Monadic Decomposition

MARGUS VEANES, Microsoft Research
NIKOLAJ BJØRNER, Microsoft Research
LEV NACHMANSON, Microsoft Research
SERGEY BEREG, The University of Texas at Dallas

Monadic predicates play a prominent role in many decidable cases, including decision procedures for symbolic automata. We are here interested in *discovering* whether a formula can be rewritten into a Boolean combination of monadic predicates. Our setting is quantifier-free formulas whose satisfiability is decidable, such as linear arithmetic. Here we develop a semi-decision procedure for extracting a monadic decomposition of a formula when it exists.

CCS Concepts: • **Theory of computation → Logic**; **Algorithm design techniques**; *Automata over infinite objects*; • **Computing methodologies → Symbolic calculus algorithms**;

Additional Key Words and Phrases: symbolic automata, variable independence, satisfiability modulo theories, monadic logic

ACM Reference format:
Margus Veanes, Nikolaj Bjørner, Lev Nachmanson, and Sergey Bereg. 2017. Monadic Decomposition. *J. ACM* 64, 2, Article 14 (May 2017), 27 pages.
DOI: 10.1145/3040488

1 INTRODUCTION

Classical decidability results of fragments of logic [8] are based on careful systematic study of restricted cases either by limiting allowed symbols of the language, limiting the syntax of the formulas, fixing the background theory, or by using combinations of such restrictions. Many decidable classes of problems, such as monadic first-order logic or the Löwenheim class [50], the Löb-Gurevich class [49], monadic second-order logic with one successor (S1S) [11], and monadic second-order logic with two successors (S2S) [62] impose at some level restrictions to *monadic* or unary predicates to achieve decidability.

Problem definition. Here we study the problem of *whether* and *how* we can transform a formula that uses multiple free variables into a *simpler* equivalent formula, but where the formula is *not* a priori syntactically or semantically restricted to any fixed fragment of logic. *Simpler* in this context means that we have eliminated all theory specific dependencies between the variables and have transformed the formula into an equivalent Boolean combination of predicates that are "essentially" unary. We call the problem *monadic decomposition*:

> *Given a finite representation of a nonempty binary relation R, decide if R is equal to a* ***finite*** *union* $\bigcup_{0 \le i < k} A_i \times B_i$ *of finitely represented* A_i *and* B_i.

A relation is called *monadic* if it has a monadic decomposition. For example, the relation $\{(x, y) \mid (x + (y \mod 2)) > 5\}$ over integers is monadic because it has the equivalent monadic decomposition $\{(x, y) \mid (x + 1 > 5 \wedge y \bmod 2 = 1) \vee (x + 0 > 5 \wedge y \bmod 2 = 0)\}$, where $k = 2$, $A_1 = \{n \mid n > 4\}$,

B_1 is odd numbers, $A_2 = \{n \mid n > 5\}$, and B_2 is even numbers. The fundamental assumption we make here is:

We have a Boolean closed class of formulas Ψ *and a* ***solver*** *for* Ψ.

In the given example Ψ would be integer linear arithmetic. More precisely, we assume a background structure $\mathfrak{U}$ with an r.e. (recursively enumerable) universe $\mathcal{U}$ and an r.e. set Ψ of formulas such that:

(1) All elements of $\mathcal{U}$ can be named by ground terms in the language of $\mathfrak{U}$.[1]
(2) All quantifier free formulas in the language of $\mathfrak{U}$ are in Ψ.[2]
(3) For all $\varphi(\bar{x}) \in \Psi$ we can decide if $\mathfrak{U} \models \exists \bar{x} \varphi(\bar{x})$, i.e., if $\varphi(\bar{x})$ is *satisfiable*.

Variable independence. Monadic decomposition was originally studied by Libkin [48] under the name *variable independence* in full first-order theories. Here we revisit and build on some of those results. The aspect that is unique to our work is the specialization to the quantifier free fragment of theory combinations and the use of a *solver* as an oracle. In the general case we do not assume that the full theory of $\mathfrak{U}$ is decidable. Our Theorem 4.6 is related to [48, Theorem 3] where the latter looks at a general class of structures whose theory is decidable and where deciding finiteness and Skolemization is definable in the language. The resulting formulas use quantifiers. Rather than assuming that Skolemization is definable in the language we use decidability of the satisfiability procedure of the solver to construct witnesses.

Working with a solver. The specialization on quantifier free formulas is important because we use *satisfiability modulo theories* or *SMT* solvers [20] in our algorithms. Theory combination in modern SMT solvers uses quantifier free formulas and becomes undecidable in the more general case when quantifiers are allowed [10, 37, 51]. This limits what we are allowed to express in $\mathfrak{U}$ without sacrificing decidability – in general, only positive existential quantifiers are allowed, while other quantifiers are not.

When $\varphi(\bar{x})$ is satisfiable it follows (from $\mathcal{U}$ being r.e.) that we can also effectively generate a *witness* $\bar{a}$ such that $\varphi(\bar{a})$ holds. *Effectiveness* means that there is a solver that uses a finite number of steps for deciding satisfiability and for finding a witness. A *finite representation* of a relation is given by a formula from Ψ. This formulation is very natural from the standpoint of modern SMT solvers. Practically, one can consider an SMT solver as an extension of an effective representation of a Boolean algebra. The theory specific extensions of the solver are used to construct quantifier free formulas in different subtheories but have otherwise no direct influence on the algorithms that we discuss here. We show an executable python script of the main algorithm using the SMT solver Z3 [19] in the Appendix; it demonstrates concretely how such interaction with an SMT solver works in practice.

Semidecision procedure. A formula $\varphi(x, y) \in \Psi$ denotes the relation $R = \{(a, b) \in \mathcal{U} \times \mathcal{U} \mid \mathfrak{U} \models \varphi(a, b)\}$. The main two questions that we are interested in are: 1) deciding if R is monadic; 2) constructing a monadic decomposition of R if R is monadic. The key insight is given by the following equivalence relation $\sim$ over $A = \{a \mid \exists b\, R(a, b)\}$. Let also $R(a)$ denote the image of a by R defined as the set $\{b \mid R(a, b)\}$.

$$x \sim x' \stackrel{\text{def}}{=} \forall y\, y'((R(x, y) \wedge R(x', y')) \Rightarrow (R(x', y) \wedge R(x, y')))$$

Observe that, for all $x, x' \in A$, $x \sim x'$ is equivalent to the condition $R(x) = R(x')$, i.e., that x and x' are indistinguishable with respect to their images by R. Let $A_\sim$ denote the set of $\sim$-equivalence

[1] By slight abuse of notation we use elements of $\mathcal{U}$ also as terms.
[2] Formal equality symbol is $\simeq$ and is always present, i.e., we assume logic *with* equality.

classes of A. Now: 1) $A_\sim$ is finite iff R is monadic, see Lemma 4.3, so we can decide if R is monadic by deciding if $A_\sim$ is finite; 2) we can effectively enumerate witnesses for all the $\sim$-equivalence classes by using the solver as an oracle. This gives us, in the monadic case, a way to subdivide A into its equivalence classes $A_\sim$, by using the solver for Ψ to exhaustively enumerate witnesses that cover $A_\sim$. Such witnesses can subsequently be used to construct formulas for the monadic decomposition. If, on the other hand, R is not monadic, then this enumeration will not terminate, leaving us with a semidecision procedure in the general case. It also follows from Theorem 4.12 that the procedure ***mondec*** is a semidecision procedure that terminates and produces a monadic decomposition ***mondec***(φ) iff φ is monadic.

For the semidecision procedure of monadic decomposition to work, there are no assumptions on $\mathfrak{U}$ and Ψ other than the ones listed above. The technique works in all theories where a *solver* is available, such as *linear arithmetic* over integers or rationals, *bit-vectors*, *arrays*, *algebraic data types*, *algebraic reals*, as well as *combinations of theories* for which a DPLL(T) [29] procedure with Nelson-Oppen [53] style theory combination is known.

Decidable cases. While the general case of monadic decomposition is undecidable [48, Proposition 2], for some theories, such as integer linear arithmetic, it is possible to decide if $A_\sim$ is finite. We also consider the problem of monadic decomposition in the presence of uninterpreted function symbols from a signature Σ. In this case we are no longer dealing with a single background structure, but with the class of all of its possible Σ-expansions. We show that monadic decomposition is decidable for *EUF*, where equality is the only interpreted symbol, in Theorem 5.5.

Applications. Monadic decomposition is a general simplification technique with many potential applications. We discuss briefly several concrete examples in the next section, and focus on more details in § 7. Monadic decomposition can be used in many different combinations of theories and in many different contexts where it is useful to simplify formulas by eliminating variable dependencies, such as *program analysis*, *theorem proving*, and *compiler optimization* and where solvers such as Z3 [19, 20], CVC4 [4], and Yices [24**?**], are actively being used today. New areas where the use of SMT solvers is being investigated are *network protocols* and *query optimization*.

The rest of the paper is organized as follows. § 2 describes some motivating scenarios. In § 3 and § 4 the problem is defined formally, we prove the main decomposition Theorem 4.6 and correctness of the main algorithm ***mondec*** in Theorem 4.12. We show the general case to be undecidable as a corollary of [48, Proposition 2]. In § 5 we show some decidable cases, in particular, Theorem 5.5 shows that monadic decomposition is decidable in EUF. § 6 provides some evaluation using a micro-benchmark. § 7 describes in more detail concrete applications where we first encountered the need for monadic decomposition. § 8 is related work. § 9 concludes.

2 MOTIVATION

String analysis. In the context of analysis of string manipulating code, characters are often treated as bytes or 8-bit bit-vectors[3] as in extended ASCII, or 16-bit bit-vectors[4] as in UTF16 encoded strings that is the de facto standard in all programming environments. A popular formalism that helps to scale the analysis of string manipulating code to such large alphabets is the theory of *symbolic finite automata* (SFAs) and *symbolic finite transducers* (SFTs) [39, 73]. Instead of concrete characters, SFAs use character predicates.

In encoders it is common that characters are composed and decomposed using arithmetic operations. Certain composition operations lead to transitions that read a sequence of characters at

[3]Numbers between 0 and $\mathtt{FF}_{16}$.
[4]Numbers between 0 and $\mathtt{FFFF}_{16}$.

once [17], such as this one from state p to state q, where the sequence (x, y) is read in one transition:

$$(p) \xrightarrow{\lambda(x,y).127<(64(x \bmod 32)+(y \bmod 64))\leq 2047 \wedge x=192+(x \bmod 32) \wedge y=138+(y \bmod 64)} (q)$$

and whose guard checks that two consecutive bytes x and y form a valid two-byte UTF8 encoding of a Unicode character. Such formulas hinder SFA based analysis because of the lookahead. In this case, monadic decomposition of the guard produces the equivalent predicate $\lambda(x, y).194 \leq x \leq 223 \wedge 128 \leq y \leq 191$. Therefore, the above transition can be split into two transitions, by introducing an intermediate state r, where each transition reads a single character at a time:

$$(p) \xrightarrow{\lambda x.194\leq x\leq 223} (r) \xrightarrow{\lambda y.128\leq y\leq 191} (q)$$

Monadic decomposition may in general introduce nondeterminism into the automaton but this can be resolved through SFA determinization and does not affect the intended semantics in this context. We revisit some analysis scenarios in more detail in Section 7.

Compiler optimizations. Monadic decomposition can be used to simplify expressions and thus enable new (or enhance existing) automatic compiler optimization techniques [1] by localizing or removing variable dependencies. It may also be used to enable parallelization of automata based string processing procedures in typical scenarios by eliminating lookahead [74]. String processing often involves pipelines of composed procedures such as Base64 decoding, UTF8 decoding, and Html decoding, that are expressed as *symbolic transducers*, and where several characters need to be read and combined first into bigger units before their processing can continue in the next step of the pipeline. Such procedures often introduce multi-character lookahead that hinders parallelization. Parallelization of finite state machines is based on encoding the transition function of the state machine (without lookahead) as matrix multiplication [46, 52], the latter is parallelizable due to associativity.

Theorem proving. In the context of automated first-order resolution based theorem proving modulo theories, *Skolemization* may benefit from monadic decomposition by enabling simpler Skolem functions [44]. The use of SMT solvers in this context comes into play when the classical resolution technique is extended to work modulo background theories [42, 43]. In theory based reasoning a formula is typically first Skolemized. For example, consider integer linear arithmetic and the sentence

$$\forall x \exists y (0 \leq x \leq 1 \Rightarrow (0 \leq y \wedge x + y \leq 1)).$$

The main objective of *antiprenexing* [3] is to minimize the arity of the introduced Skolem functions. In this case antiprenexing and subsequent Skolemization produces the formula

$$\forall x (0 \leq x \leq 1 \Rightarrow (0 \leq f(x) \wedge x + f(y) \leq 1)).$$

However, introduction of the unary Skolem function f can be avoided in this context because the formula $0 \leq x \leq 1 \Rightarrow (0 \leq y \wedge x + y \leq 1)$ is monadic and the initial formula is, after monadic decomposition, equivalent to

$$\forall x \exists y (0 \leq x \leq 1 \Rightarrow ((x \simeq 0 \wedge 0 \leq y \leq 1) \vee (x \simeq 1 \wedge y \simeq 0)))$$

In this case antiprenexing removes y from the scope of x and the subsequent Skolemization step will replace y by an uninterpreted constant.

There is also a clear connection to *quantifier elimination* in this case. When the quantifier free body of a formula in prenex normal form is monadic (when generalized to multiple variables), then monadic decomposition followed by antiprenexing becomes effectively a quantifier elimination procedure.

Program analysis. Monadic decomposition can be used to break down dependencies between program variables and thus simplify various symbolic techniques that are used in the context of modern program analysis [56]. The use of an SMT solver as a black box is particularly well suited in this context because it allows seamless combination of different theories for different data types.

