The $\mathbb{K}$ Summarizer

Runtime Verification, Inc.

July 21, 2022

The purpose of this document is to provide a formal specification of the $\mathbb{K}$ summarizer (`https://github.com/runtimeverification/erc20-verification/tree/master/ksummarize`) that is mathematically sound and is faithful to the implementation. We will also study the case studies of the summarizer for semantics-based compilation (SBC), model checking, and formal verification.

1 Preliminaries

Throughout this document, we assume that there is an underlying matching logic theory Γ^L that defines the formal semantics of some programming language L. In Figure 1 we list the standard notations that will be used in this document.

2 Matching and Unification

In this section we formalize matching and unification as proving matching logic theorems. Some definitions and results are from Arusoaie and Lucanu's paper `https://arxiv.org/pdf/1811.02835.pdf`.

2.1 Substitution Patterns

Definition 1. Given a substitution

$$\tau \equiv [\varphi_1/x_1, \ldots, \varphi_n/x_n]$$

we define a corresponding *substitution pattern*

$$\varphi^\tau \equiv (x_1 = \varphi_1) \wedge \cdots \wedge (x_n = \varphi_n)$$

Proposition 2. $\Gamma^L \vdash \varphi^\tau \to (\psi = \psi\tau)$.

Proof. This (derived) proof rule is called (Equality Elimination). □

The following standard notations will be used:

- φ, ψ: an arbitrary pattern.
- t: a term pattern, built from element variables and functional symbols.
- p: a predicate pattern.
- $t \wedge p$: a constraint term.
- $\Gamma^L \vdash \varphi$: φ is provable from Γ^L.
- $\tau \equiv [\varphi_1/x_1, \ldots, \varphi_n/x_n]$: a substitution.
- $\varphi\tau$ or $\varphi[\varphi_1/x_1, \ldots, \varphi_n/x_n]$: applying the substitution to φ.
- $\lceil\varphi\rceil$: the definedness pattern of φ.
- $\bullet\varphi$: "one-path next" of φ.
- $\circ\varphi$: "all-path next" of φ, defined as $\circ\varphi \equiv \neg\bullet\neg\varphi$.
- STOP: stopped/terminal states; abbreviation for $\circ\bot$.
- $\mathsf{NONSTOP}$: non-stopped states; abbreviation for $\bullet\top$.
- $\circ_s\varphi$: "strongly all-path next" of φ, defined as $\circ_s\varphi \equiv \circ\varphi \wedge \mathsf{NONSTOP}$.
- $\varphi \Rightarrow_1^{\exists} \varphi'$: "one-step one-path execution"; abbreviation for $\varphi \to \bullet\varphi'$.
- $\varphi \Rightarrow_1^{\forall} \varphi'$: "one-step all-path execution"; abbreviation for $\varphi \to \circ_s\varphi'$.
- $\varphi \Rightarrow_*^{\forall} \varphi'$: "all-path execution"; abbreviation for $\varphi \to \mu X \,.\, \varphi \vee \circ X$.
- more to be added ...

Figure 1: List of Notations

2.2 Matching

Definition 3. Let φ and ψ be two patterns without free set variables. We call φ the *left pattern* and ψ the *right pattern*.[1] We say that φ *matches* ψ if

$$\Gamma^L \vdash \varphi \to \exists FV(\psi) \,.\, \psi$$

Example 4. Suppose that f is a function symbol. Then $f(0,0)$ matches $f(x,y)$, because

$$\Gamma^L \vdash f(0,1) \to \exists x \exists y \,.\, f(x,y)$$

Example 5. Suppose that f is a function symbol. Then $f(0,z)$ matches $f(x,y)$ because

$$\Gamma^L \vdash f(0,z) \to \exists x \exists y \,.\, f(x,y)$$

Note that the above is equivalent to

$$\Gamma^L \vdash \forall z \,.\, (f(0,z) \to \exists x \exists y \,.\, f(x,y))$$

In other words, all variables in the left pattern are universally quantified.

Example 6. Suppose that f is a function symbol. Then $f(0,x)$ matches $f(x,y)$ because

$$\Gamma^L \vdash f(0,x) \to \exists x \exists y \,.\, f(x,y)$$

Note that x in the right pattern is quantified/bound. Generally speaking, all the variables in the right pattern are quantified and can be freely renamed. It allows us to assume that φ and ψ have distinct free variables without loss of generality.

