decision-making process. Thus, this type of models and their analysis appear to be an interesting approach in which technology supports business' needs. However, diffusion of this type of solution in non-strictly technological domains is scarce. Nevertheless, as decision-making is essential in every organization the usage of this kind of approach can be useful in the most distinct domains and efforts to model their characteristics ought to be made.

The process of making efficient choices is both essential and difficult. Having tons of available information and being involved in an extremely dynamic environment hinders the process even more. Technological support, therefore, should be not only appropriate but also fast; the need for a high quality performance and scalable solution arises. Taking this into account, it turns out to be mandatory to evaluate alternative ways of analyzing variability model. It is necessary to examine possibilities, and suggest approaches that surpass what constraint programming (CP) offers.

In this study, attending to the previously mentioned concerns, we created a model in the health domain with regards to ischemic etiology[1]. We also examined the SMT algorithms and evaluated

[1] Etiology refers to the study of causes. In this case we refer to the pursuit for an explanation of an ischemic attack as a result of analyzing the characteristics of the stroke.

their characteristics in order to suggest the usage of two of them as an alternative for analyzing the variability models.

2. BASIC CONCEPTS

In the aim of providing support to organization's strategies and objectives, IT governance has focused on making information useful through an adequate and careful management of available data. Especially in decision-making context IT drives its efforts towards restructuring information in a way that will enable analysis of distinct scenarios and aid in the process of making the most appropriate choices. Administrating information for these purposes can be achieved through different approaches being the product line paradigm one of them.

2.1 FEATURE MODELS

According to this paradigm the variability spectrum of a certain domain can be modeled by exhibiting the various characteristics of each product. In each model, the similarities and differences among the features (characteristics) are identified to establish the particularities of every product of a specific product line. The variability of a determined domain is exposed by features models (FM) which allow elements, relationships and options to be displayed.

Feature models have two main elements: 1) a tree diagram; and 2) a set of constraints that enables relationships to be shown. The nodes of the diagram represent product's features which when selected have the Boolean value of *true* and are *false* otherwise. Features can be either atomic (tree leaves) or represent a set of characteristics (nodes with children). The set of constraints can exemplify relationships of three types namely, those between a node and its children, those between different tree branches, and a third one accounting for integrity restrictions.

Restrictions which represent parental relationships are known as tree constraints, which can be of four kinds: alternative exclusive, alternative, optional and mandatory. The first ones can be understood as an exclusive *or* only one children can be selected if the parent is chosen. The alternative relationships are those in which at least one of the children should be picked, while in the optional ones there is no restriction on the number of choices that should be selected. The obligatory type accounts for the cases in which all the options should be chosen.

Integrity restrictions represent relationships characterized by the fact that the selection of one or more features requires either the presence or the absence of a set or a single feature. In the first case, the relationship is expressed through the keyword *requires*, while when picking a set of features forces the exclusion of others we make use of the code word *excludes*. The relationship among different branches of the tree, known as cross-tree constraints, can be understood as an extension of these integrity restrictions. Cross-tree constraints express not only *requires* and *excludes*, but also implications and other type of relationships among branches.

Feature models with these characteristics can be created using different tools and frameworks designed for this purpose; FeatureIDE is one of them. It is an open-code framework, developed as an eclipse plug-in, that aids in the construction of variability models. This framework allows the creation of multiple feature models with mandatory, alternative, optional, and *or* tree constraints. It also lets features to be defined as concrete, abstract, hidden, and indeterminate hidden. FeatureIDE as well, provides elements to create integrity and cross-tree constraints (Batory, 2005). The following diagram illustrates the type of models that can be constructed with FeatureIDE.

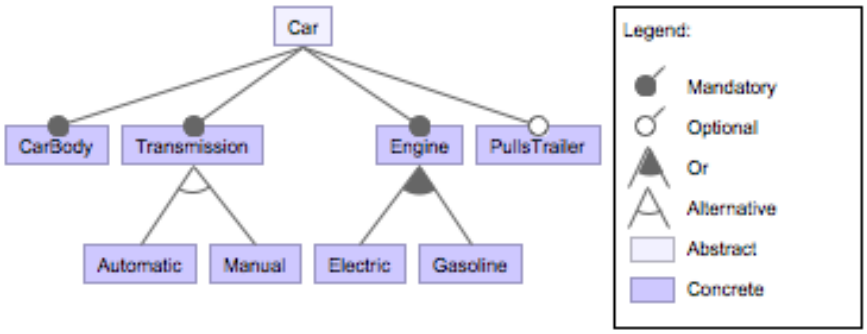

Fig.1 Car Model.

Figure 1 exemplifies a model corresponding to the car's product line. It shows that **CarBody**, **Transmission** and **Engine** are obligatory, while **PullsTrailer** appears to be an optional feature as a car may or may not have one. **Transmission** can be **Automatic** or **Manual**, but it is also allowed for the car to have both types which is modeled through an alternative constraint. **Engine** is

modeled to be either **Electric** or to use **Gasoline**, but can't have both of them; this is illustrated through the use of an or-constraint[2].

Figure 2 shows that Cross-tree constraints can also be created with FeatureIDE's plug-in. In this example, what is modeled through the restrictions is that if the car chosen has **Automatic** transmission then it should be **Electric**, otherwise it must use **Gasoline**. (Here the model is slightly modified to allow the constraints to be valid.)

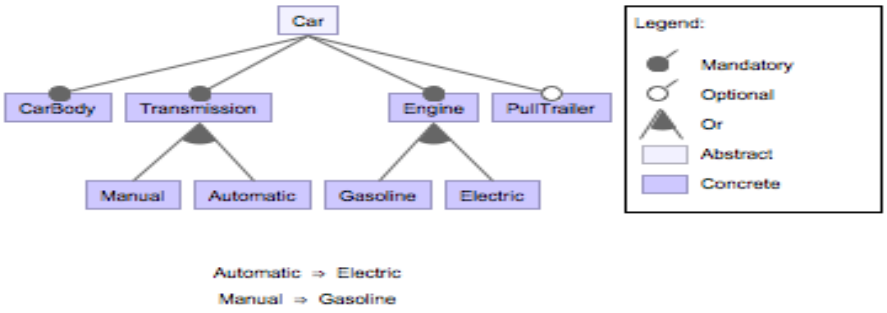

Fig.2 Car Model with Cross-tree constraints.

Feature models created using frameworks like FeatureIDE, allow domains' variability to be displayed. Nevertheless, they have a limited scope since stakeholder's interests can't be considered in the decision-making process we are looking forward to provide support to. Hence, an extension of these models, which includes non-functional properties, has been developed. This robust version allows every feature to be identified by its selection status, but also by its characteristics. Attributes can be thought of an extension of a feature as they include information that doesn't refer to functionalities; it can't be not expressed using feature models. The concept of feature's attributes can be illustrated in a more comprehensible way through the car's product line model. In this particular case, we can extend the feature model referring, for example, to the feature called **Carbody** to which we could associate an attribute that accounts for its color. *Color* can acquire different values depending on the stakeholder's choices; it can be defined as black, blue, silver,

[2] The model's main purpose is not to be feasible, but to illustrate FeaturesIDE's models and the different types of constraints it manages.

etc. The model is now an extended version that includes feature's properties and is commonly known as attributed feature model (AFM) (Ochoa, Gonzalez, and Thüm, 2015).

The inclusion of attributes in AFM doesn't accounts for what makes attributed models particularly interesting in the decision-making context. Actually, what explains AFM's value is the possibility to create new types of constraints and to perform more appropriate analysis. Having attributes allows solution constraints to be constructed due to the fact that this are related with the optimization of attribute's values. These particular constraints enable organization's strategies and objectives to be contemplated in the decision making. Through solution constraints stakeholder's interests and expectations are not only reflected, but also taken into account in the analysis (Ochoa, Gonzalez, Verano, and Castro, 2016).

With this in mind, AFM's development seems to provide a more appropriate aid for organizations than the one FM delivers. Therefore, it is convenient to explore tools which can support the building of both types of models. AFM's creation is not necessarily supported by frameworks used for the construction of FM; in fact, FeatureIDE doesn't allow the creation of attributes, hence we should rely on other type of tools. FaMa, a framework essentially employed for the analysis and creation of variability models, comes across as an appropriate option. This tool is constructed following the product line paradigm and is written in Java. Its main objective is to allow automatized analysis of FM and AFM and to help in the construction of favorable products for each organization (Benavides, Segura, Trinidad, and Ruiz-Cortes, 2007). FaMa offers the possibility to build models that can be evaluated in terms of validity and error detection, it also allows the creation of different types of constraints, products, and configurations.

In both AFM and FM's configurations can be determined through the selection of certain features and deselection of others. Depending upon the consistence with the restrictions established models being built are considered to be either valid or invalid. Furthermore, configurations can be classified as partial or complete according to the choices that have been made. When every feature is either selected or deselected having a value assigned (either true or false respectively) the configuration is considered to be complete, and it is partial otherwise. If it is both complete and valid then it represents a product.

Every product is the result of a series of decisions made in conformity with established rules or constraints. Tree, cross-tree, and integrity constraints are evaluated to determine if a configuration is valid and thus (if complete) can be considered as a product. In attributed models solution constraints are as well evaluated and some tools, such as FaMa, even permit cross-model restrictions to be considered. This last type of constraint makes reference to the existence of various models which can be associated among them. Those relationships are the ones cross-model constraints illustrate making it possible to consider them in the decision-making process (Ochoa, et al., 2016).

Models along with configurations, restrictions, and products are the main input for tools meant to evaluate the relevance and appropriateness of each decision. The usage of these tools enhances the visualization of diverse decision scenarios, the comparison of expected results, and the evaluation of distinct options. At this stage, solution constraints are considered; minimizing costs, reducing time, optimizing processes, diminishing risks and other particular interests are contemplated within the analysis performed.

Aided by solvers and various tools, products that attend to the specific business' needs can be created; determining which are the best choices in a certain context. However, establishing the most suitable product and the decisions to be made, might turn out to be a thorny task if conflict among configurations arises. Although single configurations are validated, a set of configurations as a whole may be conflicting. In other words, the choice of a particular stakeholder in a configuration could be exclusive with respect to the selection made by another stakeholder which will mean that the set of configurations is invalid (Ochoa, et al., 2015).

The previously mentioned situation appears to be very common because usually there are multiple stakeholders engaged on the same decision-making process. Different roles played by distinct stakeholders, as well as diverse knowledge and levels of expertise account for the existence of various configurations which might turn out to be incompatible. As a consequence, the need to solve these possible conflicts arises.

2.2 CoCo

One possible approach, known as model-based transformation, can be illustrated through the usage

of a domain-specific language such *Co*nflicts among *Co*nfigurations resolution (abbreviated as CoCo). This language, as its names suggests, provides assistance for resolving configuration's conflicts. CoCo works with FM and AFM and admits the specification of attributes as non-functional properties of features and the creation of business-related decision rules. The specified properties, together with the defined configurations and the corresponding feature model (in its extended version) are transformed into a constraint satisfying problem (CSP). The answer to these type of problems turns out to be a non-conflicting set of configuration which is found through the execution of the decision rules. In this way CoCo guarantees a solution of conflicting configuration's issues which is appropriate and aligned with business strategies and priorities.

In CoCo attributes are defined through five properties: an attribute type reference, an impact variable, a minimum and maximum value and some observations. The reference corresponds to an attribute type[3] specifically defined for the organization, the impact variable identifies where the particular non-functional property affects the business, the maximum and minimum values, through which a range for the attribute is established, and a set of observations which contains qualitative information that may be valuable when analyzing different options. These attributes provide useful aid for the subsequent creation of the CSP.

Solution constraints are essential for resolving conflicting configurations, but they aren't actually created for that purpose. With this in mind, a language like CoCo implements decision rules with a special specification. This type of rules is defined by each organization as decisive and impartial criteria to solve stakeholders' conflicts. Decision rules complement other constraints allowing stakeholders' interests to be better defined and thus have a greater influence in the decision-making. Furthermore, they help to reduce the set of alternatives to a subset that is better aligned with business interests. CoCo has two types of decision rules: optimization and hard-limit rules.

Through the conjunction of non-functional properties and decision rules CoCo offers a very useful support to organizations in the decision-making as it considers business objectives and interests in the process. This type of approach is based on a paradigm called constraint programming,

[3] Attribute types in CoCo help to define how evaluation is performed in a specific organization. They can be thought as a set of information containing: an identification that establishes the dimension it represents (such as cost, time, risks, etc.), a unit measure (for example dollars if we refer to costs dimension), and a frequency unit defining the periodicity of the dimension impact (monthly, daily, among others).

accounting for the translation of the model into a constraint satisfying problem to provide analysis and resolutions for conflicting configurations.

2.3 CP-PROGRAMMING

Constraint programming paradigm deals with CSPs which can represent the problem of selecting the adequate set of features that satisfy certain constraint. "CSPs can be defined as tuples ⟨X,P,C⟩, where $X = \{X_1,...,X_n\}$ represents a set of n features, $P_{n\times m} = \{P_{[1,1]},...,P_{[n,m]}\}$ represents a matrix with a set of $(n * m)$ attribute values related to each feature in X, and $C = \{C_1, . . . , C_k\}$ represents a set of k constraints" (Ochoa, Alves, Gonzalez, Castro and Saake, 2017, p.7).

CSP is only one of the possible approaches to variability models' creation and analysis; different reliable constraint solvers have been used to check configurations' validity. One of the most common approximations is the usage of Boolean satisfiability problem (SAT) based solvers. This type of solvers is specifically designed to determine if there exists an interpretation that satisfies a given Boolean formula. Through this approach variability and feature models related problems can be solved, but its expressiveness is scarce as configuration problems should be casted into Boolean formulas in order to be understood by SAT-based solvers. This makes the SAT's approach less preferable than CSP as the latter allows the usage of several variable types (Michels, Ganesh, Hubaux, and Heymans, 2017).

3. SMT-LIB

CP-based approaches in product line's field and variability models' context have become very popular mainly because the translation of FM and specially AFM turns out to be feasible and simple. In fact, this type of transformation can be performed in just three basic steps[4]. Furthermore, as some studies reveal this approach proves to be useful as results obtained through it have turned out to be satisfactory. According to Ochoa and Gonzalez (2017) CSP conforms to the established constraints, and obtains valid product. It surpasses other approximations such as evolution algorithms (EA) where some constraints are violated which makes necessary the implementation of controls to minimize violations. CSP-based solutions are also considered as a

[4] The description of this process isn't included in this paper as it exceeds its scope. For further information about the topic it is recommended to refer to Ochoa, et al., 2015.

feasible since they support arithmetic, Boolean and optimization functions. The aforementioned is quite desirable because it provides expressiveness which is fundamental for the solution constraints' building (Ochoa, and Gonzalez, 2017).

Feature, tree constraints as well as feature attributes can be mapped into a CSP. Features are translated as Boolean, attributes as integers, and tree constraints can be built using equality operations and operators such as *and, or, not,* and implication. Cross-tree and solution constraints can also be mapped. Additionally, CP-based solutions enable resource constraints' specifications and provides support for single objective optimization requirements. (Ochoa, et al., 2017) These characteristics explain why CP-approach appears as a suitable one.

However, when evaluated in terms of performance and scalability the adequacy of this strategy is questioned. Although this approach supports run-time configuration as well as interaction with multiple product lines, when AFM are large its performance is ameliorated; dealing with large scale models, and multiple constraints deteriorates performance. As established by Ochoa, et al., (2017) studies made on CP-based algorithms have shown that they take exponential times in solving big problems which evidences their performance's failures. This suggests that an examination of other type of algorithms, solvers and approaches should be considered in order to analyze options for improving the actual implementation.

