Monadic Decomposition

Margus Veanes[1], Nikolaj Bjørner[1], Lev Nachmanson[1], and Sergey Bereg[2]

[1] Microsoft Research
{margus,nbjorner,levnach}@microsoft.com
[2] The University of Texas at Dallas
besp@utdallas.edu

Abstract. 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 over a decidable background theory, such as arithmetic and we here develop a semi-decision procedure for extracting a monadic decomposition of a formula when it exists.

1 Introduction

Classical decidability results of fragments of logic [7] 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 [29], the Löb-Gurevich class [28], monadic second-order logic with one successor (S1S) [8], and monadic second-order logic with two successors (S2S) [35] impose at some level restrictions to *monadic* or unary predicates to achieve decidability.

Here we propose and study an orthogonal 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 an effective representation of a nonempty binary relation R, decide if R equals a* ***finite*** *union $\bigcup_{0\leq i<k} R_i$ of k Cartesian products $R_i = A_i \times B_i$, and if so, construct such R_i effectively.*

The fundamental assumption that we are making here is:

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

More precisely, we assume a background structure $\mathfrak{U}$ with an r.e. (recursively enumerable) universe $\mathcal{U}$ (so all elements $a \in \mathcal{U}$ can be named; we write a also for a term denoting a) and an r.e. set Ψ of formulas such that:

1. If $a \in \mathcal{U}$, x is a variable and $\varphi \in \Psi$ then $\varphi[x/a] \in \Psi$,
2. If $\psi, \varphi \in \Psi$ then $\psi \wedge \varphi, \psi \vee \varphi, \neg\varphi \in \Psi$.
3. Satisfiability of $\varphi(\bar{x}) \in \Psi$ (i.e., $\mathfrak{U} \models \exists\bar{x}\varphi(\bar{x})$) is decidable by the solver.

When $\varphi(\bar{x})$ is satisfiable it follows that we can also effectively generate a *witness* $\bar{a}$ such that $\varphi(\bar{a})$ holds, because $\mathcal{U}$ is r.e.. *Effectiveness* means that the solver uses a finite number of steps for deciding satisfiability and for finding a witness. An *effective representation* of a relation is given by a formula from Ψ. The above formulation is very natural from the standpoint of modern logical inference engines, because Ψ embodies the basic properties supported by any state-of-the-art *satisfiability modulo theories* (SMT) solver [12]. One observation that we can immediately make about Ψ is that it is (without loss of generality) closed under formation of tuples, i.e., we can always group variables together and view the group as a single variable. We can also note certain properties that Ψ *cannot* express. For example, Ψ cannot represent formulas $\varphi_L(x)$ that are at least as expressive as deterministic context free languages L. Otherwise construct φ_L such that $w \in L$ iff $\varphi_L(w)$ holds; then $\varphi_L(x) \wedge \varphi_{L'}(x)$ is satisfiable if and only if $L \cap L' \neq \emptyset$, but that is an undecidable problem [22].

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 that we can define the following *equivalence* relation over $A = \{a \mid \exists b\, R(a, b)\}$,

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

Moreover, we can *decide* if $a \sim a'$ because $a \not\sim a'$ has the equivalent form $\exists y\, y'\, \psi(y, y')$ for some $\psi(y, y') \in \Psi$. This gives us a systematic way of how to subdivide A into equivalence classes $A_{\sim}$, namely by using the solver for Ψ to enumerate enough witnesses that cover $A_{\sim}$. The main technical lemma is that there are finitely many such witnesses if and only if R is monadic. The question of *deciding* if R is monadic is not completely settled here. We show that the problem is decidable for integer linear arithmetic and real algebraic polynomial arithmetic but the general case is an open problem.

As the main strength of this approach we see its *simplicity* combined with its *generality*. For monadic decomposition to work, there are no assumptions on Ψ other than the ones listed above. The technique works in all theories where a *solver* is available, such as *linear arithmetic*, *bit-vectors*, *arrays*, *uninterpreted function symbols*, *algebraic data types*, *algebraic reals*, as well as combinations thereof. The technique provides a general simplification principle, tantamount to a semantic normal form. It can be used in many different contexts where it is useful to simplify formulas by eliminating variable dependencies, such as *program analysis*, *optimization*, *theorem proving*, and *compiler optimization*. It also provides a new way how to investigate new decidability results.

Rest of the paper: § 2 describes the motivation. In § 3 and § 4 the problem is defined formally, we prove the main decomposition Theorem 1, correctness of the main algorithm, Theorem 2, and we prove some decidable cases, Theorems 3 and 4. § 5 provides some evaluation. § 6 is related work. § 7 concludes.

$\lambda x.(0 \leq x \leq \mathtt{7F}_{16})/[x]$

$\lambda x.(\mathtt{7F}_{16} < x \leq \mathtt{7FF}_{16})/[6 \cdot x_{\langle 10,6\rangle},\ 2 \cdot x_{\langle 5,0\rangle}]$

$\lambda x.(\mathtt{7FF}_{16} < x \leq \mathtt{FFFF}_{16} \wedge \neg \mathit{Surrogate}(x))/[14 \cdot x_{\langle 15,12\rangle}, 2 \cdot x_{\langle 11,6\rangle},\ 2 \cdot x_{\langle 5,0\rangle}]$

$\lambda x.(\mathtt{FFFF}_{16} < x \leq \mathtt{10FFFF}_{16})/[30 \cdot x_{\langle 20,18\rangle}, 2 \cdot x_{\langle 17,12\rangle}, 2 \cdot x_{\langle 11,6\rangle},\ 2 \cdot x_{\langle 5,0\rangle}]$

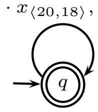


Fig. 1. SFT *EncUTF8*: UTF8 encoder for valid Unicode code points; $x_{\langle h,l\rangle}$ extracts bits from h to l from x, e.g., $8_{\langle 3,2\rangle} = 2$; *Surrogate* $\stackrel{\text{def}}{=} \lambda x.\mathtt{D800}_{16} \leq x \leq \mathtt{DFFF}_{16}$, surrogates are not valid code points; $x \cdot y$ denotes bit-append, e.g., $6 \cdot x_{\langle 10,6\rangle} = \mathtt{C0}_{16} + x_{\langle 10,6\rangle}$.

