# Effectively Monadic Predicates

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

We ran into the following decision problem that we named *monadic decomposition*:

> "Given an effective representation of a binary relation $R \subseteq A \times B$, decide if $R$ is a *finite* union $\bigcup_{0 \le i < k} R_i$ of some nonempty Cartesian products $R_i = A_i \times B_i$, and $k > 0$, and if so, construct such $R_i$ effectively. Call $k$ the *width* of the decomposition."

At first glance this seemed to be a standard problem one might look up in some classical literature on recursion theory because we work in a fixed background structure $\mathfrak{U}$ with an re (recursively enumerable) universe $\mathcal{U}$.[1] However, the exact circumstances are somewhat unusual. We assume an re set $\Psi$ of (open) formulas such that:

1. If $a \in \mathcal{U}$ and $x$ is a variable then $x \doteq a, a \doteq x \in \Psi$,[2] and if $\psi \in \Psi$ then $\psi[x/a] \in \Psi$.
2. If $\psi, \varphi \in \Psi$ then $\psi \wedge \varphi, \psi \vee \varphi, \neg\varphi \in \Psi$.
3. $\Psi$ is *decidable*: given $\psi(\bar{x}) \in \Psi$, we can decide if $\psi(\bar{x})$ is *satisfiable*, i.e., if $\mathfrak{U} \models \exists\bar{x}\psi(\bar{x})$.

When $\psi(\bar{x})$ is satisfiable it follows that we can also effectively generate a *witness* $\bar{a}$ such that $\psi(\bar{a})$ holds, because $\mathcal{U}$ is re. What makes this setup unusual from a classical standpoint is the last item. Essentially, we assume an unlimited supply of "uninterpreted constants" (the free variables) and that we can decide satisfiability and construct satisfiable interpretations for those constants.[3]

From the standpoint of program analysis with state-of-the-art satisfiability modulo theories (SMT) solvers the above setup illustrates a basic use of any SMT solver that supports model generation [3]. Next, we illustrate the concrete program analysis context that caused us to investigate monadic decomposition.

For security analysis of string sanitizers [4, 8], it is often useful to express string constraints by using *symbolic finite automata* [7] or SFAs. In an SFA, labels on transitions are *predicates* over a character theory rather than concrete characters.

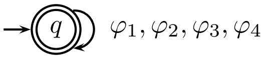
SFAs can for example express valid sequences of inputs (e.g., inputs to a decoder that do not cause exceptions) as well as potential sets of attack vectors (e.g., outputs that may cause

[1] Thus all $a \in \mathcal{U}$ can be named effectively; to avoid clutter, we write $a$ also for a term denoting $a$.

[2] The symbol $\doteq$ is used as the formal equality symbol.

[3] Such interpretations are essentially *expansions* of $\mathfrak{U}$, using standard terminology of model theory.

a security vulnerability). In the first case a symbolic automaton can be obtained by extracting the domain automaton of the decoder. Suppose that a single character is a sequence of one up to four bytes (say *byte*$^{1..4}$). The particular decoder we have in mind here is a *UTF8 decoder.* A symbolic automaton extracted from a UTF8 decoder may have the following (seemingly trivial) structure:

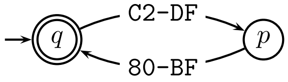


There is a single state $q$ and four loops each with label $\varphi_n$. Suppose that $\varphi_n(x)$ holds iff $|x| = n$ and $f_n(x[0], \ldots, x[n-1])$ computes a Unicode code point from the $n$ bytes (i.e., $f_n$ does not reject them), e.g., $f_2(\texttt{C5}_{16}, \texttt{92}_{16}) = \texttt{152}_{16}$ but $f_2(\texttt{FF}_{16}, \texttt{FF}_{16})$ throws an exception. The condition $\varphi_n$ involves fairly nontrivial arithmetic operations and is extracted from path conditions and output expressions of the decoder for the case that handles $n$ input bytes (the value $n$ is determined by the first byte).