EA-based approaches seem to be an alternative as they show good performance and scalability, providing solutions for large-scale models within few seconds. Nevertheless, as aforementioned they can't guarantee constraint's satisfaction, or take into account stakeholders' preferences. Being this of great importance when providing support for the decision-making process makes this alternative quite undesirable. As a result, it turns out to be is mandatory to search for a combined option in which we take advantage of what each alternative offers in order to respond to product's requests, stakeholder's interests, and to scalability and performance requirements.

Given these circumstances we decided to evaluate an approach based on SMT algorithms which aren't part of CoCo's solvers considering the possibility of including them to improve its performance. SMT solvers are considered as an option, mainly because they offer high efficiency and expressiveness which has been evidenced through their multiple implementations for the most diverse purposes. This type of solvers is often based on the SAT paradigm which has already been

used in the context of variability models. SAT-based solution although efficient, are not as expressive as CP-based alternatives which are instead less efficient than the formers. SMT offers a combination of the two approaches as an effort to get the best of both alternatives: the efficiency of the SAT solvers and the expressiveness which characterizes CP-based solutions (Michels, et al., 2017). SMT solvers turn out to be both efficient and expressive for many applications, including configuration problems, what makes SMT a viable alternative to be considered when trying to improve SAT and CSP solvers' performance.

SMT is an area that studies how to check the satisfiability of first-order formulas with respect to some logical theory T. This theory uses specialized inference methods, which when employed contribute to their efficiency (Barret, Stump, and Tinelli, 2010).

SMT's usage in variability model's context not only proves to be desirable due to its performance but also because of the documentation available to aid in SMT's implementation. The latter is a consequence of the existence of an initiative called SMT-LIB whose main purpose is precisely to facilitate research and development in SMT. This enterprise has accomplished to provide appropriate descriptions of background theories, develop input and output languages for solvers, and establish its specific benchmarks[5], among other achievements. Additionally, SMT-LIB has made it possible to evaluate SMT systems mainly through the execution of an annual competition (SMT-COMP) based on the library's benchmarks (Barret, et al., 2010).

Evaluating the SMT-LIB algorithms is very useful as it helps us to determine which of them is the most suitable for a specific use. The SMT-COMP offers a first approach to compare the algorithms, however other aspects should be taken into account in order to make a more complete examination. This being so, it is pertinent to examine some characteristics of each of the SMT solvers[6] which can be identified through four main features: 1) an underlying logic; 2) a background theory; 3) input formulas; and 4) an interface (Barret, et al., 2010). The underlying logic may be modal, temporal, of first-order, of second-order, among others, the background theory refers to the one against which every model and formula is checked to establish its

[5] A benchmark in this context should be understood as a logical formula that is meant to be checked in terms of satisfiability with respect to some particular theory.
[6] These solvers can be defined as any type of software system implementing a procedure for satisfiability modulo over some given theory.

satisfiability, the input formulas are the type of input that the solver is able to interpret, and the interface is the set of functionalities that the solver provides. Taking this into consideration we performed a careful evaluation of the SMT solvers considering the aforementioned features, SMT-COMP's results, and some aspects of its implementation. Table 1 illustrates this comparison.

Name	Features	Implementation	Performance
ABsolver	**Underlying logic:** It is an extensible implementation of SMT which allows an integration of various subordinate solvers. Its solver algorithm is based on a Boolean abstraction and a constraint solver which searches for a solution. ABsolver is DPLL-based. **Background theories:** Boolean, linear and non-linear arithmetics. **Input formulas:** standard DIMACS format SAT problems. The background constraints are expressed in a custom language, encoded in the DIMACS comments. **Interface:** it checks formulas´ satisfability to the corresponding background theories.	**Api:** C++ **OS Platform:** Linux When ABSolver is integrated with other appropriate solvers it can either become a stand-alone tool or be part of a system library. In the last case it aids in the extension of constraint handling systems. It follows an abstraction/refinement approach in which an abstract SAT problem is produced and solved first. In ABSolver each constraint is represented with a Boolean variable, which is replaced by its respective constraint depending on its value (if it is true the corresponding constraint should be satisfied, else the negation of the constraint is added to the constraint´s set). The resulting constraint set is passed to a specific solver for the background theory of the problem which tries to establish its satisfiability. If a solution is found the problem has been resolved, else the Boolean abstraction should be refined and the process should start again. To improve its performance an optimization to the abstraction/refinement approach has been implemented: once a SAT solver has determined a solution it is first generalized before solving the corresponding constraint problem. The idea is to find a minimal assignment which yields smaller constraint problems. Each variable can have one of the tree values: true, false, or don't care. If the variable´s value is the last one, the constraint represented by it is ignored which results in a smaller constraint set.	Solvers which employ an iterative approach perform better in test runs (taking less time to resolve the problems), still ABSolver has a comparatively stable and reliable performance. In fact with the generalization optimization it can solve most test runs in additional time which is only greater by a constant factor. ABSOLVERDC is comparable to CVC 3, but it is clearly slower than MathSAT and Yices. Nevertheless, in the last case 60% of all the compared benchmarks are solved by it within a runtime which is only larger by a constant factor. Although ABSolver can deal with non-linear problems the solutions it presents may not be reliable due to the local nature of the solver and to rounding errors IPOPT (a numerical optimization tool it uses to deal with non-linear constraints).

Name	Features	Implementation	Performance
Alt-Ergo	**Underlying logic:** It is an automatic solver for mathematical formulas which uses a polymorphic first-order logic. It is a SAT-solver based algorithm that combines Shostak-like and Nelson-Oppen like approaches for reasoning modulo theories. **Background theories:** empty theory, linear integer, rational and non-linear arithmetic, polymorphic arrays, enumerated and record datatypes, associative and commutative (AC) symbols, bit vectors, and quantifiers. **Input formulas:** specific input language with prenex polymorphism. It also supports SMT-Lib 2 language having a less efficient performance on SMT-Lib files. **Interface:** proving formulas especially in context of deductive program verification, it is mainly used by Why3 (a platform for this type of verifications) as its main prover. Alt-Ergo as well, proves correctness of programs in C and Java. It is used as back-end of different tools such as Frame-C suite (to prove formulas generated from C code), Spark toolset (to check formulas produced from Ada programs), CAVEAT, Cubicle, among others. Alt-Ergo is also used for verifications in B modelizations and for cryptographic protocols verification.	**Api:** OCaml. **OS Platform:** Linux, Mac OS, Windows. Alt-Ergo is implemented with a highly modular architecture and it consists only of 7500 lines of OCaml code. Its core is made of three basic elements: a Depth First Search (DFS) based SAT solver, a quantifier instantiation engine based in E-matching, and a set of decision procedures associated with its built-in theories. It has as well a Semantic Combination of Congruence Closure with Solvable Theories (CC(X)) core. Alt-Ergo is distributed through an open source license called Cecil-C.	Alt-Ergo´s performance (it 0.95 version) has been compared with some SMT-solvers which support quantifiers. Specifically, with CVC3(v 2.4.1), YICES (v 1.0.38) and Z3 (v 3.2 and v 4.2). For the experiments logical formulas generated from WHY3's gallery of programs were used. The results show that it was the solver with the best reliability making 0 mistakes in its verifications. Additionally, it was the one which proved more formulas' validity although it took more time than the rest (but time´s difference was not significant). It shows a better performance than the other solvers and has made significant advances in comparison to its previews versions (in terms of efficiency).

Name	Features	Implementation	Performance
Barcelogic	**Underlying logic:** Barcelogic is the SMT-solver developed by a group of the Technical University of Catalonia. It works as an efficient logic-based tool and is featured to solve optimization problems. The solver is DPLL-based, and works with an extension of the congruence closure algorithm through the solver for EUF, as well as with an extension of the shortest path algorithm implemented in the DL solver.	**Api:** C++ **OS Platform:** Linux Barcelogic implements a DPLL(T) framework which consists of a general DPLL(X) engine, whose parameter X can be instantiated with a solver for a theory T. The engine enumerates propositional models of the formula, while the solver verifies that each model is consistent with the theory T. For better performance it works with several adaptive heuristics to find the right solver used for consistency and theory propagation. Its DPLL(X) engine implements two-watched literal scheme for unit propagation, the first-UIP learning scheme, and VSIDS-like decision heuristics. Its system has basically three components a parser and a preprocessor, the DPLL(X) Boolean engine, and the theory solvers (EUF, DL, and LA solvers).	It is competitive with other SAT solvers and is actually very similar to them. It implements basic ideas used in zChaff and MiniSAT. In the 2005 SMT Competition, Barcelogic won all four divisions for which it had a theory solver: EUF, IDL and RDL (integer and real Difference Logic respectively), and UFIDL (combining EUF and IDL). It is considered to be very efficient and scalable. In 2009 it competed as well in this competition winning que quantifier-free nonlinear integer arithmetics division (QF_NIA) being the solver that has solved the greatest amount of that type of problems by 2013.
	Background theories: equalities uninterpreted functions (EUF or empty theory), difference logic (DL), integer, non-linear and linear arithmetics (LA), Booleans and combinations of these theories.		
	Input formulas: SMT-LIB format.		
	Interface: Barcelogic checks satisfiability, allows model generation when the formulas are valid, performs predicate abstraction (extracting finite-state abstractions from systems whose space may be infinite). It also provides functions of Max-SMT and Max-SAT such as finding assignments of variables that maximize the number of satisfied constraints.		

Name	Features	Implementation	Performance
Beaver	**Underlying logic:** It is a solver for bit-vector logic (QF_BV). which makes it useful for low-level description systems in languages such as C and Verilog. Beaver is SAT-based **Background theories:** quantifier-free finite precision bit-vector logic (QF_BV). **Input formulas:** bit-vector formulas. **Interface:** Beaver features satisfiability as well as model generation for satisfiable formulas. I t also is used for software verification and testing, hardware verification, and non-linear operations.	**Api:** OCaml **OS Platform:** Linux and Windows Beaver uses an eager approach to encode SMT problems into SAT problems, through word-level and bit-level transformations. It transforms the bit-vector arithmetic formula into a Boolean circuit and then into a SAT problem (this is the eager approach). It massages the bit-vector to Boolean formulas directly or to intermediate formula in which simplifications are applied in order to get the final Boolean formula for the SAT solver. To improve its performance, it develops an application-driven engineering of a set of simplification methods such as backward and forward constraint and equality propagation through event queues. It also implements bit-vectors rewrite rules and offline optimization of Boolean circuit templates for operators. After these simplifications it gets an intermediate Boolean formula which is represented through a and-inverter graph in which further simplifications are performed.	The type of SAT solver used impacts greatly in its performance; the use of non-causal solvers might be beneficial for its efficiency (this was evidenced through experiments in which the best performance was achieved using NFLSAT.) Nevertheless, this is not always the case, actually Boolean structure and the form of atomic constraints is what determines if non-clausure solvers are beneficial. Constraint propagation also improves its performance, speeding up the solver.

Name	Features	Implementation	Performance
Boolector	**Underlying logic:** It is a solver for quantifier-free finite precision bit-vector logic and arrays (QF_AUFBV). As it implements the extensional theory of arrays it becomes useful especially for HW and SW verification tasks as arrays allow the modelling of software memory and hardware caches. Boolector is SAT-based. **Background theories:** quantifier-free bit-vectors, arrays, lambda expressions, and EUF. **Input formulas:** SMT-LIB format and BTOR (low-level bit-vector format with clean semantics). **Interface:** Boolector not only establishes satisfiability, but also aids in creating concrete models for bit-vectors and arrays. It is used for incremental model checking of safety properties (allowed by BTOR), under-approximation, and converting formulas from SMT-LIB to BTOR and vice versa.	**Api:** C and Python. **OS Platform:** Linux It has a parser that receives the input formulas and builds an abstract syntax Directed acyclic graphs (DAG). The graph is simplified through rewriting rules provided and applied by a rewriter engine. Boolector has an array consistency checker which evaluates the assignments done by the SAT solver to establish its consistency to the theory. For bit-vectors it performs an under-approximation approach, using r PicoSat as the corresponding SAT solver for this task. For the formula refinement process, it has two approaches: bit-vectors are translated into SAT, while the arrays are abstracted using bit-vector variables. Here the SAT solver is called by the refinement loop for it to search for an appropriate assignment. This one is evaluated by the array consistency (if the solver actually classifies the formula as satisfiable). If there appears to be no consistency, then a lemma on demand that explains the actual incoherence is added as formula refinement. It performs term rewriting, and bit-blasting for bit-vectors and lemma on demand for arrays.	In 2017 Boolector´s latest version participated in 5 divisions the SMT competition (BV, QF_ABV, QF_AUFBV, QF_BV, and QF_UFBV) winning in three of them. In Quantifier-free bit vector (QF_BV) it won in sequential performance but was surpassed in parallel by MinkeyRink. It won in QF_ABV, and also in QF_UFBV.

Name	Features	Implementation	Performance
CVC3	**Underlying logic:** CVC3 is an extension of Cooperative validity checker which works as an automatic theorem prover. It supports the SMT-LIB initiave. CVC3 ´S logic comes from first order theories, it is SAT and DPLL-based. **Background theories**: rational, integer, linear and non-linear arithmetics, arrays, tuples, records, inductive data types, bit vectors, and equality over uninterpreted function symbols (EUF). CVC3 also is able to manage abstract data types mainly arbitrary recursive and mutually recursive. It provides as well support for quantifiers. **Input formulas:** first order logic, formulas with and quantifiers and SMT-lib queries. It accepts one or more assertion formulas with a query formula. It can translate as well benchmarks to and from SMT-LIB. **Interface:** Its main function is to check if a certain formula is valid in a concrete theory under a set of assumptions. CVC3 also allows formula´s creation and predicate subtyping, proof and model generation. It can also be used to produce proofs and to verify the categorization of benchmarks in SMT as well as for checking the syntax of new benchmarks for the library.	**Api:** C and C++ **OS Platform:** Linux CVC combines decision procedures for certain logical theories into one decision procedure for the theories union. In CVC3 implementation a new search engine is incorporate making it easier to plug in different DPLL engine implementations. The latest was achieved through the development of an abstract API. CVC3 has two SAT solvers integrated: zChaff and mini SAT. It has as well several user interfaces: high levels apis, an interactive command driven, and a file interface. The main Api is responsible for the formula creation and the validation/satisfiability checking. CVC3 implements a deduction engine which l inks the capabilities of the DPLL engine for Boolean reasoning with those of the theory solver. The DPLL is based on a Boolean SAT solver, while the Theory Solver relies on the decision procedures of the built-in theories. It has heuristics for reasoning about quantifiers and a stack-based push and pop approach for incremental use.	Its performance with bit vectors has improved with respect to CVC Lite, still it is naive based on bit-blasting and preprocessing. Core aspects of the design make it less efficient than other tools as it leads to high memory use, and the use of heavy weight computation. CVC3 was released as a major overhaul of portions of CVC Lite with an addition of better decision procedure implementations. It outperforms CVCLite in all benchmarks by a factor of 2 or 3, but is outperformed by CVC4. In 2015 it participated for the last time in the SMT-competition in 19 divisions. It only won in the UFLRA division. Its performance has been surpassed by the latest implementation of CVC, i.e. CVC4, and Z3 in most of the divisions in which it participated. Actually it was surpassed by CVC4 in NIA (non-linear integer arithmetics), and by both Z3 and CVC4 in QF_UFNIA (quantifier-free free sort and function symbols) AUFLIRA, AUFNIRA, AUFLIA, LIA, and LIRA (linear fragments of real and integer arithmetics). Bit-vector wise it was again outperformed by CVC4 and Z3 in UF, BV, and UFBV divisions. It has the best performance in UFLRA.

