The Systematic Design of Responsibility Analysis by Abstract Interpretation

CHAOQIANG DENG and PATRICK COUSOT, New York University

Given a behavior of interest, automatically determining the corresponding responsible entity (i.e., the root cause) is a task of critical importance in program static analysis. In this article, a novel definition of responsibility based on the abstraction of trace semantics is proposed, which takes into account the cognizance of observer, which, to the best of our knowledge, is a new innovative idea in program analysis. Compared to current dependency and causality analysis methods, the responsibility analysis is demonstrated to be more precise on various examples.

However, the concrete trace semantics used in defining responsibility is uncomputable in general, which makes the corresponding concrete responsibility analysis undecidable. To solve this problem, the article proposes a sound framework of abstract responsibility analysis, which allows a balance between cost and precision. Essentially, the abstract analysis builds a trace partitioning automaton by an iteration of over-approximating forward reachability analysis with trace partitioning and under/over-approximating backward impossible failure accessibility analysis, and determines the bounds of potentially responsible entities along paths in the automaton. Unlike the concrete responsibility analysis that identifies exactly a single action as the responsible entity along every concrete trace, the abstract analysis may lose some precision and find multiple actions potentially responsible along each automaton path. However, the soundness is preserved, and every responsible entity in the concrete is guaranteed to be also found responsible in the abstract.

CCS Concepts: • **Theory of computation → Program analysis**; **Abstraction;**

Additional Key Words and Phrases: Responsibility analysis, abstract interpretation, cognizance, forward reachability analysis, backward accessibility analysis, trace partitioning, dependency, causality

ACM Reference format:
Chaoqiang Deng and Patrick Cousot. 2021. The Systematic Design of Responsibility Analysis by Abstract Interpretation. *ACM Trans. Program. Lang. Syst.* 44, 1, Article 3 (December 2021), 90 pages.
https://doi.org/10.1145/3484938

1 INTRODUCTION

Determining the responsible entity (or, say, the root cause) of given behaviors of interest is an essential problem in the field of program analysis, especially for safety and security critical systems. For instance, given an undesired behavior (e.g., rounding error, buffer overflow) in the program, it is of significance to identify the responsible entity and configure the program accordingly, such that the undesired behavior can be prevented; similarly, determining the responsible entity for desired

The work presented in this article was supported in part by NSF Grant CNS-1446511 and NSF Grant CCF-1617717.
Authors' address: C. Deng and P. Cousot, New York University, 60 Fifth Avenue, New York City, NY, USA, 10011; emails: {deng, pcousot}@cs.nyu.edu.

0164-0925/2021/12-ART3 $15.00
https://doi.org/10.1145/3484938

behaviors (e.g., no occurrence of runtime errors, non-interference) helps programmers discovering solutions that ensure the program safety and security.

In this article, we aim at designing a generic theoretical framework of responsibility analysis, which can be instantiated in a broad range of cases to determine responsibility automatically. Contrary to accountability mechanisms [29, 39, 76] that track down perpetrators after the fact, we need to detect the responsible entity before deploying the analyzed system. Due to the massive scale of modern software, it is virtually impossible to identify responsible entities manually. Thus, the only possible solution is to use the static analysis, which can examine all possible executions of a program without executing them.

Currently, the existing static analysis techniques, such as dependency analysis [1, 12, 73], taint analysis [64], and program slicing [75], do help in narrowing down the scope of possible locations of responsible entities. However, no matter whether adopting semantic or syntactic methods, these techniques are not precise enough to explicitly identify responsibility. Meanwhile, the recent research on causality, such as the actual causality [34, 35, 63] that is based on the structural equations model (SEM) [15], succeeds in detecting cause-effect relationships in various scenarios that cannot be handled well by classic counterfactual causality [51, 52]. While the actual causality analysis is initially designed for artificial intelligence, it has been extended to reason about computational models and applied to bounded model checking [9, 47, 49]. Nevertheless, it cannot directly analyze the program syntax, and the adopted SEM unnecessarily misses some information (e.g., the temporal ordering of actions, whether an entity is free to make choices or not) that are inherent in the program semantics and indispensable in determining responsibility accurately.

To solve the above problems, we propose a novel definition of responsibility (Section 3) based on the abstraction of trace semantics, which is demonstrated to be more precise than current dependency/causality analysis. Roughly speaking, to the cognizance of an observer, an action a_{R} is responsible for the behavior $\mathcal{B}$ of interest in a given execution if and only if, according to the observer's observation, a_{R} is free to make choices, and such a choice is the first one that guarantees the occurrence of $\mathcal{B}$ in that execution. It is worth noting that our definition of responsibility in this article is a variant of the original one proposed in References [25, 26], such that the revised definition is more suitable for analyzing programs and handling system behaviors of complex structure.

In addition, another major challenge encountered is that the concrete trace semantics used in the definition of responsibility is uncomputable in general, making the corresponding concrete responsibility analysis undecidable. To solve this problem, this article seeks to compute a sound over-approximation of responsibility by abstract interpretation [18, 20, 21]. Abstract interpretation is a general theory to reason on computer programs by the approximations of program semantics, and its principle is to replace the computations on concrete semantics with computations in computer-represented abstractions that, for the sake of efficiency, only represent a selected subset of program properties and ignore others [61]. Specifically, here we propose to abstract the program trace semantics by trace partitioning automata (Section 5) that can be constructed by over-approximating forward (possible success) reachability analysis with trace partitioning [53, 70]. Furthermore, together with the under-approximating [60, 61] and over-approximating backward impossible failure accessibility analysis (Section 4), we present a sound framework of abstract responsibility analysis (Section 7), which determines the possible range of responsible entities along paths in the automaton. Although the abstract analysis loses precision to some extent, the soundness is preserved. That is to say, it is guaranteed that every action that is found responsible in the concrete must be also determined responsible in the abstract, and every action that is not found responsible in the abstract cannot be responsible in the concrete.

The application of responsibility analysis is pervasive. Although the implementation of an automatic responsibility analyzer is not provided here, we have demonstrated its effectiveness by various simple examples (Section 3.3), which includes the access control, negative balance/buffer overflow, division by zero/login attack, and information leakage.

Contribution. In this article, we present preliminary work towards the fully automatic responsibility analysis by abstract interpretation. To be more precise, the key contributions are:

- A novel definition of responsibility based on the abstract interpretation of trace semantics is introduced, where the observer's cognizance allows analyzing the responsibility from the perspective of different observers. Compared with the initial definition [25, 26] that is defined on event trace semantics, the revised definition in this article is based on the abstraction of state trace semantics, making it more suitable to analyze programs. Moreover, the definition is enhanced to handle the lattice of system behaviors of more complex structures.
- Two types of reachability/accessibility semantics are formalized: the forward possible success reachability semantics and the backward impossible failure accessibility semantics. Similar to the under-approximating backward impossible failure accessibility analysis proposed by Miné [60, 61], we take three popular numerical abstract domains (i.e., intervals [19], polyhedra [24], and octagons [56–58]) as examples, and discuss the design of an over-approximating forward possible success reachability analysis, as well as an over-approximating backward impossible failure accessibility analysis.
- The trace partitioning domain proposed by Mauborgne and Rival [53, 70] is extended with partitioning directives based on program invariants, and trace partitioning automata are introduced to intuitively represent partitioned trace semantics.
- The method of specifying behaviors of interest and the observer's cognizance in the abstract domain is presented, as well as a sound approach of checking the validity of partitioning directives with respect to the given abstract cognizance.
- A sound framework of abstract responsibility analysis is proposed, which consists of an iteration of over-approximating forward possible success reachability analysis with trace partitioning and under/over-approximating backward impossible failure accessibility analysis.

Outline. Section 2 introduces the syntax and semantics of a transition system, which is generic to handle programs written in various languages. Section 3 discusses the characteristics of responsibility, formalizes the concrete definition of responsibility as an abstraction of the program trace semantics, and exemplifies the applications of responsibility analysis. The next two sections are the basis for abstract responsibility analysis: Section 4 formalizes four types of reachability/accessibility semantics and summarizes the design of over-approximating forward possible success reachability analysis and under/over-approximating backward impossible failure accessibility analysis; Section 5 proposes to construct the trace partitioning automaton by over-approximating forward reachability analysis with trace partitioning. Section 6 describes the user specification of behaviors and cognizance in the abstract, and Section 7 illustrates the framework of abstract responsibility analysis. Section 8 reviews related work, and Section 9 concludes and discusses future work. Last, we have the proofs of all non-trivial results, most of which are relegated to Appendix A.

2 PROGRAM SYNTAX AND SEMANTICS

In this article, programs are modeled as transition systems [71, Ch.2.4], providing a language-independent small-step operational semantics that is generic to handle various programming languages (including a simple language introduced here, which is similar to a subset of C language).

$$
\begin{array}{lllll}
v & \in & \mathbb{V} & & \text{values} \\
\chi & \in & \mathbb{X} & & \text{variables} \\
\rho & \in & \mathbb{M} & \triangleq \quad \mathbb{X} \mapsto \mathbb{V} & \text{environments (memory states)} \\
\ell & \in & \mathbb{L} & & \text{program points (control states, labels)} \\
s & \in & \mathbb{S} & \triangleq \quad \mathbb{L} \times \mathbb{M} & \text{states} \\
a & \in & \mathbb{A} & & \text{actions (atomic instructions)}
\end{array}
$$

Fig. 1. Transition system domains.

2.1 Program Syntax

Transition Systems. As illustrated in Figure 1, $\mathbb{V}$ is the set of all possible values. $\mathbb{X}$ is the set of variables. $\mathbb{M}$ is the set of environments, each of which maps all the variables to their values. $\mathbb{L}$ is the set of program points; specially, $\ell^i \in \mathbb{L}$ is the initial program point, and $\ell^f \in \mathbb{L}$ is the final program point. $\mathbb{S} = \mathbb{L} \times \mathbb{M}$ is the set of states, each of which is a pair of a program point $\ell \in \mathbb{L}$ and an environment $\rho \in \mathbb{M}$. Specially, $\mathbb{S}^i \in \wp(\mathbb{S})$ denotes the set of initial states, which can be implemented as $\mathbb{S}^i = \{\ell^i\} \times \mathbb{M}$ in practice; $\mathbb{S}^f \in \wp(\mathbb{S})$ denotes the set of final states, which is implemented as $\mathbb{S}^f = \{\ell^f\} \times \mathbb{M}$ and represents correct program termination; and ω denotes the error state, which represents the incorrect program termination (e.g., division by zero). By abuse of notation, ω also denotes the error in expression evaluations and the error environment. $\mathbb{A}$ is the set of all actions (i.e., atomic instructions) in the program, e.g., assignments, Boolean tests, skip, external inputs, random number generations, variable initialization, and so on.

The *transition relation* can be defined as $\rightarrow \in \wp(\mathbb{S} \times \mathbb{A} \times \mathbb{S})$, such that $\langle s, a, s' \rangle \in \rightarrow$ (or, $s \xrightarrow{a} s'$) denotes an atomic step from one state s to another state s' after executing the action a. Alternatively, we can omit the action a, and define the transition relation as $\rightarrow \in \wp(\mathbb{S} \times \mathbb{S})$, such that an atomic step from s to s' is denoted as $s \rightarrow s'$. To be consistent with the notations in Reference [70], here we adopt the later definition of transition relation, and the omitted actions can be easily retrieved from the program source code. In addition, it is assumed that there is no outgoing transition from final states or the error state (i.e. $\forall s \in \mathbb{S}^f \cup \{\omega\}. \forall s' \in \mathbb{S}. s \not\rightarrow s'$).

A *transition system* (or, *program*) $\mathrm{P} = \langle \mathbb{S}^i, \rightarrow \rangle$ is defined as a pair of the set of initial states and the transition relation, which is generic to represent programs written in various languages.

A Simple Language. To formalize the forward reachability analysis and backward accessibility analysis for numerical programs in Section 4, we instantiate the transition system by a simple programming language in Figure 2. It is similar to the language used in Reference [60], except the ternary operation (or conditional operation) "*bexpr* ? *expr* : *expr*," which can be equivalently represented by a conditional. More precisely, $\mathbb{X}$ is a finite fixed set of real-valued variables (i.e., $\mathbb{V} = \mathbb{R}$ are real numbers and can be further restricted to integers $\mathbb{Z}$), *expr* denotes numerical expressions, and *bexpr* denotes Boolean expressions. Besides, an interval $[a; b]$ returns a random number between the left bound a and the right bound b, which facilitates directly representing non-determinism. Specially, when $a = b$, the interval $[a; b]$ denotes the constant number a. In addition, it is assumed that each atomic statement is associated with a unique program point ℓ from $\mathbb{L}$.

Every program written in this simple language can be modeled by a transition system, in which each action is either an assignment $\chi := expr$ or a Boolean test *bexpr*. In addition, as shown in Figure 3, for every (numerical or Boolean) expression e, its semantics $[\![e]\!]\rho$ is defined as the set of all possible (numerical or Boolean) values that e may take in a given environment ρ (tt denotes true, while ff denotes false); for each action a, we define an *environment transfer*

$$
\begin{array}{rcll}
\mathit{prog} & ::= & \mathit{stat} & \text{program} \\
\mathit{stat} & ::= & \mathcal{x} := \mathit{expr} & \text{assignment: } \mathcal{x} \in \mathbb{X} \\
 & | & \text{if}\,(\mathit{bexpr})\,\{\mathit{stat}\}\,\text{else}\,\{\mathit{stat}\} & \text{conditional} \\
 & | & \text{while}\,(\mathit{bexpr})\,\{\mathit{stat}\} & \text{while loop} \\
 & | & \text{assert}\,(\mathit{bexpr}) & \text{assertion} \\
 & | & \mathit{stat};\ \mathit{stat} & \text{sequential statements} \\
\mathit{expr} & ::= & [a; b] & \text{interval: } a, b \in \mathbb{R} \cup \{-\infty, \infty\} \\
 & | & \mathcal{x} & \text{variable } \mathcal{x} \in \mathbb{X} \\
 & | & -\mathit{expr} & \text{unary operation} \\
 & | & \mathit{expr} \cdot \mathit{expr} & \text{binary operation: } \cdot \in \{+, -, \times, /\} \\
 & | & \mathit{bexpr}\ ?\ \mathit{expr} : \mathit{expr} & \text{ternary operation} \\
\mathit{bexpr} & ::= & \mathit{expr} \bowtie \mathit{expr} & \text{comparison: } \bowtie \in \{<, \leq, >, \geq, =, \neq\} \\
 & | & \mathit{bexpr} \vee \mathit{bexpr} & \text{logical or operation} \\
 & | & \mathit{bexpr} \wedge \mathit{bexpr} & \text{logical and operation} \\
 & | & \neg \mathit{bexpr} & \text{logical negation operation}
\end{array}
$$

Fig. 2. The syntax of a simple language.

$$
\begin{array}{rcll}
\tau\{\!|\, a \,|\!\} & \in & \wp(\mathbb{M}) \mapsto \wp(\mathbb{M} \cup \{\omega\}) & \text{environment transfer function} \\
\tau\{\!|\, \mathcal{x} := e \,|\!\}\mathrm{M} & \triangleq & \{\rho[\mathcal{x} \mapsto v] \mid \rho \in \mathrm{M} \wedge v \in [\![e]\!]\rho\} \cup \{\omega \mid \exists \rho \in \mathrm{M}.\ \omega \in [\![e]\!]\rho\} & \\
\tau\{\!|\, b \,|\!\}\mathrm{M} & \triangleq & \{\rho \mid \rho \in \mathrm{M} \wedge \mathtt{tt} \in [\![b]\!]\rho\} \cup \{\omega \mid \exists \rho \in \mathrm{M}.\ \omega \in [\![b]\!]\rho\} & \\
[\![\mathit{expr}]\!] & \in & \mathbb{M} \mapsto \wp(\mathbb{R} \cup \{\omega\}) & \text{numerical expression semantics} \\
[\![[a; b]]\!]\rho & \triangleq & \{v \in \mathbb{R} \mid a \leq v \leq b\} & \\
[\![\mathcal{x}]\!]\rho & \triangleq & \{\rho(\mathcal{x})\} & \\
[\![-e]\!]\rho & \triangleq & \{-v \mid v \in [\![e]\!]\rho\} \cup \{\omega \mid \omega \in [\![e]\!]\rho\} & \\
[\![e_1 \cdot e_2]\!]\rho & \triangleq & \{v_1 \cdot v_2 \mid v_1 \in [\![e_1]\!]\rho \wedge v_2 \in [\![e_2]\!]\rho \wedge (v_2 \neq 0 \vee \cdot \neq /)\} \cup \{\omega \mid 0 \in [\![e_2]\!]\rho \wedge \cdot = /\} & \\
[\![b\ ?\ e_1 : e_2]\!]\rho & \triangleq & \{v \in [\![e_1]\!]\rho \mid \mathtt{tt} \in [\![b]\!]\rho\} \cup \{v \in [\![e_2]\!]\rho \mid \mathtt{ff} \in [\![b]\!]\rho\} \cup \{\omega \mid \omega \in [\![b]\!]\rho\} & \\
[\![\mathit{bexpr}]\!] & \in & \mathbb{M} \mapsto \wp(\{\mathtt{tt}, \mathtt{ff}, \omega\}) & \text{boolean expression semantics} \\
[\![e_1 \bowtie e_2]\!]\rho & \triangleq & \{\mathtt{tt} \mid \exists v_1 \in [\![e_1]\!]\rho, v_2 \in [\![e_2]\!]\rho.\ v_1 \bowtie v_2\} \cup \{\mathtt{ff} \mid \exists v_1 \in [\![e_1]\!]\rho, v_2 \in [\![e_2]\!]\rho.\ v_1 \not\bowtie v_2\} & \\
 & & \cup \{\omega \mid \omega \in [\![e_1]\!]\rho \cup [\![e_2]\!]\rho\} & \\
[\![b_1 \vee b_2]\!]\rho & \triangleq & \{\mathtt{tt} \mid \mathtt{tt} \in [\![b_1]\!]\rho \cup [\![b_2]\!]\rho\} \cup \{\mathtt{ff} \mid \mathtt{ff} \in [\![b_1]\!]\rho \cap [\![b_2]\!]\rho\} \cup \{\omega \mid \omega \in [\![b_1]\!]\rho \cup [\![b_2]\!]\rho\} & \\
[\![b_1 \wedge b_2]\!]\rho & \triangleq & \{\mathtt{tt} \mid \mathtt{tt} \in [\![b_1]\!]\rho \cap [\![b_2]\!]\rho\} \cup \{\mathtt{ff} \mid \mathtt{ff} \in [\![b_1]\!]\rho \cup [\![b_2]\!]\rho\} \cup \{\omega \mid \omega \in [\![b_1]\!]\rho \cup [\![b_2]\!]\rho\} & \\
[\![\neg b]\!]\rho & \triangleq & \{\mathtt{tt} \mid \mathtt{ff} \in [\![b]\!]\rho\} \cup \{\mathtt{ff} \mid \mathtt{tt} \in [\![b]\!]\rho\} \cup \{\omega \mid \omega \in [\![b]\!]\rho\} &
\end{array}
$$

Fig. 3. The environment transfer functions and expression semantics.

function $\tau\{\!|\, a \,|\!\} \in \wp(\mathbb{M}) \xrightarrow{\nearrow} \wp(\mathbb{M} \cup \{\omega\})$, which maps a set of environments before a to the set of reachable environments after it, including the error state ω if the execution of a encounters an error. From these environment transfer functions, the corresponding transition relation $\rightarrow$ can be easily derived: For any atomic action of the form $^{\ell}a^{\ell'}$, the transition relation is $\{\langle \ell,\ \rho\rangle \rightarrow \langle \ell',\ \rho'\rangle \mid \rho, \rho' \in \mathbb{M} \wedge \rho' \in \tau\{\!|\, a \,|\!\}(\{\rho\})\} \cup \{\langle \ell,\ \rho\rangle \rightarrow \omega \mid \rho \in \mathbb{M} \wedge \omega \in \tau\{\!|\, a \,|\!\}(\{\rho\})\}$; for any program P, its transition relation is the union of transition relations defined for all its atomic actions.

2.2 Program Semantics

Traces. For any program (i.e., transition system), an execution is represented by a finite or infinite sequence of states, which is called as a *trace*; the *program semantics* is a set of such executions, and a *trace property* is a set of traces that have this property.

In the following, we write $\sigma = s_0 \cdots s_{n-1}$ to denote a finite trace of exactly length n, where the $(i+1)^{th}$ state $s_i = \langle \ell_i, \rho_i \rangle \in \mathbb{S}$ along the trace is denoted as $\sigma_{[i]}$; equivalently, such a finite trace of length n can be represented as a mapping from natural numbers in $[0, n-1]$ to states. Similarly, we write $\sigma = s_0 \cdots s_i \cdots$ to denote an infinite trace that does not terminate, and such a trace can be equivalently represented as a mapping from all natural numbers to states. Specially, ε denotes the empty trace. In addition, by abuse of notation, we say a state s belongs to a trace σ (i.e., $s \in \sigma$), if there exists a natural number $i \in \mathbb{N}$ such that $\sigma_{[i]} = s$.

For any trace σ, its length $|\sigma|$ is the number of states in σ. Specially, the length of the empty trace $|\varepsilon|$ is 0; for an infinite trace σ, its length $|\sigma|$ is denoted as ∞.

$$\sigma \in \mathbb{S}^+ \triangleq \bigcup_{n \geqslant 1} ([0, n-1] \mapsto \mathbb{S}) \qquad \text{finite traces}$$

$$\sigma \in \mathbb{S}^* \triangleq \{\varepsilon\} \cup \mathbb{S}^+ \qquad \text{empty or finite traces}$$

$$\sigma \in \mathbb{S}^\infty \triangleq \mathbb{N} \mapsto \mathbb{S} \qquad \text{infinite traces}$$

$$\sigma \in \mathbb{S}^{+\infty} \triangleq \mathbb{S}^+ \cup \mathbb{S}^\infty \qquad \text{finite or infinite traces}$$

$$\sigma \in \mathbb{S}^{*\infty} \triangleq \{\varepsilon\} \cup \mathbb{S}^{+\infty} \qquad \text{empty or finite or infinite traces}$$

The *concatenation* of a finite trace $\sigma = s_0 \cdots s_{n-1}$ and a state s is defined by juxtaposition σs such that $\sigma s = s_0 \cdots s_{n-1} s$; the *concatenation* of a finite trace $\sigma = s_0 \cdots s_{n-1}$ and a transition $\tau = s_{n-1} \rightarrow s_n$ is denoted as $\sigma\tau$ such that $\sigma\tau = s_0 \cdots s_{n-1} s_n$; the *concatenation* of a finite traces $\sigma = s_0 \cdots s_{n-1}$ and another (finite or infinite) trace $\sigma' = s'_0 \cdots$ is denoted as $\sigma\sigma'$ such that $\sigma\sigma' = s_0 \cdots s_{n-1} s'_0 \cdots$; the *concatenation* of an infinite trace σ and another trace (or a state, a transition) is the same as σ itself.

A trace σ is said to be $\preceq$ - less than or equal to another trace σ' if and only if σ is a prefix of σ'. Besides, for any set $\mathcal{T}$ of traces, we define $\mathrm{Pref}(\mathcal{T})$ as the set of prefixes of traces in $\mathcal{T}$.

$$\sigma \preceq \sigma' \triangleq |\sigma| \leqslant |\sigma'| \wedge \forall\, 0 \leqslant i < |\sigma| : \sigma_{[i]} = {\sigma'}_{[i]} \qquad \text{ordering of traces}$$

$$\mathrm{Pref} \in \wp(\mathbb{S}^{*\infty}) \mapsto \wp(\mathbb{S}^{*\infty}) \qquad \text{prefixes of traces}$$

$$\mathrm{Pref}(\mathcal{T}) \triangleq \{\sigma' \in \mathbb{S}^{*\infty} \mid \exists \sigma \in \mathcal{T}.\ \sigma' \preceq \sigma\}$$

Trace semantics. For a program $\mathrm{P} = \langle \mathbb{S}^i, \rightarrow \rangle$, a valid *intermediate (partial) trace* σ is a finite or infinite trace, along which every two successive states are bounded by the transition relation $\rightarrow$. The *intermediate (partial) trace semantics* $[\![\mathrm{P}]\!]^{\mathrm{It}} \in \wp(\mathbb{S}^{*\infty})$ is the set of all valid intermediate traces for P, i.e., $[\![\mathrm{P}]\!]^{\mathrm{It}} \triangleq \{s_0 \cdots s_{n-1} \in \mathbb{S}^* \mid \forall i \in [0, n-2].\ s_i \rightarrow s_{i+1}\} \cup \{s_0 \cdots s_i \cdots \in \mathbb{S}^\infty \mid \forall i \in \mathbb{N}.\ s_i \rightarrow s_{i+1}\}$. This semantics is a formal description of the executions of P, which start from any state and stop at any time or do not ever stop.

A valid *prefix trace* σ is a finite or infinite trace, such that it starts from an initial state $s \in \mathbb{S}^i$ and every two successive states along the trace are related by the transition relation $\rightarrow$. The *prefix trace semantics* $[\![\mathrm{P}]\!]^{\mathrm{Pref}} \in \wp(\mathbb{S}^{*\infty})$ of P is the set of valid prefix traces, i.e., $[\![\mathrm{P}]\!]^{\mathrm{Pref}} \triangleq \{\sigma \in [\![\mathrm{P}]\!]^{\mathrm{It}} \mid \sigma_{[0]} \in \mathbb{S}^i\} = \{s_0 \cdots s_{n-1} \in \mathbb{S}^* \mid s_0 \in \mathbb{S}^i \wedge \forall i \in [0, n-2].\ s_i \rightarrow s_{i+1}\} \cup \{s_0 \cdots s_i \cdots \in \mathbb{S}^\infty \mid s_0 \in \mathbb{S}^i \wedge \forall i \in \mathbb{N}.\ s_i \rightarrow s_{i+1}\}$. This semantics is a formal description of the executions of P, which start from any initial state and stop at any time or do not ever stop.

A valid *maximal trace* σ is a finite or infinite trace, such that it starts from an initial state $s \in \mathbb{S}^i$, every two successive states are related by the transition relation $\rightarrow$, and either it terminates at a final state $s' \in \mathbb{S}^f$ or the error state ω, or it does not ever terminate. The *maximal trace semantics*

```
ℓ1 :  apv := 1;                                   //Approval: positive - yes, zero or negative - no
ℓ2 :  i1 := [-1; 2];                              //Input from 1st admin
ℓ3 :  apv := (i1 ≤ 0) ? -1 : apv;
ℓ4 :  i2 := [-1; 2];                              //Input from 2nd admin
ℓ5 :  apv := (i2 ≤ 0) ? -1 : apv;
ℓ6 :  typ := [1; 2];                              //Input from system settings
ℓ7 :  acs := apv × typ;
ℓ8 :                                              //Access the object o here, and it fails when acs ≤ 0
```

Fig. 4. Access control program example.

$[\![\mathrm{P}]\!]^{\mathsf{Max}} \in \wp(\mathbb{S}^{*\infty})$ of P is the set of valid maximal traces, i.e., $[\![\mathrm{P}]\!]^{\mathsf{Max}} \triangleq \{s_0 \cdots s_{n-1} \in \mathbb{S}^* \mid s_0 \in \mathbb{S}^i \wedge \forall i \in [0, n-2].\, s_i \rightarrow s_{i+1} \wedge s_{n-1} \in \mathbb{S}^f \cup \{\omega\}\} \cup \{s_0 \cdots s_i \cdots \in \mathbb{S}^\infty \mid s_0 \in \mathbb{S}^i \wedge \forall i \in \mathbb{N}.\, s_i \rightarrow s_{i+1}\}$. This semantics is a formal description of the executions of P, which start from any initial state and stop only at final states or crash or last forever. It is not hard to see that $[\![\mathrm{P}]\!]^{\mathsf{Max}} \subseteq [\![\mathrm{P}]\!]^{\mathsf{Pref}} \subseteq [\![\mathrm{P}]\!]^{\mathsf{It}}$. In addition, the prefix trace semantics $[\![\mathrm{P}]\!]^{\mathsf{Pref}}$ is an abstraction of the maximal trace semantics $[\![\mathrm{P}]\!]^{\mathsf{Max}}$ via the function Pref, i.e., $[\![\mathrm{P}]\!]^{\mathsf{Pref}} = \mathsf{Pref}([\![\mathrm{P}]\!]^{\mathsf{Max}})$.