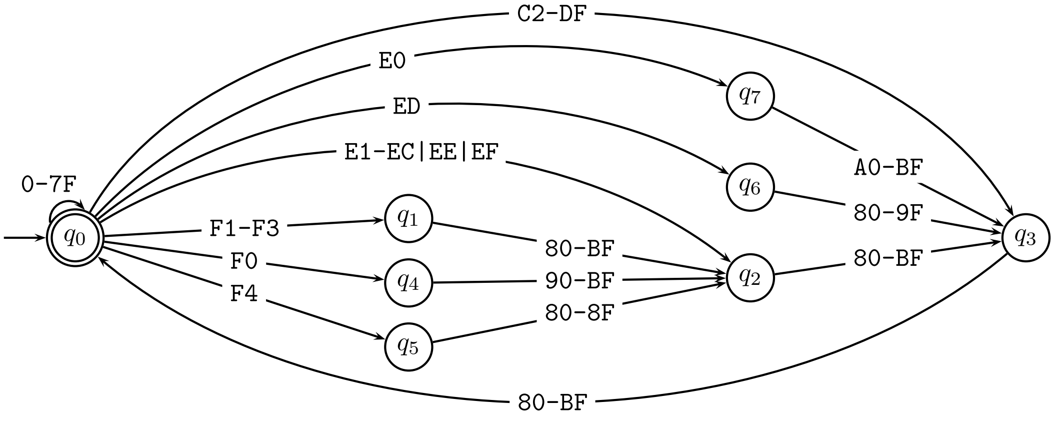
For example, the sequence $[[\texttt{4F}_{16}], [\texttt{C5}_{16}, \texttt{92}_{16}], [\texttt{45}_{16}]]$ is accepted by the above SFA and stands for the UTF8 encoding of the string "OŒE".

For further analysis, the above SFA is not very useful. The character boundaries imposed by the above SFA are in some sense "fake". For most purposes, e.g., to decide if the decoder admits over-encodings, one would like to intersect it with other property SFAs, expressed for example by regexes (regular expressions over Unicode characters), and to check emptiness of the resulting automata. In order to do so, the alphabet type must be reduced from *byte*$^{1..4}$ to *byte*. The predicate $\varphi_1$ is already in the right form, it is $0 \le x \le \texttt{7F}_{16}$.

One approach to do this is to compute monadic decompositions (generalized for $n > 2$) of each $\psi_n(x_0, \ldots, x_{n-1}) = \varphi_n([x_0, \ldots, x_{n-1}])$ (for $n > 1$) and to expand the SFA. Take the case of $n = 2$. It turns out that $\psi_2(x, y)$ is equivalent to $\texttt{C2}_{16} \le x \le \texttt{DF}_{16} \wedge \texttt{80}_{16} \le y \le \texttt{BF}_{16}$. Therefore, the transition $q \overset{\varphi_2}{\to} q$ above can be replaced by two transitions:



The predicates $\psi_3$ and $\psi_4$ can be decomposed similarly (their monadic decompositions have higher widths). It is interesting to observe that after the decomposition of all the predicates and after further *minimization* we obtain the following SFA, provided the decoder is correct:



This SFA corresponds to the following regex

```
^([\x00-\x7F]|[\xC2-\xDF][\x80-\xBF]
|(\xE0[\xA0-\xBF]|\xED[\x80-\x9F]|[\xE1-\xEC\xEE\xEF][\x80-\xBF])[\x80-\xBF]
|(\xF0[\x90-\xBF]|[\xF1-\xF3][\x80-\xBF]|\xF4[\x80-\x8F])[\x80-\xBF]{2})*$
```

that describes all valid UTF8 encoded strings. Any string that is not accepted by this regex is either malformed or possibly over-encoded. Some UTF8 decoders do allow over-encoded strings in order to be more robust by avoiding exceptions as much as possible, but it is not always safe to do so, because over-encoding may expose security vulnerabilities [6, 5].

The above usage scenario is but a sample out of a large range of possible analysis tasks of string routines where monadic decomposition plays an important role. It was proved recently that certain analysis problems over ESFTs (extended symbolic finite transducers, introduced in [2]), remain decidable only when the ESFT is *Cartesian* [1]. The current paper implies that the main result in [1] also holds for *monadic* ESFTs, because, by using the decision procedure presented here, we can effectively reduce *monadic* ESFTs to equivalent Cartesian ESFTs.

Although we have only illustrated the application for one specific case, we believe that there are several other areas where monadic decomposition can be useful. In particular, because the technique is completely generic, no theory-specific assumptions are made about $\mathfrak{U}$.

In the following we describe the problem formally and prove some basic results.