Name	Features	Implementation	Performance
CVC4	**Underlying logic:** CVC4 is a very efficient automatic theorem prover for satisfiability modulo theories (SMT) problems. It can be used to prove the validity and satisfiability of first-order formulas in many built-in theories and their combination. CVC4 is the fifth generation in the Cooperating Validity Checker family of tools (CVC, CVC Lite, CVC3) but doesn´t incorporate code from any previous version. It supports the features of CVC3 and SMT-Lib 2 and optimize the core system architecture. It tries to incorporate the algorithmic and engineering advances. The solver is DPLL-based **Background theories:** rational, integer linear and non-linear arithmetics, arrays, tuples, records, inductive data types, bit-vectors, strings, and equality over uninterpreted functions separation logics, and finite sets. **Input formulas:** first-order logic with polymorphic types. It supports quantifiers with heuristic instantiation, SMT-Lib, Z3 extended commands, native CVC format as well as the Thousands of Problems for Theorem Provers library formulas (TPTP). **Interface:** CVC4 aids in satisfiability checking and has model generation abilities. It can be used for research or commercial purposes. It also allows verification, test generation, proofs construction and establishment of unsatisfiable cores.	**Api:** C++ **OS Platform:** Linux, Mac OS, Windows CVC4 represents a complete re-evaluation of the core architecture of the latest version i.e. CVC3 to provide a better performance and to be used as a research vehicle for the future. Its implementation was done through a clean room start approach, rather than taking CVC3 and redesigning it. Although it seems to have almost no resemblance with CVC3 it is very similar: it is implemented as a DPLL (T) solver and has a delegation path to different decision procedure implementations for its built-in theories. CVC4 is built around a central core of three basic engines. The first one is SMT engine which is the main interface point to the solver. It is in charge of validity checking of formulas and functions to require proofs and models and manipulation of assumptions. The second engine is related with the propositional solver, allowing different SAT solvers to be plugged in to CVC4. Finally, the theory Engine which manages all the decision procedure implementations. CVC4 is conceived as a better version of CVC3 with a cleaner and most consistent library API. It has a more modular and flexible core, a smaller memory footprint and better performance. It uses a more light-weighted approach for theorem computation, and incorporates numerous managers in charge of managing subsystems. CVC4 incorporates newly-designed and implemented decision procedures for EUF, arrays, bit vector, and inductive datatypes. It also implements a version of the Simplex method for arithmetics.	It has increased performance in terms of expression subsystem, and its solving apparatus also works better. It solves as twice solvers as CVC3 is capable of and actually does it in a faster way. The latest w was evidenced in SMT-comp of 2010. CVC4 also has better control of preprocessing. In 2017s SMT competition CVC4 participated in 43 out of the 45 divisions, showing in each of them an outstanding performance. It won in nine of them namely: AUFDTLIA (array declaration theory linear integer arithmetic), AUFLIA, QF_ANIA, QF_NIA, QF_LRA, UF_DT, UFLIA, UFIDL (integer difference logic) , AUFBVDTLIA, and QFDT. In the last two divisions it was the only solver which participated. In the rest of the divisions it was surpassed mainly by Z3 4.5 and vampire 4.2. Some solvers such as Yices2, and VeriT also showed a better performance than CVC4 in other divisions.

Name	Features	Implementation	Performance
DPT	**Underlying logic:** The decision procedure toolkit is a system of cooperating decision procedures for answering satisfiability queries. It is DPLL-based. **Background theories:** EUF. **Input formulas:** SMT-LIB and DIMACS CNF files. **Interface:** includes a propositional satisfiability solver and a satisfiability solver for the theory of uninterpreted functions. DPT works as a platform for experiments in SMT solvers and their applications.	**Api:** OCAMl **OS Platform:** Linux Modern implementation of DPLL(T) with various theory solvers. It combines theory decisions procedures with the DPLL style SAT solver. It implements MiniSAT solver.	No info available.
iSAT	**Underlying logic:** iSAT is a satisfiability checker of Boolean combinations of both linear and non-linear arithmetics constraints over real and integer valued variables (non-linear involve transcendental functions). It works as a combination of SAT solvers techniques and interval-based arithmetic constraints solving. iSAT is DPLL-based. **Background theories:** linear, non-linear and floating point arithmetics (support for the last one is implemented in the iSAT3 version). **Input formulas:** Boolean combinations of linear and non-linear constraints. **Interface:** It can work with transcendental functions and allows the verification of system´s safety properties. iSAT has two modes of operation: (i) satisfiability checker for a single formula (ii) trace finder for a hybrid system through bounded model checking (BMC). In the second mode it aims to find a run of bounded length which starts in an initial state of the system, obeys its transitions, and ends in a state in which certain properties hold. The idea is to build a formula whose satisfiability depends on the existence of a trace for which the above mentioned characteristics hold.	**Api:** Not reported. **OS Platform:** Linux iSAT preforms a backtrack search to prune the search space until getting one which is small enough and has no contradictions according to the formula´s constraints. Its search is done in two steps: a decision (blindly picking through a splitting interval approach) followed by a deduction step which explores the previously generated space. Deduction may find a conflict and indicate the need of a backtrack. It is based on solving techniques from DPLL-style such as lazy evaluation, conflict-driven learning, non-chronological backtracking and restarts. iSAT3 supports bitwise integer operation, uses literals and includes advanced formula preprocessing.	It can´t decide all given formulae due to its method: interval arithmetics and splitting intervals yield a highly incomplete deduction calculus. The performance of its BMC mode of proceeding depends on the solver in which it is based, therefore iSAT team tries to speed up these pieces of SW to gain efficiency. Improvements to enhance performance have been implemented through formula preprocessing and solver core. In the first approach the abstract syntax tree (AST) to build the representation of the input formulas was modified by a graph (ASG) representation. Solver-core wise iSAT developed an implementation that allows it to operate directly on literals and not through intervals as before. The change from AST to ASG improved its performance as it allowed a significant drop in solving times. Changes in the solver core, as well as the adaptation of decision heuristics, and the addition of other propositional SAT solving techniques (recursive conflict minimization for example) gave it a performance boost. These improvements have accelerated the solver between one and two orders of magnitude.

Name	Features	Implementation	Performance
MathSAT	**Underlying logic:** It is a long term and cooperative project which provides an efficient satisfiability solver whose last version is MathSAT 5. It is a DPLL-based solver. **Background theories:** empty theory, linear arithmetic, bit vectors, arrays, floating point (few support this one), EUF. **Input formulas:** SMT-LIB 2, SMT-LIBv1.2, DIMACS **Interface:** MathSAT allows satisfiability checking and provides some SMT-extended tasks. It provides unsat core verification, interpolation, incremental support, as well as a framework for third-party SAT-solvers integration. It produces models and proofs and implements an ALLSMT functionality which can enumerate complete sets of theory consistent partial assignments for a satisfiable formula.	**Api:** C/C++, Python, Java **OS Platform:** Linux, Mac OS, Windows Its architecture is composed by the following elements: a preprocessor, a constraint encoder, a theory manager, a proof and model generator, a SAT engine and a set of individual theory solvers. It has an object-oriented paradigm, and uses ad-hoc variants of data structures which aid in performance's improvement. This results in a low-level optimization. MathSAT implements a push and pop incremental stack-based interface which allows multiple satisfiability checks over a changing clause database. The theory manager is responsible for the communication between the SAT engine and the theory solvers. This particular design allows new theories to be added and removed in an easy way. The SAT engine and the theory solvers work in a DPLL-based lazy approach. The SAT engine is either the default one (MiniSAT-style solver) or a third party engine that has been plugged.	MathSAT doesn`t provide support for quantifiers yet, still many performance improvements included have been implemented in the latest version of MathSAT, being its fifth release much more efficient than MathSAT4. From the solving perspective this new version´s support has been extended to fully work with arrays, deal with mixed rational integers, and include as well floating point theories. Theory aware SAT preprocessing has been implemented (SAT-style Boolean preprocessing), aiding in the support of incrementality.

Name	Features	Implementation	Performance
MiniSMT	**Underlying logic:** Simple SMT solver for non-linear arithmetics based on Yices and MiniSAT. Its main purpose is to reason about possibly irrational quantifier-free non-linear arithmetic through a reduction to SAT/SMT. It is Yices and SAT-based. **Background theories:** irrational and non-linear arithmetics. **Input formulas:** SMT-LIB. **Interface:** MiniSMT establishes satisfiability and allows model generation for satisfiable instances of non-linear arithmetic quickly. Its main feature is the support for irrational domains. It is also used as a termination tool (especially, but not exclusively, for term rewrite systems).	**Api:** Not reported. **OS Platform:** Linux Integral domains are managed through bit-blasting to SAT, and by transformation to bit-vector arithmetics enabling solvers for these logics to perform their job. Non-integral domains are as well supported by a reduction to the integral settings.	It is only dedicated to satisfiable instances and performs poorly for large dimensions. MiniSMT participated in the 2010 SMT competition winning in the QF_NIA and QF_NRA divisions. When compared with CVC3 and Barcelogic over SAT and BV it shows efficiency being the one which resolved the greatest amount of problems (267 in SAT and 268 in BV out of 470). Its execution time shows to be better than the one of CVC3, but it takes longer than Barcelogic. Experiments performed over matrix constraints of one and two dimensions in natural, rational and integer domains show that MiniSMT has the best performance in terms of execution times and problems solved. In these experiments the solvers compared where: nsol, CVC3 and Barcelogic. MiniSMT is very powerful on SMT-LIB benchmarks, it has the best performance on matrix benchmarks, and is the most powerful tool on rational domains. It is the fastest on small and rational domain and is the only with efficient support for irrational domains: it outperforms current SMT on those domains.

Name	Features	Implementation	Performance
OpenCog	**Underlying logic:** A satisfiability modulo theories solver built in as a part of a generic graph database that holds terms, atomic formulas, sentences and relationships as hypergraphs.	**API:** C++ (uses Boost and templates), plus Python, Haskell and Scheme in some parts. **OS Platform:** Linux OpenCog consists of a core framework with several modules. This framework is composed by MindAgent that is a software module to performs computations. MindAgents can be thought of as processes or threads; in most cases, they should be implemented as threads.	TA performance suite would collect into one place a large variety of different test cases, instrument them properly so as to measure speed and memory usage, and then report the results, in a completely automated fashion. An existing but simplistic benchmark utility for the AtomSpace exists in https://github.com/opencog/atomspace/tree/master/opencog/benchmark
	Background theories: This solver is a pattern matching by a probabilistic truth-value interpretation. OpenCog is based on probabilistic logic networks (PLN).		
	Input formulas: They are described in a probabilistic generic program MOSES (Meta-Optimizing Semantic Evolutionary Search which performs supervised learning). This algorithm requires a scoring function or training data to be specified as input. As output, it generates a program that, when executed, approximates the scoring function. MOSES also use well-known optimization algorithms such a hill-climbing, simulated annealing or estimation of distribution algorithms (EDA) such as Bayesian optimization (BOA/hBOA).		
	Interface: MOSES currently supports representations for several domains like proposition formulas, actions (mini programming languages describing actions, typically used in artificial intelligence), arithmetic formulas and predicate logic. This algorithm may support SAT or satisfiability-modulo-theory (SMT) but remains unexplored.		

Name	Features	Implementation	Performance
OpenSMT/OpenSMT2	**Underlying logic:** OpenSMT was designed with the idea of easy extension and customization of a new solver principally based on MiniSAT. Building a new solver requires to implement a simple interface derived from TSolver class. **Background theories:** OpenSMT implements the theories: quantifier-free uninterpreted functions with equalities (QF_UF), linear real arithmetics (QF_LRA), difference in linear real arithmetic (QF_RDL), differences of the integers (QF_IDL), fixed-with bit vectors (QR_BV), combination of uninterpreted functions and differences of the integers (QR_UFIDL), combination of uninterpreted functions and linear arithmetic on the rationales (QF_UFLRA). **Input formulas:** SMT-lib language. **Interface:** A feature of OpenSMT is to provide an automatic mechanism to compute reasons for theory-deductions by means of the consistency check algorithm. This makes it easier the implementation of a new solver, as the solver is required just to detect deduction without having to provide an immediate explanation for them, and without storing any additional information. This mechanism enables the possibility of performing theory-propagation for inherently complex theories such as bit-vectors.	**Api:** OpenSMT provide three library Interfaces for C, C++, ad Python. **OS Platform**: Linux, MacOS OpenSMT consist of a client-server architecture based on TCP/IP with the aim of solving SMT instances harnessing parallel computing. OpenSMT includes a parser for SMT-LIB language, a state-of-the-art SAT-Solver, and a clean interface for plugging in new theory-solvers; a template (empty) theory-solver is provided, to facilitate the development of solvers for other logics. There are two versions of OpenSMT, the second is currently under active development.	Solvers in OpenSMT competed in the SMT-COMP 2014 placing in the mid-range in the QF_UF category. The version submitted in the competition had inefficiencies such a native implementation of theory propagation. there is the link that contains the results of the competition http://smtcomp.sourceforge.net/2014/results-QF_UF.shtml?v=1403902163
raSAT	**Underlying logic:** raSAT is a solver for polynomial constraints over reals. For inequalities equations, it applies raSAT Loop which consists in a simple iterative approximation refinement that extends the interval constraint propagation with testing to boost SAT detection. For equations, a non-constructive reasoning based on the generalized intermediate value theorem is applied. **Background theories:** raSAT SMT solver is organized in a very lazy approach for an arithmetic theory T over reals. As a preprocessing, raSAT converts a polynomial constraint into conjunctive normal form (CNF) by Tseitin conversion In addition, the APCs are preprocessed so that the constraint becomes a CNF containing only > and =. Then, first, each APC is assigned a Boolean value (true or false) by an SAT solver such that ψ is evaluated to true. Second, the Boolean assignment is checked for consistency against the theory T. **Input Formulas:** SMT-lib language. **Interface:** the SAT solver miniSAT manages the Boolean part of the DPLL procedure.	**Api:** Not reported. **OS Platform:** Linux	raSAT has participated SMT Competition 2015, in two categories of main tracks, QF NRA and QF NIA. The results was 3rd in QF NRA, raSAT solved 7952 over 10184 and 2nd in QF NIA, raSAT solved 7917 over 8475.

Name	Features	Implementation	Performance
SatEEn	**Underlying logic:** SatEEn (SAT Enumeration Engine) is an SMT (Satisfiability Modulo Theories) solver that deals with Quantifier-Free Linear Arithmetic Logic and Quantifier-Free Difference Logic**.**	**Api:** Not reported **OS Platform:** Not reported	SatEEn won first places in QF_LIA and QF_IDL divisions and second place in QF_RDL division of SMT-COMP'09.
	Background Theories: SatEEn adopts the simplex based algorithm for Linear Arithmetic Logic and uses finite instantiation method adaptively for Difference Logic.		
	Input formulas: SMT-lib language.		
	Interface: it has been applied to Constrained Random Simulation in Cadence simulator.		
SMTInter-pol	**Underlying logic:** SMTInterpol is an SMT Solver that can compute formulas for various theories.	**Api:** Java **OS Platform:** Any operating system that supports Java Version 6 or above.	This solver competed in the SMT-COMP placing in second in most of the problems tested. The link of the results is http://smtcomp.sourceforge.net/2017/results-toc.shtml
	Background theories: SMTInterpol supports the combination of the quantifier-free fragments of the theories of uninterpreted functions, linear real arithmetic, linear integer arithmetic and arrays.		
	Input formulas: SMT-lib language.		
	Interface: The commands that supports are: get interpolants, include, reset, get model, simplify and timed.		