2 Motivation

We start by describing the concrete application that originally lead us to investigate monadic decomposition. We then list other potential applications.

Symbolic Automata and Transducers. In the context of web security, it is important to understand and analyze various properties of *sanitizers* [38]. Sanitizers are special purpose string encoders that escape or remove potentially dangerous strings in order to prevent *cross site scripting* (XSS) attacks. *Bek* is a programming language that is specifically designed for this purpose [20] and builds on the theory and algorithms of *Symbolic Finite Transducers* or *SFTs* [44]. Monadic decomposition is a useful technique for enabling many analyses involving SFTs. One such case is to decide if the range of an SFT is regular and, if so, to construct the corresponding symbolic automaton or SFA. Unlike in the classical case [32, 45], a *range automaton* of an SFT is not always regular but accepted by an *Extended* SFA or *ESFA* (SFA with bounded lookahead over the input) and intersection emptiness of ESFAs is undecidable [9]. Transforming an ESFA into and SFA, when possible, requires monadic decomposition.

Figure 1 illustrates an SFT *EncUTF8* that performs UTF8 encoding that is also used by some sanitizers [1] as the first encoding step. The input to *EncUTF8* is a sequence of Unicode code points, that are integers ranging from 0 to $\mathtt{10FFFF}_{16}$, and the output is a sequence of bytes. Each of the four transitions of *EncUTF8* corresponds to the number of bytes needed in the encoding of the code point.[3] For example *EncUTF8*$([\mathtt{1F60A}_{16}]) = [\mathtt{F0}_{16}, \mathtt{9F}_{16}, \mathtt{98}_{16}, \mathtt{8A}_{16}]$, where $\mathtt{1F60A}_{16}$ is the code point of the ☺ emoticon [40].

For example, the second rule of *EncUTF8* becomes the following transition of the range ESFA and has lookahead 2, i.e., it reads 2 bytes at a time

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

The existential quantifier over x can be eliminated automatically by using any known quantifier elimination technique for integer linear arithmetic [31]. 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

[3] The corresponding encoder in [10, Figure 3] uses 5 states and 11 transitions because there the input is assumed to be UTF16 encoded.

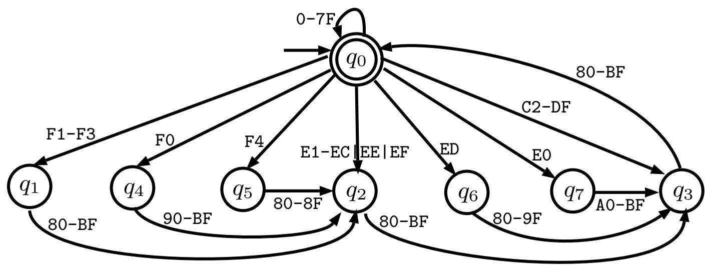


Fig. 2. Minimal symbolic automaton that recognizes valid UTF8 encoded strings.

equivalent transition $q \xrightarrow{\lambda(y,z).\mathtt{7F}_{16}<(y_{\langle 4,0\rangle}\cdot z_{\langle 5,0\rangle})\leq\mathtt{7FF}_{16}\wedge y=6\cdot y_{\langle 4,0\rangle}\wedge z=2\cdot z_{\langle 5,0\rangle}} q$. Next, *monadic decomposition* of the guard yields the following equivalent transition, $q \xrightarrow{\lambda(y,z).y_{\langle 5,1\rangle}\neq 0\wedge y_{\langle 5,5\rangle}=0\wedge y=6\cdot y_{\langle 4,0\rangle}\wedge z=2\cdot z_{\langle 5,0\rangle}} q$ that, after simplification, is equivalent to the following two transition path $q \xrightarrow{\lambda y.\mathtt{C2}_{16}\leq y\leq\mathtt{DF}_{16}} q_3 \xrightarrow{\lambda z.\mathtt{80}_{16}\leq z\leq\mathtt{BF}_{16}} q$ where q_3 is a new state. The ESFA rules with lookahead 3 and 4 are a bit more challenging and yield monadic decompositions with higher widths. After further *minimization* [11] of the resulting SFA we obtain the SFA in Figure 2 that accepts the range of *EncUTF8*.

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 [30]. 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.

Program synthesis. The range SFA construction of *EncUTF8* illustrates another potential usage. We can *automatically* invert *EncUTF8* into a *UTF8 decoder DecUTF8* in a way that guarantees the correctness criterion that for all valid input sequences s, $\mathit{DecUTF8}(\mathit{EncUTF8}(s)) = s$, by using the SFA in Figure 2 as the control-flow graph of the corresponding transducer and by inverting the individual rules of the encoder.

Linear optimization. A new SMT based optimization algorithm SYMBA is described in [27] that uses linear real arithmetic objective functions and an SMT solver as a black box. Monadic decomposition is a potential simplification technique of objective functions in this context [4].

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 [26]. The use of SMT solvers in this context comes into play when the classical resolution technique is extended to work modulo background theories [24, 25].

Compiler technology. Monadic decomposition can be used to simplify expressions and thus enable new (or enhance existing) automatic compiler optimization techniques [3]. Moreover, it may be used for code parallelization.

3 Monadic predicates

We assume a decidable background $\mathfrak{U}$ as described above. The Boolean type is BOOL with truth values $\{\top, \bot\}$. In our expressions, all variables are typed and all terms and formulas are well-typed. The subuniverse of elements of type τ is denoted by $\mathcal{U}^\tau$. 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 types of x_i are clear from the context, for the corresponding predicate and $[\![\varphi]\!]$ for the n-ary relation defined by φ.

