Abstract Symbolic Automata

Mixed syntactic/semantic similarity analysis of executables

Mila Dalla Preda[1] Roberto Giacobazzi[1,3] Arun Lakhotia[2] Isabella Mastroeni[1]

[1]University of Verona [2]University of Louisiana [3] Irdeto Canada

mila.dallapreda@univr.it, roberto.giacobazzi@univr.it, arun@louisiana.edu, isabella.mastroeni@univr.it

Abstract

We introduce a model for mixed syntactic/semantic approximation of programs based on symbolic finite automata (SFA). The edges of SFA are labeled by predicates whose semantics specifies the denotations that are allowed by the edge. We introduce the notion of abstract symbolic finite automaton (ASFA) where approximation is made by abstract interpretation of symbolic finite automata, acting both at syntactic (predicate) and semantic (denotation) level. We investigate in the details how the syntactic and semantic abstractions of SFA relate to each other and contribute to the determination of the recognized language. Then we introduce a family of transformations for simplifying ASFA. We apply this model to prove properties of commonly used tools for similarity analysis of binary executables. Following the structure of their control flow graphs, disassembled binary executables are represented as (concrete) SFA, where states are program points and predicates represent the (possibly infinite) I/O semantics of each basic block in a constraint form. Known tools for binary code analysis are viewed as specific choices of symbolic and semantic abstractions in our framework, making symbolic finite automata and their abstract interpretations a unifying model for comparing and reasoning about soundness and completeness of analyses of low-level code.

Categories and Subject Descriptors D.3.1 [*Programming Languages*]: Formal Definitions and Theory; F.3.2 [*Logics and Meanings of Programs*]: Semantics of Programming Languages—program analysis

General Terms Languages.

Keywords Symbolic automata, abstract interpretation.

1. Introduction

The problem. Similarity analysis is a key component in mining and understanding huge software enclaves, including code, e,g., coming from malware repositories, specifications, analyses and other heterogeneous meta-data. This is particularly relevant when dealing with binary executables, which, besides representing a large portion of existing malware, also represent a highly malleable often hard to analyze carrier. This is due to its unstructured nature, allowing self-modification, overlapping instructions, and untyped computations where data and code coexist without any predefined (static) boundary.

In order to mine both semantic meanings and syntactic patterns from programs, existing tools for similarity analysis of binary executables always employ mixed syntactic/symbolic and semantic representations of programs. At syntactic level properties concerning the control flow graph, such as in *BinHunt* [15] and *BinDiff* [12, 25], or feature vectors concerning sequences of instructions, are used together with graph-isomorphism, sequence comparison algorithms, and hash functions for extracting structural similarities in code. At semantic level, more advanced semantic properties such as those extracted from symbolic executions, dynamic analysis and emulation, such as those used in *BinJuice* [18] and *BinHunt* [15], are employed for bypassing semantic preserving code transformations for code obfuscation, e.g., for similarity analysis in malware detection. The use of mixed syntactic/semantic representation of code in similarity analysis is becoming a good practice because pure semantic similarity is too complex and often undecidable while pure syntactic similarities is too imprecise and prone to false negatives due to code obfuscation techniques. This is precisely what happens in most known tools and methods for dissecting and comparing programs in order to extract semantic similarities from syntactically different code. However, none of these tools have a formal semantic model in which relative precision and soundness can be formally proved. This paper is intended to fill this gap.

Our contribution. We attack this problem by observing that most known methods employed in similarity analysis of disassembled binaries can be seen as peculiar abstract interpretations of *symbolic finite state automata* (SFA). Symbolic finite automata, introduced in [23] and further developed in [8, 9], provide the ideal formal setting in order to treat within the same model the abstraction of both the syntactic structure of programs and their intended semantics.

SFA have been introduced as an extension of traditional finite state automata for modeling languages with a potential infinite alphabet. Transitions in SFA are therefore modeled as constraints interpreted in a given Boolean algebra, providing the semantic interpretation of constraints, and therefore the (potentially infinite) structural components of the language recognized (see [9, 23]).

Our main contribution is the introduction of the notion of *abstract symbolic finite automaton*, where approximation is made by abstract interpretation of standard SFA. Abstract interpretation here acts both at syntactic (predicate), topological (graph), and semantic (denotation) level. We investigate in details how the syntactic, topological, and semantic abstractions of SFA relate to each other and interfere when automata, at different levels of abstractions, are compared with respect to their recognized language.

The abstraction respectively on syntactic predicates and semantic structures corresponds precisely to the abstract interpretation of the underlying Boolean algebra of a concrete SFA M, resulting in a

POPL '15, January 15–17, 2015, Mumbai, India.
Copyright © 2015 ACM 978-1-4503-3300-9/15/01…$15.00.
http://dx.doi.org/10.1145/2676726.2676986

different SFA A whose language recognized is an over approximation of the language of M. The key aspect here is to maintain a relative compatibility between syntactic abstractions on predicates and constraint formulae and the abstractions of their semantics. This intuitively means that the approximate predicates and their interpretation provide, one over the others, coherent partitions of objects (respectively interpretations and predicates).

Topological abstraction means instead changing the graph structure of SFA, yet keeping correctness, namely providing an over approximation of the recognized language of M. This is achieved by generalizing a minimization algorithm proposed in [9] with respect to a family of equivalence relations on SFA states. The result is a simplification of M which is still correct in the sense of abstract interpretation with respect to M.

Abstract SFA provide a general enough model for representing syntactic and semantic properties of arbitrary programming languages. We apply our model in the attempt to formalize and prove properties of two commonly used tools for similarity analysis of binary executables, notably ***BinJuice*** [18] and ***BinDiff***. Following the structure of their control flow graphs, disassembled binary executables are represented as (concrete) SFA, where states are program points between basic blocks and predicates represent the (possibly infinite) I/O semantics of each basic block in a constraint form. Tools for binary-level similarity analysis are then formalized as abstract interpretations of these concrete SFA. By studying the properties of the corresponding abstractions we can provide a first unifying model for formally proving properties for these tools. Moreover, our model suggests potential refinements of similarity analyses for disassembled binaries such as the possibility of extracting minimal SFA from binaries as canonical signatures for code fragments.

2. Preliminaries

Mathematical Notation. Given two sets S and T, we denote with $\wp(S)$ the powerset of S, $\wp^{\mathrm{re}}(S)$ the set of recursive enumerable (r.e.) subsets of S, with $S \smallsetminus T$ the set-difference between S and T, with $S \subset T$ strict inclusion and with $S \subseteq T$ inclusion. S^* denotes the set of all finite sequences of elements in S. A set L with ordering relation $\leq$ is a poset and it is denoted as $\langle L, \leq\rangle$. A poset $\langle L, \leq\rangle$ is a lattice if $\forall x.y \in L$ we have that $x \vee y$ and $x \wedge y$ belong to L. A lattice $\langle L, \leq\rangle$ is complete when for every $X \subseteq L$ we have that $\bigvee X, \bigwedge X \in L$. As usual a complete lattice L, with ordering $\leq$, least upper bound (lub) $\vee$, greatest lower bound (glb) $\wedge$, greatest element (top) $\top$, and least element (bottom) $\bot$ is denoted by $\langle L, \leq, \vee, \wedge, \top, \bot\rangle$. Given $f : S \longrightarrow T$ and $g : T \longrightarrow Q$ we denote with $g \circ f : S \longrightarrow Q$ their composition, i.e., $g \circ f = \lambda x.g(f(x))$. $f : L \longrightarrow D$ on complete lattices is *additive (co-additive)* if for any $Y \subseteq L, f(\vee_L Y) = \vee_D f(Y)$ $(f(\wedge_L Y) = \wedge_D f(Y))$. Continuity holds when f preserves *lubs*'s of chains. Co-continuity is dually defined. For a continuous function f: $\mathit{lfp}(f) = \bigwedge\left\{ x \,\middle|\, x = f(x) \right\} = \bigvee_{n\in\mathbb{N}} f^n(\bot)$ where $f^0(\bot) = \bot$ and $f^{n+1}(\bot) = f(f^n(\bot))$.

Abstract Interpretation. Abstract domains can be equivalently formalized either as Galois connections or closure operators on a given concrete domain which is a complete lattice C (cf. [4]). Let C and A be complete lattices, a pair of monotone functions $\alpha : C \longrightarrow A$ and $\gamma : A \longrightarrow C$ forms a *Galois connection (GC)* between C and A if for every $x \in C$ and $y \in A$ we have $\alpha(x) \leq_A y \Leftrightarrow x \leq_C \gamma(y)$. α (resp. γ) is the *left-adjoint* (resp. *right-adjoint*) to γ (resp. α) and it is additive (resp. co-additive). If $\langle\alpha, \gamma\rangle$ is a GC between C and A then $\gamma \circ \alpha \in \mathit{uco}(C)$. If $\rho \in \mathit{uco}(C)$ then $\langle\rho, \mathtt{id}\rangle$ is a CG between C and $\rho(C)$. Given an additive (resp. co-additive) function α (resp. γ) we have a GC $\langle\alpha, \alpha^+\rangle$ (resp. $\langle\gamma^-, \gamma\rangle$) by considering its right (resp. left) adjoint $\alpha^+ = \lambda x. \bigvee\{y \mid \alpha(y) \leq x\}$ (resp. $\gamma^- = \lambda x. \bigwedge\{y \mid x \leq \gamma(y)\}$). An *upper closure operator* (or simply a *closure*) on a poset $\langle L, \leq\rangle$ is an operator $\rho : L \longrightarrow L$ which is monotone, idempotent, and extensive (i.e., $x \leq \rho(x)$). We denote with $\mathit{uco}(L)$ the set of all closure operators on the poset L. If C is a complete lattice, then $\langle \mathit{uco}(C), \sqsubseteq, \sqcup, \sqcap, \lambda x.\, C, \mathtt{id}\rangle$ forms a complete lattice [24], which is the set of all possible abstractions of C, where the bottom is $\mathtt{id} = \lambda x.x$ and for every $\rho, \eta \in \mathit{uco}(C)$, ρ is *more concrete than* η iff $\rho \sqsubseteq \eta$ iff $\forall y \in C.\ \rho(y) \leq \eta(y)$ iff $\eta(C) \subseteq \rho(C)$, $(\sqcap_{i\in I}\rho_i)(x) = \wedge_{i\in I}\rho_i(x)$; $(\sqcup_{i\in I}\rho_i)(x) = x$ iff $\forall i \in I.\ \rho_i(x) = x$. $\rho \in \mathit{uco}(C)$ is disjunctive when $\rho(C)$ is a join-sublattice of C which holds iff ρ is additive (cf. [4]). $\rho \in \mathit{uco}(\wp(C))$ is *partitioning* (or induces a partition) if it is additive and $\{\rho(\{c\})\}_{c\in C}$ is a partition of C [17]. If $\rho \in \mathit{uco}(\wp(C))$ then the most abstract partitioning closure containing ρ:

$$\Pi(\rho) \overset{\text{def}}{=} \bigsqcup \left\{ \beta \in \mathit{uco}(\wp(C)) \,\middle|\, \beta \sqsubseteq \rho \wedge \beta \text{ is partitioning } \right\}.$$

The key aspect of partitioning closures is that they preserve the structure of Boolean algebras.

If $f : C \longrightarrow C$ is a continuous function and $\rho \in \mathit{uco}(C)$ is an abstraction, then f always has a *best correct approximation* in $\rho(C)$ which is $f^\rho \overset{\text{def}}{=} \rho \circ f \circ \rho$. Any approximation $f^\sharp : \rho(C) \longrightarrow \rho(C)$ of f in $\rho(C)$ is *sound* if $f^\rho \sqsubseteq f^\sharp$. In this case we have the fixpoint soundness $\rho(\mathit{lfp} f) \leq \mathit{lfp}(f^\rho) \leq \mathit{lfp}(f^\sharp)$(cf. [3]). $f^\sharp$ is *complete* when $\rho \circ f = f^\sharp \circ \rho$ which holds iff $\rho \circ f = \rho \circ f \circ \rho$ (cf. [16]). Therefore the possibility of defining a complete approximation $f^\sharp$ of f on some abstract domain ρ only depends on f and ρ. In this case we have: $\rho(\mathit{lfp} f) = \mathit{lfp}(f^\rho) = \mathit{lfp}(f^\sharp)$. In the following, for any semantics $[\![\cdot]\!] : \mathcal{S} \longrightarrow \mathcal{D}$ mapping syntactic objects in $\mathcal{S}$ into denotations in $\mathcal{D}$ such that $[\![\cdot]\!]$ is an element in the set of fixpoint semantics $\mathfrak{S} \subseteq \mathcal{S} \longrightarrow \mathcal{D}$ inductively defined as follows

$$\mathfrak{S} \quad ::= \quad f : \mathcal{S} \longrightarrow \mathcal{D} \ \mid\ \mathit{lfp}(\mathfrak{S}) \ \mid\ \mathfrak{S} \circ \mathfrak{S}$$

and if $\rho \in \mathit{uco}(\mathcal{D})$, we denote by $[\![\cdot]\!]^\rho \in \mathfrak{S}^\rho \subseteq \mathcal{S} \longrightarrow \rho(\mathcal{D})$ the corresponding best correct approximation which is defined inductively on the structure of $\mathfrak{S}$ as follows:

$$\mathfrak{S}^\rho \quad ::= \quad \rho \circ f \circ \rho \ \mid\ \mathit{lfp}(\mathfrak{S}^\rho) \ \mid\ \mathfrak{S}^\rho \circ \mathfrak{S}^\rho$$

It is known that $[\![\cdot]\!]^\rho$ is sound and, whenever ρ is complete for the basic semantic operators f defining $[\![\cdot]\!] \in \mathfrak{S}$, then $[\![\cdot]\!]^\rho$ is complete, i.e. for any $s \in \mathcal{S}$: $\rho([\![s]\!]) = [\![s]\!]^\rho$ (cf. [4, 16]).

