Electronic Communications of the EASST
Volume 74 (2017)

7th International Symposium on Leveraging Applications of Formal Methods, Verification and Validation

-

Doctoral Symposium, 2016

Generating Optimal Decision Functions from Rule Specifications

Frederik Gossen Tiziana Margaria

16 pages

Guest Editors: Anna-Lena Lamprecht
ECEASST Home Page: http://www.easst.org/eceasst/ ISSN 1863-2122

Generating Optimal Decision Functions from Rule Specifications

Frederik Gossen **Tiziana Margaria**

CSIS and Lero - The Irish Software Research Centre, University of Limerick
{frederik.gossen, tiziana.margaria}@lero.ie

Abstract: In this paper we sketch an approach and a tool for rapid evaluation of large systems of weighted decision rules. The tool re-implements the patented miAamics approach, originally devised as a fast technique for multicriterial decision support. The weighted rules are used to express performance critical decision functions. MiAamics optimizes the function and generates its efficient implementation fully automatically. Being declarative, the rules allow experts to define rich sets of complex functions without being familiar with any general purpose programming language. The approach also lends itself to optimize existing decision functions that can be expressed in the form of these rules.

The proposed approach first transforms the system of rules into an intermediate representation of Algebraic Decision Diagrams. From this data structure, we generate code in a variety of commonly used target programming languages.

We illustrate the principle and tools on a small, easily comprehensible example and present results from experiments with large systems of randomly generated rules. The proposed representation is significantly faster to evaluate and often of smaller size than the original representation.

Possible miAamics applications to machine learning concern reducing ensembles of classifiers and allowing for a much faster evaluation of these classification functions. It can also naturally be applied to large scale recommender systems where performance is key.

Keywords: Optimal Decision Functions, Rapid Decision Making, System of Rules, Algebraic Decision Diagram, Machine Learning, MiAamics

1 Introduction

As known from computational science, highly frequently invoked functions crucially impact the overall performance of a program. Optimizing such functions is thus particularly likely to pay off. Examples from literature concern for instance loop and nested loop optimizations: they motivated researchers to find ways of expressing the computations in a way that allows for automatic optimization and code generation. In many cases these functions take as an input a description of a situation or a case in case-based reasoning, and return a decision in form of an enumerable value. We call this class of functions *decision functions*.

To represent decision functions, we focus on a rule-based representation where each rule evaluates a situation and assigns a weight to one or more target elements from a set of possible

results. The target elements with the highest cumulative weight[1] are returned as result of the evaluation. We call a potentially large collection of these rules a *system of rules*, described in detail and formally defined in Section 3.

MiAamics. Our approach bases on the award winning approach developed from 2000 to 2001 by METAFrame Technologies GmbH, meanwhile patented [HSM13], and on the ideas of the first miAamics implementation, used also to solve the Semantic Web Service Challenge problems [Kub05, KMS+09]. The core idea was to represent rules as Algebraic Decision Diagrams (ADDs) so that the decision structures can be efficiently precomputed, and evaluate extremely quickly at run time. The ideal application would foresee relatively stable rule sets and many time-critical evaluations for different cases, so that the computation effort needed to be spent only once, and the evaluations would be instantaneous. Indeed, one of the first application domains was personalisation of advertisement and offers in online shops [SMB01, SMB00].

In contrast to the implementation we present in this paper, the original idea was to return a description of target elements, not the elements themselves. In the context of personalization, it would return the pointer to the advertisement, or to the offer page. The use of descriptors/pointers allowed keeping the decision structure in main memory, with full evaluation independent of any database or network access, and to dynamically add elements at runtime, but it imposes one step of indirection.

Simplicity. This rule-based approach is a good example for the simplicity principle proposed in [MS10]. Decisions functions are not fully general, but they elegantly capture the bulk of decision problems in a fashion amenable to powerful optimization techniques that are strong enough to enable rapid decision making. Rules and systems of rules are also much simpler to state and manage than the complex decision structure that they collectively describe. In our previous experience, marketing and campaign managers with no ability to program nor familiarity with databases or query languages were perfectly able to master the rule formats and the rule collections.

Classifiers. Classifiers in machine learning are one domain where rapid evaluation of a decision function is crucial and the proposed rule-based representation is applicable. Classifiers are learned from example data only once, but the learned function is often invoked highly frequently. This can be costly when the evaluation of a learned model or a function is expensive. Past research on rule extraction from such classifiers [Iqb12, Jac05] was partly motivated to get a better understanding of the classification method, but rule extraction can also yield a different representation to evaluate the classifier. These rules are in fact very similar to the system of rules in the miAamics approach. The rule-based representation is therefore well applicable to this class of decision functions.

DSL. The system of rules can also be seen as a domain-specific (description) language giving experts the opportunity to state decision functions in a simple way, that does not require them

[1] The cumulative weight is defined as the algebraic sum of the weights of all applicable rules.

to be familiar with any general purpose programming language. At the same time an efficient implementation is guaranteed by the miAamics approach. Domain experts can focus on the decision function, and not have to worry about efficiency at all. This approach is beneficial even for domain experts familiar with a general-purpose programming language: They can define decision functions once, and generate the corresponding implementation fully automatically in a variety of different target languages.

The central question is how to transform a potentially large systems of rules to an efficient representation and implementation, that ideally saves both memory and evaluation time compared to the original representation of the system of rules.

In Section 2 we present the data structures used in miAamics, itself presented in detail in Section 3. In Section 4 we show how to easily generate and compile code in a variety of different programming languages from the decision data structure, so that it can be included in any application directly. We present the results to date in Section 5 including an evaluation of the miAamics approach for a large system of random rules. Section 7 gives an outlook of our plans to apply miAamics to real world tasks.