Name	Features	Implementation	Perfor-mance
SMCHR	**Underlying logic:** SMCHR (Satisfiability Modulo Constraint Handling Rules) is an implementation of SMT (Satisfiability Modulo Theories) where the theory can be implemented in CHR (Constraint Handling Rules). CHR are a high level rule-based programming language for specification and implementation of constraint solvers.	**Api:** C **OS Platform:** The SMCHR system is currently 64-bit (x86_64/AMD64) for Linux, Mac OSX and Windows. The runtime is based on the DPLL design.	No info available.
	Background theories: SMCHR is an extension of the theorical operational semantics. An execution state is conformed by a 4-tuple <G, S, B, T>. Both G and S are sets of reified constraints (b<->c, where b is a propositional variable). G is a goal and S is a constraint store. On the other hand, B is a conjunction of built-in constraints and T is a set of tuples of the form (r, b1,..., bn) where r is a rule identifier and b1...bn are propositional variables. At the end, there are a solver D that supports Boolean and equality constraints.		
	Input formulas: There are two main types of rules, simplification rules (implication in both directions) and propagation rules (Implication in only one direction). A constraint solver is specified by a set of rules.		
	Interface: The semantics that provide SMCHR are Decide, Solve, Introduce and Apply. A failed state occurs when the built-in sorted in B is unsatisfiable. However, the abstract operational semantics does not specify how and when the Decide transition is applied. For this, SMCHR use a variant of DPLL combined with CHR solving.		
SMT-Rat	**Underlying logic:** SMT-Rat is a SMT compliant implementations of methods for solving non-linear real and integer arithmetic NRA/NIA formulas referenced into models that can be combined theories in order to extend the supporting logics.	SMT-Rat is a C++ library tested under Linux and MacOS. The architecture is composed for modules that fixes a common interface for the SMT compliant implementation of procedures to search the satisfiability question of the supported logics of SMT-Rat. They can be composed to a solver according to a user defined strategy. Also, there are a manager for solve tasks and provide the API.	Delta debugging is an automated testing for analysis of the behavior, for example an error.
	Background theories: The heart of STM solver usually forms a SAT solver. In SMT-Rat the module abstracts the input formula to propositional logic and uses efficient SAT solvers MiniSAT to find a Boolean assignment of the abstraction. Then the module implements the simplex method applying it to linear constraints of any conjunction of NRA/NA constraints.		
	Input formulas: There are a class named Formula that represents SMT formulas which is defined the abstract grammar.		
	Interface: Each module contains a vector with input formulas and the main functionalities are inform() to inform the constraints in a formula, add() to add a formula in the vector, check() to check the satisfiability, push(), pop() and model() to obtain the variables of the formulas.		

Name	Features	Implementation	Performance
SONO-LAR	**Underlying logic:** SONOLAR is a solver for nonlinear arithmetic. **Background theories:** The solver supports the following logics: bit vectors, Bit vectors with arrays, IEEE 754 floating point operations (QF_BV, QF_ABV). **Input formulas:** SMT including: quantifier-free, fixed-sized and bit vector logic. **Interface:** SONOLAR uses bit-blasting to translate bit vector constraints to a Boolean formula and lets a SAT solver decide the satisfiability. The series of word-level simplification rules are applied to the input formula which is then converted to an And-Inverter Graph. After performing bit-level simplifications a Boolean CNF formula is generated and fed to a SAT solver. MiniSAT 2.2.0 and Glucose 3.0 are used as SAT solvers	**Api:** C/C++ **OS Platforms:** Linux and Windows.	There are sample models that use SONOLAR and times of performance are in medium range. The results of this benchmarks: http://www.informatik.uni-bremen.de/agbs/testingbenchmarks/
Spear	**Underlying logic:** Bit vector arithmetic (QF_BV) and a Boolean satisfiability are the theories under the formula is checked. **Background theories:** linear, non-linear and floating point arithmetics (support for the last one is implemented in the iSAT3 version). **Input formulas:** The input formulas are in Dinamics CNF format, that is a standard file format of graph consisting in a number of clauses where clause is a disjunction of a number of variables or their negations. Dinamics CNF is composed by CNF format and SAT format. Other input formula is Spear format that supports bit vectors up to 64 bits and all standard bit vector operations. For the last, Spear also accepts input formulas in SMTlib bit vector arithmetic formula. **Interface:** The operations supported are: bitwise operators (AND, OR, XOR, NOT), predicates operators that produce Boolean results and always require two operations (zero flag, carry flag, negative flag, overflow flag, implication, equal, not equal. unsigned, less or equal, greater or equal, less than, greater than), in then else operators, arithmetic operator, shift operators, cast operators.	**Api:** Not reported. **OS Platform:** The tool is available for research usage under Linux, Cygwin and MacOS. Mac OS X binary is dynamically linked, while all others are linked statically. The main tool used for bug is Calysto. It is a scalable and precise static checker for general purpose code. Calysto checks pointer properties and user provided assertions.	Spear is highly optimized because operations with constants (multiplication, reminder, division) are encoded in a special way so as to minimize the size of the circuit and the number of create Boolean variables. On the other hand, the library interface enables various novel optimization techniques, like reusing learned lemmas among multiple runs.

Name	Features	Implementation	Performance
STP	**Underlying logic:** STP is an efficient decision procedure for the validity (or satisfiability) of formulas from a quantifier-free many-sorted theory of fixed-width bit vectors and (non-extensional) one-dimensional arrays.	**Api:** C, Java **OS Platform:** Windows The basic architecture of STP essentially follows the idea of word-level preprocessing followed by translation to SAT (We use MINISAT and CRYPTOMINISAT). In particular, STP introduces several new heuristics for the preprocessing step, including abstraction-refinement in the context of arrays, a new bit vector linear arithmetic equation solver, and some interesting simplifications. These heuristics help to achieve several magnitudes of order performance over other tools, and also over straight-forward translation to SAT.	STP placed 2nd in the bit vector category in the SMTCOMP 2014, just after the proprietary Boolector system STP placed 2nd in the bit vector category in the SMTCOMP 2011. STP won the bit vector category at SMTCOMP 2010. STP won the SMTCOMP 2006 competition (Bit-vector category) in 2006
	Background theories: STP decides satisfiability over bit vectors, arrays and predicates.		
	Input formulas: The file based input formats that STP reads are the: CVC, SMT-LIB1, and SMT-LIB2 formats. The SMT-LIB2 format is the recommended file format, because it is parsed by all modern bit vector solvers. STP implements a subset of the SMT-LIB2 language; not all SMT-LIB2 features are implemented.		
	Interface: The functions in STP's input language include concatenation, extraction, left/right shift, sign-extension, unary minus, addition, multiplication, (signed) modulo/division, bitwise Boolean operations, if-then-else terms, and array reads and writes. The predicates in the language include equality and (signed) comparators between bit vector terms.		

Name	Features	Implementation	Performance
SWORD	**Underlying logic:** To solve formulas the instances has to be rewritten, i.e., rules for distributivism and commutativity are applied. The resulting of this procedure is translated to an AIG data structure that not only supports and-nodes but also iff-nodes. After rewriting, the bit level data structure is converted into CNF and given to the solvers.	**Api:** C++ **OS Platform:** Linux	SWORD will participate in the QF BV division at the SMT competition 2009 using random seed 10659.
	Background theories: The Solve Engine (Solvers' module) is a DPLL style decision procedure as deployed in many state of the art SAT solvers (While free variables remain, a free variable is assigned, and its implications are propagated. If a conflict occurs, it is analyzed and a conflict clause is learnt.). There is a choice between using the SAT solver's decision heuristic and the decision heuristic of Solve Engine.		
	Input formulas: The parse routine of the solver is based on the grammar of Smtlib2.		
	Interface: SWORD is implemented by modules that can be instantiated for any sub-unit of the formula under consideration. This usage is because they are used for propagation where a translation of CNF would be too expensive and high-level information is exploited inside modules.		

Name	Features	Implementation	Performance
VeriT	**Underlying logic:** VeriT accommodate the formula processing and the proof search performed by the solver. The processing steps are represented using an extensible set of inference rules extend the calculus by support transformations such as β –reduction and congruence with λ –abstractions that can appear in higher- order problems.	**Api:** C, Java **OS Platform:** Linux, OS, Windows The tool is open-source and distributed under the BSD license.	Since SMT-COMP 2016, VeriT has seen little improvements on what concerns the competition. Most relevant improvements are related to trigger-based instantiation. VeriT participates in the following divisions: ALIA AUFLIA AUFLIRA LIA UF UFIDL UFLIA UFLRA QF ALIA QF AUFLIA QF IDL QF LIA QF LRA QF RDL QF UF QF UFIDL QF UFLIA QF UFLRA.
	Background theories: The Solver Module is CDCL(T) that is performed in a stratified manner. SAT solver handles the propositional reasoning, a combination of theory solvers tackles the ground (variable-free) reasoning, and an instantiation module takes care of the first-order reasoning. The plan is to adapt the instantiation module so that it can heuristically instantiate quantifiers with functional variables, and to extend VeriT's underlying modular engine for computing substitutions. But currently the only rules that must be adapted are those concerned with quantifier instantiation		
	Input formulas: The input format is the SMT-LIB 2.0 language and DIMACS, but VeriT is intend also to be used as a standalone library and incorporated in third-party software. The extension of the SMT language (DINAMICS) was in a pragmatic way to accommodate higher-order constructs: higher-order functions with partial applications, λ -abstractions, and quantifiers ranging over higher-order variables. This extension is inspired by the work on TIP (Tools for Inductive Provers), which is another pragmatic extension of SMT-LIB.		
	Interface: VeriT relies on third-party software using them either as libraries or as stand-alone applications. VeriT is also used or integrated with other software		

Name	Features	Implementation	Performance
Yices	**Underlying logic:** Yices is an efficient SMT which supports a rich combination of first-order theories. It is a DPLL-based SAT solver.	**Api:** C++ **OS Platform:** Linux, Mac OS, Windows It is distributed as both a library and a standalone tool. It integrates a DPLL-style SAT solver with specialized theory solvers that handle the first order theories. The core theory solver handles EUF (implementing a congruence closure algorithm) while the satellite solvers provide support for other theories such as arithmetic, bit vectors, or arrays. Its DPLL implementation is quite flexible allowing interaction with the core and the satellite solvers. The SAT solving algorithm is a modern variant of DPLL which uses used in Yices two watched literals, non-chronological backtracking with learned clauses and a VSIDS-like search heuristic. As linear arithmetic solver it uses a Simplex-based algorithm which proves to be efficient specially for sparse problems. When it has to deal with dense difference-logic problem it relies on a specialized solver based on an incremental form of the Floyd-Warshall algorithm which increases its performance. It employs a dynamic form of Ackermann's reduction to handle uninterpreted functions. Yices uses three methods for dealing with universally quantified expressions; its main approach is an extension of e-graph matching.	It uses a specialized propagation rule that aids in improving its efficiency. Yices participated in 21 divisions of the SMT competition of 2017. It won in 11 of them: QF_ALIA, QF_AUFLIA, QF_AX, QF_IDL, QF_NRA, QF_RDL, QF_UF, QF_UFIDL, QF_UFLIA, QF_UFLRA, and QF_UFNIA. In most of the other divisions it had a second place being outperformed mainly by z3, CVC4, and MathSAT.
	Background theories: Bit vectors, arrays, recursive datatypes, rational and integer linear arithmetic, and equality over uninterpreted function symbols. First order quantifiers are supported as well. As it uses Yice´s native language the solver provides support for additional theories.		
	Input formulas: It has its own language, but can manage as well SMT-LIB formulas. The Yices language is related to the PVS and SAL languages but it has a Lisp-like syntax.		
	Interface: Yices offers basic SMT-solving feature, solves weighted MAX-SMT problems (where numerical weights are added to assertions to allow a special search in which the main goal is to satisfy assignments of maximal weight). It also establishes unsatisfiable cores, and constructs models. It receives a formula from a file and decides if it is satisfiable, or not, it can even show an unknown status if it fails to prove that the formula is unsatisfiable. It provides as well bounded model checking, theorem proving, and integrated learning and reasoning. Yices is used by the SAL model checking environment as the main decision procedure, and it is integrated to the PVS theorem prover. Yices is the main component of the probabilistic consistency engine used in SRI's CALO system used as MAX-SMT solver.		

Name	Features	Implementation	Performance
Z3	**Underlying logic:** Z3 is a very efficient theorem prover used in software testing, analysis and verification applications. It is SAT-solver based which is presented as a state-of-the art theorem prover from Microsoft Research. It is a low level tool which works best as a component of other bigger and more robust tools that require solving logical formulas. Z3 exposes many APIs to enable those tools to map into Z3. It doesn`t exist as a stand-alone editor and there are not user-centric facilities for interacting with Z3. The language syntax used in the front ends favor simplicity in contrast to linguistic convenience.	**Api:** C/C++, .Net, OCaml, Python, Java, Haskell **OS Platform:** Linux, Mac OS, Windows, FreeBSD Front-ends interact with Z3 using either a textual format or a binary API. Z3`s architecture is based on a modern DPLL-based SAT solver, a core theory solver (which provides support for equalities and uninterpreted functions) satellite solvers (handling the rest of the theories it supports), and an E-matching abstract machine (for quantifiers). The simplifier processes the input formulas using an incomplete and very efficient rewriting. It applies standard algebraic reduction rules, and performs limited contextual simplification. Then the compiler takes the simplified formulas and converts them into an special data-structure that consists of a set of clauses and congruence-closure nodes. The congruence closure core receives truth assignments to atoms from the SAT solver, the equalities that are asserted by the SAT solver are then propagated by this core using an E-graph following. Nodes in the E-graph point to one or more theory solvers. The core propagates as well as the effects of the theory solvers, such as inferred equalities and atoms assigned to true or false and the atoms that may be produced by the theory solvers. These atoms are subsequently owned and assigned by the SAT solver. The SAT solver uses two-watch literals for Boolean constraint propagation, lemma learning with conflict clauses, and performs non-chronological backtracking. Z3 performs quantifier instantiations through E-matching using new algorithms that identify matches on E-graphs incrementally and efficiently.	During the 2017 SMT-competition it participated in 40 divisions, winning 13 of them: UFLRA, UFBV, BV, QF_NRA, QF_LIRA, QF_FP, QF_BVFP (floating point), NIA, AUFNIA, AUFLIRA, LRA, LIA, ALIA. In the rest it is mostly outperformed by CVC4, Yices2, and Vampire 4.
	Background theories: linear, non-linear, integer and real arithmetics, array, uninterpreted functions, fixed-size bit-vectors, extensional arrays, quantifiers, record, recursive and mutually recursive datatypes. It also uses partial orders and tuples.		
	Input formulas: Simplify, SMT-LIB, Z3, and Dimacs.		
	Interface: Z3 allows model generation, satisfiability and validity checking, verification of unsat cores, proofs and test case generation. It provides as well candidate models, MAX-SMT with multi-objectives functions, and ALLSAT tasks (predicate abstraction).		

Having analyzed the SMT solvers we selected two of them which we pretend to use as instruments in the variability problems' context. Our priority was to provide high performance and scalability as CSP-based solvers tend to have failures in these aspects when evaluating large scale models, we also considered expressiveness as a very desirable characteristic. Thus, in our decision-making solvers which are based on diverse background theories and offered valuable functionalities were defined as the most appropriate.