Symbolic Finite Automata. Symbolic automata and finite state transducers have been introduced to deal with specifications involving a potentially infinite alphabet of symbols [8, 9, 23]. We follow [9] in specifying symbolic automata in terms of effective Boolean algebra. Consider an effective Boolean algebra $\mathcal{A} = \langle \mathfrak{D}_\mathcal{A}, \Psi_\mathcal{A}, [\![\cdot]\!], \bot, \top, \wedge, \vee, \neg\rangle$, with domain elements in a r.e. set $\mathfrak{D}_\mathcal{A}$, a r.e. set of predicates $\Psi_\mathcal{A}$ closed under boolean connectives $\wedge$, $\vee$ and $\neg$. The semantic function $[\![\cdot]\!] : \Psi_\mathcal{A} \longrightarrow \wp(\mathfrak{D}_\mathcal{A})$ is a partial recursive function such that $[\![\bot]\!] = \varnothing$, $[\![\top]\!] = \mathfrak{D}_\mathcal{A}$, and $\forall \varphi, \phi \in \Psi_\mathcal{A}$ we have that $[\![\varphi \vee \phi]\!] = [\![\varphi]\!] \cup [\![\phi]\!]$, $[\![\varphi \wedge \phi]\!] = [\![\varphi]\!] \cap [\![\phi]\!]$, and $[\![\neg\varphi]\!] = \mathfrak{D}_\mathcal{A} \smallsetminus [\![\varphi]\!]$. In the following we abuse notation by denoting with $[\![\cdot]\!]$ also its additive lift to $\wp(\Psi_\mathcal{A})$, i.e., for any $\Phi \in \wp(\Psi_\mathcal{A})$: $[\![\Phi]\!] = \left\{ [\![\varphi]\!] \,\middle|\, \varphi \in \Phi \right\}$. For $\varphi \in \Psi_\mathcal{A}$ we write $\mathit{IsSat}(\varphi)$ when $[\![\varphi]\!] \neq \varnothing$ and say that φ is *satisfiable*. $\mathcal{A}$ is decidable if IsSat is decidable.

DEFINITION 2.1. *A symbolic automaton (SFA) is $\langle \mathcal{A}, Q, q_0, F, \Delta\rangle$ where $\mathcal{A}$ is an effective Boolean algebra, Q is a finite set of states, $q_0 \in Q$ is the initial state, $F \subseteq Q$ is the set of final states and $\Delta \subseteq Q \times \Psi_\mathcal{A} \times Q$ is a finite set of transitions.*

A transition in $M = \langle \mathcal{A}, Q, q_0, F, \Delta\rangle$ labeled φ from state p to state q, $(p, \varphi, q) \in \Delta$ is often denoted $p \xrightarrow{\varphi} q$. φ is called the *guard* of the transition. An a-move of a SFA M is a transition $p \xrightarrow{\varphi} q$ such that $a \in [\![\varphi]\!]$, also denoted $p \xrightarrow{a} q$. The language recognized by a state $q \in Q$ in M is defined as:

$$\mathscr{L}_q(M) = \left\{ a_1, \dots, a_n \in \mathfrak{D}_\mathcal{A} \;\middle|\; \begin{array}{l} \forall 1 \leq i \leq n.\ p_{i-1} \xrightarrow{a_i} p_i \\ p_0 = q,\ p_n \in F \end{array} \right\}$$

in this case, $\mathscr{L}(M) = \mathscr{L}_{q_0}(M)$. We assume *complete SFA*, namely where all states hold an out-going a-move, for any character $a \in \mathfrak{D}$. This can be simply achieved by adding a shaft-state $q_\perp \in Q$ such that $q_\perp \xrightarrow{\top} q_\perp \in \Delta$ and for all states q lacking an out-going a-move, for $a \in \mathfrak{D}$, then $q \xrightarrow{\neg\beta} q_\perp \in \Delta$ with $\beta = \bigvee \{ \varphi \mid q \xrightarrow{\varphi} p \wedge p \in Q \}$.

The following terminology holds for SFA: M is *deterministic* whenever $p \xrightarrow{\varphi} q, p \xrightarrow{\beta} q' \in \Delta$: if $IsSat(\varphi \wedge \beta)$ then $q = q'$. M is *clean* if for all $p \xrightarrow{\varphi} q \in \Delta$: p is reachable from q_0 and $IsSat(\varphi)$. M is *normalized* if for all $p, q \in Q$: there is at most one move from p to q. M is *minimal* if M is deterministic, clean, normalized and for all $p, q \in Q$:

$$p = q \;\Leftrightarrow\; \mathscr{L}_q(M) = \mathscr{L}_p(M)$$

Given a SFA $M = \langle \mathcal{A}, Q, q_0, F, \Delta \rangle$ and $\equiv \subseteq Q \times Q$, we define the *quotient SFA* $M_{/\equiv} \overset{\text{def}}{=} \langle \mathcal{A}, Q', q_0', F', \Delta' \rangle$ as follows: $Q' = \{ [q]_\equiv \mid q \in Q \}$, $\Delta' \subseteq Q' \times \Psi_\mathcal{A} \times Q'$ is such that $\Delta' = \{ ([q]_\equiv, \Phi, [q']_\equiv) \mid (p, \Phi, q') \in \Delta, p \in [q]_\equiv \}$, $q_0' = [q_0]_\equiv$, and $F' = \{ [q]_\equiv \mid q \in F \}$.

3. Abstracting Symbolic Automata

Approximating symbolic automata means building different automata recognizing an upper approximation of the original recognized language. This can be achieved by abstract interpretation of the underlying effective Boolean algebra $\mathcal{A}$ and by approximating the automaton's structure. When acting on the Boolean algebra we may either approximate the domain of denotations $\mathfrak{D}_\mathcal{A}$ where formulae and predicates are interpreted, or approximate the predicates in $\Psi_\mathcal{A}$ where formulae are built. In both cases we need to obtain as result an abstract effective Boolean algebra.

3.1 Abstract effective Boolean algebras

The duality of syntax and semantics is perfectly encoded in SFA by the underlying algebraic structure of effective Boolean algebras. They represent the universe of predicates and formulae (later called *syntax*) as well as the domain for their interpretation and semantics, providing the structure for expressing the language recognized by the given SFA. The abstraction of syntactic and semantic structures applies on sets of predicates and semantic structures representing, as usual in abstract interpretation, properties respectively of predicates and semantics. In the following $\mathcal{A} = \langle \mathfrak{D}_\mathcal{A}, \Psi_\mathcal{A}, [\![\cdot]\!], \perp, \top, \wedge, \vee, \neg \rangle$ is an effective Boolean Algebra.

DEFINITION 3.1 (Semantic abstraction). *Let $\mathcal{A}$ be an effective Boolean Algebra and $\rho \in uco(\wp(\mathfrak{D}_\mathcal{A}))$ be a partitioning abstraction of its domain of denotations. The semantic abstraction of $\mathcal{A}$ w.r.t. ρ, denoted $\langle\!|\rho|\!\rangle$-abstraction, is the effective Boolean algebra*

$$\mathcal{A}^\rho = \langle \mathfrak{D}^\rho_\mathcal{A}, \Psi_\mathcal{A}, [\![\cdot]\!]^\rho, \rho(\perp), \top, \wedge, \vee^\rho, \neg^\rho \rangle$$

where:

$$\mathfrak{D}^\rho_\mathcal{A} = \bigcup \{ \rho(d) \mid d \in \mathfrak{D}_\mathcal{A} \}$$

$$[\![\cdot]\!]^\rho : \Psi_\mathcal{A} \longrightarrow \wp(\mathfrak{D}^\rho_\mathcal{A}) \text{ such that } [\![\varphi]\!]^\rho = \rho([\![\varphi]\!]) = \bigcup \{ \rho(d) \mid d \in [\![\varphi]\!] \}$$

$$\varphi_1, \varphi_2 \in \Psi_\mathcal{A} : [\![\varphi_1 \vee^\rho \varphi_2]\!]^\rho = [\![\varphi_1]\!]^\rho \cup [\![\varphi_2]\!]^\rho$$

$$\varphi \in \Psi_\mathcal{A} : [\![\neg^\rho \varphi]\!]^\rho = \mathfrak{D}^\rho_\mathcal{A} \smallsetminus [\![\varphi]\!]^\rho$$

Before abstracting predicates, i.e., syntax, we have to guarantee the effectiveness of symbolic computation in the SFA. Next lemma proves that if S is a set, whenever $\eta \in uco(\wp(S))$ is additive η maps any r.e. subset X of S into a r.e. (abstract) subset $\eta(X)$ of S.

LEMMA 3.2. *If $X \subseteq S$ is r.e. and $\eta \in uco(\wp(S))$ is additive, then $\eta(X)$ is r.e., namely $\eta \in uco(\wp^{\mathtt{re}}(S))$.*

By Lemma 3.2, because η is a recursive function, and by Kleene's characterization of recursive enumerable sets, the range of η over r.e. sets is itself r.e. (see [21]).

THEOREM 3.3. *If S is a set and $\eta \in uco(\wp^{\mathtt{re}}(S))$ is additive then $\{ \eta(X) \mid X \subseteq S \wedge X \text{ is r.e.} \}$ is r.e.*

DEFINITION 3.4 (Syntactic abstraction). *Let $\mathcal{A}$ be an effective Boolean Algebra and let $\eta \in uco(\wp^{\mathtt{re}}(\Psi_\mathcal{A}))$ be an additive abstraction of predicates. The syntactic abstraction of $\mathcal{A}$ w.r.t. η, denoted $\langle\eta\rangle$-abstraction, is the effective Boolean algebra*

$$\mathcal{A}_\eta = \langle \mathfrak{D}_\mathcal{A}, \eta(\wp^{\mathtt{re}}(\Psi_\mathcal{A})), [\![\cdot]\!], \perp, \top, \wedge, \vee, \neg \rangle$$

where $[\![\cdot]\!] : \eta(\wp^{\mathtt{re}}(\Psi_\mathcal{A})) \longrightarrow \wp(\mathfrak{D}_\mathcal{A})$ is defined as in SFA.

If we have both a $\langle\!|\rho|\!\rangle$-abstraction and a $\langle\eta\rangle$-abstraction of an effective Boolean algebra $\mathcal{A}$, then we define the combined abstraction $\langle\!|\rho|\!\rangle\langle\eta\rangle$-abstraction of $\mathcal{A}$ by combining them as follows. Let $\rho \in uco(\wp(\mathfrak{D}_\mathcal{A}))$ and $\eta \in uco(\wp^{\mathtt{re}}(\Psi_\mathcal{A}))$. The abstraction of $\mathcal{A}$ w.r.t. ρ and η is the effective Boolean algebra

$$\mathcal{A}^\rho_\eta = \langle \mathfrak{D}^\rho_\mathcal{A}, \eta(\wp^{\mathtt{re}}(\Psi_\mathcal{A})), [\![\cdot]\!]^\rho, \rho(\perp), \top, \wedge, \vee^\rho, \neg^\rho \rangle$$

It is clear that $\mathcal{A}_\eta = \mathcal{A}^{\mathtt{id}}_\eta$ and $\mathcal{A}^\rho = \mathcal{A}^\rho_{\mathtt{id}}$. In the following of the paper we assume that $\langle\!|\rho|\!\rangle$- and $\langle\eta\rangle$-abstractions satisfy the hypothesis in Definition 3.1 and 3.4 respectively.

THEOREM 3.5. *If $\mathcal{A}$ is decidable then for any $\rho \in uco(\wp(\mathfrak{D}_\mathcal{A}))$ and $\eta \in uco(\wp^{\mathtt{re}}(\Psi_\mathcal{A}))$, $\mathcal{A}^\rho_\eta$ is decidable.*

Note that, in the definition of symbolic automata there is a strong relation in the underlying effective Boolean algebra $\mathcal{A}$ between the domain of denotations $\mathfrak{D}_\mathcal{A}$ and the set of predicates $\Psi_\mathcal{A}$ used to symbolically represent them. This means that, if we abstract the domain of denotations by considering $\rho \in uco(\wp(\mathfrak{D}_\mathcal{A}))$, leaving unchanged $\Psi_\mathcal{A}$ we are implicitly changing the interpretation of predicates in $\mathfrak{D}_\mathcal{A}$. On the other hand, if we abstract the predicates by considering $\eta \in uco(\wp^{\mathtt{re}}(\Psi_\mathcal{A}))$ we explicitly describe how symbols are abstracted and the semantics is simply the collection of all the semantics denoting the same abstracted predicate. This leads to the following notion of *compatible abstractions*.

3.2 Compatible syntactic and semantic abstractions

Let us consider a $\langle\!|\rho|\!\rangle$-abstraction of $\mathcal{A}$, we aim at characterizing the syntactic abstractions that produce abstract predicates which may have semantics in $\mathfrak{D}^\rho_\mathcal{A}$. This is captured by the notion of $\langle\!|\rho|\!\rangle$-compatibility of a syntactic abstraction. Any semantic $\langle\!|\rho|\!\rangle$-abstraction naturally induces a corresponding syntactic $\langle\Omega(\rho)\rangle$-abstraction with $\Omega(\rho) \in uco(\wp(\Psi_\mathcal{A}))$ defined as follows:

$$\Omega(\rho) \overset{\text{def}}{=} \lambda\Phi.\ \bigcup \{ \Phi' \mid [\![\Phi']\!] \subseteq [\![\Phi]\!]^\rho \}$$

Analogously, any syntactic $\langle\eta\rangle$-abstraction naturally induces a corresponding semantic $\langle\!|\mho(\eta)|\!\rangle$-abstraction with $\mho(\eta) \in uco(\wp(\mathfrak{D}^\rho_\mathcal{A}))$. In order to characterize when and how a syntactic abstraction induces a semantic abstraction, we need to characterize the syntactic abstraction that precisely corresponds to the semantics $[\![\cdot]\!]$, namely the abstraction collecting all the predicates having the same semantics $[\![\cdot]\!]$. This is precisely $\Omega(\mathtt{id})$, which can be rewritten as $\lambda\Phi.\ [\![[\![\Phi]\!]]\!]^+$. Here $[\![\cdot]\!]^+$ is the *adjoint semantic function* defined as follows:

$$[\![\cdot]\!]^+ \overset{\text{def}}{=} \lambda X \in \wp(\mathfrak{D}_\mathcal{A}).\ \bigcup \{ \Phi \mid [\![\Phi]\!] \subseteq X \}$$

Then we can define the induced $\langle\!|\mho(\eta)|\!\rangle$-abstraction:

$$\mho(\eta) \overset{\text{def}}{=} \lambda X. \bigcup \{\, Y \mid \; [\![Y]\!]^+ \subseteq \eta([\![X]\!]^+) \;\}$$

Observe that, when $[\![\cdot]\!] : \Psi_{\mathcal{A}} \to \wp(\mathfrak{D}_{\mathcal{A}})$ is surjective, namely when there exists at least one predicate for each possible semantics in $\wp(\mathfrak{D}_{\mathcal{A}})$ we have that $\mho(\mathtt{id}) = \mathtt{id}$. Indeed, $\mathtt{id} \in \mathit{uco}(\wp^{\mathtt{re}}(\Psi_{\mathcal{A}}))$ considers every single predicate and we have a predicate for each semantic object so in this case we have no effects on the semantics and $\mho(\mathtt{id})$ return precisely the identity on the semantics.