Example 1 (Access Control). The program in Figure 4 can be interpreted as an access control program for an object *o* (e.g., a confidential file) such that *o* can be accessed if and only if both two admins approve the access and the permission type of *o* from system settings is greater than or equal to "read only." For the sake of clarity, it is assumed that in this example the evaluation of an interval returns only integers (i.e., $\mathbb{V} = \mathbb{Z}$), and the analysis is similar to analyzing real numbers. Specifically, in lines 2 and 4, the variable *i1* and *i2* assigned by a non-deterministic integer from $[-1; 2]$ are used to mimic external inputs that correspond to the decisions of two independent admins, where a positive value (i.e., 1 or 2) represents approving the access to *o*, while 0 or a negative value (i.e., -1) represents rejecting the access; in line 6, the variable *typ* assigned by $[1; 2]$ mimics the action of reading the permission type of *o* specified in the system settings (e.g., we can assume that 1 represents "read only," and 2 represents "read and write," which is similar to the file permissions system in Unix); in line 8, the access to *o* succeeds only when the value of *acs* is strictly positive (i.e., 1 or 2), which guarantees that both admins approve the access and the permission type of *o* is at least as high as "read only."

Here, we are interested in: When the access to o fails in the execution (referred as "Access Failure," i.e., $acs \leq 0$ at point ℓ_8), which action (actions) shall be responsible?

Throughout this article, we use this example to illustrate responsibility analysis. To begin with, we represent the above program as a transition system: the set of program points $\mathbb{L} = \{\ell_1, \ell_2, \ell_3, \ell_4, \ell_5, \ell_6, \ell_7, \ell_8\}$; the set of variables $\mathbb{X} = \{apv, i1, i2, typ, acs\}$; the set of environments $\mathbb{M} = \mathbb{X} \mapsto \mathbb{Z}$, where $\mathbb{Z}$ is the set of integers; the set of states $\mathbb{S} = \mathbb{L} \times \mathbb{M}$, the set of initial states $\mathbb{S}^i = \{\ell_1\} \times \mathbb{M}$, and the set of final states $\mathbb{S}^f = \{\ell_8\} \times \mathbb{M}$. Moreover, the transition relation $\rightarrow$ can be easily derived from the environment transfer functions for the atomic actions (e.g., $\tau\llparenthesis i1 := [-1; 2]\rrparenthesis$, $\tau\llparenthesis apv := (i1 \leq 0)\ ?\ -1 : apv\rrparenthesis$).

Next, consider its trace semantics. It is obvious that there is no valid infinite trace and there is no possibility to reach the error state ω, thus a valid maximal trace must start from the initial point ℓ_1 and terminate at the final point ℓ_8. More precisely, its maximal trace semantics $[\![\mathrm{P}]\!]^{\mathsf{Max}} = \{\langle \ell_1, \rho_1\rangle\langle \ell_2, \rho_2\rangle\langle \ell_3, \rho_3\rangle\langle \ell_4, \rho_4\rangle\langle \ell_5, \rho_5\rangle\langle \ell_6, \rho_6\rangle\langle \ell_7, \rho_7\rangle\langle \ell_8, \rho_8\rangle \mid (\rho_1 \in \mathbb{M}) \wedge (\rho_2 = \rho_1[apv \mapsto 1]) \wedge (\rho_3 = \rho_2[i1 \mapsto v_1] \wedge v_1 \in \{-1, 0, 1, 2\}) \wedge (\rho_4 = \rho_3[apv \mapsto ((\rho_3(i1) <= 0)? - 1 : \rho_3(apv))]) \wedge (\rho_5 = \rho_4[i2 \mapsto v_2] \wedge v_2 \in \{-1, 0, 1, 2\}) \wedge (\rho_6 = \rho_5[apv \mapsto ((\rho_5(apv) >= 1 \wedge \rho_5(i2) <= 0)? - 1 : \rho_5(apv))]) \wedge (\rho_7 = \rho_6[typ \mapsto v_3] \wedge v_3 \in \{1, 2\}) \wedge (\rho_8 = \rho_7[acs \mapsto \rho_7(apv) \times \rho_7(typ)])\}$. In addition, the trace property

"Access Failure" can be represented by a set of maximal traces in which the value of *acs* is less than or equal to 0 at point ℓ_8, i.e., $\{\sigma \in [\![\mathrm{P}]\!]^{\mathrm{Max}} \mid \exists \rho \in \mathbb{M}.\ \sigma_{[7]} = \langle \ell_8,\ \rho \rangle \wedge \rho(acs) <= 0\}$.

3 THE DEFINITION OF RESPONSIBILITY

The objective of this section is to give a formal definition of responsibility in the concrete. To start with, we introduce a simple but thought-provoking example of forest fire, which characterizes the difference of responsibility with dependency and causality. Inspired by this forest fire example, we suggest an informal definition of responsibility, which is quite intuitive and applicable to analyzing various behaviors. Furthermore, we design a framework of concrete responsibility analysis, in which the responsibility is formally defined as an abstraction of the trace semantics. The application of responsibility analysis is demonstrated to be pervasive by examples, including negative balance/buffer overflow, division by zero/login attack, and information leakage.

3.1 The Characteristics of Responsibility

Here, we start with a simple intuitive example of forest fire used in defining actual cause [34, 35] and further characterize three indispensable elements in defining responsibility.

Example 2 (Forest Fire). Suppose that two arsonists drop lit matches in different parts of a dry forest on a windy day, and both cause trees to start burning. Here it is assumed that one arsonist named A drops the lit match before the other arsonist B, and there are two scenarios. In the first scenario, called the disjunctive scenario, either match by itself suffices to burn down the whole forest. That is to say, even if only one match were lit, the forest would burn down. In the second scenario, called the conjunctive scenario, two lit matches are necessary to burn down the whole forest; if only one match were lit, some trees would be burnt, but the fire dies down before the whole forest is destroyed.

It is quite natural to bring up this question: Who shall be responsible for burning down the whole forest? The literature has several possible answers. First, a simple but popular solution is the *dependency analysis* [1, 12, 17, 48, 73] that determines how entities (e.g., values of variables) depend upon other entities. Suppose we use variables to represent all the entities in the forest fire example (e.g., the decisions of two arsonists, the status of matches, the fire condition of the forest, the weather), then the real-life example of forest fire can be viewed as an equivalent computer program. By the definition of dependency, in both scenarios the forest fire depends on those two arsonists, as well as many other non-decisive factors, such as the wind, which influences the spreading speed of forest fire. Such a result is correct, but far from precise: The wind could not either enforce or prevent the fire, and it is against the intuition to take such non-decisive factors as responsible entities.

Next, the classic *counterfactual causality* [51, 52] determines causality according to a criterion: An event e is a cause of the occurrence of another event e' if and only if were e not to occur, e' would not happen. Such a criterion excludes non-decisive factors (e.g., the wind) from causes. For instance, if there was no wind, then the forest would still be burnt down, hence the wind is not a cause of the forest fire. However, the counterfactual causality may be too strict in some circumstances. Take the disjunctive scenario of forest fire as an example, if A (respectively, B) did not drop a lit match, then the forest would still be burnt down due to the other arsonist, hence the forest fire does not counterfactually depend on any single event, and no entity is determined as the cause of fire.

Last, consider the *actual cause* [34, 35, 63] that is based on the structural equations model (SEM) [15] and allows "contingent counterfactual dependency." More precisely, events are represented by variable values in the SEM, and an event e is an actual cause of another event e' if there exists a

contingency (where the values for other variables may be changed) such that e' counterfactually depends on e. Taking the disjunctive scenario of forest fire as an example, the arsonist A is determined as an actual cause of the forest fire, since the forest fire counterfactually depends on A's action of dropping a lit match under the contingency where the other arsonist B does not drop a lit match; in a similar way, the arsonist B is also determined as an actual cause of the forest fire. Such a structural model method has allowed for a great progress in causality analysis, solving many problems of previous approaches. In addition, it has been extended to reason about computational models and applied to bounded model checking [9, 47, 49]. However, as an abstraction of the concrete semantics, the SEM unnecessarily misses the following three essential points in determining responsibility:

(P1) *The temporal ordering of events/actions should be taken into account.* For instance, in the forest fire example, the SEM cannot tell the difference whether A or B drops a lit match first, hence determines both arsonists as the cause of forest fire. Such a result may not seem to be absurd, but imagine the case that the forest has already been burnt down by A before B lit her/his match in the disjunctive scenario. In such a case, it is against intuition to put B as a cause of the forest fire. To deal with this problem, Reference [11] suggests to modify the SEM and introduce some new variable to distinguish whether the forest was actually destroyed by A or B, which is difficult to accomplish in practice for programs. In contrast, a much simpler method is to keep the temporal ordering of events/actions, such that only the first action that guarantees the behavior of interest is counted as the responsible entity. For instance, in the disjunctive scenario of forest fire, the action of dropping a lit match by A ensures burning down the whole forest even before B makes her/his choice, thus only A is responsible for burning down the forest; in the conjunctive scenario of forest fire, the lit match dropped by A starts a forest fire that would die down unless B drops a second lit match, thus A is only responsible for starting a forest fire, while B is responsible for burning down the whole forest.

(P2) *The responsible entity must be free to make choices.* In the forest fire example, suppose that the match is taken as a separate entity, and its value (i.e., lit or not-lit) solely depends on the corresponding arsonist. In the SEM of actual cause, both the arsonists and the matches are represented by endogenous variables and further determined as actual causes of burning down the forest. However, it is common sense that the match itself does not have a choice to light or not, and it is inappropriate to identify matches as responsible entities for the forest fire. Hence, only the action that can make choices at its own discretion can possibly be responsible for a behavior. Specifically, in computer programs, such actions include but are not limited to user inputs, system settings, parameters of procedures or modules, variable initialization, random number generations, and the parallelism. To be more accurate, it is the external subject (who does the input, configures the system settings, etc.) that is free to make choices, but we say that actions like user inputs are free to make choices, as an abuse of language.

(P3) *It is necessary to explicitly specify "to whose cognizance" when analyzing the responsibility.* All the above reasoning on causality is implicitly based on the cognizance/knowledge of an omniscient observer who knows everything that occurred, yet it is non-trivial to consider the cognizance of a non-omniscient observer. For instance, we can adopt the cognizance of the second arsonist B in the forest fire example. In the disjunctive scenario, if B is aware that A has already dropped a lit match in the forest, then B is not responsible to his/her cognizance, since B knows that the forest is guaranteed to burn down no matter whether she/he drops a lit match or not; otherwise, if B does not know a lit match has been dropped, then B is responsible for the forest fire to her/his cognizance, although she/he is not responsible to the cognizance of an omniscient observer. Similarly, in the conjunctive scenario, if B is aware

that A has already dropped a lit match in the forest, then B understands that it is her/his own action that ensures burning down the whole forest, hence she/he shall take the responsibility to her/his cognizance; otherwise, if B does not know that a lit match has been dropped, then she/he does not expect the whole forest to be burnt down, hence to her/his own cognizance B is only responsible for starting a forest fire, but not burning down the whole forest. In most cases, the cognizance of an omniscient observer will be adopted, but not always.

An Informal Definition of Responsibility. To take the above three points into account, this article proposes responsibility whose informal definition is as follows.

Definition 1 (Responsibility, Informally). To the cognizance of an observer, an action a_{R} is responsible for the behavior $\mathcal{B}$ of interest in a given execution if and only if, according to the observer's observation, a_{R} is free to make choices, and such a choice is the first one that guarantees the occurrence of $\mathcal{B}$ in that execution.

It is necessary to point out that, for the whole system whose concrete semantics is a set of executions, there may exist more than one action that is responsible for $\mathcal{B}$. Nevertheless, in every single execution where $\mathcal{B}$ occurs, there is only one action that is responsible for $\mathcal{B}$. To decide which action in an execution is responsible, the execution alone is not sufficient, and it is required to reason on the whole semantics to exhibit the action's "free choices" and guarantee of $\mathcal{B}$. Thus, responsibility is not a trace property (neither safety nor liveness property), but a hyper-property [16], which is a property of sets of execution traces.

In the following, we consider the access control program example in Figure 4 again and discuss the advantage of the responsibility analysis over the classic dependency/causality analysis in an informal way, while the responsibility analysis procedure of this access control program will be formalized in Section 3.2.

Example 3 (Access Control, Continued). For the access control program in Figure 4, the question that we are interested in is: When the access to o fails in the program execution (referred as "Access Failure," i.e., $acs \leq 0$ at point ℓ_8), which action (actions) shall be responsible?

First, we consider the dependency analysis and corresponding slicing techniques. No matter whether we adopt the syntactic dependency or semantic dependency, it is not hard to see that the value of acs at point ℓ_8 depends on the value of apv and typ at point ℓ_7, which further depend on the inputs from the two admins and system settings. That is to say, the behavior "access failure" depends on all variables in the program, thus program slicing techniques (both syntactic slicing [75] and semantic slicing [69]) would take the whole program as the slice related with access failure. Although the slicing technique intends to rule out parts of the program that are completely irrelevant with the behavior of interest, it is too imprecise to be practically useful in this example. For instance, the computed slice includes the actions such as $apv := 1$, $apv := (i1 \leq 0)\ ?\ -1 : apv$ and $acs := apv \times typ$, which have no free choices: They are completely deterministic and act merely as the intermediary between causes and effects, thus shall not be treated as responsible entities. Moreover, similar to the wind in the forest fire example, the action $typ := [1; 2]$ representing the input from system settings is a non-decisive factor for the access failure behavior (i.e., no matter whether typ is 1 or 2, it cannot either enforce or prevent the access failure), although it does affect the value of acs at point ℓ_8. Therefore, the dependency analysis and slicing are not precise enough to identify responsible entities.

Second, using the counterfactual causality proposed by Lewis [51, 52], we can exclude non-decisive factors (i.e., the action $typ := [1; 2]$ in this example), but it fails to find any cause of the access failure behavior in the executions where the inputs from both admins are negative or zero. For example, in the execution where $i1 = 0$ and $i2 = 0$, neither inputs would be determined as the

cause, because the behavior of access failure does not counterfactually depend on either of them. More precisely, if the input from one admin (either *i1* or *i2*) is changed to a strictly positive value (i.e., 1 or 2), then the access failure would still occur due to the input 0 from the other admin.

Third, we consider the definition of actual cause proposed by Halpern and Pearl [34, 35, 63], and represent the access control program by a SEM: three non-deterministic inputs from admins and system settings (i.e., $[-1;2]$ and $[1;2]$) are represented by exogenous variables, and each program variable is presented by an endogenous variable, whose value is deterministically decided by the values of other (exogenous or endogenous) variables. Similar to the counterfactual causality by Lewis, non-decisive factors (i.e., the assignment to *typ*) would not be counted as actual causes. Yet, the actual cause allows reasoning counterfactual dependency under a contingency, such that it can identify causes in the executions where the inputs from both two admins are negative or zero. For example, in the execution where the inputs from two admins are 0, both $i1 = 0$ and $i2 = 0$ are determined as actual causes of access failure, because the access failure counterfactually depends on $i1 = 0$ (respectively, $i2 = 0$) under the contingency where the value of *i2* (respectively, *i1*) is changed to 1 or 2. Besides, similar to the dependency analysis, the intermediate events between causes and effects (e.g., $apv = -1$ and $acs = -1$) are also determined as actual causes of access failure.

Last, compared with the above dependency/causality analysis, the responsibility analysis according to the Definition 1 would be much more precise, and it can accurately identify the responsible entities of access failure in various cases. Here, we list the entire desired responsibility analysis results, while the detailed procedure of producing such results is formalized in the next section. (1) To the cognizance of an omniscient observer: For any execution, if the input from the 1st admin is negative or zero, then no matter what the other two inputs are, only the action $i1 := [-1;2]$ (which represents the input from the 1st admin) is responsible for the access failure behavior, because it guarantees the access failure even before the 2nd admin inputs her/his decision; if the input from the 1st admin is positive and the input from the 2nd admin is negative or zero, then only the action $i2 := [-1;2]$ (which represents the input from the 2nd admin) is responsible for the access failure behavior, because the positive input from the 1st admin does not either enforce or prevent the access failure, while the negative or zero input from the 2nd admin is the first action that guarantees the access failure; otherwise, if the inputs from both admins are positive, then the access failure behavior does not occur, thus there is no responsible entity. (2) To the cognizance of a non-omniscient observer who does not know the input from the 1st admin: For any execution, if the input from the 2nd admin is negative or zero, then no matter what the input from the 1st admin is, only the action $i2 := [-1;2]$ (i.e., the input from the 2nd admin) is responsible for the access failure behavior, because from the knowledge of the non-omniscient observer, the access failure behavior is ensured only after the 2nd admin inputs a negative value or zero; otherwise, if the input from the 2nd admin is positive, then whether the access failure occurs or not is uncertain from the perspective of non-omniscient observer, thus no entity is responsible for the access failure.

After finishing the responsibility analysis, it is time for the user to configure permissions granted to each responsible entity at her/his discretion. In this example, suppose the cognizance of an omniscient observer is adopted, then we find that only the inputs from two admins are possibly responsible for the access failure behavior. If the two admins are authorized to control the access, then their permissions to input negative values or zero can be kept; otherwise, if those two admins have no authorization to decline the access to o, then their permissions to input negative values or zero shall be removed.

3.2 The Framework of Concrete Responsibility Analysis

To put the informal definition of responsibility (Definition 1) into effect, we design a framework of concrete responsibility analysis as illustrated in Figure 5, which essentially consists of three

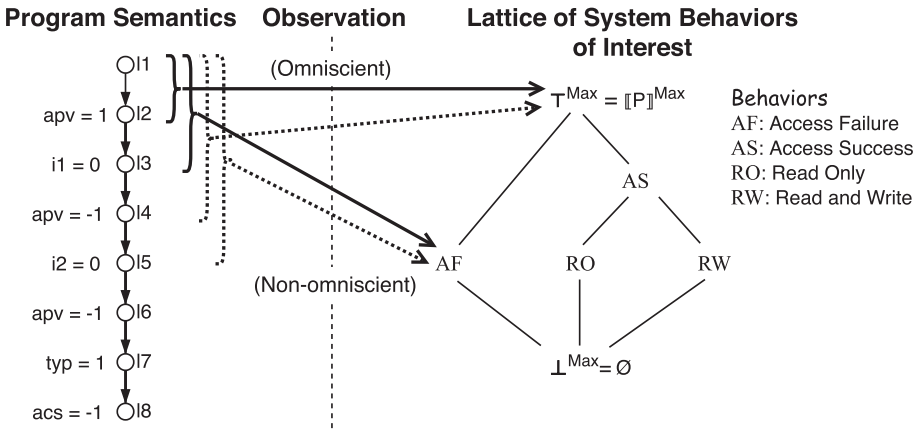

Fig. 5. Framework of concrete responsibility analysis for access control example.

components: (1) *Program semantics*, i.e., the set of all possible executions, each of which can be analyzed individually. (2) *A lattice of system behaviors of interest*, which is ordered such that the stronger/stricter a behavior is, the lower is its position in the lattice. (3) *An observation function for each observer*, which maps every (probably unfinished) execution to a behavior in the lattice that is guaranteed to occur, even though such a behavior may not have occurred yet.

In this framework of concrete analysis, if an observer's observation finds that the guaranteed behavior grows stronger after extending an execution by an action, then the extended part of execution (i.e., the action) must be responsible for ensuring the occurrence of the stronger behavior. Consider the example in Figure 5 that sketches the analysis for a certain execution of the access control program, where the inputs from both admins are zeros while the input from system settings is one. Suppose in the lattice of system behaviors, the top $\top^{\text{Max}}$ represents the behavior "not sure if the access to o fails or not," AF represents the behavior of access failure, and AS represents the behavior of access success, whose formal definitions are given in Section 3.2.2. The solid arrow from executions to the lattice stands for the observation of an omniscient observer who knows everything, while the dashed arrow stands for the observation of a non-omniscient observer who is unaware of the input from 1st admin.

As illustrated in Figure 5, the omniscient observer finds that the execution from point ℓ_1 to point ℓ_2 can guarantee only $\top^{\text{Max}}$ (i.e., before the 1st admin inputs her/his decision, whether the access to o fails or succeeds is undecided), while the stronger behavior AF is guaranteed when the execution reaches point ℓ_3 (i.e., after the 1st admin inputs zero, it is ensured that the access to o will fail, even though it has not occurred yet). Thus, to the cognizance of the omniscient observer, the action between point ℓ_2 and ℓ_3 (i.e., $i1 := [-1; 2]$ representing the input from 1st admin) is responsible for the access failure behavior. In contrast, the non-omniscient observer finds that all the executions upto point ℓ_4 guarantee $\top^{\text{Max}}$ (i.e., she/he does not know the 1st admin already inputs 0, thus believes that the access failure behavior is not guaranteed yet), and AF is guaranteed only after point ℓ_5 is reached (i.e., only after the 2nd admin inputs zero, the non-omniscient observer knows that the access failure is ensured to occur). Hence, to the cognizance of the non-omniscient observer, the action between point ℓ_4 and point ℓ_5 (i.e., $i2 := [-1; 2]$ representing the input from 2nd admin) is responsible for the access failure. It is easy to see that such results are consistent with Example 3.

More formally, this section presents the program trace semantics, builds a lattice of system behaviors by trace properties, proposes an observation function that derives from the observer's

cognizance and an inquiry function on system behaviors. Furthermore, this section formally defines responsibility as an abstraction of program semantics, using the observation function. To strengthen the intuition of responsibility analysis, the analysis of the access control program example will be illustrated step-by-step in the following:

3.2.1 Program Semantics. Generally speaking, no matter what type of program we are concerned with and no matter which programming language is used to implement that program, the corresponding program semantics can be represented as a set of execution traces.

We assume programs to be modeled as a transition system of the form $\mathrm{P} = \langle \mathbb{S}^i, \rightarrow \rangle$, as introduced in Section 2, where the intermediate trace semantics $[\![\mathrm{P}]\!]^{\mathsf{It}}$ is defined as the set of traces such that every two successive states are related by $\rightarrow$; the prefix trace semantics $[\![\mathrm{P}]\!]^{\mathsf{Pref}}$ is defined as the set of intermediate traces that start from initial states; and the maximal trace semantics $[\![\mathrm{P}]\!]^{\mathsf{Max}}$ is the set of prefix traces that either terminate at final states or the error state ω, or do not ever terminate. A trace σ is said to be *valid* for a program P if and only if $\sigma \in [\![\mathrm{P}]\!]^{\mathsf{Pref}}$. Obviously, the intermediate/prefix/maximal trace semantics do preserve the temporal ordering of actions, which is missed by the SEM used by actual causes [34, 35, 63].

Specifically, for the access control program example in Figure 4, its definition of maximal trace semantics refers to Example 1. For the sake of simplicity, it is assumed that the initial environment is fixed (e.g., the value of each variable is assumed to be 0 at the initial point ℓ_1), hence its maximal trace semantics $[\![\mathrm{P}]\!]^{\mathsf{Max}}$ consists of 32 traces that correspond to different input values from two admins and the system settings.

3.2.2 Lattice of System Behaviors of Interest.

Trace Property. A *trace property* is a set of traces. For any given system, many behaviors can be represented as a maximal trace property $\mathcal{T} \in \wp([\![\mathrm{P}]\!]^{\mathsf{Max}})$.

Example 4 (Access Control, Continued). For the access control program example in Figure 4, the behavior "Access Success" AS (i.e., the access to o succeeds) is represented by a set of maximal traces such that *acs* is strictly positive at point ℓ_8, i.e., $\mathsf{AS} = \{\sigma \in [\![\mathrm{P}]\!]^{\mathsf{Max}} \mid \exists \rho \in \mathbb{M}.\ \sigma_{[7]} = \langle \ell_8, \rho \rangle \wedge \rho(acs) > 0\}$. More precisely, along every trace in AS, both the admin inputs and the system settings are either 1 or 2. Since the initial environment is assumed to be fixed, AS consists of eight different traces.

The behavior "Access Failure" AF (i.e., the access to o fails) is represented as a set of maximal traces such that the value of *acs* is less than or equal to zero at point ℓ_8, which is the complement of AS, i.e., $\mathsf{AF} = [\![\mathrm{P}]\!]^{\mathsf{Max}} \backslash \mathsf{AS} = \{\sigma \in [\![\mathrm{P}]\!]^{\mathsf{Max}} \mid \exists \rho \in \mathbb{M}.\ \sigma_{[7]} = \langle \ell_8, \rho \rangle \wedge \rho(acs) <= 0\}$. It is not hard to see that, along every trace in AF, at least one input from the two admin is -1 or 0, while the input from system settings is 1 or 2. Hence, AF consists of 24 different traces.

Furthermore, the behavior AS can be split into two parts: $\mathsf{RO} = \{\sigma \in [\![\mathrm{P}]\!]^{\mathsf{Max}} \mid \exists \rho \in \mathbb{M}.\ \sigma_{[7]} = \langle \ell_8, \rho \rangle \wedge \rho(acs) = 1\}$ represents a stronger behavior "Read Only access is granted," which consists of four traces; and $\mathsf{RW} = \{\sigma \in [\![\mathrm{P}]\!]^{\mathsf{Max}} \mid \exists \rho \in \mathbb{M}.\ \sigma_{[7]} = \langle \ell_8, \rho \rangle \wedge \rho(acs) = 2\}$ represents another behavior "Read and Write access is granted," which also consists of four traces.

Lattice of System Behaviors of Interest. Here, we build a complete lattice of maximal trace properties, each of which represents a behavior of interest. Typically, such a lattice is of the form $\langle \mathcal{L}^{\mathsf{Max}}, \subseteq, \top^{\mathsf{Max}}, \bot^{\mathsf{Max}}, \cup, \cap \rangle$, where

- $\mathcal{L}^{\mathsf{Max}} \in \wp(\wp([\![\mathrm{P}]\!]^{\mathsf{Max}}))$ is a set of behaviors of interest, each of which is represented by a maximal trace property;
- $\top^{\mathsf{Max}} = [\![\mathrm{P}]\!]^{\mathsf{Max}}$, i.e., the top of the lattice is the weakest maximal trace property, which holds in every valid maximal trace;

— $\bot^{\mathsf{Max}} = \emptyset$, i.e., the bottom of the lattice is the strongest property such that no valid trace has this property, hence it is used to represent the property of invalidity;
— $\subseteq$ is the standard set inclusion operation;
— $\cup\!\!\!\cdot$ and $\cap\!\!\!\cdot$ are join and meet operations, which might not be the standard $\cup$ and $\cap$, since $\mathcal{L}^{\mathsf{Max}}$ is a subset of $\wp(\llbracket \mathrm{P} \rrbracket^{\mathsf{Max}})$ but not necessarily a sublattice.

For any given system, there is possibly more than one way to build the complete lattice of maximal trace properties, depending on which behaviors are of interest. A special case of lattice is the power set of maximal trace semantics, i.e., $\mathcal{L}^{\mathsf{Max}} = \wp(\llbracket \mathrm{P} \rrbracket^{\mathsf{Max}})$, which can be used to examine the responsibility for every possible behavior in the system. However, in most cases, a single behavior is of interest, and it is sufficient to adopt a lattice with only four elements: $\mathcal{B}$ representing the behavior of interest, $\llbracket \mathrm{P} \rrbracket^{\mathsf{Max}} \backslash \mathcal{B}$ representing the complement of the behavior of interest, as well as the top $\llbracket \mathrm{P} \rrbracket^{\mathsf{Max}}$ and bottom $\emptyset$. Particularly, if $\mathcal{B}$ is equal to $\llbracket \mathrm{P} \rrbracket^{\mathsf{Max}}$, i.e., every valid maximal trace in the system has this behavior of interest, then a trivial lattice with only the top and bottom is built, from which no responsibility can be found, making the corresponding analysis futile.

