Calculational Design of a Regular Model Checker by Abstract Interpretation*

Patrick Cousot[1]

CS, CIMS, NYU, New York, NY, USA
pcousot@cims.nyu.edu

Abstract. Security monitors have been used to check for safety program properties at runtime, that is for any given execution trace. Such security monitors check a safety temporal property specified by a finite automaton or, equivalently, a regular expression. Checking this safety temporal specification for all possible execution traces, that is the program semantics, is a static analysis problem, more precisely a model checking problem, since model checking specializes in temporal properties. We show that the model checker can be formally designed by calculus, by abstract interpretation of a formal trace semantics of the programming language. The result is a structural sound and complete model checker, which proceeds by induction on the program syntax (as opposed to the more classical approach using computation steps formalized by a transition system). By Rice theorem, further hypotheses or abstractions are needed to get a realistic model checking algorithm.

Keywords: Abstract interpretation · Calculational design · Model checking.

1 Introduction

Model checking [9,41] consists in proving that a model of a given program/computer system satisfies a temporal specification[1]. Traditionally, the model of the given program/computer system is a transition system and its semantics is the set of traces generated by the transition system. The temporal specification is usually one of the many variants of temporal logics such as the Linear Time Temporal logic (LTL), the Computation Tree Logic (CTL), or the combination CTL* of the two. The semantics of the temporal specification is a set of traces. The problem is therefore to check that the set of traces of the semantics of the given program/computer system is included in the set of traces of the semantics of the temporal specification. This is a Galois connection-based abstraction and so a model checking algorithm can be designed by calculus. To show that we consider a non-conventional temporal specification using regular expressions [31] and a structural fixpoint prefix-closed trace semantics which differs from the

* Supported by NSF Grant CCF-1617717.

[1] We define model checking as the verification of temporal properties and do not reduce it to the reachability analysis (as done *e.g.* in [10, Ch. 15, 16, 17, *etc.*]) since reachability analysis predates model checking [14] including for the use of transition systems [12].

R. M. Hierons and M. Mosbah (Eds.): ICTAC 2019, LNCS 11884, pp. 3–21, 2019.
https://doi.org/10.1007/978-3-030-32505-3_1

traditional small-step operational semantics specified by a transition system. There are properties of traces that are not expressible in temporal logic but are easily expressible using regular expressions [46].

2 Syntax and Trace Semantics of the Programming Language

Syntax Programs are a subset of C with the following context-free syntax.

$\mathtt{x}, \mathtt{y}, \ldots \in \mathbb{X}$		variable ($\mathbb{X}$ not empty)
$\mathtt{A} \in \mathbb{A}$	$::= \mathtt{1} \mid \mathtt{x} \mid \mathtt{A}_1 - \mathtt{A}_2$	arithmetic expression
$\mathtt{B} \in \mathbb{B}$	$::= \mathtt{A}_1 < \mathtt{A}_2 \mid \mathtt{B}_1$ **nand** $\mathtt{B}_2$	boolean expression
$\mathtt{E} \in \mathbb{E}$	$::= \mathtt{A} \mid \mathtt{B}$	expression
$\mathtt{S} \in \mathbb{S}$	$::=$	statement
	$\mathtt{x} = \mathtt{A}$ **;**	assignment
	;	skip
	if (B **)** S **if (** B **)** S **else** S	conditionals
	while (B **)** S **break ;**	iteration and break
	{ Sl **}**	compound statement
$\mathtt{Sl} \in \mathbb{S}\mathbb{l}$	$::= \mathtt{Sl}\ \ \mathtt{S} \mid \epsilon$	statement list
$\mathtt{P} \in \mathbb{P}$	$::= \mathtt{Sl}$	program

A break exits the closest enclosing loop, if none this is a syntactic error. If P is a program then **int main () { P }** is a valid C program. We call "[program] component" $\mathtt{S} \in \mathbb{P}\mathtt{c} \triangleq \mathbb{S} \cup \mathbb{S}\mathbb{l} \cup \mathbb{P}$ either a statement, a statement list, or a program. We let $\lhd$ be the syntactic relation between immediate syntactic components. For example, if $\mathtt{S} =$ **if (B)** $\mathtt{S}_t$ **else** $\mathtt{S}_f$ then $\mathtt{B} \lhd \mathtt{S}$, $\mathtt{S}_t \lhd \mathtt{S}$, and $\mathtt{S}_f \lhd \mathtt{S}$.

Program labels Labels are not part of the language, but useful to discuss program points reached during execution. For each program component S, we define

at⟦S⟧	the program point at which execution of S starts;
aft⟦S⟧	the program exit point after S, at which execution of S is supposed to normally terminate, if ever;
esc⟦S⟧	a boolean indicating whether or not the program component S contains a **break ;** statement escaping out of that component S;
brk-to⟦S⟧	the program point at which execution of the program component S goes to when a **break ;** statement escapes out of that component S;
brks-of⟦S⟧	the set of labels of all **break ;** statements that can escape out of S;
in⟦S⟧	the set of program points inside S (including at⟦S⟧ but excluding aft⟦S⟧ and brk-to⟦S⟧);
labs⟦S⟧	the potentially reachable program points while executing S either at, in, or after the statement, or resulting from a break.

Prefix trace semantics Prefix traces are non-empty finite sequences $\pi \in \mathbb{S}^+$ of states where states $\langle \ell, \rho \rangle \in \mathbb{S} \triangleq (\mathbb{L} \times \mathbb{E}\mathbb{v})$ are pairs of a program label $\ell \in \mathbb{S}$ designating the next action to be executed in the program and an environment $\rho \in \mathbb{E}\mathbb{v} \triangleq \mathbb{X} \to \mathbb{V}$ assigning values $\rho(\mathtt{x}) \in \mathbb{V}$ to variables $\mathtt{x} \in \mathbb{X}$. A trace π can be finite $\pi \in \mathbb{S}^+$ or infinite $\pi \in \mathbb{S}^\infty$ (recording a non-terminating computation) so $\mathbb{S}^{+\infty} \triangleq \mathbb{S}^+ \cup \mathbb{S}^\infty$. Trace concatenation $\frown$ is defined as follows

$$\begin{array}{ll} \pi_1\sigma_1 \frown \sigma_2\pi_2 & \text{undefined if } \sigma_1 \neq \sigma_2 \quad \pi_1 \frown \sigma_2\pi_2 \triangleq \pi_1 \text{ if } \pi_1 \in \mathbb{S}^\infty \text{ is infinite} \\ \pi_1\sigma_1 \frown \sigma_1\pi_2 \triangleq \pi_1\sigma_1\pi_2 & \text{if } \pi_1 \in \mathbb{T}^+ \text{ is finite} \end{array}$$

In pattern matching, we sometimes need the empty trace $\ni$. For example if $\sigma\pi\sigma' = \sigma$ then $\pi = \ni$ and so $\sigma = \sigma'$.

Formal definition of the prefix trace semantics The prefix trace semantics $\mathcal{S}^*[\![\mathtt{S}]\!]$ is given structurally (by induction on the syntax) using fixpoints for the iteration.

- *The prefix traces of an assignment statement* $\mathtt{S} ::= \ell\ \mathtt{x = A\ ;}$ (where $\mathsf{at}[\![\mathtt{S}]\!] = \ell$) either stops in an initial state $\langle \ell, \rho \rangle$ or is this initial state $\langle \ell, \rho \rangle$ followed by the next state $\langle \mathsf{aft}[\![\mathtt{S}]\!], \rho[\mathtt{x} \leftarrow \mathcal{A}[\![\mathtt{A}]\!]\rho] \rangle$ recording the assignment of the value $\mathcal{A}[\![\mathtt{A}]\!]\rho$ of the arithmetic expression to variable $\mathtt{x}$ when reaching the label $\mathsf{aft}[\![\mathtt{S}]\!]$ after the assignment.

$$\mathcal{S}^*[\![\mathtt{S}]\!] = \{\langle \ell, \rho \rangle \mid \rho \in \mathbb{E}\mathbb{v}\} \cup \{\langle \ell, \rho \rangle \langle \mathsf{aft}[\![\mathtt{S}]\!], \rho[\mathtt{x} \leftarrow \mathcal{A}[\![\mathtt{A}]\!]\rho] \rangle \mid \rho \in \mathbb{E}\mathbb{v}\} \tag{1}$$

The value of an arithmetic expression $\mathtt{A}$ in environment $\rho \in \mathbb{E}\mathbb{v} \triangleq \mathbb{X} \to \mathbb{V}$ is $\mathcal{A}[\![\mathtt{A}]\!]\rho \in \mathbb{V}$:

$$\mathcal{A}[\![\mathtt{1}]\!]\rho \triangleq 1 \qquad \mathcal{A}[\![\mathtt{x}]\!]\rho \triangleq \rho(\mathtt{x}) \qquad \mathcal{A}[\![\mathtt{A}_1 - \mathtt{A}_2]\!]\rho \triangleq \mathcal{A}[\![\mathtt{A}_1]\!]\rho - \mathcal{A}[\![\mathtt{A}_2]\!]\rho \tag{2}$$

- *The prefix trace semantics of a break statement* $\mathtt{S} ::= \ell\ \mathtt{break\ ;}$ either stops at ℓ or goes on to the break label $\mathsf{brk\text{-}to}[\![\mathtt{S}]\!]$ (which is defined as the exit label of the closest enclosing iteration).

$$\mathcal{S}^*[\![\mathtt{S}]\!] \triangleq \{\langle \ell, \rho \rangle \mid \rho \in \mathbb{E}\mathbb{v}\} \cup \{\langle \ell, \rho \rangle \langle \mathsf{brk\text{-}to}[\![\mathtt{S}]\!], \rho \rangle \mid \rho \in \mathbb{E}\mathbb{v}\} \tag{3}$$

- *The prefix trace semantics of a conditional statement* $\mathtt{S} ::= \mathtt{if}\ \ell\ \mathtt{(B)}\ \mathtt{S}_t$ is
  - either the trace $\langle \ell, \rho \rangle$ when the observation of the execution stops on entry of the program component for initial environment ρ;
  - or, when the value of the boolean expression $\mathtt{B}$ for ρ is false $\mathtt{ff}$, the initial state $\langle \ell, \rho \rangle$ followed by the state $\langle \mathsf{aft}[\![\mathtt{S}]\!], \rho \rangle$ at the label $\mathsf{aft}[\![\mathtt{S}]\!]$ after the conditional statement;
  - or finally, when the value of the boolean expression $\mathtt{B}$ for ρ is true $\mathtt{tt}$, the initial state$\langle \ell, \rho \rangle$ followed by a prefix trace of $\mathtt{S}_t$ starting $\mathsf{at}[\![\mathtt{S}_t]\!]$ in environment ρ (and possibly ending $\mathsf{aft}[\![\mathtt{S}_t]\!] = \mathsf{aft}[\![\mathtt{S}]\!]$).

$$\begin{array}{rl} \widehat{\mathcal{S}}^*[\![\mathtt{S}]\!] \triangleq & \{\langle \ell, \rho \rangle \mid \rho \in \mathbb{E}\mathbb{v}\} \cup \{\langle \ell, \rho \rangle \langle \mathsf{aft}[\![\mathtt{S}]\!], \rho \rangle \mid \mathcal{B}[\![\mathtt{B}]\!]\rho = \mathtt{ff}\} \\ & \cup \{\langle \ell, \rho \rangle \langle \mathsf{at}[\![\mathtt{S}_t]\!], \rho \rangle \pi \mid \mathcal{B}[\![\mathtt{B}]\!]\rho = \mathtt{tt} \wedge \langle \mathsf{at}[\![\mathtt{S}_t]\!], \rho \rangle \pi \in \widehat{\mathcal{S}}^*[\![\mathtt{S}_t]\!]\} \end{array} \tag{4}$$

Observe that definition (4) includes the case of a conditional within an iteration and containing a break statement in the true branch $\mathtt{S}_t$. Since $\mathsf{brk\text{-}to}[\![\mathtt{S}]\!] =$

$\mathsf{brk\text{-}to}[\![\mathtt{S}_t]\!]$, from $\langle \mathsf{at}[\![\mathtt{S}_t]\!],\ \rho\rangle\pi\langle \mathsf{brk\text{-}to}[\![\mathtt{S}_t]\!],\ \rho'\rangle \in \mathscr{S}^*[\![\mathtt{S}_t]\!]$ and $\mathscr{B}[\![\mathtt{B}]\!]\rho = \mathtt{tt}$, we infer that $\langle \mathsf{at}[\![\mathtt{S}]\!],\ \rho\rangle\langle \mathsf{at}[\![\mathtt{S}_t]\!],\ \rho\rangle\pi\langle \mathsf{brk\text{-}to}[\![\mathtt{S}]\!],\ \rho'\rangle \in \mathscr{S}^*[\![\mathtt{S}]\!]$.

- The prefix traces of the *prefix trace semantics of a non-empty statement list* $\mathtt{Sl} ::= \mathtt{Sl}'\ \mathtt{S}$ are the prefix traces of $\mathtt{Sl}'$ or the finite maximal traces of $\mathtt{Sl}'$ followed by a prefix trace of $\mathtt{S}$.

$$\begin{aligned} \widehat{\mathscr{S}}^*[\![\mathtt{Sl}]\!] &\triangleq \widehat{\mathscr{S}}^*[\![\mathtt{Sl}']\!] \cup \widehat{\mathscr{S}}^*[\![\mathtt{Sl}']\!] \mathbin{\hat{;}} \mathscr{S}^*[\![\mathtt{S}]\!] \\ \mathscr{S} \mathbin{\hat{;}} \mathscr{S}' &\triangleq \{\pi \mathbin{\hat{;}} \pi' \mid \pi \in \mathscr{S} \wedge \pi' \in \mathscr{S}' \wedge \pi \mathbin{\hat{;}} \pi' \text{ is well-defined}\} \end{aligned} \tag{5}$$

Notice that if $\pi \in \widehat{\mathscr{S}}^*[\![\mathtt{Sl}']\!]$, $\pi' \in \mathscr{S}^*[\![\mathtt{S}]\!]$, and $\pi \mathbin{\hat{;}} \pi' \in \widehat{\mathscr{S}}^*[\![\mathtt{Sl}]\!]$ then the last state of π must be the first state of π' and this state is $\mathsf{at}[\![\mathtt{S}]\!] = \mathsf{aft}[\![\mathtt{Sl}']\!]$ and so the trace π must be a maximal terminating execution of $\mathtt{Sl}'$ *i.e.* $\mathtt{S}$ is executed only if $\mathtt{Sl}'$ terminates.