Let R be an n-ary relation for some $n \geq 2$ and of type $\prod_{i=1}^{n} \tau_i$.[4] R is *Cartesian* if there exist sets $U_i \subseteq \mathcal{U}^{\tau_i}$, for $1 \leq i \leq n$, such that $R = \prod_{i=1}^{n} U_i$. R is *monadic* if there exists finite $k > 0$ and Cartesian R_i, for $1 \leq i \leq k$, s.t. $R = \bigcup_{i=1}^{k} R_i$; $\{R_i\}_{i=1}^{k}$ is called a *monadic decomposition of R of width k*. R is *k-monadic* if R has a monadic decomposition of width k. The *(monadic) width* of R is the smallest k such that R is *k-monadic*. Note that R has width 1 iff it is Cartesian.

Example 1. Let φ be the predicate $\lambda(x,y).(x + (y \mod 2)) > 5$, where x and y have integer type. Then $R = [\![\varphi]\!]$ is the corresponding binary relation over integers. R is not Cartesian but it is 2-monadic because $R = ([\![\lambda x.x > 5]\!] \times [\![\lambda y.\top]\!]) \cup ([\![\lambda x.x > 4]\!] \times [\![\lambda y.odd(y)]\!])$. ⊠

We lift the notions to predicates. A *unary* formula is a formula with at most one free variable. An *explicitly monadic* formula is some Boolean combination of unary formulas. Observe that the difference between monadic and explicitly monadic, is that the first notion is semantic (depends on $\mathfrak{U}$) while the second is syntactic (independent of $\mathfrak{U}$).

4 Monadic decomposition

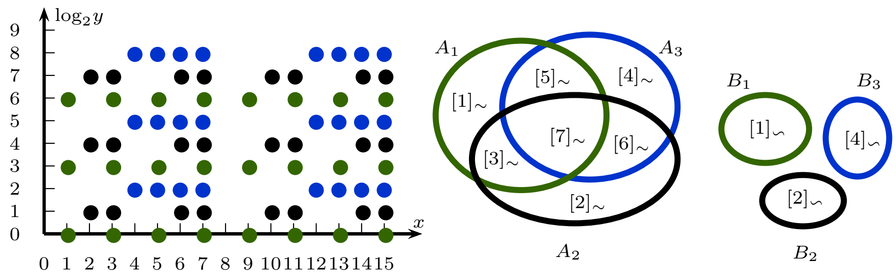
We are interested in the following two problems: 1) Deciding if a predicate φ is monadic; 2) Given a monadic predicate φ, effectively constructing a monadic decomposition of φ. We restrict our attention to *binary* predicates. The decomposition can be reduced recursively to the binary case and applied to n-ary predicates with $n > 2$, such as the range predicates arising from the third and fourth rules of *EncUTF8* in Figure 1.

4.1 Deciding if a predicate is monadic

Consider any term $f(x)$ in the background theory denoting a function over integers. Let $\varphi_f(x,y)$ be the formula $f(x) \doteq y$.[5] Then $\varphi_f(x,y)$ is monadic iff there exists k such that $\varphi_f(x,y)$ is equivalent to $\bigvee_{i<k} \alpha_i(x) \wedge \beta_i(y)$. Since there can only be one y for a given x (because f is a function) it follows that $|[\![\beta_i]\!]| = 1$ for

[4] Type $\prod_{i=1}^{2} \tau_i$ is also denoted $\tau_1 \times \tau_2$.

[5] We assume that formal equality $\doteq$ is allowed.



(a) (b)

Fig. 3. Let $R^k(x,y) \stackrel{\text{def}}{=} y{>}0 \wedge y\&(y{-}1){=}0 \wedge x\&(y \bmod (2^k-1)){\neq}0$ over $\mathbb{N}$, & is bit-wise-AND; a) *Geometrical view of* R^3: (x,y) is marked iff $R^3(x,y)$ holds; if two Y-cuts Y_m and Y_n are identical then $m \sim n$, e.g., $1 \sim 9$; if two X-cuts X_m and X_n are identical then $m \backsim n$, e.g., $2^2 \backsim 2^5$; b) *Venn Diagram view of* R^3: $R^3 = \bigcup_{i=1}^{3} A_i \times B_i$.

all $i < k$. So φ_f is monadic iff f is bounded (finite-valued). While boundedness of f is an undecidable problem in general by using Rice's Theorem [37], we cannot use this argument because we cannot even encode context free languages in Ψ, so much less arbitrary recursive languages. We show in Section 4.4 that the question is decidable for some cases, but the general case is an open problem.

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

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

Define the relations:

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

For $a \in A$, define the *Y-cut of R by a* as the set $Y_a = \{b \mid R(a,b)\}$. Similarly, for $b \in B$, define the *X-cut of R by b* as the set $X_b = \{a \mid R(a,b)\}$. The idea of cuts can be illustrated geometrically. See Figure 3(a). The following properties are used below.

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

Lemma 1 holds obviously 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 the main lemma.

Lemma 2. *R is monadic $\Leftrightarrow$ the number of $\sim$-equivalence classes is finite.*

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 $b \in Y_{a_i}$. But $Y_{a_i} = Y_a$ by Lemma 1 because $a_i \sim a$, so $b \in Y_a$, 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 $I \subseteq \{i \mid 0 \le i < n\}$ 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 $a \in A$, Y_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 $2^n - 1$ where n is the monadic width of R, because the number m of different (nonempty) minterms M_I is, due to the powerset construction, at most $2^n - 1$.

$\Leftarrow$: 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 $B_i = Y_{a_i}$ for $0 \le i < n$. Thus if $(a,b) \in A_i \times B_i$ then $a \sim a_i$ and $b \in Y_{a_i}$, i.e., $Y_a = Y_{a_i}$ and $b \in Y_{a_i}$. So $b \in Y_a$, i.e., $(a,b) \in R$. Conversely, if $(a,b) \in R$ then $b \in Y_a$. But $Y_a = Y_{a_i} = B_i$, 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 a simple iterative procedure to compute a *witness set* W_A that covers $A_\sim$. We use the negated form of $\sim$:

$$x \not\sim x' \Leftrightarrow \exists y\, y' (\varphi(x,y) \wedge \varphi(x',y') \wedge (\neg\varphi(x',y) \vee \neg\varphi(x,y')))$$

So, for all $a, a' \in A$, $a \not\sim a'$ 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 the formula $\bigwedge_{a \in W_A} x \not\sim a$.
2. If there exists a such that $\psi(a)$ holds then $W_A := W_A \cup \{a\}$ else terminate.