## 2 Monadic predicates

Let $R$ be an $n$-ary relation for some $n \geq 2$. $R$ is *Cartesian* if there exist unary relations $U_i$, for $i < n$, such that $R$ is the direct product ($n$-way Cartesian product) $\prod_{i<n} U_i$. $R$ is *monadic* if there exists $k > 0$ and Cartesian $R_i$, for $i < k$, such $R = \bigcup_i R_i$; $\{R_i\}_{i<k}$ is called a *monadic decomposition of $R$ of width $k$*. $R$ is *$k$-monadic* if $R$ has a monadic decomposition of width $k$. Note that 1-monadic is the same as Cartesian. We lift the notions to predicates, i.e., effective representations of relations, a predicate $\varphi$ is *$k$-monadic* if there exist $k$ Cartesian predicates $\varphi_i$ such that $\varphi$ is equivalent to $\bigvee_{i<k} \varphi_i$.

We assume a decidable background $\mathfrak{U}$ as described above. The Boolean type is BOOL with truth values $\{\mathtt{T}, \mathtt{F}\}$. In our expressions, all variables are typed and all terms and formulas are welltyped. We use $\lambda$-expressions to define anonymous functions and predicates. The type of the elements is implicit and determined by the context. We write $[\![\varphi]\!]$ for the relation defined by a predicate $\varphi$.

**Example 1.** *Let $\varphi$ 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 $R = ([\![\lambda x.x > 5]\!] \times [\![\lambda y.\mathtt{T}]\!]) \cup ([\![\lambda x.x > 4]\!] \times [\![\lambda y.odd(y)]\!])$. $\varphi$ is satisfiable, for example $(5, 3) \in [\![\varphi]\!]$.* ⊠ ⊠

We assume that *tuples* are part of the background, i.e., if we have types $\sigma_i$ for some $i$, $0 \leq i < k$, and some $k \geq 1$ then we also have direct product types $\prod_{i=0}^{k-1} \sigma_i$. This does not violate the third assumption on $\Psi$, we can always treat a variable $x : \sigma_1 \times \sigma_2$ as two variables $x_1 : \sigma_1$ and $x_2 : \sigma_2$. The only operations on a tuple are constructing it and projecting its elements, tuples can always be eliminated by introducing more variables.

The arity of $R$ and what constitutes a monadic decomposition of $R$ clearly depends on the argument types, this information is implicitly assumed. For example, we may have a ternary relation $R$ over $\prod_{i=1}^{3} \mathbb{Z}$ and effectively transform it into a binary relation over $(\mathbb{Z} \times \mathbb{Z}) \times \mathbb{Z}$.

## 3 Monadic decomposition

We are interested in the following two problems.

1. Deciding if a predicate $\varphi$ is monadic.

2. Given a monadic predicate $\varphi$, effectively constructing a monadic decomposition of $\varphi$.

We restrict our attention to *binary* predicates (without loss of generality).

Once we have solved 1 and 2 for binary predicates, generalization to $n$-ary predicates, where $n > 2$, is relatively straightforward: Suppose $\lambda(x, y, z).\varphi(x, y, z)$ is given, where $x$, $y$ and $z$ have types $A$, $B$ and $C$, respectively. First, decide if $\psi = \lambda(w, z).\varphi(\mathit{first}(w), \mathit{second}(w), z)$ is monadic where the type of $w$ is $A\times B$. If $\psi$ is not monadic then neither is $\varphi$. Suppose $\psi$ is 2-monadic with a monadic decomposition $\lambda(w, z).(\psi_{11}(w) \wedge \psi_{12}(z)) \vee ((\psi_{21}(w) \wedge \psi_{22}(z))$. Second, decide if the binary predicates $\lambda(x, y).\psi_{11}(\langle x, y\rangle)$ and $\lambda(x, y).\psi_{21}(\langle x, y\rangle)$ are monadic. If at least one of them is not monadic then neither is $\varphi$. If both are monadic, then use their monadic decompositions in place of $\psi_{11}$ and $\psi_{21}$, and use standard distributive laws of the Boolean connectives to derive a monadic decomposition of $\varphi$.

## 3.1 Deciding if a predicate is monadic

We conjecture that this problem is undecidable in general.