- *The prefix finite trace semantic definition* $\mathscr{S}^*[\![\mathtt{S}]\!]$ (7) *of an iteration statement of the form* $\mathtt{S} ::= \mathtt{while}^{\ell}\ \mathtt{(B)}\ \mathtt{S}_b$ where $\ell = \mathsf{at}[\![\mathtt{S}]\!]$ is the $\subseteq$-least solution $\mathsf{lfp}^{\subseteq}\, \mathscr{F}^*[\![\mathtt{S}]\!]$ to the equation $X = \mathscr{F}^*[\![\mathtt{S}]\!](X)$. Since $\mathscr{F}^*[\![\mathtt{S}]\!] \in \wp(\mathbb{S}^+) \to \wp(\mathbb{S}^+)$ is $\subseteq$- monotone (if $X \subseteq X'$ then $\mathscr{F}^*[\![\mathtt{S}]\!](X) \subseteq \mathscr{F}^*[\![\mathtt{S}]\!](X')$ and $\langle \wp(\mathbb{S}^+), \subseteq, \varnothing, \mathbb{S}^+, \cup, \cap\rangle$ is a complete lattice, $\mathsf{lfp}^{\subseteq}\, \mathscr{F}^*[\![\mathtt{S}]\!]$ exists by Tarski's fixpoint theorem and can be defined as the limit of iterates [15]. In definition (7) of the transformer $\mathscr{F}^*[\![\mathtt{S}]\!]$, case (7.a) corresponds to a loop execution observation stopping on entry, (7.b) corresponds to an observation of a loop exiting after 0 or more iterations, and (7.c) corresponds to a loop execution observation that stops anywhere in the body $\mathtt{S}_b$ after 0 or more iterations. This last case covers the case of an iteration terminated by a break statement (to $\mathsf{aft}[\![\mathtt{S}]\!]$ after the iteration statement).

$$\mathscr{S}^*[\![\mathtt{while}^{\ell}\ \mathtt{(B)}\ \mathtt{S}_b]\!] = \mathsf{lfp}^{\subseteq}\, \mathscr{F}^*[\![\mathtt{while}^{\ell}\ \mathtt{(B)}\ \mathtt{S}_b]\!] \tag{6}$$

$$\begin{aligned} \mathscr{F}^*_{\mathbb{S}}[\![\mathtt{while}^{\ell}\ \mathtt{(B)}\ \mathtt{S}_b]\!]\, X \triangleq\ & \{\langle \ell,\ \rho\rangle \mid \rho \in \mathbb{E}\mathtt{v}\} && \text{(a)} \\ & \cup \{\pi_2\langle \ell',\ \rho\rangle\langle \mathsf{aft}[\![\mathtt{S}]\!],\ \rho\rangle \mid \pi_2\langle \ell',\ \rho\rangle \in X \wedge \mathscr{B}[\![\mathtt{B}]\!]\,\rho = \mathtt{ff} \wedge \ell' = \ell\}^2 && \text{(b)} \\ & \cup \{\pi_2\langle \ell',\ \rho\rangle\langle \mathsf{at}[\![\mathtt{S}_b]\!],\ \rho\rangle \cdot \pi_3 \mid \pi_2\langle \ell',\ \rho\rangle \in X \wedge \mathscr{B}[\![\mathtt{B}]\!]\,\rho = \mathtt{tt} \wedge \\ & \qquad \langle \mathsf{at}[\![\mathtt{S}_b]\!],\ \rho\rangle \cdot \pi_3 \in \mathscr{S}^*[\![\mathtt{S}_b]\!] \wedge \ell' = \ell\} && \text{(c)} \end{aligned}$$

- The other cases are similar.

Semantic properties As usual in abstract interpretation [16], we represent properties of entities in a universe $\mathbb{U}$ by a subset of this universe. So a property of elements of $\mathbb{U}$ belongs to $\wp(\mathbb{U})$. For example "to be a natural" is the property $\mathbb{N} \triangleq \{n \in \mathbb{Z} \mid n \geqslant 0\}$ of the integers $\mathbb{Z}$. The property "n is a natural" is "$n \in \mathbb{N}$". By program component (safety) property, we understand a property of their prefix trace semantics $\mathscr{S}^*[\![\mathtt{S}]\!] \in \wp(\mathbb{S}^+)$. So program properties belong to $\wp(\wp(\mathbb{S}^+))$. The *collecting semantics* is the strongest program property, that is the singleton $\{\mathscr{S}^*[\![\mathtt{S}]\!]\}$.

[2] A definition of the form $d(\vec{x}) \triangleq \{f(\vec{x}') \mid P(\vec{x}', \vec{x})\}$ has the variables $\vec{x}'$ in $P(\vec{x}', \vec{x})$ bound to those of $f(\vec{x}')$ whereas $\vec{x}$ is free in $P(\vec{x}', \vec{x})$ since it appears neither in $f(\vec{x}')$ nor (by assumption) under quantifiers in $P(\vec{x}', \vec{x})$. The $\vec{x}$ of $P(\vec{x}', \vec{x})$ is therefore bound to the $\vec{x}$ of $d(\vec{x})$.

3 Specifying computations by regular expressions

Stephen Cole Kleene introduced regular expressions and finite automata to specify execution traces (called events) of automata (called nerve nets) [31]. Kleene proved in [31] that regular expressions and (non-deterministic) finite automata can describe exactly the same classes of languages (see [43, Ch. 1, Sect. 4]). He noted that not all computable execution traces of nerve nets can be (exactly) represented by a regular expression. The situation is the same for programs for which regular expressions (or equivalently finite automata) can specify a superset of the prefix state trace semantics $\mathcal{S}^*[\![\mathtt{S}]\!]$ of program components $\mathtt{S} \in \mathbb{P}\mathtt{c}$.

Example 1 (Security monitors). An example is Fred Schneider's security monitors [44,35] using a finite automata specification to state requirements of hardware or software systems. They have been used to check for safety program properties at runtime, that is for any given execution trace in the semantics $\mathcal{S}^*[\![\mathtt{S}]\!]$. The safety property specified by the finite automaton or, equivalently, an regular expression is temporal *i.e.* links events occurring at different times in the computation (such as a file must be opened before being accessed and must eventually be closed). □

Syntax of regular expressions Classical regular expressions denote sets of strings using constants (empty string ε, literal characters a, b, *etc.*) and operator symbols (concatenation $\bullet$, alternation $\mid$, repetition zero or more times * or one or more times $^+$). We replace the literal characters by invariant specifications $\mathtt{L} : \mathtt{B}$ stating that boolean expression $\mathtt{B}$ should be true whenever control reaches any program point in the set $\mathtt{L}$ of program labels. The boolean expression $\mathtt{B}$ may depend on program variables $\mathtt{x}, \mathtt{y}, \ldots \in \mathbb{X}$ and their initial values denoted $\underline{\mathtt{x}}, \underline{\mathtt{y}}, \ldots \in \underline{\mathbb{X}}$ where $\underline{\mathbb{X}} \triangleq \{\underline{\mathtt{x}} \mid \mathtt{x} \in \mathbb{X}\}$.

	$\mathtt{L} \in \wp(\mathbb{L})$	sets of program labels
	$\mathtt{x}, \mathtt{y}, \ldots \in \mathbb{X}$	program variables
	$\underline{\mathtt{x}}, \underline{\mathtt{y}}, \ldots \in \underline{\mathbb{X}}$	initial values of variables
	$\mathtt{B} \in \mathbb{B}$	boolean expressions such that $\mathtt{vars}[\![\mathtt{B}]\!] \subseteq \mathbb{X} \cup \underline{\mathbb{X}}$
$\mathtt{R} \in$	$\mathbb{R}$	regular expressions (8)
$\mathtt{R} ::=$	ε	empty
$\mid$	$\mathtt{L} : \mathtt{B}$	invariant $\mathtt{B}$ at $\mathtt{L}$
$\mid$	$\mathtt{R}_1\mathtt{R}_2$ (or $\mathtt{R}_1 \bullet \mathtt{R}_2$)	concatenation
$\mid$	$\mathtt{R}_1 \mid \mathtt{R}_2$	alternative
$\mid$	$\mathtt{R}_1^* \mid \mathtt{R}_1^+$	zero/one or more occurrences of $\mathtt{R}$
$\mid$	$(\mathtt{R}_1)$	grouping

We use abbreviations to designate sets of labels such as $\mathtt{?} : \mathtt{B} \triangleq \mathbb{L} : \mathtt{B}$ so that $\mathtt{B}$ is invariant, $\ell : \mathtt{B} \triangleq \{\ell\} : \mathtt{B}$ so that $\mathtt{B}$ is invariant at program label ℓ, $\neg\ell : \mathtt{B} \triangleq \mathbb{L} \setminus \{\ell\} : \mathtt{B}$ when $\mathtt{B}$ holds everywhere but at program point ℓ, *etc.*

Example 2. $(\texttt{?}\ \texttt{:}\ \texttt{tt})^*$ holds for any program. $(\texttt{?}\ \texttt{:}\ \texttt{x}\ \texttt{>=}\ \texttt{0})^*$ states that the value of $\texttt{x}$ is always positive or zero during program execution. $(\texttt{?}\ \texttt{:}\ \texttt{x}\ \texttt{>=}\ \underline{\texttt{x}})^*$ states that the value of $\texttt{x}$ is always greater than or equal to its initial value $\underline{\texttt{x}}$ during execution. $(\texttt{?}\ \texttt{:}\ \texttt{x}\ \texttt{>=}\ \texttt{0})^* \bullet \ell\ \texttt{:}\ \texttt{x}\ \texttt{==}\ \texttt{0} \bullet (\texttt{?}\ \texttt{:}\ \texttt{x}\ \texttt{<}\ \texttt{0})^*$ states that the value of $\texttt{x}$ should be positive or zero and if program point ℓ is ever reached then $\texttt{x}$ should be 0, and if computations go on after program point ℓ then $\texttt{x}$ should be negative afterwards. □

Example 3. Continuing Ex. 1 for security monitors, the basic regular expressions are names a of program actions. We can understand such an action a as designating the set $\texttt{L}$ of labels of all its occurrences in the program. If necessary, the boolean expression $\texttt{B}$ can be used to specify the parameters of the action. □

There are many regular expressions denoting the language $\{\mathfrak{d}\}$ containing only the empty sequence $\mathfrak{d}$ (such that ε, $\varepsilon\varepsilon$, ε^*, *etc.*), as shown by the following grammar.

$$\mathscr{R}_\varepsilon \ni \texttt{R} ::= \varepsilon \mid \texttt{R}_1\texttt{R}_2 \mid \texttt{R}_1 \mid \texttt{R}_2 \mid \texttt{R}_1^* \mid \texttt{R}_1^+ \mid (\texttt{R}_1) \qquad \text{empty regular expressions}$$

For specification we use only non-empty regular expressions $\texttt{R} \in \mathscr{R}^+$ since traces cannot be empty.

$$\mathscr{R}^+ \ni \texttt{R} ::= \texttt{L}\ \texttt{:}\ \texttt{B} \mid \varepsilon\texttt{R}_2 \mid \texttt{R}_1\varepsilon \mid \texttt{R}_1\texttt{R}_2 \mid \texttt{R}_1 \mid \texttt{R}_2 \mid \texttt{R}_1^+ \mid (\texttt{R}_1) \qquad \text{non-empty r.e.}$$

We also have to consider regular expressions $\texttt{R} \in \mathscr{R}^\downarrow$ containing no alternative $\mid$.

$$\mathscr{R}^\downarrow \ni \texttt{R} ::= \varepsilon \mid \texttt{L}\ \texttt{:}\ \texttt{B} \mid \texttt{R}_1\texttt{R}_2 \mid \texttt{R}_1^* \mid \texttt{R}_1^+ \mid (\texttt{R}_1) \qquad \mid\text{-free regular expressions}$$

Relational semantics of regular expressions The semantics (2) of expressions is changed as follows ($\underline{\varrho}(\texttt{x})$ denotes the initial values $\underline{\texttt{x}}$ of variables $\texttt{x}$ and $\rho(\texttt{x})$ their current value, $\uparrow$ is the alternative denial logical operation)

$$\begin{array}{ll}
\mathscr{A}[\![1]\!]\underline{\varrho}, \rho \triangleq 1 & \mathscr{A}[\![\texttt{A}_1 - \texttt{A}_2]\!]\underline{\varrho}, \rho \triangleq \mathscr{A}[\![\texttt{A}_1]\!]\underline{\varrho}, \rho - \mathscr{A}[\![\texttt{A}_2]\!]\underline{\varrho}, \rho \\
\mathscr{A}[\![\underline{\texttt{x}}]\!]\underline{\varrho}, \rho \triangleq \underline{\varrho}(\texttt{x}) & \mathscr{B}[\![\texttt{A}_1 < \texttt{A}_2]\!]\underline{\varrho}, \rho \triangleq \mathscr{A}[\![\texttt{A}_1]\!]\underline{\varrho}, \rho < \mathscr{A}[\![\texttt{A}_2]\!]\underline{\varrho}, \rho \\
\mathscr{A}[\![\texttt{x}]\!]\underline{\varrho}, \rho \triangleq \rho(\texttt{x}) & \mathscr{B}[\![\texttt{B}_1\ \texttt{nand}\ \texttt{B}_2]\!]\underline{\varrho}, \rho \triangleq \mathscr{B}[\![\texttt{B}_1]\!]\underline{\varrho}, \rho \uparrow \mathscr{B}[\![\texttt{B}_2]\!]\underline{\varrho}, \rho
\end{array} \tag{8}$$

We represent a non-empty finite sequence $\sigma_1 \ldots \sigma_n \in \mathbb{S}^+ \triangleq \bigcup_{n \in \mathbb{N}\setminus\{0\}} [1, n] \to \mathbb{S}$ of states $\sigma_i \in \mathbb{S} \triangleq (\mathbb{L} \times \mathbb{E}\mathbb{v})$ by a map $\sigma \in [1, n] \to \mathbb{S}$ (which is the empty sequence $\sigma = \mathfrak{d}$ when $n = 0$).