In the process of selection, and in order to reduce the options, we decided to consider only those solvers that participated in the last SMT-COMP (2017's competition). This, as their results allow a feasible comparison and also because the fact of being part of this type of competition evidences a concern about performance addressed through periodic improvements (in most of the cases). This being so, the contemplated solvers were: Boolector, CVC4, MathSAT, Yices2, Z3, OpenSMT, SMT Interpol and VeriT. The last three weren't picked as winners in any of the benchmarks in which they participated, as a consequence we decided not to choose them. MathSAT only won one of the 11 competitions in which it participated, consequently it seemed not to be the best choice. Boolector proved to have the best performance in three benchmarks, nevertheless its results were very similar to those obtained by Yices2 which proves to be a more competitive solver; as a result, we didn't consider any further evaluation of this solver. Z3 proved to be the most competitive solver winning 13 out of 40 of the competitions in which it participated, Yices2 got 11 out of 21 and CVC4 nine out of 43. Although these results suggested that Z3 and Yices2 were the best selection, we decided to evaluate other aspects on these three solvers in order to make our final pick.

Considering the importance of expressiveness, we evaluated which background theories were supported in each of these solvers; differences weren't conclusive as the three provide support to a big range of theories. In terms of input formulas, we could observe some distinctions, but as CVC4, Yices2 and Z3 could interpret SMT-LIB based formulas, which are the ones we pretend to construct, this criterion didn't help us in our final decision. The functionalities they offer are as well quite similar, the three are supported in the same platforms and they show good performance and prove to be scalable. In fact, the aspects evaluated in the previously presented table evidenced that any of these solvers could be adequate for our purpose which indicates that the detailed evaluation wasn't determinant. Consequently, we relied on the SMT-COMP results and chose

Yices2 and Z3 as the two algorithms to provide us with an analysis of some features models.

4. SOLUTION PROPOSAL

Having determined the algorithms that could be used to extend the actual approaches for evaluating feature and extended feature models, we aimed to increase the range variability models' possibilities from another perspective. As previously mentioned this type of models can be very useful especially in decision-making processes, but their implementation has been limited mostly to technologically-based contexts. The aforementioned leads to a squandering of its possible benefits, thus we meant to extend its use through the construction of a model in a different domain: a health related one.

In the medical domain decision-making processes are essential, consequently variability models may turn out to be, if adequately used, an important support. Furthermore, since time is crucial in this context, and technology can aid in providing useful information in an efficient way, this approach can be useful to quicken the process. One of the possible applications of feature models in the medical context is the field of etiology because knowing the causes of certain diseases is fundamental to define what type of treatment, tests and medicines are the appropriate ones. This is especially significant when the disease being studied turns out to be mortal if the patient doesn't receive early assistance. Additionally, if it turns out that the diseases' consequences can be mitigated with special treatments through an adequate knowledge of the causes, it becomes necessary to determine which are the most probable ones. Furthermore, if establishing the causes is not straightforward variability models and their subsequent analysis support can be vital. This being said, the domain of cerebrovascular accidents(CVA) seems to be one which will become greatly benefited through the use of feature models. Hence, we proceeded to explore it and considered modelling it.

CVAs, commonly known as strokes, are accidents related with blood vessels problems. They occur when the brain's blood flow is interrupted by the blockage or rupture of a vessel. CVAs require immediate assistance as they may be mortal or result in permanent brain damage; the quickest the treatment is performed the better the prognosis.

In strokes brain cells die either as a result of a blockage or of a blood vessel's rupture; they are

known as ischemic and hemorrhagic CVAs respectively. (Ellen, 2016). According to the Texas Heart Institute approximately 87% of CVAs correspond to the ischemic type, and the other 13% are hemorrhagic (Texas Heart Institute, 2016). Ischemic strokes, the most common ones, can be either embolic or thrombotic. The first occur when a clot form in any part of the body and gets to a brain's blood vessel, while in thrombotic ones the clot is initially located in the brain. Hemorrhagic strokes are caused by ruptures that interrupt the normal blood flow; they may occur in a brain's blood vessel being intracerebral, or the rupture may arise in the membrane that surrounds the brain which account for those known as subarachnoid hemorrhages (Mayo Clinic, 2017).

Both types of CVAs are extremely dangerous and can have numerous complications, hence detecting and treating them on time is essential. Fortunately, there are symptoms which help people to recognize their existence; some of them may include sudden headaches, dizziness, face, leg, or arm paralysis specially on one side of the body, walking difficulties, loss of balance and coordination, and darkened vision. Nevertheless, some strokes are asymptomatic which may hinder their detection (Mayo Clinic, 2017).

Even when symptoms are perceived further evaluation is required in order to determine if what happened was actually a stroke. Keeping this in mind, a complete physical examination is executed, history is evaluated, and blood tests and a series of diagnostic exams are performed[7]. Additionally, risk factors such as hypertension, diabetes, smoking and drinking habits, and cardiovascular diseases are evaluated. Depending upon the analysis of these results, and specially taking into account the type of stroke certain treatments are applied; this should drive the patient into a recovery period. During this final stage, other studies are performed in order to establish consequences of the stroke, possible complication and long-term treatments that must be applied (Ellen, 2016).

Although in most cases the patient's available information allows the doctors to provide an adequate service, knowing the causes of the accident would be useful for decision-making during the whole assistance, and can even help to improve service's quality in terms of the treatment's

[7] Among the diagnostic tests the most common ones are the angiogram, the computed tomography (CT) scan, magnetic resonance imagining (MRI), echocardiogram, and electrocardiogram. The specifications of each examination are not included in this paper, as they are considered to be out of our scope.

adequacy. Nevertheless, it isn't an easy task to define what causes CVAs; in many cases it is actually impossible to determine it. However, efforts to explore CVAs' causes have not ceased as knowledge about them not only for every patient, but also for medical research is considered to be fundamental. The aforementioned information proves to be a great input for this search, consequently patient's medical history is an important source to support etiology's studies.

Considering these circumstances, we decided to explore if using variability models could aid in the search for causes. In a first attempt, we built a model based on a medical history to help us visualize the domain's variability. We pretended to view how specific characteristics of a patient's stroke strongly suggested a determinate cause. As a result, we obtained various branches which represented the disease from different perspectives namely, medical history, tests and results, diagnostics, medicines, and procedures. The abstraction made from the medical history allowed us to translate the relationships among the features in each branch into restrictions in the model. The idea was to suggest causes for every configuration (which accounted for each patients' history) and later on to enrich the model with other configurations (additional medical histories). However, posterior analysis drove us to the conclusion that having only one medical history to work with wouldn't be enough to create a general model. Besides that, our primary source wasn't representative as it came from the history of a patient whose characteristics didn't illustrate the typical CVA's case.

In order to model a more general situation we decided to explore other type of approaches finding various researches on CVA's etiology. Taking into account that ischemic accidents are the most common ones, we decided to focus our efforts on the establishment of their causes. We found an interesting study and two algorithms used in ischemic stroke's etiology: 1) a MRI-based algorithm for acute ischemic stroke subtype classification (MAGIC)[8]; and 2) an algorithm for the identification and diagnostic evaluation of patients with cryptogenic ischemic stroke or transient ischemic stroke (cryptogenic algorithm). MAGIC algorithm is composed basically by five steps

[8] MAGIC was used in a study in which 6,624 patients were enrolled; it proved to be a feasible system that could actually aid in classifying stroke subtypes. This study was performed based on a stroke registry of the Clinical Research Center for Stroke with patients hospitalized from July 2011 to May 2013. Results pointed out that the overall intra-class correlation coefficient (ICC) value, which indicates its degree of reliability, was 0.43 (95% CI, 0.31- 0.57) for MAGIC. Further information about the study can be found in (YoungChai, et al., 2014).

1) consideration of other determined causes of stroke; 2) searching for small-vessel occlusions; 3) consideration of relevant artery problems; 4) attendance to recanalization results after thrombolytic therapy; and 5) consideration of recanalization status without thrombolytic therapies. The idea behind this algorithm is that performing a step can determine the cause of the stroke, if that occurs the following steps are omitted; otherwise (if the cause hasn't been established) the subsequent steps should be performed. In other words, the algorithm stops either when a cause is found or when the five steps have been completed (YoungChai, SooJoo, Jong-Won, et al., 2014). With this in mind, it is convenient to explore what is done in each of the steps.

Step 1 refers to evaluating special characteristics that suggest that the stroke was caused by a strange reason due to the patient's medical problems. Step 2 is performed to examine the lesions in order to establish if there are small-vessel occlusions, cardioembolic sources, or relevant stenosis whose existence may suggest a determinate cause (through this step it can be determined that the cause of the stroke was SVO, LAA-LC LAA-BR or UD ≥ 2[9]). Step 3 considers the case when large vessels occlusions exist, which suggest some tests to be performed. Based on the characteristics of the occlusions and the exam's results, stroke's causes may be determined (through step three it can be defined that the cause of the stroke was CE, LAA, LAA-NG, UD negative or UD ≥ 2[10]). Step 4 analyzes the results of thrombolytic therapies, when applied, and establishes the corresponding causes according to them (in this step the stroke's causes considered are CE, LAA, or UD ≥ 2). Finally, Step 5 evaluates, through follow-up tests, the recanalization status of the occluded artery when thrombolytic therapies weren't performed, determining that the cause was CE, LAA or UD ≥ 2. The details of each step are illustrated in Figure 3.

[9] SVO, small vessel occlusion; LAA, large artery atherosclerosis; LAA-BR, large artery atherosclerosis with branch atheromatous; and UD ≥ 2, two or more undetermined causes.

[10] CE, cardioembolism; LAA, large artery atherosclerosis; LAA-NG, large artery atherosclerosis with normal angiography.

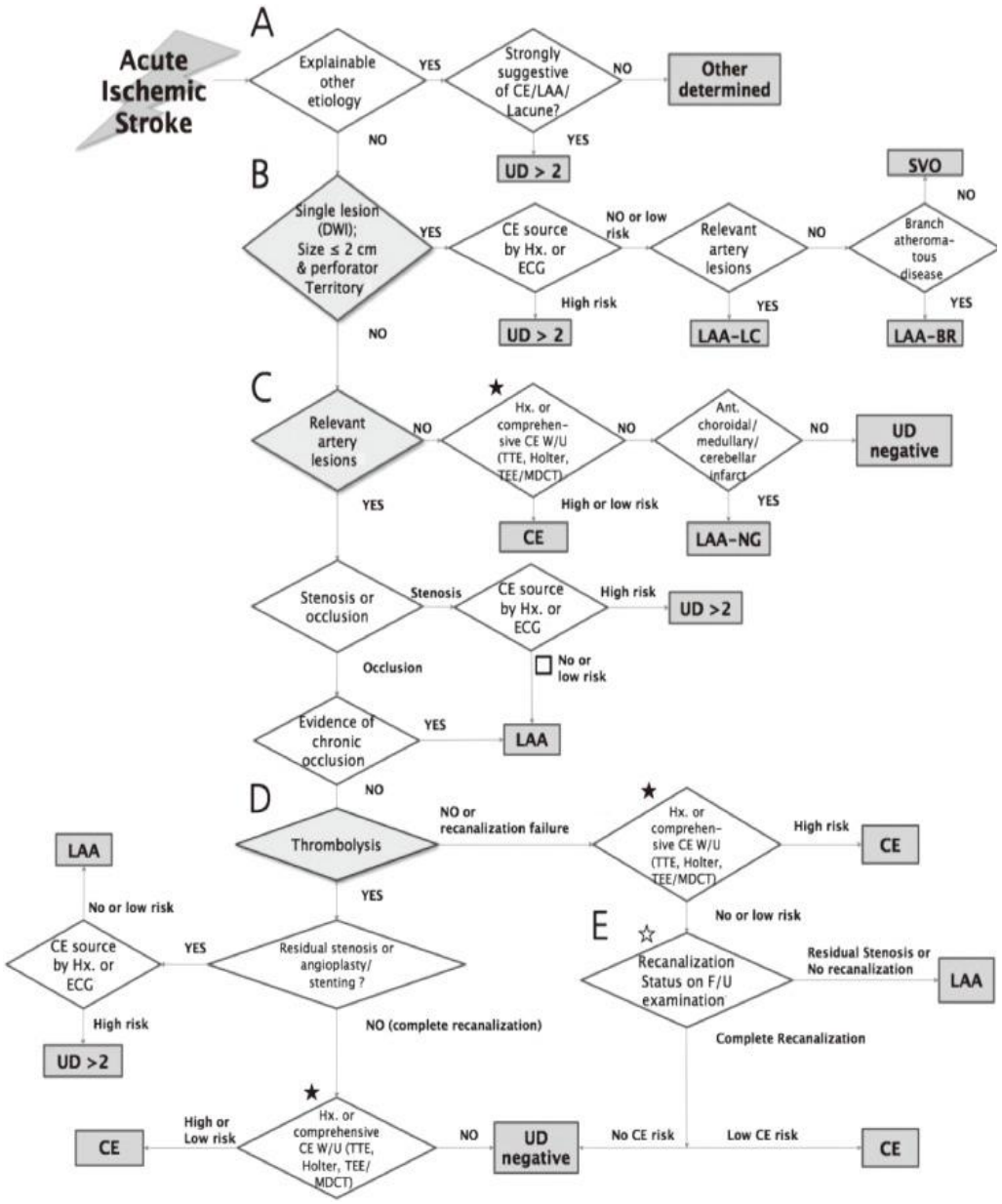

Fig. 3 MAGIC Algorithm[11]*(YoungChai, et al., 2014, p.163).*

[11] (A) Step 1. (B) Step 2. (C) Step 3. (D) Step 4. (E) Step 5. LAA, large artery atherosclerosis; SVO, small vessel occlusion; CE, cardioembolism; UD, undetermined cause; UD ≥ 2, two or more undetermined causes; DWI, diffusion weighted image; Hx, history; ECG, electrocardiography; LAA-LC, large artery atherosclerosis with lacunae; LAA-BR, branch atheromatous dis- ease; W/U, work-up; TTE, transthoracic echocardiography; TEE, transesophageal echocardiography; MDCT, multi- detector row computerized tomography; Ant, Anterior; LAA-NG, large artery atherosclerosis with normal angiography; F/U, follow-up.

MAGIC was used as the main input for the model which was built considering the suggested steps. The information used to evaluate what is considered in every step, in order to determine the causes, was modelled as features in the different branches of the model. To establish if a cause could be defined or if the following steps should be performed cross-tree constraints were created; the considerations taken in every step were the main source for developing the pertinent constraints. For example, in step 2 if the lesion's size is small and cardio embolisms of high risks are founded this suggests that the cause is CE; which turns out to be a constraint through which the selection of the features implies the selection of CE as a cause. The branches of model were built based on the cryptogenic algorithm which should be examined in detail.

The cryptogenic algorithm as well as MAGIC searches to establish the strokes' causes; it doesn't directly suggest which is the most probable one but determines steps to be followed in order to determine it (in this case steps correspond to evaluations). This algorithm should be executed in the same way as MAGIC: if a cause is determined the process is completed, otherwise further tests should be performed. There are four types of evaluations namely, physical, standard, advanced, and specialized, being each type of test more specific than the previous ones. After the examinations, considered as part of each evaluation step, are performed their results are analyzed and it is established if the stroke is considered cryptogenic or if a cause can be determined. The specific details of the algorithm, its steps and its corresponding examinations are illustrated in Figure 4.

With the support of these two algorithms and some data gathered from the medical history the final proposal was developed. We built a model with various branches corresponding to the types of evaluations taken into account in the cryptogenic algorithm i.e., history, physical, standard, specialized and advanced evaluations. However, information needed to execute MAGIC and important data from the medical history couldn't be included in those branches; we created a procedures branch were that data was modelled, and a branch with the causes was added. The constraints, as aforementioned, were mainly built-up based on MAGIC's considerations. Figures 5-8 illustrate some of the feature model's branches; this model should be viewed just a first attempt to explore CVA's etiology domain.