Consider any term $f(x)$ in the background theory denoting a function over integers. Let $\varphi_f(x, y)$ be the formula $f(x) \doteq y$. 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 all $i < k$. So $\varphi_f$ is monadic iff $f$ is bounded (finite-valued). Deciding if $f$ is bounded is an undecidable problem in general. However, we have not investigated the necessary conditions on the background that would cause undecidability of this problem.

## 3.2 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 $A$ stands for $\{a \mid \exists b\, R(a, b)\}$ and $B$ stands for $\{b \mid \exists a\, R(a, b)\}$. Define the relations:

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

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 following properties are used below.

**Lemma 1.** *Let $R$ and $A$ be given as above.*

1. *It holds for all $a, a' \in A$ that $a \sim a' \Leftrightarrow Y_a = Y_{a'}$.*
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 $Y_a \subseteq Y_{a'}$. Let $(a, b) \in R$. We need to 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 $Y_a = Y_{a'}$. Let $(a, b), (a', b') \in R$, i.e., $b \in Y_a$ and $b' \in Y_{a'}$. So, by $Y_a = Y_{a'}$, we have $b \in Y_{a'}$ and $b' \in Y_a$. It follows that $(a, b'), (a', b) \in R$.

*Proof of 2*: Immediate by using 1. □

**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 $Y_{a_i} = Y_a$ 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 $A_i \subseteq \tilde{A}_i$.

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. We can 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 $(M_I \cup M_J) \times B_i$ by $(M_I \times B_i) \cup (M_J \times B_i)$,
- 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 $2^n - 1$, that is the maximum number $m$ of minterms, where $n$ is the smallest width of a monadic decomposition of $R$.

$\Leftarrow$: 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 $0 \leq i < n$. 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 $b \in B_i$. It follows that $\{A_i \times B_i\}_{i<n}$ is a monadic decomposition of $R$. □

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 $\varphi$, 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 $(a',b') \in R_j \setminus R_i$.

Let $(a_0, b_0) \in [\![\varphi]\!]$ and let $W_A = \{a_0\}$. Iterate the following procedure.

1. Let $\psi(x) = \bigwedge_{a \in W_A} x \nsim a$
2. If $\psi$ is satisfiable and $\psi(a)$ holds then update $W_A := W_A \cup \{a\}$ else terminate.

Observe that satisfiability checking of $\psi$ as well as generating the witness $a$ is decidable becuse we can transform $\psi$ to prenex normal form as an $\exists$-formula and treat all the existential variables as free variables, i.e., the resulting formula is in $\Psi$. When $\psi$ becomes unsatisfiable then any further element from $A$ must be $\sim$-equivalent to one of the elmenets already in $W_A$, while all elements in $W_A$ belong to distinct $\sim$-equivalence classes. Therefore, if $\varphi$ 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 $\varphi$ is Cartesian then $\psi$ is unsatisfiable initially, because then $A_\sim = \{[a_0]_\sim\}$.

**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 set $W_B$ such that $B_\backsim = \{[b]_\backsim \mid b \in W_B\}$. Let

$$\begin{array}{lrcl}
(\text{for } b \in W_B) & \beta_{[b]_\backsim}(y) & \overset{\text{def}}{=} & (\bigwedge\limits_{a \in W_A \cap X_b} \varphi(a,y)) \wedge \neg(\bigvee\limits_{a \in W_A \setminus X_b} \varphi(a,y)) \\
(\text{for } a \in W_A) & \alpha_{[a]_\sim}(x) & \overset{\text{def}}{=} & (\bigwedge\limits_{b \in W_B \cap Y_a} \varphi(x,b)) \wedge \neg(\bigvee\limits_{b \in W_B \setminus Y_a} \varphi(x,b))
\end{array}$$

Observe that $\alpha_{[a]_\sim}$ is well-defined because for all $a' \in [a]_\sim$ we have that $Y_a = Y_{a'}$. Similarly for $\beta_{[b]_\backsim}$. One can show that $\llbracket \beta_{[b]_\backsim} \rrbracket = [b]_\backsim$ and $\llbracket \alpha_{[a]_\sim} \rrbracket = [a]_\sim$ by using standard laws of logic. □

Lemma 3 is essentially a quantifier elimination result 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 that also gives us a brute force method for monadic decomposition.