The relational semantics $\mathscr{S}^r[\![\texttt{R}]\!] \in \wp(\mathbb{E}\mathbb{v} \times \mathbb{S}^*)$ of regular expressions $\texttt{R}$ relates an arbitrary initial environment $\underline{\varrho} \in \mathbb{E}\mathbb{v}$ to a trace $\pi \in \mathbb{S}^*$ by defining how the states of the trace π are related to that initial environment $\underline{\varrho}$.

$$\begin{array}{ll}
\mathscr{S}^r[\![\varepsilon]\!] \triangleq \{\langle \underline{\varrho}, \mathfrak{d}\rangle \mid \underline{\varrho} \in \mathbb{E}\mathbb{v}\} & \mathscr{S}^r[\![\texttt{R}]\!]^1 \triangleq \mathscr{S}^r[\![\texttt{R}]\!] \\
\mathscr{S}^r[\![\texttt{L}\ \texttt{:}\ \texttt{B}]\!] \triangleq \{\langle \underline{\varrho}, \langle \ell, \rho\rangle\rangle \mid \ell \in \texttt{L} \wedge \mathscr{B}[\![\texttt{B}]\!]\underline{\varrho}, \rho\} & \mathscr{S}^r[\![\texttt{R}]\!]^{n+1} \triangleq \mathscr{S}^r[\![\texttt{R}]\!]^n \circledcirc \mathscr{S}^r[\![\texttt{R}]\!] \\
\mathscr{S}^r[\![\texttt{R}_1\texttt{R}_2]\!] \triangleq \mathscr{S}^r[\![\texttt{R}_1]\!] \circledcirc \mathscr{S}^r[\![\texttt{R}_2]\!] & \mathscr{S}^r[\![\texttt{R}^*]\!] \triangleq \bigcup_{n \in \mathbb{N}} \mathscr{S}^r[\![\texttt{R}]\!]^n \\
\mathscr{S} \circledcirc \mathscr{S}' \triangleq \{\langle \underline{\varrho}, \pi \cdot \pi'\rangle \mid \langle \underline{\varrho}, \pi\rangle \in \mathscr{S} \wedge \langle \underline{\varrho}, \pi'\rangle \in \mathscr{S}'\} & \mathscr{S}^r[\![\texttt{R}^+]\!] \triangleq \bigcup_{n \in \mathbb{N}\setminus\{0\}} \mathscr{S}^r[\![\texttt{R}]\!]^n \\
\mathscr{S}^r[\![\texttt{R}_1 \mid \texttt{R}_2]\!] \triangleq \mathscr{S}^r[\![\texttt{R}_1]\!] \cup \mathscr{S}^r[\![\texttt{R}_2]\!] & \mathscr{S}^r[\![(\texttt{R})]\!] \triangleq \mathscr{S}^r[\![\texttt{R}]\!]
\end{array} \tag{9}$$

Example 4. The semantics of the regular expression $\mathsf{R} \triangleq \ell : \mathtt{x} = \underline{\mathtt{x}} \bullet \ell' : \mathtt{x} = \underline{\mathtt{x}} + 1$ is $\mathcal{S}^r[\![\mathsf{R}]\!] = \{\langle \underline{\varrho}, \langle \ell, \rho\rangle\langle \ell', \rho'\rangle\rangle \mid \rho(\mathtt{x}) = \underline{\varrho}(\mathtt{x}) \wedge \rho'(\mathtt{x}) = \underline{\varrho}(\mathtt{x}) + 1\}$. □

4 Definition of regular model checking

Let the prefix closure $\mathsf{prefix}(\Pi)$ of a set $\Pi \in \wp(\mathbb{E}^\nu \times \mathbb{S}^+)$ of stateful traces be

$$\mathsf{prefix}(\Pi) \triangleq \{\langle \underline{\varrho}, \pi\rangle \mid \pi \in \mathbb{S}^+ \wedge \exists \pi' \in \mathbb{S}^* \,.\, \langle \underline{\varrho}, \pi \cdot \pi'\rangle \in \Pi\} \qquad \text{prefix closure.}$$

The following Def. 1 defines the model checking problem $\mathsf{P}, \underline{\varrho} \vDash \mathsf{R}$ as checking that the semantics of the given program $\mathsf{P} \in \mathbb{P}$ meets the regular specification $\mathsf{R} \in \mathbb{R}^+$ for the initial environment $\underline{\varrho}$[3].

Definition 1 (Model checking).

$$\mathsf{P}, \underline{\varrho} \vDash \mathsf{R} \triangleq (\{\underline{\varrho}\} \times \mathcal{S}^*[\![\mathsf{P}]\!]) \quad \subseteq \quad \mathsf{prefix}(\mathcal{S}^r[\![\mathsf{R} \bullet (? : \mathtt{tt})^*]\!])$$

The prefix closure prefix allows the regular specification R to specify traces satisfying a prefix of the specification only, as in $\ell\ \mathtt{x = x + 1}\ ;\ell' \vDash \ell : \mathtt{x} = \underline{\mathtt{x}} \bullet \ell' : \mathtt{x} = \underline{\mathtt{x}} + 1 \bullet \ell'' : \mathtt{x} = \underline{\mathtt{x}} + 3$. The extension of the specification by $(? : \mathtt{tt})^*$ allows for the regular specification R to specify only a prefix of the traces, as in $\ell\ \mathtt{x = x + 1}\ ;\ell'\ \mathtt{x = x + 2}\ ;\ell'' \vDash \ell : \mathtt{x} = \underline{\mathtt{x}} \bullet \ell' : \mathtt{x} = \underline{\mathtt{x}} + 1$. Model checking is a boolean abstraction $\langle \wp(\mathbb{S}^+), \subseteq\rangle \underset{\alpha_{\underline{\varrho},\mathsf{R}}}{\overset{\gamma_{\underline{\varrho},\mathsf{R}}}{\leftrightarrows}} \langle \mathbb{B}, \Leftarrow\rangle$ where $\alpha_{\underline{\varrho},\mathsf{R}}(\Pi) \triangleq (\{\underline{\varrho}\} \times \Pi) \subseteq \mathsf{prefix}(\mathcal{S}^r[\![\mathsf{R} \bullet (? : \mathtt{tt})^*]\!])$.

5 Properties of regular expressions

Equivalence of regular expressions We say that regular expressions are equivalent when they have the same semantics *i.e.* $\mathsf{R}_1 \backsimeq \mathsf{R}_2 \triangleq (\mathcal{S}^r[\![\mathsf{R}_1]\!] = \mathcal{S}^r[\![\mathsf{R}_2]\!])$.

Disjunctive normal form dnf of regular expressions As noticed by Kleene [31, p. 14], regular expressions can be put in the equivalent disjunctive normal form of Hilbert—Ackermann. A regular expression is in disjunctive normal form if it is of the form $(\mathsf{R}_1 \mid \ldots \mid \mathsf{R}_n)$ for some $n \geqslant 1$, in which none of the R_i, for $1 \leqslant i \leqslant n$, contains an occurrence of $\mid$. Any regular expression R has a disjunctive normal form $\mathsf{dnf}(\mathsf{R})$ defined as follows.

$$\begin{array}{rlrl}
\mathsf{dnf}(\varepsilon) &\triangleq \varepsilon & \mathsf{dnf}(\mathsf{L} : \mathsf{B}) &\triangleq \mathsf{L} : \mathsf{B} \\
\mathsf{dnf}(\mathsf{R}_1 \mid \mathsf{R}_2) &\triangleq \mathsf{dnf}(\mathsf{R}_1) \mid \mathsf{dnf}(\mathsf{R}_2) & \mathsf{dnf}(\mathsf{R}^+) &\triangleq \mathsf{dnf}(\mathsf{R}\mathsf{R}^*) \\
\mathsf{dnf}(\mathsf{R}^*) &\triangleq \mathsf{let}\ \mathsf{R}^1 \mid \ldots \mid \mathsf{R}^n = \mathsf{dnf}(\mathsf{R})\ \mathsf{in}\ ((\mathsf{R}^1)^* \ldots (\mathsf{R}^n)^*)^* & \mathsf{dnf}((\mathsf{R})) &\triangleq (\mathsf{dnf}(\mathsf{R})) \\
\mathsf{dnf}(\mathsf{R}_1\mathsf{R}_2) &\triangleq \mathsf{let}\ \mathsf{R}_1^1 \mid \ldots \mid \mathsf{R}_1^{n_1} = \mathsf{dnf}(\mathsf{R}_1)\ \mathsf{and}\ \mathsf{R}_2^1 \mid \ldots \mid \mathsf{R}_2^{n_2} = \mathsf{dnf}(\mathsf{R}_2)\ \mathsf{in} \displaystyle\bigmid_{i=1}^{n_1} \bigmid_{j=1}^{n_2} \mathsf{R}_1^i\mathsf{R}_2^j & &
\end{array}$$

[3] We understand "regular model checking" as checking temporal specifications given by a regular expression. This is different from [1] model checking transition systems which states are regular word or tree languages.

The Lem. 1 below shows that normalization leaves the semantics unchanged. It uses the fact that $(\mathtt{R}_1 \mid \mathtt{R}_2)^* \eqsim (\mathtt{R}_1{}^*\mathtt{R}_2{}^*)^*$ where the $\mathtt{R}_1$ and $\mathtt{R}_2$ do not contain any $\mid$ [29, Sect. 3.4.6, p. 118]. It shows that normalization in (12) can be further simplified by $\varepsilon\mathtt{R} \eqsim \mathtt{R}\varepsilon \eqsim \mathtt{R}$ and $(\varepsilon)^* \eqsim \varepsilon$ which have equivalent semantics.

Lemma 1. $\mathsf{dnf}(\mathtt{R}) \eqsim \mathtt{R}$.

first and next of regular expressions Janusz Brzozowski [7] introduced the notion of derivation for regular expressions (extended with arbitrary Boolean operations). The derivative of a regular expression $\mathtt{R}$ with respect to a symbol a, typically denoted as $D_a(\mathtt{R})$ or $a^{-1}\mathtt{R}$, is a regular expression given by a simple recursive definition on the syntactic structure of $\mathtt{R}$. The crucial property of these derivatives is that a string of the form $a\sigma$ (starting with the symbol a) matches an expression $\mathtt{R}$ iff the suffix σ matches the derivative $D_a(\mathtt{R})$ [7,38,2].

Following this idea, assume that a non-empty regular expression $\mathtt{R} \in \mathscr{R}^+$ has been decomposed into disjunctive normal form $(\mathtt{R}_1 \mid \ldots \mid \mathtt{R}_n)$ for some $n \geqslant 1$, in which none of the $\mathtt{R}_i$, for $i \in [1,n]$, contains an occurrence of $\mid$. We can further decompose each $\mathtt{R}_i \in \mathscr{R}^+ \cap \mathscr{R}^\downarrow$ into $\langle \mathtt{L} : \mathtt{B}, \mathtt{R}'_i \rangle = \mathsf{fstnxt}(\mathtt{R}_i)$ such that

- $\mathtt{L} : \mathtt{B}$ recognizes the first state of sequences of states recognized by $\mathtt{R}_i$;
- the regular expression $\mathtt{R}'_i$ recognizes sequences of states after the first state of sequences of states recognized by $\mathtt{R}_i$.

We define fstnxt for non-empty $\mid$-free regular expressions $\mathtt{R} \in \mathscr{R}^+ \cap \mathscr{R}^\downarrow$ by structural induction, as follows.

$$
\begin{aligned}
&\mathsf{fstnxt}(\mathtt{L} : \mathtt{B}) \triangleq \langle \mathtt{L} : \mathtt{B}, \varepsilon \rangle && (10)\\
&\mathsf{fstnxt}(\mathtt{R}_1\mathtt{R}_2) \triangleq \mathsf{fstnxt}(\mathtt{R}_2) && \text{if } \mathtt{R}_1 \in \mathscr{R}_\varepsilon\\
&\mathsf{fstnxt}(\mathtt{R}_1\mathtt{R}_2) \triangleq \mathsf{let}\ \langle \mathtt{R}_1^f, \mathtt{R}_1^n \rangle = \mathsf{fstnxt}(\mathtt{R}_1)\ \mathsf{in}\ \langle\!|\, \mathtt{R}_1^n \in \mathscr{R}_\varepsilon\ ?\ \langle \mathtt{R}_1^f, \mathtt{R}_2 \rangle : \langle \mathtt{R}_1^f, \mathtt{R}_1^n \bullet \mathtt{R}_2 \rangle \,|\!\rangle && \text{if } \mathtt{R}_1 \notin \mathscr{R}_\varepsilon\\
&\mathsf{fstnxt}(\mathtt{R}^+) \triangleq \mathsf{let}\ \langle \mathtt{R}^f, \mathtt{R}^n \rangle = \mathsf{fstnxt}(\mathtt{R})\ \mathsf{in}\ \langle\!|\, \mathtt{R}^n \in \mathscr{R}_\varepsilon\ ?\ \langle \mathtt{R}^f, \mathtt{R}^* \rangle : \langle \mathtt{R}^f, \mathtt{R}^n \bullet \mathtt{R}^* \rangle \,|\!\rangle\\
&\mathsf{fstnxt}((\mathtt{R})) \triangleq \mathsf{fstnxt}(\mathtt{R})
\end{aligned}
$$

The following Lem. 2 shows the equivalence of an alternative-free regular expression and its first and next decomposition.