3 MONADIC RELATIONS

We assume $\mathfrak{U}$ and Ψ as described before. The Boolean type is BOOL with truth values $\{\top, \bot\}$. We use λ-expressions to define anonymous functions and relations, given $\varphi(\bar{x}) \in \Psi$ where all the free variables of $\varphi(\bar{x})$ are among $\bar{x} = (x_1, \ldots, x_n)$, we write $\lambda\bar{x}.\varphi(\bar{x})$ or simply φ, when the arity n and the $\bar{x}$ are clear from the context. We let $[\![\varphi]\!]$ denote the n-ary relation defined by φ.

We recall the following definitions from [48] with small notational adjustments. Consider a sequence of distinct variables $\bar{x} = (x_1, \ldots, x_n)$ and a partition P of $X = \{x_i\}_{i=1}^n$.[5] P is *finest* if $P = \{\{x\}\}_{x \in X}$. A formula $\varphi(\bar{x})$ *respects the partition* P if $\varphi(\bar{x})$ is a Boolean combination of formulas each having its free variables in a single block in P. Two variables x_i and x_j are *independent in* $\varphi(\bar{x})$ if there exists a partition P such that x_i and x_j belong to two different blocks of P and φ respects P. The *variable independence problem* is to decide if x_i and x_j are independent in $\varphi(\bar{x})$. The *variable partition problem* is to decide if a formula φ respects a given partition P of its free variables. Both problems are effectively equivalent [48, Lemma 1].

In order to decide if all variables are pairwise independent we can decide if φ respects the finest variable partition. In order to decide if two variables x_i and x_j are independent in $\varphi(\bar{x})$ it is enough to partition X into two blocks, one containing x_i and the other containing x_j, because if φ respects P then it trivially also respects any variable partition P' that is *coarser* than P.[6] We can view blocks $B \in P$ of variables as single block-variables x_B, and reduce the variable independence problem to the case $\bar{x} = (x_B, x_C)$ by making individual variables in a block B be projections from x_B, and decide if φ respects the variable partition $\{\{x_B\}, \{x_C\}\}$. We can therefore focus on *finest* partitions without loss of generality.

Let R be an n-ary relation for some $n \geq 2$. R is *Cartesian* if R is the direct product $\prod_{i=1}^n U_i$ of some sets U_i. R is *monadic* if R equals to a finite union $\cup_{i=1}^k R_i$ of Cartesian relations R_i, called a *monadic decomposition of* R. The *(monadic) width* of R is the smallest such k if R is monadic, $k = \omega$ otherwise. Note that R has width 1 iff it is Cartesian. We say that a formula $\varphi(\bar{x})$ or the predicate $\lambda\bar{x}.\varphi(\bar{x})$ is *monadic* iff $[\![\varphi]\!]$ is monadic. The following proposition relates monadic predicates to variable partitioning in Ψ.

PROPOSITION 3.1. *A formula $\varphi \in \Psi$ is monadic iff there exists an equivalent formula $\psi \in \Psi$ that respects the finest partition of its free variables.*

PROOF. The direction $\Leftarrow$ follows by taking the DNF of ψ and defining the Cartesian components from its disjuncts. For direction $\Rightarrow$ use Theorem 4.6. □

Example 3.2. Let $\varphi(x, y)$ be the formula $(x + (y \mod 2)) > 5$, where $\mathcal{U}$ is integers. Then $R = [\![\varphi]\!]$ is the corresponding binary relation over integers. R is not Cartesian but it is monadic and has monadic width 2 because a monadic decomposition of R is

$$([\![\lambda x.x > 5]\!] \times [\![\lambda y.\top]\!]) \cup ([\![\lambda x.x > 4]\!] \times [\![\lambda y.odd(y)]\!]).$$

Another possible monadic decomposition of R is

$$([\![\lambda x.x > 5]\!] \times [\![\lambda y.even(y)]\!]) \cup ([\![\lambda x.x > 4]\!] \times [\![\lambda y.odd(y)]\!]).$$

[5] All the blocks in P are nonempty pairwise disjoint sets, and their union equals X.
[6] P' is *coarser than* P if each block of P is a subset of some block of P'.

So monadic decompositions are clearly not unique. □

Example 3.3. Let $\varphi(x_1, x_2, y)$ be the formula $(x_1 + x_2 + (y \mod 2)) > 5$, where the universe $\mathcal{U}$ is integers. Then φ is not monadic, but similarly to Example 3.2, the formula $\varphi'(x, y) = \varphi(\pi_1(x), \pi_2(x), y)$ is monadic because $[\![\varphi']\!] = ([\![\lambda x.\pi_1(x) + \pi_2(x) > 5]\!] \times [\![\lambda y.\top]\!]) \cup ([\![\lambda x.\pi_1(x) + \pi_2(x) > 4]\!] \times [\![\lambda y.odd(y)]\!])$. Equivalently, there is a formula equivalent to $\varphi'(x_1, x_2, y)$ that respects the partition $\{\{x_1, x_2\}, \{y\}\}$. □

As noted in [48], a naive algorithm to finding a formula that respects a partition P and is equivalent to φ is to exhaustively enumerate all possible formulas $\psi(\bar{x})$ that respect P and for each one decide if $\mathfrak{U} \models \forall \bar{x}(\varphi(\bar{x}) \Leftrightarrow \psi(\bar{x}))$. After a finite number of steps such a formula ψ is eventually found if it exists. This statement is also true here, by using Proposition 3.1 and that $\neg(\varphi(\bar{x}) \Leftrightarrow \psi(\bar{x})) \in \Psi$, so we can decide validity of $\forall \bar{x}(\varphi(\bar{x}) \Leftrightarrow \psi(\bar{x}))$ by checking unsatisfiability of $\neg(\varphi(\bar{x}) \Leftrightarrow \psi(\bar{x}))$.

A *unary* formula is a formula with at most one free variable. A *monadic normal form* or *MNF* of a formula is an equivalent Boolean combination of unary formulas. Observe that the difference between being monadic and being in monadic normal form, is that the first notion is semantic (depends on $\mathfrak{U}$) while the second is syntactic (independent of $\mathfrak{U}$). An important point about MNF, when it exists, is that it does not have to be in disjunctive normal form (DNF) as the Cartesian based definition would suggest. A formula φ being in MNF is equivalent to saying that φ respects the finest partition of its free variables that is equivalent to saying that all free variables of φ are pairwise independent.

An open problem, that we are not going to discuss here, but that is relevant in many practical contexts is how to discover the "best" partition of the free variables, so that dependent free variables fall into the same block, as for example $\{x_1, x_2\}$ in Example 3.3.

4 MONADIC DECOMPOSITION

We are interested in the following two problems: 1) Deciding if a predicate φ is monadic; 2) Given a monadic predicate φ, effectively constructing its MNF. We restrict our attention to *binary* predicates first.

4.1 Deciding if a predicate is monadic

Consider any formula $P(x)$ in Ψ denoting a subset of some infinite domain, integers say, and let $\phi_P(x, y)$ be the formula $P(x) \wedge x \simeq y$. Then ϕ_P is monadic iff $[\![P]\!]$ is finite. Deciding finiteness is an undecidable problem in general by using Rice's Theorem [64]. Can we use a similar argument here by finding a structure $\mathfrak{U}$ that allows us to reduce the halting problem to deciding finiteness? The following proposition is a corollary of [48, Proposition 2(b)] and Proposition 3.1 but we include the direct argument here to make the connection to deciding finiteness more transparent.

Proposition 4.1. *There exists $\mathfrak{U}$ such that monadic decomposition is undecidable in $\mathfrak{U}$.*

Proof. Consider the theory of partial computations of Turing Machines [66]. The atomic formulas have the form $P(\ulcorner M \urcorner, \ulcorner w \urcorner, \ulcorner t \urcorner)$ encoding statements: t is a valid (partial) trace of the Turing machine M with input word w. The theory is decidable and satisfies the requirements for $\mathfrak{U}$ and Ψ. For fixed M and w define the predicate $P_{M,w}$ as $\lambda x.P(\ulcorner M \urcorner, \ulcorner w \urcorner, x)$. Then $[\![P_{M,w}]\!]$ is finite iff M halts on w. Thus, $\phi_{P_{M,w}}$ is monadic iff M halts on w. □

Use of formal equality $\simeq$ plays a key role in the above constructions. It is unclear what the implications are if $\simeq$ is disallowed.

4.2 Decomposition procedure

In the following, we provide a brute force semidecision procedure for monadic decomposition. While the procedure is complete for monadic predicates, in the nonmonadic case it will not terminate. The input is a binary predicate $\varphi(x, y) \in \Psi$.

Let $R = [\![\varphi]\!] \subseteq A \times B$, where we assume that $R \neq \emptyset$ and

$$A = \{a \mid \exists b\, R(a, b)\}, \quad B = \{b \mid \exists a\, R(a, b)\}.$$

Define the relations:

$$\begin{aligned} x \sim x' &\stackrel{\text{def}}{=} \forall y\, y'((\varphi(x, y) \wedge \varphi(x', y')) \Rightarrow (\varphi(x', y) \wedge \varphi(x, y'))) \\ y \backsim y' &\stackrel{\text{def}}{=} \forall x\, x'((\varphi(x, y) \wedge \varphi(x', y')) \Rightarrow (\varphi(x', y) \wedge \varphi(x, y'))) \end{aligned}$$

Intuitively, $x \sim x'$ means that x and x' are indistinguishable with respect to their images by R, i.e., $R(x) = R(x')$, and $y \backsim y'$ means that y and y' are indistinguishable with respect to their preimages by R, i.e., $R^{-1}(y) = R^{-1}(y')$ where $R^{-1}(y) \stackrel{\text{def}}{=} \{x \mid R(x, y)\}$. The following lemma makes this intuition precise.

Lemma 4.2. *Let R and A be given as above. 1) For all $a, a' \in A$: $a \sim a'$ if and only if $R(a) = R(a')$. 2) The relation $\sim$ is an equivalence relation over A.*

Proof. *Proof of 1*: Let $a, a' \in A$.

- $\Rightarrow$: Assume $a \sim a'$. We show that $R(a) \subseteq R(a')$. Let $(a, b) \in R$. We show that $(a', b) \in R$. There is some b' such that $(a', b') \in R$. So, by definition of $\sim$, $(a', b), (a, b') \in R$.
- $\Leftarrow$: Assume $R(a) = R(a')$. Let $(a, b), (a', b') \in R$, i.e., $b \in R(a)$ and $b' \in R(a')$. So, by $R(a) = R(a')$, we have $b \in R(a')$ and $b' \in R(a)$. It follows that $(a, b'), (a', b) \in R$.

Proof of 2: Immediate by using 1. □

By symmetry, Lemma 4.2 holds also for $\backsim$ and B. We let $[a]_\sim$ (resp. $[b]_\backsim$) denote the equivalence class $\{e \in A \mid e \sim a\}$ (resp. $\{e \in B \mid e \backsim b\}$). The following is a technical lemma used for proving Theorem 4.6. Similar statement was first proved in [48, Lemma 4 in Theorem 3]. The proof here also shows a relationship between the number of $\sim$-equivalence classes and the monadic width that we will make use of below.

Lemma 4.3. *R is monadic $\Leftrightarrow$ the number of $\sim$-equivalence classes is finite. Moreover, if R is monadic then the number of $\sim$-equivalence classes is at most $2^k - 1$ where k is the monadic width of R.*

Proof. $\Rightarrow$: Assume R has a monadic decomposition $\{A_i \times B_i\}_{i<n}$. Let $\tilde{A}_i = \bigcup_{a \in A_i} [a]_\sim$. We show first that $\{\tilde{A}_i \times B_i\}_{i<n}$ is also a monadic decomposition of R. Suppose $(a, b) \in \tilde{A}_i \times B_i$. So there is $a_i \in A_i$ such that $a \sim a_i$. Since $(a_i, b) \in A_i \times B_i$ it follows that $(a_i, b) \in R$, so $b \in R(a_i)$. But $R(a_i) = R(a)$ by Lemma 4.2 because $a_i \sim a$, so $b \in R(a)$, i.e., $(a, b) \in R$. The direction $R \subseteq \bigcup_{i<n} \tilde{A}_i \times B_i$ is immediate because $R \subseteq \bigcup_{i<n} A_i \times B_i$ and $A_i \subseteq \tilde{A}_i$.

Next, we normalize $\{\tilde{A}_i \times B_i\}_{i<n}$ into a form $\{A'_i \times B'_i\}_{i<m}$ where each A'_i ends up being exactly one $\sim$-equivalence class of A. For all $I \subseteq \{i \mid 0 \leq i < n\}$ let M_I be the *minterm* $(\bigcap_{i \in I} \tilde{A}_i) \setminus (\bigcup_{j \notin I} \tilde{A}_j)$. By using standard Boolean laws, each $\tilde{A}_i$ is a finite union of disjoint nonempty minterms. Apply the following equivalence preserving transformations to the monadic decomposition $\{\tilde{A}_i \times B_i\}_{i<n}$ until no more transformations can be made:

- replace $(M_I \cup M) \times B_i$ by $(M_I \times B_i) \cup (M \times B_i)$,
- replace $(M_I \times B_i) \cup (M_I \times B_j)$ by $M_I \times (B_i \cup B_j)$.

Let the resulting decomposition be $\{A'_i \times B'_i\}_{i<m}$, where, for all $a \in A$ and $b \in B$, we have $(a,b) \in R$ iff there exists exactly one i such that $(a,b) \in A'_i \times B'_i$. In other words, for all $a \in A$, $R(a)$ is the set B'_i such that $a \in A'_i$. It follows that $a \sim a'$ for all $a, a' \in A'_i$.

Thus, the number of $\sim$-equivalence classes is bounded by $2^k - 1$ where k is the monadic width of R, because the number m of different (nonempty) minterms M_I is, due to the powerset construction, at most $2^k - 1$.