Example 5 (Access Control, Continued). For the access control program, there are two possible ways to build the lattice of maximal trace properties. To start with, we consider the lattice displayed in Figure 5, which consists of six elements. Regarding whether the access to o fails or not, the top $\top^{\mathsf{Max}} = \llbracket \mathrm{P} \rrbracket^{\mathsf{Max}}$ is split into two properties "Access Failure" AF and "Access Success" AS, which are formally defined in Example 4 such that $\mathsf{AF} \cup\!\!\!\cdot\, \mathsf{AS} = \llbracket \mathrm{P} \rrbracket^{\mathsf{Max}}$ and $\mathsf{AF} \cap\!\!\!\cdot\, \mathsf{AS} = \emptyset$. Furthermore, regarding whether the write access is granted or not, AS is split into "Read Only access is granted" RO and "Read and Write access is granted" RW, such that $\mathsf{RO} \cup\!\!\!\cdot\, \mathsf{RW} = \mathsf{AS}$ and $\mathsf{RO} \cap\!\!\!\cdot\, \mathsf{RW} = \emptyset$. With the assistance of such a lattice of system behaviors, we can determine not only the responsible entity for access failure/success, but also the entity in charge of the write access.

Meanwhile, if we are interested in only one behavior (e.g., "Access Failure" AF), then RO and RW can be simply removed from the lattice and we can get a lattice with four elements.

Prediction Abstraction. Although the maximal trace property is well-suited to represent system behaviors, it does not reveal the point along the maximal trace from which a property is guaranteed to hold later in the execution. Thus, we propose to abstract every maximal trace property $\mathcal{X} \in \mathcal{L}^{\mathsf{Max}}$ into a set $\alpha_{\mathsf{Pred}}(\llbracket \mathrm{P} \rrbracket^{\mathsf{Max}})\mathcal{X}$ of prefixes of maximal traces in $\mathcal{X}$, excluding those whose maximal prolongation may not satisfy the property $\mathcal{X}$. This abstraction is called *prediction abstraction*, and $\alpha_{\mathsf{Pred}}(\llbracket \mathrm{P} \rrbracket^{\mathsf{Max}})\mathcal{X}$ is called the *prediction trace property* corresponding to $\mathcal{X}$. It is easy to see that $\alpha_{\mathsf{Pred}}(\llbracket \mathrm{P} \rrbracket^{\mathsf{Max}})\mathcal{X}$ is a superset of $\mathcal{X}$ and is not necessarily prefix-closed.

$$\begin{array}{rcll} \alpha_{\mathsf{Pred}}(\llbracket \mathrm{P} \rrbracket^{\mathsf{Max}}) & \in & \wp(\mathbb{S}^{*\infty}) \mapsto \wp(\mathbb{S}^{*\infty}) & \text{prediction abstraction} \\ \alpha_{\mathsf{Pred}}(\llbracket \mathrm{P} \rrbracket^{\mathsf{Max}})\mathcal{X} & \triangleq & \{\sigma \in \mathsf{Pref}(\mathcal{X}) \mid \forall \sigma' \in \llbracket \mathrm{P} \rrbracket^{\mathsf{Max}}.\ \sigma \preceq \sigma' \Rightarrow \sigma' \in \mathcal{X}\} & \\ \gamma_{\mathsf{Pred}}(\llbracket \mathrm{P} \rrbracket^{\mathsf{Max}}) & \in & \wp(\mathbb{S}^{*\infty}) \mapsto \wp(\mathbb{S}^{*\infty}) & \text{prediction concretization} \\ \gamma_{\mathsf{Pred}}(\llbracket \mathrm{P} \rrbracket^{\mathsf{Max}})\mathcal{Y} & \triangleq & \{\sigma \in \mathcal{Y} \mid \sigma \in \llbracket \mathrm{P} \rrbracket^{\mathsf{Max}}\} = \mathcal{Y} \cap \llbracket \mathrm{P} \rrbracket^{\mathsf{Max}} & \end{array}$$

By the above definition, for any program P, every valid maximal trace σ' that is greater than or equal to a prefix trace σ in $\alpha_{\mathsf{Pred}}(\llbracket \mathrm{P} \rrbracket^{\mathsf{Max}})\mathcal{X}$ is guaranteed to have the maximal trace property $\mathcal{X}$ (i.e., $\sigma' \in \mathcal{X}$). Hence, the prefix traces in $\alpha_{\mathsf{Pred}}(\llbracket \mathrm{P} \rrbracket^{\mathsf{Max}})\mathcal{X}$ gives a hint on the point along the maximal trace from which the property $\mathcal{X}$ is guaranteed to hold. Formally, we have the following lemma:

Lemma 1. *For any maximal trace property $\mathcal{X} \in \wp(\llbracket \mathrm{P} \rrbracket^{\mathsf{Max}})$, if a prefix trace σ belongs to the prediction trace property $\alpha_{\mathsf{Pred}}(\llbracket \mathrm{P} \rrbracket^{\mathsf{Max}})\mathcal{X}$, then σ guarantees the satisfaction of property $\mathcal{X}$ (i.e., every valid maximal trace that is greater than or equal to σ is guaranteed to have property $\mathcal{X}$).*

Moreover, we have a Galois isomorphism between maximal trace properties and prediction trace properties:

$$\langle \wp([\![\mathrm{P}]\!]^{\mathsf{Max}}),\ \subseteq\rangle \underset{\alpha_{\mathsf{Pred}}([\![\mathrm{P}]\!]^{\mathsf{Max}})}{\overset{\gamma_{\mathsf{Pred}}([\![\mathrm{P}]\!]^{\mathsf{Max}})}{\leftleftarrows}} \langle \bar{\alpha}_{\mathsf{Pred}}\{[\![\mathrm{P}]\!]^{\mathsf{Max}}\}(\wp([\![\mathrm{P}]\!]^{\mathsf{Max}})),\ \subseteq\rangle, \tag{1}$$

where the abstract domain is obtained by a function $\bar{\alpha}_{\mathsf{Pred}}\{[\![\mathrm{P}]\!]^{\mathsf{Max}}\} \in \wp(\wp(\mathbb{S}^{*\infty})) \mapsto \wp(\wp(\mathbb{S}^{*\infty}))$ such that $\bar{\alpha}_{\mathsf{Pred}}\{[\![\mathrm{P}]\!]^{\mathsf{Max}}\}(\mathfrak{X}) \triangleq \{\alpha_{\mathsf{Pred}}([\![\mathrm{P}]\!]^{\mathsf{Max}})\mathcal{X} \mid \mathcal{X} \in \mathfrak{X}\}$.

Corollary 1. *Given the semantics $[\![\mathrm{P}]\!]^{\mathsf{Max}}$ and lattice $\mathcal{L}^{\mathsf{Max}}$ of system behaviors, for any maximal trace property $\mathcal{T} \in \mathcal{L}^{\mathsf{Max}}$, if a trace σ belongs to the prediction trace property that corresponds to $\mathcal{T}$, then every valid trace greater than σ belongs to that prediction trace property too. I.e., $\forall \mathcal{T} \in \mathcal{L}^{\mathsf{Max}}.\ \forall \sigma, \sigma' \in [\![\mathrm{P}]\!]^{\mathsf{Pref}}.\ (\sigma \in \alpha_{\mathsf{Pred}}([\![\mathrm{P}]\!]^{\mathsf{Max}})\mathcal{T} \wedge \sigma \preceq \sigma') \Rightarrow \sigma' \in \alpha_{\mathsf{Pred}}([\![\mathrm{P}]\!]^{\mathsf{Max}})\mathcal{T}$.*

Corollary 2. *Given the semantics $[\![\mathrm{P}]\!]^{\mathsf{Max}}$ and the lattice $\mathcal{L}^{\mathsf{Max}}$ of system behaviors, for any maximal trace property $\mathcal{T} \in \mathcal{L}^{\mathsf{Max}}$ and any valid prefix trace π that is not maximal, if every valid prefix trace πs that concatenates π with a new event s belongs to the prediction trace property $\alpha_{\mathsf{Pred}}([\![\mathrm{P}]\!]^{\mathsf{Max}})\mathcal{T}$, then π belongs to $\alpha_{\mathsf{Pred}}([\![\mathrm{P}]\!]^{\mathsf{Max}})\mathcal{T}$ too.*

Formally, $\forall \mathcal{T} \in \mathcal{L}^{\mathsf{Max}}.\ \forall \pi \in [\![\mathrm{P}]\!]^{\mathsf{Pref}} \backslash [\![\mathrm{P}]\!]^{\mathsf{Max}}.\ (\forall s \in \mathbb{S}.\ \pi s \in [\![\mathrm{P}]\!]^{\mathsf{Pref}} \Rightarrow \pi s \in \alpha_{\mathsf{Pred}}([\![\mathrm{P}]\!]^{\mathsf{Max}})\mathcal{T}) \Rightarrow \pi \in \alpha_{\mathsf{Pred}}([\![\mathrm{P}]\!]^{\mathsf{Max}})\mathcal{T}$.

Example 6 (Access Control, Continued). By the function $\alpha_{\mathsf{Pred}}([\![\mathrm{P}]\!]^{\mathsf{Max}})$, each behavior in the lattice $\mathcal{L}^{\mathsf{Max}}$ of Example 5 can be abstracted into a prediction trace property:

- $\alpha_{\mathsf{Pred}}([\![\mathrm{P}]\!]^{\mathsf{Max}})\top^{\mathsf{Max}} = [\![\mathrm{P}]\!]^{\mathsf{Pref}}$, i.e., every valid prefix trace in $[\![\mathrm{P}]\!]^{\mathsf{Pref}}$ guarantees $\top^{\mathsf{Max}}$.
- $\alpha_{\mathsf{Pred}}([\![\mathrm{P}]\!]^{\mathsf{Max}})\mathtt{AF} = \{\sigma \in [\![\mathrm{P}]\!]^{\mathsf{Pref}} \mid \exists \rho_1 \in \mathbb{M}, v \in \{-1, 0\}, v' \in \{1, 2\}.\ \langle \ell_1,\ \rho_1\rangle\langle \ell_2,\ \rho_2 = \rho_1[apv \mapsto 1]\rangle\langle \ell_3,\ \rho_3 = \rho_2[i1 \mapsto v]\rangle \preceq \sigma \vee \langle \ell_1,\ \rho_1\rangle\langle \ell_2,\ \rho_2 = \rho_1[apv \mapsto 1]\rangle\langle \ell_3,\ \rho_3 = \rho_2[i1 \mapsto v']\rangle\langle \ell_4,\ \rho_4 = \rho_3\rangle\langle \ell_5,\ \rho_5 = \rho_4[i2 \mapsto v]\rangle \preceq \sigma\}$. For any valid prefix trace σ, if at least one input from the two admins is -1 or 0, then the behavior "Access Failure" AF is guaranteed to occur in all the maximal traces that are greater than or equal to σ.
- $\alpha_{\mathsf{Pred}}([\![\mathrm{P}]\!]^{\mathsf{Max}})\mathtt{AS} = \{\sigma \in [\![\mathrm{P}]\!]^{\mathsf{Pref}} \mid \exists \rho_1 \in \mathbb{M}, v \in \{1, 2\}, v' \in \{1, 2\}.\ \langle \ell_1,\ \rho_1\rangle\langle \ell_2,\ \rho_2 = \rho_1[apv \mapsto 1]\rangle\langle \ell_3,\ \rho_3 = \rho_2[i1 \mapsto v]\rangle\langle \ell_4,\ \rho_4 = \rho_3\rangle\langle \ell_5,\ \rho_5 = \rho_4[i2 \mapsto v']\rangle \preceq \sigma\}$. For any valid prefix trace σ, if the inputs from both admins are 1 or 2, then "Access Success" AS is guaranteed.
- $\alpha_{\mathsf{Pred}}([\![\mathrm{P}]\!]^{\mathsf{Max}})\mathtt{RO} = \{\sigma \in [\![\mathrm{P}]\!]^{\mathsf{Pref}} \mid \exists \rho_1 \in \mathbb{M}, v \in \{1, 2\}, v' \in \{1, 2\}.\ \langle \ell_1,\ \rho_1\rangle\langle \ell_2,\ \rho_2 = \rho_1[apv \mapsto 1]\rangle\langle \ell_3,\ \rho_3 = \rho_2[i1 \mapsto v]\rangle\langle \ell_4,\ \rho_4 = \rho_3\rangle\langle \ell_5,\ \rho_5 = \rho_4[i2 \mapsto v']\rangle\langle \ell_6,\ \rho_6 = \rho_5\rangle\langle \ell_7,\ \rho_7 = \rho_6[typ \mapsto 1]\rangle \preceq \sigma\}$. For any valid trace, if the inputs from both admins are 1 or 2 and the input from system settings is 1, then it guarantees "Read Only access is granted" RO.
- $\alpha_{\mathsf{Pred}}([\![\mathrm{P}]\!]^{\mathsf{Max}})\mathtt{RW} = \{\sigma \in [\![\mathrm{P}]\!]^{\mathsf{Pref}} \mid \exists \rho_1 \in \mathbb{M}, v \in \{1, 2\}, v' \in \{1, 2\}.\ \langle \ell_1,\ \rho_1\rangle\langle \ell_2,\ \rho_2 = \rho_1[apv \mapsto 1]\rangle\langle \ell_3,\ \rho_3 = \rho_2[i1 \mapsto v]\rangle\langle \ell_4,\ \rho_4 = \rho_3\rangle\langle \ell_5,\ \rho_5 = \rho_4[i2 \mapsto v']\rangle\langle \ell_6,\ \rho_6 = \rho_5\rangle\langle \ell_7,\ \rho_7 = \rho_6[typ \mapsto 2]\rangle \preceq \sigma\}$. For any valid trace, if the inputs from both admins are 1 or 2 and the input from system settings is 2, then it guarantees "Read and Write access is granted" RW.
- $\alpha_{\mathsf{Pred}}([\![\mathrm{P}]\!]^{\mathsf{Max}})\bot^{\mathsf{Max}} = \emptyset$, i.e., no valid trace can guarantee the bottom $\bot^{\mathsf{Max}}$.

3.2.3 Observation of System Behaviors. Let $[\![\mathrm{P}]\!]^{\mathsf{Max}}$ be the maximal trace semantics and $\mathcal{L}^{\mathsf{Max}}$ be the lattice of system behaviors designed as in Section 3.2.2. Given any prefix trace $\sigma \in \mathbb{S}^{*\infty}$, an observer can learn some information from it, more precisely, a maximal trace property $\mathcal{T} \in \mathcal{L}^{\mathsf{Max}}$ that is guaranteed by σ from the observer's perspective. In this section, an *observation* function $\mathbb{O}$ is proposed to represent such a "property learning process" of the observer, which is formally defined in the following three steps.

(1) Inquiry Function. First, an *inquiry function* $\mathbb{I}$ is defined to map every trace $\sigma \in \mathbb{S}^{*\infty}$ to the strongest maximal trace property in $\mathcal{L}^{\mathsf{Max}}$ that σ can guarantee.

$\mathbb{I} \in \wp(\mathbb{S}^{*\infty}) \mapsto \wp(\wp(\mathbb{S}^{*\infty})) \mapsto \mathbb{S}^{*\infty} \mapsto \wp(\mathbb{S}^{*\infty})$ inquiry (2)

$\mathbb{I}(\llbracket \mathrm{P} \rrbracket^{\mathsf{Max}}, \mathcal{L}^{\mathsf{Max}}, \sigma) \triangleq$

$\quad \mathit{let}\ \alpha_{\mathsf{Pred}}(\mathcal{S})\mathcal{T} = \{\psi \in \mathsf{Pref}(\mathcal{T}) \mid \forall \psi' \in \mathcal{S}.\ \psi \preceq \psi' \Rightarrow \psi' \in \mathcal{T}\}\ \mathit{in}$ abstraction from (1)

$\quad\quad \cap\{\mathcal{T} \in \mathcal{L}^{\mathsf{Max}} \mid \sigma \in \alpha_{\mathsf{Pred}}(\llbracket \mathrm{P} \rrbracket^{\mathsf{Max}})\mathcal{T}\}$

Specially, for every invalid trace $\sigma \notin \llbracket \mathrm{P} \rrbracket^{\mathsf{Pref}}$, there does not exist any $\mathcal{T} \in \mathcal{L}^{\mathsf{Max}}$ such that $\sigma \in \alpha_{\mathsf{Pred}}(\llbracket \mathrm{P} \rrbracket^{\mathsf{Max}})\mathcal{T}$, thus $\mathbb{I}(\llbracket \mathrm{P} \rrbracket^{\mathsf{Max}}, \mathcal{L}^{\mathsf{Max}}, \sigma) = \emptyset = \bot^{\mathsf{Max}}$. In contrast, for any valid trace $\sigma \in \llbracket \mathrm{P} \rrbracket^{\mathsf{Pref}}$, it is ensured that $\sigma \in \alpha_{\mathsf{Pred}}(\llbracket \mathrm{P} \rrbracket^{\mathsf{Max}})\top^{\mathsf{Max}}$, hence $\mathbb{I}(\llbracket \mathrm{P} \rrbracket^{\mathsf{Max}}, \mathcal{L}^{\mathsf{Max}}, \sigma) \neq \bot^{\mathsf{Max}}$. Therefore, $\mathbb{I}(\llbracket \mathrm{P} \rrbracket^{\mathsf{Max}}, \mathcal{L}^{\mathsf{Max}}, \sigma) = \bot^{\mathsf{Max}}$ if and only if σ is invalid.

Example 7 (Access Control, Continued). Using the maximal trace semantics $\llbracket \mathrm{P} \rrbracket^{\mathsf{Max}}$ from Example 1 and the lattice of system behaviors $\mathcal{L}^{\mathsf{Max}}$ from Example 5, here we define the inquiry function $\mathbb{I}$ for the access control program such that for any initial environment $\rho_1 \in \mathbb{M}$:

- $\mathbb{I}(\llbracket \mathrm{P} \rrbracket^{\mathsf{Max}}, \mathcal{L}^{\mathsf{Max}}, \langle \ell_1,\ \rho_1 \rangle \langle \ell_2,\ \rho_2 = \rho_1[\mathit{apv} \mapsto 1] \rangle) = \top^{\mathsf{Max}}$, i.e., every prefix trace that terminates at point ℓ_2 (before the admins input their decisions) can guarantee only $\top^{\mathsf{Max}}$.
- $\mathbb{I}(\llbracket \mathrm{P} \rrbracket^{\mathsf{Max}}, \mathcal{L}^{\mathsf{Max}}, \langle \ell_1,\ \rho_1 \rangle \langle \ell_2,\ \rho_2 = \rho_1[\mathit{apv} \mapsto 1] \rangle \langle \ell_3,\ \rho_3 = \rho_2[\mathit{i1} \mapsto 0] \rangle) = \mathtt{AF}$, i.e., after the 1st admin inputs 0, the behavior "Access Failure" AF is guaranteed.
- $\mathbb{I}(\llbracket \mathrm{P} \rrbracket^{\mathsf{Max}}, \mathcal{L}^{\mathsf{Max}}, \langle \ell_1,\ \rho_1 \rangle \langle \ell_2,\ \rho_2 = \rho_1[\mathit{apv} \mapsto 1] \rangle \langle \ell_3,\ \rho_3 = \rho_2[\mathit{i1} \mapsto 1] \rangle) = \mathbb{I}(\llbracket \mathrm{P} \rrbracket^{\mathsf{Max}}, \mathcal{L}^{\mathsf{Max}}, \langle \ell_1,\ \rho_1 \rangle \langle \ell_2,\ \rho_2 = \rho_1[\mathit{apv} \mapsto 1] \rangle \langle \ell_3,\ \rho_3 = \rho_2[\mathit{i1} \mapsto 1] \rangle \langle \ell_4,\ \rho_4 = \rho_3 \rangle) = \top^{\mathsf{Max}}$, i.e., if the first admin inputs 1, then only the top $\top^{\mathsf{Max}}$ can be guaranteed before the second admin inputs her/his decision.
- $\mathbb{I}(\llbracket \mathrm{P} \rrbracket^{\mathsf{Max}}, \mathcal{L}^{\mathsf{Max}}, \langle \ell_1,\ \rho_1 \rangle \langle \ell_2,\ \rho_2 = \rho_1[\mathit{apv} \mapsto 1] \rangle \langle \ell_3,\ \rho_3 = \rho_2[\mathit{i1} \mapsto 1] \rangle \langle \ell_4,\ \rho_4 = \rho_3 \rangle \langle \ell_5,\ \rho_5 = \rho_4[\mathit{i2} \mapsto 0] \rangle) = \mathtt{AF}$, i.e., after the second admin inputs 0, the behavior "Access Failure" AF is guaranteed.
- $\mathbb{I}(\llbracket \mathrm{P} \rrbracket^{\mathsf{Max}}, \mathcal{L}^{\mathsf{Max}}, \langle \ell_1,\ \rho_1 \rangle \langle \ell_2,\ \rho_2 = \rho_1[\mathit{apv} \mapsto 1] \rangle \langle \ell_3,\ \rho_3 = \rho_2[\mathit{i1} \mapsto 1] \rangle \langle \ell_4,\ \rho_4 = \rho_3 \rangle \langle \ell_5,\ \rho_5 = \rho_4[\mathit{i2} \mapsto 1] \rangle) = \mathtt{AS}$, i.e., if both two admin inputs 1, then "Access Success" AS is guaranteed.
- $\mathbb{I}(\llbracket \mathrm{P} \rrbracket^{\mathsf{Max}}, \mathcal{L}^{\mathsf{Max}}, \langle \ell_1,\ \rho_1 \rangle \langle \ell_2,\ \rho_2 = \rho_1[\mathit{apv} \mapsto 1] \rangle \langle \ell_3,\ \rho_3 = \rho_2[\mathit{i1} \mapsto 1] \rangle \langle \ell_4,\ \rho_4 = \rho_3 \rangle \langle \ell_5,\ \rho_5 = \rho_4[\mathit{i2} \mapsto 1] \rangle \langle \ell_6,\ \rho_6 = \rho_5 \rangle \langle \ell_7,\ \rho_7 = \rho_6[\mathit{typ} \mapsto 1] \rangle) = \mathtt{RO}$, i.e., if both two admin input 1, then after the input from system settings is set as 1, a stronger property "Read Only access is granted" RO is guaranteed.
- $\mathbb{I}(\llbracket \mathrm{P} \rrbracket^{\mathsf{Max}}, \mathcal{L}^{\mathsf{Max}}, \langle \ell_1,\ \rho_1 \rangle \langle \ell_2,\ \rho_2 = \rho_1[\mathit{apv} \mapsto 1] \rangle \langle \ell_3,\ \rho_3 = \rho_2[\mathit{i1} \mapsto 1] \rangle \langle \ell_4,\ \rho_4 = \rho_3 \rangle \langle \ell_5,\ \rho_5 = \rho_4[\mathit{i2} \mapsto 1] \rangle \langle \ell_6,\ \rho_6 = \rho_5 \rangle \langle \ell_7,\ \rho_7 = \rho_6[\mathit{typ} \mapsto 2] \rangle) = \mathtt{RW}$, i.e., if both two admin input 1, then after the input from system settings is set as 2, a stronger property "Read and Write access is granted" RW is guaranteed.

Corollary 3. *Given the semantics $\llbracket \mathrm{P} \rrbracket^{\mathsf{Max}}$ and lattice $\mathcal{L}^{\mathsf{Max}}$ of system behaviors, if the inquiry function $\mathbb{I}$ maps a trace σ to a maximal trace property $\mathcal{T} \in \mathcal{L}^{\mathsf{Max}}$, then σ guarantees the satisfaction of $\mathcal{T}$ (i.e., every valid maximal trace that is greater than or equal to σ is guaranteed to have property $\mathcal{T}$).*

Lemma 2. *Given the semantics $\llbracket \mathrm{P} \rrbracket^{\mathsf{Max}}$ and lattice $\mathcal{L}^{\mathsf{Max}}$ of system behaviors, the inquiry function $\mathbb{I}(\llbracket \mathrm{P} \rrbracket^{\mathsf{Max}}, \mathcal{L}^{\mathsf{Max}})$ is decreasing on the inquired trace σ: the greater (longer) σ is, the stronger property it can guarantee. I.e., $\forall \sigma, \sigma' \in \mathbb{S}^{*\infty}.\ \sigma \preceq \sigma' \Rightarrow \mathbb{I}(\llbracket \mathrm{P} \rrbracket^{\mathsf{Max}}, \mathcal{L}^{\mathsf{Max}}, \sigma) \supseteq \mathbb{I}(\llbracket \mathrm{P} \rrbracket^{\mathsf{Max}}, \mathcal{L}^{\mathsf{Max}}, \sigma')$.*

Corollary 4. *Given the semantics $\llbracket \mathrm{P} \rrbracket^{\mathsf{Max}}$ and lattice $\mathcal{L}^{\mathsf{Max}}$ of behaviors, $\forall \sigma \in \llbracket \mathrm{P} \rrbracket^{\mathsf{Pref}} \setminus \llbracket \mathrm{P} \rrbracket^{\mathsf{Max}}$. $\mathbb{I}(\llbracket \mathrm{P} \rrbracket^{\mathsf{Max}}, \mathcal{L}^{\mathsf{Max}}, \sigma) = \bigsqcup_{s \in \mathbb{S}} \mathbb{I}(\llbracket \mathrm{P} \rrbracket^{\mathsf{Max}}, \mathcal{L}^{\mathsf{Max}}, \sigma s) = \bigsqcup_{\sigma s \in \llbracket \mathrm{P} \rrbracket^{\mathsf{Pref}}} \mathbb{I}(\llbracket \mathrm{P} \rrbracket^{\mathsf{Max}}, \mathcal{L}^{\mathsf{Max}}, \sigma s)$.*

(2) Cognizance Function. As discussed in (P3) of Section 3.1, it is necessary to take the observer's cognizance into account. Specifically, in program security, the cognizance can represent attackers'

capabilities, e.g., what they can learn from program executions (see Section 3.3.2 for more details). Given a trace σ (not necessarily valid), if the observer cannot distinguish σ from some other traces, then she/he does not have an omniscient cognizance of σ, and the *cognizance* function $\mathbb{C}(\sigma)$ is defined to include all traces indistinguishable from σ.

$$\mathbb{C} \in \mathbb{S}^{*\infty} \mapsto \wp(\mathbb{S}^{*\infty}) \qquad \text{cognizance (3)}$$

$$\mathbb{C}(\sigma) \triangleq \{\sigma' \in \mathbb{S}^{*\infty} \mid \text{the observer cannot distinguish } \sigma' \text{ from } \sigma\}$$

Such a cognizance function is extensive, i.e., $\forall \sigma \in \mathbb{S}^{*\infty}.\ \sigma \in \mathbb{C}(\sigma)$. In particular, there is an *omniscient observer* and its corresponding cognizance function is denoted as $\mathbb{C}_o$ such that $\forall \sigma \in \mathbb{S}^{*\infty}.\ \mathbb{C}_o(\sigma) = \{\sigma\}$, which means that every trace is unambiguous to the omniscient observer.

To facilitate the proof of some desired properties for the observation function defined later, two assumptions are made here without loss of generality:

(A1) The cognizance of a trace $\sigma\sigma'$ is the concatenation of cognizances of σ and σ'. I.e., $\forall \sigma, \sigma' \in \mathbb{S}^{*\infty}.\ \mathbb{C}(\sigma\sigma') = \{\pi\pi' \mid \pi \in \mathbb{C}(\sigma) \wedge \pi' \in \mathbb{C}(\sigma')\}$. Specially, we require that $\mathbb{C}(\varepsilon) = \{\varepsilon\}$, otherwise $\mathbb{C}(\sigma) = \mathbb{C}(\sigma\varepsilon) = \{\pi\pi' \mid \pi \in \mathbb{C}(\sigma) \wedge \pi' \in \mathbb{C}(\varepsilon)\}$ would include infinite many traces for any σ.

(A2) Given an invalid trace, the cognizance function would not return a valid trace. I.e., $\forall \sigma \in \mathbb{S}^{*\infty}.\ \sigma \notin [\![\mathrm{P}]\!]^{\mathsf{Pref}} \Rightarrow \mathbb{C}(\sigma) \cap [\![\mathrm{P}]\!]^{\mathsf{Pref}} = \emptyset$.

In practice, we can define an equivalence relation on traces that satisfy the above two assumptions, thus it is assumed that the observer cannot distinguish two traces if and only if they are equivalent. That is to say, for any trace σ, $\mathbb{C}(\sigma)$ is a class of traces that are equivalent to σ, and $\{\langle \sigma, \sigma' \rangle \mid \sigma' \in \mathbb{C}(\sigma)\}$ is an equivalence relation. By (A2) the cognizance cannot abstract an invalid trace into a valid one, it is therefore different from the abstractions in Reference [30] to define the "power of an attacker."

Corollary 5. *For any cognizance function $\mathbb{C}$, we have* $\bigcup_{s \in \mathbb{S}} \mathbb{C}(s) \supseteq \mathbb{S}$.