Observe that satisfiability checking of ψ in the above procedure as well as generating the witness a is decidable because we can transform ψ to prenex normal form as an $\exists$-formula and treat all the existential variables as free variables.

In other words, the resulting formula is in Ψ. 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 2, 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 *witness set* W_B is analogous to computation of W_A. Observe that $|W_B|, |W_A| < 2^n$ where n is the monadic width of φ, which follows from the proof of Lemma 2. We also have that $n \leq |W_B|, |W_A|$.

Example 2. Consider the relation $R = R^3$ in Figure 3. The width of R is 3. We have $A_\sim = \{[a]_\sim \mid 1 \leq a \leq 7\}$ where $[a]_\sim = \{n \mid n_{\langle 2,0\rangle} = a\}$ and $B_\backsim = \{[2^0]_\backsim, [2^1]_\backsim, [2^2]_\backsim\}$ were $[2^m]_\backsim = \{2^n \mid n \bmod 3 = m\}$. Figure 3(b) illustrates the equivalence classes as nonempty regions of a Venn Diagram view of R. ⊠

Lemma 3. *If R is monadic then, for all $\mathbf{a} \in A_\sim$ and $\mathbf{b} \in B_\backsim$, we can effectively construct $\alpha_\mathbf{a}, \beta_\mathbf{b} \in \Psi$ such that $[\![\alpha_\mathbf{a}]\!] = \mathbf{a}$ and $[\![\beta_\mathbf{b}]\!] = \mathbf{b}$.*

Proof. By using Lemma 2 let W_A be constructed as above, so $A_\sim = \{[a]_\sim \mid a \in W_A\}$. Similarly to W_A, construct a finite W_B s.t. $B_\backsim = \{[b]_\backsim \mid b \in W_B\}$. Let

$$\text{(for } b \in W_B) \quad \beta_b(y) \stackrel{\text{def}}{=} \beta_{[b]_\backsim}(y) \stackrel{\text{def}}{=} \Big(\bigwedge_{a \in W_A \cap X_b} \varphi(a, y)\Big) \wedge \Big(\bigwedge_{a \in W_A \setminus X_b} \neg\varphi(a, y)\Big)$$

$$\text{(for } a \in W_A) \quad \alpha_a(x) \stackrel{\text{def}}{=} \alpha_{[a]_\sim}(x) \stackrel{\text{def}}{=} \Big(\bigwedge_{b \in W_B \cap Y_a} \varphi(x, b)\Big) \wedge \Big(\bigwedge_{b \in W_B \setminus Y_a} \neg\varphi(x, b)\Big)$$

Observe that α_a is well-defined because for all $a' \in [a]_\sim$ we have that $Y_a = Y_{a'}$. Similarly for β_b. One can show that $[\![\beta_b]\!] = [b]_\backsim$ and $[\![\alpha_a]\!] = [a]_\sim$. Fix $a \in W_A$ and consider the definition of α_a. Suppose $W_B \cap Y_a = \{b_1, b_2\}$ and $W_B \setminus Y_a = \{b_3, b_4\}$. Then $[a]_\sim \subseteq X_{b_1} \cap X_{b_2}$ and $[a]_\sim \subseteq (X_{b_3} \cup X_{b_4})^c$. So $[a]_\sim \subseteq [\![\alpha_a]\!]$. For the direction $[\![\alpha_a]\!] \subseteq [a]_\sim$ take $a' \in [\![\alpha_a]\!]$. Suppose, by way of contradiction, that, $a \not\sim a'$ and thus $Y_{a'} \neq Y_a$. Then there exists $b \in W_B \setminus Y_a$ such that $a' \in X_b$. But, by definition of α_a, $X_b \cap [\![\alpha_a]\!] = \emptyset$, which contradicts that $a' \in X_b$ and $a' \in [\![\alpha_a]\!]$. ⊠

Lemma 3 is essentially a quantifier elimination property that allows us to elimiate the $\forall$ quantifier from the definition of $\lambda x.x \sim a$ (resp. $\lambda y.y \backsim b$) 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.

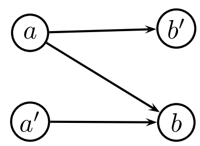
Theorem 1. *If $\varphi(x, y)$ is monadic then*
a) $\varphi(x, y)$ is equivalent to $\lambda(x, y). \bigvee_{a \in W_A} (\alpha_a(x) \wedge \varphi(a, y))$.
b) $\varphi(x, y)$ is equivalent to $\lambda(x, y). \bigvee_{b \in W_B} (\beta_b(y) \wedge \varphi(x, b))$.
c) $\varphi(x, y)$ is equivalent to $\lambda(x, y). \bigvee_{a \in W_A, b \in W_B, (a,b) \in [\![\varphi]\!]} (\alpha_a(x) \wedge \beta_b(y))$.

Proof. We prove (a). The other cases are similar. By Lemma 3 we have $[\![\alpha_a]\!] = [a]_\sim$. By construction of W_A we have that, for all $a \in W_A$ we have $[a]_\sim \times Y_a \subseteq [\![\varphi]\!]$ where $[a]_\sim \times Y_a = [\![\lambda(x, y).\alpha_a(x) \wedge \varphi(a, y)]\!]$. In the other direction, if $(a, b) \in [\![\varphi]\!]$ then $a \in [\![\alpha_a]\!]$ and $b \in Y_a$. In other words, $(a, b) \in [\![\lambda(x, y).\alpha_a(x) \wedge \varphi(a, y)]\!]$. ⊠

Theorem 1 does not guarantee smallest monadic width. Example 3 shows that the monadic width may be strictly smaller than $\min(|W_B|, |W_A|)$.