$\Leftarrow$: Assume that the number of $\sim$-equivalence classes is finite. Let $A = \bigcup_{i=0}^{n-1} A_i$ where $A_i = [a_i]_\sim$. Let $B_i = R(a_i)$ for $0 \leq i < n$. Thus if $(a,b) \in A_i \times B_i$ then $a \sim a_i$ and $b \in R(a_i)$, i.e., $R(a) = R(a_i)$ and $b \in R(a_i)$. So $b \in R(a)$, i.e., $(a,b) \in R$. Conversely, if $(a,b) \in R$ then $b \in R(a)$. But $R(a) = R(a_i) = B_i$, for some $i < n$, where $a \in A_i$ and $b \in B_i$. Thus, $\{A_i \times B_i\}_{i<n}$ is a monadic decomposition of R. □

Next, we provide an iterative procedure to compute a *witness set* W_A that covers $A_\sim$. We use the negated form of $\sim$:

$$x \nsim x' \quad \Leftrightarrow \quad \exists y\, y'(\varphi(x,y) \wedge \varphi(x',y') \wedge (\neg\varphi(x',y) \vee \neg\varphi(x,y')))$$

So, for all $a, a' \in A$, $a \nsim a'$ means that a and a' must participate in distinct Cartesian components of a monadic decomposition of φ, i.e., if $\{R_i\}_{i<k}$ is a monadic decomposition of R, then there exist $b, b' \in B$ and $i \neq j$ such that $(a,b) \in R_i \setminus R_j$ and $(a',b') \in R_j \setminus R_i$.

Computation of W_A : Let $(a_0, b_0) \in [\![\varphi]\!]$ and let $W_A = \{a_0\}$. Repeat:

(1) Let $\psi(x)$ be theormula $\bigwedge_{a \in W_A} x \nsim a$.
(2) If there exists a such that $\psi(a)$ holds then $W_A := W_A \cup \{a\}$ else terminate.

Recall that Ψ is closed under Boolean connectives. This means that the formula $\psi(x)$ with all the existentially quantified variables (which all occur positively) replaced by fresh free variables $\bar{z}$ is a formula $\psi'(x,\bar{z})$ that belongs to Ψ and therefore, the satisfiability checking of ψ', and thus ψ, as well as generating the witness a is decidable. When ψ becomes unsatisfiable then any further element from A must be $\sim$-equivalent to one of the elements already in W_A, while all elements in W_A belong to distinct $\sim$-equivalence classes. Therefore, if φ is monadic then the process terminates by Lemma 4.3, and upon termination W_A is a finite collection of witnesses that divides A into a set $A_\sim$ of $\sim$-equivalence classes $[a]_\sim$ for $a \in W_A$. For example, if φ is Cartesian then ψ is unsatisfiable initially, because then $A_\sim = \{[a_0]_\sim\}$.

Computation of W_B is analogous to computation of W_A. Observe that $\max(|W_B|, |W_A|) < 2^k$ where k is the monadic width of φ, which follows from Lemma 4.3. Also $k \leq \min(|W_B|, |W_A|)$. So the size difference between $|W_B|$ and $|W_A|$ is at most exponential.

Example 4.4. Consider the relation

$$R^k(x,y) \stackrel{\text{def}}{=} y{>}0 \wedge y\&(y{-}1){=}0 \wedge x\&(y \bmod (2^k - 1)){\neq}0$$

defined over 32-bit bit-vectors where & is bit-wise-AND. The width of R^k is k. R^3 is illustrated in Figure 1. Figure 1(a) illustrates a geometrical view of R^3 where (x,y) is marked iff $R^3(x,y)$ holds. We have

$$A_\sim = \{[a]_\sim \mid 1 \leq a \leq 7\} \text{ where } [a]_\sim = \{n \mid n_{\langle 2,0\rangle} = a\}$$

and

$$B_\backsim = \{[2^0]_\backsim, [2^1]_\backsim, [2^2]_\backsim\} \text{ where } [2^m]_\backsim = \{2^n \mid n \bmod 3 = m\}.$$

If $R(m)$ and $R(n)$ are identical then $m \sim n$, e.g., $1 \sim 9$. If $R^{-1}(m)$ and $R^{-1}(n)$ are identical then $m \backsim n$, e.g., $2^2 \backsim 2^5$.

Figure 1(b) illustrates the equivalence classes as nonempty regions of a Venn Diagram view of R^3. $R^3 = \bigcup_{i=1}^{3} A_i \times B_i$. □

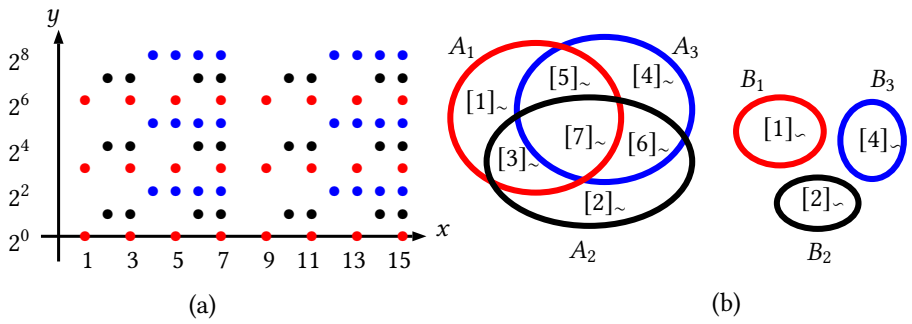


Fig. 1. a) Geometrical view of R^3; b) Venn diagram view of R^3; R^k is defined in Example 4.4.

Lemma 4.5. *If R is monadic then, for all $[a]_\sim \in A_\sim$ and $[b]_\smile \in B_\smile$, we can effectively construct $\alpha_a, \beta_a \in \Psi$ such that $[\![\alpha_a]\!] = [a]_\sim$ and $[\![\beta_b]\!] = [b]_\smile$.*

Proof. By using Lemma 4.3 let W_A be constructed as above, so $A_\sim = \{[a]_\sim \mid a \in W_A\}$. Construct a finite W_B similarly such that $B_\smile = \{[b]_\smile \mid b \in W_B\}$. Let

$$(\text{for } b \in W_B) \quad \beta_b(y) \stackrel{\text{def}}{=} \Big(\bigwedge_{a \in W_A \cap R^{-1}(b)} \varphi(a,y)\Big) \wedge \Big(\bigwedge_{a \in W_A \setminus R^{-1}(b)} \neg\varphi(a,y)\Big)$$

$$(\text{for } a \in W_A) \quad \alpha_a(x) \stackrel{\text{def}}{=} \Big(\bigwedge_{b \in W_B \cap R(a)} \varphi(x,b)\Big) \wedge \Big(\bigwedge_{b \in W_B \setminus R(a)} \neg\varphi(x,b)\Big)$$

Observe that $[\![\alpha_a]\!] = [\![\alpha_{a'}]\!]$ for all $a' \sim a$. Similarly for β_b. Fix $a \in W_A$ and consider the definition of α_a. We prove that $[\![\alpha_a]\!] = [a]_\sim$. Proof of $[\![\beta_b]\!] = [b]_\smile$ is similar.

Suppose $W_B \cap R(a) = \{b_i\}_{i \in I}$ and $W_B \setminus R(a) = \{b_j\}_{j \in J}$ where $W_B = \{b_i\}_{i \in I \cup J}$. For any $i \in I$ we have that $[a]_\sim \subseteq R^{-1}(b_i)$ because $a \in R^{-1}(b_i)$. For any $j \in J$ we have that $[a]_\sim \cap R^{-1}(b_j) = \emptyset$ because $a \notin R^{-1}(b_j)$. Then

$$[a]_\sim \subseteq \Big(\bigcap_{i \in I} R^{-1}(b_i)\Big) \setminus \Big(\bigcup_{j \in J} R^{-1}(b_j)\Big) = [\![\Big(\bigwedge_{i \in I} \varphi(x,b_i)\Big) \wedge \neg\Big(\bigvee_{j \in J} \varphi(x,b_j)\Big)]\!] = [\![\alpha_a]\!]$$

For the direction $[\![\alpha_a]\!] \subseteq [a]_\sim$ take $a' \in [\![\alpha_a]\!]$. Suppose, by way of contradiction that, $a \nsim a'$ and thus $R(a') \neq R(a)$. Then there exists $b \in W_B \setminus R(a)$ such that $a' \in R^{-1}(b)$. But then, by definition of α_a, $\neg\varphi(a',b)$ holds, contradicting that that $a' \in R^{-1}(b)$ ($\varphi(a',b)$ holds). □

Lemma 4.5 is essentially a quantifier elimination property that allows us to eliminate the $\forall$ quantifier from the definition of $\lambda x.x \sim a$ (resp. $\lambda y.y \smile b$) by stating that it is enough to consider the elements in W_B (resp. W_A). We can now prove the following result. It gives us a brute force method for monadic decomposition.

Theorem 4.6. *If $\varphi(x,y)$ is monadic then*
a) $\varphi(x,y)$ is equivalent to $\lambda(x,y).\ \bigvee_{a \in W_A}(\alpha_a(x) \wedge \varphi(a,y))$.
b) $\varphi(x,y)$ is equivalent to $\lambda(x,y).\ \bigvee_{b \in W_B}(\beta_b(y) \wedge \varphi(x,b))$.
c) $\varphi(x,y)$ is equivalent to $\lambda(x,y).\ \bigvee_{a \in W_A, b \in W_B, (a,b) \in [\![\varphi]\!]}(\alpha_a(x) \wedge \beta_b(y))$.

Proof. We prove (a). The other cases are similar. By Lemma 4.5 we have $[\![\alpha_a]\!] = [a]_\sim$. By construction of W_A we have that, for all $a \in W_A$ we have $[a]_\sim \times R(a) \subseteq [\![\varphi]\!]$ where $[a]_\sim \times R(a) = [\![\lambda(x,y).\alpha_a(x) \wedge \varphi(a,y)]\!]$. In the other direction, if $(a,b) \in [\![\varphi]\!]$ then $a \in [\![\alpha_a]\!]$ and $b \in R(a)$. In other words, $(a,b) \in [\![\lambda(x,y).\alpha_a(x) \wedge \varphi(a,y)]\!]$. □

Theorem 4.6 does not guarantee smallest monadic width. Example 4.7 shows that the monadic width may be strictly smaller than $\min(|W_B|, |W_A|)$.

Example 4.7. Take

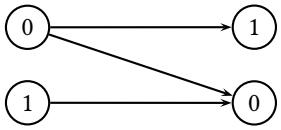
$$R = \{(1,1),(2,2),(3,3),(4,4),(5,1),(5,2),(3,5),(4,5)\}$$

where $A = B = \{1,2,3,4,5\}$. Then $|W_A| = 5$ and $|W_B| = 5$ but R has width 4:

$$R = (\{1,5\} \times \{1\}) \cup (\{2,5\} \times \{2\}) \cup (\{3\} \times \{3,5\}) \cup (\{4\} \times \{4,5\}).$$

□

Example 4.8. Let $\phi(x,y) := (0 \le x \le 1 \wedge 0 \le y \le 1 \wedge x + y < 2)$. The example illustrates a case where ϕ is satisfied by a finite model of the form:



We get the following predicates by using Lemma 4.5 and applying some simplifications.

$$\alpha_0(x) \stackrel{\text{def}}{=} x \simeq 0, \quad \alpha_1(x) \stackrel{\text{def}}{=} x \simeq 1, \quad \beta_0(y) \stackrel{\text{def}}{=} y \simeq 0, \quad \beta_1(y) \stackrel{\text{def}}{=} y \simeq 1$$

Monadic decomposition of ϕ reconstructs the formula

$$\alpha_0(x) \wedge \beta_0(y) \ \vee \ \alpha_0(x) \wedge \beta_1(y) \ \vee \ \alpha_1(x) \wedge \beta_0(y)$$

by using Theorem 4.6(c). Case $\alpha_1(x) \wedge \beta_1(y)$ is not included because $\phi(1,1)$ is false. □

4.3 Algorithm *mondec*

The algorithm suggested by Theorem 4.6 has some disadvantages. It forces the decomposition to be in disjunctive normal form (DNF) with respect to the unary sub-formulas, while an exponentially more succinct MNF may exist. Moreover, it forces full exploration of both the set W_A and the set W_B. Furthermore, it does not easily generalize to non-binary relations. Here we describe an algorithm ***mondec*** that addresses all these concerns.

First, we lift the definitions of $\sim$ (resp. $\backsim$) to all $x, x', y, y' \in \mathcal{U}$:

$$x \nsim x' \stackrel{\text{def}}{=} \exists z(\neg(\varphi(x,z) \Leftrightarrow \varphi(x',z))), \quad y \nsim y' \stackrel{\text{def}}{=} \exists z(\neg(\varphi(z,y) \Leftrightarrow \varphi(z,y'))).$$

Equivalently

$$x \sim x' \stackrel{\text{def}}{=} R(x) = R(x'), \quad y \backsim y' \stackrel{\text{def}}{=} R^{-1}(y) = R^{-1}(y').$$

In other words, we let $a_1 \sim a_2$ when $R(a_1) = R(a_2) = \emptyset$, and, symmetrically, $b_1 \backsim b_2$ when $R^{-1}(b_1) = R^{-1}(b_2) = \emptyset$. This is consistent with the earlier definition (due to Lemma 4.2) when $\sim$ was applied to elements in A and $\backsim$ was applied to elements in B. Here it is easier to work with $\mathcal{U}$ because the equivalence classes are *identical* for φ and $\neg\varphi$.

Example 4.9. Consider the $\sim$-equivalence classes in Figure 1. Then $[0]_\sim$ is the complement of $A_1 \cup A_2 \cup A_3$. □

Instead of creating what amounts to a DNF, we use case analysis on $\varphi(a,y) \wedge \varphi(x,b)$ for all $([a]_{\sim}, [b]_{\smile}) \in \mathcal{U}_{\sim} \times \mathcal{U}_{\smile}$. The output may be any formula in MNF, not necessarily in DNF. We avoid full exploration of $\mathcal{U}_{\sim}$ and $\mathcal{U}_{\smile}$ by maintaining a shared datastructure where the equivalence classes are represented implicitly.