2 Algebraic Decision Diagrams

In our approach, we use *Ordered Binary Decision Diagrams* (BDDs) [Bry86, Ake78, BM02] to compactly represent Boolean Expressions. For a given ordering of their predicates they are a canonical representation of the boolean function and can be minimized efficiently. All logical operations can be performed directly on the data structure.

As generalization of OBDDs, *Algebraic Decision Diagrams* (ADDs) [FMY97, BFG+93] are used to represent rules as well as the whole system of rules in miAamics. This is the central data structure we use in miAamics, and we briefly introduce it here. For more detailed information on Algebraic Decision Diagrams (ADDs) we refer the reader to [BFG+93].

ADDs generalize BDDs in that they are associated with an arbitrary algebraic structure rather than the standard boolean algebra. Formally, an algebraic structure is a carrier set $\mathscr{S}$ that is associated with a set of operations $\mathscr{O} \subset (\mathscr{S} \times \mathscr{S} \to \mathscr{S})$. While BDDs represent functions of the form $\mathbb{B}^n \to \mathbb{B}$, ADDs represent functions of the from $\mathbb{B}^n \to \mathscr{S}$. All the algebraic structure's operations on the carrier set $\mathscr{S}$ can be lifted to the data structure.

ADDs are directed acyclic graphs $(\mathscr{V} \cup \mathscr{T}, \mathscr{E})$ with a distinguished root node $\Phi \in \mathscr{V}$. Inner nodes $v \in \mathscr{V}$ are associated with a Boolean predicate $x(v)$. All inner nodes $\mathscr{V}$ have exactly two outgoing edges that determine the *then* respectively the *else* successor of the node. Terminal nodes $t \in \mathscr{T}$ have no outgoing edges and hold a result value $s(t) \in \mathscr{S}$ from the algebraic structure's carrier set. The predicates of an ADD's inner nodes are ordered and the sequence of predicates on every path starting with the root node Φ must conform to this order. Figure 1a shows an example of an ADD with an associated algebraic structure over real numbers.

To determine the result of the function represented by an ADD for a given assignment of the Boolean predicates one starts with the root node Φ and traces down to a terminal node along the valuation of the ordered predicates: For every inner node $v \in \mathscr{V}$ the corresponding predicate $x(v)$ is considered. If it $x(v) = 1$ the trace continues with the *then* successor otherwise it continues with the *else* successor. The reached terminal node $t \in \mathscr{T}$ holds the result of the function $s(t)$.

Due to the predicate order enforced by the data structure, every evaluation trace is no longer than the total number of Boolean predicates.

The order of predicates allows for efficient operations on ADDs, similarly to how operations are performed on BDDs. All operations from the algebraic structure can be lifted to the level of ADDs, ensuring a canonical and redundancy-free representation [Bry86, BM02]. As this algebraic structure may be defined over a carrier set $\mathscr{S}$ of arbitrary size, in contrast to BDDs ADDs are not limited to two terminals. In miAamics we use real values to express rule weights, a $\perp$-value to indicate irrelevant paths, and strings to identify target elements.

3 The miAamics Approach

To rapidly evaluate decision functions represented by potentially large systems of rules, in miAamics the initial full representation of the system of rules is partially evaluated and the result is represented as an ADD. This data structure lends itself to be easily transformed to (efficient) code in a variety of different target programming languages, so that the final decision structure is seamlessly integrated in any program.

3.1 A Small Comprehensible Example

To explain how the miAamics approach works we will consider a comprehensible decision function: a small recommender system to decide what wine to serve with a dish. In this example, the function's argument describes the situation: the served dish. We will base the decision on whether or not the dish contains *cheese*, *meat* and *fish*. The decision outcome is the recommendation which wine to serve out of a selection of available wines. For simplicity wines are characterized by three attributes: *red*, *white* and *sweet*. There are many rules for a good choice of wine, but some of them are more important than others. In this example we give high priority to serve white wine with every meal that contains cheese or fish.

3.2 System of Rules

A decision function can be specified in many different ways. A natural language formulation is often expressed as indications, i.e. a collection of rules. We focus on rules with *premises*, a *conclusion* and a *weight* that select one or more elements from a finite set of possible choices called *target elements*.

The premises: Each rule applies in certain *situations* only. Situations are characterized by properties expressed as Boolean predicates that we call *condition predicates*. The *premises* Boolean expression over these condition predicates defines in which situations the rule applies and in which it does not. As the premises of different rules may overlap, multiple rules may apply in a given situation.

The conclusion selects one or more elements from the finite set of *target elements* as the rule evaluation outcome. Just like situations, target elements are characterized by Boolean properties that we call *target predicates*. The *conclusion* of a rule is a Boolean expression over these predicates that selects a subset of the target elements. The conclusion can be seen as the rule's contribution to the overall decision.

The weight of a rule indicates its importance or priority. When a rule triggers, it assigns this weight to all target elements in its *conclusion*. The higher a rule's weight, the stronger its selected target elements will be recommended towards the final decision.

In our wine selection example, the situations considered by the rules concern different categories of dishes, characterized by the ingredients *cheese*, *meat* and *fish*. These are our three (Boolean) condition predicates. Each rule will recommend as conclusions one or more appropriate wines, our target elements. They are characterized as *red*, *white* and *sweet*, our three (Boolean) target predicates. The weight of a rule indicates its relevance, usually as a relative indicator to the other rules. We will include in our example a highly relevant rule that recommends white wines for cheese or fish dishes.