Example 3. 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. 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 3 and simplifications.

$$\alpha_a(x) \stackrel{\text{def}}{=} x \doteq a, \quad \alpha_{a'}(x) \stackrel{\text{def}}{=} x \doteq a', \quad \beta_b(y) \stackrel{\text{def}}{=} y \doteq b, \quad \beta_{b'}(y) \stackrel{\text{def}}{=} y \doteq b'$$

where $a = 0, a' = 1, b = 0, b' = 1$. Monadic decomposition of ϕ reconstructs the formula $\alpha_a(x)\wedge\beta_b(y) \vee \alpha_a(x)\wedge\beta_{b'}(y) \vee \alpha_{a'}(x)\wedge\beta_b(y)$ by using Theorem 1(c). Case $\alpha_{a'}(x) \wedge \beta_{b'}(y)$ is not included because $\phi(1,1)$ is false. ⊠

4.3 Another decomposition algorithm

If implemented directly, Theorem 1 suggests creating a decomposition which is in a disjunctive normal form (DNF) with respect to the unary sub-formulas. Instead of creating what amounts to a DNF, we can use case analysis on $\varphi(a,y)\wedge\varphi(x,b)$ for all $([a]_\sim,[b]_\backsim) \in A_\sim \times B_\backsim$. The output may be any explicitly monadic formula, not necessarily in DNF. Moreover, Theorem 1 suggests full exploration of W_A and W_B. We show how to avoid this by using lifted versions of the definitions of $\sim$ and $\backsim$. We lift the definitions of $\sim$ (resp. $\backsim$) to all elements of the type of x (resp. y). We define $a_1 \sim a_2 \stackrel{\text{def}}{=} Y_{a_1} = Y_{a_2}$ and $b_1 \backsim b_2 \stackrel{\text{def}}{=} X_{b_1} = X_{b_2}$. This is consistent with the earlier definition (due to Lemma 1) and is simpler to work with because the equivalence classes cover the full universe (of the given type) and are identical for φ and $\neg\varphi$. For example, consider the equivalence classes $\mathbb{N}_\sim$ in Figure 3. Then $[0]_\sim = \mathbb{N} \setminus (A_1 \cup A_2 \cup A_3)$. Thus

$$x \not\sim x' \Leftrightarrow \exists z(\neg(\varphi(x,z) \Leftrightarrow \varphi(x',z))), \quad y \not\backsim y' \Leftrightarrow \exists z(\neg(\varphi(z,y) \Leftrightarrow \varphi(z,y'))).$$

We introduce a procedure named ***mondec*** that given a monadic predicate $\varphi(x,y)$ produces an equivalent explicitly monadic predicate ***mondec***(φ); it uses a recursive procedure $\boldsymbol{\delta}$. The argument π of $\boldsymbol{\delta}$ below is the path condition and ν is the the accumulated side condition; the purpose of ν is to ensure *new* combinations from $A_\sim \times B_\backsim$. Here A (resp. B) is the set of all values of the type of x (resp. y). We write $(\psi\,?\,\phi_t : \phi_f)$ for $((\psi \wedge \phi_t) \vee (\neg\psi \wedge \phi_f))$.

$$
\begin{aligned}
\boldsymbol{mondec}(\varphi) &\stackrel{\text{def}}{=} \boldsymbol{\delta}(\top,\top), \quad \text{where} \\
\boldsymbol{\delta}(\nu,\pi) &\stackrel{\text{def}}{=} \begin{cases} \bot, & \text{if } \boldsymbol{unsat}(\pi \wedge \varphi); \\ \top, & \text{else if } \boldsymbol{unsat}(\pi \wedge \neg\varphi); \\ (\psi^a_b\,?\,\boldsymbol{\delta}(\nu \wedge \nu^a_b, \pi \wedge \psi^a_b) : \boldsymbol{\delta}(\nu \wedge \nu^a_b, \pi \wedge \neg\psi^a_b)), & \text{else let } (a,b) \models \nu, \end{cases} \\
\nu^a_b &\stackrel{\text{def}}{=} a \not\sim x \vee y \not\backsim b, \\
\psi^a_b &\stackrel{\text{def}}{=} \varphi(a,y) \wedge \varphi(x,b).
\end{aligned}
$$

Theorem 2. *If φ is monadic then $\boldsymbol{mondec}(\varphi)$ is defined and $\boldsymbol{mondec}(\varphi)$ is an explicitly monadic predicate that is equivalent to φ.*

Proof. Assume φ is monadic. Assume also that φ is satisfiable or else it is trivially equivalent to the explicitly monadic predicate $\bot$. Let A and B be as above. By using Lemma 2, $A_\sim$ and $B_\backsim$ are finite. Observe that the argument ν of $\boldsymbol{\delta}$ remains of the form that all existential quantifiers occur *positively* in it, so the selection of $(a,b) \models \nu$ in $\boldsymbol{\delta}$ is decidable (using the solver for Ψ).

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

First, we show that $\boldsymbol{mondec}(\varphi)$ is well-defined (terminates) by showing that there are finitely many nodes. A new node ψ^a_b 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 ψ^a_b is eliminated by the constraint ν^a_b. Termination follows because $A_\sim$ and $B_\backsim$ are finite and $\psi^a_b \Leftrightarrow \psi^{a'}_{b'}$ 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 $a \not\sim a'$ or $b \not\backsim b'$. Moreover, and without loss of generality, π_1 is of the form $\pi \wedge \psi$ where ψ is a conjunction of predicates ψ^c_d or $\neg\psi^c_d$ for some $c \in A$ and $d \in B$. We have, by definition of $\boldsymbol{\delta}$, that π has the form

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

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 ψ^c_d such that $(c,d) \models \neg\nu$ is useless because it makes ψ^c_d equivalent to some $\psi^{a_i}_{b_i}$ for some i, $1 \leq i \leq n$, and so $\pi \wedge \psi^c_d$ or $\pi \wedge \neg\psi^c_d$ is either equivalent to π or to $\bot$. Therefore, ν must be satisfiable or else π_1 cannot be constructed.