The algorithm ***mondec*** is described in Figure 2. It takes as input a monadic formula φ and creates an MNF of φ in terms of nested if-then-else formulas that is either $\top$ or $\bot$ or has the form $\textsc{if}(\psi, \phi_t, \phi_f)$ that represents the equivalent formula $((\psi \wedge \phi_t) \vee (\neg\psi \wedge \phi_f))$, and where ϕ_t and ϕ_f are again nested if-then-else formulas.

$$
\begin{aligned}
\boldsymbol{mondec}_{x,y}(\varphi) &\stackrel{\text{def}}{=} \boldsymbol{dec}(\top, \top), \quad \text{where} \\
\boldsymbol{dec}(\nu, \pi) &\stackrel{\text{def}}{=} \begin{cases} \bot, & \text{if not } \boldsymbol{sat}(\pi \wedge \varphi); \\ \top, & \text{else if not } \boldsymbol{sat}(\pi \wedge \neg\varphi); \\ \textsc{if}(\boldsymbol{\pi}^a_b, \boldsymbol{dec}(\nu \wedge \boldsymbol{\nu}^a_b, \pi \wedge \boldsymbol{\pi}^a_b), \boldsymbol{dec}(\nu \wedge \boldsymbol{\nu}^a_b, \pi \wedge \neg\boldsymbol{\pi}^a_b))), \\ \qquad \text{else, where } (a,b) \text{ are such that } x = a, y = b \models_{\mathfrak{U}} \nu. \end{cases} \\
\boldsymbol{\nu}^a_b &\stackrel{\text{def}}{=} a \nsim x \vee y \nsim b, \\
\boldsymbol{\pi}^a_b &\stackrel{\text{def}}{=} \varphi(a,y) \wedge \varphi(x,b).
\end{aligned}
$$

Fig. 2. Algorithm ***mondec***.

Each "node" ψ in $\textsc{if}(\psi, \phi_t, \phi_f)$ has the form $\varphi(a,y) \wedge \varphi(x,b)$ for some $a, b \in \mathcal{U}$. The if-then-else formula is created recursively by the procedure $\boldsymbol{dec}(\nu, \pi)$ where ν is the condition that removes redundant solutions from $\mathcal{U}_{\sim} \times \mathcal{U}_{\smile}$ while π is the "path condition" or the condition from the root of the if-then-else formula to the current node, that determines the termination condition of the branch. If π or $\pi \wedge \varphi$ is unsatisfiable, then π implies $\neg\varphi$ and the leaf is $\bot$. Else, if $\pi \wedge \neg\varphi$ is unsatisfiable, this means that π implies φ and the leaf is $\top$. If both $\pi \wedge \varphi$ and $\pi \wedge \neg\varphi$ are satisfiable then there exists a new pair (a,b) that can be used to build a new node and the decomposition continues locally. The check that (a,b) satisfies ν guarantees that the pair $([a]_{\sim}, [b]_{\smile})$ has not already been used along this path. Observe that neither W_A nor W_B are explicitly computed. In the case of decomposition involving more than 2 variables, say y is a pair (y_1, y_2), ***mondec*** can be invoked recursively on node subformulas $\varphi(a,y)$, i.e., the definition of $\boldsymbol{\pi}^a_b$ becomes $\boldsymbol{mondec}_{y_1,y_2}(\varphi(a,y)) \wedge \varphi(x,b)$.

Upon termination, the disjunction of all branches that end with $\top$ amounts to an MNF of φ, or equivalently, the disjunction of all branches that end with $\bot$ amounts to an MNF of $\neg\varphi$.

Example 4.10. We illustrate execution of ***mondec*** with the example formula φ taken from Example 3.2, $\varphi = (x + (y \bmod 2)) > 5$. When choosing a witness (a,b) we use the additional heuristic that $\{x = a, y = b\} \models \nu \wedge \pi \wedge \varphi$. In the initial call to $\boldsymbol{dec}(\top, \top)$ both φ and $\neg\varphi$ are satisfiable. Choose $(a,b) = (8,0)$. Then $\boldsymbol{\pi}^8_0$ equals $8 + (y \bmod 2) > 5 \wedge x + (0 \bmod 2) > 5$ that simplifies to $x > 5$. The following if-then-else formula is created

$$\boldsymbol{dec}(\top, \top) = \textsc{if}(x > 5, \boldsymbol{dec}(\boldsymbol{\nu}^8_0, x > 5), \boldsymbol{dec}(\boldsymbol{\nu}^8_0, x \leq 5))$$

In the call $\boldsymbol{dec}(\boldsymbol{\nu}^8_0, x > 5)$ the condition $x > 5 \wedge \neg((x + (y \bmod 2)) > 5)$ is unsatisfiable which means that $\boldsymbol{dec}(\boldsymbol{\nu}^8_0, x > 5) = \top$. In the call $\boldsymbol{dec}(\boldsymbol{\nu}^8_0, x \leq 5)$, both formulas $x \leq 5 \wedge \varphi$ and $x \leq 5 \wedge \neg\varphi$ are satisfiable. Choose the next witness $(a,b) = (5,1)$. Then $\boldsymbol{\pi}^5_1$ equals $5 + (y \bmod 2) > 5 \wedge x + (1 \bmod 2) > 5$ that simplifies to $(y \bmod 2) > 0 \wedge x > 4$. So

$$\boldsymbol{dec}(\boldsymbol{\nu}^8_0, x \leq 5) = \textsc{if}(\boldsymbol{\pi}^5_1, \boldsymbol{dec}(\boldsymbol{\nu}^8_0 \wedge \boldsymbol{\nu}^5_1, x \leq 5 \wedge \boldsymbol{\pi}^5_1), \boldsymbol{dec}(\boldsymbol{\nu}^8_0 \wedge \boldsymbol{\nu}^5_1, x \leq 5 \wedge \neg\boldsymbol{\pi}^5_1))$$

In the call to $\boldsymbol{dec}(\boldsymbol{\nu}_0^8 \wedge \boldsymbol{\nu}_1^5, x \leq 5 \wedge \boldsymbol{\pi}_1^5)$ the formula $x \leq 5 \wedge \boldsymbol{\pi}_1^5 \wedge \neg\varphi$ is unsatisfiable, so

$$\boldsymbol{dec}(\boldsymbol{\nu}_0^8 \wedge \boldsymbol{\nu}_1^5, x \leq 5 \wedge \boldsymbol{\pi}_1^5) = \top.$$

In the call to $\boldsymbol{dec}(\boldsymbol{\nu}_0^8 \wedge \boldsymbol{\nu}_1^5, x \leq 5 \wedge \neg\boldsymbol{\pi}_1^5)$ the formula $x \leq 5 \wedge \neg\boldsymbol{\pi}_1^5 \wedge \varphi$ is unsatisfiable, so

$$\boldsymbol{dec}(\boldsymbol{\nu}_0^8 \wedge \boldsymbol{\nu}_1^5, x \leq 5 \wedge \neg\boldsymbol{\pi}_1^5) = \bot.$$

Thus, the final MNF of φ equals $\text{IF}(x > 5, \top, \text{IF}((y \bmod 2) > 0 \wedge x > 4, \top, \bot))$ which is equivalent to the disjunction $x > 5 \vee (x \leq 5 \wedge (y \bmod 2) > 0 \wedge x > 4)$. □

Example 4.11. To illustrate ***mondec*** with a slightly more complicated input formula, take $\varphi(x, y)$ to be the predicate R^3 in Figure 1. Consider the result of $\boldsymbol{mondec}(\varphi)$ that starts with $(4, 4) \models \varphi$ so

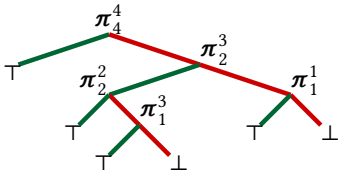


Fig. 3. $\boldsymbol{mondec}(R^3)$.

the root is $\boldsymbol{\pi}_4^4$. In the depiction of $\boldsymbol{mondec}(\varphi)$ in Figure 3, the left subtree of a node is the true case and right subtree of a node is the false case. For example, $\neg\boldsymbol{\pi}_4^4 \wedge \boldsymbol{\pi}_2^3 \wedge \boldsymbol{\pi}_2^2$ is a branch that implies φ, it covers the case $A_2 \times B_2$ in Figure 1(b). □

THEOREM 4.12. *$\boldsymbol{mondec}(\varphi)$ terminates iff φ is monadic. If $\boldsymbol{mondec}(\varphi)$ terminates then $\boldsymbol{mondec}(\varphi)$ is in MNF and $\boldsymbol{mondec}(\varphi)$ is equivalent to φ.*

PROOF. Assume φ is monadic. Assume also that φ is satisfiable or else it is trivially equivalent to $\bot$. Let A and B be as above. By using Lemma 4.3, $A_\sim$ and $B_\backsim$ are finite. Observe that the argument ν of ***dec*** remains of the form that all existential quantifiers occur *positively* in it, so the selection of $(a, b) \models \nu$ in ***dec*** is decidable (using the solver for Ψ).

The procedure ***mondec*** creates an if-then-else formula that can be thought of as a binary tree whose leaves are either $\top$ or $\bot$ and whose nodes are formulas $\boldsymbol{\pi}_b^a$ for some $a \in A$ and $b \in B$. The formula $\boldsymbol{mondec}(\varphi)$ is in MNF because each $\boldsymbol{\pi}_b^a$ is in MNF.

First, we show that $\boldsymbol{mondec}(\varphi)$ is well-defined (terminates) by showing that there are finitely many nodes. A new node $\boldsymbol{\pi}_b^a$ is created only when there exists $a \in A$ and $b \in B$ such that $(a, b) \models \nu$. In the subsequent recursive calls, any node that is equivalent to $\boldsymbol{\pi}_b^a$ is eliminated by the constraint $\boldsymbol{\nu}_b^a$. Termination follows because $A_\sim$ and $B_\backsim$ are finite and $\boldsymbol{\pi}_b^a \Leftrightarrow \boldsymbol{\pi}_{b'}^{a'}$ iff $a \sim a'$ and $b \backsim b'$.

Next, we show that ν must be satisfiable if both $\pi \wedge \varphi$ and $\pi \wedge \neg\varphi$ are satisfiable. Let $(a, b) \models \pi \wedge \varphi$ and $(a', b') \models \pi \wedge \neg\varphi$. We know that it is possible to strengthen π to π_1 so that π_1 is equivalent to $\alpha_a(x) \wedge \beta_b(y)$ and currently this is not the case because $a \nsim a'$ or $b \nbacksim b'$. Moreover, and without loss of generality, π_1 is of the form $\pi \wedge \psi$ where ψ is a conjunction of predicates $\boldsymbol{\pi}_d^c$ or $\neg\boldsymbol{\pi}_d^c$ for some $c \in A$ and $d \in B$. We have, by definition of ***dec***, that π has the form

$$\bigwedge_{i=1}^{m} \boldsymbol{\pi}_{b_i}^{a_i} \wedge \bigwedge_{i=m+1}^{n} \neg\boldsymbol{\pi}_{b_i}^{a_i}$$

for some $n \geq m \geq 0$ and $n \geq 1$, and that $\neg\nu$ is equivalent to $\bigvee_{i=1}^{n} a_i \sim x \wedge b_i \backsim y$. Thus, any use of a predicate $\boldsymbol{\pi}_d^c$ such that $(c, d) \models \neg\nu$ is useless because it makes $\boldsymbol{\pi}_d^c$ equivalent to some $\boldsymbol{\pi}_{b_i}^{a_i}$ for some i, $1 \leq i \leq n$, and so $\pi \wedge \boldsymbol{\pi}_d^c$ or $\pi \wedge \neg\boldsymbol{\pi}_d^c$ is either equivalent to π or to $\bot$. Therefore, ν must

be satisfiable or else π_1 cannot be constructed. Observe that if $\pi \wedge \boldsymbol{\pi}_b^a$ or $\pi \wedge \neg\boldsymbol{\pi}_b^a$ is unsatisfiable then the corresponding call to ***dec*** will return $\bot$.

To show that $\boldsymbol{mondec}(\varphi) \Leftrightarrow \varphi$ is immediate from the definition of ***dec***. First, consider a branch π in $\boldsymbol{mondec}(\varphi)$ ending in $\top$. We know that π implies φ as a condition for $\top$. The case $\neg\boldsymbol{mondec}(\varphi) \Rightarrow \neg\varphi$ is symmetrical by considering branches π in $\boldsymbol{mondec}(\varphi)$ ending in $\bot$. □

4.4 Improved search heuristic

We write $(a, b) \models \psi$ for $\{x = a, y = b\} \models_{\mathfrak{U}} \psi$ where x and y are the only free variables in ψ. We let $\boldsymbol{mondec}_1$ be a variant of ***mondec*** with the following difference, $\boldsymbol{mondec}_1$ uses the following strengthened constraint for selecting a witness (a, b) in Figure 2:

$$(a, b) \models \nu \wedge \pi \wedge \varphi$$

This heuristic states that, for selecting the witness (a, b), do so in the context of $\pi \wedge \varphi$. It follows from the proof of Theorem 4.6 that selecting $(a, b) \models \nu$ avoids formulas $\boldsymbol{\pi}_b^a$ that already occur (in equivalent form) as conjunct or negated conjunct of π. Since $\pi \wedge \varphi$ is satisfiable, say $(a, b) \models \pi \wedge \varphi$, but π does not imply φ (because $\pi \wedge \neg\varphi$ is also satisfiable) it must be possible to strengthen π to $\pi \wedge \boldsymbol{\pi}_b^a$ while at the same time $(a, b) \models \nu$. Observe that from $(a, b) \models \varphi$ follows that $(a, b) \models \boldsymbol{\pi}_b^a$.