The full recommender system contains a potentially large collection of such rules, that we call a *system of rules* $\mathscr{R}$. To determine the overall decision for a given situation, all rules of $\mathscr{R}$ are evaluated separately. For every rule applicable in the given situation its weight is assigned to the foreseen target elements. All weights assigned to a target element are algebraically added, yielding the accumulated weight for each target element. The final outcome of a system of rules is the set of all highest weighted target elements. While applying each rule separately is simple and intuitive, in practice it is not very efficient. We will see how the miAamics approach allows for a significantly faster evaluation of large systems of rules using an efficient decision structure.

We define the system of rules $\mathscr{R}$ formally, along the input language of the miAamics tool. The set of all target elements $\mathscr{E}$ is defined as a set of unique target elements of the form

$$e_i := name_i, \mathscr{T}_i \quad \text{with} \quad name_i \in [a-z]+ \quad \text{and} \quad \mathscr{T}_i \subseteq \mathscr{T}$$

where $\mathscr{T}$ is the set of all target predicates. Target elements e_i must be unique with regard to both name $name_i$ and characterization $\mathscr{T}_i$. Note that not all the potential $2^{|\mathscr{T}|}$ have to exist.

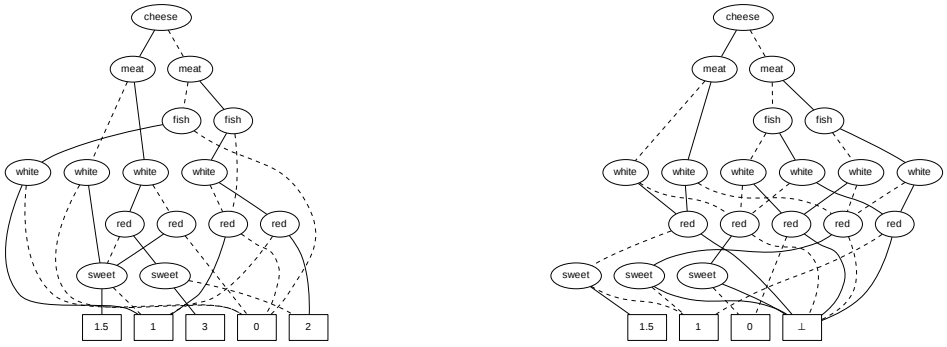
The actual system of rules $\mathscr{R}$ is a potentially large set of rules ranging over the set of target elements. Rules have the form

$$r_j := \texttt{if } c \texttt{ add } w \texttt{ to } t \quad \text{with} \quad c_j \in BE(\mathscr{C}),\ t_j \in BE(\mathscr{T}),\ \text{and} \quad w_j \in \mathbb{R}$$

where $BE(\mathscr{C})$ are Boolean expressions over the condition predicates $\mathscr{C}$ and $BE(\mathscr{T})$ are Boolean expressions over the target predicates $\mathscr{T}$. The rule condition c_j defines the situations in which a rule is applicable, and the rule target t_j defines the set of target elements that are assigned the weight w_j if the rule applies. Given a target element e_i, its characterization $\mathscr{T}_i$ defines the assignment of target predicates corresponding to its selection (in original miAamics language, the target profile).

As we use ADDs, we admit positive and negative weights, expressing indicators in favour or against a certain recommendation. Thus there can be partial or total compensation provening from different rules expressing a variety of recommendation aspects for a given situation profile. In particular, any rule can be neutralized by an identical rule with the negated weight, that cancels its effect and makes it disappear from the decision structure.

Naive evaluation of a system of rules for any given situation by evaluating each rule separately at runtime is simple, and still the standard of many current personalization approaches, but inefficient. MiAamics instead reduces such potentially large systems of rules to a decision structure that saves both memory and runtime.



(a) Rule ADD A_r

(b) Rule ADD over existing target elements A_{re}

Figure 1: MiAamics rule ADDs indicating accumulated rule weights.

3.3 Building the Decision Data Structure

Given as inputs a system of rules $\mathscr{R}$ and the set of target elements $\mathscr{E}$, the miAamics tool derives the ADD that for any given situation described in $\mathscr{R}$ returns all the corresponding highest weighted target elements. This ADD is an efficient global data structure: its evaluation requires at most one test per condition predicate, which is significantly faster than the naive individual rule evaluation already for quite small sets of rules.

To compute the global decision structure for the entire system of rules $\mathscr{R}$, in miAamics we initially build an individual ADD for each rule. Given a rule $(c,w,t) \in \mathscr{R}$ the represented function returns the weight w if both the rule's condition c and its target t are satisfied and 0 otherwise. The rule ADD therefore considers all the condition and target predicates $\mathscr{C} \cup \mathscr{T}$. Once we have the individual ADDs, we can compute the total weight assigned to a target element by summing all these ADDs with the standard ADD sum operation to a resulting rule ADD A_r, as in Figure 1a.

As the sum of ADDs can be computed incrementally, there is no need to keep more than two ADDs in memory at the same time: the current sum ADD and the ADD of the next rule to be added.