Proof. This corollary follows the fact that the cognizance function $\mathbb{C}$ is extensive. □

Example 8 (Access Control, Continued). For the access control program, consider the cognizance function for two different observers.

(i) For an omniscient observer: $\forall \sigma \in \mathbb{S}^{*\infty}.\ \mathbb{C}_o(\sigma) = \{\sigma\}$.

(ii) For an observer who is unaware of the input from 1st admin or the value of *apv*: $\mathbb{C}(\langle \ell_1, \rho_1 \rangle \langle \ell_2, \rho_2 = \rho_1[apv \mapsto 1] \rangle \langle \ell_3, \rho_3 = \rho_2[i1 \mapsto 0] \rangle) = \{\langle \ell_1, \rho_1 \rangle \langle \ell_2, \rho_2 = \rho_1[apv \mapsto 1] \rangle \langle \ell_3, \rho_3 = \rho_2[i1 \mapsto -1] \rangle, \langle \ell_1, \rho_1 \rangle \langle \ell_2, \rho_2 = \rho_1[apv \mapsto 1] \rangle \langle \ell_3, \rho_3 = \rho_2[i1 \mapsto 0] \rangle, \langle \ell_1, \rho_1 \rangle \langle \ell_2, \rho_2 = \rho_1[apv \mapsto 1] \rangle \langle \ell_3, \rho_3 = \rho_2[i1 \mapsto 1] \rangle, \langle \ell_1, \rho_1 \rangle \langle \ell_2, \rho_2 = \rho_1[apv \mapsto 1] \rangle \langle \ell_3, \rho_3 = \rho_2[i1 \mapsto 2] \rangle\}$, i.e., this observer cannot distinguish whether the input from 1st admin is -1 or 0 or 1 or 2.

Similarly, for a prefix trace in which the inputs from both two admins are zeros, $\mathbb{C}(\langle \ell_1, \rho_1 \rangle \langle \ell_2, \rho_2 = \rho_1[apv \mapsto 1] \rangle \langle \ell_3, \rho_3 = \rho_2[i1 \mapsto 0] \rangle \langle \ell_4, \rho_4 = \rho_3 \rangle \langle \ell_5, \rho_5 = \rho_4[i2 \mapsto 0] \rangle) = \{\langle \ell_1, \rho_1 \rangle \langle \ell_2, \rho_2 = \rho_1[apv \mapsto 1] \rangle \langle \ell_3, \rho_3 = \rho_2[i1 \mapsto -1] \rangle \langle \ell_4, \rho_4 = \rho_3 \rangle \langle \ell_5, \rho_5 = \rho_4[i2 \mapsto 0] \rangle, \langle \ell_1, \rho_1 \rangle \langle \ell_2, \rho_2 = \rho_1[apv \mapsto 1] \rangle \langle \ell_3, \rho_3 = \rho_2[i1 \mapsto 0] \rangle \langle \ell_4, \rho_4 = \rho_3 \rangle \langle \ell_5, \rho_5 = \rho_4[i2 \mapsto 0] \rangle, \langle \ell_1, \rho_1 \rangle \langle \ell_2, \rho_2 = \rho_1[apv \mapsto 1] \rangle \langle \ell_3, \rho_3 = \rho_2[i1 \mapsto 1] \rangle \langle \ell_4, \rho_4 = \rho_3 \rangle \langle \ell_5, \rho_5 = \rho_4[i2 \mapsto 0] \rangle, \langle \ell_1, \rho_1 \rangle \langle \ell_2, \rho_2 = \rho_1[apv \mapsto 1] \rangle \langle \ell_3, \rho_3 = \rho_2[i1 \mapsto 2] \rangle \langle \ell_4, \rho_4 = \rho_3 \rangle \langle \ell_5, \rho_5 = \rho_4[i2 \mapsto 0] \rangle\}$ consists of four traces, such that the value of *i1* is not distinguishable while the value of *i2* is. In the same way, the cognizance on other traces can be defined.

(3) Observation Function. For an observer with cognizance function $\mathbb{C}$, given a single trace σ, the observer cannot distinguish σ with other traces in $\mathbb{C}(\sigma)$. To formalize the information that the observer can learn from σ, we apply the inquiry function $\mathbb{I}$ on each trace in $\mathbb{C}(\sigma)$ and get a

set of maximal trace properties. By joining them together, we get the strongest property in $\mathcal{L}^{\mathsf{Max}}$ that σ can guarantee from the observer's perspective. Such a process is defined as the *observation* function $\mathbb{O}(\llbracket \mathrm{P} \rrbracket^{\mathsf{Max}}, \mathcal{L}^{\mathsf{Max}}, \mathbb{C}, \sigma)$.

$$\mathbb{O} \in \wp(\mathbb{S}^{*\infty}) \mapsto \wp(\wp(\mathbb{S}^{*\infty})) \mapsto (\mathbb{S}^{*\infty} \mapsto \wp(\mathbb{S}^{*\infty})) \mapsto \mathbb{S}^{*\infty} \mapsto \wp(\mathbb{S}^{*\infty})$$

$$\begin{array}{ll}
\mathbb{O}(\llbracket \mathrm{P} \rrbracket^{\mathsf{Max}}, \mathcal{L}^{\mathsf{Max}}, \mathbb{C}, \sigma) \triangleq & \text{observation (4)} \\
\quad \textit{let } \alpha_{\mathsf{Pred}}(\mathcal{S})\mathcal{T} = \{\psi \in \mathsf{Pref}(\mathcal{T}) \mid \forall \psi' \in \mathcal{S}.\ \psi \preceq \psi' \Rightarrow \psi' \in \mathcal{T}\} \textit{ in} & \text{abstraction from (1)} \\
\quad\quad \textit{let } \mathbb{I}(\mathcal{S}, \mathcal{L}, \psi) = \cap\{\mathcal{T} \in \mathcal{L} \mid \psi \in \alpha_{\mathsf{Pred}}(\mathcal{S})\mathcal{T}\} \textit{ in} & \text{inquiry from (2)} \\
\quad\quad\quad \cup\{\mathbb{I}(\llbracket \mathrm{P} \rrbracket^{\mathsf{Max}}, \mathcal{L}^{\mathsf{Max}}, \sigma') \mid \sigma' \in \mathbb{C}(\sigma)\}. &
\end{array}$$

From the above definition, it is easy to see that, for every invalid trace $\sigma \notin \llbracket \mathrm{P} \rrbracket^{\mathsf{Pref}}$, we have $\mathbb{O}(\llbracket \mathrm{P} \rrbracket^{\mathsf{Max}}, \mathcal{L}^{\mathsf{Max}}, \mathbb{C}, \sigma) = \bot^{\mathsf{Max}}$, since every trace σ' in $\mathbb{C}(\sigma)$ is invalid by (A2) and $\mathbb{I}(\llbracket \mathrm{P} \rrbracket^{\mathsf{Max}}, \mathcal{L}^{\mathsf{Max}}, \sigma') = \bot^{\mathsf{Max}}$. In addition, for an omniscient observer with cognizance function $\mathbb{C}_o$, its observation function $\mathbb{O}(\llbracket \mathrm{P} \rrbracket^{\mathsf{Max}}, \mathcal{L}^{\mathsf{Max}}, \mathbb{C}_o, \sigma) = \mathbb{I}(\llbracket \mathrm{P} \rrbracket^{\mathsf{Max}}, \mathcal{L}^{\mathsf{Max}}, \sigma)$.

Corollary 6. *Given the semantics $\llbracket \mathrm{P} \rrbracket^{\mathsf{Max}}$ and lattice $\mathcal{L}^{\mathsf{Max}}$ of system behaviors, for any observer with cognizance $\mathbb{C}$, if the corresponding observation function maps a trace σ to a maximal trace property $\mathcal{T} \in \mathcal{L}^{\mathsf{Max}}$, then σ guarantees the satisfaction of property $\mathcal{T}$ (i.e., every valid maximal trace that is greater than or equal to σ is guaranteed to have property $\mathcal{T}$).*

Corollary 7. *Given the semantics $\llbracket \mathrm{P} \rrbracket^{\mathsf{Max}}$, the lattice $\mathcal{L}^{\mathsf{Max}}$ of system behaviors, and the cognizance function $\mathbb{C}$, we have: $\forall \sigma \in \llbracket \mathrm{P} \rrbracket^{\mathsf{Pref}} \backslash \llbracket \mathrm{P} \rrbracket^{\mathsf{Max}}$. $\mathbb{O}(\llbracket \mathrm{P} \rrbracket^{\mathsf{Max}}, \mathcal{L}^{\mathsf{Max}}, \mathbb{C}, \sigma) = \biguplus_{s \in \mathbb{S}} \mathbb{O}(\llbracket \mathrm{P} \rrbracket^{\mathsf{Max}}, \mathcal{L}^{\mathsf{Max}}, \mathbb{C}, \sigma s) = \biguplus_{\sigma s \in \llbracket \mathrm{P} \rrbracket^{\mathsf{Pref}}} \mathbb{O}(\llbracket \mathrm{P} \rrbracket^{\mathsf{Max}}, \mathcal{L}^{\mathsf{Max}}, \mathbb{C}, \sigma s)$.*

Lemma 3. *Given the semantics $\llbracket \mathrm{P} \rrbracket^{\mathsf{Max}}$, lattice $\mathcal{L}^{\mathsf{Max}}$ of system behaviors, and cognizance function $\mathbb{C}$, the observation function $\mathbb{O}(\llbracket \mathrm{P} \rrbracket^{\mathsf{Max}}, \mathcal{L}^{\mathsf{Max}}, \mathbb{C})$ is decreasing on the observed trace σ: the greater (longer) σ is, the stronger property it can observe. I.e., $\forall \sigma, \sigma' \in \mathbb{S}^{*\infty}$. $\sigma \preceq \sigma' \Rightarrow \mathbb{O}(\llbracket \mathrm{P} \rrbracket^{\mathsf{Max}}, \mathcal{L}^{\mathsf{Max}}, \mathbb{C}, \sigma) \supseteq \mathbb{O}(\llbracket \mathrm{P} \rrbracket^{\mathsf{Max}}, \mathcal{L}^{\mathsf{Max}}, \mathbb{C}, \sigma')$.*

Example 9 (Access Control, Continued). For an omniscient observer, the observation function is identical to the inquiry function in Example 7. If the cognizance of a non-omniscient observer defined in Example 8 is adopted, then we get an observation function that works exactly the same as the dashed arrows in Figure 5:

- $\mathbb{O}(\llbracket \mathrm{P} \rrbracket^{\mathsf{Max}}, \mathcal{L}^{\mathsf{Max}}, \mathbb{C}, \langle \ell_1,\ \rho_1 \rangle\langle \ell_2,\ \rho_2 = \rho_1[apv \mapsto 1] \rangle\langle \ell_3,\ \rho_3 = \rho_2[i1 \mapsto 0] \rangle) = \cup\{\mathbb{I}(\llbracket \mathrm{P} \rrbracket^{\mathsf{Max}}, \mathcal{L}^{\mathsf{Max}}, \langle \ell_1,\ \rho_1 \rangle\langle \ell_2,\ \rho_2 = \rho_1[apv \mapsto 1] \rangle\langle \ell_3,\ \rho_3 = \rho_2[i1 \mapsto v] \rangle) \mid v \in \{-1, 0, 1, 2\}\} = \mathsf{AF} \cup \mathsf{AF} \cup \top^{\mathsf{Max}} \cup \top^{\mathsf{Max}} = \top^{\mathsf{Max}}$, i.e., even if the 1st admin already inputs 0, only $\top^{\mathsf{Max}}$ can be guaranteed from the perspective of the non-omniscient observer.
- $\mathbb{O}(\llbracket \mathrm{P} \rrbracket^{\mathsf{Max}}, \mathcal{L}^{\mathsf{Max}}, \mathbb{C}, \langle \ell_1,\ \rho_1 \rangle\langle \ell_2,\ \rho_2 = \rho_1[apv \mapsto 1] \rangle\langle \ell_3,\ \rho_3 = \rho_2[i1 \mapsto 0] \rangle\langle \ell_4,\ \rho_4 = \rho_3 \rangle\langle \ell_5,\ \rho_5 = \rho_4[i2 \mapsto 0] \rangle) = \cup\{\mathbb{I}(\llbracket \mathrm{P} \rrbracket^{\mathsf{Max}}, \mathcal{L}^{\mathsf{Max}}, \langle \ell_1, \rho_1 \rangle\langle \ell_2, \rho_2 = \rho_1[apv \mapsto 1] \rangle\langle \ell_3, \rho_3 = \rho_2[i1 \mapsto v] \rangle\langle \ell_4,\ \rho_4 = \rho_3 \rangle\langle \ell_5,\ \rho_5 = \rho_4[i2 \mapsto 0] \rangle) \mid v \in \{-1, 0, 1, 2\}\} = \mathsf{AF} \cup \mathsf{AF} \cup \mathsf{AF} \cup \mathsf{AF} = \mathsf{AF}$, i.e., only after the 2nd admin inputs 0 (or −1), "Access Failure" AF can be guaranteed from the perspective of the non-omniscient observer.
- $\mathbb{O}(\llbracket \mathrm{P} \rrbracket^{\mathsf{Max}}, \mathcal{L}^{\mathsf{Max}}, \mathbb{C}, \langle \ell_1,\ \rho_1 \rangle\langle \ell_2,\ \rho_2 = \rho_1[apv \mapsto 1] \rangle\langle \ell_3,\ \rho_3 = \rho_2[i1 \mapsto 0] \rangle\langle \ell_4,\ \rho_4 = \rho_3 \rangle\langle \ell_5,\ \rho_5 = \rho_4[i2 \mapsto 1] \rangle) = \cup\{\mathbb{I}(\llbracket \mathrm{P} \rrbracket^{\mathsf{Max}}, \mathcal{L}^{\mathsf{Max}}, \langle \ell_1, \rho_1 \rangle\langle \ell_2, \rho_2 = \rho_1[apv \mapsto 1] \rangle\langle \ell_3, \rho_3 = \rho_2[i1 \mapsto v] \rangle\langle \ell_4,\ \rho_4 = \rho_3 \rangle\langle \ell_5,\ \rho_5 = \rho_4[i2 \mapsto 1] \rangle) \mid v \in \{-1, 0, 1, 2\}\} = \mathsf{AF} \cup \mathsf{AF} \cup \top^{\mathsf{Max}} \cup \top^{\mathsf{Max}} = \top^{\mathsf{Max}}$, i.e., if the 2nd admin inputs 1 (or 2), then only the top $\top^{\mathsf{Max}}$ can be guaranteed from the perspective of the non-omniscient observer, even if the 1st admin already inputs 0 or −1.

3.2.4 Formal Definition of Responsibility. Using the three components of responsibility analysis introduced above, responsibility is formally defined as the *responsibility abstraction* α_R in (5).

Responsibility Abstraction α_R

$$\begin{aligned}
&\alpha_R \in \wp(\mathbb{S}^{*\infty}) \mapsto \wp(\wp(\mathbb{S}^{*\infty})) \mapsto (\mathbb{S}^{*\infty} \mapsto \wp(\mathbb{S}^{*\infty})) \mapsto \wp(\mathbb{S}^{*\infty}) \mapsto \wp(\mathbb{S}^{*\infty}) \\
&\quad \mapsto \wp(\mathbb{S}^* \times (\mathbb{S} \times \mathbb{S}) \times \mathbb{S}^{*\infty}) \qquad (5)\\
&\alpha_R([\![\mathrm{P}]\!]^{\mathrm{Max}}, \mathcal{L}^{\mathrm{Max}}, \mathbb{C}, \mathcal{B}, \mathcal{T}) \triangleq \\
&\mathit{let}\ \alpha_{\mathrm{Pred}}(\mathcal{S})T = \{\sigma \in \mathrm{Pref}(T) \mid \forall \sigma' \in \mathcal{S}.\ \sigma \preceq \sigma' \Rightarrow \sigma' \in T\}\ \mathit{in} \\
&\quad \mathit{let}\ \mathbb{I}(\mathcal{S}, \mathcal{L}, \sigma) = \curlywedge\{T \in \mathcal{L} \mid \sigma \in \alpha_{\mathrm{Pred}}(\mathcal{S})T\}\ \mathit{in} \\
&\quad\quad \mathit{let}\ \mathbb{O}(\mathcal{S}, \mathcal{L}, \mathbb{C}, \sigma) = \curlyvee\{\mathbb{I}(\mathcal{S}, \mathcal{L}, \sigma') \mid \sigma' \in \mathbb{C}(\sigma)\}\ \mathit{in} \\
&\quad\quad\quad \{\langle \sigma_{\mathrm{H}},\ \tau_{\mathrm{R}},\ \sigma_{\mathrm{F}} \rangle \mid \sigma_{\mathrm{H}}\tau_{\mathrm{R}}\sigma_{\mathrm{F}} \in \mathcal{T}\ \wedge\ \emptyset \subsetneq \mathbb{O}([\![\mathrm{P}]\!]^{\mathrm{Max}}, \mathcal{L}^{\mathrm{Max}}, \mathbb{C}, \sigma_{\mathrm{H}}\tau_{\mathrm{R}}) \subseteq \mathcal{B}\ \wedge \\
&\quad\quad\quad\quad\quad \mathbb{O}([\![\mathrm{P}]\!]^{\mathrm{Max}}, \mathcal{L}^{\mathrm{Max}}, \mathbb{C}, \sigma_{\mathrm{H}}) \nsubseteq \mathcal{B}\}
\end{aligned}$$

Specifically, the first parameter is the maximal trace semantics $[\![\mathrm{P}]\!]^{\mathrm{Max}}$, the second parameter is the lattice $\mathcal{L}^{\mathrm{Max}}$ of system behaviors, the third parameter is the cognizance function of a given observer, the fourth parameter is the behavior $\mathcal{B}$ whose responsibility is of interest, and the last parameter is the analyzed traces $\mathcal{T}$.

For every trace $\sigma \in \mathcal{T}$ to be analyzed, we split it into three parts such that $\sigma = \sigma_{\mathrm{H}}\tau_{\mathrm{R}}\sigma_{\mathrm{F}}$, where $\sigma_{\mathrm{H}} = s_0 \cdots s_{r-1} \in \mathbb{S}^*$ represents the *History* part of trace σ, the transition $\tau_{\mathrm{R}} = s_{r-1} \xrightarrow{a_{\mathrm{R}}} s_r$ represents the *Responsible* part of trace σ (which is a transition between two states, and the corresponding action a_{R} can be retrieved from the source code), and $\sigma_{\mathrm{F}} = s_r \cdots \in \mathbb{S}^{*\infty}$ represents the *Future* part of trace σ.

If $\emptyset \subsetneq \mathbb{O}([\![\mathrm{P}]\!]^{\mathrm{Max}}, \mathcal{L}^{\mathrm{Max}}, \mathbb{C}, \sigma_{\mathrm{H}}\tau_{\mathrm{R}}) \subseteq \mathcal{B} \wedge \mathbb{O}([\![\mathrm{P}]\!]^{\mathrm{Max}}, \mathcal{L}^{\mathrm{Max}}, \mathbb{C}, \sigma_{\mathrm{H}}) \nsubseteq \mathcal{B}$ holds, then σ_{H} does not guarantee the behavior $\mathcal{B}$, while $\sigma_{\mathrm{H}}\tau_{\mathrm{R}}$ guarantees a behavior that is at least as strong as $\mathcal{B}$ and is not the invalid trace property represented by $\bot^{\mathrm{Max}} = \emptyset$. Therefore, to the cognizance $\mathbb{C}$ of a given observer, the transition $\tau_{\mathrm{R}} = s_{r-1} \xrightarrow{a_{\mathrm{R}}} s_r$ (or, say, the action a_{R}) is said to be *responsible* for the behavior $\mathcal{B}$ in the trace $\sigma_{\mathrm{H}}\tau_{\mathrm{R}}\sigma_{\mathrm{F}}$.

Since $\alpha_R([\![\mathrm{P}]\!]^{\mathrm{Max}}, \mathcal{L}^{\mathrm{Max}}, \mathbb{C}, \mathcal{B})$ preserves joins on analyzed traces $\mathcal{T}$, we have a Galois connection [68]: $\langle \wp(\mathbb{S}^{*\infty}),\ \subseteq \rangle \underset{\alpha_R([\![\mathrm{P}]\!]^{\mathrm{Max}},\ \mathcal{L}^{\mathrm{Max}},\ \mathbb{C},\ \mathcal{B})}{\overset{\gamma_R([\![\mathrm{P}]\!]^{\mathrm{Max}},\ \mathcal{L}^{\mathrm{Max}},\ \mathbb{C},\ \mathcal{B})}{\leftrightarrows}} \langle \wp(\mathbb{S}^* \times (\mathbb{S} \times \mathbb{S}) \times \mathbb{S}^{*\infty}),\ \subseteq \rangle$.

It is worth noting that, compared with our original definition of responsibility abstraction α_R in References [25, 26] (which adopts the condition $\emptyset \subsetneq \mathbb{O}([\![\mathrm{P}]\!]^{\mathrm{Max}}, \mathcal{L}^{\mathrm{Max}}, \mathbb{C}, \sigma_{\mathrm{H}}\tau_{\mathrm{R}}) \subseteq \mathcal{B} \subsetneq \mathbb{O}([\![\mathrm{P}]\!]^{\mathrm{Max}}, \mathcal{L}^{\mathrm{Max}}, \mathbb{C}, \sigma_{\mathrm{H}})$), definition (5) proposed in this article is more generic: When the lattice of system behavior $\mathcal{L}^{\mathrm{Max}}$ is of complex structure (i.e., it consists of more than four elements), the observation $\mathbb{O}([\![\mathrm{P}]\!]^{\mathrm{Max}}, \mathcal{L}^{\mathrm{Max}}, \mathbb{C}, \sigma_{\mathrm{H}})$ may return a behavior that is incomparable with $\mathcal{B}$; as long as $\emptyset \subsetneq \mathbb{O}([\![\mathrm{P}]\!]^{\mathrm{Max}}, \mathcal{L}^{\mathrm{Max}}, \mathbb{C}, \sigma_{\mathrm{H}}\tau_{\mathrm{R}}) \subseteq \mathcal{B}$ holds after extending σ_{H} with τ_{R}, we know the transition τ_{R} shall be responsible for $\mathcal{B}$.

Theorem 1. *If τ_{R} is said to be responsible for a behavior $\mathcal{B}$ in a valid trace $\sigma_{\mathrm{H}}\tau_{\mathrm{R}}\sigma_{\mathrm{F}}$, then $\sigma_{\mathrm{H}}\tau_{\mathrm{R}}$ guarantees the occurrence of behavior $\mathcal{B}$, and there must exist another valid prefix trace $\sigma_{\mathrm{H}}\tau'_{\mathrm{R}}$ such that the behavior $\mathcal{B}$ is not guaranteed.*

Now recall the three essential characteristics for defining responsibility (i.e., the temporal ordering of actions, free choices, and the observer's cognizance) in Section 3.1. It is obvious that the responsibility abstraction α_R has taken both the temporal ordering of actions and the observer's cognizance into account. As for the free choices, from Theorem 1 it is easy to find that, if the transition τ_{R} is completely determined by its history trace σ_{H} and is not free to make choices (i.e., $\forall \sigma_{\mathrm{H}}\tau_{\mathrm{R}}, \sigma_{\mathrm{H}}\tau'_{\mathrm{R}} \in [\![\mathrm{P}]\!]^{\mathrm{Pref}}.\ \tau_{\mathrm{R}} = \tau'_{\mathrm{R}}$), then τ_{R} cannot be responsible for any behavior in the trace $\sigma_{\mathrm{H}}\tau_{\mathrm{R}}\sigma_{\mathrm{F}}$.

3.2.5 Concrete Responsibility Analysis . To sum up, the responsibility analysis in the concrete typically consists of four steps: (1) collect the system's trace semantics $[\![\mathrm{P}]\!]^{\mathsf{Max}}$ (in Section 2.2 and 3.2.1); (2) build the lattice of system behaviors of interest $\mathcal{L}^{\mathsf{Max}}$ (in Section 3.2.2); (3) derive an inquiry function $\mathbb{I}$ from $\mathcal{L}^{\mathsf{Max}}$, define a cognizance function $\mathbb{C}$ for each observer, and create the corresponding observation function $\mathbb{O}$ (in Section 3.2.3); (4) specify the behavior $\mathcal{B} \in \mathcal{L}^{\mathsf{Max}}$ of interest and the analyzed traces $\mathcal{T} \in \wp([\![\mathrm{P}]\!]^{\mathsf{Max}})$, and apply the responsibility abstraction $\alpha_R([\![\mathrm{P}]\!]^{\mathsf{Max}}, \mathcal{L}^{\mathsf{Max}}, \mathbb{C}, \mathcal{B}, \mathcal{T})$ to get the analysis result (in Section 3.2.4). Hence, the responsibility analysis is essentially an abstract interpretation of the program trace semantics.

Moreover, in definition (5) of responsibility, the sets of traces involved in the trace semantics, system behaviors, and the cognizance function are concrete. For the simple access control program, such concrete traces are explicitly displayed for the sake of clarity. However, they are uncomputable in general, and we cannot require the user to directly provide concrete traces in the implementation of responsibility analysis. To solve this problem, an abstract responsibility analysis that can soundly over-approximate the concrete responsibility analysis results is proposed in Section 7.

Example 10 (Access Control, Continued). Using the observation functions in Example 9, the abstraction α_R can analyze the responsibility of any behavior $\mathcal{B}$ in the specified set $\mathcal{T}$ of traces. If we intend to analyze "Access Failure" in every possible execution, then $\mathcal{B}$ is set as $\mathtt{AF}$, and $\mathcal{T}$ includes all valid maximal traces, i.e., $\mathcal{T} = [\![\mathrm{P}]\!]^{\mathsf{Max}}$. Thus, by the responsibility abstraction $\alpha_R([\![\mathrm{P}]\!]^{\mathsf{Max}}, \mathcal{L}^{\mathsf{Max}}, \mathbb{C}, \mathtt{AF}, [\![\mathrm{P}]\!]^{\mathsf{Max}})$, we could compute the responsibility analysis result, which is essential the same as described in Example 3 and omitted here.

In addition, if we would like to analyze the responsibility of "Read and Write access is granted," then the behavior of interest $\mathcal{B}$ shall be replaced by $\mathtt{RW}$ instead, and we can get the following result. To the cognizance of an omniscient observer, in every execution that both two admins input 1 or 2, the input from system settings (i.e., $typ := [1; 2]$) is responsible for $\mathtt{RW}$. Meanwhile, to the cognizance of the non-omniscient observer who is unaware of the input from 1st admin or the value of apv, no one would be found responsible for $\mathtt{RW}$, because whether the write access is granted or not is always uncertain due to the unknown input from 1st admin.

3.3 Applications of Responsibility Analysis

Responsibility is a broad concept, and our definition of responsibility based on the abstraction of trace semantics is applicable in various scientific fields. We have examined every example supplied in actual cause [34, 35] and found that our definition of responsibility can handle them well, in which events like "drop a lit match in the forest" or "throw a rock at the bottle" are treated as actions along the trace.

In this section, we focus on analyzing computer programs and illustrate the application of responsibility analysis by three more examples: (i) the "negative balance" problem of a withdrawal transaction, which can be equivalently viewed as the "buffer overflow" problem; (ii) a program with "division by zero" error, which can be also interpreted as a scenario of "login attack"; and (iii) the "information leakage" problem. It is worth noting that, for any behavior $\mathcal{B}$ of interest, our responsibility analysis is designed to analyze the programs where the behavior $\mathcal{B}$ does not always occur, i.e., $\mathcal{B} \subsetneq [\![\mathrm{P}]\!]^{\mathsf{Max}}$. Yet, for the programs where every trace has the behavior $\mathcal{B}$, we need to admit that the responsibility analysis cannot identify any responsible entity, unless "launching the program" is treated as a separate action and it would be found responsible for $\mathcal{B}$.

3.3.1 Example of Negative Balance/Buffer Overflow. Consider a withdrawal transaction scenario, which is simplified into a program with only three lines of code as in Figure 6 for the sake of clarity. At point ℓ_1, we read the bank account balance before the withdrawal transaction, which

```
ℓ1 :  balance := [0; INT_MAX];        //Account balance before the transaction
ℓ2 :  num := [1; INT_MAX];            //Withdrawal amount
ℓ3 :  balance := balance − num;       //Account balance after the transaction
ℓ4 :                                  //Error if balance < 0
```

Fig. 6. The withdrawal transaction program with negative balance problem.