To show that $\boldsymbol{mondec}(\varphi) \Leftrightarrow \varphi$ is immediate from the definition of $\boldsymbol{\delta}$. 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 $\bot$. ⊠

To illustrate ***mondec***, take $\varphi(x,y)$ to be the predicate R^3 in Figure 3. Consider the result of $\boldsymbol{mondec}(\varphi)$ that starts with $(4,4) \models \varphi$ so the root is ψ_4^4. In the depiction of $\boldsymbol{mondec}(\varphi)$ in Figure 4, the left subtree of a node is the true case and right subtree of a node is the false case. For example, $\neg\psi_4^4 \wedge \psi_2^3 \wedge \psi_2^2$ is a branch that implies φ, this branch covers the case $A_2 \times B_2$ in Figure 3(b).

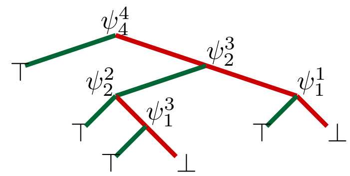


Fig. 4. $\boldsymbol{mondec}(R^3)$.

4.4 Two decidable cases

We show decidability of monadic decomposition in two cases. We leave decidabilty of monadicity for other theories and tight complexity bounds as open problems.

Consider first integer linear arithmetic. It clearly meets the requirements of $\mathfrak{U}$. Take a linear arithmetic formula $\varphi(x,y)$. 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: $\mathit{IsMonadic}(\varphi) \stackrel{\text{def}}{=} \exists \hat{x}(\forall x(x \in A \Rightarrow \exists x'(|x'| < \hat{x} \wedge x \sim x')))$

Theorem 3. *Monadic decomposition is decidable for integer linear arithmetic.*

Proof. Let $\varphi(x,y)$ be a formula in integer linear arithmetic. We show that φ is monadic $\Leftrightarrow$ $\mathit{IsMonadic}(\varphi)$ is true in Presburger arithmetic. Decidability follows by [34]. Proof of $\Rightarrow$: Assume φ is monadic. Then $A_\sim$ is finite by Lemma 2. Let $\hat{a} = \max\{\min(\mathit{abs}(C)) \mid C \in A_\sim\} + 1$. Then, for all $a \in A$, a belongs to some C in $A_\sim$, and so there is $a' \in C$ such that $|a'| = \min(\mathit{abs}(C))$ and so $|a'| < \hat{a}$ and $a \sim a'$. Proof of $\Leftarrow$: Assume $\mathit{IsMonadic}(\varphi)$ holds. Choose a witness $\hat{a}$ for $\hat{x}$ and consider the classes $\mathcal{A} = \{[a]_\sim \mid 0 \le |a| < \hat{a}\}$. It follows that $\mathcal{A} = A_\sim$ is finite, so φ is monadic by Lemma 2. ⊠

The formula $\mathit{IsMonadic}(\varphi)$ has the quantifier prefix $\exists\forall\exists\forall$ in Prenex normal form when φ is quantifier free. So there are *three* quantifier alternations in $\mathit{IsMonadic}(\varphi)$. 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 [36]. This is one exponent lower than the upper bound $2^{2^{2^{cn}}}$ known for the full Presburger arithmetic [14]. 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:

Theorem 4. *Monadic decomposition is decidable for real algebraic arithmetic with addition and multiplication.*

Proof (Sketch). The atomic subformulas of φ are of the form $p(x,y) \ge 0$, 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 $\epsilon \ge b' > b$ we have $\varphi(a,b)$ but $\neg\varphi(a,b')$. Then φ contains an atomic formula $p(x,y) \ge 0$ whose truth value changes over

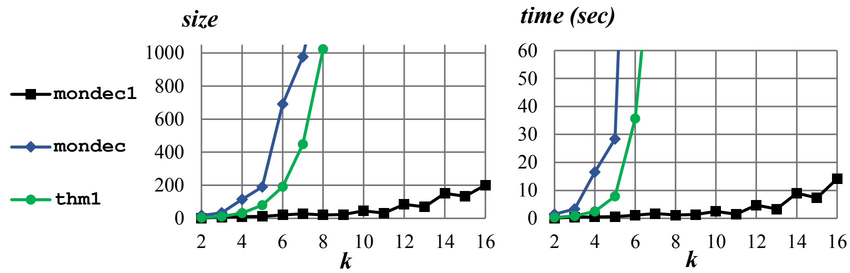


Fig. 5. Comparison of monadic decomposition algorithms.

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 Experiments

We present here a set of micro benchmarks using the sample predicate R^k from Figure 3 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 1(c), is $O(k2^k)$ because $|A_\sim| = 2^k$ and $|B_\backsim| = k$ (including the classes $[0]_\sim$ and $[0]_\backsim$). We compare three algorithms, implemented as z3 python scripts, that are indicated in Figure 5 by `thm1`, `mondec`, and `mondec1`. The output is in all cases an explicitly monadic formula in form of an if-then-else expression, its *size* is the number of ψ^a_b nodes in it, e.g., the size of the expression in Figure 4 is 5.[6] Algorithm `thm1` is based on Theorem 1 but avoids explicit DNF construction. Algorithm `mondec1` is a variant of `mondec`; its python script is shown in Appendix A. The only difference compared to `mondec` is that `mondec1` uses the following heuristic for selecting a witness $(a, b) \models \nu$:

$$(a, b) \models \text{if } \boldsymbol{sat}(\nu \wedge \varphi \wedge \pi) \text{ then } \nu \wedge \varphi \wedge \pi \text{ else if } \boldsymbol{sat}(\nu \wedge \varphi) \text{ then } \nu \wedge \varphi \text{ else } \nu$$

that amounts to changing a single line of code in the python script. In other words, for selecting new (a, b) first try to do so in the context of φ and π. The most interesting aspect about this experiment is that it shows that different heuristics can influence the performance characteristics of monadic decomposition by an exponential factor. The above heuristic reduces the size of the decomposition exponentially in this experiment, while constructing nodes in `mondec` based solely on ν provides worse performance than exhaustive search of W_A and W_B, as in `thm1`. For example, the time to decompose R^9 with `mondec` gave an ouput of size 2281 and took around 11 minutes, while with `mondec1` the output

[6] The experiments were carried out on a laptop with a 2GHz CPU.

size was 23 and the decomposition took 1.4 seconds. For the formulas arising in Section 2, all algorithms terminate in a fraction of a second. Appendix A shows the python script of `mondec` (and `mondec1`) generalized to arbitrary arities.