Semantically speaking, φ matches ψ implies that

$$|\varphi|_{\rho,M} \subseteq |\exists FV(\psi) \,.\, \psi|_{\rho,M} = \bigcup_{a_1,\ldots,a_n \in M} |\psi|_{\rho[a_1/x_1,\ldots,a_n/x_n],M}$$

where $FV(\psi) = \{x_1, \ldots, x_n\}$. If φ is a unit pattern (i.e., a functional pattern), then $|\varphi|_{\rho,M}$ is a unit set and there exist $a_1, \ldots, a_n \in M$ such that

$$|\varphi|_{\rho,M} \subseteq |\psi|_{\rho[a_1/x_1,\ldots,a_n/x_n],M}$$

Pattern matching gives us closed-form solutions of these witness values $a_1, \ldots, a_n$. Formally,

Definition 7. Let φ and ψ be two patterns and σ be a substitution pattern. We say that σ is a *solution* to the matching problem

$$\varphi \lhd_? \psi$$

if

$$\Gamma^L \vdash \varphi \to \psi\tau$$

[1] Usually φ is called the "term" and ψ the "pattern", but since matching logic formulas are already called patterns, we use left/right patterns to avoid confusion.

there exists a particular value v_i for each $x_i \in FV(\psi)$ such that $\Gamma^L \vdash \varphi \to \psi[v_1/x_1, v_2/x_2, \dots]$, where v_i depends on

Definition 8. We say that $\{\sigma_1, \dots, \sigma_n\}$ is a *complete solution* to the matching problem

$$\varphi_1 \lhd_? \psi_1, \dots, \varphi_m \lhd_? \psi_m$$

if

$$\vdash \left(\bigwedge_{i=1}^{m} \varphi_i \subseteq \psi_i \right) \leftrightarrow \varphi^{\tau_1} \vee \dots \vee \varphi^{\tau_n}$$

Proposition 9. *For terms t and s, Definition 3 coincides with the classical definition of term matching.*

Proposition 10. *φ matches ψ if and only if*

$$\vdash (\exists FV(\varphi) \,.\, \varphi) \subseteq (\exists FV(\psi) \,.\, \psi)$$

Proof. By Definition 3. □

2.3 Unification

Definition 11. Let φ and ψ be two patterns. We say that φ *unifies* with ψ if

$$\vdash \lceil (\exists FV(\varphi) \,.\, \varphi) \wedge (\exists FV(\psi) \,.\, \psi) \rceil$$

We say that $\{\sigma_1, \dots, \sigma_n\}$ is a *complete solution* to the unification problem

$$\varphi_1 =_? \psi_1, \dots, \varphi_m =_? \psi_m$$

if

$$\vdash \left(\bigwedge_{i=1}^{m} \lceil \varphi_i \wedge \psi_i \rceil \right) \leftrightarrow \varphi^{\tau_1} \vee \dots \vee \varphi^{\tau_n}$$

Proposition 12. *For terms t and s, Definition 11 coincides with the classical definition of term unification.*

2.4 Modulo Theories

Definitions 3 and 11 work with underlying theories, in which case we obtain matching/unification modulo theories.

3 Transition Systems and Rewriting Relations

TBA

4 $\mathbb{K}$ Summarizer Assuming Determinism

Throughout this section, let us assume that the underlying formal semantics are deterministic. Formally, it means that $\Gamma^L \vdash \bullet\varphi \to \circ\varphi$ for all φ of sort Cfg, which is the sort of computation configurations. We introduce the following notations:

- $t_1 \Rightarrow_1 t_2$: one-step rewriting
- $t_1 \Rightarrow_n t_2$: n-step rewriting
- $t_1 \Rightarrow_* t_2$: finite-step rewriting
- $t_1 \Rightarrow_+ t_2$: finite-step at-least-one-step rewriting

Definition 13. A $\mathbb{K}$ control-flow graph (abbreviated KCFG) $G = (V, E_r, E_a, E_s)$ is a directed graph with three types of edges, where

- the vertex set V is a set of patterns of sort Cfg;
- $E_r \subseteq V \times V$ is called the *rewriting relation*;
- $E_a \subseteq V \times V$ is called the *abstracting relation*;
- $E_s \subseteq V \times V$ is called the *splitting relation*.