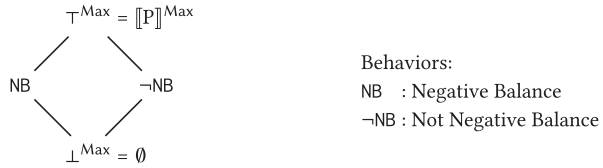

Fig. 7. Lattice of system behaviors regarding negative balance.

is assumed to be a positive integer or zero; in practice, this read action is typically implemented by a query in the database system. At point ℓ_2, the user inputs the withdrawal amount, which is assumed to be a strictly positive integer. At point ℓ_3, we update the bank account balance after the withdrawal transaction by subtracting *num* from *balance*. When the withdrawal transaction completes at point ℓ_4, if the account balance is negative (i.e., $balance < 0$), then it is an error and we would like to detect the responsible entity for it.

It is not hard to see that, the "negative balance" problem can be transformed into an equivalent "buffer overflow" problem, where a memory of size *balance* is allocated, the index at $num - 1$ is visited, and a buffer overflow error occurs when $balance \leq num - 1$ holds. Although this problem has been well studied, it suffices to demonstrate the advantages of responsibility analysis over dependency/causality analysis.

In this example, we consider only the cognizance of the omniscient observer, and the responsibility analysis consists of four steps, as discussed in Section 3.2.5:

(1) Collect the trace semantics $[\![\mathrm{P}]\!]^{\mathrm{Max}}$. In the withdrawal transaction program, each maximal trace is of length 4, and $[\![\mathrm{P}]\!]^{\mathrm{Max}} = \{\langle \ell_1, \rho_1\rangle\langle \ell_2, \rho_2\rangle\langle \ell_3, \rho_3\rangle\langle \ell_4, \rho_4\rangle \mid (\rho_1 \in \mathbb{M}) \wedge (\rho_2 = \rho_1[balance \mapsto v] \wedge v \in [0; \mathrm{INT\_MAX}]) \wedge (\rho_3 = \rho_2[num \mapsto v'] \wedge v' \in [1; \mathrm{INT\_MAX}]) \wedge (\rho_4 = \rho_3[balance \mapsto \rho_3(balance) - \rho_3(num)])\}$ consists of a very large number of traces. For example, $\langle \ell_1, \rho_1\rangle\langle \ell_2, \rho_2 = \rho_1[balance \mapsto 0]\rangle\langle \ell_3, \rho_3 = \rho_2[num \mapsto 1]\rangle\langle \ell_4, \rho_4 = \rho_3[balance \mapsto -1]\rangle$ denotes a maximal trace such that the balance before the transaction is 0 and the withdrawal amount is 1; and $\langle \ell_1, \rho_1\rangle\langle \ell_2, \rho_2 = \rho_1[balance \mapsto 5]\rangle\langle \ell_3, \rho_3 = \rho_2[num \mapsto 9]\rangle\langle \ell_4, \rho_4 = \rho_3[balance \mapsto -4]\rangle$ denotes a maximal trace such that the balance before the transaction is 5 and the withdrawal amount is 9. Both the above two traces have the negative balance problem.

(2) Build the lattice of system behaviors of interest. Since "negative balance" is the only behavior that we are interested here, we can build the lattice $\mathcal{L}^{\mathrm{Max}}$ with only four elements, as in Figure 7, where NB is the set of valid maximal traces where the value of *balance* is negative at point ℓ_4 (i.e., $\mathsf{NB} = \{\sigma \in [\![\mathrm{P}]\!]^{\mathrm{Max}} \mid \exists \rho \in \mathbb{M}.\ \sigma_{[3]} = \langle \ell_4, \rho\rangle \wedge \rho(balance) < 0\}$), and ¬NB is its complement (i.e., $\neg\mathsf{NB} = [\![\mathrm{P}]\!]^{\mathrm{Max}} \backslash \mathsf{NB} = \{\sigma \in [\![\mathrm{P}]\!]^{\mathrm{Max}} \mid \exists \rho \in \mathbb{M}.\ \sigma_{[3]} = \langle \ell_4, \rho\rangle \wedge \rho(balance) \geq 0\}$).

(3) Create the observation function. Using the omniscient observer's cognizance $\mathbb{C}_o$ such that $\mathbb{C}_o(\sigma) = \{\sigma\}$, the observation function $\mathbb{O}$ can be easily derived from the lattice $\mathcal{L}^{\mathrm{Max}}$ of system behaviors, such that:
   — $\mathbb{O}([\![\mathrm{P}]\!]^{\mathrm{Max}}, \mathcal{L}^{\mathrm{Max}}, \mathbb{C}_o, \langle \ell_1, \rho_1\rangle) = \top^{\mathrm{Max}}$, i.e., at the initial point ℓ_1, only the top behavior $\top^{\mathrm{Max}}$ can be guaranteed.

ℓ_1 : $pwd := [1; \text{INT\_MAX}]$; //The password stored in the system
ℓ_2 : $i1 := [1; \text{INT\_MAX}]$; //The first input from attacker
ℓ_3 : $i2 := [\text{INT\_MIN}; 0]$; //The second input from attacker
ℓ_4 : $res := (pwd - i1) \times i2$; //The attack result: 0 - success, otherwise - failure
ℓ_5 : $check := 1/res$; //Error if $res = 0$
ℓ_6 :

Fig. 8. The program with division by zero/login attack problem.

- $\mathbb{O}([\![\mathrm{P}]\!]^{\mathsf{Max}}, \mathcal{L}^{\mathsf{Max}}, \mathbb{C}_o, \langle \ell_1,\ \rho_1 \rangle \langle \ell_2,\ \rho_2 = \rho_1[balance \mapsto 0] \rangle) = \mathtt{NB}$, i.e., if the balance before the transaction is 0, the occurrence of "negative balance" is guaranteed even before the withdrawal amount *num* is entered;
- $\mathbb{O}([\![\mathrm{P}]\!]^{\mathsf{Max}}, \mathcal{L}^{\mathsf{Max}}, \mathbb{C}_o, \langle \ell_1,\ \rho_1 \rangle \langle \ell_2,\ \rho_2 = \rho_1[balance \mapsto v] \rangle) = \top^{\mathsf{Max}}$ where $v > 0$, i.e., if the balance before the transaction is strictly positive, whether "negative balance" occurs or not is uncertain at point ℓ_2;
- $\mathbb{O}([\![\mathrm{P}]\!]^{\mathsf{Max}}, \mathcal{L}^{\mathsf{Max}}, \mathbb{C}_o, \langle \ell_1,\ \rho_1 \rangle \langle \ell_2,\ \rho_2 = \rho_1[balance \mapsto v] \rangle \langle \ell_3,\ \rho_3 = \rho_2[num \mapsto v'] \rangle) = \mathtt{NB}$ where $v > 0$ and $v' > v$, i.e., "negative balance" is guaranteed to occur immediately after the value of *num* is set strictly greater than *balance*;
- $\mathbb{O}([\![\mathrm{P}]\!]^{\mathsf{Max}}, \mathcal{L}^{\mathsf{Max}}, \mathbb{C}_o, \langle \ell_1,\ \rho_1 \rangle \langle \ell_2,\ \rho_2 = \rho_1[balance \mapsto v] \rangle \langle \ell_3,\ \rho_3 = \rho_2[num \mapsto v'] \rangle) = \neg\mathtt{NB}$ where $v > 0$ and $v' \leq v$, i.e., "negative balance" is guaranteed not to occur immediately after the value of *num* is set less than or equal to *balance*.

(4) Last, by setting the behavior $\mathcal{B} = \mathtt{NB}$ and the analyzed traces $\mathcal{T} = [\![\mathrm{P}]\!]^{\mathsf{Max}}$, the abstraction $\alpha_R([\![\mathrm{P}]\!]^{\mathsf{Max}}, \mathcal{L}^{\mathsf{Max}}, \mathbb{C}_o, \mathcal{B}, \mathcal{T})$ can find: If the balance before the transaction is 0 (e.g., $\langle \ell_1,\ \rho_1 \rangle \langle \ell_2,\ \rho_2 = \rho_1[balance \mapsto 0] \rangle \langle \ell_3,\ \rho_3 = \rho_2[num \mapsto 1] \rangle \langle \ell_4,\ \rho_4 = \rho_3[balance \mapsto -1] \rangle$), then no matter what the withdrawal amount is, the action $balance := [0; \text{INT\_MAX}]$ is responsible for "negative balance"; otherwise, if the balance before the transaction is strictly positive (e.g., $\langle \ell_1,\ \rho_1 \rangle \langle \ell_2,\ \rho_2 = \rho_1[balance \mapsto 5] \rangle \langle \ell_3,\ \rho_3 = \rho_2[num \mapsto 9] \rangle \langle \ell_4,\ \rho_4 = \rho_3[balance \mapsto -4] \rangle$), then the action $num := [1; \text{INT\_MAX}]$ shall take the responsibility.

Using the responsibility analysis result above, we could prevent the "negative balance" behavior by configuring the program (e.g., a test guard for the withdrawal operation), such that the balance before the withdrawal transaction is ensured to be strictly positive, and the withdrawal amount is ensured to be less than or equal to the balance.

3.3.2 Example of Division by Zero/Login Attack. Consider the program in Figure 8, in which there is obviously a potential division-by-zero error at point ℓ_5. Alternatively, the division-by-zero error can be interpreted as a behavior of "login attack success" by interpreting the program as a simplified login scenario of some complex system for a malicious user (e.g., an attacker attempts to login the account of a normal user in a website).

More precisely, at point ℓ_1, the program reads the real password of a normal user that is stored in the system and saves it in the variable pwd. Typically, in practice, a password of valid format consists of letters/numbers and meets the requirement of length, while a password of invalid format contains special characters or does not meet the length requirement. For the sake of simplicity, it is assumed that the passwords of valid format are represented by positive integers in this simplified program, while the passwords of invalid format are represented by zero or negative integers. At point ℓ_2, the input $i1$ is used to mimic the attacker's attempt of entering a guessed password of valid format (i.e., a positive integer). If the guessed password coincides with the real password pwd, then the attacker succeeds to log into the normal user's account. Further, at point ℓ_3, the input $i2$ is used

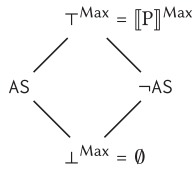

Fig. 9. Lattice of system behaviors regarding login attack.

to mimic the attacker's attempt of entering a password that is of invalid format (i.e., zero or a negative integer). Specially, the value zero represents a piece of malicious code (e.g., SQL statements) that could bypass the authentication. Thus, the attacker succeeds to log into the normal user's account if the guessed password coincides with the real password (i.e., $pwd = i1$) or the attacker injects malicious code (i.e., $i2 = 0$). Such an attack is represented by the computation of *res* at point ℓ_4, and the division by zero error at point ℓ_5 represents the behavior of login attack success.

Now the question is: Which action is responsible for "login attack success" (or, say, "division by zero")? In the following, we illustrate the four steps of responsibility analysis for "login attack success." Different from the analysis of "negative balance" in Section 3.3.1, in this example, we shall take the cognizance of an non-omniscient observer.

(1) Collect the trace semantics $[\![\text{P}]\!]^{\text{Max}}$. For the program in Figure 8, each maximal trace is of length 6, and $[\![\text{P}]\!]^{\text{Max}} = \{\langle \ell_1, \rho_1\rangle\langle \ell_2, \rho_2\rangle\langle \ell_3, \rho_3\rangle\langle \ell_4, \rho_4\rangle\langle \ell_5, \rho_5\rangle\langle \ell_6, \rho_6\rangle \mid (\rho_1 \in \mathbb{M}) \wedge (\rho_2 = \rho_1[pwd \mapsto v] \wedge v \in [1; \text{INT\_MAX}]) \wedge (\rho_3 = \rho_2[i1 \mapsto v'] \wedge v' \in [1; \text{INT\_MAX}]) \wedge (\rho_4 = \rho_3[i1 \mapsto v''] \wedge v'' \in [\text{INT\_MIN}; 0]) \wedge (\rho_5 = \rho_4[res \mapsto (\rho_4(pwd) - \rho_4(i1)) * \rho_4(i2)]) \wedge (\rho_6 = \rho_5[check \mapsto 1\backslash\rho_5(res)])\}$ consists of a large number of traces. For example, $\langle \ell_1, \rho_1\rangle\langle \ell_2, \rho_2 = \rho_1[pwd \mapsto 911]\rangle\langle \ell_3, \rho_3 = \rho_2[i1 \mapsto 911]\rangle\langle \ell_4, \rho_4 = \rho_3[i2 \mapsto -5]\rangle\langle \ell_5, \rho_5 = \rho_4[res \mapsto 0]\rangle\langle \ell_6, \omega\rangle$ denotes a maximal trace such that the guessed password ($i1$) coincides with the real password (pwd), and the execution ends with an error state representing "login attack success"; and $\langle \ell_1, \rho_1\rangle\langle \ell_2, \rho_2 = \rho_1[pwd \mapsto 911]\rangle\langle \ell_3, \rho_3 = \rho_2[i1 \mapsto 123]\rangle\langle \ell_4, \rho_4 = \rho_3[i2 \mapsto 0]\rangle\langle \ell_5, \rho_5 = \rho_4[res \mapsto 0]\rangle\langle \ell_6, \omega\rangle$ denotes a maximal trace such that the attacker enters a piece of malicious code that bypasses the authentication (i.e., $i2 = 0$). Both the above two traces have the behavior of "login attack success."

(2) Build the lattice of system behaviors of interest. Here, "login attack success" is the only behavior that we are interested in, and the corresponding lattice $\mathcal{L}^{\text{Max}}$ consists of only four elements as in Figure 9, where AS (login Attack Success) is the set of valid maximal traces where the value of *res* is zero at point ℓ_5 (i.e., $\text{AS} = \{\sigma \in [\![\text{P}]\!]^{\text{Max}} \mid \exists \rho \in \mathbb{M}.\ \sigma_{[4]} = \langle \ell_5, \rho\rangle \wedge \rho(res) = 0\}$), and ¬AS (login Attack Failure) is its complement (i.e., $\neg\text{AS} = [\![\text{P}]\!]^{\text{Max}} \backslash \text{AS} = \{\sigma \in [\![\text{P}]\!]^{\text{Max}} \mid \exists \rho \in \mathbb{M}.\ \sigma_{[4]} = \langle \ell_4, \rho\rangle \wedge \rho(res) \neq 0\}$).

(3) Create the observation function. In this case, it is intuitive to adopt the cognizance of the attacker, and it is assumed that the attacker does not know the real password of the normal user (otherwise, there is no way to prevent the login attack). Hence, a non-omniscient cognizance shall be designed such that it cannot distinguish the value of pwd, e.g., $\langle \ell_1, \rho_1\rangle\langle \ell_2, \rho_2 = \rho_1[pwd \mapsto 123]\rangle \in \mathbb{C}(\langle \ell_1, \rho_1\rangle\langle \ell_2, \rho_2 = \rho_1[pwd \mapsto 911]\rangle)$ denotes that the attacker does not know whether the real password is 123 or 911. Then, the observation function $\mathbb{O}$ can be derived from the lattice $\mathcal{L}^{\text{Max}}$ of system behaviors and the cognizance function $\mathbb{C}$, such that:
— $\mathbb{O}([\![\text{P}]\!]^{\text{Max}}, \mathcal{L}^{\text{Max}}, \mathbb{C}, \varepsilon) = \mathbb{O}([\![\text{P}]\!]^{\text{Max}}, \mathcal{L}^{\text{Max}}, \mathbb{C}, \langle \ell_1, \rho_1\rangle) = \mathbb{O}([\![\text{P}]\!]^{\text{Max}}, \mathcal{L}^{\text{Max}}, \mathbb{C}, \langle \ell_1, \rho_1\rangle\langle \ell_2, \rho_2 = \rho_1[pwd \mapsto 911]\rangle) = \top^{\text{Max}}$, i.e., before the attacker takes any action at point ℓ_2, only the top behavior $\top^{\text{Max}}$ can be guaranteed.

— $\mathbb{O}([\![\mathrm{P}]\!]^{\mathsf{Max}}, \mathcal{L}^{\mathsf{Max}}, \mathbb{C}, \langle \ell_1,\ \rho_1\rangle\langle \ell_2,\ \rho_2 = \rho_1[\mathit{pwd} \mapsto 911]\rangle\langle \ell_3,\ \rho_3 = \rho_2[i1 \mapsto 911]\rangle) = \mathbb{O}([\![\mathrm{P}]\!]^{\mathsf{Max}}, \mathcal{L}^{\mathsf{Max}}, \mathbb{C}, \langle \ell_1,\ \rho_1\rangle\langle \ell_2,\ \rho_2 = \rho_1[\mathit{pwd} \mapsto 911]\rangle\langle \ell_3,\ \rho_3 = \rho_2[i1 \mapsto 123]\rangle) = \top^{\mathsf{Max}}$, i.e., after the attacker enters the guessed password, no matter the guessed password coincides with the real password or not, only the top behavior $\top^{\mathsf{Max}}$ can be guaranteed to the cognizance of the attacker. The reason is that the attacker does not know the value of pwd, thus cannot ensure her/his guessed password is the same as the real password. More formally, $\mathbb{O}([\![\mathrm{P}]\!]^{\mathsf{Max}}, \mathcal{L}^{\mathsf{Max}}, \mathbb{C}, \langle \ell_1,\ \rho_1\rangle\langle \ell_2,\ \rho_2 = \rho_1[\mathit{pwd} \mapsto 911]\rangle\langle \ell_3,\ \rho_3 = \rho_2[i1 \mapsto 911]\rangle) = \mathbb{I}([\![\mathrm{P}]\!]^{\mathsf{Max}}, \mathcal{L}^{\mathsf{Max}}, \langle \ell_2,\ \rho_2 = \rho_1[\mathit{pwd} \mapsto 911]\rangle\langle \ell_3,\ \rho_3 = \rho_2[i1 \mapsto 911]\rangle) \cup \mathbb{I}([\![\mathrm{P}]\!]^{\mathsf{Max}}, \mathcal{L}^{\mathsf{Max}}, \langle \ell_2,\ \rho_2 = \rho_1[\mathit{pwd} \mapsto 123]\rangle\langle \ell_3,\ \rho_3 = \rho_2[i1 \mapsto 911]\rangle) \cup \ldots = \mathsf{AS} \cup \top^{\mathsf{Max}} = \top^{\mathsf{Max}}$.

— $\mathbb{O}([\![\mathrm{P}]\!]^{\mathsf{Max}}, \mathcal{L}^{\mathsf{Max}}, \mathbb{C}, \langle \ell_1,\ \rho_1\rangle\langle \ell_2,\ \rho_2 = \rho_1[\mathit{pwd} \mapsto 911]\rangle\langle \ell_3,\ \rho_3 = \rho_2[i1 \mapsto 911]\rangle\langle \ell_4,\ \rho_4 = \rho_3[i2 \mapsto -5]\rangle) = \top^{\mathsf{Max}}$, i.e., if the second input $i2$ from the attacker is not zero, then to the cognizance of the attacker, the behavior of login attack success cannot be guaranteed, even if she/he guesses the correct password in reality.

— $\mathbb{O}([\![\mathrm{P}]\!]^{\mathsf{Max}}, \mathcal{L}^{\mathsf{Max}}, \mathbb{C}, \langle \ell_1,\ \rho_1\rangle\langle \ell_2,\ \rho_2 = \rho_1[\mathit{pwd} \mapsto 911]\rangle\langle \ell_3,\ \rho_3 = \rho_2[i1 \mapsto 123]\rangle\langle \ell_4,\ \rho_4 = \rho_3[i2 \mapsto 0]\rangle) = \mathsf{AS}$, i.e., only after the attacker enters zero as the second input (i.e., succeeds to inject malicious code), login attack success is guaranteed to the cognizance of the attacker.

(4) Last, by setting the behavior $\mathcal{B} = \mathsf{AS}$ and the analyzed traces $\mathcal{T} = [\![\mathrm{P}]\!]^{\mathsf{Max}}$, the abstraction $\alpha_R([\![\mathrm{P}]\!]^{\mathsf{Max}}, \mathcal{L}^{\mathsf{Max}}, \mathbb{C}, \mathcal{B}, \mathcal{T})$ can find: Only the action $i2 := [\mathrm{INT\_MIN}; 0]$ representing entering passwords of invalid format is responsible for the behavior "login attack success," and the action $i1 := [1; \mathrm{INT\_MAX}]$ representing entering passwords of valid format is not responsible.

Meanwhile, if we take $\neg\mathsf{AS}$ as the behavior of interest $\mathcal{B}$, then the corresponding responsibility analysis would find that there is no responsible action for $\neg\mathsf{AS}$. That is to say, we cannot take any action to prevent the attacker from succeeding to login the system, since there is always a possibility (although it is low) that the attacker succeeds to guess the correct password.

Using the responsibility analysis result above, we could configure the program to exclude the value of zero from the range of second input (or, say, we forbid the attacker to enter malicious code like SQL statements), so the attacker can never ensure to login the account of a normal user.

3.3.3 Example of Information Leakage. From the example of access control in Section 3.2 as well as the two examples in Sections 3.3.1 and 3.3.2, it is not hard to see that the responsibility analysis process is essentially the same for all behaviors, and the only significant distinction among these examples is on defining the behaviors of interest and the cognizance function.

Non-interference. In this section, we consider the responsibility analysis of the behavior "information leakage," which is represented by the notion of *non-interference* [31]. More precisely, the inputs and outputs in the analyzed program are classified as either *Low* (public, low sensitivity) or *High* (private, high sensitivity). For any valid maximal trace $\sigma \in [\![\mathrm{P}]\!]^{\mathsf{Max}}$, if there is another valid maximal trace $\sigma' \in [\![\mathrm{P}]\!]^{\mathsf{Max}}$ such that they have the same low inputs but different low outputs, then the trace σ is said to leak private information, and the analyzed program is possibly insecure. If there is no valid maximal trace in the analyzed program that leaks private information (i.e., every two valid maximal traces with the same low inputs must have the same low outputs, regardless of the high inputs), then the program has the "non-interference" property, hence it is secure.

Here, we take the simple program in Figure 10 as an example, which does not have the desired "non-interference" property. At point ℓ_1, a high (private) input of positive integer is read and saved in the variable $\mathit{input\_h}$. Similarly, at point ℓ_2, a low (public) input is stored in the variable $\mathit{input\_l}$,

```
ℓ1 :  input_ℋ := [1; INT_MAX];                          //High (private) input
ℓ2 :  input_ℒ := [0; 1];                                //Low (public) input
ℓ3 :  output_ℒ := [0; 0];                               //Initialization of low (public) output
ℓ4 :  while (input_ℒ > 0 ∧ input_ℋ > 0) {
ℓ5 :        output_ℒ := output_ℒ + 1;
ℓ6 :        input_ℋ := input_ℋ − 1; }
ℓ7 :  //Here we output output_ℒ in public
```

Fig. 10. The program with potential information leakage.

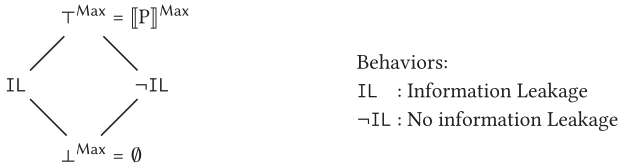

Fig. 11. Lattice of behaviors regarding information leakage.

which is assumed to be either zero or one. At point ℓ_3, a variable *output_ℒ* is initialized as zero. After the execution of the while loop between points ℓ_4 and ℓ_7, the value of *output_ℒ* is output as low (public). It is not hard to find that, although there is no direct data flow from *input_ℋ* to *output_ℒ* (e.g., an assignment *output_ℒ* := *input_ℋ*) in the program, the low output *output_ℒ* at point ℓ_7 is equal to the high input, if the value of low input *input_ℒ* is 1 at point ℓ_3. Therefore, there is a potential behavior of information leakage from *input_ℋ* to *output_ℒ* in this program.

Similar to previous examples, the responsibility analysis of information leakage consists of four steps, and we adopt the cognizance of omniscient observer.

(1) Collect the trace semantics $[\![\mathrm{P}]\!]^{\mathsf{Max}}$. For the program in Figure 10, $[\![\mathrm{P}]\!]^{\mathsf{Max}}$ consists of $2 \times$ INT_MAX maximal traces, and here we take two of them as examples:
(i) $\sigma = \langle \ell_1, \rho_1\rangle\langle \ell_2, \rho_2 = \rho_1[input\_\mathcal{H} \mapsto 2]\rangle\langle \ell_3, \rho_3 = \rho_2[input\_\mathcal{L} \mapsto 1]\rangle\langle \ell_4, \rho_4 = \rho_3[output\_\mathcal{L} \mapsto 0]\rangle\langle \ell_5, \rho_5 = \rho_4\rangle\langle \ell_6, \rho_6 = \rho_5[output\_\mathcal{L} \mapsto 1]\rangle\langle \ell_4, \rho'_4 = \rho_6[input\_\mathcal{H} \mapsto 1]\rangle\langle \ell_5, \rho'_5 = \rho'_4\rangle\langle \ell_6, \rho'_6 = \rho'_5[output\_\mathcal{L} \mapsto 2]\rangle\langle \ell_4, \rho''_4 = \rho'_6[input\_\mathcal{H} \mapsto 0]\rangle\langle \ell_7, \rho_7 = \rho''_4\rangle$. In this trace, the high input is 2, and the low input is 1. After two iterations of the while loop, the value of *output_ℒ* is assigned to 2, which is equal to the high input.
(ii) $\sigma' = \langle \ell_1, \rho_1\rangle\langle \ell_2, \rho_2 = \rho_1[input\_\mathcal{H} \mapsto 2]\rangle\langle \ell_3, \rho_3 = \rho_2[input\_\mathcal{L} \mapsto 0]\rangle\langle \ell_4, \rho_4 = \rho_3[output\_\mathcal{L} \mapsto 0]\rangle\langle \ell_7, \rho_7 = \rho_4\rangle$. Different from the previous trace σ, the low input in this trace is 0, such that the while loop is never entered, and the value of *output_ℒ* remains as 0 at point ℓ_7.

(2) Build the lattice of system behaviors of interest. In general, for the responsibility analysis of information leakage, the corresponding lattice $\mathcal{L}^{\mathsf{Max}}$ of system behaviors consists of four elements, as shown in Figure 11.
More specifically, the behavior of "Information Leakage" IL is represented as the set of valid maximal traces that leak private information, i.e., $\mathtt{IL} = \{\sigma \in [\![\mathrm{P}]\!]^{\mathsf{Max}} \mid \exists \sigma' \in [\![\mathrm{P}]\!]^{\mathsf{Max}}.\ \mathtt{low\_inputs}(\sigma) = \mathtt{low\_inputs}(\sigma') \wedge \mathtt{low\_outputs}(\sigma) \neq \mathtt{low\_outputs}(\sigma')\}$, where the functions low_inputs (respectively, low_outputs) collects the list of low inputs (respectively, low outputs) along the trace σ. In contrast, the behavior of "No information Leakage" ¬IL is the complement of IL, which is the set of valid maximal traces that do not leak private information, i.e., $\neg\mathtt{IL} = [\![\mathrm{P}]\!]^{\mathsf{Max}} \backslash \mathtt{IL} = \{\sigma \in [\![\mathrm{P}]\!]^{\mathsf{Max}} \mid \forall \sigma' \in [\![\mathrm{P}]\!]^{\mathsf{Max}}.\ \mathtt{low\_inputs}(\sigma) = \mathtt{low\_inputs}(\sigma') \Rightarrow \mathtt{low\_outputs}(\sigma) = \mathtt{low\_outputs}(\sigma')\}$.