Compatibility of a $\langle\eta\rangle$-abstraction w.r.t. $\langle\!|\rho|\!\rangle$-abstraction can therefore be defined in terms of relative abstraction of η and $\Omega(\rho)$, or analogously, in terms of relative abstraction of ρ and $\mho(\eta)$.

DEFINITION 3.6 (Semantic compatibility). *Given a $\langle\!|\rho|\!\rangle$-abstracted effective Boolean algebra $\mathcal{A}^\rho$ and a syntactic abstraction $\eta \in \mathit{uco}(\wp^{\mathtt{re}}(\Psi_{\mathcal{A}}))$, η is $\langle\!|\rho|\!\rangle$-compatible if:*

$$\eta \sqsubseteq \Omega(\rho) \tag{1}$$

Intuitively we have semantic compatibility when the syntactic abstraction is more concrete than the semantic abstraction, when they are compared on the domain of abstractions of predicates. Indeed, semantic compatibility means that the way a syntactic abstraction η partitions the set of predicates of $\mathcal{A}$ is a refinement of the partition induced by the syntactic abstraction $\Omega(\rho)$ that corresponds to the semantic abstraction ρ. We can say that when we have semantic compatibility the abstraction of the syntax distinguishes programs with the same abstract semantics, namely the abstract program provides an under-approximation of the abstract program behavior.

THEOREM 3.7. *Let $\rho \in \mathit{uco}(\wp(\mathfrak{D}_{\mathcal{A}}))$, then $\Omega(\rho)$ is the most abstract syntactic abstraction $\langle\!|\rho|\!\rangle$-compatible.*

Note that $\mathcal{A}^{\Omega(\rho)}$ may not be an effective Boolean algebra because $\Omega(\rho)(\wp^{\mathtt{re}}(\Psi_{\mathcal{A}}))$ may not be a r.e. set.

EXAMPLE 3.8. *Consider the domains depicted in Fig. 1 (the missing point labels are the set union of smaller elements). The first three domains on the left represent possible syntactic abstractions of $\wp(\Psi_{\mathcal{A}})$, where*

$$\Psi_{\mathcal{A}} \overset{\mathit{def}}{=} \{x+y>3, x \geq 3, y \geq 0, x+y>3 \wedge x \geq 3 \wedge y \geq 0\}.$$

The last domain on the right represents possible semantic abstractions of $\wp(\mathfrak{D}_{\mathcal{A}})$, where

$$\mathfrak{D}_{\mathcal{A}} \overset{\mathit{def}}{=} \{[\![x+y>3]\!], [\![x \geq 3]\!], [\![y \geq 0]\!]\}.$$

Consider for instance the semantic abstraction ρ of $\wp(\mathfrak{D}_{\mathcal{A}})$, depicted with circles on the last domain on the right. The corresponding syntactic abstraction $\Omega(\rho)$ is depicted on the three syntactic domain on the left. Considering the closures depicted on the first domain on the left we observe that the closure $\eta_1 \in \wp(\Psi_{\mathcal{A}})$ is $\langle\!|\rho|\!\rangle$-compatible being more concrete that $\Omega(\rho)$. This means that the syntactic abstraction can distinguish predicates with the same abstract semantics. In particular, while $\rho([\![x+y>3]\!]) = \rho([\![x \geq 3]\!])$ we have that $\eta_1(x+y>3) = \{x+y>3, x \geq 3\}$ while $\eta_1(x \geq 3) = \{x \geq 3\}$.

Now consider a $\langle\eta\rangle$-compatible abstraction of $\mathcal{A}$. We introduce the notion of $\langle\eta\rangle$-compatibility of a semantic abstraction.

DEFINITION 3.9 (Syntactic compatibility). *A semantic abstraction $\rho \in \mathit{uco}(\wp(\mathfrak{D}_{\mathcal{A}}))$ is $\langle\eta\rangle$-compatible for a syntactic $\langle\eta\rangle$-abstraction $\mathcal{A}_\eta$ if:*

$$\eta \sqsupseteq \Omega(\rho) \tag{2}$$

Intuitively we have syntactic compatibility when the syntactic abstraction is more abstract than the semantic abstraction when they are compared on the domain of abstractions of predicates.

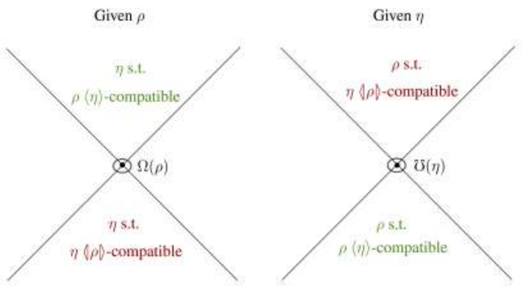

Given ρ
η s.t.
ρ $\langle\eta\rangle$-compatible
$\Omega(\rho)$
η s.t.
η $\langle\!|\rho|\!\rangle$-compatible
Given η
ρ s.t.
η $\langle\!|\rho|\!\rangle$-compatible
$\mho(\eta)$
ρ s.t.
ρ $\langle\eta\rangle$-compatible

Figure 2. Relation between compatibilities.

Indeed, syntactic compatibility means that the semantic abstraction ρ corresponds to a syntactic abstraction $\Omega(\rho)$ and that the partition on the set of predicates of $\mathcal{A}$ induced by $\Omega(\rho)$ is a refinement of the partition induced by η. In other words, when we have syntactic compatibility the abstraction η of the syntax collapses programs with different abstract semantics ρ, hence capturing behaviors that, according to ρ, are not related with the program to analyze, yet providing an over-approximation of the abstract program behavior.

THEOREM 3.10. *Let $\eta \in \mathit{uco}(\wp^{\mathtt{re}}(\Psi_{\mathcal{A}}))$, then $\mho(\eta)$ is the most concrete semantic abstraction $\langle\eta\rangle$-compatible.*

EXAMPLE 3.11. *Consider again the example in Fig. 1 introduced in Example 3.8. Consider in this case the syntactic abstraction η_3 depicted on the third domain . We observe that ρ is $\langle\eta_3\rangle$-compatible since η_3 is more abstract than $\Omega(\rho)$. This means that η_3 induces a further semantic abstraction collapsing elements with different ρ abstract semantics. In particular, $\rho([\![x+y>3 \wedge x \geq 3 \wedge x \geq 3]\!]) \neq \rho([\![x \geq 3]\!])$ while $\eta_3(x+y>3 \wedge x \geq 3 \wedge y \geq 0) = \eta_3(x \geq 3) = \top$. In this example we can also observe a syntactic abstraction η_2 (depicted on the second domain) which fails both the compatibilities since it not comparable with $\Omega(\rho)$.*

Finally, we show when a syntactic abstraction does induce an abstraction of the semantic denotations and vice versa.

LEMMA 3.12. *Let $\eta \in \mathit{uco}(\wp^{\mathtt{re}}(\Psi_{\mathcal{A}}))$:*

1. $\eta \sqsupseteq \Omega(\mathtt{id})$ *iff* $\forall \Phi \in \wp^{\mathtt{re}}(\Psi_{\mathcal{A}}).\ [\![[\![\eta(\Phi)]\!]]\!]^+ = \eta(\Phi)$
2. $\eta \sqsubseteq \Omega(\mathtt{id})$ *iff* $\forall \Phi \in \wp^{\mathtt{re}}(\Psi_{\mathcal{A}}).\ \eta([\![[\![\Phi]\!]]\!]^+) = [\![[\![\Phi]\!]]\!]^+$

THEOREM 3.13. *Let $\eta \in \mathit{uco}(\wp^{\mathtt{re}}(\Psi_{\mathcal{A}}))$, then*

$$\eta \sqsubseteq \Omega(\mathtt{id}) \;\Rightarrow\; \mho(\eta) = \mathtt{id}$$

This result tells us that when we have a syntactic abstraction distinguishing predicates with the same semantics, then we cannot abstract the semantics.

We prove that we can characterize compatibilities both in the domain of semantic abstractions and in the domain of syntactic abstractions.

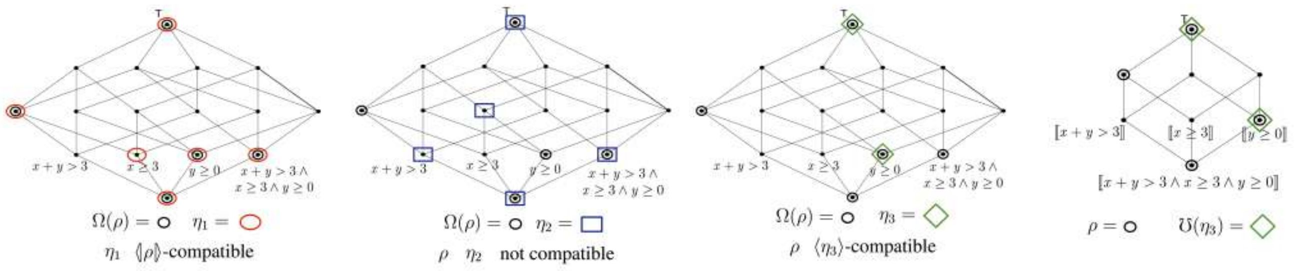
THEOREM 3.14. *Let $\eta \in \mathit{uco}(\wp^{\mathtt{re}}(\Psi_{\mathcal{A}}))$ be such that $\eta \sqsupseteq \Omega(\mathtt{id})$, and $\rho \in \mathit{uco}(\wp(\mathfrak{D}_{\mathcal{A}}))$:*

$$\Omega(\rho) \sqsubseteq \eta \qquad \mathit{iff} \qquad \rho \sqsubseteq \mho(\eta)$$

In Fig. 2 we can see the relation between the two compatibilities. In particular we observe that the two transformers, form syntax to semantics and viceversa, show a relation similar to an adjunction, as observed in the following result.

PROPOSITION 3.15. *Let $\eta \in \mathit{uco}(\wp^{\mathtt{re}}(\Psi_{\mathcal{A}}))$ and $\rho \in \mathit{uco}(\wp(\mathfrak{D}_{\mathcal{A}}))$ the following conditions holds:*

$$(1)\ \mho(\Omega(\rho)) \sqsupseteq \rho \qquad\qquad (2)\ \Omega(\mho(\eta)) \sqsubseteq \eta.$$



⊤
$x + y > 3$
$x \geq 3$
$y \geq 0$
$x + y > 3 \wedge x \geq 3 \wedge y \geq 0$
$\Omega(\rho) = \circ$ $\eta_1 = \bigcirc$
η_1 $\langle\!|\rho|\!\rangle$-compatible
⊤
$x + y > 3$
$x \geq 3$
$y \geq 0$
$x + y > 3 \wedge x \geq 3 \wedge y \geq 0$
$\Omega(\rho) = \circ$ $\eta_2 = \square$
ρ η_2 not compatible
⊤
$x + y > 3$
$x \geq 3$
$y \geq 0$
$x + y > 3 \wedge x \geq 3 \wedge y \geq 0$
$\Omega(\rho) = \circ$ $\eta_3 = \diamond$
ρ $\langle\eta_3\rangle$-compatible
⊤
$[\![x + y > 3]\!]$
$[\![x \geq 3]\!]$
$[\![y \geq 0]\!]$
$[\![x + y > 3 \wedge x \geq 3 \wedge y \geq 0]\!]$
$\rho = \circ$ $\mho(\eta_3) = \diamond$

Figure 1. Compatible abstractions.

EXAMPLE 3.16. *Consider again the example in Fig. 1. For η_3 which satisfies the hypotheses of Th. 3.14, we have a corresponding semantic abstraction $\mho(\eta_3)$ (depicted on the right) which is indeed more abstract than ρ.*

As a corollary of the previous results we show when a $\langle\!|\rho|\!\rangle\langle\eta\rangle$-abstraction of $\mathcal{A}$ satisfies both the compatibilities. The computational cost of making analyses compatible is still to be explored.

PROPOSITION 3.17. *Let $\eta \in uco(\wp^{\mathtt{re}}(\Psi_{\mathcal{A}}))$ and $\rho \in uco(\wp(\mathfrak{D}_{\mathcal{A}}))$ such that $\Omega(\rho) \in uco(\wp^{\mathtt{re}}(\Psi_{\mathcal{A}}))$, the following facts are equivalent:*

1. *η is $\langle\!|\rho|\!\rangle$-compatible and ρ is $\langle\eta\rangle$-compatible;*
2. *$\eta = \Omega(\rho)$;*
3. *$\rho = \mho(\eta)$.*

3.3 Abstracting symbolic automata

Consider a SFA $M = \langle \mathcal{A}, Q, q_0, F, \Delta\rangle$ and the $\langle\!|\rho|\!\rangle\langle\eta\rangle$-abstraction of the effective Boolean algebra $\mathcal{A}$, denoted as $\mathcal{A}^{\rho}_{\eta}$. We define the symbolic finite automaton corresponding to M on the abstract effective Boolean algebra $\mathcal{A}^{\rho}_{\eta}$ as $M^{\rho}_{\eta} \overset{\text{def}}{=} \langle \mathcal{A}^{\rho}_{\eta}, Q, q_0, F, \Delta_{\eta}\rangle$ where:

$$\Delta_{\eta} \overset{\text{def}}{=} \left\{ \, (q, \eta(\varphi), q') \;\middle|\; (q, \varphi, q') \in \Delta \, \right\}$$

Note that $M_{\eta} = M^{\mathtt{id}}_{\eta}$ and $M^{\rho} = M^{\rho}_{\mathtt{id}}$. In the following we prove that when abstracting the underling effective Boolean algebra of an SFA we over-approximate the recognized language, providing a sound approximation in the sense of abstract interpretation.