We write $\varphi \rightsquigarrow_r \psi$ ($\varphi \rightsquigarrow_a \psi$ and $\varphi \rightsquigarrow_s \psi$, resp.) for the three types of edges.

Definition 13 defines KCFGs in its most general form as directed graphs with patterns as its nodes and three types of edges. The actual KCFGs that are generated by the $\mathbb{K}$ summarizer are of a more specific form, which we refer to as *regular* KCFGs. A regular KCFG has *constrained terms* of the form $t \wedge p$ as its nodes, where t is a term and p is a predicate. The rewriting relation E_r corresponds to the executions of basic blocks. The abstracting relation E_a corresponds to pattern matching. The splitting relation E_s corresponds to adding additional path conditions. In the following we will formalize regular KCFGs that are constructed by the $\mathbb{K}$ summarizer.

4.1 Input to the $\mathbb{K}$ Summarizer

Formal Semantics. Let $S = \{lhs_i \Rightarrow rhs_i \mid i = 1, \ldots, |S|\}$ be the set of semantic rules. Further, let $S_{\text{cut}} \cup S_{\text{ncut}}$ be a division of S where rules in S_{cut} are called *cut rules*. Rules in S_{ncut} are called *non-cut rules*. The left-hand sides of the cut-rules are called *cut patterns*. The set of all cut patterns is defined as

$$P_{\text{cut}} = \{lhs_i \mid lhs_i \Rightarrow rhs_i \in S_{\text{cut}}\}$$

Initial Pattern. Let $ct_0 \equiv t_0 \wedge p_0$ be the constrained term that represents the initial configuration that is given to the $\mathbb{K}$ summarizer for which a KCFG is constructed.

Target Patterns. Let P_{target} be a set of patterns called the *target patterns*. When a target pattern is reached the $\mathbb{K}$ summarizer stops the construction of the KCFG.

Abstracting Function. Let abs be the abstracting function from constrained terms to constrained terms, with the property that for any ct there exists a substitution σ such that

$$\Gamma^L \vdash ct = \mathsf{abs}(ct)\sigma$$

4.2 Regular KCFGs

Definition 14. Given the input in Section 4.1 A *regular* KCFG w.r.t. ct_0

$$G = (V, ct_0, E_r, E_a, E_s)$$

satisfies the following conditions.

1. (V, E_r, E_a, E_s) is a KCFG.
2. V is a set of constrained terms and $ct_0 \in V$.
3. For every $ct \in V$, there can be only one type of outgoing edges (rewriting, abstracting, or splitting).
4. For every $ct_1 \rightsquigarrow_r ct_2$, $\Gamma^L \vdash ct_1 \Rightarrow_+ ct_2$. Also, ct_2 is the only successor node of ct_1 in G.
5. For every $ct_1 \rightsquigarrow_a ct_2$, there exists a substitution σ such that $\Gamma^L \vdash ct_1 = ct_2\sigma$. We shall write $ct_1 \rightsquigarrow_a^\sigma ct_2$ to indicate the witness substitution σ. Also, ct_2 is the only successor node of ct_1 in G.
6. For every $ct_1 \rightsquigarrow_s ct_2$, there exists p such that $\Gamma^L \vdash ct_2 = ct_1 \wedge p$. We shall write $ct_1 \rightsquigarrow_s^p ct_2$ to indicate the witness predicate p.
7. Let $ct \in V$ be a node that has outgoing spitting edges and $ct_1, \ldots, ct_n$ are all of its (E_s-)successors:

   $$ct \rightsquigarrow_s^{p_1} ct_1, \ldots, ct \rightsquigarrow_s^{p_n} ct_n$$

   Then $\Gamma^L \vdash p_1 \vee \cdots \vee p_n$.
8. $(G, E_r \cup E_s)$ forms a tree.
9. There are no consecutive edges of the same type because they can be collapsed (using the `remove_intermediate_covers` routine).

We call the above regular KCFG *complete* if all the target nodes (i.e., nodes that do not have any outgoing edges) are target patterns given in P_{target}.