For the program in Figure 10, $\mathtt{IL} = \{\sigma \in [\![\mathrm{P}]\!]^{\mathsf{Max}} \mid \exists \rho, \rho' \in \mathbb{M}.\ \sigma_{[1]} = \langle \ell_2,\ \rho\rangle \wedge \sigma_{[|\sigma|-1]} = \langle \ell_7,\ \rho'\rangle \wedge \rho(\mathit{input\_h}) = \rho'(\mathit{output\_l})\} = \{\sigma \in [\![\mathrm{P}]\!]^{\mathsf{Max}} \mid \exists \rho.\ \sigma_{[2]} = \langle \ell_3,\ \rho\rangle \wedge \rho(\mathit{input\_l}) = 1\}$ (i.e., IL is the set of valid maximal traces where the value of *output_l* at ℓ_7 is equal to the high input, which is also the set of valid maximal traces where the value of *input_l* is 1 at ℓ_3); $\neg\mathtt{IL} = \{\sigma \in [\![\mathrm{P}]\!]^{\mathsf{Max}} \mid \exists \rho \in \mathbb{M}.\ \sigma_{[|\sigma|-1]} = \langle \ell_7,\ \rho\rangle \wedge \rho(\mathit{output\_l}) = 0\} = \{\sigma \in [\![\mathrm{P}]\!]^{\mathsf{Max}} \mid \exists \rho.\ \sigma_{[2]} = \langle \ell_3,\ \rho\rangle \wedge \rho(\mathit{input\_l}) = 0\}$ (i.e., ¬IL is the set of valid maximal traces where the value of *output_l* is 0 at ℓ_7, which is also the set of valid maximal traces where the value of *input_l* is 0 at ℓ_3).

(3) Create the observation function. Using the omniscient observer's cognizance $\mathbb{C}_o$, the observation function $\mathbb{O}$ can be easily derived from $\mathcal{L}^{\mathsf{Max}}$ such that:
   — $\mathbb{O}([\![\mathrm{P}]\!]^{\mathsf{Max}}, \mathcal{L}^{\mathsf{Max}}, \mathbb{C}_o, \langle \ell_1,\ \rho_1\rangle) = \top^{\mathsf{Max}}$, i.e., at the initial point ℓ_1, it is uncertain if the information leakage occurs or not, hence only $\top^{\mathsf{Max}}$ is guaranteed.
   — $\mathbb{O}([\![\mathrm{P}]\!]^{\mathsf{Max}}, \mathcal{L}^{\mathsf{Max}}, \mathbb{C}_o, \langle \ell_1,\ \rho_1\rangle\langle \ell_2,\ \rho_2 = \rho_1[\mathit{input\_h} \mapsto v]\rangle) = \top^{\mathsf{Max}}$, i.e., after the high input *input_h* is entered, no matter what value it is, only the top behavior $\top^{\mathsf{Max}}$ can be guaranteed before the low input *input_l* is entered.
   — $\mathbb{O}([\![\mathrm{P}]\!]^{\mathsf{Max}}, \mathcal{L}^{\mathsf{Max}}, \mathbb{C}_o, \langle \ell_1,\ \rho_1\rangle\langle \ell_2,\ \rho_2 = \rho_1[\mathit{input\_h} \mapsto v]\rangle\rangle\langle \ell_3,\ \rho_3 = \rho_2[\mathit{input\_l} \mapsto 1]\rangle = \mathtt{IL}$, i.e., the behavior of information leakage is guaranteed to occur immediately after the low input *input_l* is set as 1.
   — $\mathbb{O}([\![\mathrm{P}]\!]^{\mathsf{Max}}, \mathcal{L}^{\mathsf{Max}}, \mathbb{C}_o, \langle \ell_1,\ \rho_1\rangle\langle \ell_2,\ \rho_2 = \rho_1[\mathit{input\_h} \mapsto v]\rangle\rangle\langle \ell_3,\ \rho_3 = \rho_2[\mathit{input\_l} \mapsto 0]\rangle = \neg\mathtt{IL}$, i.e., the behavior of information leakage is guaranteed not to occur immediately after the low input *input_l* is set as 0.

(4) Last, by setting the behavior $\mathcal{B} = \mathtt{IL}$ and the analyzed traces $\mathcal{T} = [\![\mathrm{P}]\!]^{\mathsf{Max}}$, the abstraction $\alpha_R([\![\mathrm{P}]\!]^{\mathsf{Max}}, \mathcal{L}^{\mathsf{Max}}, \mathbb{C}_o, \mathcal{B}, \mathcal{T})$ can find that only the action $\mathit{input\_l} := [0; 1]$ representing a low input is responsible for the information leakage, while the action $\mathit{input\_h} := [1; \mathrm{INT\_MAX}]$ representing a high input is not responsible.

After the responsibility analysis of information leakage completes, it is of interest to discuss the procedure of configuring the analyzed program, especially for the programs where the information leakage is acceptable or even desirable under certain circumstances. For instance, imagine a more complex analyzed program that is a social network, where every user can enter some public information (e.g., name, gender) as well as some private information (e.g., birth date, photos). If the private information of any user called A flows to another user called B (e.g., the user B accesses a photo uploaded by A), then it can be viewed as a behavior "information leakage" IL defined above, and we would like to analyze the corresponding responsibility. After the responsibility analysis is finished, if the responsible entity is determined as an action of A who is the owner of the private information (e.g., A sets her/his own photos public, or A adds B as a friend) or an action of the system administrator, then this information leakage is safe and the corresponding responsible actions can be kept. In contrast, if the responsible entity is determined as an action of B or other unauthorized users (e.g., B exploits a bug of the system such that she/he can access the private information of any other user without authorization), then such an information leakage behavior is undesired, and the corresponding responsible actions shall be eliminated to fix the system.

4 FORWARD REACHABILITY AND BACKWARD ACCESSIBILITY ANALYSIS

Abstract interpretation [18, 20, 21] is a mathematical theory to reason on the executions of computer programs. It formalizes formal methods and allows to discuss the guarantees they provide such as soundness (the conclusions about programs are always correct under suitable explicitly stated hypotheses), completeness (all true facts are provable), or incompleteness (showing the limits of applicability of the formal method).

This section presents some notations and techniques in the abstract interpretation framework, which will be referenced later in designing the abstract responsibility analysis. To be more precise, Section 4.1 introduces the abstract domain of environments and invariants. In Section 4.2, we define the classic forward (possible success) reachability semantics of a program as an abstraction of the trace semantics and sketch the design of an over-approximating abstract forward reachability analysis, which can automatically infer program invariants. In Section 4.3, the backward impossible failure accessibility semantics is defined as the adjoint of forward reachability semantics, which specifies the sufficient precondition for a given postcondition to hold. Compared with the classic forward reachability analysis, the abstract backward impossible failure accessibility analysis has not been well studied yet, and there are few literature on this topic. We summarize the under-approximating abstract backward analysis proposed by Miné [60, 61] and propose a similar over-approximating abstract backward analysis, both of which will be used to determine responsibility in the abstract.

4.1 Abstract Domains

The concrete trace semantics of transition systems introduced in Section 2.2 is not computable in general, thus we propose to abstract sets of concrete traces into invariants. To accomplish that, this section introduces the abstract environment domain, the concrete invariant domain, and the abstract invariant domain.

4.1.1 Abstract Environment Domain. Let $\langle \mathcal{D}^\sharp_{\mathbb{M}}, \sqsubseteq^\sharp_{\mathbb{M}}, \bot^\sharp_{\mathbb{M}}, \top^\sharp_{\mathbb{M}}, \sqcup^\sharp_{\mathbb{M}}, \sqcap^\sharp_{\mathbb{M}} \rangle$ be an *abstract environment domain*, and $\gamma_{\mathbb{M}} \in \mathcal{D}^\sharp_{\mathbb{M}} \mapsto \wp(\mathbb{M})$ be the corresponding concretization function that associates each abstract element $\mathrm{M}^\sharp \in \mathcal{D}^\sharp_{\mathbb{M}}$ to the set of concrete environments it represents. In particular, $\mathcal{D}^\sharp_{\mathbb{M}}$ features an infimum $\bot^\sharp_{\mathbb{M}}$ and a supremum $\top^\sharp_{\mathbb{M}}$ such that $\gamma_{\mathbb{M}}(\bot^\sharp_{\mathbb{M}}) = \emptyset$ and $\gamma_{\mathbb{M}}(\top^\sharp_{\mathbb{M}}) = \mathbb{M}$, and an abstract join operator $\sqcup^\sharp_{\mathbb{M}}$ that soundly approximates the concrete join operator $\cup$ (more precisely, $\forall \mathrm{M}^\sharp, \mathrm{M}^{\sharp\prime} \in \mathcal{D}^\sharp_{\mathbb{M}}.\ \gamma_{\mathbb{M}}(\mathrm{M}^\sharp) \cup \gamma_{\mathbb{M}}(\mathrm{M}^{\sharp\prime}) \subseteq \gamma_{\mathbb{M}}(\mathrm{M}^\sharp \sqcup^\sharp_{\mathbb{M}} \mathrm{M}^{\sharp\prime})$).

This article focuses on the analysis of numerical programs and takes three popular abstract domains that can express constraints on program variables as examples. The *interval domain* introduced in Reference [19] bounds the value of numerical variables by minimal and maximal values between which all reachable values of a variable must stand, and each abstract element in this domain can be defined as a mapping from program variables to intervals (e.g., $\chi \in [l, h] \wedge y \in [l', h']$). It is a simple but useful domain, and it has been applied not only to prove the absence of integers or array index overflows but also to detect unseen inputs of neural networks [37]. However, the interval domain is not expressive enough to be useful for a relational reachability analysis, in which the constraints involving more than one variable are needed. One example of relational abstractions is the *polyhedra domain* introduced in Reference [24] that can express conjunctions of affine inequalities on variables. In this domain, an abstract element (i.e., polyhedron) is defined as a finite set of affine constraints of form $\vec{a} \cdot \vec{\chi} \geq b$ (e.g., $2 * \chi - 3 * y + 5 * z \geq 4$), where $\vec{\chi}$ denotes the vector of all variables, $\vec{a}$ denotes a vector of coefficients, and b denotes a constant. In addition, strict inequalities are supported in current polyhedron domain [5, 6, 41] . Another example is the *octagon domain* [56–58], which restricts the affine constraints used in the polyhedron domain to unit binary inequality constraints of form $\pm\chi_1 \pm \chi_2 \leq c$ (e.g., $x - y \leq 0$). The above three numerical domains are similar semantically in that they infer conjunctions of inequality constraints and represent convex sets, but they are based on different algorithms and achieve different tradeoffs between precision and efficiency. Operators in the interval domain have a linear cost in the number of variables, while octagon operators have a cubic cost. The cost of polyhedra is unbounded in theory (since it can construct arbitrarily many constraints), but it is at most exponential in practice [61, 62].

It is assumed that, for every concrete environment transfer function $\mathrm{F} \in \wp(\mathbb{M}) \mapsto \wp(\mathbb{M})$ specified for atomic actions in the program (e.g., $\tau\llbracket\ \chi := \mathit{expr}\ \rrbracket$ and $\tau\llbracket\ \mathit{bexpr}\ \rrbracket$ for the simple language described in Figure 3), the abstract environment domain $\mathcal{D}^\sharp_\mathbb{M}$ (e.g., interval/polyhedron/octagon) provides a sound abstract function $\mathrm{F}^\sharp \in \mathcal{D}^\sharp_\mathbb{M} \mapsto \mathcal{D}^\sharp_\mathbb{M}$, such that the soundness condition $\forall \mathrm{M}^\sharp \in \mathcal{D}^\sharp_\mathbb{M}.\ (\mathrm{F} \circ \gamma_\mathbb{M})(\mathrm{M}^\sharp) \subseteq (\gamma_\mathbb{M} \circ \mathrm{F}^\sharp)(\mathrm{M}^\sharp)$ holds.

In addition, it is worth noting that, in some abstract domains, we have an abstraction function $\alpha_\mathbb{M} \in \wp(\mathbb{M}) \mapsto \mathcal{D}^\sharp_\mathbb{M}$ such that $\alpha_\mathbb{M}$ and $\gamma_\mathbb{M}$ form a Galois connection $\langle \wp(\mathbb{M}),\ \subseteq\rangle \underset{\alpha_\mathbb{M}}{\overset{\gamma_\mathbb{M}}{\leftrightarrows}} \langle \mathcal{D}^\sharp_\mathbb{M},\ \sqsubseteq^\sharp_\mathbb{M}\rangle$. In this case, every concrete element $\mathrm{M} \subseteq \mathbb{M}$ (i.e., every set of concrete environments) has a best abstraction $\alpha_\mathbb{M}(\mathrm{M}) \in \mathcal{D}^\sharp_\mathbb{M}$, and every function $\mathrm{F} \in \wp(\mathbb{M}) \mapsto \wp(\mathbb{M})$ in the concrete domain has also a best abstraction $\alpha_\mathbb{M} \circ \mathrm{F} \circ \gamma_\mathbb{M} \in \mathcal{D}^\sharp_\mathbb{M} \mapsto \mathcal{D}^\sharp_\mathbb{M}$. Specifically, the interval and octagon domain have this desirable property, while the polyhedron domain does not.

Example 11 (Access Control, Continued). For the access control program in Figure 4, it is sufficient to use the interval domain as $\mathcal{D}^\sharp_\mathbb{M}$ to express environment properties such as "the access to o fails" (i.e., the value of *acs* is less than or equal to 0 at point ℓ_8), since no relational constraints on variables are required. To be more precise, the abstract environment element $\mathrm{M}^\sharp = \mathit{apv} \in [-\infty, \infty] \wedge \mathit{i1} \in [-\infty, \infty] \wedge \mathit{i2} \in [-\infty, \infty] \wedge \mathit{typ} \in [-\infty, \infty] \wedge \mathit{acs} \in [-\infty, 0]$ represents the set of environments $\gamma_\mathbb{M}(\mathrm{M}^\sharp) = \{\rho \in \mathbb{M} \mid \rho(\mathit{acs}) \leq 0\}$, in which the value of variable *acs* is less than or equal to 0 while the values of other variables are arbitrary. Similarly, an environment property "the access to o succeeds" (i.e., the value of *acs* is greater than or equal to 1 at point ℓ_8) can be over-approximated by $\mathrm{M}^{\sharp\prime} = \mathit{apv} \in [-\infty, \infty] \wedge \mathit{i1} \in [-\infty, \infty] \wedge \mathit{i2} \in [-\infty, \infty] \wedge \mathit{typ} \in [-\infty, \infty] \wedge \mathit{acs} \in [1, \infty]$.

4.1.2 Concrete Invariant Domain. For any set $\mathcal{T}$ of concrete traces (which can be either the program semantics or a trace property), we would like to abstract it into an invariant, which collects the set of environments for each program point that are visited by traces in $\mathcal{T}$. Hence, the *concrete invariant domain* $\mathcal{D}_\mathbb{I}$ is defined as $\langle \mathbb{L} \mapsto \wp(\mathbb{M}),\ \dot\subseteq\rangle$, and there exists a Galois connection between concrete traces and the concrete invariant domain $\mathcal{D}_\mathbb{I}$:

$$\langle \wp(\mathbb{S}^{*\infty}),\ \subseteq\rangle \underset{\alpha_\mathbb{IV}}{\overset{\gamma_\mathbb{IV}}{\leftrightarrows}} \langle \mathbb{L} \mapsto \wp(\mathbb{M}),\ \dot\subseteq\rangle, \tag{6}$$

where $\dot\subseteq$ is the pointwise inclusion relation, and $\alpha_\mathbb{IV}$ and $\gamma_\mathbb{IV}$ are defined as:

$$\begin{array}{rcll} \alpha_\mathbb{IV} & \in & \wp(\mathbb{S}^{*\infty}) \mapsto (\mathbb{L} \mapsto \wp(\mathbb{M})) & \text{concrete invariant abstraction} \\ \alpha_\mathbb{IV}(\mathcal{T})\ell & \triangleq & \{\rho \in \mathbb{M} \mid \exists \sigma \in \mathcal{T}.\ \langle \ell,\ \rho\rangle \in \sigma\} & \\ \gamma_\mathbb{IV} & \in & (\mathbb{L} \mapsto \wp(\mathbb{M})) \mapsto \wp(\mathbb{S}^{*\infty}) & \text{concrete invariant concretization} \\ \gamma_\mathbb{IV}(\mathsf{I}) & \triangleq & \{\sigma \in \mathbb{S}^{*\infty} \mid \forall \langle \ell,\ \rho\rangle \in \sigma.\ \rho \in \mathsf{I}(\ell)\}. & \end{array}$$

4.1.3 Abstract Invariant Domain. The concrete invariants introduced above can be further abstracted into abstract invariants, in which every set of concrete environments is represented by an abstract element in $\mathcal{D}^\sharp_\mathbb{M}$. Here, we define the abstract invariant domain $\mathcal{D}^\sharp_\mathbb{I}$ as $\langle \mathbb{L} \mapsto \mathcal{D}^\sharp_\mathbb{M},\ \dot\sqsubseteq^\sharp_\mathbb{M}\rangle$, where the program points are mapped to abstract elements in $\mathcal{D}^\sharp_\mathbb{M}$, and $\dot\sqsubseteq^\sharp_\mathbb{M}$ is the pointwise ordering induced by $\sqsubseteq^\sharp_\mathbb{M}$ (i.e., $\forall \mathsf{I}^\sharp, \mathsf{I}^{\sharp\prime} \in \mathcal{D}^\sharp_\mathbb{I}.\ \mathsf{I}^\sharp \dot\sqsubseteq^\sharp_\mathbb{M} \mathsf{I}^{\sharp\prime} \Leftrightarrow (\forall \ell \in \mathbb{L}.\ \mathsf{I}^\sharp(\ell) \sqsubseteq^\sharp_\mathbb{M} \mathsf{I}^{\sharp\prime}(\ell))$). The corresponding concretization function $\dot\gamma_\mathbb{M}$ to the concrete invariant domain is:

$$\begin{array}{rcll} \dot\gamma_\mathbb{M} & \in & \left(\mathbb{L} \mapsto \mathcal{D}^\sharp_\mathbb{M}\right) \mapsto (\mathbb{L} \mapsto \wp(\mathbb{M})) & \text{abstract invariant concretization} \\ \dot\gamma_\mathbb{M}(\mathsf{I}^\sharp)\ell & \triangleq & \gamma_\mathbb{M}(\mathsf{I}^\sharp(\ell)). & \end{array}$$

Similar to the abstract environment domain, $\mathcal{D}^\sharp_{\mathbb{I}}$ features an infimum $\bot^\sharp_{\mathbb{I}} \triangleq \lambda \ell \in \mathbb{L}.\ \bot^\sharp_{\mathbb{M}}$ and a supremum $\top^\sharp_{\mathbb{I}} \triangleq \lambda \ell \in \mathbb{L}.\ \top^\sharp_{\mathbb{M}}$ such that $\dot{\gamma}_{\mathbb{M}}(\bot^\sharp_{\mathbb{I}}) = \lambda \ell \in \mathbb{L}.\ \emptyset$ and $\dot{\gamma}_{\mathbb{M}}(\top^\sharp_{\mathbb{I}}) = \lambda \ell \in \mathbb{L}.\ \mathbb{M}$. When the environment abstraction $\alpha_{\mathbb{M}} \in \wp(\mathbb{M}) \mapsto \mathcal{D}^\sharp_{\mathbb{M}}$ does exist (e.g., in the interval or octagon domain), we can construct the corresponding pointwise abstraction function $\dot{\alpha}_{\mathbb{M}}$ for the abstract invariant domain:

$$\dot{\alpha}_{\mathbb{M}} \in (\mathbb{L} \mapsto \wp(\mathbb{M})) \mapsto \left(\mathbb{L} \mapsto \mathcal{D}^\sharp_{\mathbb{M}}\right) \qquad \text{abstract invariant abstraction}$$
$$\dot{\alpha}_{\mathbb{M}}(\mathsf{I})\ell \triangleq \alpha_{\mathbb{M}}(\mathsf{I}(\ell)).$$

Furthermore, we can build a combination of Galois connections:

$$\langle \wp(\mathbb{S}^{*\infty}),\ \subseteq \rangle \underset{\alpha_{\mathbb{TV}}}{\overset{\gamma_{\mathbb{TV}}}{\leftrightarrows}} \langle \mathbb{L} \mapsto \wp(\mathbb{M}),\ \dot{\subseteq} \rangle \underset{\dot{\alpha}_{\mathbb{M}}}{\overset{\dot{\gamma}_{\mathbb{M}}}{\leftrightarrows}} \langle \mathbb{L} \mapsto \mathcal{D}^\sharp_{\mathbb{M}},\ \dot{\sqsubseteq}^\sharp_{\mathbb{M}} \rangle,$$

such that an abstract invariant $\mathsf{I}^\sharp \in \mathbb{L} \mapsto \mathcal{D}^\sharp_{\mathbb{M}}$ over-approximates a set of concrete traces $\gamma_{\mathbb{TV}} \circ \dot{\gamma}_{\mathbb{M}}(\mathsf{I}^\sharp) = \{\sigma \in \mathbb{S}^{*\infty} \mid \forall \langle \ell,\ \rho \rangle \in \sigma.\ \rho \in \gamma_{\mathbb{M}}(\mathsf{I}^\sharp(\ell))\}$. Specially, the bottom $\bot^\sharp_{\mathbb{I}}$ represents the empty set of concrete traces ($\gamma_{\mathbb{TV}} \circ \dot{\gamma}_{\mathbb{M}}(\bot^\sharp_{\mathbb{I}}) = \emptyset$) and the top$\top^\sharp_{\mathbb{I}}$ represents the set of all possible traces ($\gamma_{\mathbb{TV}} \circ \dot{\gamma}_{\mathbb{M}}(\bot^\sharp_{\mathbb{I}}) = \mathbb{S}^{*\infty}$).

Example 12 (Access Control, Continued). For the access control program in Figure 4, its maximal trace semantics $[\![\mathrm{P}]\!]^{\mathsf{Max}}$ given in Example 1 can be over-approximated by an abstract invariant $\mathsf{I}^\sharp \in \mathbb{L} \mapsto \mathcal{D}^\sharp_{\mathbb{M}}$ such that: $\mathsf{I}^\sharp(\ell_1) = \top^\sharp_{\mathbb{M}} = \mathit{apv} \in [-\infty, \infty] \wedge \mathit{i1} \in [-\infty, \infty] \wedge \mathit{i2} \in [-\infty, \infty] \wedge \mathit{typ} \in [-\infty, \infty] \wedge \mathit{acs} \in [-\infty, \infty]$,
$\mathsf{I}^\sharp(\ell_2) = \mathit{apv} \in [1, 1] \wedge \mathit{i1} \in [-\infty, \infty] \wedge \mathit{i2} \in [-\infty, \infty] \wedge \mathit{typ} \in [-\infty, \infty] \wedge \mathit{acs} \in [-\infty, \infty]$,
$\mathsf{I}^\sharp(\ell_3) = \mathit{apv} \in [1, 1] \wedge \mathit{i1} \in [-1, 2] \wedge \mathit{i2} \in [-\infty, \infty] \wedge \mathit{typ} \in [-\infty, \infty] \wedge \mathit{acs} \in [-\infty, \infty]$,
$\mathsf{I}^\sharp(\ell_4) = \mathit{apv} \in [-1, 1] \wedge \mathit{i1} \in [-1, 2] \wedge \mathit{i2} \in [-\infty, \infty] \wedge \mathit{typ} \in [-\infty, \infty] \wedge \mathit{acs} \in [-\infty, \infty]$,
$\mathsf{I}^\sharp(\ell_5) = \mathit{apv} \in [-1, 1] \wedge \mathit{i1} \in [-1, 2] \wedge \mathit{i2} \in [-1, 2] \wedge \mathit{typ} \in [-\infty, \infty] \wedge \mathit{acs} \in [-\infty, \infty]$,
$\mathsf{I}^\sharp(\ell_6) = \mathit{apv} \in [-1, 1] \wedge \mathit{i1} \in [-1, 2] \wedge \mathit{i2} \in [-1, 2] \wedge \mathit{typ} \in [-\infty, \infty] \wedge \mathit{acs} \in [-\infty, \infty]$,
$\mathsf{I}^\sharp(\ell_7) = \mathit{apv} \in [-1, 1] \wedge \mathit{i1} \in [-1, 2] \wedge \mathit{i2} \in [-1, 2] \wedge \mathit{typ} \in [1, 2] \wedge \mathit{acs} \in [-\infty, \infty]$,
$\mathsf{I}^\sharp(\ell_8) = \mathit{apv} \in [-1, 1] \wedge \mathit{i1} \in [-1, 2] \wedge \mathit{i2} \in [-1, 2] \wedge \mathit{typ} \in [1, 2] \wedge \mathit{acs} \in [-2, 2]$.

In addition, the concrete trace property "the access to o fails" is over-approximated by another abstract invariant $\mathsf{I}^{\sharp\prime} \in \mathbb{L} \mapsto \mathcal{D}^\sharp_{\mathbb{M}}$ such that its abstract environment element attached to every program point is the same as $\mathsf{I}^\sharp$ defined above, except the value bound of *acs* at point ℓ_8 is refined to $[-2, 0]$. More precisely, when $\ell = \ell_8$, $\mathsf{I}^{\sharp\prime}(\ell) = \mathit{apv} \in [0, 1] \wedge \mathit{i1} \in [0, 1] \wedge \mathit{i2} \in [0, 1] \wedge \mathit{typ} \in [1, 2] \wedge \mathit{acs} \in [-2, 0]$; otherwise, $\mathsf{I}^{\sharp\prime}(\ell) = \mathsf{I}^\sharp(\ell)$.

4.2 Forward Reachability Analysis

Given a program P, the corresponding forward reachability semantics specifies the set of states that are possibly reachable at any program point. Section 4.2.1 formalizes a variant of the classic forward reachability (invariant) semantics, which is defined as an abstraction of the program's intermediate trace semantics. Section 4.2.2 briefly presents an abstract forward reachability analysis that soundly over-approximates the concrete forward reachability semantics.

4.2.1 *Forward Reachability Semantics.*

(1) Classic Forward Reachability Semantics. In the literature, usually the forward reachability semantics of a program is defined as an abstraction of its prefix trace semantics, which attaches to each program point a set of environments that are possibly encountered during any execution from

a given set of initial environments. More precisely, given a set of initial environments $\mathrm{M}^i \in \wp(\mathbb{M})$, the forward reachability semantics $\mathcal{S}_{\overrightarrow{r}}[\![\mathrm{P}]\!](\mathrm{M}^i) \in \mathbb{L} \mapsto \wp(\mathbb{M})$ is defined as a mapping from each program point ℓ to a set of environments at ℓ that are reachable from M^i. The formal definition is given as below, where $\langle \ell^i,\ \rho^i\rangle\sigma\langle \ell,\ \rho\rangle$ denotes the concatenation of an initial state $\langle \ell^i,\ \rho^i\rangle$, a (possibly empty) finite trace σ and a state $\langle \ell,\ \rho\rangle$. Specially, if σ is empty and $\langle \ell,\ \rho\rangle$ is equal to $\langle \ell^i,\ \rho^i\rangle$, then $\langle \ell^i,\ \rho^i\rangle\sigma\langle \ell,\ \rho\rangle$ represents a trace with only one state $\langle \ell^i,\ \rho^i\rangle$.

$$\begin{array}{rcll} \mathcal{S}_{\overrightarrow{r}}[\![\mathrm{P}]\!] & \in & \wp(\mathbb{M}) \mapsto (\mathbb{L} \mapsto \wp(\mathbb{M})) & \text{classic forward reachability semantics} \\ \mathcal{S}_{\overrightarrow{r}}[\![\mathrm{P}]\!](\mathrm{M}^i)\ell & \triangleq & \{\rho \in \mathbb{M} \mid \exists \sigma \in \mathbb{S}^*, \rho^i \in \mathrm{M}^i.\ \langle \ell^i,\ \rho^i\rangle\sigma\langle \ell,\ \rho\rangle \in [\![\mathrm{P}]\!]^{\mathrm{Pref}}\} & \end{array}$$

The classic forward reachability semantics defined above specifies an invariant property of the program executions. If the set of initial environments M^i is taken as a precondition, then $\mathcal{S}_{\overrightarrow{r}}[\![\mathrm{P}]\!](\mathrm{M}^i)\ell$ is an invariant at ℓ, which holds if and when the execution of P starting with an initial state satisfying M^i reaches program point ℓ. Such a forward reachability semantics is quite useful in verifying program correctness.

(2) Forward (Possible Success) Reachability Semantics. To build a Galois connection between the forward reachability semantics and the backward accessibility semantics (defined in Section 4.3) and facilitate the trace partitioning by invariants during the forward reachability analysis (introduced later in Section 5), we define a variant of forward reachability semantics, in which the considered execution traces are not required to start from the initial point ℓ^i.