Lemma 2. *Let* $\mathtt{R} \in \mathscr{R}^+ \cap \mathscr{R}^\downarrow$ *be a non-empty* $\mid$*-free regular expression and* $\langle \mathtt{L} : \mathtt{B}, \mathtt{R}' \rangle = \mathsf{fstnxt}(\mathtt{R})$. *Then* $\mathtt{R}' \in \mathscr{R}^\downarrow$ *is* $\mid$*-free and* $\mathtt{R} \eqsim \mathtt{L} : \mathtt{B} \bullet \mathtt{R}'$.

6 The model checking abstraction

The model checking abstraction in Section 4 is impractical for structural model checking since *e.g.* when checking that a trace concatenation $\pi_1{}^\frown\pi_2$ of a statement list $\mathtt{Sl} ::= \mathtt{Sl}'\ \mathtt{S}$ for a specification $\mathtt{R}$ where π_1 is a trace of $\mathtt{Sl}'$ and π_2 is a trace of $\mathtt{S}$, we first check that π_1 satisfies $\mathtt{R}$ and then we must check π_2 for a continuation $\mathtt{R}_2$ of $\mathtt{R}$ which should be derived from π_1 and $\mathtt{R}$. This is not provided by the boolean abstraction $\underline{\alpha}_{\varrho,\mathtt{R}}$ which needs to be refined as shown below.

Example 5. Assume we want to check ℓ_1 `x = x + 1 ;`ℓ_2 `x = x + 2 ;`ℓ_3 for the regular specification `? : x =` $\underline{\mathtt{x}}$ `•? : x =` $\underline{\mathtt{x}}+1$ `•? : x =` $\underline{\mathtt{x}}+3$ by first checking the first statement and then the second. Knowing the boolean information that ℓ_1 `x = x + 1 ;`ℓ_2 model checks for `? : x =` $\underline{\mathtt{x}}$ `•? : x =` $\underline{\mathtt{x}}+1$ is not enough. We must also know what to check the continuation ℓ_2 `x = x + 2 ;`ℓ_3 for. (This is `? : x =` $\underline{\mathtt{x}}+1$ `•? : x =` $\underline{\mathtt{x}}+3$ that is if `x` is equal to the initial value plus 1 at ℓ_2, it is equal to this initial value plus 3 at ℓ_3.) □

The model-checking $\mathscr{M}^t\langle\underline{\varrho},\ \mathtt{R}\rangle\pi$ of a trace π with initial environment $\underline{\varrho}$ for a `|`-free specification $\mathtt{R}\in\mathscr{R}^{\downarrow}$ is a pair $\langle b,\ \mathtt{R}'\rangle$ where the boolean b states whether the specification `R` holds for the trace π and `R′` specifies the possible continuations of π according to `R`, ε if none.

Example 6. For `Sl` $=\ell_1$ `x = x + 1 ;`ℓ_2 `x = x + 2 ;`ℓ_3, we have $\mathscr{S}^*[\![\mathtt{Sl}]\!]=\{\langle\ell_1,\ \rho\rangle\langle\ell_2,\ \rho[\mathtt{x}\leftarrow\rho(\mathtt{x})+1]\rangle\langle\ell_3,\ \rho[\mathtt{x}\leftarrow\rho(\mathtt{x})+3]\rangle\mid\rho\in\mathbb{E}\mathbb{v}\}$ and $\mathscr{M}^t\langle\rho,$ `? : x =` $\underline{\mathtt{x}}$ `•? : x =` $\underline{\mathtt{x}}+1$ `•? : x =` $\underline{\mathtt{x}}+3\rangle(\langle\ell_1,\ \rho\rangle\langle\ell_2,\ \rho[\mathtt{x}\leftarrow\rho(\mathtt{x})+1]\rangle\langle\ell_3,\ \rho[\mathtt{x}\leftarrow\rho(\mathtt{x})+3]\rangle)=\langle\mathtt{tt},\ \varepsilon\rangle$ (we have ignored the initial empty statement list in `Sl` to simplify the specification). □

The fact that $\mathscr{M}^t\langle\underline{\varrho},\ \mathtt{R}\rangle\pi$ returns a pair $\langle b,\ \mathtt{R}'\rangle$ where `R′` is to be satisfied by continuations of π allows us to perform program model checking by structural induction on the program in Section **8**. The formal definition is the following.

Definition 2 (Regular model checking).

– *Trace model checking ($\underline{\varrho}\in\mathbb{E}\mathbb{v}$ is an initial environment and $\mathtt{R}\in\mathscr{R}^{+}\cap\mathscr{R}^{\downarrow}$ is a non-empty and `|`-free regular expression):*

$$\mathscr{M}^t\langle\underline{\varrho},\ \varepsilon\rangle\pi\triangleq\langle\mathtt{tt},\ \varepsilon\rangle \tag{11}$$

$$\mathscr{M}^t\langle\underline{\varrho},\ \mathtt{R}\rangle\ni\triangleq\langle\mathtt{tt},\ \mathtt{R}\rangle$$

$$\mathscr{M}^t\langle\underline{\varrho},\ \mathtt{R}\rangle\pi\triangleq\mathsf{let}\ \langle\ell_1,\ \rho_1\rangle\pi'=\pi\ \mathsf{and}\ \langle\mathtt{L}\ \mathtt{:}\ \mathtt{B},\ \mathtt{R}'\rangle=\mathsf{fstnxt}(\mathtt{R})\ \mathsf{in}\qquad\pi\neq\ni$$
$$[\![\ \langle\underline{\varrho},\ \langle\ell_1,\ \rho_1\rangle\rangle\in\mathscr{S}^r[\![\mathtt{L}\ \mathtt{:}\ \mathtt{B}]\!]\ ?\ \mathscr{M}^t\langle\underline{\varrho},\ \mathtt{R}'\rangle\pi'\ \wr\ \langle\mathtt{ff},\ \mathtt{R}'\rangle\ ]\!]$$

– *Set of traces model checking (for an `|`-free regular expression $\mathtt{R}\in\mathscr{R}^{\downarrow}$):*

$$\mathscr{M}^{\downarrow}\langle\underline{\varrho},\ \mathtt{R}\rangle\Pi\triangleq\{\langle\pi,\ \mathtt{R}'\rangle\mid\pi\in\Pi\wedge\langle\mathtt{tt},\ \mathtt{R}'\rangle=\mathscr{M}^t\langle\underline{\varrho},\ \mathtt{R}\rangle\pi\} \tag{12}$$

– *Program component $\mathtt{S}\in\mathscr{P}\mathscr{c}$ model checking (for an `|`-free regular expression $\mathtt{R}\in\mathscr{R}^{\downarrow}$):*

$$\mathscr{M}^{\downarrow}[\![\mathtt{S}]\!]\langle\underline{\varrho},\ \mathtt{R}\rangle\triangleq\mathscr{M}^{\downarrow}\langle\underline{\varrho},\ \mathtt{R}\rangle(\mathscr{S}^*[\![\mathtt{S}]\!]) \tag{13}$$

– *Set of traces model checking (for regular expression $\mathtt{R}\in\mathscr{R}$):*

$$\mathscr{M}\langle\underline{\varrho},\ \mathtt{R}\rangle\Pi\triangleq\mathsf{let}\ (\mathtt{R}_1\ \mathtt{|}\ \ldots\ \mathtt{|}\ \mathtt{R}_n)=\mathsf{dnf}(\mathtt{R})\ \mathsf{in}\qquad\bigcup_{i=1}^{n}\{\pi\mid\exists\mathtt{R}'\in\mathscr{R}\,.\,\langle\pi,\ \mathtt{R}'\rangle\in\mathscr{M}^{\downarrow}\langle\underline{\varrho},\ \mathtt{R}_i\rangle\Pi\} \tag{14}$$

– *Model checking of a program component* $\mathtt{S} \in \mathbb{P}\mathfrak{c}$ *(for regular expression* $\mathtt{R} \in \mathbb{R}$*):*

$$\mathscr{M}[\![\mathtt{S}]\!]\langle \underline{\varrho}, \mathtt{R}\rangle \triangleq \mathscr{M}\langle \underline{\varrho}, \mathtt{R}\rangle(\mathscr{S}^*[\![\mathtt{S}]\!]) \qquad (18) \quad \square$$

The model checking $\mathscr{M}^t\langle \underline{\varrho}, \mathtt{R}\rangle\pi$ of a stateful trace π in (14) returns a pair $\langle b, \mathtt{R}'\rangle$ specifying whether π satisfies the specification $\mathtt{R}$ (when $b = \mathtt{tt}$) or not (when $b = \mathtt{ff}$). So if $\mathscr{M}^t\langle \rho, \mathtt{R}\rangle(\pi) = \langle \mathtt{ff}, \mathtt{R}'\rangle$ in (15) then the trace π is a counter example to the specification $\mathtt{R}$. $\mathtt{R}'$ specifies what a continuation π' of π would have to satisfy for $\pi \fatsemi \pi'$ to satisfy $\mathtt{R}$ (nothing specific when $\mathtt{R}' = \varepsilon$).

Notice that $\mathscr{M}^t\langle \underline{\varrho}, \mathtt{R}\rangle\pi$ checks whether the given trace π satisfies the regular specification $\mathtt{R}$ for initial environment $\underline{\varrho}$. Because only one trace is involved, this check can be done at runtime using a monitoring of the program execution. This is the case Fred Schneider's security monitors [44] in Ex. 1 (using an equivalent specification by finite automata).

The set of traces model checking $\mathscr{M}^\natural\langle \underline{\varrho}, \mathtt{R}\rangle\Pi$ returns the subset of traces of Π satisfying the specification $\mathtt{R}$ for the initial environment $\underline{\varrho}$. Since all program executions $\mathscr{S}^*[\![\mathtt{P}]\!]$ are involved, the model checking $\mathscr{M}^\natural[\![\mathtt{P}]\!]\langle \underline{\varrho}, \mathtt{R}\rangle$ of a program $\mathtt{P}$ becomes, by Rice theorem, undecidable.

The regular specification $\mathtt{R}$ is relational in that it may relate the initial and current states (or else may only assert a property of the current states when $\mathtt{R}$ never refer to the initial environment $\underline{\varrho}$). If $\pi\langle \ell, \rho\rangle\pi' \in \mathscr{S}^*[\![\mathtt{S}]\!]$ is an execution trace satisfying the specification $\mathtt{R}$ then $\mathtt{R}$ in (18) determines a relationship between the initial environment $\underline{\varrho}$ and the current environment ρ. For example $\mathtt{R} = \langle\{\mathsf{at}[\![\mathtt{S}]\!]\}, \mathtt{B}\rangle \bullet \mathtt{R}'$ with $\mathscr{B}[\![\mathtt{B}]\!]\underline{\varrho}, \rho = \forall \mathtt{x} \in \mathbb{X} \,.\, \underline{\varrho}(\underline{\mathtt{x}}) = \rho(\mathtt{x})$ expresses that the initial values of variables $\mathtt{x}$ are denoted $\underline{\mathtt{x}}$. $\mathscr{B}[\![\mathtt{B}]\!]\underline{\varrho}, \rho = \mathtt{tt}$ would state that there is no constraint on the initial value of variables. The difference with the invariant specifications of is that the order of computations is preserved. $\mathtt{R}$ can specify in which order program points may be reached, which is impossible with invariants [4].

The model checking abstraction (15) which, given an initial environment $\underline{\varrho} \in \mathbb{E}\mathtt{v}$ and an $\mid$-free regular specification $\mathtt{R} \in \mathbb{R}^\natural$, returns the set of traces satisfying this specification is the lower adjoint of the Galois connection

$$\langle \wp(\mathbb{S}^+), \subseteq\rangle \underset{\mathscr{M}^\natural\langle \underline{\varrho}, \mathtt{R}\rangle}{\overset{\gamma_{\mathscr{M}^\natural\langle \underline{\varrho}, \mathtt{R}\rangle}}{\leftrightarrows}} \langle \wp(\mathbb{S}^+), \subseteq\rangle \quad \text{for } \mathtt{R} \in \mathbb{R}^\natural \text{ in (15)} \qquad (16)$$

[4] By introduction of an auxiliary variable $\mathtt{C}$ incremented at each program step one can simulate a trace with invariants. But then the reasoning is not on the original program $\mathtt{P}$ but on a transformed program $\overline{\mathtt{P}}$. Invariants in $\overline{\mathtt{P}}$ holding for a given value of c of $\mathtt{C}$ also hold at the position c of the traces in $\mathtt{P}$. This king of indirect reasoning is usually heavy and painful to maintain when programs are modified since values of counters are no longer the same. The use of temporal specifications has the advantage of avoiding the reasoning on explicit positions in the trace.

If $\langle C, \leqslant \rangle$ is a poset, $\langle \mathcal{A}, \sqsubseteq, \sqcup, \sqcap \rangle$ is a complete lattice, $\forall i \in [1, n]$. $\langle C, \leqslant \rangle \underset{\alpha_i}{\overset{\gamma_i}{\leftrightarrows}} \langle \mathcal{A}, \sqsubseteq \rangle$ then $\langle C, \leqslant \rangle \underset{\alpha}{\overset{\gamma}{\leftrightarrows}} \langle \mathcal{A}, \sqsubseteq \rangle$ where $\alpha \triangleq \dot{\bigsqcup_{i=1}^{n}} \alpha_i$ and $\gamma = \dot{\bigsqcap_{i=1}^{n}} \gamma_i$, pointwise. This implies that

$$\langle \wp(\mathbb{S}^+), \subseteq \rangle \underset{\mathscr{M}\langle \underline{\varrho}, \mathtt{R} \rangle}{\overset{\gamma_{\mathscr{M}\langle \underline{\varrho}, \mathtt{R} \rangle}}{\leftrightarrows}} \langle \wp(\mathbb{S}^+), \subseteq \rangle \quad \text{for } \mathtt{R} \in \mathbb{R} \text{ in (17)} \tag{17}$$

To follow the tradition that model checking returns a boolean answer this abstraction can be composed with the boolean abstraction

$$\langle \wp(\mathbb{S}^+), \subseteq \rangle \underset{\alpha_{\mathscr{M}\langle \underline{\varrho}, \mathtt{R} \rangle}}{\overset{\gamma_{\mathscr{M}\langle \underline{\varrho}, \mathtt{R} \rangle}}{\leftrightarrows}} \langle \mathbb{B}, \Leftarrow \rangle \tag{18}$$

where $\alpha_{\mathscr{M}\langle \underline{\varrho}, \mathtt{R} \rangle}(X) \triangleq (\{\underline{\varrho}\} \times X) \subseteq \mathscr{M}\langle \underline{\varrho}, \mathtt{R} \rangle(X)$.