Figure 1a visualizes the obtained structure for the following small example system of rules

$$\begin{aligned}\mathscr{R} := \{\ & \texttt{if}\ (\mathit{cheese} \vee \mathit{fish})\ \texttt{add}\ 1.0\ \texttt{to}\ \mathit{white}\ , \\ & \texttt{if}\ \mathit{meat}\ \texttt{add}\ 1.0\ \texttt{to}\ \mathit{red} \\ & \texttt{if}\ \mathit{cheese}\ \texttt{add}\ 0.5\ \texttt{to}\ (\mathit{sweet} \wedge \mathit{white})\ , \\ & \texttt{if}\ (\mathit{meat} \wedge \mathit{cheese})\ \texttt{add}\ 0.5\ \texttt{to}\ (\mathit{sweet} \wedge \mathit{red})\ \}.\end{aligned}$$

This system of rules recommends categories of wines based on their properties, i.e. wine profiles, without knowing what actual wines are available: using a good taxonomy of predicates as target predicates we can express pretty detailed domain knowledge independently of the concrete instances (here the wine sorts) at hand. We will in fact use the distinction between recommendations in principle and feasible recommendations to optimize the decision structure. In our case,

the first rule expresses the recommendation to serve white wine with every dish that contains cheese or fish. The last rule will turn out to be redundant because there will be no sweet red wine in out set of target elements $\mathscr{E}$: this recommendation is not feasible with the wine cellar at hand.

Feasibility of recommendation is linked to usefulness: When transforming the rule ADD A_r to return only the highest weighted target elements it is important to ensure that these elements actually exist. As rules express general knowledge, they can target non-existing or non available elements. We express the set of available target elements by means of a Binary Decision Diagram B_e over target predicates $\mathscr{T}$, which is simply the disjunction of all existing target elements $\mathscr{E}$. To restrict the target consideration to existing elements when selecting the highest weight, we transform the generic rule ADD A_r to suppress non-existing elements. This filtered rule ADD A_{re} is obtained with a binary operation on the two ADDs A_r and B_e that evaluates to a reserved value $\perp$ if the element does not exist and to the accumulated rule weight otherwise. Formally the operation can be defined as

$$f : \mathbb{R} \times \mathbb{B} \to \mathbb{R} \cup \{\perp\} \quad \text{defined as } f(w, b_{exists}) = \begin{cases} w & \text{if } b_{exists} \\ \perp & \text{otherwise.} \end{cases}$$

Given a situation, A_{re} returns the sum of weights for existing target elements as before and $\perp$ for non-existing ones. Figure 1b visualizes the result for the wine selection example with the following target elements

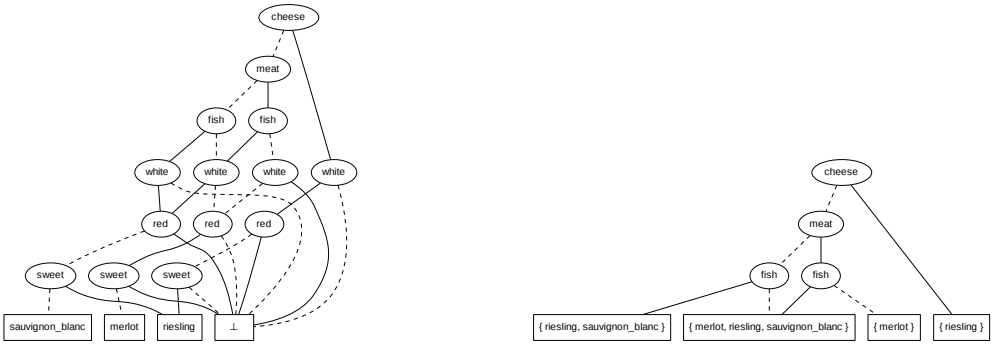
$$\begin{aligned} \mathscr{E} := \{\ & (\mathit{sauvignon\ blanc}, \{\ \mathit{white}\ \}), \\ & (\mathit{riesling}, \{\ \mathit{sweet}, \mathit{white}\ \}), \\ & (\mathit{merlot}, \{\ \mathit{red}\ \})\ \}. \end{aligned}$$

The redundant last rule from the example disappears in the A_{re} ADD because none of the available wines $\mathscr{E}$ is both sweet and red. All paths where the target predicates sweet and red evaluate to true lead to the $\perp$-terminal.

To find the highest weights for every situation, we consider the maximum over the target predicates. The maximum weight still depends on the situation. It is therefore a function from condition predicates to $\mathbb{R} \cup \{\perp\}$ and can consequently be represented as an ADD A_{re}^{max}.

We are actually interested in the top recommendations, i.e., the concrete target elements associated with the maximum accumulated weight, so we compose A_{re} with the elements, replacing the accumulated weights $w \in \mathbb{R} \cup \{\perp\}$ in A_{re}'s terminals with a reference to the actual element $e_i \in \mathscr{E}$ described by the target predicates along that path if $A_{re} = A_{re}^{max}$. Where $A_{re} \neq A_{re}^{max}$ we replace the terminal with the $\perp$-value to indicate the irrelevance of the element described along this path in the associated situation. Figure 2a visualizes the resulting ADD A_f, representing all elements with the highest accumulated weight per situation. For any assignment of condition and target predicates, A_f returns the corresponding top-weighted target element e_i if it exists, or $\perp$ otherwise.

The final ADD $A_{f'}$ should return the best recommendations for each situation, i.e. only evaluate the situation, testing the condition predicates, and return for each situation the set of top weighted elements. We thus replace the target predicates from the ADD with terminal nodes that hold the set of target elements. We reorder the ADD predicates so that condition predicates



(a) Rule ADD over highest weighted target elements A_f

(b) Final ADD $A_{f'}$ testing only condition predicates

Figure 2: MiAamics ADDs indicating best target elements.

precede the target predicates, and replace the sub-ADD that operates only on target predicates with the set of target elements it represents. In the final data structure visualized in Figure 2b, the terminal nodes hold the sets with appropriate wine recommendations.