6 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 [39]. 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 [13, 18, 6] could be relevant in this context.

Monadic fragments. Unary relations play a key role in many decision problems and decidable logics. *Monadic first-order logic*, or the Löwenheim class [29], is the classical example of a decidable fragment of first-order logic where all symbols are unary relation symbols. The Löb-Gurevich class [28], is the extension of the Löwenheim class where also unary function symbols are allowed. Both classes are decidable by having the *finite model property* [7]. *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 [8] and decidability of the monadic second-order theory *S2S* of the binary tree with two successor relations by Rabin [35]. 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 [41] their decidability to this. Grädel discusses this topic further in [17] and its relation to the *guarded fragment* [5]. Unlike in modal logics, simple extensions of the guarded fragment cause undecidability [16], one exception is the *monadic* guarded fragment with two variables and equivalence relations that does have the tree model property [15]. The theorems of Büchi and Rabin have also been revisited and extended by Gurevich through game based techniques [18]. Another technique discussed in [18] is the use of the *Feferman-Vaught* generalized products [13] as a model-theoretic method for establishing decidability results in the context of monadic second-order logic.

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* [33]. Symbolic finite transducers were introduced in [44], a different notion of symbolic transducers is also studied in [33]. The extension from SFTs to ESFTs is introduced in [10]. Equivalence of ESFTs, properties of ESFAs, and the notion of Cartesian ESFTs are studied in [9]. The monadic decomposition problem first surfaced in the context of trying to lift algorithms for symbolic automata *without lookahead* to symbolic automata *with lookahead*. In classical automata theory this problem does not exist because 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 Secion 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 [43] in the context of SMT solvers, and more algorithms are discussed in [21] in the more specialized context of string analysis. A symbolic automata toolkit is described in [42]. SFT algorithms, in particular equivalence checking, are studied in [44] and their use for web security is discussed in [20]. A new minimization algorithm of SFAs was recently presented in [11], showing that the new algorithm can enable some analysis scenarios involving monadic second-order logic that did not scale with earlier techniques; the reduction itself from monadic second-order formulas to SFAs is essentially the classical one [39] and the performance is compared to Mona [19, 23].

7 Conclusion

We introduced the problem of monadic decomposition of predicates in decidable theories. Theorem 1 provided an effective means to computing a monadic decomposition and we described an implementation with correctness proof, Theorem 2, that avoids expanding solutions directly into DNF; it leverages a Shannon decomposition. We left the general case of *decidability* of monadic decomposition as an open problem. Deciding if a predicate is monadic in a specific background theory is another interesting open problem. While we show that the problem is decidable for integer linear arithmetic and polynomial real algebraic arithmetic, we have not investigated concrete algorithms for these cases.

Acknowledgements

We thank the anonymous reviewers for their constructive feedback that greatly helped to improve the paper.

A Monadic decomposition in python

Below is a self-contained python script `mondec` that computes a monadic decomposition of a predicate R with given variables. It uses z3.

```
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 d(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 the heuristic from Section 5
        m = lastSat(nu, None, fmls)              #---- try to extend nu with fmls
        assert(m != None)                        #---- nu must be consistent
        a,b = [ m.evaluate(z,True) for z in x ],[ m.evaluate(z,True) for z in y ]
        psi_ab = And(R(a+y), R(x+b))
        phi_a, phi_b = mondec(lambda z: R(a+z),y), mondec(lambda z: R(z+b),x)
        nu.push()
        nu.add(nu_ab(R, x, y, a, b))             #---- extend nu to exlude case: x~a and y~b
        t, f = d(nu, And(pi, psi_ab)), d(nu, And(pi, Not(psi_ab)))
        nu.pop()
        return If(And(phi_a, phi_b), t, f)
    return d(Solver(),BoolVal(True))             #---- nu is initially a fresh z3 solver

def test_mondec(k):                              #---- decompose R^k from Figure 3
    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 of decomposition
    print "mondec1(", R([x,y]), ") ="; print res
FLAG = True                                      #---- run as mondec1
test_mondec(2)                                   #---- decompose R^2
```

Running it produces the following decomposition of R^2 where R^k is defined in Figure 3. The output corresponds to the expression $(\psi_2^2 \,?\, \top : (\psi_1^5 \,?\, \top : \bot))$ where ψ_b^a is the formula $R^2(a,y) \wedge R^2(x,b)$. The script can be run online using Z3Py [2].

```
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))
```