7 Soundness and completeness of the model checking abstraction

The following Th. 1 shows that the Def. 1 of model checking a program semantics for a regular specification is a sound and complete abstraction of this semantics.

Theorem 1 (Model checking soundness ($\Leftarrow$) and completeness ($\Rightarrow$)).

$$\mathtt{P}, \underline{\varrho} \vDash \mathtt{R} \Leftrightarrow \alpha_{\mathscr{M}\langle \underline{\varrho}, \mathtt{R} \rangle}(\mathscr{S}^*[\![\mathtt{P}]\!])$$

At this point we know, by (18) and Th. 1 that a model checker $\mathscr{M}[\![\mathtt{S}]\!]\langle \underline{\varrho}, \mathtt{R} \rangle$ is a sound and complete abstraction $\mathscr{M}\langle \underline{\varrho}, \mathtt{R} \rangle(\mathscr{S}^*[\![\mathtt{S}]\!])$ of the program component semantics $\mathscr{S}^*[\![\mathtt{S}]\!]$ which provides a counter example in case of failure. This allows us to derive a structural model checker $\widehat{\mathscr{M}}[\![\mathtt{P}]\!]\langle \underline{\varrho}, \mathtt{R} \rangle$ in Section 8 by calculational design.

8 Structural model checking

By Def. 1 of the model checking of $\mathtt{S}, \underline{\varrho} \vDash \mathtt{R}$ of a program $\mathtt{P} \in \mathbb{P}$ for a regular specification $\mathtt{R} \in \mathbb{R}^+$ and initial environment $\underline{\varrho}$, Th. 1 shows that a model checker $\mathscr{M}[\![\mathtt{P}]\!]\langle \underline{\varrho}, \mathtt{R} \rangle$ is a sound and complete abstraction $\mathscr{M}\langle \underline{\varrho}, \mathtt{R} \rangle(\mathscr{S}^*[\![\mathtt{P}]\!])$ of the program semantics $\mathscr{S}^*[\![\mathtt{P}]\!]$. This abstraction does not provide a model checking algorithm specification.

The standard model checking algorithms [10] use a transition system (or a Kripke structure variation [32]) for hardware and software modeling and proceeding by induction on computation steps.

In contrast, we proceed by structural induction on programs, which will be shown in Th. 2 to be logically equivalent (but maybe more efficient since fixpoints are computed locally). The structural model checking $\widehat{\mathscr{M}}[\![\mathtt{P}]\!]\langle \underline{\varrho}, \mathtt{R} \rangle$ of the program $\mathtt{P}$ proceeds by structural induction on the program structure:

$$\begin{cases} \widehat{\mathscr{M}}[\![\mathtt{S}]\!]\langle \underline{\varrho}, \mathtt{R}\rangle \mathrel{\hat{=}} \widehat{\mathscr{F}}[\![\mathtt{S}]\!](\prod_{\mathtt{S}' \lhd \mathtt{S}} \widehat{\mathscr{M}}[\![\mathtt{S}']\!])\langle \underline{\varrho}, \mathtt{R}\rangle \\ \mathtt{S} \in \mathscr{P}\mathscr{C} \end{cases}$$

where the transformer $\widehat{\mathscr{F}}$ uses the results of model checking of the immediate components $\mathtt{S}' \lhd \mathtt{S}$ and involves a fixpoint computation for iteration statements.

The following Th. 2 shows that the algorithm specification is correct, that is $\widehat{\mathscr{M}}[\![\mathtt{S}]\!] = \mathscr{M}[\![\mathtt{S}]\!]$ for all program components $\mathtt{S}$. So together with Th. 1, the structural model checking is proved sound and complete.

Theorem 2. $\forall \mathtt{S} \in \mathscr{P}\mathscr{C}, \mathtt{R} \in \mathscr{R}, \underline{\varrho} \in \mathbb{E}^{\forall} . \widehat{\mathscr{M}}^{\natural}[\![\mathtt{S}]\!]\langle \underline{\varrho}, \mathtt{R}\rangle = \mathscr{M}^{\natural}[\![\mathtt{S}]\!]\langle \underline{\varrho}, \mathtt{R}\rangle$ *and* $\widehat{\mathscr{M}}[\![\mathtt{S}]\!]\langle \underline{\varrho}, \mathtt{R}\rangle = \mathscr{M}[\![\mathtt{S}]\!]\langle \underline{\varrho}, \mathtt{R}\rangle$.

The proof of Th. 2 is by calculational design and proceeds by structural induction on the program component $\mathtt{S}$. Assuming $\mathscr{M}[\![\mathtt{S}']\!] = \widehat{\mathscr{M}}[\![\mathtt{S}']\!]$ for all immediate components $\mathtt{S}' \lhd \mathtt{S}$ of statement, the proof for each program component $\mathtt{S}$ has the following form.

$\mathscr{M}[\![\mathtt{S}]\!]\langle \underline{\varrho}, \mathtt{R}\rangle$

$\mathrel{\hat{=}}$ $\mathscr{M}\langle \underline{\varrho}, \mathtt{R}\rangle(\mathscr{S}^*[\![\mathtt{S}]\!])$ ⟅(18)⟆

$=$ $\mathscr{M}\langle \underline{\varrho}, \mathtt{R}\rangle(\mathscr{F}[\![\mathtt{S}]\!](\prod_{\mathtt{S}' \lhd \mathtt{S}} \mathscr{S}^*[\![\mathtt{S}']\!])\langle \underline{\varrho}, \mathtt{R}\rangle)$

⟅by structural definition $\mathscr{S}^*[\![\mathtt{S}]\!] = \mathscr{F}[\![\mathtt{S}]\!](\prod_{\mathtt{S}' \lhd \mathtt{S}} \mathscr{S}^*[\![\mathtt{S}']\!])$ of the stateful prefix trace semantics in Section **2**⟆

$=$ ... ⟅calculus to expand definitions, rewrite and simplify formulæ by algebraic laws⟆

$=$ $\widehat{\mathscr{F}}[\![\mathtt{S}]\!](\prod_{\mathtt{S}' \lhd \mathtt{S}} \mathscr{M}[\![\mathtt{S}']\!])\langle \underline{\varrho}, \mathtt{R}\rangle$

⟅by calculational design to commute the model checking abstraction on the result to the model checking of the arguments of $\mathscr{S}^*[\![\mathtt{S}]\!]$⟆

$=$ $\widehat{\mathscr{F}}[\![\mathtt{S}]\!](\prod_{\mathtt{S}' \lhd \mathtt{S}} \widehat{\mathscr{M}}[\![\mathtt{S}']\!])\langle \underline{\varrho}, \mathtt{R}\rangle$ ⟅ind. hyp.⟆

$\mathrel{\hat{=}}$ $\widehat{\mathscr{M}}[\![\mathtt{S}]\!]\langle \underline{\varrho}, \mathtt{R}\rangle$ ⟅by defining $\widehat{\mathscr{M}}[\![\mathtt{S}]\!] \mathrel{\hat{=}} \widehat{\mathscr{F}}[\![\mathtt{S}]\!](\prod_{\mathtt{S}' \lhd \mathtt{S}} \widehat{\mathscr{M}}[\![\mathtt{S}']\!])$⟆

For iteration statements, $\mathscr{F}[\![\mathtt{S}]\!](\prod_{\mathtt{S}' \lhd \mathtt{S}} \mathscr{S}^*[\![\mathtt{S}']\!])\langle \underline{\varrho}, \mathtt{R}\rangle$ is a fixpoint, and this proof involves the fixpoint transfer theorem [16, Th. 7.1.0.4 (3)] based on the commutation of the concrete and abstract transformer with the abstraction. The calculational design of the structural model checking $\widehat{\mathscr{M}}[\![\mathtt{S}]\!]$ is shown below.

Definition 3 (Structural model checking).

– *Model checking a program* $\mathtt{P} ::= \mathtt{Sl}^{\ell}$ *for a temporal specification* $\mathtt{R} \in \mathscr{R}$ *with alternatives.*

$$\widehat{\mathscr{M}}[\![\mathtt{P}]\!]\langle \underline{\varrho},\ \mathtt{R}\rangle \triangleq \mathsf{let}\,(\mathtt{R}_1 \mid \ldots \mid \mathtt{R}_n) = \mathsf{dnf}(\mathtt{R})\ \mathsf{in} \quad (19)$$
$$\bigcup_{i=1}^{n} \{\pi \mid \exists \mathtt{R}' \in \mathbb{R}\ .\ \langle \pi,\ \mathtt{R}'\rangle \in \widehat{\mathscr{M}}^{\downarrow}[\![\mathtt{Sl}]\!]\langle \underline{\varrho},\ \mathtt{R}_i\rangle\}$$

Proof. — In case (22) of a program $\mathtt{P} ::= \mathtt{Sl}\ \ell$, the calculational design is as follows.

$$\begin{aligned}
& \mathscr{M}[\![\mathtt{P}]\!]\langle \underline{\varrho},\ \mathtt{R}\rangle \\
\triangleq\ & \mathscr{M}\langle \underline{\varrho},\ \mathtt{R}\rangle(\mathscr{S}^*[\![\mathtt{P}]\!]) && \text{⟅(18)⟆} \\
=\ & \mathsf{let}\,(\mathtt{R}_1 \mid \ldots \mid \mathtt{R}_n) = \mathsf{dnf}(\mathtt{R})\ \mathsf{in} \bigcup_{i=1}^{n} \{\pi \mid \exists \mathtt{R}' \in \mathbb{R}\ .\ \langle \pi,\ \mathtt{R}'\rangle \in \mathscr{M}^{\downarrow}\langle \underline{\varrho},\ \mathtt{R}_i\rangle(\mathscr{S}^*[\![\mathtt{P}]\!])\} \\
& && \text{⟅ (17)⟆} \\
=\ & \mathsf{let}\,(\mathtt{R}_1 \mid \ldots \mid \mathtt{R}_n) = \mathsf{dnf}(\mathtt{R})\ \mathsf{in} \bigcup_{i=1}^{n} \{\pi \mid \exists \mathtt{R}' \in \mathbb{R}\ .\ \langle \pi,\ \mathtt{R}'\rangle \in \mathscr{M}^{\downarrow}\langle \underline{\varrho},\ \mathtt{R}_i\rangle(\mathscr{S}^*[\![\mathtt{Sl}]\!])\} \\
& && \text{⟅def. of } \mathscr{S}^*[\![\mathtt{P}]\!] \triangleq \mathscr{S}^*[\![\mathtt{Sl}]\!]\text{⟆} \\
=\ & \mathsf{let}\,(\mathtt{R}_1 \mid \ldots \mid \mathtt{R}_n) = \mathsf{dnf}(\mathtt{R})\ \mathsf{in} \bigcup_{i=1}^{n} \{\pi \mid \exists \mathtt{R}' \in \mathbb{R}\ .\ \langle \pi,\ \mathtt{R}'\rangle \in \widehat{\mathscr{M}}^{\downarrow}\langle \underline{\varrho},\ \mathtt{R}_i\rangle(\mathscr{S}^*[\![\mathtt{Sl}]\!])\} \\
& && \text{⟅ind. hyp.⟆} \\
=\ & \mathsf{let}\,(\mathtt{R}_1 \mid \ldots \mid \mathtt{R}_n) = \mathsf{dnf}(\mathtt{R})\ \mathsf{in} \bigcup_{i=1}^{n} \{\pi \mid \exists \mathtt{R}' \in \mathbb{R}\ .\ \langle \pi,\ \mathtt{R}'\rangle \in \widehat{\mathscr{M}}^{\downarrow}[\![\mathtt{Sl}]\!]\langle \underline{\varrho},\ \mathtt{R}_i\rangle\} && \text{⟅(16)⟆} \\
=\ & \widehat{\mathscr{M}}[\![\mathtt{P}]\!]\langle \underline{\varrho},\ \mathtt{R}\rangle && \text{⟅(22)⟆}\ \square
\end{aligned}$$

Definition 3 (Structural model checking, contn'd)

– Model checking an empty temporal specification ε.

$$\widehat{\mathscr{M}}^{\downarrow}[\![\mathtt{S}]\!]\langle \underline{\varrho},\ \varepsilon\rangle \triangleq \{\langle \pi,\ \varepsilon\rangle \mid \pi \in \mathscr{S}^*[\![\mathtt{S}]\!]\} \quad (20)$$

– It is assumed below that $\mathtt{R} \in \mathbb{R}^{\downarrow} \cap \mathbb{R}^{+}$ is a non-empty, alternative $\mid$-free regular expression.