To be more precise, instead of collecting reachable states from a set of initial environments M^i, here the precondition $\mathrm{I}_{\mathrm{pre}} \in \mathbb{L} \mapsto \wp(\mathbb{M})$ is specified by sets of environments attached to any (not necessarily initial) program point, and the forward (possible success) reachability semantics $\mathcal{S}_{\overrightarrow{ps}}[\![\mathrm{P}]\!]$ collects all the reachable states in the intermediate execution traces, which start from states satisfying the precondition $\mathrm{I}_{\mathrm{pre}}$.

$$\begin{array}{rcll} \mathcal{S}_{\overrightarrow{ps}}[\![\mathrm{P}]\!] & \in & (\mathbb{L} \mapsto \wp(\mathbb{M})) \mapsto (\mathbb{L} \mapsto \wp(\mathbb{M})) & \text{forward reachability semantics} \\ \mathcal{S}_{\overrightarrow{ps}}[\![\mathrm{P}]\!](\mathrm{I}_{\mathrm{pre}})\ell' & \triangleq & \{\rho' \in \mathbb{M} \mid \exists \sigma \in \mathbb{S}^*, \ell \in \mathbb{L}, \rho \in \mathrm{I}_{\mathrm{pre}}(\ell).\ \langle \ell,\ \rho\rangle\sigma\langle \ell',\ \rho'\rangle \in [\![\mathrm{P}]\!]^{\mathrm{It}}\} & \end{array}$$

Given a precondition $\mathrm{I}_{\mathrm{pre}} \in \mathbb{L} \mapsto \wp(\mathbb{M})$, the forward (possible success) reachability semantics $\mathcal{S}_{\overrightarrow{ps}}[\![\mathrm{P}]\!](\mathrm{I}_{\mathrm{pre}})\ell$ specifies an invariant at each point ℓ, which holds if and when an execution of P starting with a state satisfying $\mathrm{I}_{\mathrm{pre}}$ reaches the point ℓ.

To distinguish from the classic forward reachability semantics and the forward impossible failure reachability semantics introduced in Reference [18], the semantics $\mathcal{S}_{\overrightarrow{ps}}[\![\mathrm{P}]\!]$ defined above is formally named "*forward possible success reachability semantics*." Nevertheless, in the rest of this article, the notation of *forward reachability semantics* (where "possible success" or its abbreviation "ps" is omitted) refers to $\mathcal{S}_{\overrightarrow{ps}}[\![\mathrm{P}]\!]$.

It is easy to see that the classic forward reachability semantics $\mathcal{S}_{\overrightarrow{r}}[\![\mathrm{P}]\!] \in \wp(\mathbb{M}) \mapsto (\mathbb{L} \mapsto \wp(\mathbb{M}))$ is an abstraction of our definition $\mathcal{S}_{\overrightarrow{ps}}[\![\mathrm{P}]\!] \in (\mathbb{L} \mapsto \wp(\mathbb{M})) \mapsto (\mathbb{L} \mapsto \wp(\mathbb{M}))$, and $\mathcal{S}_{\overrightarrow{r}}[\![\mathrm{P}]\!](\mathrm{M}^i)$ is equal to $\mathcal{S}_{\overrightarrow{ps}}[\![\mathrm{P}]\!](\mathrm{I}_{\mathrm{pre}})$ if $\mathrm{I}_{\mathrm{pre}} = \lambda\ell \in \mathbb{L}.\ (\ell == \ell^i)\ ?\ \mathrm{M}^i : \emptyset$.

(3) Forward (Possible Success) Reachability Semantics in Fixpoint Form. The forward reachability semantics $\mathcal{S}_{\overrightarrow{ps}}[\![\mathrm{P}]\!]$ of a program $\mathrm{P} = \langle \mathbb{S}^i, \rightarrow\rangle$ can be defined by structural induction on the language-specific syntax of the program or in the fixpoint form with a forward transfer function $\mathrm{F}_{\overrightarrow{ps}}[\![\mathrm{P}]\!]$:

$$\mathcal{S}^{\rightarrow}_{\overline{ps}}[\![\mathrm{P}]\!] \in (\mathbb{L} \mapsto \wp(\mathbb{M})) \mapsto (\mathbb{L} \mapsto \wp(\mathbb{M})) \quad \text{forward reachability semantics}$$
$$\mathcal{S}^{\rightarrow}_{\overline{ps}}[\![\mathrm{P}]\!](\mathsf{I}_{\mathrm{pre}}) \triangleq \mathrm{lfp}^{\dot{\subseteq}}_{\mathsf{I}_{\mathrm{pre}}} \mathrm{F}^{\rightarrow}_{\overline{ps}}[\![\mathrm{P}]\!]$$
$$\mathrm{F}^{\rightarrow}_{\overline{ps}}[\![\mathrm{P}]\!] \in (\mathbb{L} \mapsto \wp(\mathbb{M})) \mapsto (\mathbb{L} \mapsto \wp(\mathbb{M})) \quad \text{forward transfer function}$$
$$\mathrm{F}^{\rightarrow}_{\overline{ps}}[\![\mathrm{P}]\!]\mathsf{I} \triangleq \mathsf{I} \dot{\cup} \lambda \ell' \in \mathbb{L}.\{\rho' \in \mathbb{M} \mid \exists \ell \in \mathbb{L}, \rho \in \mathsf{I}(\ell).\ \langle \ell,\ \rho\rangle \rightarrow\ \langle \ell',\ \rho'\rangle\},$$

where $\dot{\subseteq}$ and $\dot{\cup}$ are pointwise extensions of the standard inclusion relation $\subseteq$ and union operator $\cup$, respectively, and $\mathrm{lfp}^{\dot{\subseteq}}_{\mathsf{I}}\ \mathrm{F}$ denotes the $\dot{\subseteq}$-least fixpoint of F that is $\dot{\subseteq}$-greater than or equal to I, if it exists, which is the case for $\mathrm{F}^{\rightarrow}_{\overline{ps}}[\![\mathrm{P}]\!]$ by Tarski's fixpoint theorem [72].

Essentially, the monotonic function $\mathrm{F}^{\rightarrow}_{\overline{ps}}[\![\mathrm{P}]\!]$ described above can be constructed by combining atomic forward transfer functions, each of which is typically defined for an atomic action (instruction/computation step) in the program and associates a set of environments before the action with the set of environments reachable after the action.

More formally, here we assume that for every pair of program points $\langle \ell,\ \ell'\rangle$ in the program P, an atomic transfer function $\mathrm{F}_{\ell\rightarrow\ell'}[\![\mathrm{P}]\!] \in \wp(\mathbb{M}) \mapsto \wp(\mathbb{M})$ is provided such that for any set M of environments at point ℓ, the function $\mathrm{F}_{\ell\rightarrow\ell'}[\![\mathrm{P}]\!](\mathrm{M})$ returns the set of environments at point ℓ' that are reachable from M: (i) if $\ell = \ell'$, then $\mathrm{F}_{\ell\rightarrow\ell'}[\![\mathrm{P}]\!](\mathrm{M}) = \mathrm{M}$; (ii) if $\ell \neq \ell'$ and there is not an atomic action from ℓ to ℓ', then $\mathrm{F}_{\ell\rightarrow\ell'}[\![\mathrm{P}]\!](\mathrm{M}) = \emptyset$; and (iii) otherwise, there is an atomic action from ℓ to ℓ', then $\mathrm{F}_{\ell\rightarrow\ell'}[\![\mathrm{P}]\!](\mathrm{M})$ is the set of environments after executing the action from M. Taking the simple language in Figure 2 as an example, there are only two types of atomic actions: For an assignment ${}^{\ell_1}\chi := e^{\ell_2}$, the corresponding atomic transfer function $\mathrm{F}_{\ell_1\rightarrow\ell_2}[\![\mathrm{P}]\!](\mathrm{M}) = \tau\{\!|\ \chi := e\ |\!\}\mathrm{M}$, which is defined in Figure 3; similarly, for a Boolean test ${}^{\ell_1}b^{\ell_2}$, the corresponding atomic transfer function $\mathrm{F}_{\ell_1\rightarrow\ell_2}[\![\mathrm{P}]\!](\mathrm{M}) = \tau\{\!|\ b\ |\!\}\mathrm{M}$.

Therefore, the definition of forward transfer function $\mathrm{F}^{\rightarrow}_{\overline{ps}}[\![\mathrm{P}]\!]$ can be rephrased into:

$$\mathrm{F}^{\rightarrow}_{\overline{ps}}[\![\mathrm{P}]\!] \in (\mathbb{L} \mapsto \wp(\mathbb{M})) \mapsto (\mathbb{L} \mapsto \wp(\mathbb{M})) \quad \text{forward transfer function}$$
$$\mathrm{F}^{\rightarrow}_{\overline{ps}}[\![\mathrm{P}]\!]\mathsf{I} \triangleq \lambda \ell' \in \mathbb{L}.\ \cup_{\ell\in\mathbb{L}} \mathrm{F}_{\ell\rightarrow\ell'}[\![\mathrm{P}]\!](\mathsf{I}(\ell)).$$

Example 13 (Access Control, Continued). For the access control program, the forward transfer function $\mathrm{F}^{\rightarrow}_{\overline{ps}}[\![\mathrm{P}]\!]$ can be derived by combining the following atomic transfer functions: $\tau\{\!|\ apv := 1\ |\!\}$, $\tau\{\!|\ i1 := [-1;2]\ |\!\}$, $\tau\{\!|\ apv := (i1 \leq 0)\ ?\ -1 : apv\ |\!\}$, $\tau\{\!|\ i2 := [-1;2]\ |\!\}$, $\tau\{\!|\ apv := (apv \geq 1 \wedge i2 \leq 0)\ ?\ -1 : apv\ |\!\}$, $\tau\{\!|\ typ := [1;2]\ |\!\}$, and $\tau\{\!|\ acs := apv \times typ\ |\!\}$. Then, from a precondition $\mathsf{I}_{\mathrm{pre}} \in \mathbb{L} \mapsto \wp(\mathbb{M})$ such that $\mathsf{I}_{\mathrm{pre}}(\ell_1) = \mathbb{M}$ and $\mathsf{I}_{\mathrm{pre}}(\ell) = \emptyset$ for $\ell \neq \ell_1$, we can compute the forward reachability semantics $\mathcal{S}^{\rightarrow}_{\overline{ps}}[\![\mathrm{P}]\!](\mathsf{I}_{\mathrm{pre}})$ by the least fixpoint $\mathrm{lfp}^{\dot{\subseteq}}_{\mathsf{I}_{\mathrm{pre}}} \mathrm{F}^{\rightarrow}_{\overline{ps}}[\![\mathrm{P}]\!]$, which is equal to the classic invariant semantics.

To be more precise, the result $\mathcal{S}^{\rightarrow}_{\overline{ps}}[\![\mathrm{P}]\!](\mathsf{I}_{\mathrm{pre}})$ is listed in Table 1, in which the constraints on environment like "$\rho(apv) = 1$" is written as "$apv = 1$" for short.

4.2.2 Over-approximating Abstract Forward Reachability Analysis. Although the concrete forward reachability semantics $\mathcal{S}^{\rightarrow}_{\overline{ps}}[\![\mathrm{P}]\!]$ can be easily computed in the Example 13 (since there are no infinite loops in the access control program and the variable values are bounded integers), it is not computable in general, and an over-approximation is necessary.

(1) Over-approximating Abstract Forward Transfer Function. For the forward transfer function $\mathrm{F}^{\rightarrow}_{\overline{ps}}[\![\mathrm{P}]\!] \in (\mathbb{L} \mapsto \wp(\mathbb{M})) \mapsto (\mathbb{L} \mapsto \wp(\mathbb{M}))$ on the concrete invariant domain, we need to construct

Table 1. Concrete Forward Reachability Semantics for the Access Control Program

ℓ	$\mathcal{S}^{\overrightarrow{ps}}[\![\mathrm{P}]\!](\mathsf{I}_{\mathrm{pre}})\ell$
ℓ_1	$\mathbb{M}$
ℓ_2	$\{\rho \in \mathbb{M} \mid apv = 1\}$
ℓ_3	$\{\rho \in \mathbb{M} \mid apv = 1 \wedge i1 \in \{-1, 0, 1, 2\}\}$
ℓ_4	$\{\rho \in \mathbb{M} \mid apv = 1 \wedge i1 \in \{1, 2\}\} \cup \{\rho \in \mathbb{M} \mid apv = -1 \wedge i1 \in \{-1, 0\}\}$
ℓ_5	$\{\rho \in \mathbb{M} \mid apv = 1 \wedge i1 \in \{1, 2\} \wedge i2 \in \{-1, 0, 1, 2\}\}$ $\cup \{\rho \in \mathbb{M} \mid apv = -1 \wedge i1 \in \{-1, 0\} \wedge i2 \in \{-1, 0, 1, 2\}\}$
ℓ_6	$\{\rho \in \mathbb{M} \mid apv = 1 \wedge i1 \in \{1, 2\} \wedge i2 \in \{1, 2\}\}$ $\cup \{\rho \in \mathbb{M} \mid apv = -1 \wedge i1 \in \{1, 2\} \wedge i2 \in \{-1, 0\}\}$ $\cup \{\rho \in \mathbb{M} \mid apv = -1 \wedge i1 \in \{-1, 0\} \wedge i2 \in \{-1, 0, 1, 2\}\}$
ℓ_7	$\{\rho \in \mathbb{M} \mid apv = 1 \wedge i1 \in \{1, 2\} \wedge i2 \in \{1, 2\} \wedge typ \in \{1, 2\}\}$ $\cup \{\rho \in \mathbb{M} \mid apv = -1 \wedge i1 \in \{1, 2\} \wedge i2 \in \{-1, 0\} \wedge typ \in \{1, 2\}\}$ $\cup \{\rho \in \mathbb{M} \mid apv = -1 \wedge i1 \in \{-1, 0\} \wedge i2 \in \{-1, 0, 1, 2\} \wedge typ \in \{1, 2\}\}$
ℓ_8	$\{\rho \in \mathbb{M} \mid apv = 1 \wedge i1 \in \{1, 2\} \wedge i2 \in \{1, 2\} \wedge typ = 1 \wedge acs = 1\}$ $\cup \{\rho \in \mathbb{M} \mid apv = 1 \wedge i1 \in \{1, 2\} \wedge i2 \in \{1, 2\} \wedge typ = 2 \wedge acs = 2\}$ $\cup \{\rho \in \mathbb{M} \mid apv = -1 \wedge i1 \in \{1, 2\} \wedge i2 \in \{-1, 0\} \wedge typ = 1 \wedge acs = -1\}$ $\cup \{\rho \in \mathbb{M} \mid apv = -1 \wedge i1 \in \{1, 2\} \wedge i2 \in \{-1, 0\} \wedge typ = 2 \wedge acs = -2\}$ $\cup \{\rho \in \mathbb{M} \mid apv = -1 \wedge i1 \in \{-1, 0\} \wedge i2 \in \{-1, 0, 1, 2\} \wedge typ = 1 \wedge acs = -1\}$ $\cup \{\rho \in \mathbb{M} \mid apv = -1 \wedge i1 \in \{-1, 0\} \wedge i2 \in \{-1, 0, 1, 2\} \wedge typ = 2 \wedge acs = -2\}$

an abstract forward transfer function $\hat{\mathrm{F}}^{\sharp}_{\overrightarrow{ps}}[\![\mathrm{P}]\!] \in (\mathbb{L} \mapsto \mathcal{D}^{\sharp}_{\mathbb{M}}) \mapsto (\mathbb{L} \mapsto \mathcal{D}^{\sharp}_{\mathbb{M}})$ that over-approximates $\mathrm{F}_{\overrightarrow{ps}}[\![\mathrm{P}]\!]$, where the symbol $\hat{}$ denotes over-approximations.

In Section 4.1.1, it is assumed that for each transfer function $\mathrm{F}_{\ell\to\ell'}[\![\mathrm{P}]\!] \in \wp(\mathbb{M}) \mapsto \wp(\mathbb{M})$ defined for atomic actions, the abstract environment domain $\mathcal{D}^{\sharp}_{\mathbb{M}}$ provides an abstract function $\hat{\mathrm{F}}^{\sharp}_{\ell\to\ell'}[\![\mathrm{P}]\!] \in \mathcal{D}^{\sharp}_{\mathbb{M}} \mapsto \mathcal{D}^{\sharp}_{\mathbb{M}}$ such that $\forall \mathrm{M}^{\sharp} \in \mathcal{D}^{\sharp}_{\mathbb{M}}.\ (\mathrm{F}_{\ell\to\ell'}[\![\mathrm{P}]\!] \circ \gamma_{\mathbb{M}})(\mathrm{M}^{\sharp}) \subseteq (\gamma_{\mathbb{M}} \circ \hat{\mathrm{F}}^{\sharp}_{\ell\to\ell'}[\![\mathrm{P}]\!])(\mathrm{M}^{\sharp})$. For instance, the interval/polyhedron/octagon domain provides the over-approximating abstract transfer versions $\tau^{\sharp}\!\!\rfloor x := e \lfloor\!\!\rfloor$ and $\tau^{\sharp}\!\!\rfloor b \lfloor\!\!\rfloor$ for $\tau\!\!\rfloor x := e \lfloor\!\!\rfloor$ and $\tau\!\!\rfloor b \lfloor\!\!\rfloor$. Therefore, $\hat{\mathrm{F}}^{\sharp}_{\overrightarrow{ps}}[\![\mathrm{P}]\!]$ can be constructed by the join of $\hat{\mathrm{F}}^{\sharp}_{\ell\to\ell'}[\![\mathrm{P}]\!]$ functions:

$$\begin{array}{lcll} \hat{\mathrm{F}}^{\sharp}_{\overrightarrow{ps}}[\![\mathrm{P}]\!] & \in & \left(\mathbb{L} \mapsto \mathcal{D}^{\sharp}_{\mathbb{M}}\right) \mapsto \left(\mathbb{L} \mapsto \mathcal{D}^{\sharp}_{\mathbb{M}}\right) & \text{abstract forward transfer function} \\ \hat{\mathrm{F}}^{\sharp}_{\overrightarrow{ps}}[\![\mathrm{P}]\!]\mathsf{I}^{\sharp} & \triangleq & \lambda \ell' \in \mathbb{L}.\ \sqcup^{\sharp}_{\mathbb{M}\ \ell\in\mathbb{L}}\ \hat{\mathrm{F}}^{\sharp}_{\ell\to\ell'}[\![\mathrm{P}]\!](\mathsf{I}^{\sharp}(\ell)). & \end{array}$$

The abstract function $\hat{\mathrm{F}}^{\sharp}_{\overrightarrow{ps}}[\![\mathrm{P}]\!]$ is monotonic and obeys soundness condition:

$$\forall \mathsf{I}^{\sharp} \in \mathbb{L} \mapsto \mathcal{D}^{\sharp}_{\mathbb{M}}.\ \mathrm{F}_{\overrightarrow{ps}}[\![\mathrm{P}]\!] \circ \dot{\gamma}_{\mathbb{M}}(\mathsf{I}^{\sharp}) \mathrel{\dot{\subseteq}} \dot{\gamma}_{\mathbb{M}} \circ \hat{\mathrm{F}}^{\sharp}_{\overrightarrow{ps}}[\![\mathrm{P}]\!](\mathsf{I}^{\sharp}). \tag{7}$$

By the monotonic property and the soundness condition (7) of $\hat{\mathrm{F}}^{\sharp}_{\overrightarrow{ps}}[\![\mathrm{P}]\!]$, we know that: For any $\mathsf{I}_{\mathrm{pre}} \in \mathbb{L} \mapsto \wp(\mathbb{M})$ and $\mathsf{I}^{\sharp}_{\mathrm{pre}} \in \mathbb{L} \mapsto \mathcal{D}^{\sharp}_{\mathbb{M}}$, if $\mathsf{I}_{\mathrm{pre}} \mathrel{\dot{\subseteq}} \dot{\gamma}_{\mathbb{M}}(\mathsf{I}^{\sharp}_{\mathrm{pre}})$, then $\mathrm{lfp}^{\dot{\subseteq}}_{\mathsf{I}_{\mathrm{pre}}} \mathrm{F}_{\overrightarrow{ps}}[\![\mathrm{P}]\!] \mathrel{\dot{\subseteq}} \dot{\gamma}_{\mathbb{M}}(\mathrm{lfp}^{\dot{\subseteq}^{\sharp}_{\mathbb{M}}}_{\mathsf{I}^{\sharp}_{\mathrm{pre}}} \hat{\mathrm{F}}^{\sharp}_{\overrightarrow{ps}}[\![\mathrm{P}]\!])$. That is to say, the concrete forward reachability semantics $\mathcal{S}^{\overrightarrow{ps}}[\![\mathrm{P}]\!](\mathsf{I}_{\mathrm{pre}})$ can be soundly over-approximated by the least fixpoint of the abstract forward transfer function $\hat{\mathrm{F}}^{\sharp}_{\overrightarrow{ps}}[\![\mathrm{P}]\!]$.

(2) Widening. In most abstract environment domains (e.g., intervals, polyhedra, octagons), there may exist infinite increasing chains, hence the iteration of $\hat{\mathrm{F}}^{\sharp}_{\overrightarrow{ps}}[\![\mathrm{P}]\!]$ may not converge in finite time. To address this problem, we need a widening operator $\triangledown_{\mathbb{I}} \in (\mathbb{L} \mapsto \mathcal{D}^{\sharp}_{\mathbb{M}}) \times (\mathbb{L} \mapsto \mathcal{D}^{\sharp}_{\mathbb{M}}) \mapsto (\mathbb{L} \mapsto \mathcal{D}^{\sharp}_{\mathbb{M}})$ on the abstract invariant domain, which satisfies the following soundness and termination conditions:

(i) $\forall x^{\sharp}, y^{\sharp} \in \mathbb{L} \mapsto \mathcal{D}^{\sharp}_{\mathbb{M}}.\ \dot{\gamma}_{\mathbb{M}}(x^{\sharp}) \dot{\cup} \dot{\gamma}_{\mathbb{M}}(y^{\sharp}) \dot{\subseteq} \dot{\gamma}_{\mathbb{M}}(x^{\sharp} \triangledown_{\mathbb{I}} y^{\sharp})$;

(ii) for any sequence $(x^{\sharp}_i)_{i\in\mathbb{N}}$, the sequence $(y^{\sharp}_i)_{i\in\mathbb{N}}$ defined as $y^{\sharp}_0 = x^{\sharp}_0$ and $\forall i \in \mathbb{N}.\ y^{\sharp}_{i+1} = y^{\sharp}_i \triangledown_{\mathbb{I}} x^{\sharp}_{i+1}$ converges in finite time.

The implementation of $\triangledown_{\mathbb{I}} \in (\mathbb{L} \mapsto \mathcal{D}^{\sharp}_{\mathbb{M}}) \times (\mathbb{L} \mapsto \mathcal{D}^{\sharp}_{\mathbb{M}}) \mapsto (\mathbb{L} \mapsto \mathcal{D}^{\sharp}_{\mathbb{M}})$ naturally follows the widening operator $\triangledown_{\mathbb{M}} \in \mathcal{D}^{\sharp}_{\mathbb{M}} \times \mathcal{D}^{\sharp}_{\mathbb{M}} \mapsto \mathcal{D}^{\sharp}_{\mathbb{M}}$ provided by the abstract environment domain, such that $\mathrm{I}^{\sharp} \triangledown_{\mathbb{I}} \mathrm{I}^{\sharp\prime} \triangleq \lambda \ell \in \mathbb{L}.\ \mathrm{I}^{\sharp}(\ell) \triangledown_{\mathbb{M}} \mathrm{I}^{\sharp\prime}(\ell)$. It is easy to prove such a definition of $\triangledown_{\mathbb{I}}$ obeys the soundness and termination conditions, and we omit it here.

(3) Abstract Forward Reachability Semantics. Given a precondition represented by $\mathrm{I}^{\sharp}_{\mathrm{pre}} \in \mathbb{L} \mapsto \mathcal{D}^{\sharp}_{\mathbb{M}}$, the corresponding concrete reachability semantics $\mathcal{S}_{\overrightarrow{ps}}[\![\mathrm{P}]\!](\dot{\gamma}_{\mathbb{M}}(\mathrm{I}^{\sharp}_{\mathrm{pre}}))$ is the least fixpoint of function $\mathrm{F}_{\overrightarrow{ps}}[\![\mathrm{P}]\!]$ that is greater than or equal to $\dot{\gamma}_{\mathbb{M}}(\mathrm{I}^{\sharp}_{\mathrm{pre}})$. That is to say, $\mathcal{S}_{\overrightarrow{ps}}[\![\mathrm{P}]\!](\dot{\gamma}_{\mathbb{M}}(\mathrm{I}^{\sharp}_{\mathrm{pre}})) = \mathrm{lfp}^{\dot{\subseteq}}_{\dot{\gamma}_{\mathbb{M}}(\mathrm{I}^{\sharp}_{\mathrm{pre}})} \mathrm{F}_{\overrightarrow{ps}}[\![\mathrm{P}]\!]$. By Cousot and Cousot's upward iteration with widening theorem, $\mathrm{lfp}^{\dot{\subseteq}}_{\dot{\gamma}_{\mathbb{M}}(\mathrm{I}^{\sharp}_{\mathrm{pre}})} \mathrm{F}_{\overrightarrow{ps}}[\![\mathrm{P}]\!]$ can be soundly over-approximated by the limit of a ultimately stationary sequence $(\mathrm{I}^{\sharp}_i)_{i\in\mathbb{N}}$, where $\mathrm{I}^{\sharp}_0 = \mathrm{I}^{\sharp}_{\mathrm{pre}}$ and $\forall i \in \mathbb{N}.\ \mathrm{I}^{\sharp}_{i+1} = \mathrm{I}^{\sharp}_i \triangledown_{\mathbb{I}} \hat{\mathrm{F}}^{\sharp}_{\overrightarrow{ps}}[\![\mathrm{P}]\!](\mathrm{I}^{\sharp}_i)$.

$$\forall \mathrm{I}^{\sharp}_{\mathrm{pre}} \in \mathbb{L} \mapsto \mathcal{D}^{\sharp}_{\mathbb{M}}.\ \mathrm{lfp}^{\dot{\subseteq}}_{\dot{\gamma}_{\mathbb{M}}(\mathrm{I}^{\sharp}_{\mathrm{pre}})} \mathrm{F}_{\overrightarrow{ps}}[\![\mathrm{P}]\!] \dot{\subseteq} \dot{\gamma}_{\mathbb{M}} \left(\lim_{\mathrm{I}^{\sharp}_{\mathrm{pre}}} \lambda \mathrm{I}^{\sharp}.\ \mathrm{I}^{\sharp} \triangledown_{\mathbb{I}} \hat{\mathrm{F}}^{\sharp}_{\overrightarrow{ps}}[\![\mathrm{P}]\!](\mathrm{I}^{\sharp}) \right). \tag{8}$$

In the rest of this article, the abstract forward reachability semantics $\mathcal{S}^{\sharp}_{\overrightarrow{ps}}[\![\mathrm{P}]\!]$ refers to the following definition, which gives a sound over-approximation of the concrete reachability semantics and can be computed in finite time:

$$\begin{array}{rcll} \mathcal{S}^{\sharp}_{\overrightarrow{ps}}[\![\mathrm{P}]\!] & \in & \left(\mathbb{L} \mapsto \mathcal{D}^{\sharp}_{\mathbb{M}}\right) \mapsto \left(\mathbb{L} \mapsto \mathcal{D}^{\sharp}_{\mathbb{M}}\right) & \text{abstract forward reachability semantics} \\ \mathcal{S}^{\sharp}_{\overrightarrow{ps}}[\![\mathrm{P}]\!](\mathrm{I}^{\sharp}_{\mathrm{pre}}) & \triangleq & \lim_{\mathrm{I}^{\sharp}_{\mathrm{pre}}} \lambda \mathrm{I}^{\sharp}.\ \mathrm{I}^{\sharp} \triangledown_{\mathbb{I}} \hat{\mathrm{F}}^{\sharp}_{\overrightarrow{ps}}[\![\mathrm{P}]\!](\mathrm{I}^{\sharp}). & \end{array}$$

Example 14 (Access Control, Continued). For the access control program in Figure 4, we use the interval domain as the abstract environment domain $\mathcal{D}^{\sharp}_{\mathbb{M}}$. Given an abstract precondition $\mathrm{I}^{\sharp}_{\mathrm{pre}} \in \mathbb{L} \mapsto \mathcal{D}^{\sharp}_{\mathbb{M}}$ such that $\mathrm{I}^{\sharp}_{\mathrm{pre}}(\ell_1) = \top^{\sharp}_{\mathbb{M}}$ and $\mathrm{I}^{\sharp}_{\mathrm{pre}}(\ell) = \bot^{\sharp}_{\mathbb{M}}$ for $\ell \neq \ell_1$, the corresponding abstract forward reachability semantics $\mathcal{S}^{\sharp}_{\overrightarrow{ps}}[\![\mathrm{P}]\!](\mathrm{I}^{\sharp}_{\mathrm{pre}})\ell$ is listed in Table 2.