The rule recommending white wine with cheese or fish dishes influences the final decision structure with a comparably high weight of 1.0. For every dish that contains fish or cheese the decision structure thus recommends *riesling* or *sauvignon blanc* in our example. Only in case of dishes that contain both fish and meat the requirement appears to be violated. This is due to the second rule of the system that recommends red wine with all dishes that contain meat.

Every condition predicate in the final decision structure $A_{f'}$ is evaluated at most once. For sufficiently large systems of rules this data structure therefore allows for a significantly faster evaluation than by individual rule evaluation. The worst case size of the decision diagrams is exponential in the number of predicates, but it is unaffected by the number of rules. This is an essential property: miAamicss most advantageous with large systems of rules ranging over a comparably small number of predicates.

4 Code Generation

The final ADD $A_{f'}$ is optimal in size[2], very efficient to evaluate, and easy to use for direct interpretation at runtime. A given situation is an assignment of condition predicates, so one can easily trace in the miAamics tool the path from the ADD root to the terminal node containing the top recommendation set. However, it is desirable to use the decision structure independently of the miAamics tool, as code embeddable in any user program. To this aim miAamics provides a variety of ways to export its data structure so that it can be easily analysed, interpreted, and compiled to native code.

[2] For a fixed predicate ordering.

The data structure can be exported to `*.dot` format, the input format for Graphviz's dot tool [GN00]. This allows easy visualization of the decision structures, e.g. to generate figures for this paper (cf. Fig. 1 and Fig. 2), and also to reason about the decision structure and to understand it.

The ADD can also be exported in a simple textual format, so that external applications can parse the decision structure and use it in the same way miAamics does internally. This way, an application may analyse the data structure automatically, or work as an interpreter that evaluates the decision structure for a given situation.

While parsing and interpreting the decision structure is faster than evaluating the rule system in the naive fashion, it is still one step of indirection in comparison with a direct integration as source code in the user program. To embed the decision structure directly in code, miAamics provides code generators for a wide range of popular target programming languages including C, C++, Java, C#, Python and JavaScript. These code generators take the decision structure and generate the code of the corresponding function. The function takes as input a situation in form of an assignment of condition predicates and returns the set of target elements with the highest accumulated weight for that given situation.

The characteristics of the target programming language determine how directly the code can be generated from the ADD decision structure. It is important to maintain the properties of ADDs also in the generated target code, meaning that the length of traces through the decision structure should not increase and the size of the representation should not increase by more than a factor. We look at the ADD's graph structure and generate code in the target programming language node by node. It is quite obvious how to test condition predicates and how to return the result from the generated function, but the challenge is to efficiently implement transitions between nodes, and this depends on the target programming language's control flow features. There are two categories of target programming languages for which code generation is very similar:

- Languages comprising `goto`-statements: This control flow feature allows to jump through the code, allowing a direct implementation of ADD node transitions. In this category of target programming languages miAamics today supports C, C++, and C#.
- Languages without `goto`-support: Transitions can be generated as function invocations with one function per node or the decision data structure can be resembled in memory. In this category miAamics supports Java, JavaScript, and Python today.

4.1 Transitions as Goto-statements

Using `goto`-statements, transition from one line in the code to any other can be efficiently implemented. This allows to generate the code per node in an arbitrary order and jump between the different nodes. This is fast and indeed allows to generate every node only once also preserving the length of traces through the decision structure. `Goto`-statements are generally regarded as highly unreadable [Dij68], and this is why many programming languages do not support them at all. However, for generated code that aims at performance, readability is no priority, so this efficiency is welcome. The level on which to read and analyse the decision structure is either as the system of rules from which it was generated or a graphical representation of the decision structure, never the generated code.

4.2 Simulating Goto-statements

If `goto`-statements are not natively supported by the target programming language, the same behaviour has to be simulated. One can trace through the code node by node using auxiliary functions. For every node exactly one function is generated: In case of a constant node this function simply returns the associated value, in case of an internal nodes its predicate is evaluated to delegate the remaining evaluation to one of its successor nodes respectively functions in the generated code. As exactly one function is generated per node again the size of the decision structure does not increase by more than a factor also preserving the length of all traces. Alternatively, the generated code can resemble the decision structure in memory on initialization to then use the structure like an interpreter. In this case code generation allows to skip the parser step in the target application.

5 Results

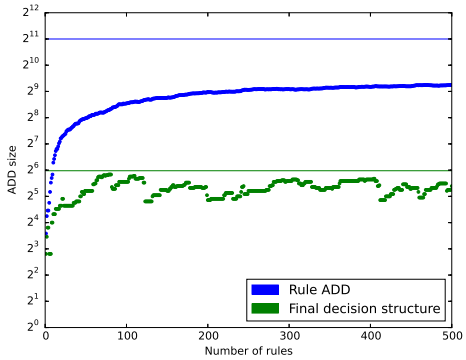
5.1 Implementation in C++

While the idea of miAamics has its roots in 2000 [HSM13, KMS+09, Kub05], and it was provided as a stand-alone and integrated implementation, we reimplemented it now in C++ as a command line tool that in particular supports code generation for native embedding a variety of target languages, and visualization of the data structures. The new miAamics tool was implemented from scratch in C++ using the CUDD library [Som15] for Binary Decision Diagrams and Algebraic Decision Diagrams. The tool reads the rules and the set of target elements in its own input format and generates on demand the corresponding internal individual ADDs. From these individual representations it generates the global ADD decision structure, ready to be exported in a variety of different formats for visualization and for use in other applications. The code generators additionally allow to generate the decision structure as functions in a variety of different programming languages, so that the obtained decision structure can be compiled together with the using application, in which it seamlessly integrates. The current version of miAamics supports C, C++, C#, Java, JavaScript and Python as target programming languages. All these generators preserve the decision structure's properties in the code: the same length of traces through the decision structure, and the size of the representation in code is not greater than the decision structure by more than a factor.