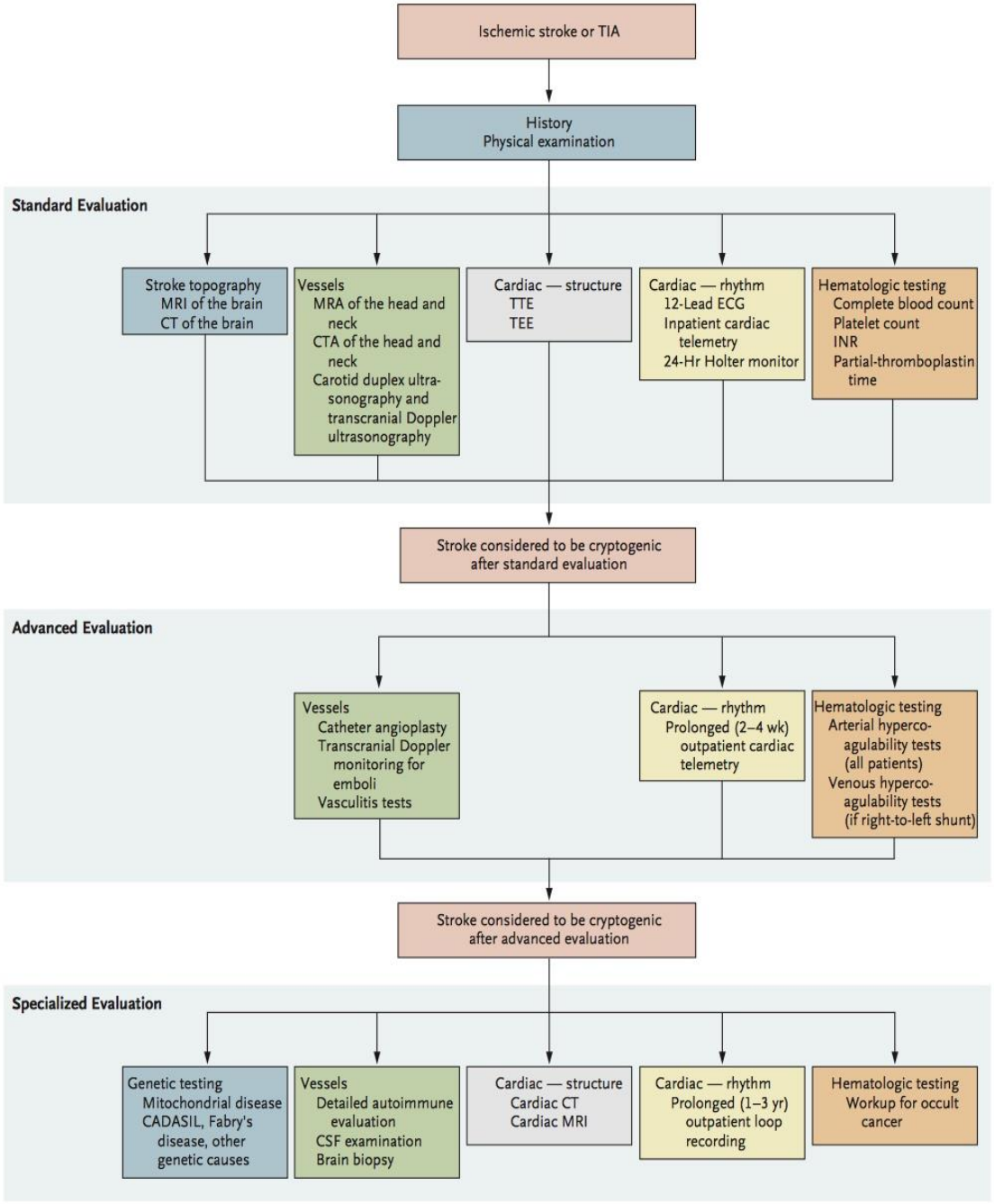

Fig. 4 Cryptogenic Algorithm[12](Saver, 2016).

[12] CADASIL denotes cerebral autosomal dominant arteriopathy with subcortical infarcts and leukoencephalopathy, CSF cerebrospinal fluid, CTA computed tomographic angiography, ECG electrocardiogram, INR international normalized ratio, MRA magnetic resonance angiography, TEE transesophageal echocardiography, and TTE transthoracic echocardiography.

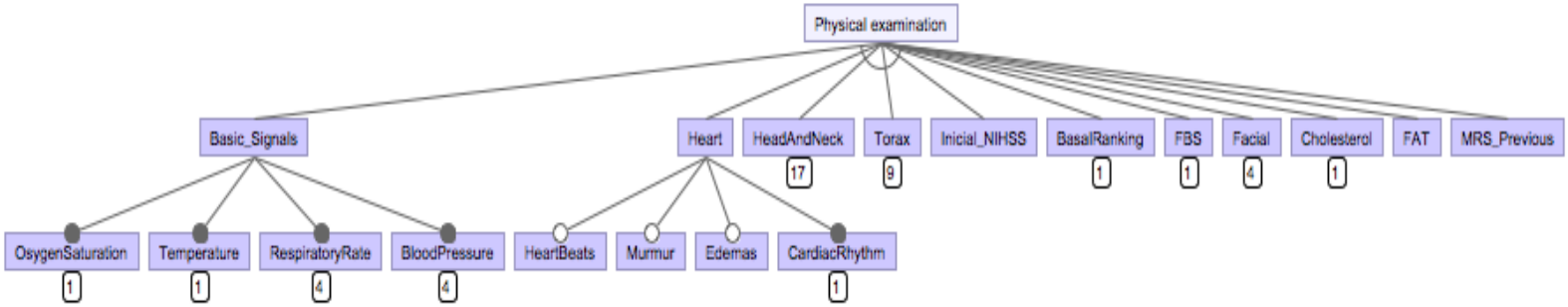

Fig. 5 Physical Examination branch.

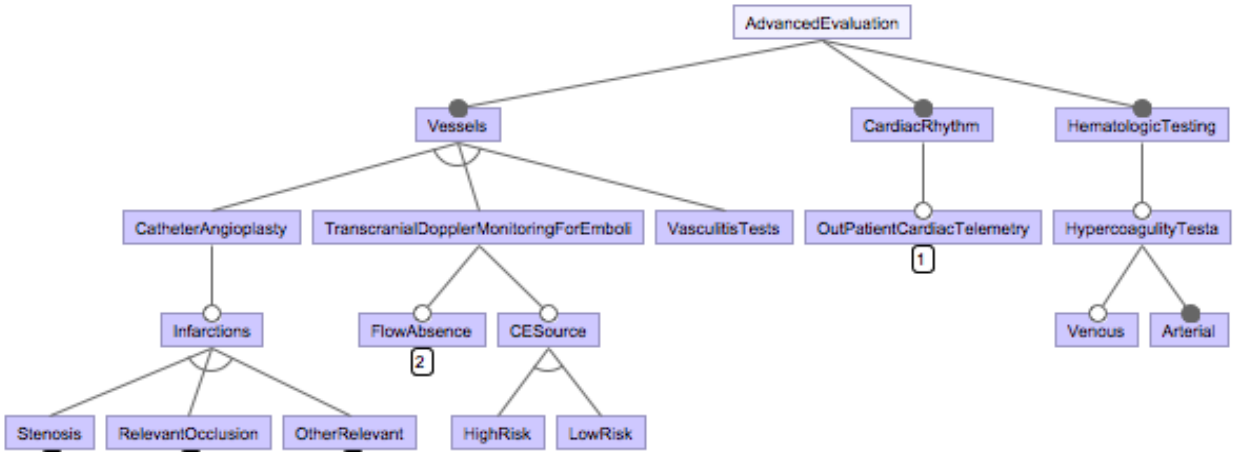

Fig. 6 Advanced Evaluation branch.

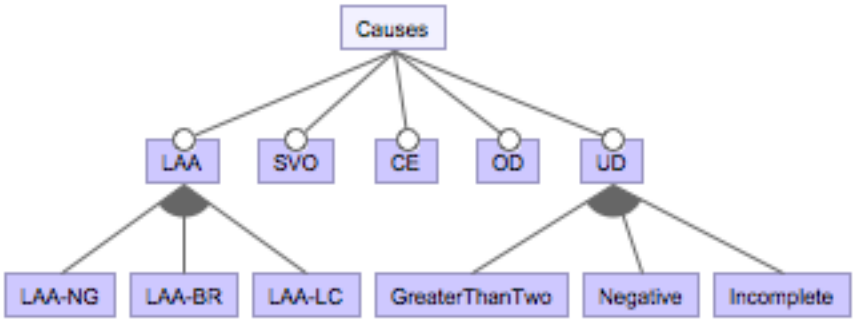

Fig. 7 Causes branch.

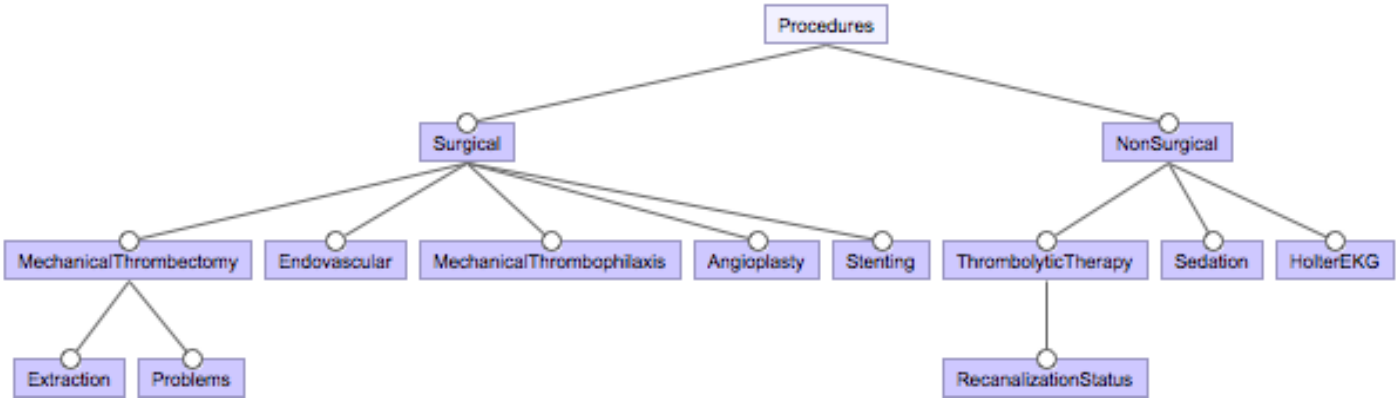

Fig. 8 Procedures branch.

The previous figures show branches of the whole model, illustrating just the features considered and the hierarchic structures. The attributes are not included because, as previously mentioned, FeatureIDE doesn't support AFM creation. Nevertheless, the complete model was constructed in FaMa with constraints and attributes being the AFM that will be used as input for the analysis that will be performed in CoCo. Our purpose isn't only to exemplify how feature models can be employed in domains where its usage hasn't been contemplated, but also to evidence the support they can provide in decision-making processes. Thus, we meant to take advantage of the possibility of performing analysis with the model using a tool that has proven to be especially efficient for this goal.

CoCo is able to interpret FaMa models, providing analysis, evaluating solutions constraints, and suggesting appropriate configurations. Its approach is to transform AFM, the decision rules, and the stakeholders' configurations into two particular CSP implementations that will be evaluated by Choco and JaCoP solvers. This CP approach has proven to give suitable results that conform with the established constraints and to provide expressiveness which is desirable for solution constraints and decision rules' building. Nevertheless, this type of approximation doesn't give a satisfactory response to performance and scalability concerns. Consequently, we explored the possibility of employing SMT algorithms because of their high efficiency and expressiveness.

After a detailed evaluation of SMT solvers we decided that Z3 and Yices2 were the best options, considering the possibility of later on including them in CoCo to improve its performance. To achieve this, it is necessary to previously explore how Z3 and Yices2 can be used in the feature

model's context as they haven't been employed before in this particular way. In this regard, our proposal is quite ambitious considering that although the SMT approach has been visualized as a feasible and efficient solution for feature model's analysis there aren't actual implementations in this direction. Taking this into account, the first approximation is to translate the AFM into a format that can be read and interpreted by these solvers, i.e. an input formula supported by them. Considering that both solvers can understand SMT-based formulas we decided to transform the AFM into SMT language.

As a first attempt we translated a small AFM through which we could efficiently illustrate the transformation process, making plausible how each element should be traduced. We took the Car model as the AFM example and we expressed it in SMT language. After exploring SMT syntax and its structures we concluded that the best way to do it was using a tree defined as a recursive datatype in SMT language. The following code illustrates how this datatype is defined:

```
(declare-datatypes ( (Tree 1) (TreeList 1) ) (
  ; Tree
  (par (X) ( (node (value X) (children (TreeList X)) )))
  ; TreeList
  (par (Y) ( (empty)
    (insert (head (Tree Y)) (tail (TreeList Y))) ))))
```

Fig. 9 Tree structure in SMT.

The tree is constituted by two structures: a **Tree** and a **TreeList.** The **Tree** contains a **node** with a corresponding value (which defines the type of values that will be contained in the whole tree) and a **TreeList** which represents the children of the **node**. The **TreeList** has a **head** and a **tail** which are a **Tree** and a **TreeList** respectively (Barret, et al., 2010).

In order to achieve an appropriate translation, we should pair every AFM's element with its correspondent in SMT language, thus we developed Table 2 with that purpose in mind. The first step is to define the model with its features; each node represents a feature and its children are either feature attributes or other features. Taking into account, that the whole tree can only have one type of values, we decided to make Boolean trees. This allows us to create configurations; if

the feature is selected then it has *true* as its value and it is f*alse* otherwise. Nevertheless, this imposes limitations on attributes' values as they can only be Boolean because the datatype definition forces all its elements to be of the same type. This fact doesn't preclude the construction of the model, but diminishes expressiveness. The following excerpt of code exemplifies the building of the model in which the principal **node** is the **car** which has as its children the features **body** and **engine** contained in the **TreeList** called **general**.

```
 (declare-datatypes (T) ((Tree carTree (node (value T) (children
TreeList)))
                        (TreeList list (cons (head Tree) (tail
TreeList)))))
(declare-const car (Tree Bool))
(declare-const general (TreeList Bool))
(declare-const body (Tree Bool))
(declare-const engine (Tree Bool))
```

Fig. 10 Car model building in SMT.

After building the structure we should generate the constraints, to do so we basically defined a function and then assert it. Considering that every element has a Boolean value the functions are defined in these terms, allowing us to create tree, integrity, and cross-tree constraints[13]. For example, to express the fact that according to our model car's engine can only be defined as electric or using gas we developed a cross-tree constraint that illustrates it. The code which accounts for it consists of the definition of a function, called *conjecture1*, followed by its assertion. The following code shows how this actually looks.

```
(define-fun conjecture1 () Bool (=> (value electric) (not (value
gas))))
(assert conjecture1)
```

Fig. 11 Cross-tree constraint in SMT.

[13] The specifications that explain how to translate each type of constraint as well as the operators used are illustrated in Table 2.

We should also be able to create configurations establishing which features are selected and determine values for some attributes, to do so we use asserts. In accordance with the previously referred constraint we can exemplify an excerpt of a configuration in which the car is chosen to be electric. The subsequent code gives value to both electric and gas, but in fact we could not assign an specific value to *gas* as the preceding constraints will automatically make it false to guarantee that the model is valid.

```
(assert (= (value electric) true))
(assert (= (value gas) false))
```

Fig. 12 Configuration definition in SMT.

Finally, we should translate cross-model constraints when we have more than one model. To exemplify this, we created another model which we called **Renault** and we made constraints that represent the relationships between both models (car and Renault). The model represents a Renault car that can have different models, the constraint accounts for the fact that if the model is **Twizy** then it has to be **electric**[14]. The Renault's model is illustrated through Figure 13, the cross-model constraint's code corresponds to the one shown in Figure 14, and the translation considerations are compiled in Table 2.