THEOREM 3.18. *Given a SFA $M = \langle \mathcal{A}, Q, q_0, F, \Delta\rangle$, two closures $\eta \in uco(\wp^{\mathtt{re}}(\Psi_{\mathcal{A}}))$ and $\rho \in uco(\wp(\mathfrak{D}_{\mathcal{A}}))$, the abstract effective Boolean algebra $\mathcal{A}^{\rho}_{\eta}$ and the corresponding SFA $M^{\rho}_{\eta} = \langle \mathcal{A}^{\rho}_{\eta}, Q, q_0, F, \Delta_{\eta}\rangle$. Then: $\mathscr{L}(M) \subseteq \mathscr{L}(M^{\rho}_{\eta})$.*

For this reason is the following we abuse terminology and refer to the SFA whose underlying Boolean algebra is an $\langle\!|\rho|\!\rangle\langle\eta\rangle$-abstraction of a Boolean algebra $\mathcal{A}$ as an $\langle\!|\rho|\!\rangle\langle\eta\rangle$-abstract SFA. Moreover, we can observe that given two abstract Boolean algebra $\mathcal{A}^{\rho_1}_{\eta_1}$ and $\mathcal{A}^{\rho_2}_{\eta_2}$ and an SFA M on $\mathcal{A}$, then the relation between the languages recognized respectively by $M^{\rho_1}_{\eta_1}$ and by $M^{\rho_2}_{\eta_2}$ corresponds to the relation existing between the best correct approximation of the semantics $[\![\cdot]\!]$ with respect to the pair of abstractions ρ_1, η_1 and ρ_2, η_2. This is formally stated in the following Proposition.

PROPOSITION 3.19. *Consider a SFA $M = \langle \mathcal{A}, Q, q_0, F, \Delta\rangle$, the closures $\eta_1, \eta_2 \in uco(\wp^{\mathtt{re}}(\Psi_{\mathcal{A}}))$ and $\rho_1, \rho_2 \in uco(\wp(\mathfrak{D}_{\mathcal{A}}))$, then:*

$$\begin{array}{rcl} \mathscr{L}(M^{\rho_1}_{\eta_1}) \subseteq \mathscr{L}(M^{\rho_2}_{\eta_2}) & \Leftrightarrow & \rho_1 \circ [\![\cdot]\!] \circ \eta_1 \sqsubseteq \rho_2 \circ [\![\cdot]\!] \circ \eta_2 \\ & \Leftarrow & \rho_1 \sqsubseteq \rho_2 \,\wedge\, \eta_1 \sqsubseteq \eta_2 \end{array}$$

4. Minterms

A notion which plays a central role in our transformations of SFA is the notion of *minterm*. This notion has been introduced in [9] for

```
MINTERMS$_{\mathcal{A}}(\Phi) \overset{\text{def}}{=}$
    $tree$ := new $Tree(\top_{\mathcal{A}}$, null, null$)$;
    foreach $\varphi$ in $\Phi$ $tree.Refine(\varphi)$;
        return Leaves$(tree)$;
      //The minterms are the leaf predicates
class $Tree$
    Predicate $\psi$; Tree $left$; Tree $right$;
    $Refine(\varphi) \overset{\text{def}}{=}$
    if ($IsSat_{\mathcal{A}}(\psi \wedge \varphi)$ and $IsSat_{\mathcal{A}}(\psi \wedge \neg\varphi)$)
        if ($left$ = null)      // If the tree is a leaf then split $\psi$
            $left$ := new $Tree(\psi \wedge \varphi$, null, null$)$;
            $right$ := new $Tree(\psi \wedge \neg\varphi$, null, null$)$;
         else $left.Refine(\varphi)$; $right.Refine(\varphi)$;
```

Figure 3. Minterm generation algorithm.

providing a minimal and univocal representation of the predicates in a given set of predicates, e.g., the guards of a given program. In this context we observe some peculiar properties of minterms which make them powerful tools for reasoning on semantics in a syntactic way. A minterm is a minimal satisfiable boolean combination of all predicates occurring in a given SFA. Minterms can be generated from a set of predicates by the algorithm proposed in [9] and reported in Fig. 3. As observed in [9] the set of minterms of an SFA may be expensive to compute, indeed in the worst case the complexity of the algorithm that computes the minterms is exponential in the number of guards of the SFA.

4.1 Basic properties of Minterms

The minterm generation for a formula φ produces a tree T_{φ} that satisfies the following basic properties.

PROPOSITION 4.1. *Let* tree *be the tree built during the minterm generation, starting from a set $\Phi \in \wp^{\mathtt{re}}(\Psi_{\mathcal{A}})$ of predicates. Given $\varphi \in \Psi_{\mathcal{A}}$, let us denote by T_{φ} the subtree of* tree *having φ as root. Then the following properties hold:*

1. *Let $Leaves(T_{\varphi}) = \{\varphi_1, \ldots, \varphi_k\}$, then $\varphi \Leftrightarrow \bigvee_{i \in \{1..k\}} \varphi_i$;*
2. *Any $\varphi \in$ MINTERMS(Φ) φ satisfiable implies that for all $\varphi' \in$ MINTERMS$(\Phi) \smallsetminus \{\varphi\}$ is not satisfiable.*
3. *For all $\varphi_1, \varphi_2 \in \Phi$ we have that $\varphi_1 \wedge \varphi_2$ is satisfiable iff $Leaves(T_{\varphi_1}) \cap Leaves(T_{\varphi_2}) \neq \varnothing$;*
4. *For any $\varphi_1, \varphi_2 \in \Phi$ we have that $\varphi_1 \Rightarrow \varphi_2$ is satisfiable with φ_1 satisfiable iff $Leaves(T_{\varphi_1})_{\mathrm{SAT}} \subseteq Leaves(T_{\varphi_2})$[1];*

The following proposition shows that the semantics of minterms is a partition of the domain $\mathfrak{D}_{\mathcal{A}}$ of denotations.

[1] where Leaves$(T_{\varphi_1})_{\mathrm{SAT}} \overset{\text{def}}{=} \left\{ \varphi \in \text{Leaves}(T_{\varphi_1}) \;\middle|\; \varphi \text{ is satisfiable } \right\}$.

PROPOSITION 4.2. *Let $\mathcal{A} = \langle \mathfrak{D}_\mathcal{A}, \Psi_\mathcal{A}, [\![\cdot]\!], \bot, \top, \wedge, \vee, \neg \rangle$ be an effective Boolean algebra, then $\{ [\![\varphi]\!] \mid \varphi \in \textsc{MinTerms}(\Psi_\mathcal{A}) \}$ is a partition of $\mathfrak{D}_\mathcal{A}$.*

4.2 Approximated Minterms

Minterms change their structure when the underlying Boolean algebra is approximated by abstract interpretation. We consider an effective Boolean algebra $\mathcal{A} = \langle \mathfrak{D}_\mathcal{A}, \Psi_\mathcal{A}, [\![\cdot]\!], \bot, \top, \wedge, \vee, \neg \rangle$, where the semantic function $[\![\cdot]\!] : \Psi_\mathcal{A} \to \wp(\mathfrak{D}_\mathcal{A})$ is surjective. Consider a subset $\Psi \subseteq \Psi_\mathcal{A}$ of such predicates, for example the set of predicates that label a given SFA. We define the syntactic abstraction $\eta_\Psi \in uco(\wp^{\mathtt{re}}(\Psi_\mathcal{A}))$ as that abstraction of predicates that observes precisely only the predicates in Ψ and abstract in $\top$ any other predicate. Let $\varphi \in \Psi_\mathcal{A}$, then η_Ψ is formally defined as additive lift of:

$$\eta_\Psi(\{\varphi\}) \overset{\text{def}}{=} \begin{cases} \{\varphi\} & \text{if } \varphi \in \Psi \\ \top & \text{otherwise} \end{cases}$$

Note that the fixpoints of η_Ψ is $\eta_\Psi(\wp^{\mathtt{re}}(\Psi_\mathcal{A})) = \wp(\Psi) \cup \{\top\}$. Of course η_Ψ corresponds to an abstraction $\mho(\eta_\Psi)$ on the semantics that precisely observes only the semantics of the predicates in Ψ, as stated by the following result.

LEMMA 4.3. *Let $\mathcal{A} = \langle \mathfrak{D}_\mathcal{A}, \Psi_\mathcal{A}, [\![\cdot]\!], \bot, \top, \wedge, \vee, \neg \rangle$ be an effective Boolean algebra and consider $\eta_\Psi \in uco(\wp^{\mathtt{re}}(\Psi_\mathcal{A}))$ which is $\langle\!|\mathtt{id}|\!\rangle$-compatible, then:*

$$\mho(\eta_\Psi)(\wp(\mathfrak{D}_\mathcal{A})) = \{ [\![\Phi]\!] \mid \Phi \in \wp^{\mathtt{re}}(\Psi_\mathcal{A}) \}$$

Observe that η_Ψ is $\langle\!|\mathtt{id}|\!\rangle$-compatible if whenever there is a predicate in Ψ then Ψ contains also all the predicates with the same semantics. The closure $\mho(\eta_\Psi) \in uco(\wp(\mathfrak{D}_\mathcal{A}))$ may not be partitioning in general, so we consider $\Pi(\mho(\eta_\Psi))$ and we observe that the equivalence classes of the partition induced by $\Pi(\mho(\eta_\Psi))$ on $\mathfrak{D}_\mathcal{A}$ are precisely the semantics of the minterms of Ψ.

PROPOSITION 4.4. *Let $\mathcal{A} = \langle \mathfrak{D}_\mathcal{A}, \Psi_\mathcal{A}, [\![\cdot]\!], \bot, \top, \wedge, \vee, \neg \rangle$ be an effective Boolean algebra, and consider $\Psi \subseteq \Psi_\mathcal{A}$ such that the abstraction $\eta_\Psi \in uco(\wp^{\mathtt{re}}(\Psi_\mathcal{A}))$ is $\langle\!|\mathtt{id}|\!\rangle$-compatible, then:*

$$\{ [\![\varphi]\!] \mid \varphi \in \textsc{MinTerms}(\Psi) \} = \{ \Pi(\mho(\eta_\Psi))(d) \mid d \in \mathfrak{D}_\mathcal{A} \}$$

It is now interesting to observe what happens when we consider a generic syntactic abstraction $\eta \in uco(\wp^{\mathtt{re}}(\Psi_\mathcal{A}))$ such that $\eta_\Psi \sqsubseteq \eta$, namely that further abstracts the set of predicates Ψ that we are considering. In this case, the semantics of the minterms of the approximated predicates $\eta(\Psi)$ are precisely given by the abstraction $\Pi(\mho(\eta))$ of the semantics of the minterms of Ψ.

THEOREM 4.5. *Let $\mathcal{A} = \langle \mathfrak{D}_\mathcal{A}, \Psi_\mathcal{A}, [\![\cdot]\!], \bot, \top, \wedge, \vee, \neg \rangle$ be an effective Boolean algebra, and consider $\Psi \subseteq \Psi_\mathcal{A}$ such that the abstraction $\eta_\Psi \in uco(\wp^{\mathtt{re}}(\Psi_\mathcal{A}))$ is $\langle\!|\mathtt{id}|\!\rangle$-compatible, and an abstraction $\eta \in uco(\wp^{\mathtt{re}}(\Psi_\mathcal{A}))$ such that $\eta_\Psi \sqsubseteq \eta$. Then:*

$$\{ [\![\varphi]\!] \mid \varphi \in \textsc{MinTerms}(\eta(\Psi)) \} = \\ \{ \Pi(\mho(\eta))([\![\varphi]\!]) \mid \varphi \in \textsc{MinTerms}(\Psi) \}$$

This means that the semantics of the minterms of a set of abstract predicates is precisely the abstraction of the semantics of the original predicates.

EXAMPLE 4.6. *Let $\mathcal{A} = \langle \mathfrak{D}_\mathcal{A}, \Psi_\mathcal{A}, [\![\cdot]\!], \bot, \top, \wedge, \vee, \neg \rangle$ be an effective Boolean algebra where $\Psi_\mathcal{A} = \{ x \in N \mid N \subseteq \mathbb{Z} \}$, and the semantic function $[\![\cdot]\!] : \Psi_\mathcal{A} \to \wp(\mathbb{Z})$ is naturally defined as $[\![x \in N]\!] = N$.*
Let us consider the following subset of $\Psi_\mathcal{A}$:

$$\Psi = \{ x \in \{4,6\}, x \in \{5,6\}, x \in \{-5\}, x \in \{-8\} \}$$

the corresponding set of minterms is $\textsc{MinTerms}(\Psi)$:

$$\begin{aligned}
&\{ (x \in \{4,6\} \wedge x \in \{5,6\}), \\
&(x \in \{4,6\} \wedge \neg x \in \{5,6\}), \\
&(\neg x \in \{4,6\} \wedge x \in \{5,6\}), \\
&(\neg x \in \{4,6\} \wedge \neg x \in \{5,6\} \wedge x \in \{-5\}), \\
&(\neg x \in \{4,6\} \wedge \neg x \in \{5,6\} \wedge \neg x \in \{-5\} \wedge x \in \{-8\}), \\
&(\neg x \in \{4,6\} \wedge \neg x \in \{5,6\} \wedge \neg x \in \{-5\} \wedge \neg x \in \{-8\})\}
\end{aligned}$$

observe that:

$$\{ [\![\varphi]\!] \mid \varphi \in \textsc{MinTerms}(\Psi) \} = \\ \{ \{4\}, \{6\}, \{5\}, \{-5\}, \{-8\}, \mathbb{Z} \smallsetminus \{4,6,5,-5,-8\} \}$$

the closure $\eta_\Psi \in uco(\wp^{\mathtt{re}}(\Psi_\mathcal{A}))$ is defined as the additive lift of:

$$\eta_\Psi(\{x \in N\}) \overset{\mathit{def}}{=} \begin{cases} \{x \in N\} & \mathit{if}\ \{x \in N\} \in \Psi \\ \top & \mathit{otherwise} \end{cases}$$

and, as states in Proposition 4.4 we have that:

$$\{ \Pi(\mho(\eta_\Psi))(d) \mid d \in \mathbb{Z} \} = \\ \{ \{4\}, \{6\}, \{5\}, \{-5\}, \{-8\}, \mathbb{Z} \smallsetminus \{4,6,5,-5,-8\} \}$$

Let $\mathbb{Z}^+ \overset{\mathit{def}}{=} \{ v \mid v \geq 0 \}$ and $\mathbb{Z}^- \overset{\mathit{def}}{=} \{ v \mid v < 0 \}$ and let the closure $\eta_{Sign} \in uco(\wp^{\mathtt{re}}(\Psi_\mathcal{A}))$ defined as the additive lift of:

$$\eta_{Sign}(\{x \in N\}) \overset{\mathit{def}}{=} \begin{cases} \{x \in \mathbb{Z}^+\} & \mathit{if}\ N \subseteq \mathbb{Z}^+ \\ \{x \in \mathbb{Z}^-\} & \mathit{if}\ N \subseteq \mathbb{Z}^- \\ \top & \mathit{otherwise} \end{cases}$$

Observe that the $\textsc{MinTerms}(\eta_{Sign}(\Psi))$ is the set

$$\{ \{x \in \mathbb{Z}^+\}, \{x \in \mathbb{Z}^-\} \}$$

and the semantics of the minterms of $\eta_{Sign}(\Psi)$ is:

$$\{ [\![\varphi]\!] \mid \varphi \in \textsc{MinTerms}(\eta_{Sign}(\Psi)) \} \quad = \quad \{\mathbb{Z}^+, \mathbb{Z}^-\}$$

Moreover, as shown in Theorem 4.5:

$$\{ \Pi(\mho(\eta_{Sign}))([\![\varphi]\!]) \mid \varphi \in \textsc{MinTerms}(\Psi) \} = \{\mathbb{Z}^+, \mathbb{Z}^-\}$$

5. Topological SFA abstraction

In Section 3 we have seen how an SFA can be abstracted by abstracting its underlying Boolean algebra. This abstraction does not influence directly the topological structure of SFA. When dealing with automata, the natural way of thinking about automata simplification (or abstraction) is the merge of states. In general, we can define a simplification operation on automata that collapses states wrt a given equivalence relation over states. Namely, the equivalence relation establish the criteria that the simplification uses for merging states.