– Model checking a statement list $\mathtt{Sl} ::= \mathtt{Sl}'\ \mathtt{S}$

$$\begin{aligned}
\widehat{\mathscr{M}}^{\downarrow}[\![\mathtt{Sl}]\!]\langle \underline{\varrho},\ \mathtt{R}\rangle \triangleq\ & \widehat{\mathscr{M}}^{\downarrow}[\![\mathtt{Sl}']\!]\langle \underline{\varrho},\ \mathtt{R}\rangle && (21) \\
& \cup \{\langle \pi \cdot \langle \mathsf{at}[\![\mathtt{S}]\!],\ \rho\rangle \cdot \pi',\ \mathtt{R}''\rangle \mid \langle \pi \cdot \langle \mathsf{at}[\![\mathtt{S}]\!],\ \rho\rangle,\ \mathtt{R}'\rangle \in \widehat{\mathscr{M}}^{\downarrow}[\![\mathtt{Sl}']\!]\langle \underline{\varrho},\ \mathtt{R}\rangle \wedge \\
& \qquad \langle \langle \mathsf{at}[\![\mathtt{S}]\!],\ \rho\rangle \cdot \pi',\ \mathtt{R}''\rangle \in \widehat{\mathscr{M}}^{\downarrow}[\![\mathtt{S}]\!]\langle \underline{\varrho},\ \mathtt{R}'\rangle\}
\end{aligned}$$

– Model checking an empty statement list $\mathtt{Sl} ::= \epsilon$

$$\begin{aligned}
\widehat{\mathscr{M}}^{\downarrow}[\![\mathtt{Sl}]\!]\langle \underline{\varrho},\ \mathtt{R}\rangle \triangleq\ & \mathsf{let}\ \langle \mathtt{L} : \mathtt{B},\ \mathtt{R}'\rangle = \mathsf{fstnxt}(\mathtt{R})\ \mathsf{in} && (22) \\
& \{\langle \langle \mathsf{at}[\![\mathtt{Sl}]\!],\ \rho\rangle,\ \mathtt{R}'\rangle \mid \langle \underline{\varrho},\ \langle \mathsf{at}[\![\mathtt{Sl}]\!],\ \rho\rangle\rangle \in \mathscr{S}^r[\![\mathtt{L} : \mathtt{B}]\!]\}
\end{aligned}$$

(In practice the empty statement list ϵ needs not be specified so we could eliminate that need by ignoring ϵ in the specification $\mathtt{R}$ and defining $\widehat{\mathscr{M}}^{\natural}[\![\mathtt{Sl}]\!]\langle \underline{\varrho}, \mathtt{R}\rangle \triangleq \{\langle\langle \mathsf{at}[\![\mathtt{Sl}]\!], \rho\rangle, \mathtt{R}\rangle \mid \rho \in \mathbb{E}\mathtt{v}\}$.)

– Model checking an assignment statement $\mathtt{S} ::= \ell\ \mathtt{x = A\ ;}$

$$
\begin{aligned}
\widehat{\mathscr{M}}^{\natural}[\![\mathtt{S}]\!]\langle \underline{\varrho}, \mathtt{R}\rangle \triangleq\ & \mathsf{let}\ \langle \mathtt{L : B}, \mathtt{R}'\rangle = \mathsf{fstnxt}(\mathtt{R})\ \mathsf{in} && (23)\\
& \{\langle\langle \mathsf{at}[\![\mathtt{S}]\!], \rho\rangle, \mathtt{R}'\rangle \mid \langle \underline{\varrho}, \langle \mathsf{at}[\![\mathtt{S}]\!], \rho\rangle\rangle \in \mathscr{S}^{\mathtt{r}}[\![\mathtt{L : B}]\!]\} && (a)\\
& \cup \{\langle\langle \mathsf{at}[\![\mathtt{S}]\!], \rho\rangle\langle \mathsf{aft}[\![\mathtt{S}]\!], \rho[\mathtt{x} \leftarrow \mathscr{A}[\![\mathtt{A}]\!]\rho]\rangle, \varepsilon\rangle \mid \mathtt{R}' \in \mathscr{R}_{\varepsilon} \wedge && (b)\\
& \quad \langle \underline{\varrho}, \langle \mathsf{at}[\![\mathtt{S}]\!], \rho\rangle\rangle \in \mathscr{S}^{\mathtt{r}}[\![\mathtt{L : B}]\!]\} && \\
& \cup \{\langle\langle \mathsf{at}[\![\mathtt{S}]\!], \rho\rangle\langle \mathsf{aft}[\![\mathtt{S}]\!], \rho[\mathtt{x} \leftarrow \mathscr{A}[\![\mathtt{A}]\!]\rho]\rangle, \mathtt{R}''\rangle \mid \mathtt{R}' \notin \mathscr{R}_{\varepsilon} \wedge && (c)\\
& \quad \langle \underline{\varrho}, \langle \mathsf{at}[\![\mathtt{S}]\!], \rho\rangle\rangle \in \mathscr{S}^{\mathtt{r}}[\![\mathtt{L : B}]\!] \wedge \langle \mathtt{L}' \mathtt{ : B}', \mathtt{R}''\rangle = \mathsf{fstnxt}(\mathtt{R}') \wedge && \\
& \quad\ \langle \underline{\varrho}, \langle \mathsf{aft}[\![\mathtt{S}]\!], \rho[\mathtt{x} \leftarrow \mathscr{A}[\![\mathtt{A}]\!]\rho]\rangle\rangle \in \mathscr{S}^{\mathtt{r}}[\![\mathtt{L}' \mathtt{ : B}']\!]\} &&
\end{aligned}
$$

For the assignment $\mathtt{S} ::= \ell\ \mathtt{x = A\ ;}$ in (26), case (a) checks the prefixes that stops at ℓ whereas (b) and (c) checks the maximal traces stopping after the assignment. In each trace checked for the specification $\mathtt{R}$, the states are checked successively and the continuation specification is returned together with the checked trace, unless the check fails. Checking the assignment $\mathtt{S} ::= \ell\ \mathtt{x = A\ ;}$ in (26) for $\langle \mathtt{L : B}, \mathtt{R}'\rangle = \mathsf{fstnxt}(\mathtt{R})$ consists in first checking $\mathtt{L : B}$ at ℓ and then checking on $\mathtt{R}'$ after the statement. In case (b), $\mathtt{R}'$ is empty so trivially satisfied. Otherwise, in case (c), $\langle \mathtt{L}' \mathtt{ : B}', \mathtt{R}''\rangle = \mathsf{fstnxt}(\mathtt{R}')$ so $\mathtt{L}' \mathtt{ : B}'$ is checked after the statement while $\mathtt{R}''$ is the continuation specification.

Proof. — In case (26) of an assignment statement $\mathtt{S} ::= \ell\ \mathtt{x = A\ ;}$, the calculational design is as follows.

$$
\begin{aligned}
& \mathscr{M}^{\natural}[\![\mathtt{S}]\!]\ \langle \underline{\varrho}, \mathtt{R}\rangle\\
=\ & \{\langle \pi, \mathtt{R}'\rangle \mid \pi \in \mathscr{S}^{*}[\![\mathtt{Sl}]\!] \wedge \langle \mathtt{tt}, \mathtt{R}'\rangle = \mathscr{M}^{t}\langle \underline{\varrho}, \mathtt{R}\rangle\pi\} && \wr(16)\text{ and }(15)\int\\
=\ & \{\langle \pi, \mathtt{R}'\rangle \mid \pi \in \{\langle \ell, \rho\rangle \mid \rho \in \mathbb{E}\mathtt{v}\} \cup \{\langle \ell, \rho\rangle\langle \mathsf{aft}[\![\mathtt{S}]\!], \rho[\mathtt{x} \leftarrow \upsilon]\rangle \mid \rho \in \mathbb{E}\mathtt{v} \wedge \upsilon = \mathscr{A}[\![\mathtt{A}]\!]\rho \wedge \langle \mathtt{tt}, \mathtt{R}'\rangle = \mathscr{M}^{t}\langle \underline{\varrho}, \mathtt{R}\rangle\pi\} && \wr(1)\int\\
=\ & \{\langle\langle \ell, \rho\rangle, \mathtt{R}'\rangle \mid \rho \in \mathbb{E}\mathtt{v} \wedge \langle \mathtt{tt}, \mathtt{R}'\rangle = \mathscr{M}^{t}\langle \underline{\varrho}, \mathtt{R}\rangle\langle \ell, \rho\rangle\} \cup\\
& \{\langle\langle \ell, \rho\rangle\langle \mathsf{aft}[\![\mathtt{S}]\!], \rho[\mathtt{x} \leftarrow \upsilon]\rangle, \mathtt{R}'\rangle \mid \rho \in \mathbb{E}\mathtt{v} \wedge \upsilon = \mathscr{A}[\![\mathtt{A}]\!]\rho \langle \mathtt{tt}, \mathtt{R}'\rangle = \mathscr{M}^{t}\langle \underline{\varrho}, \mathtt{R}\rangle\langle \ell, \rho\rangle\langle \mathsf{aft}[\![\mathtt{S}]\!], \rho[\mathtt{x} \leftarrow \upsilon]\rangle\} && \wr\text{def. }\cup\text{ and }\in\int\\
=\ & \{\langle\langle \ell, \rho\rangle, \mathtt{R}'\rangle \mid \langle \mathtt{tt}, \mathtt{R}'\rangle = \mathsf{let}\ \langle \mathtt{L : B}, \mathtt{R}''\rangle = \mathsf{fstnxt}(\mathtt{R})\ \mathsf{in}\ (\!|\ \langle \underline{\varrho}, \langle \ell, \rho\rangle\rangle \in \mathscr{S}^{\mathtt{r}}[\![\mathtt{L : B}]\!]\ ?\ \langle \mathtt{tt}, \mathtt{R}''\rangle \wr \langle \mathtt{ff}, \mathtt{R}'\rangle\ |\!)\} \cup\\
& \{\langle\langle \ell, \rho\rangle\langle \mathsf{aft}[\![\mathtt{S}]\!], \rho[\mathtt{x} \leftarrow \upsilon]\rangle, \mathtt{R}'\rangle \mid \upsilon = \mathscr{A}[\![\mathtt{A}]\!]\rho \wedge \langle \mathtt{tt}, \mathtt{R}'\rangle = \mathsf{let}\ \langle \mathtt{L : B}, \mathtt{R}''\rangle = \mathsf{fstnxt}(\mathtt{R})\ \mathsf{in}\ (\!|\ \langle \underline{\varrho}, \langle \ell, \rho\rangle\rangle \in \mathscr{S}^{\mathtt{r}}[\![\mathtt{L : B}]\!]\ ?\ \mathscr{M}^{t}\langle \underline{\varrho}, \mathtt{R}''\rangle\langle \mathsf{aft}[\![\mathtt{S}]\!], \rho[\mathtt{x} \leftarrow \upsilon]\rangle \wr \langle \mathtt{ff}, \mathtt{R}''\rangle\ |\!)\} && \wr(14)\int\\
=\ & \{\langle\langle \ell, \rho\rangle, \mathtt{R}'\rangle \mid \langle \mathtt{L : B}, \mathtt{R}'\rangle = \mathsf{fstnxt}(\mathtt{R}) \wedge \langle \underline{\varrho}, \langle \ell, \rho\rangle\rangle \in \mathscr{S}^{\mathtt{r}}[\![\mathtt{L : B}]\!]\} \cup\\
& \{\langle\langle \ell, \rho\rangle\langle \mathsf{aft}[\![\mathtt{S}]\!], \rho[\mathtt{x} \leftarrow \upsilon]\rangle, \mathtt{R}'\rangle \mid \upsilon = \mathscr{A}[\![\mathtt{A}]\!]\rho \wedge \exists \mathtt{R}'' \in \mathscr{R}\ .\ \langle \mathtt{L : B}, \mathtt{R}''\rangle = \mathsf{fstnxt}(\mathtt{R}) \wedge \langle \underline{\varrho}, \langle \ell, \rho\rangle\rangle \in \mathscr{S}^{\mathtt{r}}[\![\mathtt{L : B}]\!] \wedge (\!|\ \mathtt{R}'' \in \mathscr{R}_{\varepsilon}\ ?\ \mathtt{tt} \wr \mathscr{M}^{t}\langle \underline{\varrho}, \mathtt{R}''\rangle\langle \mathsf{aft}[\![\mathtt{S}]\!], \rho[\mathtt{x} \leftarrow \upsilon]\rangle = \langle \mathtt{tt}, \mathtt{R}'\rangle\ |\!)\} && \\
& \qquad \wr\text{def. = and } \mathscr{M}^{t}\langle \underline{\varrho}, \varepsilon\rangle\pi \triangleq \langle \mathtt{tt}, \varepsilon\rangle \text{ by } (14)\int
\end{aligned}
$$

$$= \{\langle\langle \ell,\ \rho\rangle,\ \mathtt{R}'\rangle \mid \langle \mathtt{L}\ \mathtt{:}\ \mathtt{B},\ \mathtt{R}'\rangle = \mathsf{fstnxt}(\mathtt{R}) \wedge \langle \underline{\varrho},\ \langle \ell,\ \rho\rangle\rangle \in \mathcal{S}^{\mathrm{r}}[\![\mathtt{L}\ \mathtt{:}\ \mathtt{B}]\!]\} \cup$$
$$\{\langle\langle \ell, \rho\rangle\langle \mathsf{aft}[\![\mathtt{S}]\!], \rho[\mathtt{x} \leftarrow v]\rangle, \mathtt{R}'\rangle \mid v = \mathcal{A}[\![\mathtt{A}]\!]\rho \wedge \exists \mathtt{R}'' \in \mathcal{R}\ .\ \langle \mathtt{L}\ \mathtt{:}\ \mathtt{B}, \mathtt{R}''\rangle = \mathsf{fstnxt}(\mathtt{R}) \wedge \langle \underline{\varrho},$$
$$\langle \ell,\ \rho\rangle\rangle \in \mathcal{S}^{\mathrm{r}}[\![\mathtt{L}\ \mathtt{:}\ \mathtt{B}]\!] \wedge (\!|\ \mathtt{R}'' \in \mathcal{R}_\varepsilon\ ?\ \mathtt{tt}\ :\ \mathsf{let}\ \langle \mathtt{L}'\ \mathtt{:}\ \mathtt{B}',\ \mathtt{R}'''\rangle = \mathsf{fstnxt}(\mathtt{R}'')\ \mathsf{in}\ \langle \underline{\varrho},\ \langle \mathsf{aft}[\![\mathtt{S}]\!],$$
$$\rho[\mathtt{x} \leftarrow v]\rangle\rangle \in \mathcal{S}^{\mathrm{r}}[\![\mathtt{L}'\ \mathtt{:}\ \mathtt{B}']\!]\ |\!)\} \qquad \wr(14)\int$$