As we will demonstrate in Section 6, this heuristic is important for performance of ***mondec***. Otherwise $(a, b) \models \nu$ may be chosen so that $\pi \wedge \boldsymbol{\pi}_b^a$ is unsatisfiable and therefore π implies $\neg\boldsymbol{\pi}_b^a$ and hence π is equivalent with $\pi \wedge \neg\boldsymbol{\pi}_b^a$. In this case the constructed node $\boldsymbol{\pi}_b^a$ in the if-then-else formula serves no purpose from the point of view of case analysis with respect to pairs from $\mathcal{U}_\sim \times \mathcal{U}_\sim$ that are relevant in the current branch condition π.

5 SOME DECIDABLE CASES

We illustrate decidability of monadic decomposition in some cases. Proposition 5.1 is also shown in [48, Corollary 7]. Its proof is nevertheless informative because it adds additional insight about problem structure and complexity. We also prove that monadic decomposition is decidable in EUF, i.e., in the presence of uninterpreted function symbols but no interpreted symbols besides equality ($\simeq$). Moreover, for *o-minimal* structures [68], there exists a uniform polynomial time decision procedure [48, Proposition 9] for variable independence. O-minimality means that there is a symbol that is interpreted as a linear order over the universe and every definable subset is a finite union of points and open intervals with respect to this order. Further decidable cases, are mentioned in Section 8.

5.1 Arithmetic

Consider first integer linear arithmetic. It clearly meets the requirements of $\mathfrak{U}$. Take a linear arithmetic formula $\varphi(x, y)$.

Proposition 5.1. *Monadic decomposition is decidable for integer linear arithmetic.*

Proof. Let $\varphi(x, y)$ be a formula in integer linear arithmetic. Let the predicate $\sim$ be defined as above, let '$x \in A$' stand for the formula $\exists y\varphi(x, y)$. Construct the following quantified formula:

$$\mathit{IsMonadic}(\varphi) \stackrel{\text{def}}{=} \exists\hat{x}(\forall x(x \in A \Rightarrow \exists x'(|x'| < \hat{x} \wedge x \sim x')))$$

We show that φ is monadic iff $\mathit{IsMonadic}(\varphi)$ is true in Presburger arithmetic. Decidability follows by [61]. Proof of $\Rightarrow$: Assume φ is monadic. Then $A_\sim$ is finite by Lemma 4.3. Let

$$\hat{a} = \max\{\min(\mathit{abs}(C)) \mid C \in A_\sim\} + 1,$$

where $abs(C)$ is the set of absolute values in C. Then, for all $a \in A$, a belongs to some C in $A_\sim$, and so there is $a' \in C$ such that $|a'| = \min(abs(C))$ and so $|a'| < \hat{a}$ and $a \sim a'$. Proof of $\Leftarrow$: Assume $IsMonadic(\varphi)$ holds. Choose a witness $\hat{a}$ for $\hat{x}$ and consider the classes $\mathcal{A} = \{[a]_\sim \mid 0 \leq |a| < \hat{a}\}$. It follows that $\mathcal{A} = A_\sim$ is finite, so φ is monadic by Lemma 4.3. □

The formula $IsMonadic(\varphi)$ has the quantifier prefix $\exists\forall\exists\forall$ in Prenex normal form when φ is quantifier free. So there are *three* quantifier alternations in $IsMonadic(\varphi)$. This implies an upper bound on time complexity $2^{2^{cn^7}}$ for some constant c and size n of φ for deciding if φ is monadic [63]. This is one exponent lower than the upper bound $2^{2^{2^{cn}}}$ known for the full Presburger arithmetic [28]. Moreover, the structure of the formula is quite specific and may justify the design of a special purpose algorithm.

Likewise, but for a different reason:

Proposition 5.2. *Monadic decomposition is decidable for real algebraic arithmetic with addition and multiplication.*

Proof. The atomic subformulas of φ are of the form $p(x, y) \geq 0$, where $p(x, y)$ is in general a multi-variate polynomial. Thus, for every value b, $\varphi(x, b)$ is a uni-variate polynomial, and the sign of such polynomials induce a finite set of intervals that partition the reals. Without loss of generality consider the case for an a, b and ϵ, such that for all b' where $\epsilon \geq b' > b$ we have $\varphi(a, b)$ but $\neg\varphi(a, b')$. Then φ contains an atomic formula $p(x, y) \geq 0$ whose truth value changes over b, b'. Monadicity of φ fails if it is determined by signs of polynomials $p(x, y)$ that depend on both x and y (recall that polynomials are continuous and differentiable). Thus, we can limit the search for a monadic decomposition up to the maximal number of regions induced by the polynomials in φ. This (potentially very large) number is bounded by the polynomial degrees and number of atomic subformulas. □

5.2 Uninterpreted functions

Here we briefly consider the problem of monadic decomposition in the presence of uninterpreted function symbols from a signature Σ. In this case we are no longer dealing with a single background structure, but with the class of all of its possible Σ-expansions. We limit the discussion here to pure logic, with equality as the only interpreted symbol, also called *EUF*. Deciding satisfiability of quantifier free formulas in EUF is decidable [35] and in fact NP-complete [8, Proposition 6.4.27], with practical decision procedures based on congruence closure [54] and completion [23, 47].

Consider a quantifier free Σ-formula $\varphi(x, y)$ with two free variables x and y. We lift the definition of monadic formulas to EUF as follows: $\varphi(x, y)$ is *monadic* if it has an equivalent monadic normal form (MNF) (recall the definition of MNF from section 3). For example, the formula $x \simeq y \wedge c \simeq x$ is monadic because it is equivalent to $x \simeq c \wedge y \simeq c$, while the formula $x \simeq y$ is not monadic.

Many classical techniques use disjunctive or conjunctive normal forms to reduce decision problems to a simpler form. Here, it is in general not enough to consider the DNF of a formula $\varphi(x, y)$ and to decide if the individual disjuncts are monadic or not in order to determine if $\varphi(x, y)$ is monadic or not. A trivial example is $e \vee \neg e$ for some equation e. The following example illustrates a slightly more complicated situation.

Example 5.3. Suppose φ has the form $(\psi \wedge \neg e_1 \wedge \neg e_2) \vee (\psi' \wedge e_1') \vee (\psi' \wedge e_2')$ where ψ is equivalent with ψ' and where ψ is in MNF but e_1, e_2, e_1' and e_2' are not monadic. Then φ has the equivalent form $\psi \wedge (\neg(e_1 \vee e_2) \vee e_1' \vee e_2')$. Moreover, suppose $\psi \models e_1 \Leftrightarrow e_1'$ and $\psi \models e_2 \Leftrightarrow e_2'$. Then φ is equivalent to ψ and thus φ is monadic. □

Moreover, as the following example illustrates, it is not enough to consider all subterms already occurring in a monadic formula as the only source of terms for constructing its MNF.

Example 5.4. Consider the formula $f(y) \simeq g(h(x)) \wedge h(x) \simeq f(c)$. Its MNF (that is essentially its only possible MNF) is $f(y) \simeq g(f(c)) \wedge h(x) \simeq f(c)$ where $g(f(c))$ is a term that does not occur in the original formula. □

We show next that monadicity is decidable in EUF. We make use of the following basic notions from term rewriting [23]. A term s is *embedded* in a term t, in symbols $s \leq_{\text{emb}} t$ if either s is a constant and $s = t$, or $s = f(s_1, \ldots, s_n)$ and $t = f(t_1, \ldots, t_n)$ for some n-ary $f \in \Sigma$, $n \geq 1$, and s_i is embedded in t_i for $1 \leq i \leq n$, or if s is embedded in a subterm of t.

A crucial point is that all *simplification orderings* [21] $\prec$ in term rewriting contain the strict embedding relation $<_{\text{emb}}$, i.e., if $s <_{\text{emb}} t$ then $s \prec t$. Σ is assumed to be *finite*, this is needed to guarantee that $\leq_{\text{emb}}$ is well-founded [22]. A simplification order $\prec$ extends $<_{\text{emb}}$ to a total order over terms. A central task, called *completion*, is to replace a set of equations E with a simpler set of directed equations (or rewrite rules) where all equations are of the form $l \to r$ with $r \prec l$. Let $E^{\prec}$ stand for some completion of E. Clearly, there are finitely many possible $E^{\prec}$.

Theorem 5.5. *The problem of deciding if a quantifier free formula φ in EUF is monadic is decidable.*

Proof. Let φ be given. All variables in φ are, from the point of view of rewriting, treated the same way as uninterpreted constants. First, we construct a finite set $\mathcal{K}_\varphi$ (*Kruskal closure* of φ) of terms that is going to be used as an over-approximation of all terms that are relevant during the search of an MNF candidate of φ. Let E be the set of all equations that occur in φ. Define $\mathcal{K}_\varphi$ to be the set of all terms t_i occurring in all possible sequences

$$t_0 \vdash_{E_1^{\prec}} t_1 \vdash_{E_2^{\prec}} t_2 \cdots \vdash_{E_i^{\prec}} t_i \cdots$$

where t_0 is a subterm that occurs in φ, $t_i \not\leq_{\text{emb}} t_j$ for $i < j$ and $t_i \vdash_R t_{i+1}$ means that t_{i+1} is obtained from t_i by rewriting a subterm of t_i using a rewrite rule in R. Each E_i is a subset of E. Each such sequence is finite by Kruskal's Tree theorem [45]. Moreover, since E is finite and the number of possible completions of E is finite, given t_i, the number of possible t_{i+1}'s such that $t_i \vdash_{E_i^{\prec}} t_{i+1}$ holds is finite. So the set $\mathcal{K}_\varphi$ is finite by König's Lemma. The sets $E_i^{\prec}$ above may potentially be pairwise different completions.

Now consider all possible conjunctions of literals over $\mathcal{K}_\varphi \times \mathcal{K}_\varphi$, these correspond to all possible pairs of subsets over $\mathcal{K}_\varphi \times \mathcal{K}_\varphi$ (one subset for atoms and one subset for negated atoms). The set Θ of all possible Boolean combinations of such conjunctions is also finite. So Θ represents DNFs of all possible formulas whose equations belong to $\mathcal{K}_\varphi \times \mathcal{K}_\varphi$.

We show that φ is monadic $\Leftrightarrow$ there exists $\psi \in \Theta$ such that ψ is in MNF and ψ is equivalent to φ, where the latter check is decidable because it reduces to unsatisfiability in EUF. The only nontrivial direction is $\Rightarrow$. Assume φ is monadic. So φ has an MNF ψ and there exists an equivalence preserving transformation from $\varphi = \psi_0$ to $\psi = \psi_n$ where each step of the transformation from ψ_i to ψ_{i+1} uses either, de MorganŠs laws, distributivity of connectives, logical simplifications, or in some logical context where a set of equations E_j holds, rewrites a term using some equation in E_j as a rewrite rule in some direction. This process may introduce new equations, but some subsets of the original E are enough to characterize all such rewriting steps by using Birkhoff's Theorem [7]. Suppose there is some term t in ψ that does not occur in $\mathcal{K}_\varphi$. Then there exists some t_0 in φ and

$$t_0 \vdash_{E_1} t_1 \vdash_{E_2} t_2 \cdots \vdash_{E_n} t_n = t.$$

where each E_i is a subset of E. Each equation in E_i is viewed as a rewrite rule in both directions, e.g., an equation $x \simeq f(x)$ would correspond to two rules $x \to f(x)$ and $f(x) \to x$ (where the first

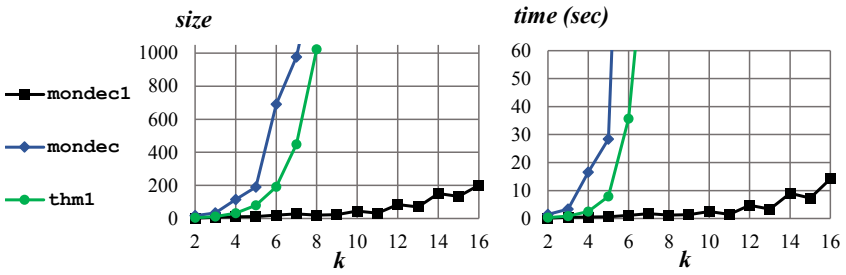


Fig. 4. Micro benchmark.

one violates the embedding relation because $x <_{\text{emb}} f(x)$).[7] To show that there exists $\psi' \in \Theta$ that is equivalent to ψ it is enough to show that we can replace such t by some term t' from $\mathcal{K}_\varphi$ such that $vars(t') \subseteq vars(t)$, which preserves MNF, where $vars(u)$ denotes the set of all variables in a term u.

Observe that if $u <_{\text{emb}} v$ then $vars(u) \subseteq vars(v)$. We can consider a class of simplification orders $\prec$ over terms such that if $vars(u) \subsetneq vars(v)$ then $u \prec v$ because then $v \not<_{\text{emb}} u$ and if $vars(u) \not\subseteq vars(v)$ and $vars(v) \not\subseteq vars(u)$ then both $u \prec v$ as well as $v \prec u$ are possible extensions because then $u \not<_{\text{emb}} v$ and $v \not<_{\text{emb}} u$. In particular, we can choose which variable is "smallest", depending on which one (if any) that occurs in t. An example of the latter case would be an equation $f(x) \simeq g(y)$. We now have that, for some such order $\prec$ and completions of E_i that respect $\prec$, there exists a sequence

$$t'_0 = t_0 \vdash^*_{E_1^\prec} t'_1 \vdash^*_{E_2^\prec} t'_2 \cdots \vdash^*_{E_n^\prec} t'_n = t'.$$

where $E_i \models t'_i \simeq t_i$ and $t'_j \preceq t'_i$ for $j \geq i$ by virtue of using the completed rule sets and by transitivity of the simplification order [21]. Moreover $vars(t') \subseteq vars(t)$ for some $\prec$. This means that we can replace the occurrence of t in ψ by t' and the resulting formula will be equivalent to ψ and remains in MNF. Assume, w.l.o.g., that $t'_i \neq t'_{i+1}$. It follows that $t'_i \not\leq_{\text{emb}} t'_j$ for $i < j$ and so $t' \in \mathcal{K}_\varphi$. □

6 MICRO BENCHMARK

We present here a micro benchmark by revisiting the sample predicate R^k from Example 4.4 and by letting k range from 2 to 16; k also happens to be the monadic width of R^k.