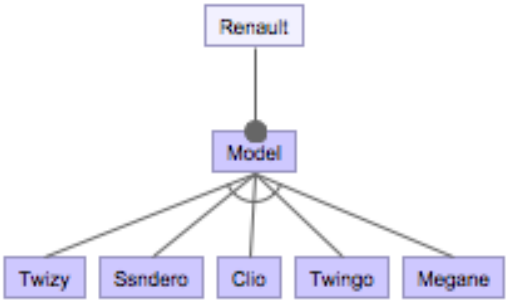

Fig. 13 Renault model.

[14] The Renault's model can be viewed as a branch of the car model nevertheless, it was modelled in this way to exemplify cross-model constraints. Additionally, it is also important to notice that Renault's models should be an attribute whose values are Twizy, Sandero, etc., but attributes can't be shown in FeatureIDE's models.

```
(define-fun conjecture7 () Bool (=> (value electric) (value Twizy)))
(assert conjecture7)
```

Fig. 14 Cross-model constraint in SMT.

Model	SMT
Model Structure	(declare-datatypes (T) ((Tree leaf (node (value T) (children TreeList))) (TreeList listTree (cons (head Tree) (tail TreeList)))))
Initialization Model	(assert (not (= NAME_TREE (as leaf (Tree Bool)))))
Tree	(declare-const NAME_TREE (Tree Bool))
Initialization tree value (node)	(assert (=(value NAME_TREE) true))
List	(declare-const NAME_LIST (TreeList Bool))
Initialization List	(assert (not (= NAME_LIST (as listTree (TreeList Bool)))))
List referenced to tree	(assert (=(children NAME_TREE) NAME_LIST))
Trees differenced to lists	(assert (not (= ANOTHER_TREE (as leaf (Tree Bool))))) (assert (not (= ANOTHER_LIST (as listTree (TreeList Bool)))))
Head	(assert (=(head NAME_LIST) ANOTHER_TREE))
Tail	(assert (=(tail NAME_LIST) ANOTHER_LIST))

Constraints	SMT
AND	(define-fun conjecture1 () Bool ((and (and (value NAME_TREE) (value ANOTHER_TREE)))
OR	(define-fun conjecture2 () Bool ((or (or (value NAME_TREE) (value ANOTHER_TREE)))
NOT	(define-fun conjecture3 () Bool ((not (value NAME_TREE))))
IMPLIES	(define-fun conjecture4 () Bool (=> (value NAME_TREE) (value ANOTHER_TREE)))
NOT IMPLIES	(define-fun conjecture5 () Bool (=> (value NAME_TREE) (not (value ANOTHER_TREE))))

With the model expressed in SMT language we can use the algorithms to evaluate and analyze it, taking into account that solution constraints should be constructed in the platform we decide to use to perform the analysis. Z3 and Yices2 can receive as input formulas SMT-based ones, as consequence they can actually interpret the translated model. The following step is to provide the model to the corresponding algorithms and run each of its Apis in a platform that allows the analysis to be performed. With the translation standards, any feature model can be translated into SMT language, as a result we could actually transform the CVA's FaMa written model into a SMT based one. Later on, the analysis that Z3 and Yices2 provide can be performed and an evaluation of its results can be accomplished. This will indicate either that the chosen solvers are appropriate and that an extension of CoCo which implements them is a valuable effort that should be done, or that other approaches should be considered as this one didn't prove to be efficient enough.

5. VALIDATION

The adequacy of both Z3 and Yices to extend CoCo, as well as the efficiency and feasibility model constructed to illustrate the ACV's domain should be evaluated. The importance of these validations seemed evident, as a result we contemplated different strategies to perform them. We explored diverse ways that could lead us to pertinent results as to conclude whether our initial approach and our solution proposal was adequate. In the first place, we considered evaluating the algorithms through the interpretation of a small model written in SMT language (the car model presented in the previous section). In the second place, we planned to validate our model, written

as a AFM in FaMa, with the aid of CoCo's analysis. Furthermore, we intended to perform some tests with domain's experts.

Z3's and Yices' validation wasn't possible due to the fact that there is not enough documentation about SMT files' reading. Nevertheless, we came across with interesting Java's implementations that allowed the construction of models in each of these solvers. However, this type of models' building didn't seem to be compatible with AFM construction which confirms the need to translate AFM into SMT language. Still the possibilities that can flourish from the exploration of Yice's and Z3's models are very interesting and should be explored as they can complement the analysis and even the AFM models' intents.

We intended as well to use Z3 and Yices to evaluate our model, but that didn't take place as no implementation that allowed SMT-based model could be developed. Still, SMT-lib algorithms-based evaluation wasn't our first approach for validating the ACV's model; our first trial was to analyze CoCo's interpretation and analytic results to rate the appropriateness of the model. However, this validation wasn't performed as CoCo's actual implementation can't read AFM models and ACV's model's feasibility was achieved through the usage of attributes which evidences the need of a AFM model to represent it. We could still try to translate the AFM approach through and FM model that represents the attributes as features and has constraints which account for these transformations; nevertheless, FaMa's expressiveness isn't enough for this to be performed. The incapability to evaluate our model with CoCo's aid indicates not only the need to extend the software in order to broaden the set of inputs it is able to interpret, but also the possibilities that the mentioned extension could bring for CoCo's performance and analysis. Additionally, the exhibition of FaMa's expressive limits reveals the necessity of searching for alternative frameworks for AFM's building.

Given these circumstances we were lead to the consideration of a different approach to validate our model; we decided to explore the option of making an analysis through a Java's implementation. The first idea that came across was to read the model written in FaMa and ask the pertinent questions that could suggest if it needed to be improved or if it accounted as a desirable representation. We followed FaMa's documentation considering that it would be relevant to

validate the number of products built through our model and to visualize one of them. Although the FaMa's manual assured that this type of operations could be performed it didn't advised about the fact that the questions could only be asked when working with FM. As a consequence, we couldn't analyze these results for our specific case given that a FM implementation of our ACV's model wasn't an option; however, we were able to confirm that our model was valid.

With this in mind, we tried to build our model through a specific data structure through which we could validate it, ask the adequate questions, and create solution constraints. Moreover, we explored some alternatives that had been suggested when approaching this type of analysis in variability models. This approximation turned out to be a project itself with hundreds of details to be considered and the need of an ambitious development to be performed. Apparently, there have been numerous initiates searching for implementations that allow variability and feature models' analysis to be carried out with Java's aid. Within the scope of our investigation we weren't able to find an enterprise that satisfied our search or that could offer the validation we were looking for, although we find some interesting proposals.

As a result, of our research we considered the possibility of creating either some tree-like structure or a graph to represent our model. The first alternative appeared as straight-forward as the structure built in the SMT approach was actually a tree; nevertheless, some problems emerged immediately suggesting that this might not be the best choice. A tree allows hierarchy to be modeled, accounting in this way for parental relationships, however an appropriate translation of optional and *or* constraints don't seem to available through this type of structure, additionally the traditional search approach of tree-like structure won't be able to represent variability. The aforementioned doesn't imply that we should discard this option without further consideration, but indicates the need to create a special version of the structure in order to be able to use it; the adequacy of it is called into question. The second alternative appears to be better and there is actually one initiative that makes use of graphs to approach variability models' analysis which supports the suitability of this option.

We came across three different approaches, the first of which included graphs, that appear to take advantage of Java in order to make analysis in the variability's context. In the Software product lines' paradigm, in which variability is considered, efforts related with the implementation of

compositional algorithmic verification suggest the use of graphs to represent the models involved. Schaefer (2011) exploring the importance of scalability in product line verification recommends a compositional approach and a relativization of the global models' properties on local assumptions. He shows how this can be achieved using maximal flow graphs that replace assumptions when verifying globally defined properties[15]. This approximation proposes the use of abstract methods to define the model as an abstract representation of the program's structure and behavior. The method graph appears as an instance of the model obtained by ignoring all data from the specific implementation; a flow graph is a collection of these method graphs. Using Java this type of approach becomes feasible; in every new Java class method graphs are created and implemented exemplifying how the whole flow graph is constructed. In this way, the model can be built, and scalability issues aren't a problem (Schaefer, Gurov, and Soleimanifard, 2011).

The graph's approach seems to be useful for building the model, but it is not possible to effectively try out this solution proposal. Furthermore, no information about the possibility of creating solutions' constraints is available, although its construction seems plausible. It is necessary to evaluate carefully this alternative as to evaluate its adequacy for our search as there is no documentation that exposes the type of analysis that can be performed here. Nevertheless, it seems a valid approximation that enforces the idea of using graphs to create feature models.

A second alternative is proposed as hierarchical variability models are considered in the software product lines' paradigm. Gurov's (2011) recommendation appoints to the creation of Java classes that represent products each of which have a related collection of values. Each class implements a method for setting the elements' values, and two more methods namely, process and compute. This approach allows hierarchy to be modeled, which was not explicit in the previous attempt, nevertheless it is not clear neither in this case how analysis can be performed. Actually, many questions arise within this approach mainly related with the way in which products should be constructed, and if scalability can be achieved through this solution. The graph-based proposal seems more feasible especially considering the size of our model, but no definite conclusions can be exposed as a more profound is needed in order to determine the appropriateness of this solution.

[15] We don't explore or explain the compositional verification approach as it far exceeds the scope of this paper. Further information can be obtained in Schaefer, Gurov, and Soleimanifard, 2011.

The third, and last, option explored seems to be the most appropriate and also the easiest to implement. It is a tool similar to FeatureIDE which includes attributes offering the possibility to create AFMs and not only FMs. The variant management tool used in Beuche 's and Dalgarno's (2006) proposal for product line engineering with feature models is known as pure::*variants* and it supports the association of named attributes with features. "This allows numeric values or enumerated values to be conveniently associated with features e.g. the wind force required to activate the storm alarm could be represented as a "Threshold" attribute of the feature "Storm Alert"" (Beuche and Dalgarno, 2006, p. 13).

This tool has an integrated model transformation and is able to associate solution elements with problems feature, through rule's creation, that account for the creation of solution constraints. It also allows the automatic selection of features when creating products and configurations in an easy way. *pure::variants* [pure] works with *Family models*; a special type of models with hierarchical structure consisting of logical items of the solution architecture. The management variant tool also gets information from source code files which enables the building of a solution from a valid feature configuration (Beuche and Dalgarno, 2006).

The usage of this tool (shown in Figure 14) appears as a very appropriate solution, nevertheless as Beuche and Dalgarno (2006) evidence "managing variable solutions only at the architectural level is insufficient". The need to support variation point at the implementation levels arises and additional work becomes mandatory; in this case, this necessity is met through a simple object-oriented design concept implemented in C++. Product's derivation is now possible as the components of the solution on the implementation level can be included in the Family model and associated with the previously defined rules.

This being done individual products can be created and analysis about the number of possible products can be performed. Valid selection of characteristics from the feature models is guided by pure::variants; once it is found this feature's list with the *Family model* are used as input for the variant model's production. The rules are checked and the products satisfying them are presented as part of the solution (Beuche and Dalgarno, 2006).

The usage of pure::variants with the aid of a particular development to support variation point at implementation levels appears as the most convenient alternative. This, not only taking into account its characteristics and advantages, but also considering our model, its size, the need to use attributes in order to make it feasible, and the type of analysis we want to carry out. Pure::variants can be regarded as an extension of FeatureIDE to support the creation of AFM, incorporating the validations that FaMa allows only for FMs which are now available for attributed feature model. This being said, a detailed evaluation of this alternative is highly recommended and a validation of our model through it turns out to be quite desirable; we suggest this to be later on performed.

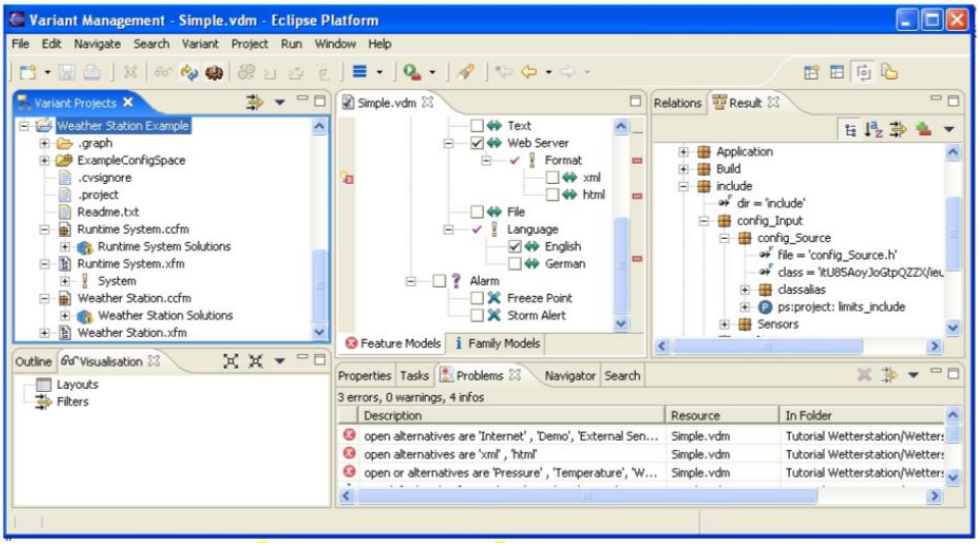

Fig. 15 pure::variants screen shot showing part of the solution space (Beuche and Dalgarno, 2006, p.15).

6. CONCLUSIONS

Our approach to the models' variability's context allowed us to explore how through FMs and AFMs technology, specifically IT's area, can aid organizations in decision-making processes and in the pursue of their stakeholder's objectives. These types of models, especially AFMs, support the visualization of domain's variability in diverse contexts. The attributes contained in these models not only add expressiveness to variability approaches by specifying products' characteristics, but also grant the possibility of establishing rules that reflect stakeholder's aims.

The aforementioned makes this approach a very desirable one, since it tends to resolve the tension between the organization and the support that technology can provide; feature models and their corresponding analysis allow an automatic configuration of products that is driven to the business' goals. Furthermore, they help in the understanding of every domain's particularities, and even include characteristics which allow a better comprehension of the organization's identity. The modelling of this kind of singularities appears as a proof of the potential impact that technology can have in the achievement of a more efficient decision process. This, as a consequence of the fact that a structured understanding of the business is fundamental for the evaluation of alternatives.

Having this in mind, it is indubitable that an extension of the usage of variability models' is an option that should be considered as it may turn out to be beneficial in many aspects. Even more important is the inclusion of this approach in non-technologically driven organizations in which its contribution hasn't been contemplated yet. Taking advantage of automatic product configuration can be determinant for ameliorating efficiency in choosing the best alternative.

Our investigation makes plausible how IT can support critical decision making in complex domains such as the health-related ones. In these contexts, it doesn't only provide a complete and clear picture of particular circumstances, but also accelerates alternatives' evaluation. Considering that time is decisive, especially in health issues, variability model's support may be even more appreciated than in other type of domains where its paradigm has been traditionally used. The aid for establishing causes, as our model pretends to give, may be determinant for accomplishing

medical centers' goals and also, and even more remarkable, for ameliorating the quality of patients' lives.

Etiology should be considered to be a specific health area in which the usage of FMs and AFMs can be of great aid. In CVAs' case the fact that establishing its causes has turned out to be such a thorny task evidences the relevance of modeling this domain. Thus, and taking into account that it is mandatory to be as precise as possible in deciding the most probable causes, it is essential that the model, its results, and it analysis are accurate. Under these assumptions, the proposed model was constructed in accordance with two neurological algorithms whose exactitude has been proved through several studies. Nevertheless, we strongly recommend that expert's validations are performed on the features and attributes included in the final model. It is also mandatory to confirm that the relationships among them are adequately captured in the proposed model; and that the constraints are appropriate for the specific domain. After validating the model pertinent analysis should be performed, which make turn out to be useful in etiology's domain. These analysis, as well as their results should be evaluated by experts and collated with recent studies; this is suggested as future work.