DEFINITION 5.1. *Consider a SFA $M = \langle \mathcal{A}, Q, q_0, F, \Delta \rangle$ and an equivalence relation $R \subseteq Q \times Q$ over its states. We denote with $Sim_R(M)$ the SFA obtained by simplifying M wrt R, namely the SFA computed as the quotient of M wrt R, i.e., $Sim_R(M) = M_{/R}$.*

Thus, SFA simplification is the operation of quotient made parametric on the equivalence relation used to merge states. It is easy to observe that for every equivalence relation R, the SFA $Sim_R(M)$ resulting from SFA simplification recognizes at least the language recognized by M. Indeed when we merge states we keep all the transitions of the original SFA and we may add some new *spurious* ones.

PROPOSITION 5.2. *Consider a SFA $M = \langle \mathcal{A}, Q, q_0, F, \Delta \rangle$. For any equivalence relation $R \subseteq Q \times Q$ we have that $\mathscr{L}(M) \subseteq \mathscr{L}(Sim_R(M))$.*

Given two equivalence relations R and R', we write $R \preceq R'$ when R is a refinement of R'. Of course the coarser is the equivalence relation the wider is the language recognized by the corresponding simplified SFA.

PROPOSITION 5.3. *Consider a SFA $M = \langle \mathcal{A}, Q, q_0, F, \Delta \rangle$ and two equivalence relations $R, R' \subseteq Q \times Q$ such that $R \preceq R'$. Then $\mathscr{L}(Sim_R(M)) \subseteq \mathscr{L}(Sim_{R'}(M))$.*

Another important property of topological abstractions is that they do not change the set of minterms, since they do not change the predicates. In the following we report a simplification algorithm where the predicates of the SFA to simplify are first rewritten as disjunction of minterms (line 3-7). Thus, whenever the equivalence relation R deals with properties of the languages of strings that reaches or starts from a state, it may be easier to check these properties on minterms instead of checking them on the language of denotations. (Examples will be provided in the following).

```
Simplify($M, R$)

1.  Input: $M = \langle \mathcal{A}, Q, q_0, F, \Delta \rangle$, $R \subseteq Q \times Q$,
2.      $\mathcal{A} = \langle \mathfrak{D}_{\mathcal{A}}, \Psi_{\mathcal{A}}, [\![\cdot]\!], \bot, \top, \wedge, \vee, \neg \rangle$
3.  $\mathcal{M}t(M) \overset{\text{def}}{=} \text{MinTerms}\left(\left\{ \psi \;\middle|\; \begin{array}{l} \exists p, q \in Q. \\ p \xrightarrow{\psi} q \in \Delta \end{array} \right\}\right)$
4.  $M' = \langle \mathcal{A}', Q, q_0, F, \Delta' \rangle$:
5.      $\mathcal{A}' \overset{\text{def}}{=} \langle \mathfrak{D}_{\mathcal{A}}, \mathcal{M}t(M), [\![\cdot]\!], \bot, \top, \wedge, \vee, \neg \rangle$
6.      $\mu(\psi) \overset{\text{def}}{=} \bigvee \{ \varphi \in \mathcal{M}t(M) \mid \varphi \in \text{Leaves}(T_\psi) \}$
7.      $\Delta' \overset{\text{def}}{=} \left\{ p \xrightarrow{\mu(\psi)} q \;\middle|\; \exists \psi.\, p \xrightarrow{\psi} q \right\}$
8.  Output: $M'' = M'_{/R}$
```

5.1 Examples of SFA Simplifications

Minimization. D'Antoni and Veanes in [9] have extended the standard algorithm of Hopcroft for finite state automata minimization to SFA. This operation is based on the idea of refining an initial partition by checking all the possible moves depending on the considered alphabet symbol. In FSA this is feasible because they have a finite alphabet. In SFA the alphabet is r.e., hence in general infinite. For this reason the algorithm proposed iterates this check on predicates/symbols in a way that makes the number of possible iteration finite: instead of checking transitions for each alphabet symbol, the check is made for each minterm (see [9] for details).

Observe that this SFA minimization algorithm can be seen as a simplification wrt. the equivalence relation that relates all and only the states that are reached exactly by the same language of minterms. Consider a SFA $M = \langle \mathcal{A}, Q, q_0, F, \Delta \rangle$, and for every $q \xrightarrow{\psi} p \in \Delta$ let $\mu(\psi)$ be the predicate ψ written as a disjunction of minterms (namely as the disjunction of the leaves of the subtree T_ψ with root ψ of the tree generated during the construction of the minterms of the considered SFA). We define the language of strings of minterms that reaches a state q as:

$$\dot{\mathscr{L}}(q) \overset{\text{def}}{=} \left\{ \mu(\psi_1) \dots \mu(\psi_{n-1}) \;\middle|\; \begin{array}{l} \exists n \in \mathbb{N} : \exists q_1 \dots q_n \in Q : \\ \forall i \in [1, n[. q_i \xrightarrow{\mu(\psi_i)} q_{i+1} \in \Delta \\ q_n = q \end{array} \right\}$$

Let $\doteqdot \subseteq Q \times Q$ be such that $q \doteqdot p$ iff $\dot{\mathscr{L}}(q) = \dot{\mathscr{L}}(p)$. Observe that for the properties of minterms proved in the previous section, we have that checking the language of minterms or checking the language of denotations is equivalent, since minterms provide a minimal and unequivocal representation of predicates. Let $Min(M)$ denote the minimization of M.

PROPOSITION 5.4. $Min(M) = Sim_{\doteqdot}(M)$.

k-Minimization. According to the above formalization of SFA minimization, we can weaken minimization by defining a relation over states that observes the language of stings of minterms of a fixed length k that reaches a given state. To this end, given an SFA $M = \langle \mathcal{A}, Q, q_0, F, \Delta \rangle$, and for every $q \xrightarrow{\psi} p \in \Delta$ let $\mu(\psi)$ be the predicate ψ written as a disjuction of minterms, we define the language $\dot{\mathscr{L}}_k(q)$, which is the language of strings of length k that can reach the state q:

$$\dot{\mathscr{L}}_k(q) \overset{\text{def}}{=} \left\{ \mu(\psi_1) \dots \mu(\psi_{k-1}) \;\middle|\; \begin{array}{l} \exists q_1 \dots q_k \in Q : \\ \forall i \in [1, .k[. q_i \xrightarrow{\mu(\psi_i)} q_{i+1} \in \Delta \\ q_k = q \end{array} \right\}$$

Let $\doteqdot_k \subseteq Q \times Q$ be such that $q \doteqdot_k p$ iff $\dot{\mathscr{L}}_k(q) = \dot{\mathscr{L}}_k(p)$. Let $Min_k(M)$ denote the simplification of M wrt $\doteqdot_k$. The following examples illustrate the difference between minimization and k-minimization.

EXAMPLE 5.5. *Consider the SFA M in Fig. 4 on the left. It is clear that the predicates* $x\,$odd *and* $(x+1)\,$even *are equivalent, as well as predicates* $y\,$even *and* $(y+1)\,$odd*. This is captured by the minimization algorithm of D'Antoni and Veanes that correctly collapses state q_4 with q_5 and q_7 with q_8. The minimized algorithm $Min(M)$ is shown in Fig. 4 at the top on the right. Observe that the edge between q_2 and $\{q_4, q_5\}$, as well as the edge between $\{q_7, q_8\}$ and q_9, is labeled by one of the two equivalent predicates. Of course, the SFA M and $Min(M)$ recognize the same language. In order to clarify the difference between minimization and k-minimization at the bottom right of Fig. 4 we report the result obtained by applying the simplification algorithm wrt $\doteqdot_k$ where $k = 1$ at the SFA M. Observe that the simplification algorithm with $k = 1$ merges the state q_6 with the states q_7 and q_8, as shown in the resulting SFA $Min_1(M)$. Indeed, the states q_6, q_7 and q_8 are reached by the same language of strings of length 1 (in this simple case all the denotations with y positive). The edge between $\{q_6, q_7, q_8\}$ and q_9 is labeled by* true *since it corresponds to* $y\,$odd $\vee$ $y\,$even*. We can observe that the language recognized by $Min_1(M)$ is greater than the one recognized by M. Let us consider the pairs (n_1, n_2) with $n_1, n_2 \in \mathbb{Z}$ where the first number denotes the values of x and the second the values of y. For example we have that the string of pairs $(1,2)(2,4)(4,8)(8,16) \in \mathscr{L}(Min_1(M))$ while it does not belong to $\mathscr{L}(M) = \mathscr{L}(Min(M))$.*

Of course when the value of k increases it increases also the precision of the simplification wrt $\doteqdot_k$ by collapsing states that are equivalent, namely at the limit with k increasing the k-minimization becomes minimization.

THEOREM 5.6. *Given two states p and q we have that $p \doteqdot q$ iff $\forall k \in \mathbb{N}.\, p \doteqdot_k q$.*

EXAMPLE 5.7. *Observe that if we compute the simplification of the SFA M in the example in Fig. 4 wrt $\doteqdot_k$ and $k = 2$ we obtain the minimized SFA, namely $Min(M) = Min_2(M)$. Indeed, if we consider the language of words of length 2 that reach a given state we can no longer merge q_6 with q_7 and q_8.*

k-Invariant. Minterms provide a systematic simplification of SFA based on the extraction of invariant properties that hold for the language of strings that reach (or start) from a given state. Consider an SFA $M = \langle \mathcal{A}, Q, q_0, F, \Delta \rangle$, and for every $q \xrightarrow{\psi} p \in \Delta$ let $\mu(\psi)$ be the predicate ψ written as a disjunction of minterms. Consider a state $q \in Q$. For every string $\mu(\psi_1) \dots \mu(\psi_k) \in \dot{\mathscr{L}}_k(q)$ of length k that reaches the state q we have that *IsSat*$(\bigwedge_{i \in [1,k]} \mu(\psi_i))$ is true iff all the disjunctions $\mu(\psi_i)$ of minterms share at least one minterm. This because, thanks to the properties of minterms, only

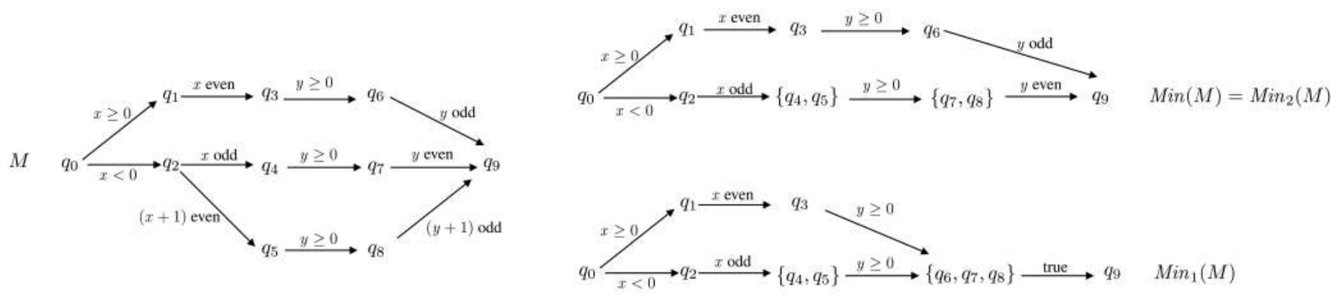


Figure 4. Minimization and k-Minimization

one minterm at the time can be true. Let

$$Inv\{\mu(\psi_i)\}_{i\in[1,k]} \overset{\text{def}}{=} \left\{ \varphi \in \textsc{MinTerms} \;\middle|\; \forall i \in [1,k].\, IsSat(\varphi \wedge \mu(\psi_i)) \right\}$$

It is the set of *all* the minterms shared by *all* the $\mu(\psi_i)$, which provides the invariant property of the corresponding string. Indeed, thanks to minterms this satisfiability can be checked syntactically. We can therefore define the following equivalence relation $\overset{\text{inv}}{\equiv}_k \subseteq Q \times Q$ such that $q \overset{\text{inv}}{\equiv}_k p$ iff $\Im_k(q) = \Im_k(p)$ where

$$\Im_k(q) = \left\{ Inv\{\mu(\psi_i)\}_{i\in[1,k]} \;\middle|\; \mu(\psi_1)\ldots\mu(\psi_k) \in \dot{\mathscr{L}}_k(q) \right\}$$

Thus, $\overset{\text{inv}}{\equiv}_k$ collapses states reached by paths that have the same k-invariant property. We can observe that, if two states share the same k-language then they surely share the same k-invariant, while the opposite may not be true since the language fixes an order in the constraints that the commutativity of the conjunction relaxes.

THEOREM 5.8. *Given two states p and q, and $k \in \mathbb{N}$, we have that $p \dot{\equiv}_k q$ implies $\forall k' \leq k.\, p \overset{inv}{\equiv}_{k'} q$.*

EXAMPLE 5.9. *Consider again the automaton M in Fig. 4. The minterms generated by its predicates are given in the table in Fig. 5: each i denotes the minterm M_i obtained as the conjunction between the constraint on x and on y, for instance 3 stays for the minterm $M_3 = (x$ even $\wedge\, x \geq 0 \wedge y$ even $\wedge\, y < 0)$. In Fig. 5 we rewrite M where on each edge the predicates are denoted as the set of the minterms specifying it. For instance $x \geq 0 \equiv \bigvee_{i\in[1,8]} M_i$. Note that on this automaton the k-invariant generates the same transformation as the k-minimization as showed in Fig. 4. Consider instead the automaton M_1 on the right. In this case, the languages recognized by q_7 and q_8 are different, for instance the trace $(-1,3)(3,-4)(3,5) \in \dot{\mathscr{L}}_3(q_7)$ is not in $\dot{\mathscr{L}}_3(q_8)$ since $(3,-4)$ does not satisfy the predicate between q_2 and q_5 in M_1, i.e., $y \geq 0$. If we consider 3-invariant then we observe that the invariant on the path $q_0q_2q_4q_7$ is $M_{13} \wedge M_{14}$ and the same is for the path $q_0q_2q_5q_8$, hence we can collapse the states q_7 and q_8.*

5.2 Topological abstraction of abstract SFA

It is worth noting that abstraction in SFA may influence the automata simplification. In this section we prove that the efficacy of simplification, and in particular of minimization and k-minimization, in SFA is strictly related with the degree of abstraction of their semantics or syntax.