The worst case scenario of the size of a monadic decomposition of R^k, according to Theorem 4.6(c), is $O(k2^k)$ because $|A_\sim| = 2^k$ and $|B_\smile| = k$ (including the classes $[0]_\sim$ and $[0]_\smile$). We compare three algorithms, implemented as Z3 python scripts, that are indicated in Figure 4 by `thm1`, `mondec`, and `mondec1`. The output is in all cases in MNF represented by an if-then-else formula, its *size* is the number of $\boldsymbol{\pi}^a_b$ nodes in it, e.g., the size of the expression in Figure 3 is 5.[8] Algorithm `thm1` is based on Theorem 4.6. Algorithm `mondec1` is a variant of `mondec` as discussed above; the python script of `mondec1` is shown in the Appendix.

The most interesting aspect about the experiment is that it shows that different (very simple) heuristics can influence the performance characteristics of monadic decomposition by an exponential factor. The heuristic in `mondec1` reduces the size of the decomposition exponentially in this experiment, while constructing nodes in `mondec` based solely on v, provides worse performance than exhaustive search of W_A and W_B, as in `thm1`. For example, the time to decompose R^9

[7] Recall that variables are treated as uninterpreted constants here, so we are dealing with ground rewriting.
[8] The experiments were carried out on a laptop with a 2GHz CPU.

with mondec gave an output of size 2281 and took around 11 minutes, while with mondec1 the output size was 23 and the decomposition took 1.4 seconds. The reason why the naive selection of $(a, b) \models \nu$ in Figure 2 performs so poorly compared to $(a, b) \models \nu \wedge \pi \wedge \varphi$, is that in the first case a lot of irrelevant nodes are created in the if-then-else formula, while in the second case the search is directed by the current path π and the formula φ.

The formulas arising in Section 2 are too small for comparison among the algorithms, but thm1 is in general impractical because it always requires a DNF and requires all the witnesses in W_A as well as W_B, while mondec (and mondec1) takes advantage of subexpression sharing in the Shannon expansion that the constructed if-then-else formula corresponds to. Moreover, mondec works for any number of variables, not just two.

7 SAMPLE APPLICATIONS

We describe two different applications of monadic decomposition. Both are related to string processing applications. The first application illustrates safety analysis of *Bek* programs that describe string sanitizers [39]. The second application illustrates composition of symbolic automata with lookahead. The concrete scenario that is illustrated considers equivalence checking of ESFTs [17] where ESFTs need to be *Cartesian* and transformation to Cartesian form requires monadic decomposition. Monadic decomposition is also used during exploration of symbolic finite transducers that use *registers* [74] where a register may be eliminated or made implicit by *grouping* two or more consecutive characters together. Later, the grouped characters may need to be monadically decomposed to expose parallelism.

The scenarios illustrate cases where monadic decomposition is done using an implementation of ***mondec***. Although the scenarios are developed here by hand, they mimic fully automated steps in larger tool chains. The additional time overhead caused by monadic decomposition, when it works, is typically in the order of tens of milliseconds and is negligible for these examples. Our practical experience with ***mondec*** is still quite limited at this stage. The main limiting factors have been, difficulty of integration into other tools, use of complex theories like nonlinear arithmetic, and need for quantifier elimination before monadic decomposition can be applied, as illustrated in the next section.

7.1 String sanitizer analysis

Bek is a programming language that is designed for implementing sanitizers [39]. Sanitizers are special purpose string encoders that escape or remove potentially dangerous strings in order to prevent unauthorized script execution. Analysis of Bek programs builds on the theory and algorithms of *Symbolic Finite Transducers* or *SFTs* [73]. One challenging property of SFTs is that the *range*, i.e., the set of all possible output sequences, of an SFT is not necessarily regular even though its domain is always regular. *Regularity* in the symbolic setting is defined as *acceptance by a symbolic finite automaton* or *SFA*. While monadic decomposition is *necessary* to decide range regularity of SFTs it is *insufficient* because in general we also need to apply quantifier elimination prior to attempting monadic decomposition.

Regularity is important because it enables the application of a whole range of SFA algorithms, such as *intersection* and *complementation*, and breaks the boundary between decidable and undecidable cases. In contrast, a range automaton of an SFT is in general an *Extended* SFA or ESFAs, an SFAs with *lookahead* where a single transition may read more than one character in a single atomic step when transitioning from one state to the next state. Intersection emptiness of ESFAs is undecidable [17]. This is in sharp contrast with the classical theory of finite transductions where

it is known, as a consequence of Nivat's Representation Theorem of finite transductions, that both the domain as well as the range of a finite transduction are regular [58, 75].

As a trivial example of an SFT whose range is not regular, take one that has an infinite alphabet, a single state q, and a single transition $q \xrightarrow{true/[x,\,x]} q$. For example, if the input is $[a, b, c]$ then the ouput is $[a, a, b, b, c, c]$. Its range is not regular because it contains a sequence $[x, y]$ if and only if $x = y$ and it would require infinitely many states to remember x for comparison with y.

Figure 5 illustrates an SFT that UTF8 encodes Unicode characters, where $x_{\langle h,l\rangle}$ extracts bits from h to l from a 32-bit bit-vector x, e.g., $8_{\langle 3,2\rangle} = 10_2$. Surrogates are not valid Unicode characters, $Surrogate(x) \stackrel{\text{def}}{=} \texttt{D800}_{16} \leq x \leq \texttt{DFFF}_{16}$. Bit-append is denoted by $x \cdot y$, e.g., $110_2 \cdot x_{\langle 10,6\rangle} = \texttt{C0}_{16} + x_{\langle 10,6\rangle}$.

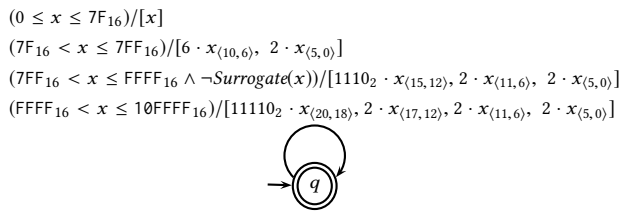


Fig. 5. SFT *EncUTF8* is a UTF8 encoder for valid Unicode code points.

In some analysis scenarios with SFTs it is useful to decide if the range of an SFT is regular and, if so, to construct the corresponding SFA. We take UTF8 encoding as an example because it is ubiquitous (even some sanitizers use UTF8 encoding as the first encoding step). Here we know that the range must be regular because the Unicode alphabet is finite (in theory at least). The input to the encoder is a sequence of Unicode code points, that are integers from 0 to $\texttt{10FFFF}_{16}$, and the output is a sequence of bytes. The complete SFT of a UTF8 encoder can be described with one state and four transitions, see Figure 5, each transition corresponds to the length of the encoding of the code point.[9]

A naive extraction of a range automaton of *EncUTF8* gives rise to an ESFA. For example, the second rule of *EncUTF8* becomes the following transition of the ESFA and has lookahead 2, i.e., it reads 2 bytes at a time

$$\lambda(y, z).\exists x(\texttt{7F}_{16} < x \leq \texttt{7FF}_{16} \wedge y = (6 \cdot x_{\langle 10,6\rangle}) \wedge z = (2 \cdot x_{\langle 5,0\rangle}))$$

The existential quantifier over x can be eliminated automatically by using any known quantifier elimination technique for integer linear arithmetic [57]. For ease of presentation we use the fact that $x = y_{\langle 4,0\rangle} \cdot z_{\langle 5,0\rangle}$. This gives us the equivalent transition

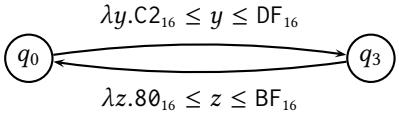
$$\lambda(y, z).\texttt{7F}_{16} < (y_{\langle 4,0\rangle} \cdot z_{\langle 5,0\rangle}) \leq \texttt{7FF}_{16} \wedge y = (6 \cdot y_{\langle 4,0\rangle}) \wedge z = (2 \cdot z_{\langle 5,0\rangle})$$

[9]The corresponding encoder in [15, Figure 3] uses 5 states and 11 transitions because there the input is assumed to be UTF16 encoded.

As it happens, the guard of the transition is Cartesian, and monadic decomposition and some simplification of the resulting formula gives rise to the following equivalent transition,

$$\lambda(y,z).\mathtt{C2}_{16} \le y \le \mathtt{DF}_{16} \wedge \mathtt{80}_{16} \le z \le \mathtt{BF}_{16}$$

that is equivalent after removing the lookahead to



where q_3 is a new state.

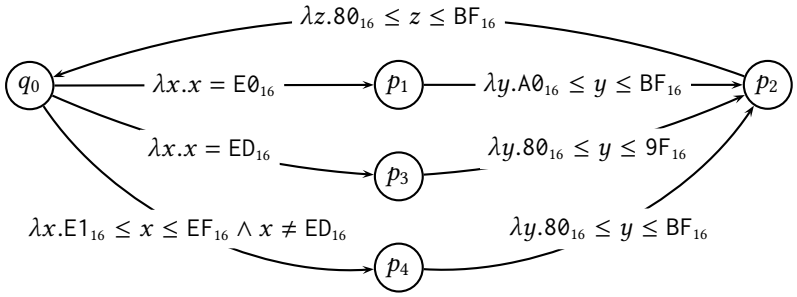
The third rule of *EncUTF8* is a bit more challenging. As above, we start with a transition where the existential quantifier has been eliminated. We get the following ESFA transition. The transition has lookahead 3, i.e., it reads a block of 3 characters at once

$$\lambda(x,y,z).\mathtt{7F}_{16}{<}x_{\langle 3,0\rangle} \cdot y_{\langle 5,0\rangle} \cdot z_{\langle 5,0\rangle}{\le}\mathtt{7FF}_{16} \wedge x{=}\mathtt{E}_{16}{\cdot}x_{\langle 3,0\rangle} \wedge y{=}2{\cdot}y_{\langle 5,0\rangle} \wedge z{=}2{\cdot}z_{\langle 5,0\rangle}$$

Monadic decomposition of the guard is applied here to ternary relations and one possible outcome is the following MNF (the monadic width is 3):

$$\begin{aligned}\lambda(x,y,z).\quad & ((x = \mathtt{E0}_{16} \wedge \mathtt{A0}_{16} \le y \le \mathtt{BF}_{16}) \vee (x = \mathtt{ED}_{16} \wedge \mathtt{80}_{16} \le y \le \mathtt{9F}_{16}) \vee \\ & (\mathtt{E1}_{16} \le x \le \mathtt{EF}_{16} \wedge x \ne \mathtt{ED}_{16} \wedge \mathtt{80}_{16} \le y \le \mathtt{BF}_{16})) \wedge \mathtt{80}_{16} \le z \le \mathtt{BF}_{16}\end{aligned}$$

that is used to replace the loop by the following equivalent set of transitions that do not use lookahead and where all the p_i are new states



The same procedure is repeated for the fourth rule. After combining all the transitions together and minimizing [16] the resulting SFA we obtain the SFA in Figure 6 that accepts the range of *EncUTF8*.

One can now show that the SFA rejects any *over-encoded* sequences such as $[\mathtt{C0}_{16}, \mathtt{AE}_{16}]$ (that decodes to '.') or $[\mathtt{C0}_{16}, \mathtt{AF}_{16}]$ (that decodes to '/') and consequently that *EncUTF8* does not over-encode inputs. The SFA can also be translated into a regular expression. Such analysis can be useful

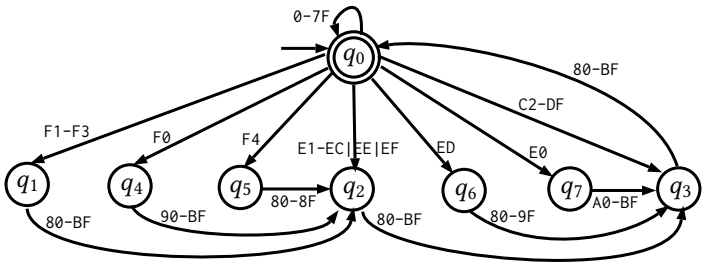


Fig. 6. Minimal SFA that recognizes valid UTF8 encoded strings.

for showing absence of security holes in encoders that could otherwise be vulnerable to exploits such as directory traversal attacks [60, 65].

7.2 ESFA parallel composition

In general, ESFAs are not closed under parallel composition, but in some situations it may nevertheless be useful to attempt to compose two ESFAs, e.g., during equivalence checking of single-valued ESFTs [17]. This problem also illustrates a use case when we dynamically select a variable partitioning and may not have to monadically decompose all the variables. The main step of composing two automata is a product construction of their transitions that is usually computed with a depth first search procedure. The problem with ESFAs is that they may not make progress in lock step.

For example, consider two ESFAs A and B. Suppose that A has a transition $p \xrightarrow{\varphi} p'$ with lookahead 2 and B has a transition $q \xrightarrow{\psi} q'$ with lookahead 3 and we want to compose the two transitions in parallel. Lookahead 2 means that $[\![\varphi]\!] \subseteq \mathcal{U} \times \mathcal{U}$ and lookahead 3 means that $[\![\psi]\!] \subseteq \mathcal{U} \times \mathcal{U} \times \mathcal{U}$.

By composing the transitions in parallel we mean that we start from the pair state (p, q) and simultaneously transition in A and in B. To do so we need to decompose $\lambda(x, y).\psi(\pi_1(x), \pi_2(x), y)$ into MNF where we partitioned the free variables (x_1, x_2, y) of $\psi(x_1, x_2, y)$ into blocks $\{\{x_1, x_2\}, \{y\}\}$. We can then introduce intermediate states (based on the resulting MNF) and continue the parallel composition of A and B in a depth first manner. Note that the order of variables in (x_1, x_2, y) is important. Automata theoretically, you read the subsection (a, b, c) such that $\psi(a, b, c)$ holds and then move past that subsection.