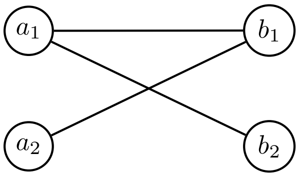
**Theorem 1.** *If $\varphi(x,y)$ is monadic then*

1. *$\varphi(x,y)$ is equivalent to $\lambda(x,y).\bigvee_{a \in W_A}(\alpha_{[a]_\sim}(x) \wedge \varphi(a,y))$.*
2. *$\varphi(x,y)$ is equivalent to $\lambda(x,y).\bigvee_{b \in W_B}(\beta_{[b]_\backsim}(y) \wedge \varphi(x,b))$.*
3. *$\varphi(x,y)$ is equivalent to $\lambda(x,y).\bigvee_{a \in W_A, b \in W_B, (a,b) \in \llbracket \varphi \rrbracket}(\alpha_{[a]_\sim}(x) \wedge \beta_{[b]_\backsim}(y))$.*

*Proof.* We prove 1. The other cases are similar. By Lemma 3 we have $\llbracket \alpha_{[a]_\sim} \rrbracket = [a]_\sim$. By construction of $W_A$ we have that, for all $a \in W_A$ we have $[a]_\sim \times Y_a \subseteq \llbracket \varphi \rrbracket$ where $[a]_\sim \times Y_a = \llbracket \lambda(x,y).\alpha_{[a]_\sim}(x) \wedge \varphi(a,y) \rrbracket$. In the other direction, if $(a,b) \in \llbracket \varphi \rrbracket$ then $a \in \llbracket \alpha_{[a]_\sim} \rrbracket$ and $b \in Y_a$. In other words, $(a,b) \in \llbracket \lambda(x,y).\alpha_{[a]_\sim}(x) \wedge \varphi(a,y) \rrbracket$. □

We write $\alpha_a$ for $\alpha_{[a]_\sim}$ and $\beta_b$ for $\beta_{[b]_\backsim}$.

**Example 2.** *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 $\phi$ is satisfied by a finite model of the form:*



*We get the following predicates*

$$\begin{array}{rcl}
\alpha_{a_1}(x) & := & x \doteq a_1 \\
\alpha_{a_2}(x) & := & x \doteq a_2 \\
\beta_{b_1}(y) & := & y \doteq b_1 \\
\beta_{b_2}(y) & := & y \doteq b_2
\end{array}$$

*where $a_1 = 0, a_2 = 1, b_1 = 0, b_2 = 1$. The monadic decomposition of $\phi$ reconstructs the formula as*

$$\alpha_{a_1}(x) \wedge \beta_{b_1}(y) \;\vee\; \alpha_{a_1}(x) \wedge \beta_{b_2}(y) \;\vee\; \alpha_{a_2}(x) \wedge \beta_{b_1}(y) \;.$$

*Note that $\alpha_{a_2}(x) \wedge \beta_{b_2}(y)$ is not included because $\phi(1,1)$ is false.* ⊠

### 3.3 Deciding if a predicate is monadic in integer linear arithmetic

Consider 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 formula in Presburger arithmetic,

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

**Theorem 2.** *Monadic decomposition is decidable for integer linear arithmetic.*

*Proof.* Let $\varphi(x,y)$ be a formula in integer linear arithmetic.

We show that $\varphi$ is monadic $\Leftrightarrow$ $\mathit{IsMonadic}(\varphi)$ is true in Presburger arithmetic.

Proof of $\Rightarrow$: Assume $\varphi$ 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 $\varphi$ is monadic by Lemma 2. □

## 4 Conclusion

We introduced the problem of monadic decomposition of predicates in decidable theories. The problem has several useful applications in program analysis. We are currently investigating more efficient algorithms for implementing monadic decomposition, other than the brute force approach that we presented. Deciding if a predicate is monadic in specific background theories besides integer linear arithmetic, is another interesting open problem.

## References

[1] 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.

[2] 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.

[3] L. De Moura and N. Bjørner. Satisfiability modulo theories: introduction and applications. *Commun. ACM*, 54(9):69–77, 2011.

[4] 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.

[5] NVD. `http://web.nvd.nist.gov/view/vuln/detail?vulnId=CVE-2008-2938`.

[6] SANS. Malware faq. `http://www.sans.org/security-resources/malwarefaq/wnt-unicode.php`.

[7] M. Veanes. Applications of symbolic finite automata. In S. Konstantinidis, editor, *CIAA 2013*, volume 7982 of *LNCS*, pages 16–23. Springer, 2013. Invited talk.

[8] 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.