References

1. *URL Encode and Decode Tool*. http://www.url-encode-decode.com.
2. *Z3 Python*. http://rise4fun.com/Z3Py.
3. A. V. Aho, M. S. Lam, R. Sethi, and J. D. Ullman. *Compilers: Principles, Techniques, and Tools (2nd Edition)*. Addison Wesley, 2nd edition, 2006.
4. A. Albarghouthi and A. Gurfinkel. Personal communication, January 2014.
5. H. Andréka, I. Németi, and J. van Benthem. Modal languages and bounded fragments of predicate logic. *Journal of Philosophical Logic*, 27:217–274, 1998.
6. A. Bés. An application of the Feferman-Vaught theorem to automata and logics for words over an infinite alphabet. *Logical Methods in Computer Science*, 4:1–23, 2008.
7. E. Börger, E. Grädel, and Y. Gurevich. *The Classical Decision Problem*. Springer, 1997.
8. J. R. Büchi. On a decision method in restricted second order arithmetic. In E. Nagel, P. Suppes, and A. Tarski, editors, *Logic, Methodology and Philosophy of Science (Proc. 1960 Internat. Congr .)*, pages 1–11. Stanford Univ. Press, 1962.
9. L. D'Antoni and M. Veanes. Equivalence of extended symbolic finite transducers. In N. Sharygina and H. Veith, editors, *CAV 2013*, volume 8044 of *LNCS*, pages 624–639. Springer, 2013.
10. L. D'Antoni and M. Veanes. Static analysis of string encoders and decoders. In R. Giacobazzi, J. Berdine, and I. Mastroeni, editors, *VMCAI 2013*, volume 7737 of *LNCS*, pages 209–228. Springer, 2013.
11. L. D'Antoni and M. Veanes. Minimization of symbolic automata. In *POPL'14*, pages 541–553. ACM, 2014.
12. L. De Moura and N. Bjørner. Satisfiability modulo theories: introduction and applications. *Commun. ACM*, 54(9):69–77, 2011.
13. S. Feferman and R. L. Vaught. The first-order properties of algebraic systems. *Fundamenta Mathematicae*, 47:57–103, 1959.
14. M. J. Fischer and M. O. Rabin. 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*, pages 27–41, 1974.
15. H. Ganzinger, C. Meyer, and M. Veanes. The two-variable guarded fragment with transitive relations. In *LICS'99*, pages 24–34. IEEE, 1999.
16. E. Grädel. On the restraining power of guards. *Journal of Symbolic Logic*, 64:1719–1742, 1998.
17. E. Grädel. Why are modal logics so robustly decidable? *Bulletin EATCS*, 68:90–103, 1999.
18. Y. Gurevich. Monadic second-order theories. In J. Barwise and S. Feferman, editors, *Model-Theoretical Logics*, chapter XIII, pages 479–506. Springer, 1985.
19. J. Henriksen, J. Jensen, M. Jørgensen, N. Klarlund, B. Paige, T. Rauhe, and A. Sandholm. Mona: Monadic second-order logic in practice. In *TACAS '95*, volume 1019 of *LNCS*. Springer, 1995.
20. P. Hooimeijer, B. Livshits, D. Molnar, P. Saxena, and M. Veanes. Fast and precise sanitizer analysis with Bek. In *Proceedings of the USENIX Security Symposium*, August 2011.
21. P. Hooimeijer and M. Veanes. An evaluation of automata algorithms for string analysis. In *VMCAI'11*, volume 6538 of *LNCS*, pages 248–262. Springer, 2011.

22. J. E. Hopcroft and J. D. Ullman. *Introduction to Automata Theory, Languages, and Computation*. Addison Wesley, 1979.
23. N. Klarlund, A. Møller, and M. I. Schwartzbach. MONA implementation secrets. *International Journal of Foundations of Computer Science*, 13(4):571–586, 2002.
24. K. Korovin. Instantiation-based automated reasoning: From theory to practice. In *Automated Deduction - CADE-22, 22nd International Conference on Automated Deduction*, volume 5663 of *LNAI*, pages 2–7. Springer, 2009.
25. K. Korovin. Inst-gen – a modular approach to instantiation-based automated reasoning. In *Programming Logics, Essays in Memory of Harald Ganzinger*. Springer, 2013.
26. K. Korovin. Personal communication, December 2013.
27. Y. Li, A. Albarghouthi, Z. Kincaid, A. Gurfinkel, and M. Chechik. Symbolic optimization with SMT solvers. In *POPL'14*, pages 607–618. ACM, 2014.
28. M. Löb. Decidability of the monadic predicate calculus with unary function symbols. *Journal of Symbolic Logic*, 32:563, 1967.
29. L. Löwenheim. Über Möglichkeiten im Relativkalkül. *Math. Annalen*, 76:447–470, 1915.
30. F. Nielson, H. R. Nielson, and C. Hankin. *Principles of Program Analysis*. Springer, 2010.
31. T. Nipkow. Linear quantifier elimination. In *Proceedings of the 4th International Joint Conference on Automated Reasoning, IJCAR '08*, volume 5195 of *LNAI*, pages 18–33. Springer, 2008.
32. M. Nivat. Transductions des langages de Chomsky. *Annales de l'institut Fourier*, 18(1):339–455, 1968.
33. G. V. Noord and D. Gerdemann. Finite state transducers with predicates and identities. *Grammars*, 4:263–286, 2001.
34. M. Presburger. 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*, pages 92–101, Warsaw, Poland, 1929.
35. M. O. Rabin. Decidability of second-order theories and automata on infinite trees. *Trans. Amer. Math. Soc.*, 141:1–35, 1969.
36. C. R. Reddy and D. W. Loveland. Presburger arithmetic with bounded quantifier alternation. In *Proceedings of the Tenth Annual ACM Symposium on Theory of Computing*, STOC'78, pages 320–325, New York, NY, USA, 1978. ACM.
37. H. G. Rice. Classes of recursively enumerable sets and their decision problems. *Trans. Amer. Math. Soc.*, 74:358–366, 1953.
38. P. Saxena, D. Molnar, and B. Livshits. Scriptgard: Preventing script injection attacks in legacy web applications with automatic sanitization. Technical Report MSR-TR-2010-128, Microsoft Research, August 2010.
39. W. Thomas. Languages, automata, and logic. In *Handbook of Formal Languages*, volume 3, pages 389–455. Springer, 1996.
40. The Unicode Consortium. *The Unicode Standard 6.3, Emoticons*. http://unicode.org/charts/PDF/U1F600.pdf.
41. M. Y. Vardi. Why is modal logic so robustly decidable? In N. Immerman and P. G. Kolaitis, editors, *Descriptive Complexity and Finite Models*, volume 31 of *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, pages 149–184. American Mathematical Society, 1996.
42. M. Veanes and N. Bjørner. Symbolic automata: The toolkit. In C. Flanagan and B. König, editors, *TACAS 2012*, volume 7214 of *LNCS*. Springer, 2012.

43. M. Veanes, N. Bjørner, and L. de Moura. Symbolic automata constraint solving. In C. Fermüller and A. Voronkov, editors, *LPAR-17*, volume 6397 of *LNCS/ARCoSS*, pages 640–654. Springer, 2010.
44. M. Veanes, P. Hooimeijer, B. Livshits, D. Molnar, and N. Bjørner. Symbolic finite state transducers: Algorithms and applications. In *POPL'12*, pages 137–150, 2012.
45. S. Yu. Regular languages. In G. Rozenberg and A. Salomaa, editors, *Handbook of Formal Languages*, volume 1, pages 41–110. Springer, 1997.