$$= \mathsf{let}\ \langle \mathtt{L}\ \mathtt{:}\ \mathtt{B},\ \mathtt{R}'\rangle = \mathsf{fstnxt}(\mathtt{R})\ \mathsf{in}$$
$$\{\langle\langle \ell,\ \rho\rangle,\ \mathtt{R}'\rangle \mid \langle \underline{\varrho},\ \langle \ell,\ \rho\rangle\rangle \in \mathcal{S}^{\mathrm{r}}[\![\mathtt{L}\ \mathtt{:}\ \mathtt{B}]\!]\}$$
$$\cup \{\langle\langle \ell,\ \rho\rangle\langle \mathsf{aft}[\![\mathtt{S}]\!],\ \rho[\mathtt{x} \leftarrow v]\rangle,\ \varepsilon\rangle \mid v = \mathcal{A}[\![\mathtt{A}]\!]\rho \wedge \langle \underline{\varrho},\ \langle \ell,\ \rho\rangle\rangle \in \mathcal{S}^{\mathrm{r}}[\![\mathtt{L}\ \mathtt{:}\ \mathtt{B}]\!] \wedge \mathtt{R}' \in \mathcal{R}_\varepsilon\}$$
$$\cup \{\langle\langle \ell,\ \rho\rangle\langle \mathsf{aft}[\![\mathtt{S}]\!],\ \rho[\mathtt{x} \leftarrow v]\rangle,\ \mathtt{R}''\rangle \mid v = \mathcal{A}[\![\mathtt{A}]\!]\rho \wedge \langle \underline{\varrho},\ \langle \ell,\ \rho\rangle\rangle \in \mathcal{S}^{\mathrm{r}}[\![\mathtt{L}\ \mathtt{:}\ \mathtt{B}]\!] \wedge \mathtt{R}' \notin$$
$$\mathcal{R}_\varepsilon \wedge \mathsf{let}\ \langle \mathtt{L}'\ \mathtt{:}\ \mathtt{B}',\ \mathtt{R}''\rangle = \mathsf{fstnxt}(\mathtt{R}')\ \mathsf{in}\ \langle \underline{\varrho},\ \langle \mathsf{aft}[\![\mathtt{S}]\!],\ \rho[\mathtt{x} \leftarrow v]\rangle\rangle \in \mathcal{S}^{\mathrm{r}}[\![\mathtt{L}'\ \mathtt{:}\ \mathtt{B}']\!]\}$$
$$\wr\mathrm{def.}\ \cup\int$$

$$= \widehat{\mathcal{M}}^{\natural}[\![\mathtt{S}]\!]\ \langle \underline{\varrho},\ \mathtt{R}\rangle \qquad \wr(26)\int\ \square$$

Definition 3 (Structural model checking, continued)

– Model checking a conditional statement $\mathtt{S} ::= \mathtt{if}\ ^{\ell}\ \mathtt{(B)}\ \mathtt{S}_t$

$$\widehat{\mathcal{M}}^{\natural}[\![\mathtt{S}]\!]\langle \underline{\varrho},\ \mathtt{R}\rangle \triangleq \mathsf{let}\ \langle \mathtt{L}'\ \mathtt{:}\ \mathtt{B}',\ \mathtt{R}'\rangle = \mathsf{fstnxt}(\mathtt{R})\ \mathsf{in} \tag{24}$$
$$\{\langle\langle \mathsf{at}[\![\mathtt{S}]\!],\ \rho\rangle,\ \mathtt{R}'\rangle \mid \langle \underline{\varrho},\ \langle \mathsf{at}[\![\mathtt{S}]\!],\ \rho\rangle\rangle \in \mathcal{S}^{\mathrm{r}}[\![\mathtt{L}'\ \mathtt{:}\ \mathtt{B}]\!]\}$$
$$\cup \{\langle\langle \mathsf{at}[\![\mathtt{S}]\!],\ \rho\rangle\langle \mathsf{at}[\![\mathtt{S}_t]\!],\ \rho\rangle\pi,\ \mathtt{R}''\rangle \mid \mathcal{B}[\![\mathtt{B}]\!]\rho = \mathtt{tt} \wedge$$
$$\langle \underline{\varrho},\ \langle \mathsf{at}[\![\mathtt{S}]\!],\ \rho\rangle\rangle \in \mathcal{S}^{\mathrm{r}}[\![\mathtt{L}'\ \mathtt{:}\ \mathtt{B}']\!] \wedge$$
$$\langle\langle \mathsf{at}[\![\mathtt{S}_t]\!],\ \rho\rangle\pi,\ \mathtt{R}''\rangle \in \widehat{\mathcal{M}}^{\natural}[\![\mathtt{S}_t]\!]\langle \underline{\varrho},\ \mathtt{R}'\rangle\}$$
$$\cup \{\langle\langle \mathsf{at}[\![\mathtt{S}]\!],\ \rho\rangle\langle \mathsf{aft}[\![\mathtt{S}]\!],\ \rho\rangle,\ \varepsilon\rangle \mid\ \mid \mathcal{B}[\![\mathtt{B}]\!]\rho = \mathtt{ff} \wedge \mathtt{R}' \in \mathcal{R}_\varepsilon \wedge$$
$$\langle \underline{\varrho},\ \langle \mathsf{at}[\![\mathtt{S}]\!],\ \rho\rangle\rangle \in \mathcal{S}^{\mathrm{r}}[\![\mathtt{L}'\ \mathtt{:}\ \mathtt{B}]\!]\}$$
$$\cup \{\langle\langle \mathsf{at}[\![\mathtt{S}]\!],\ \rho\rangle\langle \mathsf{aft}[\![\mathtt{S}]\!],\ \rho\rangle,\ \mathtt{R}''\rangle \mid \mathcal{B}[\![\mathtt{B}]\!]\rho = \mathtt{ff} \wedge \mathtt{R}' \notin \mathcal{R}_\varepsilon \wedge$$
$$\langle \underline{\varrho},\ \langle \mathsf{at}[\![\mathtt{S}]\!],\ \rho\rangle\rangle \in \mathcal{S}^{\mathrm{r}}[\![\mathtt{L}'\ \mathtt{:}\ \mathtt{B}']\!] \wedge \langle \mathtt{L}''\ \mathtt{:}\ \mathtt{B}'',\ \mathtt{R}''\rangle = \mathsf{fstnxt}(\mathtt{R}') \wedge$$
$$\langle \underline{\varrho},\ \langle \mathsf{aft}[\![\mathtt{S}]\!],\ \rho\rangle\rangle \in \mathcal{S}^{\mathrm{r}}[\![\mathtt{L}''\ \mathtt{:}\ \mathtt{B}'']\!]\}$$

– Model checking a break statement $\mathtt{S} ::= {}^{\ell}\mathtt{break}\ \mathtt{;}$

$$\widehat{\mathcal{M}}^{\natural}[\![\mathtt{S}]\!]\langle \underline{\varrho},\ \mathtt{R}\rangle \triangleq \mathsf{let}\ \langle \mathtt{L}\ \mathtt{:}\ \mathtt{B},\ \mathtt{R}'\rangle = \mathsf{fstnxt}(\mathtt{R})\ \mathsf{in} \tag{25}$$
$$\{\langle\langle \mathsf{at}[\![\mathtt{S}]\!],\ \rho\rangle,\ \mathtt{R}'\rangle \mid \langle \underline{\varrho},\ \langle \mathsf{at}[\![\mathtt{S}]\!],\ \rho\rangle\rangle \in \mathcal{S}^{\mathrm{r}}[\![\mathtt{L}\ \mathtt{:}\ \mathtt{B}]\!]\}$$
$$\cup \{\langle\langle \mathsf{at}[\![\mathtt{S}]\!],\ \rho\rangle\langle \mathsf{brk\text{-}to}[\![\mathtt{S}]\!],\ \rho\rangle,\ \varepsilon\rangle \mid \mathtt{R}' \in \mathcal{R}_\varepsilon \wedge$$
$$\langle \underline{\varrho},\ \langle \mathsf{at}[\![\mathtt{S}]\!],\ \rho\rangle\rangle \in \mathcal{S}^{\mathrm{r}}[\![\mathtt{L}\ \mathtt{:}\ \mathtt{B}]\!]\}$$
$$\cup \{\langle\langle \mathsf{at}[\![\mathtt{S}]\!],\ \rho\rangle\langle \mathsf{brk\text{-}to}[\![\mathtt{S}]\!],\ \rho\rangle,\ \mathtt{R}''\rangle \mid \mathtt{R}' \notin \mathcal{R}_\varepsilon \wedge$$
$$\langle \underline{\varrho},\ \langle \mathsf{at}[\![\mathtt{S}]\!],\ \rho\rangle\rangle \in \mathcal{S}^{\mathrm{r}}[\![\mathtt{L}\ \mathtt{:}\ \mathtt{B}]\!] \wedge \langle \mathtt{L}'\ \mathtt{:}\ \mathtt{B}',\ \mathtt{R}''\rangle = \mathsf{fstnxt}(\mathtt{R}') \wedge$$
$$\langle \underline{\varrho},\ \langle \mathsf{brk\text{-}to}[\![\mathtt{S}]\!],\ \rho\rangle\rangle \in \mathcal{S}^{\mathrm{r}}[\![\mathtt{L}'\ \mathtt{:}\ \mathtt{B}']\!]\}$$

– Model checking an iteration statement $\mathtt{S} ::= \mathtt{while}\ ^{\ell}\ \mathtt{(B)}\ \mathtt{S}_b$

$$\widehat{\mathscr{M}}^{\natural}[\![\mathtt{S}]\!]\langle \underline{\varrho},\ \mathtt{R}\rangle \triangleq \mathsf{lfp}^{\subseteq}\,(\widehat{\mathscr{F}}^{\natural}[\![\mathtt{S}]\!]\langle \underline{\varrho},\ \mathtt{R}\rangle) \tag{26}$$

$$\widehat{\mathscr{F}}^{\natural}[\![\mathtt{S}]\!]\langle \underline{\varrho},\ \mathtt{R}\rangle\ X \triangleq \mathsf{let}\ \langle \mathtt{L}' : \mathtt{B}',\ \mathtt{R}'\rangle = \mathsf{fstnxt}(\mathtt{R})\ \mathsf{in} \tag{27}$$

$$\{\langle \langle \mathsf{at}[\![\mathtt{S}]\!],\ \rho\rangle,\ \mathtt{R}'\rangle \mid \rho \in \mathbb{E}^{\forall} \wedge \langle \underline{\varrho},\ \langle \mathsf{at}[\![\mathtt{S}]\!],\ \rho\rangle\rangle \in \mathscr{S}^{\mathsf{r}}[\![\mathtt{L}' : \mathtt{B}']\!]\} \tag{a}$$

$$\cup\ \{\langle \pi_2\langle \mathsf{at}[\![\mathtt{S}]\!],\ \rho\rangle\langle \mathsf{aft}[\![\mathtt{S}]\!],\ \rho\rangle,\ \varepsilon\rangle \mid \langle \pi_2\langle \mathsf{at}[\![\mathtt{S}]\!],\ \rho\rangle,\ \varepsilon\rangle \in X \wedge \mathscr{B}[\![\mathtt{B}]\!]\ \rho = \mathtt{ff}\}$$

$$\cup\ \{\langle \pi_2\langle \mathsf{at}[\![\mathtt{S}]\!],\ \rho\rangle\langle \mathsf{aft}[\![\mathtt{S}]\!],\ \rho\rangle,\ \varepsilon\rangle \mid \langle \pi_2\langle \mathsf{at}[\![\mathtt{S}]\!],\ \rho\rangle,\ \mathtt{R}''\rangle \in X \wedge \mathscr{B}[\![\mathtt{B}]\!]\ \rho = \mathtt{ff} \wedge \mathtt{R}'' \notin \mathscr{R}_\varepsilon \wedge \langle \mathtt{L}' : \mathtt{B}',\ \mathtt{R}'\rangle = \mathsf{fstnxt}(\mathtt{R}'') \wedge \mathtt{R}' \in \mathscr{R}_\varepsilon \wedge \langle \underline{\varrho},\ \langle \mathsf{at}[\![\mathtt{S}]\!],\ \rho\rangle\rangle \in \mathscr{S}^{\mathsf{r}}[\![\mathtt{L}' : \mathtt{B}']\!]\} \tag{b}$$

$$\cup\ \{\langle \pi_2\langle \mathsf{at}[\![\mathtt{S}]\!],\ \rho\rangle\langle \mathsf{aft}[\![\mathtt{S}]\!],\ \rho\rangle,\ \mathtt{R}'\rangle \mid \langle \pi_2\langle \mathsf{at}[\![\mathtt{S}]\!],\ \rho\rangle,\ \mathtt{R}''\rangle \in X \wedge \mathscr{B}[\![\mathtt{B}]\!]\ \rho = \mathtt{ff} \wedge \mathtt{R}'' \notin \mathscr{R}_\varepsilon \wedge \langle \mathtt{L}' : \mathtt{B}',\ \mathtt{R}'''\rangle = \mathsf{fstnxt}(\mathtt{R}'') \wedge \mathtt{R}''' \notin \mathscr{R}_\varepsilon \wedge \langle \underline{\varrho},\ \langle \mathsf{at}[\![\mathtt{S}]\!],\ \rho\rangle\rangle \in \mathscr{S}^{\mathsf{r}}[\![\mathtt{L}' : \mathtt{B}']\!] \wedge \langle \mathtt{L}'' : \mathtt{B}'',\ \mathtt{R}'\rangle = \mathsf{fstnxt}(\mathtt{R}''') \wedge \langle \underline{\varrho},\ \langle \mathsf{aft}[\![\mathtt{S}]\!],\ \rho\rangle\rangle \in \mathscr{S}^{\mathsf{r}}[\![\mathtt{L}'' : \mathtt{B}'']\!]\} \tag{c}$$