EXAMPLE 5.10. *Consider the SFA M in Fig. 4 and assume that we want to abstract from the parity of y. Hence we define abstraction η_1 on the predicates of M as $\eta_1(y\,\text{odd}) = \eta_1(y\,\text{even}) = \eta_1((y+1)\,\text{odd}) =$ true and as the identity on the other predicates. In this example we do not abstract the semantics and we consider $\rho =$ `id`. Let M_{η_1} be the SFA wrt the considered abstraction (where the predicates of M are substituted with their abstraction according to η_1). By applying minimization to this SFA we obtain the SFA $Min(M_{\eta_1})$ depicted at the top left of Fig. 6. We can observe that, due to the predicate abstraction η_1, the minimization of M_{η_1} collapses more states than the minimization of M and therefore: $\mathscr{L}(Min(M)) \subseteq \mathscr{L}(Min(M_{\eta_1}))$. For example the string of pairs $(1,2)(2,4)(4,8)(8,16) \in \mathscr{L}(Min(M_{\eta_1}))$ while it does not belong to $\mathscr{L}(Min(M))$.*

We have an analogous situation in the case of k-minimization. Consider the predicate abstraction η_2 such that $\eta_1(x\,\text{odd}) = \eta_1(x\,\text{even}) = \eta_1((x+1)\,\text{even}) =$ true and as the identity on the other predicates, and let $\rho =$ `id`. By applying the simplification algorithm wrt. $\dot{\equiv}_k$ with $k = 1$ to the SFA M_{η_2} we obtain the SFA at the top right of Fig. 6. Also in this case, due to the abstraction η_2 the simplification algorithm collapses more states and therefore: $\mathscr{L}(Min_1(M)) \subseteq \mathscr{L}(Min_1(M_{\eta_2}))$. For example the string of pairs $(1,2)(3,6)(5,10)(7,14) \in \mathscr{L}(Min_1(M_{\eta_2}))$ while it does not belong to $\mathscr{L}(Min_1(M))$.

Let $\mathbb{S}$ denote the set of SFA and let us define the following ordering relation $\dot{\leq}$ on $\mathbb{S}$ modeling precisely the relative precision of SFA with respect to language containment and size of the automaton, where given $M_1 = \langle \mathcal{A}, Q_1, q_0^1, F_1, \Delta_1 \rangle$ and $M_2 = \langle \mathcal{A}, Q_2, q_0^2, F_2, \Delta_2 \rangle \in \mathbb{S}$ we have that:

$$M_1 \dot{\leq} M_2 \Leftrightarrow \mathscr{L}(M_1) \subseteq \mathscr{L}(M_2) \vee \mathscr{L}(M_1) = \mathscr{L}(M_2) \wedge |Q_2| \leq |Q_1|$$

It is immediate to observe that $\langle \mathbb{S}, \dot{\leq} \rangle$ is a possibly non-complete lattice. Given the SFA simplification $Sim_R : \mathbb{S} \to \mathbb{S}$, a SFA $M = \langle \mathcal{A}, Q, q_0, F, \Delta \rangle$ and a $\langle\!|\rho|\!\rangle\langle\eta\rangle$-abstraction of the effective Boolean algebra $\mathcal{A}$ we wonder when the diagram in Fig. 7 commutes. In general we have that when we simplify the SFA after the abstraction of the underlying algebra we obtain an SFA that is more abstract than the one obtained by applying simplification before the abstraction. The intuition beyond this is that the abstraction of the underlying Boolean algebra could make equivalent edges of the original SFA that are not equivalent and this may cause the merge of states that would not be merged when simplifying original SFA.

PROPOSITION 5.11. *Given $M = \langle \mathcal{A}, Q, q_0, F, \Delta \rangle \in \mathbb{S}$, the closures $\eta \in uco(\wp(\Psi_{\mathcal{A}}))$ and $\rho \in uco(\wp(\mathfrak{D}_{\mathcal{A}}))$ and a relation R, we have that: $Sim_R(M)^\rho_\eta \dot{\leq} Sim_R(M^\rho_\eta)$.*

EXAMPLE 5.12. *At the bottom left of Fig. 6 we show the result of abstracting the Boolean algebra after the SFA minimization. We observe that even if $Min(M_{\eta_1})$ and $Min(M)_{\eta_1}$ recognize the same language the automata obtained by minimizing after the abstraction of the underlying Boolean algebra has less states than the one computed by abstracting the Boolean algebra after the minimization. We have a similar result for k-minimization as we*

y \ x	even +	odd +	even -	odd -
even +	1	5	9	13
odd +	2	6	10	14
even -	3	7	11	15
odd -	4	8	12	16

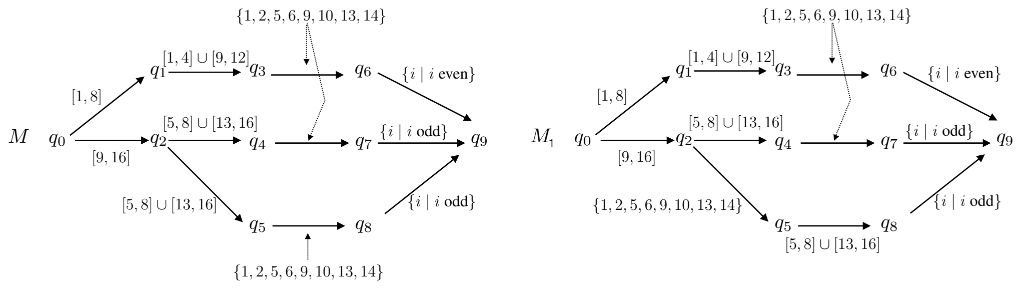
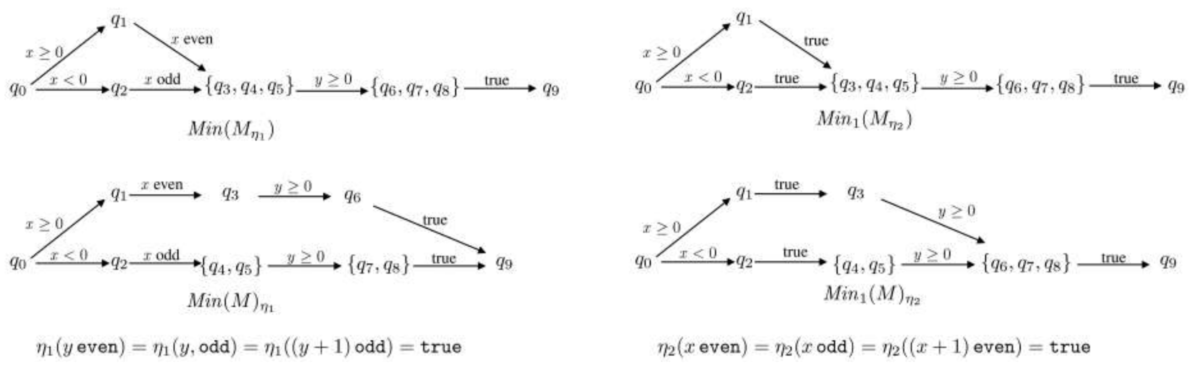


Figure 5. k-Invariant transformation



$x \geq 0$
$x < 0$
x even
x odd
$y \geq 0$
true
q_0 q_1 q_2 q_3 q_6 q_9
$\{q_3, q_4, q_5\}$
$\{q_6, q_7, q_8\}$
$\{q_4, q_5\}$
$\{q_7, q_8\}$
$Min(M_{\eta_1})$
$Min_1(M_{\eta_2})$
$Min(M)_{\eta_1}$
$Min_1(M)_{\eta_2}$
$\eta_1(y\,\texttt{even}) = \eta_1(y, \texttt{odd}) = \eta_1((y+1)\,\texttt{odd}) = \texttt{true}$
$\eta_2(x\,\texttt{even}) = \eta_2(x\,\texttt{odd}) = \eta_2((x+1)\,\texttt{even}) = \texttt{true}$

Figure 6. Minimization and k-Minimization in presence of Abstraction

$$\begin{tikzcd}
M^\rho_\eta \arrow[r, "Sim_R"] & Sim_R(M)^\rho_\eta \dot{\preceq} Sim_R(M^\rho_\eta) \\
M \arrow[u, "\llparenthesis\rho\rrparenthesis\langle\eta\rangle"] \arrow[r, "Sim_R"] & \tau(M) \arrow[u, "\llparenthesis\rho\rrparenthesis\langle\eta\rangle"']
\end{tikzcd}$$

Figure 7. Completeness of SFA simplification

can see by comparing the SFA at the bottom right and top right of Fig. 6.

6. Programs as SFA

In this section we specify the approximate semantics of a program as the language recognized by a SFA. We consider programs in imperative computational model and assume to have access their correct control flow graph (CFG). The CFG of a program is a graph where nodes are given by sequences of non branching instructions. More formally, let $\mathbf{I}$ be the instruction set containing both branching and non-branching instructions. We denote with $\mathbb{I} \subseteq \mathbf{I}$ the set of non-branching instructions and with $\mathbb{C}$ the set of boolean expressions over program states that are guards of the branching instructions. Let $\texttt{c}$ range over $\mathbb{C}$ and $\texttt{b}$ range over $\mathbb{I}^*$. The CFG of a program $P \in \mathbf{I}^*$ is a graph $G_P = (N_P, E_P)$ where the set $N_P \subseteq \mathbb{I}^*$ of nodes specifies the basic blocks of P, namely the maximal sequences of sequential instructions of P, while the set of edges $E_P \subseteq N_P \times \mathbb{C} \times N_P$ denotes the guarded transitions of P. In particular, a labeled edge $(\texttt{b}, \texttt{c}, \texttt{b}') \in E_P$ means that the execution of P flows from $\texttt{b}$ to $\texttt{b}'$ when the execution of $\texttt{b}$ leads to a program state that satisfies condition $\texttt{c}$. When a basic block $\texttt{b}$ has no outgoing edges in E_P we say that it is final, denoted $\texttt{b} \in \mathit{Final}[G_P]$. We denote with $\mathit{in}[\texttt{b}]$ and $\mathit{out}[\texttt{b}]$ respectively the entry and exit point of the basic block $\texttt{b}$, and with $\mathbb{PP}[G_P]$ the block delimiters of G_P, namely the set of all the entry and exit points of the basic blocks of G_P, namely:

$$\mathbb{PP}[G_P] \overset{\text{def}}{=} \left\{\, \mathit{in}[\texttt{b}] \,\middle|\, \texttt{b} \in N_P \,\right\} \cup \left\{\, \mathit{out}[\texttt{b}] \,\middle|\, \texttt{b} \in N_P \,\right\}$$

Let Σ, ranged over by s, be the set of possible program states. Let $\texttt{exec} : \mathbb{I}^* \longrightarrow \wp^{\texttt{re}}(\Sigma \times \Sigma)$ be the function that defines the semantics of basic blocks, namely the pairs of input/output states that model the execution of sequences of instructions. When $(s, s') \in \texttt{exec}(\texttt{b})$ it means that the execution of the sequence of instructions $\texttt{b}$ transforms state s into state s'. Let us denote with $s \models \texttt{c}$ the fact that the boolean condition $\texttt{c}$ is satisfied by state $s \in \Sigma$.

We define the set of executions of the CFG of a program P the sequences of basic blocks and guards that can be encountered along a path of $G_P = (N_P, E_P)$. Formally:

$$\mathit{Exe}[G_P] \overset{\text{def}}{=} \left\{ \texttt{b}_0\texttt{c}_1\texttt{b}_1\texttt{c}_2 \ldots \texttt{c}_k\texttt{b}_k \,\middle|\, \begin{array}{l} \forall 0 \leq i < k : \\ (\texttt{b}_i, \texttt{c}_{i+1}, \texttt{b}_{i+1}) \in E_P \end{array} \right\} \qquad (3)$$

We consider a safety semantics, namely the semantics of all prefixes of execution traces of a given program P [19]. The execution trace semantics of a program P, denoted $\llbracket P \rrbracket$, is therefore the set of all finite executions starting from the entry point of the starting basic block $\texttt{b}_0$ in the CFG G_P of P. Let $\mathit{Init}_P \subseteq \Sigma$ be the set of possible initial states of program P. Formally, for each $s_0 \in \mathit{Init}_P$:

$$\begin{aligned} \llbracket P \rrbracket(s_0) \overset{\text{def}}{=} \{&(s_0, s_1)(s_1, s_1)(s_1, s_2) \ldots (s_k, s_k)(s_k, s_{k+1}) \mid \\ &\texttt{b}_0\texttt{c}_1\texttt{b}_1 \ldots \texttt{c}_k\texttt{b}_k \in \mathit{Exe}[G_P], \\ &\forall 0 < i \leq k : s_i \models \texttt{c}_i, (s_{i-1}, s_i) \in \texttt{exec}(\texttt{b}_{i-1})\} \end{aligned}$$

$$\llbracket P \rrbracket \overset{\text{def}}{=} \bigcup \left\{\, \llbracket P \rrbracket(s_0) \,\middle|\, s_0 \in \mathit{Init}_P \,\right\}$$

In order to define the SFA that corresponds to the CFG semantics of a given program we need to define an effective Boolean algebra that it is suitable for the representation of program execution. For this reason we define the following effective Boolean algebra where predicates are either basic blocks of instructions or guards of branching instructions, representing the syntactic structure of the program, and the denotations are pairs of input/output states:

$$\mathfrak{P} \overset{\text{def}}{=} \langle \Sigma \times \Sigma, \mathbb{I}^* \cup \mathbb{C}, \{\!|\cdot|\!\}, \bot, \top, \wedge, \vee, \neg \rangle$$

where the semantic function $\{\!|\cdot|\!\} : \mathbb{I}^* \cup \mathbb{C} \longrightarrow \wp^{\text{re}}(\Sigma \times \Sigma)$ is defined as follows for $\varphi \in \mathbb{I}^* \cup \mathbb{C}$:

$$\{\!|\varphi|\!\} \overset{\text{def}}{=} \begin{cases} \{ (s, s') \mid (s, s') \in \mathtt{exec(b)} \} & \text{if } \varphi = \mathtt{b} \in \mathbb{I}^* \\ \{ (s, s) \mid s \models \mathtt{c} \} & \text{if } \varphi = \mathtt{c} \in \mathbb{C} \end{cases}$$

we denote with $\{\!|\cdot|\!\}$ also its point-wise extension to $\wp^{\text{re}}(\mathbb{I}^* \cup \mathbb{C})$.