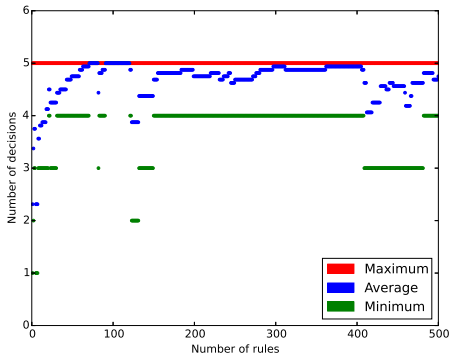
5.2 Evaluation with a Random System of Rules

We evaluated the new tool with regard to memory consumption and running time of the generated decision structure. We decided to use a large set of randomly generated rules in order to avoid the clustering typical to real life systems, where a number of rules typically refer to the same situation or target properties, thus easily inducing optimizations and sharing in the ADDs. In particular, using increasing systems of randomly generated rules allowed us to study the effect of hundreds of thousands of rules that one would hardly write manually. This can however easily be the result of a preceding generation process and thus be a useful intermediate representation.

Scalability with the number of rules is the typical weakness of rule-based approaches that



(a) Growth of the rule ADD and the final decision structure with an increasing number of randomly generated rules. The horizontal lines mark upper bounds for the ADD size.



(b) Minimal, maximal and average number of predicates to be evaluated among a path in the final decision structure.

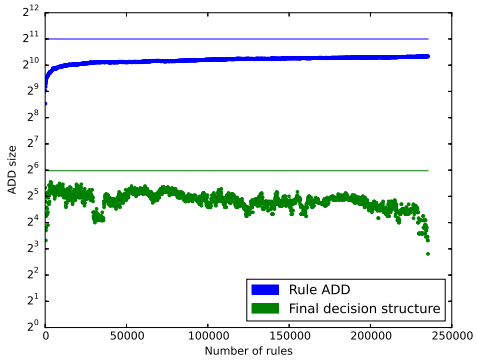
Figure 3: Evaluation with a system of 500 random rules.

evaluate at runtime. We thus looked at the behaviour of the miAamics approach for very large rule systems. To profile the decision structure for a large system of rules, we thus generated many random rules over a given set of condition and target predicates, and measured properties of the decision structure with an increasing number of rules.

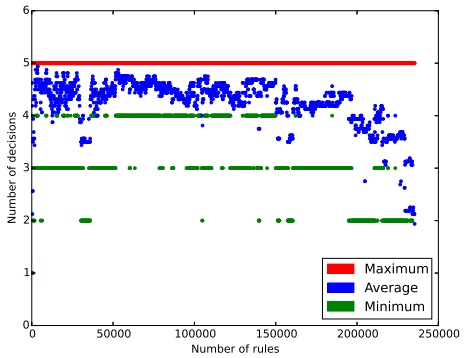
We experimented with 5 condition predicates and 5 target predicates, and considered all the $2^5 = 32$ possible target elements. The rules were generated so that each of them applies to exactly one of the 32 target elements, chosen uniformly at random. The weight w of the rule was chosen uniformly at random in $w \in [0,1]$. Choosing a different bounds would change all weights only by a factor and an offset, but it would not make a difference when selecting the maximum among the weights. Rule conditions were generated as a random Boolean Expression as follows. We generate a syntax tree with 5 levels of Boolean operations. Conjunction, disjunction and negation are chosen equally likely on each level. For the leaf nodes one of the 5 condition predicates are chosen uniformly at random. In this way, we generate random Boolean expressions for conditions that are reasonably complex yet likely to be satisfiable.

Keeping these parameters constant allows us to observe the behaviour of the miAamics decision structure for a growing system of random rules. For the experiment, we add one random rule after the other and record the size of the rule ADD A_r and of the final decision structure $A_{f'}$. We also record the minimal, maximal, and average number of predicates tested in the final decision structure over all possible assignments of condition predicates.

In a first experiment, we observe the behaviour for a comparably small system of up to 500 random rules. The result is visualized in Figure 3. Although the rule ADD grows faster for the first 100 rules than it does for the last 200 rules, it does grow until the 500^{th} rule was added. Interestingly, the size of the final decision structure does not remotely show a monotonic growth: its size drops several times almost in half. Also the number of predicate tests in the final decision structure is not monotone. In fact additional rules can reduce both the minimal number and the



(a) Growth of the rule ADD and the final decision structure with an increasing number of randomly generated rules. The horizontal lines mark upper bounds for the ADD size.



(b) Minimal, maximal and average number of predicates to be evaluated among a path in the final decision structure.

Figure 4: Evaluation with a system of 235,400 random rules.

average number of tests. Note that even the maximum number of test 5 is still a rapid evaluation compared to evaluating each rule separately.

In our second experiment we generated 235,400 random rules in the same way as in the previous experiment. We recorded the same properties, but we computed them only every 100 rules. The result is visualized in Figure 4. The previous observations also hold for this significantly larger system of rules. Between the 200,000 and 250,000 rules the final decision structure appears to shrink while the rule ADD keeps growing. We suspect that this effect is some form of convergence related to the law of large numbers.