$$\cup\ \{\langle \pi_2\langle \mathsf{at}[\![\mathtt{S}]\!],\ \rho\rangle\langle \mathsf{at}[\![\mathtt{S}_b]\!],\ \rho\rangle\pi_3,\ \varepsilon\rangle \mid \langle \pi_2\langle \mathsf{at}[\![\mathtt{S}]\!],\ \rho\rangle,\ \varepsilon\rangle \in X \wedge \mathscr{B}[\![\mathtt{B}]\!]\ \rho = \mathtt{tt} \wedge \langle \mathsf{at}[\![\mathtt{S}_b]\!],\ \rho\rangle\pi_3 \in \mathscr{S}^{*}[\![\mathtt{S}_b]\!]\} \tag{d}$$

$$\cup\ \{\langle \pi_2\langle \mathsf{at}[\![\mathtt{S}]\!],\ \rho\rangle\langle \mathsf{at}[\![\mathtt{S}_b]\!],\ \rho\rangle\pi_3,\ \varepsilon\rangle \mid \langle \pi_2\langle \mathsf{at}[\![\mathtt{S}]\!],\ \rho\rangle,\ \mathtt{R}''\rangle \in X \wedge \mathscr{B}[\![\mathtt{B}]\!]\ \rho = \mathtt{tt} \wedge \mathtt{R}'' \notin \mathscr{R}_\varepsilon \wedge \langle \mathtt{L} : \mathtt{B},\ \varepsilon\rangle = \mathsf{fstnxt}(\mathtt{R}'') \wedge \langle \underline{\varrho},\ \langle \mathsf{at}[\![\mathtt{S}]\!],\ \rho\rangle\rangle \in \mathscr{S}^{\mathsf{r}}[\![\mathtt{L} : \mathtt{B}]\!] \wedge \langle \mathsf{at}[\![\mathtt{S}_b]\!],\ \rho\rangle\pi_3 \in \mathscr{S}^{*}[\![\mathtt{S}_b]\!]\} \tag{e}$$

$$\cup\ \{\langle \pi_2\langle \mathsf{at}[\![\mathtt{S}]\!],\ \rho\rangle\langle \mathsf{at}[\![\mathtt{S}_b]\!],\ \rho\rangle\pi_3,\ \mathtt{R}'\rangle \mid \langle \pi_2\langle \mathsf{at}[\![\mathtt{S}]\!],\ \rho\rangle,\ \mathtt{R}''\rangle \in X \wedge \mathscr{B}[\![\mathtt{B}]\!]\ \rho = \mathtt{tt} \wedge \mathtt{R}'' \notin \mathscr{R}_\varepsilon \wedge \langle \mathtt{L} : \mathtt{B},\ \mathtt{R}''''\rangle = \mathsf{fstnxt}(\mathtt{R}'') \wedge \langle \underline{\varrho},\ \langle \mathsf{at}[\![\mathtt{S}]\!],\ \rho\rangle\rangle \in \mathscr{S}^{\mathsf{r}}[\![\mathtt{L} : \mathtt{B}]\!] \wedge \mathtt{R}'''' \notin \mathscr{R}_\varepsilon \wedge \langle \mathtt{L}' : \mathtt{B}',\ \mathtt{R}'''\rangle = \mathsf{fstnxt}(\mathtt{R}'''') \wedge \langle \underline{\varrho},\ \langle \mathsf{at}[\![\mathtt{S}_b]\!],\ \rho\rangle\rangle \in \mathscr{S}^{\mathsf{r}}[\![\mathtt{L}' : \mathtt{B}']\!] \wedge \langle \langle \mathsf{at}[\![\mathtt{S}_b]\!],\ \rho\rangle\pi_3,\ \mathtt{R}'\rangle \in \mathscr{M}^{\natural}[\![\mathtt{S}_b]\!]\langle \underline{\varrho},\ \mathtt{R}'''\rangle\} \tag{f}$$

— The model checking of an iteration statement $\mathtt{S} ::= \mathtt{while}^{\,\ell}\ \mathtt{(B)}\ \mathtt{S}_b$ in (30) checks one more iteration (after checking the previous ones as recorded by X) while the fixpoint (29) repeats this check for all iterations. Case (a) checks the prefixes that stops at loop entry ℓ. (b) and (c) check the exit of an iteration when the iteration condition is false, (b) when the specification stops at loop entry ℓ before leaving and (c) when the specification goes further. (d), (e) and (f) check one more iteration when the iteration condition is true. In case (d), the continuation after the check of the iterates is empty so trivially satisfied by any continuation of the prefix trace. In case (e), the continuation after the check of the iterates just impose to verify $\mathtt{L} : \mathtt{B}$ on iteration entry and nothing afterwards. In case (f) the continuation after the check of the iterates requires to verify $\mathtt{L} : \mathtt{B}$ at the loop entry, $\mathtt{L}' : \mathtt{B}'$ at the body entry, and the rest $\mathtt{R}'''$ of the specification for the loop body (which returns the possibly empty continuation specification $\mathtt{R}'$). The cases (b) to (f) are mutually exclusive.

9 Notes on implementations and expressivity

Of course further hypotheses and refinements would be necessary to get an effective algorithm as specified by the Def. 3 of structural model checking. A common hypothesis in model checking is that the set of states $\mathbb{S}$ is finite. Traces may still be infinite so the fixpoint computation (29) may not converge. However, infinite traces on finite states must involve an initial finite prefix followed by a finite cycle (often called a lasso). It follows that the infinite set of prefix traces can be finitely represented by a finite set of maximal finite traces and finite lassos. Regular expressions `L : B` can be attached to the states as determined by the analysis, and there are finitely many of them in the specification. These finiteness properties can be taken into account to ensure the convergence of the fixpoint computation in (29).

A symbolic representation of the states in finite/lasso stateful traces may be useful as in symbolic execution [30] or using BDDs [6] for boolean encodings of programs. By Kleene theorem [43, Theorem 2.1, p. 87], a convenient representation of regular expressions is by (deterministic) finite automata *e.g.* [34]. Symbolic automata-based algorithms can be used to implement a data structure for operations over sets of sequences [26].

Of course the hypothesis that the state space is finite and small enough to scale up and limit the combinatorial blow up of the finite state-space is unrealistic [11]. In practice, the set of states $\mathbb{S}$ is very large, so abstraction and a widening/dual narrowing are necessary. A typical trivial widening is bounded model checking (*e.g.* widen to all states after n fixpoint iterations) [5]. Those of [36] are more elaborated.

10 Conclusion

We have illustrated the idea that model checking is an abstract interpretation, as first introduced in [17]. This point of view also yields specification-preserving abstract model checking [18] as well as abstraction refinement algorithms [23].

Specifications by temporal logics are not commonly accepted by programmers. For example, in [37], the specifications had to be written by academics. Regular expressions or path expressions [8] or more expressive extensions might turn out to be more familiar. Moreover, for security monitors the false alarms of the static analysis can be checked at runtime [44,35].

Convergence of model checking requires expressivity restrictions on both the considered models of computation and the considered temporal logics. For some expressive models of computation and temporal logics, state finiteness is not enough to guarantee termination of model checking [17,24]. Finite enumeration is limited, even with symbolic encodings. Beyond finiteness, scalability is always a problem with model checking and the regular software model checking algorithm $\widetilde{\mathscr{M}}$ is no exception, so abstraction and induction are ultimately required to reason on programs.

Most often, abstract model checking uses homomorphic/partitioning abstractions *e.g.* [4]. This is because the abstraction of a transition system on concrete

states is a transition system on abstract states so model checkers are reusable in the abstract. However, excluding edgy abstractions as in [13], state-based finite abstraction is very restrictive [24] and do not guarantee scalability (*e.g.* SLAM [3]). Such restrictions on abstractions do not apply to structural model checking so that abstractions more powerful than partitioning can be considered.

As an alternative approach, a regular expression can be automatically extracted by static analysis of the program trace semantics that recognizes all feasible execution paths and usually more [19]. Then model-checking a regular specification becomes a regular language inclusion problem [33].

References

1. Abdulla, P.A., Jonsson, B., Nilsson, M., Saksena, M.: A survey of regular model checking. In: CONCUR. Lecture Notes in Computer Science, vol. 3170, pp. 35–48. Springer (2004)
2. Alur, R., Mamouras, K., Ulus, D.: Derivatives of quantitative regular expressions. In: Models, Algorithms, Logics and Tools. Lecture Notes in Computer Science, vol. 10460, pp. 75–95. Springer (2017)
3. Ball, T., Levin, V., Rajamani, S.K.: A decade of software model checking with SLAM. Commun. ACM **54**(7), 68–76 (2011)
4. Ball, T., Podelski, A., Rajamani, S.K.: Boolean and cartesian abstraction for model checking C programs. In: TACAS. Lecture Notes in Computer Science, vol. 2031, pp. 268–283. Springer (2001)
5. Biere, A., Cimatti, A., Clarke, E.M., Strichman, O., Zhu, Y.: Bounded model checking. Advances in Computers **58**, 117–148 (2003)
6. Bryant, R.E.: Graph-based algorithms for boolean function manipulation. IEEE Trans. Computers **35**(8), 677–691 (1986)
7. Brzozowski, J.A.: Derivatives of regular expressions. J. ACM **11**(4), 481–494 (1964)
8. Campbell, R.H., Habermann, A.N.: The specification of process synchronization by path expressions. In: Symposium on Operating Systems. Lecture Notes in Computer Science, vol. 16, pp. 89–102. Springer (1974)
9. Clarke, E.M., Emerson, E.A.: Design and synthesis of synchronization skeletons using branching-time temporal logic. In: Logic of Programs. Lecture Notes in Computer Science, vol. 131, pp. 52–71. Springer (1981)
10. Clarke, E.M., Henzinger, T.A., Veith, H., Bloem, R. (eds.): Handbook of Model Checking. Springer (2018)
11. Clarke, E.M., Klieber, W., Novácek, M., Zuliani, P.: Model checking and the state explosion problem. In: LASER Summer School. Lecture Notes in Computer Science, vol. 7682, pp. 1–30. Springer (2011)
12. Cousot, P.: Méthodes itératives de construction et d'approximation de points fixes d'opérateurs monotones sur un treillis, analyse sémantique de programmes (in French). Thèse d'État ès sciences mathématiques, Université de Grenoble Alpes, Grenoble, France (March 21, 1978)
13. Cousot, P.: Partial completeness of abstract fixpoint checking. In: SARA. Lecture Notes in Computer Science, vol. 1864, pp. 1–25. Springer (2000)
14. Cousot, P., Cousot, R.: Abstract interpretation: A unified lattice model for static analysis of programs by construction or approximation of fixpoints. In: POPL. pp. 238–252. ACM (1977)

15. Cousot, P., Cousot, R.: Constructive versions of Tarski's fixed point theorems. Pacific Journal of Mathematics **81**(1), 43–57 (1979)
16. Cousot, P., Cousot, R.: Systematic design of program analysis frameworks. In: POPL. pp. 269–282. ACM Press (1979)
17. Cousot, P., Cousot, R.: Temporal abstract interpretation. In: POPL. pp. 12–25. ACM (2000)
18. Crafa, S., Ranzato, F., Tapparo, F.: Saving space in a time efficient simulation algorithm. Fundam. Inform. **108**(1-2), 23–42 (2011)
19. Cyphert, J., Breck, J., Kincaid, Z., Reps, T.: Refinement of path expressions for static analysis. PACMPL **3**(POPL) (Jan 2019)
20. Giacobazzi, R., Quintarelli, E.: Incompleteness, counterexamples, and refinements in abstract model-checking. In: SAS. Lecture Notes in Computer Science, vol. 2126, pp. 356–373. Springer (2001)
21. Giacobazzi, R., Ranzato, F.: Incompleteness of states w.r.t. traces in model checking. Inf. Comput. **204**(3), 376–407 (2006)
22. Heizmann, M., Hoenicke, J., Podelski, A.: Software model checking for people who love automata. In: CAV. Lecture Notes in Computer Science, vol. 8044, pp. 36–52. Springer (2013)
23. Hopcroft, J.E., Motwani, R., Ullman, J.D.: Introduction to automata theory, languages, and computation - international edition (2. ed). Addison-Wesley (2003)
24. King, J.C.: Symbolic execution and program testing. Commun. ACM **19**(7), 385–394 (1976)
25. Kleene, S.C.: Representation of events in nerve nets and finite automata. In: Automata Studies. pp. 3—42. Princeton University Press (1951)
26. Kripke, S.A.: Semantical considerations on modal logic. Proceedings of a Colloquium on Modal and Many-Valued Logics, Helsinki, 23–26 August, 1962, Acta Philosophica Fennica **16**, 83–94 (1963)
27. Kupferman, O.: Automata theory and model checking. In: Handbook of Model Checking, pp. 107–151. Springer (2018)
28. Lichtenstein, O., Pnueli, A.: Checking that finite state concurrent programs satisfy their linear specification. In: POPL. pp. 97–107. ACM Press (1985)
29. Mallios, Y., Bauer, L., Kaynar, D.K., Ligatti, J.: Enforcing more with less: Formalizing target-aware run-time monitors. In: STM. Lecture Notes in Computer Science, vol. 7783, pp. 17–32. Springer (2012)
30. Mauborgne, L.: Binary decision graphs. In: SAS. Lecture Notes in Computer Science, vol. 1694, pp. 101–116. Springer (1999)
31. Miller, S.P., Whalen, M.W., Cofer, D.D.: Software model checking takes off. Commun. ACM **53**(2), 58–64 (2010)
32. Owens, S., Reppy, J.H., Turon, A.: Regular-expression derivatives re-examined. J. Funct. Program. **19**(2), 173–190 (2009)
33. Queille, J., Sifakis, J.: Specification and verification of concurrent systems in CESAR. In: Symposium on Programming. Lecture Notes in Computer Science, vol. 137, pp. 337–351. Springer (1982)
34. Sakarovitch, J.: Elements of Automata Theory. Cambridge University Press (2009)
35. Schneider, F.B.: Enforceable security policies. ACM Trans. Inf. Syst. Secur. **3**(1), 30–50 (2000)
36. Wolper, P.: Temporal logic can be more expressive. Information and Control **56**(1/2), 72–99 (1983)