DEFINITION 6.1. *Let P be a program with CFG G_P. The SFA associated with P is*

$$M(P) \overset{\text{def}}{=} \langle \mathfrak{P}, \mathbb{PP}[G_P], in[\mathfrak{b}_0], \{out[\mathfrak{b}] \mid \mathfrak{b} \in Final[G_P]\}, \Delta_P \rangle$$

where $\mathfrak{b}_0$ is the starting basic block of G_P and Δ_P is defined as:

$$\Delta_P \overset{\text{def}}{=} \{ (in[\mathfrak{b}], \mathfrak{b}, out[\mathfrak{b}]) \mid \mathfrak{b} \in N_P \} \cup \{ (out[\mathfrak{b}], \mathfrak{c}, in[\mathfrak{b}']) \mid (\mathfrak{b}, \mathfrak{c}, \mathfrak{b}') \in E_P \}$$

PROPOSITION 6.2. *If P is a program then $M(P)$ is a deterministic SFA. $M(P)$ is clean if no dead-block is included in G_P.*

The language $\mathscr{L}(M(P)) \in \wp^{\text{re}}((\Sigma \times \Sigma)^*)$ recognized by the SFA $M(P)$ approximates the concrete program semantics $[\![P]\!]$ in a language of sequences of infinitely many possible input/output relations associated with each basic block. This is formally stated by the following theorem.

THEOREM 6.3. *If P is a program then for any $s_0 \in Init_P$: $[\![P]\!](s_0) \in \mathscr{L}(M(P))$.*

Given the SFA $M(P)$ that represents the CFG of a program P then it is possible to approximate the semantics of P by abstracting either the predicates, namely the syntax, or the semantics of the effective Boolean algebra underlying $M(P)$.

Let us consider the minimization simplifications. Given compatible abstractions $\rho \in uco(\wp(\Sigma \times \Sigma))$ and $\eta \in uco(\wp^{\text{re}}(\mathbb{I}^* \cup \mathbb{C}))$ and $k \in \mathbb{N}$ we have that

$$M(P) \dot{\leq} Min(M^{\rho}_{\eta}(P)) \dot{\leq} Min_k(M^{\rho}_{\eta}(P))$$

This provides a reduction of the original SFA, and therefore CFG, providing at the same time a unique approximate representation of the abstract semantics of P. This is possible thanks to the combined syntactic and semantic approximation, acting both on the code and on its interpretation. Two programs P and Q can then be considered *similar* if they have the same reduced abstract SFA up to $k \in \mathbb{N}$:

$$P \approxeq_k Q \text{ iff } k = \mathtt{max} \{ n \mid Min_n(M^{\rho}_{\eta}(P)) = Min_n(M^{\rho}_{\eta}(Q)) \}$$

This weaker notion of similarity can be improved by considering minimal SFA as canonical representation of the approximate syntax and semantics of programs:

$$P \approxeq Q \text{ iff } Min(M^{\rho}_{\eta}(P)) = Min(M^{\rho}_{\eta}(Q))$$

The following theorem is therefore immediate by construction.

THEOREM 6.4. *Let P and Q be programs, then $P \approxeq Q$ iff $\forall k \in \mathbb{N} : P \approxeq_k Q$.*

It is clear that, for decidable $\langle\!|\rho|\!\rangle\langle\eta\rangle$-abstractions, there exists $k \in \mathbb{N}$ such that $P \approxeq_k Q \implies P \approxeq Q$.

7. Formal similarity analysis of executables

The idea of *BinJuice* [18] is that the *juice* of a binary forms a template that is expected to be identical regardless of code variations due to register renaming, memory address allocation, and constant replacement. Similar ideas have been employed in *BinDiff* [12] where executables are treated as graphs of graphs: a control flow graph where each block is itself represented as a graph, which is the sequence of its instructions. While the subset of *BinDiff* considered here is sound and semantic compatible, it is computationally expensive. For large size executables, this problem has been tackled in *BinJuice* which adds a further level of abstraction to make the resulting abstract SFA more compact. In contrast to other similar tools for similarity analysis such as *DarunGrim2*, *Rdiff*, *Patchdiff*, and *Radar2*, all designed to find differences in variants of the same program for the purpose of creating patches, *BinJuice* and *BinDiff* are motivated by a different problem: Find similar code in binaries that are not known to be related. This necessitates more advanced abstractions acting on both code and semantics, therefore better showing the potential of abstract SFA.

7.1 *BinJuice*

BinJuice performs symbolic transformations on the source disassembled binary in order to transform each basic block of assembly code into a corresponding symbolic representation. The idea of symbolic execution is that the operations encoded by the assembly instructions are immediately performed when the arguments are integers, in a sort of partial evaluation local to each basic block, otherwise the same operation keeps its symbolic structure. Consider for example the following fragment of binary code and the result of its disassembly:

Binary	Assembly
`401290: b8 05 00 00 00`	`mov eax,0x5`
`401295: c3 04 00 00 00`	`add ebx,0x4`
`40129b: 6b c3`	

BinJuice performs algebraic manipulation of instructions in order to reach a canonical form. Thus, the result of symbolic execution with algebraic simplification of the previous example is:

Normalized State Updates	Constraints
`eax=5`	
`ebx=def(ebx)`$\times$`5 + 20`	`20 = 4 `$\times$` 5`

where `def(ebx)` denotes the value of `ebx` before the execution of the basic block, namely at the entry of the basic block. The syntactic information lost during symbolic execution is actually added back by the constraints on numerical values. In other words, the symbolic execution of basic blocks augmented with numerical constraints is actually an isomorphism. The key abstraction in *BinJuice* is generalization, whose idea is to use typed logical variables in order to be independent from register names. The generalization is performed by consistently replacing register names with logical variables. The replacement is consistent in that two occurrences of the same register name are always replaced by the same variable. Observe that this replacement is a purely syntactic operation. In addition to abstracting the registers used, also constants are abstracted. *BinJuice* associates a type with each logical variable to keep track of type of the original register. In the example considered before the generalization phase of *BinJuice* produces the following juice:

Juice
$A = V_1$
$B = \mathtt{def}(B) \times N_1 + N_2$
constraints: $N_2 = N_1 \times N_3$
types: $\mathtt{type}(A) = \mathtt{type}(B) = \mathtt{reg32}$

Let us consider the function $\mathcal{G}$ that generalizes a single basic block.

$$\mathcal{G} : \mathbb{I}^* \longrightarrow \wp^{\texttt{re}}(\mathit{SUpd}) \times \wp^{\texttt{re}}(\widehat{\mathbb{C}}) \times \wp^{\texttt{re}}(\mathcal{T})$$

where $\wp^{\texttt{re}}(\mathit{SUpd})$ is the domain of normalized symbolic updates while $\wp^{\texttt{re}}(\widehat{\mathbb{C}})$ is the set of constraints where register names and numerical values have been replaces by symbolic variables, and $\wp^{\texttt{re}}(\mathcal{T})$ denotes the domain of type declarations.

We say that $\mathcal{G}(\mathtt{b})$ is the *juice* of the basic block $\mathtt{b}$. Observe that $\mathcal{G}$ acts as an abstraction since there may be more than one basic block sharing the same juice. In particular, $\mathcal{G}$ can be associated with an upper closure $\mathscr{G} \in \mathit{uco}(\wp^{\texttt{re}}(\mathbb{I}^*))$ as follows:

$$\mathscr{G}(B) \overset{\text{def}}{=} \left\{ \mathtt{b}' \;\middle|\; \exists \mathtt{b} \in B.\, \mathcal{G}(\mathtt{b}) = \mathcal{G}(\mathtt{b}') \right\}$$

approximating in one single symbolic representation all basic blocks that have the same juice. We can therefore model the generalization process that *BinJuice* operates on the CFG of the disassembled binaries as an $\langle \mathscr{G} \rangle$-abstraction of the predicates of the effective Boolean algebra $\mathfrak{P}$ introduced in Section 6 for representing the CFG of programs as SFA. Here, we consider the extension of $\mathscr{G}$ to branching conditions on which it behaves like identity, $\mathscr{G} \in \mathit{uco}(\wp^{\texttt{re}}(\mathbb{I}^* \cup \mathbb{C}))$. The resulting *BinJuice* symbolic automaton on the Boolean algebra $\mathfrak{P}_{\mathscr{G}}$ associated with a disassembled program P is:

$$M_{\mathscr{G}}(P) = \langle \mathfrak{P}_{\mathscr{G}}, \mathbb{PP}[G_P], \mathit{in}[b_0], \{\mathit{out}[b] \mid b \in \mathit{Final}[G_P]\}, \Delta_{\mathscr{G}} \rangle$$

where $\mathfrak{P}_{\mathscr{G}} = \langle \Sigma \times \Sigma, \mathscr{G}(\wp^{\texttt{re}}(\mathbb{I}^* \cup \mathbb{C})), \{\!| \cdot |\!\}, \bot, \top, \wedge, \vee, \neg \rangle$,

$$\begin{aligned} \Delta_{\mathscr{G}} = & \left\{ (\mathit{in}[\mathtt{b}], \mathscr{G}(\mathtt{b}), \mathit{out}[\mathtt{b}]) \;\middle|\; (\mathit{in}[\mathtt{b}], \mathtt{b}, \mathit{out}[\mathtt{b}]) \in \Delta_P \right\} \\ & \cup \left\{ (\mathit{out}[\mathtt{b}], \mathtt{c}, \mathit{in}[\mathtt{b}']) \;\middle|\; (\mathit{out}[\mathtt{b}], \mathtt{c}, \mathit{in}[\mathtt{b}']) \in \Delta_P \right\} \end{aligned}$$

and the semantic function $\{\!| \cdot |\!\}$ is the same as defined in Section 6 but now with a reduced abstracted domain:

$$\{\!| \cdot |\!\} : \mathscr{G}(\wp^{\texttt{re}}(\mathbb{I}^* \cup \mathbb{C})) \longrightarrow \wp^{\texttt{re}}(\Sigma \times \Sigma)$$

We observe that, $\mathscr{G}$ is neither syntactic nor semantic compatible (Def. 3.6, Def. 3.9) since:

(1) it collapses simplified updates with different semantics by abstracting values and variables, for example $\mathscr{G}(eax = 5) = \mathscr{G}(eax = 7) = (X = N)$;

(2) it still distinguishes between different simplified updates sharing the same semantics, as for example $\mathscr{G}(eax = ebx * 2) \neq \mathscr{G}(eax = ebx + ebx)$. But also

$$\begin{aligned} \mathscr{G}(eax = 2 * ebx + 10, \text{constraint: } 10 = 5 * 2) = \\ X = N_1 * Y + N_2, \text{constraint: } N_2 = N_3 * N_4 \end{aligned}$$

and

$$\begin{aligned} \mathscr{G}(eax = 2 * ebx + 10, \text{constraint: } 10 = 5 + 5) = \\ X = N_1 * Y + N_2, \text{constraint: } N_2 = N_3 + N_4 \end{aligned}$$

Indeed, $\mathscr{G}$ is not comparable with $\Omega(\mathtt{id})$.

PROPOSITION 7.1. *$\mathscr{G}$ is is neither syntactic nor semantic compatible.*

This observation is also related to the incorrectness of *BinJuice* in detecting similar basic blocks indeed *BinJuice* can lead to both false positives (blocks miss-classified as equivalent) and false negatives (blocks that are erroneously classified as different).

As observed before there are two causes of semantic incompatibility: (1) merging updates with different semantics and (2) distinguishing updates with the same semantics. We are interested in over-approximating $\Omega(\mathtt{id})$, namely obtaining a closure η such that $\Omega(\mathtt{id}) \sqsubseteq \eta$, therefore avoiding (2) yet keeping (1).

A possible way for making $\mathscr{G}$ semantic compatible is to erase from the domain $\wp^{\texttt{re}}(\mathbb{I}^* \cup \mathbb{C})$ all the elements that have the same generalized symbolic updates but different constraints. Namely by erasing all syntactic constraints. In the example above, it means for instance to restrict to the blocks that have generalized update $X = N_1 * Y + N_2$ while abstracting from the constraints on N_2. It is possible to prove that *BinJuice* is semantic compatible when considering this restricted domain of blocks. This highlights the fact that *BinJuice* is sensible to the structure of the constraints. Indeed, the constraints keep track of how the numerical values present in the update have been computed and is therefore tight to the particular way in which the basic block has computed them. This means that *BinJuice* can be foiled by an attacker that changes the structure of the constraints.

Define $\pi_1 : \wp^{\texttt{re}}(\mathit{SUpd}) \times \wp^{\texttt{re}}(\widehat{\mathbb{C}}) \times \wp^{\texttt{re}}(\mathcal{T}) \to \wp^{\texttt{re}}(\mathit{SUpd})$ as the projection on the first element of the tuple of the juice. Based on this, given $\mathtt{b} \in \mathbb{I}^*$ we define the predicate abstraction $\mathcal{U}[b] \in \mathit{uco}(\wp(\mathbb{I}^*))$ that keeps only the blocks that have the same generalized update of $\mathtt{b}$ and abstract in $\top$ every other block:

$$\mathcal{U}[\mathtt{b}](\mathtt{b}') \begin{cases} \mathtt{b}' & \text{if } \pi_1(\mathscr{G}(\mathtt{b})) = \pi_1(\mathscr{G}(\mathtt{b}')) \\ \top & \text{otherwise} \end{cases}$$

As expected, for every basic block $\mathtt{b}$ we have that the predicate abstraction $\mathscr{G} \circ \mathcal{U}[\mathtt{b}]$, that extracts the juice of blocks that have the same generalized updates of $\mathtt{b}$, is such that $\mho(\mathcal{U}[\mathtt{b}])$ is syntactic $\langle \mathscr{G} \circ \mathcal{U}[\mathtt{b}] \rangle$-compatible, as stated by the following result.