6 Related Work

The miAamics approach can naturally be applied to recommender systems where the system of rules would state a strategy according to which a user is recommended products or items in a given situation. These rules can either be defined by experts or they can be derived from potentially machine learned models. ”In recent years, the interest in recommender systems has dramatically increased” [RRS11] with the most prominent examples in online shops [LSY03, SL17] and video streaming platforms [GH15, DLL+10]. While these systems recommend different things the underlying algorithms are similar and the systems suffer similar problems. Even in the context of smart cities ”recommendation techniques become essential tools assisting consumers in seeking the best available services” [ZGN+17].

The most successful and most widely used technique is collaborative filtering (CF) [ZGN+17] where a user is recommended items that ”similar” users have liked before. The algorithm typically involves processing the entire user-item matrix. It is in most cases computationally expensive [LSY03], and that’s why most implementations traditionally resorted to just a few rules/data

points to define similarity, or abstract profiles, that undermined the specificity of the resulting recommendation. Many variants are emerging that take into account not only the user-item matrix but also additional information sources to improve the overall recommendation performance [SLH14].

An alternative category of algorithms are cluster models where customers are clustered into ”similar” groups to find similar customers more efficiently. The much smaller set of items in that cluster then serves as a basis for recommendations [LSY03].

A third class are search-based recommendation methods that search for similar items. These methods scale and perform well only for small sets of previously purchased items [LSY03].

Recommendation algorithms like collaborative filtering (CF) and search-based methods that yield good recommendation performance lack the ability to scale to large user groups and extensive product ranges [LSY03, SL17]. Clustering methods on the other hand yield low recommendation performance [LSY03]. Hence there is the need for other methods Just like miAamics key to the methods scalability is that expensive computations are performed offline before the system is used. Currently, the most widespread across the web is Amazon’s item-based collaborative filtering [SL17].

As both users and items can be described declaratively by means of predicates, we traditionally used the miAamics approach to pre-compute the decision strategies entailed by traditional collaborative filtering and search-based methods. Given a user-item matrix it is also possible to derive a corresponding system of rules and consequently an efficient implementation of the corresponding recommendation strategy.

The rules used in miAamics are relatively simple and understandable and serve as a high-level declarative syntax. In that sense they are similar to Event Condition Action Rules [DGG95, ALT07], used to define application behaviour by means of rules rather than general purpose programming languages. In ECA rules the action is always the selection of target elements or items. The high-level definition of ECA allows for analysis and optimisation techniques [PPW06].

7 Conclusion

The new implementation of the miAamics approach is particularly tailored for rapid evaluation of large systems of rules in ”recommender-style” systems, stand-alone or embedded as native code in applications. We defined formally the rule format, including the description and handling of the target elements it reasons about. We explained in detail the construction of the efficient miAamics internal decision structure, with all its intermediate representations and efficiency-oriented optimizations. We showed how to generate easily property-preserving native code in a variety of different target programming languages from this optimized data structure. In this way we ensure that also the generated code is optimally efficient, in the sense that it has the same computational complexity as evaluating the data structure but it can be included in applications and compiled easily.

In a preliminary evaluation with a large system of random rules, we show that the miAamics decision data structure remains small and ensures short traces during evaluation even for large systems of rules with more than $200k$ rules. In fact, the runtime efficiency depends on the depth of the decision structure, which is determined by the number of predicates (properties)

characterizing the situation and the target elements, and not by the number of the rules. Thus the strength of the miAamics approach lies in the very compact and efficient representation of the knowledge underlying both domains and its independence of the specific relations between all these elements. This makes it particularly promising for applications where rapid decision making is important.

A promising field of application is in machine learning, where classifiers are trained with example data. These learned functions take as an input a description of a situation or a case and predict one of several classes as a result. While training these classifiers is performed only once, they are evaluated frequently. Some classifiers are expensive to evaluate and thus an approach to evaluate them faster can be crucial. For binary descriptions of situations and cases, the miAamics approach is well applicable if the classifier can be expressed by means of the proposed form of rules. In fact, rule extraction from such classifiers has been a topic in research [Iqb12, Jac05] although not necessarily to evaluate them faster but to understand them better.

Of particular interest to us are Decision Trees, Random Forests, Decision Jungles [SSK+13] because they can be easily translated to the proposed form of rules. We also aim to test our approach with more complicated classifiers. The target criteria will again be the running time needed for evaluation, and the memory consumption of the miAamics data structure in comparison with the original representation of the system of rules as well as the original classifier.

A limitation of the miAamics approach to date is that it operates on binary descriptions of situations while it is often the case that other descriptions are more appropriate. Input arguments to classifiers are often real valued. For this reason, we aim to extend the miAamics approach to support real valued situation descriptions in the future, where the Boolean predicates within the decision structure will express comparisons with constants, similarly to how constraints that had been introduced previously to solve the Semantic Web Services Challenge [KMS+09]. We expect the specific strength of miAamics to make a significant difference for the applicability to real life problems of those classifiers.

Acknowledgements: This work was supported, in part, by Science Foundation Ireland grant 13/RC/2094 and co-funded under the European Regional Development Fund through the Southern & Eastern Regional Operational Programme to Lero - the Irish Software Research Centre (www.lero.ie).

Bibliography

[Ake78] S. B. Akers. Binary Decision Diagrams. *IEEE Transactions Computers* 27(6):509–516, 1978.

[ALT07] E. E. Almeida, J. E. Luntz, D. M. Tilbury. Event-Condition-Action Systems for Reconfigurable Logic Control. *IEEE T. Automation Science and Engineering* 4(2):167–181, 2007.