The CVA's model we constructed tried to be as accurate as possible to MAGIC and cryptogenic algorithm's steps and results. As a consequence, our model included a considerable number of features, attributes, and constraints. Nevertheless, its size wasn't big enough to challenge the performance capabilities of CoCo's solvers (this in theory, because we weren't actually able to perform model validation or analysis with the aid of this software). However, specialized analysis and refinements on the model may require complex calculations and substantially increase the model's size, which may diminish the performance's quality of CoCo's and other solvers. Additionally, it has been proven through numerous studies that the traditional CP-based approach has to face scalability and performance issues that can't be easily resolved. This, encouraged the exploration of alternatives to deal with these weaknesses as huge domain's modelling is quite common in variability's context.

SMT-based algorithms arise as an adequate option as they tend to combine the best of CP and SAT's approaches to provide scalability, good performance and expressiveness. A detailed

investigation and comparison of each of SMT-lib's solvers conduced us to select the two most appropriate for our purposes namely, Yices2 and Z3. They have shown to have the best performance and expressiveness which makes them desirable, still they should be rated according to the analysis they can perform over AFMs. This wasn't validated as no SMT models were read by these algorithms; it is suggested as a future work to be performed considering the contribution that can be derived from this kind of approach.

Considering the model validation wasn't carried out through these algorithms, with the aid of CoCo, or other possible approach we can't draw conclusions about its pertinence and correctness. Nevertheless, we can point out the importance of extending Coco's input formula's set as well as the limits of FaMa's approach as the translation from AFMs to FMs in this framework is not fully supported. Besides, we identified that the analysis performed in FaMa are only available for non-extended feature models which drives us to suggest the usage of another tool for building AFMs.

A reasonable approximation for model validation seemed to be a certain Java implementation that allowed its evaluation and the performance of pertinent analysis. As previously discussed this approach could imply the translation of the model into some data structure; tree-like and graph structures were suggested by us, and classes representing products as well as a special graph implementation where recommended in initiatives we explored. Additionally, we came across with a proposal which includes the usage of a management variant tool called pure::variants which is similar to FeatureIDE but offers the possibility to create AFMs as well. The analysis it can perform appear as appropriate. These alternatives have their advantages and issues which should be evaluated in detail to determine which of them must be selected to validate the model.

The work performed leaves a lot of questions and interrogates that claim for further research and implementations. In the specific domain here evaluated a lot of possibilities emerge: 1) it is mandatory to evaluate and validate the proposed model with tools and computational but also, and even more important, with CVAs' experts 2) other studies regarding consequences, treatments, etc. related with CVAs should be considered candidates for modelling through this approach 3) the AFMs and FMs' possible contribution arises as an opportunity to use technologically aid that should be carefully evaluated as well as the possibility of starting new projects through variability

modelling. The inclusion of the health domain opens up the product lines' paradigm perspective which evidences that future work should be focus on applying it in diverse types of domains.

Validating the models also appears as a field in which a lot of investigation should be performed. Future work in this area should focus on implementations that allow extended feature models to be analyzed and constructed. Performance and scalability of solutions as well as the adequacy of the evaluations have to be considered as their importance is indubitable.

7. BIBLIOGRAPHY

Accidente cerebrovascular (2015, July 7) Retrieved from: https://medlineplus.gov/spanish/ency/article/000726.htm

Barcelogic for SMT (2005) Retrieved from: http://www.cs.upc.edu/~oliveras/bclt-main.html

Barrett C., and Berezin S. (2004) CVC Lite: A New Implementation of the Cooperating Validity Checker. In: Alur R., Peled D.A. (Eds.) Lecture Notes in Computer Science, vol. 3114, Paper presented at Computer Aided Verification. CAV 2004, Boston, MA, USA, 13-17 July (pp. 515-518) Springer, Berlin, Heidelberg

Barrett C., and Tinelli C. (2007) CVC3. In: Damm W., Hermanns H. (Eds.) Lecture Notes in Computer Science, vol. 4590, Paper presented at Computer Aided Verification. CAV 2007, Berlin, Germany, 3-7 July (pp. 298-302) Springer, Berlin, Heidelberg

Barret, C., Stump A., and Tinelli C., (2010) The SMT-LIB Standard Version 2.0. Retrieved from: http://smtlib.cs.uiowa.edu/papers/smt-lib-reference-v2.0-r10.12.21.pdf

Barrett C. et al. (2011) CVC4. In: Gopalakrishnan G., Qadeer S. (Eds.) Lecture Notes in Computer Science, vol. 6806, Paper presented at Computer Aided Verification. CAV 2011, Snowbird, USA, 14-20 July (pp. 171-177) Springer, Berlin, Heidelberg

Batory, D. (2005) Feature Models, Grammars, and Propositional Formulas. In: Obbink, H., and Pohl, K. (Eds.), Lecture Notes in Computer Science, vol. 3714, Paper presented at The 9th International Software Product Line, Rennes, France, 26-29 September (pp. 7-20). Springer, Berlin, Heidelberg

Bauer, A., Leucker, M., Schallhart, C., and Tautschnig, M. (2010) Don't care in SMT Building flexible yet efficient abstraction/refinement solvers. International Journal on Software Tools for Technology Transfer, 12, 1, (pp. 23-37).

Benavides, D., Segura, S., Trinidad, P., and Ruiz-Cortes, A. (2007) FAMA: Tooling a Framework for the Automated Analysis of Feature Models. In: Pohl, K., Heymans, P., Chul, K., and Metzger, A. (Eds.), Lero Technical Report 2007-01, Paper presented at the First International Workshop on Variability Modelling of Software-Intensive Systems (VaMoS), Limerack, Ireland. 16-18 January (pp. 129-134).

Beuche, D., Dalgarno M., (2006) Software Product Line Engineering with Feature Models, *Methods & Tools,* 14 (4) Retrieved from: http://www.methodsandtools.com/PDF/mt200604.pdf

Bofill M., Nieuwenhuis R., Oliveras, A., Rodriguez. Carbonell, and E, Rubio A. (2008) The Barcelogic SMT Solver. In: Gupta A., Malik S. (Eds.) Lecture Notes in Computer Science, vol. 5123, Paper presented at Computer Aided Verification. CAV 2008, Princeton, USA, 7-14 July (pp. 249-298). Springer, Berlin, Heidelberg

Brummayer R., and Biere A. (2009) Boolector: An Efficient SMT Solver for Bit-Vectors and Arrays. In: Kowalewski S., Philippou A. (Eds.) Lecture Notes in Computer Science, vol. 5505, Paper presented at Tools and Algorithms for the Construction and Analysis of Systems. TACAS 2009.York, UK, 22-29 March (pp. 174-177). Springer, Berlin, Heidelberg

(MathSAT) Cimatti, A., Griggio, A., Joost Schaafsma, B., and Sebastiani R., (2013) The MathSAT5 SMT Solver. In: Piterman, N., Smolka, S. (Eds.) Lecture Notes in Computer Science, vol. 7995, Paper presented at 19th International Conference, TACAS 2013, Rome, Italy, 16-24 March (pp. 93-107) Springer, Berlin, Heidelberg

Conchon, S., Contejean, E., Kanig, J., and Lescuyer, S. (2008). CC(X): Semantic Combination of Congruence Closure with Solvable Theories. Electr. Notes Theor. Comput. Sci., 198, (pp. 51-69).

de Moura L., Bjørner N. (2008) Z3: An Efficient SMT Solver. In: Ramakrishnan C.R., Rehof J. (Eds.) Lecture Notes in Computer Science, vol. 4963, Paper presented at Tools and Algorithms for the Construction and Analysis of Systems. TACAS 2008, Budapest, Hungary, 29 March - 6 April (pp. 337-340) Springer, Berlin, Heidelberg

Duck, G., (2012) SMCHR: Satisfiability Modulo Constraint Handling Rules, Theory and Practice of Logic Programming, 12, 4-5, (pp. 601-618)

Dutertre, B., De Moura, L. (2006) The Yices SMT solver. Retrieved from SRI International: http://yices.csl.sri.com/tool-paper. pdf 2, 2

Ellen, M., (2016) Cerebrovascular Accident Retrieved from Health Line: https://www.healthline.com/health/cerebrovascular-accident#overview1

FaMa Manual (2010) Retrieved from: http://www.isa.us.es/fama

Ganesh V., Dill D.L. (2007) A Decision Procedure for Bit-Vectors and Arrays. In: Damm W., Hermanns H. (Eds.) Lecture Notes in Computer Science, vol. 4590, Computer Aided Verification. CAV 2007, Berlin, Germany, 3-7 July, (pp. 519-531) Springer, Berlin, Heidelberg.

Gurov D., Østvold B.M., Schaefer I. (2012) A Hierarchical Variability Model for Software Product Lines. In: Hähnle R., Knoop J., Margaria T., Schreiner D., Steffen B. (Eds.) Communications in Computer and Information Science, vol. 336, Leveraging Applications of Formal Methods, Verification, and Validation, ISoLA 2011, Vienna, Austria, 17-18 October, (pp.181-1999) Springer Berlin, Heidelberg

Iguernelala, M., (2013) Strengthening the heart of an SMT-solver: Design and implementation of efficient decision procedures. Other [cs.OH]. Universit Paris Sud - Paris XI.

iSAT Developer Team (2010) iSAT Quick Start Guide. Retrieved from AVACS H1: https://projects.avacs.org/attachments/download/15/quickstartguide.pdf

Kumar, S., Rhishikesh, J., Limaye, S., and Seshia, S., (2009) Beaver: Engineering an Efficient SMT Solver for Bit- Vector Arithmetic. Retrieved from Electrical Engineering and Computer

Sciences University of California at Berkeley: http://www.eecs.berkeley.edu/Pubs/TechRpts/2009/EECS-2009-95.html

Mayo Clinic (2017) Accidente cerebrovascular Retrieved from Mayo Clinic: http://www.mayoclinic.org/es-es/diseases-conditions/stroke/symptoms-causes/dxc-20117265

Michels, R., Ganesh V., Hubaux, A., and Heymans, P. (2017) An SMT-based Approach to Automated Configuration. EPIC Series in Computing, vol.20, Paper presented at 10th International Workshop on Satisfiability Modulo Theories (SMT 2012), Manchester, UK, 30 June-1 July (pp. 109-119). EasyChair

Nieuwenhuis R., Oliveras A., and Tinelli C. (2005) Abstract DPLL and Abstract DPLL Modulo Theories. In: Baader F., Voronkov A. (Eds.) Lecture Notes in Computer Science, vol. 3452, Paper presented at Logic for Programming, Artificial Intelligence, and Reasoning. LPAR 2005. (pp. 36-50) Montego Bay, Jamaica, 2-6 December, Springer, Berlin, Heidelberg

Nieuwenhuis, R., Oliveras, A., and Tinelli, C. (2006) Solving SAT and SAT Modulo Theories: From an abstract Davis--Putnam--Logemann--Loveland procedure to DPLL(T). Journal of the ACM (JACM), 53, 6, (pp.937-977).

Ochoa, L., Gonzalez, O., and Thüm, T. (2015) Using Decision Rules for Solving Conflicts in Extended Feature Models. Proceeding in 8th ACM SIGPLAN International Conference on Software Language Engineering. Pittsburgh, 25-27 October, (pp. 149-160). New York: ACM.

Ochoa, L., Gonzalez, O., Verano, M., and Castro, H., (2016) Searching for Optimal Configurations Within Large-Scale Models: A Cloud Computing Domain. In: Link, S., Trujillo, J. (Eds.) Lecture Notes in Computer Science, vol. 9975, Paper presented at Advances in Conceptual Modeling. ER 2016, Gifu, Japan, 14-17 November (pp. 65-75). Springer, Cham

Ochoa, L., and Gonzalez, O., (2017) Program Synthesis for Configuring Collaborative Solutions in Feature Models. In: CiuCiu, I., et al. (Eds.), Lecture Notes in Computer Science, vol. 10034,

Paper presented at On the Move to Meaningful Internet Systems: OTM 2016 Workshops. OTM 2016, Rhodes, Greece, 24-28 October (pp. 98-108). Springer, Cham.

Ochoa, L., Alves, J., Gonzalez, O., Castro H., and Saake, G. (2017) A Survey on Scalability and Performance Concerns in Extended Product Lines Configuration. Paper presented at the Eleventh International Workshop on Variability Modelling of Software-Intensive Systems (VaMos) Eindhoven, Netherlands. 1-3 February. (pp 5-12).

Reisenberger, C., (2014) PBoolector: A Parallel SMT Solver for QF_BV by Combining Bit-Blasting with Look-Ahead. Johannes Keppler Universität Linz, Linz, Austria.
(Isat) Scheibler, K., Kupferschmid, S., and Becker, B. (2013) Recent Improvements in the SMT Solver iSAT, MBMV, 13, (pp.231-244)

Saver, J., (2016) Cryptogenic Stroke. The New England Journal of medicine, 374, 2065-207. Retrieved from NEJM: http://www.nejm.org/doi/full/10.1056/NEJMcp1503946?af=R&rss=currentIssue&

SMT-COMP 2017 (2017) Retrieved from: http://smtcomp.sourceforge.net/2017/

Schaefer I., Gurov D., Soleimanifard S. (2011) Compositional Algorithmic Verification of Software Product Lines. In: Aichernig B.K., de Boer F.S., Bonsangue M.M. (Eds.) Lecture Notes in Computer Science, vol .6957, Paper presented at Formal Methods for Components and Objects. FMCO 2010. Graz, Austria 29 November to 1 December (pp. 184 - 204).Springer, Berlin, Heidelberg

Stump A., Barrett C.W., and Dill D.L. (2002) CVC: A Cooperating Validity Checker. In: Brinksma E., Larsen K.G. (Eds.). Lecture Notes in Computer Science, vol. 2404, Paper presented at Computer Aided Verification. CAV 2002. Copenhagen, Denmark, 27-31 July (pp. 500-504). Springer, Berlin, Heidelberg

Texas Heart Institute (2016) Accidente cerebrovascular Retrieved from Texas Heart Institute: http://www.texasheart.org/HIC/Topics_Esp/Cond/strok_sp.cfm

Tung, V.X., Van Khanh, T. and Ogawa, M. (2017) raSAT: an SMT solver for polynomial constraints, *Formal Methods in System Design*, 10703 (pp. 1-38)

Wille, R., Jung, J., Sülflow, A., Drechsler, R., (2009) SWORD - Module-based SAT Solving, Algorithms and Applications for Next Generation SAT Solvers 2009, Paper presented at Dagstuhl Seminar, Dagstuhl, Germany, 8-13 November.

YoungChai, K., SooJoo, L., Jong-Won, C., Moon-Ku, H., Jong-Moo, P., Tai Hwan P., . . . Hee-Joon, B., (2014) MRI-based Algorithm for Acute Ischemic Stroke Subtype Classification. Journal of Stroke, 16, 3, 161-172. Retrieved from JoS: http://j-stroke.org/journal/view.php?doi=10.5853/jos.2014.16.3.161

Zankl H., and Middeldorp A. (2010) Satisfiability of Non-linear (Ir)rational Arithmetic. In: Clarke E.M., Voronkov A. (Eds.) Lecture Notes in Computer Science, vol. 6355, Paper presented at Logic for Programming, Artificial Intelligence, and Reasoning. LPAR 2010, Yogyakarta, Indonesia, 10-15 October (pp. 481-500) Springer, Berlin, Heidelberg