THEOREM 7.2. *$\forall b \in \mathbb{I}^*$ we have that $\Omega(\mho(\mathcal{U}[b])) \sqsubseteq \mathscr{G} \circ \mathcal{U}[b]$*

This result is a direct consequence of the definitions of $\mathcal{U}[\mathtt{b}]$ and of $\mathscr{G}$, and by Prop. 3.15-(2). Once again, this formally proves that *BinJuice* over-approximates the set of blocks with the same semantics when we restrict to blocks that have the same symbolic update.

7.2 *BinDiff*

We consider a subset of *BinDiff*, employing *instruction permutation* and *same string reference* (i.e., instructions and nodes can be matched by common string references, e.g., indicating functions that all contain code referring to the same string). All these equivalences correspond straightforwardly to abstractions of the SFA acting at syntactic and topological level. Consider the SFA $M(P)$ and the following abstractions:

Permutation. Let $\tau : \mathbb{I} \longrightarrow \mathcal{T}$ be a function associating the mnemonic op-code in $\mathcal{T}$ at each instruction in $\mathbb{I}$. Consider the lift of τ to multi-sets. Define an equivalence relation on basic blocks, viz., predicates in $M(P)$, such that for any $\mathtt{b}, \mathtt{b}' \in \mathbb{I}^* \cup \mathbb{C}$: $\mathtt{b} \equiv \mathtt{b}'$ if $\tau(\mathtt{b}) = \tau(\mathtt{b}')$. This clearly induces a partition which is a (partitioning) closure operator, denoted η_τ on predicates in $\mathbb{I}^* \cup \mathbb{C}$. In other words, τ forgets the order and the arguments of instructions. It is therefore clear that $\eta_\tau(\mathtt{b}) = \eta_\tau(\mathtt{b}') \not\Rightarrow \{\!|\mathtt{b}|\!\} = \{\!|\mathtt{b}'|\!\}$, namely η_τ may collapse blocks with different semantics meaning that it is not semantic compatible, i.e., $\eta_\tau \not\sqsubseteq \Omega(\mathtt{id})$. On the other hand, since η_τ observes precisely the multi-set of instructions, we could have blocks with the same semantics but written with different sets of instructions, i.e., $\{\!|\mathtt{b}|\!\} = \{\!|\mathtt{b}'|\!\} \not\Rightarrow \eta_\tau(\mathtt{b}) = \eta_\tau(\mathtt{b}')$ meaning that η_τ fails also the syntactic compatibility.

Same reference. Let $\mathcal{N}$ be a set of strings and $\xi : \mathbf{I} \longrightarrow \wp(\mathcal{N})$ the function associating with each basic block $\mathtt{b}$ the set of strings of $\mathcal{N}$ appearing in $\mathtt{b}$. This is clearly the left-adjoint of a GC, therefore inducing a closure η_ξ on predicates which is also a partition. This abstraction forgets any instruction considering only a set of string manipulated in the block. Again, it is quite straightforward to observe that this abstraction can both collapse blocks with different semantics and distinguish blocks with the same semantics, for instance a string may be computed without writing it explicitly. Hence also η_ξ fails both the compatibilities.

In order to make *permutations* syntactic compatible, we can indeed restrict the domain of the permutation abstraction similarly to what we have done on *BinJuice* and forcing syntactic compatibility. Let $\mathit{Instr}(B) \overset{\text{def}}{=} \left\{ \mathtt{b}' \;\middle|\; \exists \mathtt{b} \in B.\, \eta_\tau(\mathtt{b}') = \eta_\tau(\mathtt{b}) \right\}$ and

$$\mathcal{S}[\mathtt{b}](\mathtt{b}') = \begin{cases} \mathtt{b}' & \text{if } \mathit{Instr}(\mathtt{b}) = \mathit{Instr}(\mathtt{b}') \\ \top & \text{otherwise} \end{cases}$$

As expected, for every basic block $\mathtt{b}$ we have that the predicate abstraction $\mathit{Instr} \circ \mathcal{S}[\mathtt{b}]$, that collects blocks that have the same set of instructions of $\mathtt{b}$, is such that $\mho(\mathcal{S}[\mathtt{b}])$ is syntactic $\langle \mathit{Instr} \circ \mathcal{S}[\mathtt{b}] \rangle$-compatible, as stated by the following result which is a consequence of the definitions of $\mathcal{S}[\mathtt{b}]$ and of *Instr*, and by Prop. 3.15-(2).

THEOREM 7.3. $\forall \mathtt{b} \in \mathbb{I}^*$ *we have that* $\Omega(\mho(\mathcal{S}[\mathtt{b}])) \sqsubseteq \mathrm{Instr} \circ \mathcal{S}[\mathtt{b}]$.

8. Related Works

To the best of our knowledge, this is the first application of abstract interpretation to symbolic finite automata and of abstract symbolic automata to similarity analysis of binary executables. The most related work is [14], where the authors introduced the notion of *lattice automata*. Lattice automata, like SFA, allow languages over an infinite alphabet. In contrast to abstract SFA, lattice automata do not distinguish between symbolic/syntactic abstractions and semantic ones. Indeed transitions in lattice automata are constrained by elements in an atomic lattice L, which provide precisely the allowed alphabet-set along that transition. SFA are in this context strictly more general as they separate the symbolic constraints and their semantics, allowing in principle separate approximations for them.

The idea of approximating the program's data in a so called *predicate abstraction* is nowadays common practice in static program analysis. The roots of this idea are in automatic software verification (see [1, 13]). Observe that, given a program P, predicate abstraction abstracts the semantics (states) of P into a set of predicates E and then it derives an abstract program P-bool that models how the execution of P affects E. Thus, predicate abstraction corresponds to a semantic abstraction ρ that groups states w.r.t. to E, and P-bool is a possible way of representing the syntactic compatible abstraction $\Omega(\rho)$ of P. As observed in [2] predicate abstraction considers only finite abstractions, while the semantic abstraction of denotations in abstract SFA can be an infinite domain. Moreover, predicate abstraction does not allow to change the CFG of the program.

The relation between the approximation of symbolic/syntactic structures and their semantics is well known in the literature (see [6] and [11] for a recent account). In particular in [22] the authors study this relation for the systematic synthesis of optimal symbolic predicate transformers, as introduced in [20]. None of these consider the case of abstract interpretation of SFA. In [7] the authors model disassembled binaries as finite state automata (FSA). A widening of FSA is introduced for extracting syntactic code invariants in self-modifying metamorphic programs. This construction lacks of abstractions concerning the semantics of sequences of instructions.

9. Conclusion

We have studied how to weaken symbolic finite automata by abstract interpretation. The results is a general theory of approximated SFA which is parametric on the chosen abstraction. The purpose is to provide a compact and effective representation of code approximations acting both at syntactic and semantic level. Interestingly, for a Turing complete programming language, there is no syntactic abstraction which induces a compatible semantic abstraction. This follows from a simple padding argument, and it is indeed a common underlying problem in most known methods for program similarity analysis, such as *BinDiff* and *BinJuice*. Observe that the existing tools either abstract the syntax independently from the semantics, like in *BinDiff*, or represent into the syntax the abstraction of the semantics, like in predicate abstraction. In the first case we risk to fall far away form the meaning of the program to analyze, in the second case the analysis may be too much bound to the semantics without having the possibility of exploiting better syntax properties necessary in similarity analysis (e.g. in *BinDiff* and *BinJuice*). Compatibility bridges these two aspects. By semantic compatibility the abstraction of the syntax distinguishes programs with the same abstract semantics, namely the abstract program provides an under-approximation of the program behavior. By syntactic compatibility the abstraction of the syntax collapses programs with different semantics, hence capturing behaviors that are not related with the program to analyze, therefore providing an over-approximation of the program behavior. Interestingly, in our model we can restrict the form of predicates in order to have compatibility. This is what we proved in *BinJuice* and *BinDiff*, thus showing the limits of existing tools for code similarity and the possibility of systematically deriving conditions for making them syntactic compatible.

Another direction of future research is in the use of topological abstractions of SFA for extracting signatures of self-modifying code as recently studied in [7]. This requires the extension of widening operations, such as those introduced in [5, 10, 14], to abstract SFA. In this case approximate SFA provide advanced signatures in metamorphic malware analysis, incorporating both properties of way code changes during program execution (the invariant of the metamorphic engine) and additional semantic information, such as the values passed in system-calls. This may reduce the false positives occurring in [7] in signature-based detection of metamorphic malware.

Acknowledgments

This work has been conceived when Mila, Roberto, and Isabella were visiting Arun at the University of Louisiana at Lafayette USA. They thank the Center for Advanced Computer Studies for the kind hospitality. This work is partly supported by the Air Force Research Laboratory and DARPA project: AFRL FA8750-10-C-0171 and by the MIUR FIRB project FACE (Formal Avenue for Chasing malwarE) RBFR13AJFT.

References

[1] T. Ball, R. Majumdar, T. D. Millstein, and S. K. Rajamani. Automatic predicate abstraction of C programs. In M. Burke and M. L. Soffa, editors, *PLDI*, pages 203–213. ACM, 2001. ISBN 1-58113-414-2.

[2] P. Cousot. Verification by abstract interpretation. In *Verification: Theory and Practice, Essays Dedicated to Zohar Manna on the Occasion of His 64th Birthday*, volume 2772, pages 243–268. Springer, 2003.

[3] P. Cousot and R. Cousot. Abstract interpretation: A unified lattice model for static analysis of programs by construction or approximation of fixpoints. In *Conference Record of the 4th ACM Symposium on Principles of Programming Languages* (*POPL '77*), pages 238–252. ACM Press, 1977.

[4] P. Cousot and R. Cousot. Systematic design of program analysis frameworks. In *Conference Record of the 6th ACM Symposium on Principles of Programming Languages* (*POPL '79*), pages 269–282. ACM Press, 1979.

[5] P. Cousot and R. Cousot. Formal language, grammar and set-constraint-based program analysis by abstract interpretation. In *Proceedings of the Seventh ACM Conference on Functional Programming Languages and Computer Architecture*, pages 170–181. ACM Press, New York, NY, 25–28 June 1995.

[6] P. Cousot, R. Cousot, and L. Mauborgne. Theories, solvers and static analysis by abstract interpretation. *J. ACM*, 59(6):31, 2012.

[7] M. Dalla Preda, R. Giacobazzi, S. K. Debray, K. Coogan, and G. M. Townsend. Modelling metamorphism by abstract interpretation. In *Proc. of the 19th Int. Static Analysis Symp.* (*SAS '10*), volume 6337 of *Lecture Notes in Computer Science*, pages 218–235. Springer-Verlag, Berlin, 2010.

[8] L. D'Antoni and M. Veanes. Equivalence of extended symbolic finite transducers. In N. Sharygina and H. Veith, editors, *CAV*, volume 8044 of *Lecture Notes in Computer Science*, pages 624–639. Springer, 2013. ISBN 978-3-642-39798-1.

[9] L. D'Antoni and M. Veanes. Minimization of symbolic automata. In S. Jagannathan and P. Sewell, editors, *POPL*, pages 541–554. ACM, 2014. ISBN 978-1-4503-2544-8.

[10] V. D'Silva. Widening for automata. Diploma Thesis, Institut Fur Informatick, Universitat Zurich, 2006.

[11] V. D'Silva, L. Haller, and D. Kroening. Abstract satisfaction. In S. Jagannathan and P. Sewell, editors, *POPL*, pages 139–150. ACM, 2014. ISBN 978-1-4503-2544-8.

[12] H. Flake. Structural comparison of executable objects. In U. Flegel and M. Meier, editors, *DIMVA*, volume 46 of *LNI*, pages 161–173. GI, 2004. ISBN 3-88579-375-X.

[13] C. Flanagan and S. Qadeer. Predicate abstraction for software verification. In *Proc. of Conf. Record of the 29th ACM Symp. on Principles of Programming Languages* (*POPL '02*), pages 191–202. ACM Press, 2002.

[14] T. L. Gall and B. Jeannet. Lattice automata: A representation for languages on infinite alphabets, and some applications to verification. In H. R. Nielson and G. Filé, editors, *SAS*, volume 4634 of *Lecture Notes in Computer Science*, pages 52–68. Springer, 2007. ISBN 978-3-540-74060-5.

[15] D. Gao, M. Reiter, and D. Song. BinHunt: Automatically finding semantic differences in binary programs. In *Proceedings of the 10th International Conference on Information and Communications Security*, ICICS '08, pages 238–255. Springer-Verlag, 2008.

[16] R. Giacobazzi, F. Ranzato, and F. Scozzari. Making abstract interpretation complete. *Journal of the ACM*, 47(2):361–416, March 2000.

[17] S. Hunt and I. Mastroeni. The PER model of abstract non-interference. In C. Hankin and I. Siveroni, editors, *Proc. of The 12th Internat. Static Analysis Symp. (SAS '05)*, volume 3672 of *Lecture Notes in Computer Science*, pages 171–185. Springer-Verlag, 2005.

[18] A. Lakhotia, M. Dalla Preda, and R. Giacobazzi. Fast location of similar code fragments using semantic 'juice'. In *2nd Workshop on Program Protection and Reverse Engineering PPREW 2013*. ACM, 2013.

[19] I. Mastroeni and R. Giacobazzi. An abstract interpretation-based model for safety semantics. *Int. J. Comput. Math.*, 88(4):665–694, 2011.

[20] T. W. Reps, S. Sagiv, and G. Yorsh. Symbolic implementation of the best transformer. In B. Steffen and G. Levi, editors, *VMCAI*, volume 2937 of *Lecture Notes in Computer Science*, pages 252–266. Springer, 2004. ISBN 3-540-20803-8.

[21] H. Rogers. *Theory of recursive functions and effective computability*. The MIT press, 1992.

[22] A. V. Thakur, M. Elder, and T. W. Reps. Bilateral algorithms for symbolic abstraction. In A. Miné and D. Schmidt, editors, *SAS*, volume 7460 of *Lecture Notes in Computer Science*, pages 111–128. Springer, 2012. ISBN 978-3-642-33124-4.

[23] M. Veanes, P. Hooimeijer, B. Livshits, D. Molnar, and N. Bjørner. Symbolic finite state transducers: algorithms and applications. In J. Field and M. Hicks, editors, *POPL*, pages 137–150. ACM, 2012. ISBN 978-1-4503-1083-3.

[24] M. Ward. The Closure Operators of a Lattice. *Annals of Mathematics*, 43(2):191–196, 1942.

[25] Zynamics. BinDiff3.2manual., 2004. URL `http://www.zynamics.com/bindiff/manual/`.