[BFG+93] R. I. Bahar, E. A. Frohm, C. M. Gaona, G. D. Hachtel, E. Macii, A. Pardo, F. Somenzi. Algebraic Decision Diagrams and Their Applications. In *Proceedings of*

the 1993 IEEE/ACM International Conference on Computer-aided Design. Pp. 188–191. IEEE Computer Society Press, 1993.

[BM02] R. E. Bryant, C. Meinel. *Ordered Binary Decision Diagrams*. Pp. 285–307. Springer US, 2002.

[Bry86] R. E. Bryant. Graph-Based Algorithms for Boolean Function Manipulation. *IEEE Transactions Computers* 35(8):677–691, 1986.

[DGG95] K. R. Dittrich, S. Gatziu, A. Geppert. The active database management system manifesto: A rulebase of ADBMS features. In *Proc. of 2nd Int. Workshop on Rules in Database Systems (RIDS '95)*. LNCS 985, pp. 1–17. Springer Berlin Heidelberg, 1995.

[Dij68] E. W. Dijkstra. Letters to the Editor: Go to Statement Considered Harmful. *Commun. ACM* 11(3):147–148, Mar. 1968.

[DLL+10] J. Davidson, B. Liebald, J. Liu, P. Nandy, T. Van Vleet, U. Gargi, S. Gupta, Y. He, M. Lambert, B. Livingston, D. Sampath. The YouTube Video Recommendation System. In *Proceedings of the Fourth ACM Conference on Recommender Systems*. RecSys '10, pp. 293–296. ACM, New York, NY, USA, 2010.

[FMY97] M. Fujita, P. McGeer, J.-Y. Yang. Multi-Terminal Binary Decision Diagrams: An Efficient Data Structure for Matrix Representation. *Formal Methods in System Design* 10(2):149–169, 1997.

[GH15] C. A. Gomez-Uribe, N. Hunt. The Netflix Recommender System: Algorithms, Business Value, and Innovation. *ACM Trans. Manage. Inf. Syst.* 6(4):13:1–13:19, 2015.

[GN00] E. R. Gansner, S. C. North. An open graph visualization system and its applications to software engineering. *Software - Practice and Experience* 30(11):1203–1233, 2000.

[HSM13] H. Hungar, B. Steffen, T. Margaria-Steffen. Device for Generating Selection Structures, for Making Selections According to Selection Structures and for Creating Selection. United States Patent Application Publication, 2013. Appl. No.: 13/650,680.

[Iqb12] M. R. A. Iqbal. Rule Extraction from Ensemble Methods Using Aggregated Decision Trees. In *Neural Information Processing: 19th International Conference*. Pp. 599–607. Springer Berlin Heidelberg, 2012.

[Jac05] H. Jacobsson. Rule Extraction from Recurrent Neural Networks: A Taxonomy and Review. *Neural Computation* 17(6):1223–1263, 2005.

[KMS+09] C. Kubczak, T. Margaria, B. Steffen, C. Winkler, H. Hungar. *An Approach to Discovery with miAamics and jABC*. Pp. 217–234. Springer US, Boston, MA, 2009.

[Kub05] C. Kubczak. Entwicklung einer verteilten Umgebung zur Personalisierung von Web-Applikationen. Master's thesis, TU Dortmund University, 2005.

[LSY03] G. Linden, B. Smith, J. York. Amazon.com recommendations: item-to-item collaborative filtering. *IEEE Internet Computing* 7(1):76–80, 2003.

[MS10] T. Margaria, B. Steffen. Simplicity as a Driver for Agile Innovation. *Computer* 43(6):90–92, 2010.

[PPW06] A. Poulovassilis, G. Papamarkos, P. T. Wood. Event-Condition-Action Rule Languages for the Semantic Web. In Grust et al. (eds.), *Current Trends in Database Technology – EDBT 2006*. Pp. 855–864. Springer Berlin Heidelberg, Berlin, Heidelberg, 2006.

[RRS11] F. Ricci, L. Rokach, B. Shapira. Introduction to Recommender Systems Handbook. In Ricci et al. (eds.), *Recommender Systems Handbook*. Pp. 1–35. Springer, 2011.

[SL17] B. Smith, G. Linden. Two Decades of Recommender Systems at Amazon.com. *IEEE Internet Computing* 21(3):12–18, 2017.

[SLH14] Y. Shi, M. Larson, A. Hanjalic. Collaborative Filtering Beyond the User-Item Matrix: A Survey of the State of the Art and Future Challenges. *ACM Comput. Surv.* 47(1):3:1–3:45, May 2014.

[SMB00] B. Steffen, T. Margaria, V. Braun. Personalized Electronic Commerce Services. In *IFIP WG 7.3 8th Int. Conference on Telecommunication Systems Modeling and Analysis*. Nashville, Tennessee, USA, March 9-12 2000.

[SMB01] B. Steffen, T. Margaria, V. Braun. Coarse-granular model checking in practice. In Dwyer (ed.), *Model Checking Software*. Pp. 304–311. Springer Berlin Heidelberg, Berlin, Heidelberg, 2001.

[Som15] F. Somenzi. CUDD: CU Decision Diagram Package Release 3.0.0. University of Colorado at Boulder, 12 2015.

[SSK+13] J. Shotton, T. Sharp, P. Kohli, S. Nowozin, J. Winn, A. Criminisi. Decision Jungles: Compact and Rich Models for Classification. In *Proc. NIPS*. 2013.

[ZGN+17] H. Zhang, I. Ganchev, N. S. Nikolov, Z. Ji, M. O'Droma. A Hybrid Service Recommendation Prototype Adapted for the UCWW: A Smart-City Orientation. *Wireless Communications and Mobile Computing* 2017, 2017.