For example suppose that $\lambda(x, y).\psi(\pi_1(x), \pi_2(x), y)$ is Cartesian. Then it has the equivalent form $\lambda(x, y).\psi_1(\pi_1(x), \pi_2(x)) \wedge \psi_2(y)$ for some ψ_1 and ψ_2. Now introduce a new state q_1 and replace the transition $q \xrightarrow{\psi} q'$ by the transitions $q \xrightarrow{\psi_1} q_1 \xrightarrow{\psi_2} q'$. The transitions $q \xrightarrow{\psi_1} q_1$ and $p \xrightarrow{\varphi} p'$ can now be composed in parallel to $(p, q) \xrightarrow{\psi_1 \wedge \varphi} (p', q_1)$ and, provided that $\psi_1 \wedge \varphi$ is satisfiable, the parallel composition continues in depth first style from the pair state (p', q_1). The point with this scenario is that the variables x_1 and x_2 need not be independent in $\psi(x_1, x_2, y)$.

8 RELATED WORK

Study of monadic fragments of logic was started by Löwenheim in 1915 and spans a full *century* of literature by now. Work related to automata theory and its relation to monadic fragments of logic is, likewise, a very thoroughly studied topic [67]. Despite this, there is renewed interest in this topic, but with a new angle. From our perspective, this is due to many advances in *automated logical inference engines*. The angle is, how to make use of such advances in a modular way in the context of automata theoretic problems. This makes questions like the one posed in this paper relevant in many different potential application areas. Monadic decomposition can also be used to

study new decidable fragments of logics; revisiting techniques in [6, 26, 36] could be relevant in this context.

Variable independence. Systematic study of variable independence was initiated by Libkin [48], with motivation coming from database applications such as spatio-temporal analysis and query optimization in constraint databases [13, 34]. In our conference paper on monadic decomposition [72] we were unaware of Libkin's work. In particular, we learned that the general problem of monadic decomposition is undecidable as a consequence of [48, Proposition 2(b)]. Some of our results directly benefit from the work in [48] and we also complement that work by providing a new algorithm to solving variable independence problems with off-the-shelf SMT technology that may also be beneficial for database applications. The main difference is that we focus on the quantifier free fragment of theory combinations. In the general case we do not assume that the full theory is decidable. Extension of SMT with quantifiers is in itself an active research area [31]. One approach to eliminate quantifiers in theory based reasoning is to apply monadic decomposition [44]. Part of the motivation in [48] also comes from quantifier elimination in specific theories. For *o-minimal* structures [68], there exists a uniform polynomial time decision procedure [48, Proposition 9] for variable independence. For example, considering integers and the standard order $<$, the ability to define that an integer is even violates o-minimality. In the context of SMT, o-minimality seems too restrictive. Theorem 4.6 is related [48, Theorem 3] where the latter looks at a general class of structures whose theory is decidable and where deciding finiteness and Skolemization is definable in the language. The resulting formulas use quantifier alternations. The only allowed quantifiers in our case are positive occurrences of existential quantifiers, that essentially correspond to "don't care" free variables.

The problem appears also in disguise in studies involving relationships between different classes of formal languages and relations [12, 25]. A *recognizable relation* is a finite union of products of regular languages, i.e., it has a form of monadic decomposition. A *regular relation* is a relation definable in the canonical automatic structure $\mathrm{S}_{\mathrm{len}}$ [5]. Decidability of checking whether a regular relation is recognizable follows from [48, Theorem 3] because the preconditions of the theorem are easily met: $\mathrm{S}_{\mathrm{len}}$ is decidable, has an effective test for finiteness, and Skolemization is definable through lexicographic ordering. Another interesting fact about variable independence of a formula $\varphi(\bar{x})$ is that there is a unique most refined partition of $\bar{x}$ that defines its variable independence [14, Corollary 1].

Variable independence is also known to be decidable for monadic second-order logic queries over trees and a corresponding monadic decomposition is computable [27]. A restricted form of variable independence is also equivalent to non-ambiguity of tree automata [55].

Monadic fragments. Unary relations play a key role in many decision problems and decidable logics. *Monadic first-order logic*, or the Löwenheim class [50], is the classical example of a decidable fragment of first-order logic where all symbols are unary relation symbols. The Löb-Gurevich class [49], is the extension of the Löwenheim class where also unary function symbols are allowed. Both classes are decidable by having the *finite model property* [8]. *Monadic second-order logic* allows quantification over unary predicates. Among one of the most celebrated and applied decidability results are those of the monadic second-order theory *S1S* with one successor relation by Büchi [11] and decidability of the monadic second-order theory *S2S* of the binary tree with two successor relations by Rabin [62]. The ability to apply Rabin's theorem and automata based techniques to establish decidability results of a logic is often described as the logic having the *tree model property*. *Modal logics* do not have the finite model property but they do have the tree model property. Vardi attributes [69] their decidability to this. Grädel discusses this topic further in [33] and its relation

to the *guarded fragment* [2]. Unlike in modal logics, simple extensions of the guarded fragment cause undecidability [32], one exception is the *monadic* guarded fragment with two variables and equivalence relations that does have the tree model property [30]. The theorems of Büchi and Rabin have also been revisited and extended by Gurevich through game based techniques [36]. Another technique discussed in [36] is the use of the *Feferman-Vaught* generalized products [26] as a model-theoretic method for establishing decidability results in the context of monadic second-order logic. The Feferman-Vaught theorem reduces the theory of the given composition to the theories of the parts and the monadic theory of the index structure. This is analogous to separating the label theory from the finite state graph structure of an SFA.

Symbolic automata. Remarkably, the Feferman-Vaught theorem is revisited in [6] where it is shown that a special version of it is closely related to the theory of $\mathfrak{M}$*-automata* where $\mathfrak{M}$ is a first-order structure. Although $\mathfrak{M}$-automata are defined as *multi-tape* automata, by using tuples, they correspond precisely to SFAs. Independently, a variant of SFAs was originally introduced in the context of natural language processing, where they are called *predicate-augmented finite state recognizers* [59]. Symbolic finite transducers were introduced in [73], a different notion of symbolic transducers is also studied in [59]. The extension from SFTs to ESFTs is introduced in [15]. Equivalence of ESFTs, properties of ESFAs, and the notion of Cartesian ESFTs are studied in [17]. The monadic decomposition problem first surfaced in the context of trying to lift algorithms for symbolic automata *without lookahead* to symbolic automata *with lookahead*. ESFTs are also similar to k-SLTs [9] but have a different step semantics.

In classical automata theory lookahead can be eliminated by introducing more states since the alphabet is finite. Most other SFA algorithms can, in theory, be lifted to finite alphabets. For example, closure under complement [6, Proposition 2.6] is shown by reduction to NFA determinization through *minterm* construction by considering the Boolean combinations of all guards of the $\mathfrak{M}$-automaton as the finite alphabet of the NFA. Practically this approach does not scale, it suffers from an exponential blowup of the number of transitions, even before the actual NFA determinization algorithm starts.

Applications. For many analysis tasks, some of which are discussed in Section 2, monadic decomposition plays a key role in enabling the use of SFA and SFT algorithms in the context of symbolic automata and transducers. Other SFA algorithms, such as difference and complement, are discussed in [71] in the context of SMT solvers, and more algorithms are discussed in [40] in the more specialized context of string analysis. A symbolic automata toolkit is described in [70]. SFT algorithms, in particular equivalence checking, are studied in [73] and their use for web security is discussed in [39]. A new minimization algorithm of SFAs was recently presented in [16], showing that the new algorithm can enable some analysis scenarios involving monadic second-order logic that did not scale with earlier techniques. Formulas in monadic second-order logic over *finite sequences* can be reduced to SFAs [18], the reduction algorithm extends the classical one [67] and the logic extends the corresponding M2L-STR logic in Mona [38, 41].

9 CONCLUSION

We introduced the problem of monadic decomposition of predicates in decidable quantifier-free theories. Theorem 4.6 provided an effective means to computing a monadic decomposition and we described an implementation with correctness proof, Theorem 4.12, that avoids expanding solutions directly into DNF; it leverages a Shannon decomposition. We also provided a concrete executable implementation of the main algorithm using the Z3 module in python. We used results by Libkin to infer undecidability and decidability results of monadic decomposition, and showed

decidability of monadic decomposition in EUF. We left the design of concrete algorithms for deciding if a formula is monadic in specific theories as future work.

Acknowledgements

We thank Leonid Libkin for valuable comments on a preliminary version of this paper and for pointing out references to several related works. We are also very grateful to the anonymous referees for their valuable and detailed comments.

REFERENCES

[1] Alfred V. Aho, Monica S. Lam, Ravi Sethi, and Jeffrey D. Ullman. 2006. *Compilers: Principles, Techniques, and Tools (2nd Edition)* (2nd ed.). Addison Wesley.
[2] Hajnal Andréka, István Németi, and Johan van Benthem. 1998. Modal Languages and Bounded Fragments of Predicate Logic. *Journal of Philosophical Logic* 27 (1998), 217–274. Issue 3.
[3] M. Baaz, U. Egly, and A. Leitsch. 2001. Normal form transformations. In *Handbook of Automated Reasoning*, A. Robinson and A. Voronkov (Eds.). Vol. I. North Holland, Chapter 5, 273–333.
[4] Clark Barrett, Christopher L. Conway, Morgan Deters, Liana Hadarean, Dejan Jovanović, Tim King, Andrew Reynolds, and Cesare Tinelli. 2011. CVC4. In *Proceedings of the 23rd International Conference on Computer Aided Verification, CAV'11 (LNCS)*. Springer, 171–177.
[5] Michael Benedikt, Leonid Libkin, Thomas Schwentick, and Luc Segoufin. 2003. Definable Relations and First-order Query Languages over Strings. *J. ACM* 50, 5 (2003), 694–751.
[6] Alexis Bés. 2008. An application of the Feferman-Vaught Theorem to automata and logics for words over an infinite alphabet. *Logical Methods in Computer Science* 4 (2008), 1–23.
[7] Garrett Birkhoff. 1935. On the Structure of Abstract Algebras. *Mathematical Proceedings of the Cambridge Philosophical Society* 31 (1935), 433–454.
[8] Egon Börger, Erich Grädel, and Yuri Gurevich. 1997. *The Classical Decision Problem*. Springer.
[9] Matko Botincan and Domagoj Babic. 2013. Sigma*: symbolic learning of input-output specifications. *ACM SIGPLAN Notices - POPL'13* 48, 1 (2013), 443–456.
[10] Aaron R. Bradley, Zohar Manna, and Henny B. Sipma. 2006. What's Decidable About Arrays?. In *Verification, Model Checking, and Abstract Interpretation: 7^{th} International Conference, (VMCAI'06) (LNCS)*, Vol. 3855. 427–442.
[11] J. Richard Büchi. 1962. On a decision method in restricted second order arithmetic. In *Logic, Methodology and Philosophy of Science (Proc. 1960 Internat. Congr .)*, E. Nagel, P. Suppes, and A. Tarski (Eds.). Stanford Univ. Press, 1–11.
[12] Olivier Carton, Christian Choffrut, and Serge Grigorieff. 2006. Decision problems among the main subfamilies of rational relations. *RAIRO-Theor. Inf. Appl.* 40, 2 (2006), 255–275.
[13] J. Chomicki, D. Goldin, G. Kuper, and D. Toman. 2003. Variable independence in constraint databases. *IEEE Transactions on Knowledge and Data Engineering* 15, 6 (2003), 1422–1436.
[14] Stavros Cosmadakis, Gabriel Kuper, and Leonid Libkin. 2001. On the Orthographic Dimension of Definable Sets. *Inf. Process. Lett.* 79, 3 (2001), 141–145.
[15] Loris D'Antoni and Margus Veanes. 2013. Static Analysis of String Encoders and Decoders. In *VMCAI 2013 (LNCS)*, R. Giacobazzi, J. Berdine, and I. Mastroeni (Eds.), Vol. 7737. Springer, 209–228.
[16] Loris D'Antoni and Margus Veanes. 2014. Minimization of Symbolic Automata. *ACM SIGPLAN Notices - POPL'14* 49, 1 (2014), 541–553.
[17] Loris D'Antoni and Margus Veanes. 2015. Extended symbolic finite automata and transducers. *Formal Methods in System Design* (July 2015).
[18] Loris D'Antoni and Margus Veanes. 2017. Monadic Second-order Logic on Finite Sequences. In *Proceedings of the 44th ACM SIGPLAN Symposium on Principles of Programming Languages, POPL'17*. ACM, 232–245.
[19] Leonardo de Moura and Nikolaj Bjørner. 2008. Z3: An Efficient SMT Solver. In *TACAS'08 (LNCS)*. Springer.
[20] Leonardo de Moura and Nikolaj Bjørner. 2011. Satisfiability modulo theories: introduction and applications. *Commun. ACM* 54, 9 (2011), 69–77.
[21] Nachum Dershowitz. 1979. A note on simplification orderings. *Inform. Process. Lett.* 9, 5 (1979), 212 – 215.
[22] Nachum Dershowitz. 1982. Orderings for term-rewriting systems. *Theoretical Computer Science* 17, 3 (1982), 279–301.
[23] Nachum Dershowitz and David A. Plaisted. 2001. Rewriting. In *Handbook of Automated Reasoning*, Alan Robinson and Andrei Voronkov (Eds.). Vol. 1. North Holland, Chapter 9, 535–610.
[24] Bruno Dutertre. 2014. Yices 2.2. In *Computer-Aided Verification, CAV'14 (LNCS)*, Armin Biere and Roderick Bloem (Eds.), Vol. 8559. Springer, 737–744.

[25] C. C. Elgot and J. E. Mezei. 1965. On Relations Defined by Finite Automata. *IBM Journal* 10 (1965), 47–68.
[26] S. Feferman and R. L. Vaught. 1959. The first-order properties of algebraic systems. *Fundamenta Mathematicae* 47 (1959), 57–103.
[27] Emmanuel Filiot and Sophie Tison. 2008. *Regular n-ary Queries in Trees and Variable Independence*. Springer, 429–443.
[28] M. J. Fischer and M. O. Rabin. 1974. Super-Exponential Complexity of Presburger Arithmetic. In *SIAMAMS: Complexity of Computation: Proceedings of a Symposium in Applied Mathematics of the American Mathematical Society and the Society for Industrial and Applied Mathematics*. 27–41.
[29] Harald Ganzinger, George Hagen, Robert Nieuwenhuis, Albert Oliveras, and Cesare Tinelli. 2004. DPLL(T): Fast Decision Procedures. In *Computer Aided Verification, 16th International Conference, CAV'04*. 175–188.
[30] H. Ganzinger, C. Meyer, and M. Veanes. 1999. The Two-Variable Guarded Fragment with Transitive Relations. In *LICS'99*. IEEE, 24–34.
[31] Yeting Ge, Clark Barrett, and Cesare Tinelli. 2009. Solving quantified verification conditions using satisfiability modulo theories. *Annals of Mathematics and Artificial Intelligence* 55, 1-2 (2009), 101–122.
[32] Erich Grädel. 1998. On the Restraining Power of Guards. *Journal of Symbolic Logic* 64 (1998), 1719–1742.
[33] Erich Grädel. 1999. Why Are Modal Logics So Robustly Decidable? *Bulletin EATCS* 68 (1999), 90–103.
[34] Stéphane Grumbach, Philippe Rigaux, and Luc Segoufin. 1998. The DEDALE System for Complex Spatial Queries. In *SIGMOD 1998, Proceedings ACM SIGMOD International Conference on Management of Data*. 213–224.
[35] Yuri Gurevich. 1976. The decision problem for standard classes. *Journal of Symbolic Logic* 41 (1976), 460–464.
[36] Yuri Gurevich. 1985. Monadic second-order theories. In *Model-Theoretical Logics*, Jon Barwise and Sol Feferman (Eds.). Springer, Chapter XIII, 479–506.
[37] John V. Guttag, Ellis Horowitz, and David R. Musser. 1978. Abstract Data Types and Software Validation. *Commun. ACM* 21, 12 (1978), 1048–1064.
[38] J.G. Henriksen, J. Jensen, M. Jørgensen, N. Klarlund, B. Paige, T. Rauhe, and A. Sandholm. 1995. Mona: Monadic Second-order logic in practice. In *TACAS '95 (LNCS)*, Vol. 1019. Springer.
[39] P. Hooimeijer, B. Livshits, D. Molnar, P. Saxena, and M. Veanes. 2011. Fast and precise sanitizer analysis with Bek. In *Proceedings of the USENIX Security Symposium*.
[40] Pieter Hooimeijer and Margus Veanes. 2011. An Evaluation of Automata Algorithms for String Analysis. In *VMCAI'11 (LNCS)*, Vol. 6538. Springer, 248–262.
[41] Nils Klarlund, Anders Møller, and Michael I. Schwartzbach. 2002. MONA Implementation Secrets. *International Journal of Foundations of Computer Science* 13, 4 (2002), 571–586.
[42] Konstantin Korovin. 2009. Instantiation-based automated reasoning: From theory to practice. In *Automated Deduction - CADE-22, 22nd International Conference on Automated Deduction (LNAI)*, Vol. 5663. Springer, 2–7.
[43] K. Korovin. 2013. Inst-Gen – A Modular Approach to Instantiation-Based Automated Reasoning. In *Programming Logics, Essays in Memory of Harald Ganzinger*. Springer.
[44] Konstantin Korovin and Margus Veanes. 2014. Skolemization Modulo Theories. In *Mathematical Software, ICMS 2014*, Hoon Hong and Chee Yap (Eds.). LNCS, Vol. 8592. Springer, 303–306.
[45] J. B. Kruskal. 1960. Well-Quasi-Ordering, The Tree Theorem, and Vazsonyi's Conjecture. *Trans. Amer. Math. Soc.* 95, 2 (1960), 210–225.
[46] R. E. Ladner and M. J. Fischer. 1980. Parallel prefix computation. *J. ACM* 27, 4 (1980), 831–838.
[47] Dallas S. Lankford. 1975. *Canonical Inference*. Technical Report ATP-32. Southwestern University.
[48] Leonid Libkin. 2003. Variable Independence for First-order Definable Constraints. *ACM Trans. Comput. Logic* 4, 4 (Oct. 2003), 431–451.
[49] M. Löb. 1967. Decidability of the monadic predicate calculus with unary function symbols. *Journal of Symbolic Logic* 32 (1967), 563.
[50] L. Löwenheim. 1915. Über Möglichkeiten im Relativkalkül. *Math. Annalen* 76 (1915), 447–470.
[51] Anatoli Mal'cev. 1971. *The Metamathematics of Algebraic Systems*. North-Holland.
[52] Todd Mytkowicz, Madanlal Musuvathi, and Wolfram Schulte. 2014. Data-parallel Finite-state Machines. *ACM SIGPLAN Notices - ASPLOS'14* 49, 4 (2014), 529–542.
[53] Greg Nelson and Derek C. Oppen. 1979. Simplification by cooperating decision procedures. *ACM Transactions on Programming Languages and Systems* 1 (1979), 245–257.
[54] Ggerg Nelson and Derek C. Oppen. 1980. Fast Decision Procedures Based on Congruence Closure. *J. ACM* 27, 2 (April 1980), 356–364.
[55] Joachim Niehren, Laurent Planque, Jean-Marc Talbot, and Sophie Tison. 2005. N-ary Queries by Tree Automata. In *Proceedings of the 10th International Conference on Database Programming Languages, DBPL'05*. Springer, 217–231.
[56] Flemming Nielson, Hanne R. Nielson, and Chris Hankin. 2010. *Principles of Program Analysis*. Springer.

[57] Tobias Nipkow. 2008. Linear Quantifier Elimination. In *4th International Joint Conference on Automated Reasoning, IJCAR'08 (LNAI)*, Vol. 5195. Springer, 18–33.
[58] Maurice Nivat. 1968. Transductions des langages de Chomsky. *Annales de l'institut Fourier* 18, 1 (1968), 339–455.
[59] Gertjan Van Noord and Dale Gerdemann. 2001. Finite State Transducers with Predicates and Identities. *Grammars* 4 (2001), 263–286.
[60] NVD. http://web.nvd.nist.gov/view/vuln/detail?vulnId=CVE-2008-2938. (????).
[61] M. Presburger. 1929. Über die Vollständigkeit eines gewissen Systems der Arithmetik ganzer Zahlen, in welchem die Addition als einzige Operation hervortritt. In *In Comptes Rendus du I congrés de Mathématiciens des Pays Slaves*. Warsaw, Poland, 92–101.
[62] Michael O. Rabin. 1969. Decidability of second-order theories and automata on infinite trees. *Trans. Amer. Math. Soc.* 141 (1969), 1–35.
[63] C. R. Reddy and D. W. Loveland. 1978. Presburger Arithmetic with Bounded Quantifier Alternation. In *Proceedings of the Tenth Annual ACM Symposium on Theory of Computing (STOC'78)*. ACM, New York, NY, USA, 320–325.
[64] H. G. Rice. 1953. Classes of Recursively Enumerable Sets and Their Decision Problems. *Trans. Amer. Math. Soc.* 74 (1953), 358–366.
[65] SANS. http://www.sans.org/security-resources/malwarefaq/wnt-unicode.php. (????).
[66] Alexei P. Stolboushkin and Michael A. Taitslin. 1999. Finite Queries Do Not Have Effective Syntax. *Inf. Comput.* 153, 1 (1999), 99–116.
[67] Wolfgang Thomas. 1996. Languages, Automata, and Logic. In *Handbook of Formal Languages*. Vol. 3. Springer, 389–455.
[68] L. van den Dries. 1998. *Tame Topology and O-Minimal Structures*. Cambridge Univ. Press.
[69] Moshe Y. Vardi. 1996. Why is Modal Logic So Robustly Decidable?. In *Descriptive Complexity and Finite Models (DIMACS Series in Discrete Mathematics and Theoretical Computer Science)*, Neil Immerman and Phokion G. Kolaitis (Eds.), Vol. 31. American Mathematical Society, 149–184.
[70] Margus Veanes and Nikolaj Bjørner. 2012. Symbolic Automata: The Toolkit. In *TACAS 2012 (LNCS)*, C. Flanagan and B. König (Eds.), Vol. 7214. Springer.
[71] Margus Veanes, Nikolaj Bjørner, and Leonardo de Moura. 2010. Symbolic Automata Constraint Solving. In *LPAR-17 (LNCS/ARCoSS)*, C. Fermüller and A. Voronkov (Eds.), Vol. 6397. Springer, 640–654.
[72] Margus Veanes, Nikolaj Bjørner, Lev Nachmanson, and Sergey Bereg. 2014. Monadic Decomposition. In *Computer Aided Verification - 26th International Conference, CAV'14 (LNCS)*, Vol. 8559. Springer, 628–645.
[73] Margus Veanes, Pieter Hooimeijer, Benjamin Livshits, David Molnar, and Nikolaj Bjørner. 2012. Symbolic Finite State Transducers: Algorithms and Applications. In *POPL'12*. 137–150.
[74] Margus Veanes, Todd Mytkowicz, David Molnar, and Benjamin Livshits. 2015. Data-Parallel String-Manipulating Programs. *ACM SIGPLAN Notices - POPL'15* 50, 1 (2015), 139–152.
[75] Sheng Yu. 1997. Regular Languages. In *Handbook of Formal Languages*, G. Rozenberg and A. Salomaa (Eds.). Vol. 1. Springer, 41–110.

A Z3 PYTHON CODE OF *MONDEC*

The following is a python script, using the Z3 module. It implements a generalization of the abstract ***mondec*** algorithm to any number of variables. By default it runs the variant $\boldsymbol{mondec}_1$ where the search heuristic defined in Section 4.4 is turned on.

```
from z3 import *
def nu_ab(R, x, y, a, b):
    x_ = [ Const("x_%d" %i,x[i].sort()) for i in range(len(x))]
    y_ = [ Const("y_%d" %i,y[i].sort()) for i in range(len(y))]
    return Or(Exists(y_,R(x+y_)!=R(a+y_)),Exists(x_,R(x_+y)!=R(x_+b)))
def isUnsat(fml):
    s = Solver(); s.add(fml); return unsat == s.check()
def lastSat(s, m, fmls):
    if len(fmls) == 0: return m
    s.push(); s.add(fmls[0])
    if s.check() == sat: m = lastSat(s, s.model(), fmls[1:])
    s.pop(); return m

def mondec(R, variables):
    phi = R(variables);
    if len(variables)==1: return phi
    m = len(variables)/2
    x,y = variables[0:m],variables[m:]
    def dec(nu, pi):
        if isUnsat(And(pi, phi)): return BoolVal(False)
        if isUnsat(And(pi, Not(phi))): return BoolVal(True)
        fmls = [BoolVal(True)]
        if FLAG: fmls = [BoolVal(True), phi, pi] #use search heuristic
        m = lastSat(nu, None, fmls)              #try to extend nu
        assert(m != None)                        #nu must be consistent
        a = [ m.evaluate(z,True) for z in x ]
        b = [ m.evaluate(z,True) for z in y ]
        psi_ab = And(R(a+y), R(x+b))
        phi_a = mondec(lambda z: R(a+z),y)
        phi_b = mondec(lambda z: R(z+b),x)
        nu.push()
        nu.add(nu_ab(R, x, y, a, b))             #exclude: x~a and y~b
        t, f = dec(nu, And(pi, psi_ab)), dec(nu, And(pi, Not(psi_ab)))
        nu.pop()
        return If(And(phi_a, phi_b), t, f)
    return dec(Solver(),BoolVal(True))             #nu is initially true

def test_mondec(k):
    R = lambda v:And(v[1]>0,(v[1]&(v[1]-1))==0,
                     (v[0]& (v[1]%((1<<k)-1)))!=0)
    bvs = BitVecSort(2*k)                        #use 2k-bit bitvectors
    x,y = Const("x",bvs),Const("y",bvs)
    res = mondec(R,[x,y])
    assert(isUnsat(res != R([x,y])))             #check correctness
    print "mondec1(", R([x,y]), ") ="; print res
FLAG = True                                      #run as mondec1
test_mondec(2)
```

Recursive calls happen in lines 29 and 30 and terminate when the number of variables becomes 1 (line 16). The heuristic of $\boldsymbol{mondec}_1$ from Section 4.4 is triggered by the flag in line 46 and causes the call in line 24 to the `lastSat` procedure with the additional formulas in the list `fmls`. The call `lastSat(nu,None,fmls)` finds a model `m` in the context `nu` that also tries to satisfy the maximal prefix of the formulas in `fmls`.

This python script is executable and running it calls `test_mondec(2)` that prints the output

```
mondec1( And(y > 0, y & y - 1 == 0, x & y%3 != 0) ) =
If(And(And(y > 0, y & y - 1 == 0, 2 & y%3 != 0),
       And(2 > 0, 2 & 2 - 1 == 0, x & 2%3 != 0)),
   True,
   If(And(And(y > 0, y & y - 1 == 0, 5 & y%3 != 0),
          And(1 > 0, 1 & 1 - 1 == 0, x & 1%3 != 0)),
      True,
      False))
```

corresponding to the MNF $\text{IF}(\boldsymbol{\pi}^2_2, \top, (\text{IF}(\boldsymbol{\pi}^5_1, \top, \bot)))$ of the formula R^2 where R^k is defined in Example 4.4 and where $\boldsymbol{\pi}^a_b$ is the formula $R^2(a, y) \wedge R^2(x, b)$.