Compared with the concrete reachability semantics $\mathcal{S}_{\overrightarrow{ps}}[\![\mathrm{P}]\!](\mathrm{I}_{\mathrm{pre}})\ell$ in Table 1, it is obvious that $\mathcal{S}^{\sharp}_{\overrightarrow{ps}}[\![\mathrm{P}]\!](\mathrm{I}^{\sharp}_{\mathrm{pre}})\ell$ is an over-approximation and contains some spurious environments that are not reachable in the concrete (e.g., the value of *acs* cannot be 0 at ℓ_8 in the concrete).

4.3 Backward Accessibilty Analysis

Given a precondition on states, the forward reachability analysis collects states that are possibly reachable by the executions from states satisfying the precondition. Inversely, given a

Table 2. Abstract Forward Reachability Semantics for the Access Control Program

ℓ	$\mathcal{S}^{\sharp}_{\overrightarrow{ps}}[\![\mathrm{P}]\!](\mathsf{I}^{\sharp}_{\mathrm{pre}})\ell$
ℓ_1	$\top^{\sharp}_{\mathbb{M}}$
ℓ_2	$apv \in [1;1] \wedge i1 \in [-\infty;\infty] \wedge i2 \in [-\infty;\infty] \wedge typ \in [-\infty;\infty] \wedge acs \in [-\infty;\infty]$
ℓ_3	$apv \in [1;1] \wedge i1 \in [-1;2] \wedge i2 \in [-\infty;\infty] \wedge typ \in [-\infty;\infty] \wedge acs \in [-\infty;\infty]$
ℓ_4	$apv \in [-1;1] \wedge i1 \in [-1;2] \wedge i2 \in [-\infty;\infty] \wedge typ \in [-\infty;\infty] \wedge acs \in [-\infty;\infty]$
ℓ_5	$apv \in [-1;1] \wedge i1 \in [-1;2] \wedge i2 \in [-1;2] \wedge typ \in [-\infty;\infty] \wedge acs \in [-\infty;\infty]$
ℓ_6	$apv \in [-1;1] \wedge i1 \in [-1;2] \wedge i2 \in [-1;2] \wedge typ \in [-\infty;\infty] \wedge acs \in [-\infty;\infty]$
ℓ_7	$apv \in [-1;1] \wedge i1 \in [-1;2] \wedge i2 \in [-1;2] \wedge typ \in [1;2] \wedge acs \in [-\infty;\infty]$
ℓ_8	$apv \in [-1;1] \wedge i1 \in [-1;2] \wedge i2 \in [-1;2] \wedge typ \in [1;2] \wedge acs \in [-2;2]$

postcondition on states, the backward accessibility analysis collects states from which the executions reach states satisfying the postcondition.

In general, there are two types of backward accessibility analysis: (1) the *backward impossible failure accessibility analysis* computes the states, from which the executions can reach only the states satisfying the given postcondition (i.e., it is impossible to reach states that fail the postcondition); (2) the *backward possible success accessibility analysis* introduced in Reference [18] computes the states, from which the executions may reach a state satisfying the given postcondition (i.e., it is possible to succeed to reach a state satisfying the postcondition). This section discusses the backward impossible failure accessibility semantics, which is essentially equivalent to the *sufficient condition semantics* in References [60, 61]. More precisely, we briefly review the under-approximating abstract analysis introduced by Miné and propose a new over-approximating abstract backward impossible failure accessibility analysis.

4.3.1 Backward Impossible Failure Accessibility Semantics. The forward (possible success) reachability semantics $\mathcal{S}_{\overrightarrow{ps}}[\![\mathrm{P}]\!] \in (\mathbb{L} \mapsto \wp(\mathbb{M})) \mapsto (\mathbb{L} \mapsto \wp(\mathbb{M}))$ is the lower adjoint in a Galois connection, and the corresponding upper adjoint is defined as the backward impossible failure accessibility semantics $\mathcal{S}_{\overleftarrow{if}}[\![\mathrm{P}]\!] \in (\mathbb{L} \mapsto \wp(\mathbb{M})) \mapsto (\mathbb{L} \mapsto \wp(\mathbb{M}))$, such that any execution from states satisfying $\mathcal{S}_{\overleftarrow{if}}[\![\mathrm{P}]\!](\mathsf{I}_{\mathrm{post}})$ can reach only the states satisfying the given postcondition $\mathsf{I}_{\mathrm{post}} \in \mathbb{L} \mapsto \wp(\mathbb{M})$:

$$\langle \mathbb{L} \mapsto \wp(\mathbb{M}),\ \dot{\subseteq} \rangle \underset{\mathcal{S}_{\overrightarrow{ps}}[\![\mathrm{P}]\!]}{\overset{\mathcal{S}_{\overleftarrow{if}}[\![\mathrm{P}]\!]}{\leftrightarrows}} \langle \mathbb{L} \mapsto \wp(\mathbb{M}),\ \dot{\subseteq} \rangle, \tag{9}$$

where the definition of $\mathcal{S}_{\overleftarrow{if}}[\![\mathrm{P}]\!]$ is formalized as

$$\begin{aligned} \mathcal{S}_{\overleftarrow{if}}[\![\mathrm{P}]\!] &\in (\mathbb{L} \mapsto \wp(\mathbb{M})) \mapsto (\mathbb{L} \mapsto \wp(\mathbb{M})) \\ \mathcal{S}_{\overleftarrow{if}}[\![\mathrm{P}]\!](\mathsf{I}_{\mathrm{post}})\ell &\triangleq \{\rho \in \mathbb{M} \mid \forall \sigma \in \mathbb{S}^*, \ell' \in \mathbb{L}, \rho' \in \mathbb{M}.\ (\langle \ell,\ \rho\rangle\sigma\langle \ell',\ \rho'\rangle \in [\![\mathrm{P}]\!]^{\mathrm{lt}} \Rightarrow \rho' \in \mathsf{I}_{\mathrm{post}}(\ell'))\}. \end{aligned}$$

For any postcondition $\mathsf{I}_{\mathrm{post}} \in \mathbb{L} \mapsto \wp(\mathbb{M})$ that can represent a trace property of interest $\gamma_{\mathbb{TV}}(\mathsf{I}_{\mathrm{post}})$, the backward impossible failure accessibility semantics $\mathcal{S}_{\overleftarrow{if}}[\![\mathrm{P}]\!](\mathsf{I}_{\mathrm{post}})$ computes the states from which all the execution traces must have the property $\gamma_{\mathbb{TV}}(\mathsf{I}_{\mathrm{post}})$, or, say, it infers the sufficient preconditions for the postcondition $\mathsf{I}_{\mathrm{post}}$ to hold. It is of great importance to know that our $\mathcal{S}_{\overleftarrow{if}}[\![\mathrm{P}]\!]$ is equivalent to the sufficient condition semantics introduced in References [60, 61].

Backward Impossible Failure Accessibility Semantics in Fixpoint Form. Similar to the forward (possible success) reachability semantics $\mathcal{S}_{\overrightarrow{ps}}[\![\mathrm{P}]\!]$, the backward impossible failure accessibility

semantics $\mathcal{S}^{\leftarrow}_{if}[\![\mathrm{P}]\!]$ of a program $\mathrm{P} = \langle \mathbb{S}^i, \rightarrow \rangle$ can be also defined in the fixpoint form with a concrete backward transfer function $\mathrm{F}^{\leftarrow}_{if}[\![\mathrm{P}]\!]$:

$$\begin{array}{rcll} \mathcal{S}^{\leftarrow}_{if}[\![\mathrm{P}]\!] & \in & (\mathbb{L} \mapsto \wp(\mathbb{M})) \mapsto (\mathbb{L} \mapsto \wp(\mathbb{M})) & \text{backward IF accessibility semantics} \\ \mathcal{S}^{\leftarrow}_{if}[\![\mathrm{P}]\!](\mathsf{I}_{\mathrm{post}}) & \triangleq & \mathrm{gfp}^{\dot{\subseteq}}_{\mathsf{I}_{\mathrm{post}}} \mathrm{F}^{\leftarrow}_{if}[\![\mathrm{P}]\!] & \\ \mathrm{F}^{\leftarrow}_{if}[\![\mathrm{P}]\!] & \in & (\mathbb{L} \mapsto \wp(\mathbb{M})) \mapsto (\mathbb{L} \mapsto \wp(\mathbb{M})) & \text{backward IF transfer function} \\ \mathrm{F}^{\leftarrow}_{if}[\![\mathrm{P}]\!]\mathsf{I} & \triangleq & \mathsf{I} \mathbin{\dot{\cap}} \lambda \ell \in \mathbb{L}.\{\rho \in \mathbb{M} \mid \forall \ell' \in \mathbb{L}, \rho'. \langle \ell, \rho \rangle \rightarrow \langle \ell', \rho' \rangle \Rightarrow \rho' \in \mathsf{I}(\ell')\}, & \end{array}$$

where $\dot{\subseteq}$ and $\dot{\cap}$ are pointwise extensions of the standard set inclusion relation $\subseteq$ and intersection operator $\cap$, respectively, and $\mathrm{gfp}^{\dot{\subseteq}}_{\mathsf{I}} \mathrm{F}$ is the order-dual of $\mathrm{lfp}^{\dot{\subseteq}}_{\mathsf{I}} \mathrm{F}$.

As $\mathrm{F}^{\rightarrow}_{ps}[\![\mathrm{P}]\!]$ is constructed by combining atomic forward transfer functions $\mathrm{F}_{\ell \rightarrow \ell'}[\![\mathrm{P}]\!]$, we can construct $\mathrm{F}^{\leftarrow}_{if}[\![\mathrm{P}]\!]$ by atomic backward transfer functions $\mathrm{F}_{\ell \leftarrow \ell'}[\![\mathrm{P}]\!] \in \wp(\mathbb{M}) \mapsto \wp(\mathbb{M})$, which are defined for every pair of program points $\langle \ell, \ell' \rangle$ in the program such that, if there exists a single atomic action from ℓ to ℓ', then executions from environments in $\mathrm{F}_{\ell \leftarrow \ell'}[\![\mathrm{P}]\!]\mathrm{M}$ at point ℓ can only reach environments in M at point ℓ'. To be more precise: (i) if $\ell = \ell'$, then $\mathrm{F}_{\ell \leftarrow \ell'}[\![\mathrm{P}]\!](\mathrm{M}) = \mathrm{M}$; (ii) if $\ell \neq \ell'$ and there is not an atomic action from ℓ to ℓ', then $\mathrm{F}_{\ell \leftarrow \ell'}[\![\mathrm{P}]\!](\mathrm{M}) = \mathbb{M}$; and (iii) otherwise, there is an atomic action from ℓ to ℓ', then $\mathrm{F}_{\ell \leftarrow \ell'}[\![\mathrm{P}]\!](\mathrm{M})$ is the set of environments at point ℓ that guarantee the environments after executing the atomic action belong to M.

Specifically, for the simple language described in Figure 2, there are only two types of atomic actions, and for each of them we define an atomic backward transfer function. For an assignment ${}^{\ell_1}x := e^{\ell_2}$, the corresponding atomic backward transfer function $\mathrm{F}_{\ell_1 \leftarrow \ell_2}[\![\mathrm{P}]\!](\mathrm{M}) = \overleftarrow{\tau}[\![x := e]\!]\mathrm{M}$, which is defined as:

$$\overleftarrow{\tau}[\![x := e]\!]\mathrm{M} \triangleq \{\rho \in \mathbb{M} \mid \forall v \in [\![e]\!]\rho.\ \rho[x \mapsto v] \in \mathrm{M}\}.$$

Similarly, for a Boolean test ${}^{\ell_1} b^{\ell_2}$, the atomic backward transfer function $\mathrm{F}_{\ell_1 \leftarrow \ell_2}[\![\mathrm{P}]\!](\mathrm{M}) = \overleftarrow{\tau}[\![b]\!]\mathrm{M}$, which is defined as:

$$\overleftarrow{\tau}[\![b]\!]\mathrm{M} \triangleq \mathrm{M} \cup \{\rho \in \mathbb{M} \mid [\![b]\!]\rho = \{\mathrm{ff}\}\}.$$

Therefore, the definition of backward transfer function $\mathrm{F}^{\leftarrow}_{if}[\![\mathrm{P}]\!]$ can be rephrased into:

$$\begin{array}{rcll} \mathrm{F}^{\leftarrow}_{if}[\![\mathrm{P}]\!] & \in & (\mathbb{L} \mapsto \wp(\mathbb{M})) \mapsto (\mathbb{L} \mapsto \wp(\mathbb{M})) & \text{backward IF transfer function} \\ \mathrm{F}^{\leftarrow}_{if}[\![\mathrm{P}]\!]\mathsf{I} & \triangleq & \lambda \ell \in \mathbb{L}.\ \cap_{\ell' \in \mathbb{L}} \mathrm{F}_{\ell \leftarrow \ell'}[\![\mathrm{P}]\!](\mathsf{I}(\ell')). & \end{array}$$

Example 15 (Access Control, Continued). For the access control program in Figure 4, the transfer function $\mathrm{F}^{\leftarrow}_{if}[\![\mathrm{P}]\!]$ can be constructed by combining the atomic backward transfer functions: $\overleftarrow{\tau}[\![apv := 1]\!]$, $\overleftarrow{\tau}[\![i1 := [-1;2]]\!]$, $\overleftarrow{\tau}[\![apv := (i1 \leq 0)\,?\,-1 : apv]\!]$, $\overleftarrow{\tau}[\![i2 := [-1;2]]\!]$, $\overleftarrow{\tau}[\![apv := (apv \geq 1 \wedge i2 \leq 0)\,?\,-1 : apv]\!]$, $\overleftarrow{\tau}[\![typ := [1;2]]\!]$, and $\overleftarrow{\tau}[\![acs := apv \times typ]\!]$.

Suppose we are interested in inferring sufficient preconditions of the trace property "the access to o fails," a simple idea is to specify a postcondition $\mathsf{I}_{\mathrm{post}} \in \mathbb{L} \mapsto \wp(\mathbb{M})$ such that $\mathsf{I}_{\mathrm{post}}(\ell_8) = \{\rho \in \mathbb{M} \mid \rho(acs) <= 0\}$ and $\mathsf{I}_{\mathrm{post}}(\ell) = \mathbb{M}$ for $\ell \neq \ell_8$, and then compute the corresponding backward accessibility semantics $\mathcal{S}^{\leftarrow}_{if}[\![\mathrm{P}]\!](\mathsf{I}_{\mathrm{post}})$. However, such a result is too imprecise. Take the semantics at the point ℓ_7 as an example: $\mathcal{S}^{\leftarrow}_{if}[\![\mathrm{P}]\!](\mathsf{I}_{\mathrm{post}})\ell_7 = \mathsf{I}_{\mathrm{post}}(\ell_7) \cap \mathrm{F}_{\ell_7 \leftarrow \ell_8}[\![\mathrm{P}]\!](\mathsf{I}_{\mathrm{post}}(\ell_8)) = \mathbb{M} \cap \overleftarrow{\tau}[\![acs := apv \times typ]\!](\{\rho \in \mathbb{M} \mid \rho(acs) <= 0\}) = \{\rho \in \mathbb{M} \mid (\rho(apv) <= 0 \wedge \rho(typ) >= 0) \vee (\rho(apv) >= 0 \wedge \rho(typ) <= 0)\}$. This semantics does provide correct sufficient preconditions of "the access to o fails," but it is not precise enough, since the value of *typ* is never zero or negative at point ℓ_7 in the real executions.

Table 3. Concrete Backward Impossible Failure Accessibility Semantics for the Access Control Program

ℓ	$\mathcal{S}^{\leftarrow}_{if}[\![\mathrm{P}]\!](\mathrm{I}'_{\mathrm{post}})\ell$
ℓ_1	$\emptyset$
ℓ_2	$\emptyset$
ℓ_3	$\{\rho \in \mathbb{M} \mid apv = 1 \wedge i1 \in \{-1, 0\}\}$
ℓ_4	$\{\rho \in \mathbb{M} \mid apv = -1 \wedge i1 \in \{-1, 0\}\}$
ℓ_5	$\{\rho \in \mathbb{M} \mid apv = 1 \wedge i1 \in \{1, 2\} \wedge i2 \in \{-1, 0\}\}$ $\cup \{\rho \in \mathbb{M} \mid apv = -1 \wedge i1 \in \{-1, 0\} \wedge i2 \in \{-1, 0, 1, 2\}\}$
ℓ_6	$\{\rho \in \mathbb{M} \mid apv = -1 \wedge i1 \in \{1, 2\} \wedge i2 \in \{-1, 0\}\}$ $\cup \{\rho \in \mathbb{M} \mid apv = -1 \wedge i1 \in \{-1, 0\} \wedge i2 \in \{-1, 0, 1, 2\}\}$
ℓ_7	$\{\rho \in \mathbb{M} \mid apv = -1 \wedge i1 \in \{1, 2\} \wedge i2 \in \{-1, 0\} \wedge typ \in \{1, 2\}\}$ $\cup \{\rho \in \mathbb{M} \mid apv = -1 \wedge i1 \in \{-1, 0\} \wedge i2 \in \{-1, 0, 1, 2\} \wedge typ \in \{1, 2\}\}$
ℓ_8	$\{\rho \in \mathbb{M} \mid apv = -1 \wedge i1 \in \{1, 2\} \wedge i2 \in \{-1, 0\} \wedge typ = 1 \wedge acs = -1\}$ $\cup \{\rho \in \mathbb{M} \mid apv = -1 \wedge i1 \in \{1, 2\} \wedge i2 \in \{-1, 0\} \wedge typ = 2 \wedge acs = -2\}$ $\cup \{\rho \in \mathbb{M} \mid apv = -1 \wedge i1 \in \{-1, 0\} \wedge i2 \in \{-1, 0, 1, 2\} \wedge typ = 1 \wedge acs = -1\}$ $\cup \{\rho \in \mathbb{M} \mid apv = -1 \wedge i1 \in \{-1, 0\} \wedge i2 \in \{-1, 0, 1, 2\} \wedge typ = 2 \wedge acs = -2\}$

To get a more precise result, the specified postcondition $\mathrm{I}_{\mathrm{post}}$ can be refined by the intersection with the forward reachability semantics $\mathcal{S}^{\rightarrow}_{ps}[\![\mathrm{P}]\!](\mathrm{I}_{\mathrm{pre}})$ computed in Example 13, i.e., we define $\mathrm{I}'_{\mathrm{post}} = \mathrm{I}_{\mathrm{post}} \dot{\cap} \mathcal{S}^{\rightarrow}_{ps}[\![\mathrm{P}]\!](\mathrm{I}_{\mathrm{pre}})$, and the semantics $\mathcal{S}^{\leftarrow}_{if}[\![\mathrm{P}]\!](\mathrm{I}'_{\mathrm{post}})$ would be more precise, whose result is in Table 3, and the constraints on environment like "$\rho(apv) = 1$" is written as "$apv = 1$" for short. It is not hard to see that: At the point ℓ_1 or ℓ_2, there is no sufficient precondition that can guarantee the property "the access to o fails"; beginning from the point ℓ_3, the negative or zero value of *i1* guarantees "access failure"; and beginning from the point ℓ_5, the negative or zero value of *i2* guarantees "access failure."

4.3.2 Under-approximating Abstract Backward Impossible Failure Accessibility Analysis. Similar to the forward reachability semantics, the backward impossible failure accessibility semantics may be not computable in the concrete, hence it is necessary to reason on the abstract domain instead. Although classic abstract domains come with abstract transfer functions (operators) for both forward and backward analyses, these functions are over-approximating and are suitable only for inferring invariants (i.e., reachability semantics) or necessary preconditions, but not for inferring sufficient preconditions. The reason comes from that an over-approximation of the tightest program invariant (respectively, the strongest necessary precondition) is still an invariant (respectively, a necessary precondition), but an over-approximation of the weakest sufficient precondition is not a sufficient precondition anymore (which will be discussed later in Section 4.3.3), thus under-approximations are needed instead to preserve the soundness for inferring sufficient preconditions. To solve this problem, Miné [60, 61] presents under-approximating abstract operators (including a dual widening) for the classic interval/octagon/polyhedron domain, which makes inferring sufficient preconditions directly by under-approximating backward analysis possible. Other attempts to infer sufficient preconditions include References [7, 44, 50], but none of them can directly work on the classic numeric abstract domains.

In this section, we briefly summarize the framework of an under-approximating abstract backward impossible failure accessibility analysis and refer to References [60, 61] and Banal analyzer [55] for the details of implementing the under-approximating abstract operators (including a dual widening).

(1) Under-approximating Abstract Backward Transfer Function. For the transfer function $\mathrm{F}^{\leftarrow}_{if}[\![\mathrm{P}]\!] \in (\mathbb{L} \mapsto \wp(\mathbb{M})) \mapsto (\mathbb{L} \mapsto \wp(\mathbb{M}))$ on the concrete invariant domain, we need to construct the corresponding abstract backward transfer function $\check{\mathrm{F}}^{\sharp}_{\overleftarrow{if}}[\![\mathrm{P}]\!] \in (\mathbb{L} \mapsto \mathcal{D}^{\sharp}_{\mathbb{M}}) \mapsto (\mathbb{L} \mapsto \mathcal{D}^{\sharp}_{\mathbb{M}})$ (the symbol $\check{}$ denotes under-approximations), which satisfies the following soundness condition:

$$\forall \mathrm{I}^{\sharp} \in \mathbb{L} \mapsto \mathcal{D}^{\sharp}_{\mathbb{M}}.\ \dot{\gamma}_{\mathbb{M}} \circ \check{\mathrm{F}}^{\sharp}_{\overleftarrow{if}}[\![\mathrm{P}]\!](\mathrm{I}^{\sharp}) \mathrel{\dot{\subseteq}} \mathrm{F}^{\leftarrow}_{if}[\![\mathrm{P}]\!] \circ \dot{\gamma}_{\mathbb{M}}(\mathrm{I}^{\sharp}). \tag{10}$$

Since $\mathrm{F}^{\leftarrow}_{if}[\![\mathrm{P}]\!]\mathrm{I}$ is defined by combining atomic backward transfer function $\mathrm{F}_{\ell\leftarrow\ell'}[\![\mathrm{P}]\!]$ together (i.e., $\mathrm{F}^{\leftarrow}_{if}[\![\mathrm{P}]\!]\mathrm{I} \triangleq \lambda\ell \in \mathbb{L}.\ \cap_{\ell'\in\mathbb{L}} \mathrm{F}_{\ell\leftarrow\ell'}[\![\mathrm{P}]\!](\mathrm{I}(\ell'))$), it is necessary to build the under-approximating versions $\check{\mathrm{F}}^{\sharp}_{\ell\leftarrow\ell'}[\![\mathrm{P}]\!] \in \mathcal{D}^{\sharp}_{\mathbb{M}} \mapsto \mathcal{D}^{\sharp}_{\mathbb{M}}$ for atomic backward transfer functions $\mathrm{F}_{\ell\leftarrow\ell'}[\![\mathrm{P}]\!] \in \wp(\mathbb{M}) \mapsto \wp(\mathbb{M})$, such that condition (11) holds:

$$\forall \mathrm{M}^{\sharp} \in \mathcal{D}^{\sharp}_{\mathbb{M}}.\ \gamma_{\mathbb{M}} \circ \check{\mathrm{F}}^{\sharp}_{\ell\leftarrow\ell'}[\![\mathrm{P}]\!](\mathrm{M}^{\sharp}) \ \subseteq\ \mathrm{F}_{\ell\leftarrow\ell'}[\![\mathrm{P}]\!] \circ \gamma_{\mathbb{M}}(\mathrm{M}^{\sharp}). \tag{11}$$

To satisfy the soundness condition (11), we design $\check{\mathrm{F}}^{\sharp}_{\ell\leftarrow\ell'}[\![\mathrm{P}]\!]$ such that: (i) if $\ell = \ell'$, then $\check{\mathrm{F}}^{\sharp}_{\ell\leftarrow\ell'}[\![\mathrm{P}]\!](\mathrm{M}^{\sharp}) = \mathrm{M}^{\sharp}$; (ii) if $\ell \neq \ell'$ and there is not an atomic action from ℓ to ℓ', then $\check{\mathrm{F}}^{\sharp}_{\ell\leftarrow\ell'}[\![\mathrm{P}]\!](\mathrm{M}^{\sharp}) = \top^{\sharp}_{\mathbb{M}}$; and (iii) otherwise, there is an atomic action from ℓ to ℓ', then $\check{\mathrm{F}}^{\sharp}_{\ell\leftarrow\ell'}[\![\mathrm{P}]\!](\mathrm{M}^{\sharp})$ is an abstract environment element in $\mathcal{D}^{\sharp}_{\mathbb{M}}$ that guarantees $\mathrm{M}^{\sharp}$ to hold after executing the atomic action. It is obvious that the case (iii) is the difficult one, and fortunately for the atomic actions of assignments and Boolean tests in the interval/polyhedron/octagon domain, Miné has proposed the corresponding under-approximating atomic backward transfer function $\overleftarrow{\tau}^{\sharp}[\![\chi := e]\!]\mathrm{M}^{\sharp}$ and $\overleftarrow{\tau}^{\sharp}[\![b]\!]\mathrm{M}^{\sharp}$, which satisfies the soundness condition. Details are given in Sections 3.2–3.4 of Reference [61]. Now we can build the backward transfer function $\check{\mathrm{F}}^{\sharp}_{\overleftarrow{if}}[\![\mathrm{P}]\!]$ by the following definition:

$$\begin{array}{lcll} \check{\mathrm{F}}^{\sharp}_{\overleftarrow{if}}[\![\mathrm{P}]\!] & \in & \left(\mathbb{L} \mapsto \mathcal{D}^{\sharp}_{\mathbb{M}}\right) \mapsto \left(\mathbb{L} \mapsto \mathcal{D}^{\sharp}_{\mathbb{M}}\right) & \text{under-approximating backward IF function} \\ \check{\mathrm{F}}^{\sharp}_{\overleftarrow{if}}[\![\mathrm{P}]\!]\mathrm{I}^{\sharp} & \triangleq & \lambda\ell \in \mathbb{L}.\ \sqcap^{\sharp}_{\mathbb{M}\ \ell'\in\mathbb{L}} \check{\mathrm{F}}^{\sharp}_{\ell\leftarrow\ell'}[\![\mathrm{P}]\!](\mathrm{I}^{\sharp}(\ell')). & \end{array}$$

The backward transfer function $\check{\mathrm{F}}^{\sharp}_{\overleftarrow{if}}[\![\mathrm{P}]\!]$ satisfies the condition (10), and its greatest fixpoint soundly under-approximates the concrete backward impossible failure accessibility semantics:

$$\forall \mathrm{I}^{\sharp}_{\mathrm{post}} \in \mathbb{L} \mapsto \mathcal{D}^{\sharp}_{\mathbb{M}}.\ \dot{\gamma}_{\mathbb{M}}\left(\mathrm{gfp}^{\dot{\subseteq}}_{\mathrm{I}^{\sharp}_{\mathrm{post}}} \check{\mathrm{F}}^{\sharp}_{\overleftarrow{if}}[\![\mathrm{P}]\!]\right) \mathrel{\dot{\subseteq}} \mathrm{gfp}^{\dot{\subseteq}}_{\dot{\gamma}_{\mathbb{M}}\left(\mathrm{I}^{\sharp}_{\mathrm{post}}\right)} \mathrm{F}^{\leftarrow}_{if}[\![\mathrm{P}]\!] = \mathcal{S}^{\leftarrow}_{if}[\![\mathrm{P}]\!]\left(\dot{\gamma}_{\mathbb{M}}\left(\mathrm{I}^{\sharp}_{\mathrm{post}}\right)\right).$$

(2) Dual Widening. The iteration of the above defined $\check{\mathrm{F}}^{\sharp}_{\overleftarrow{if}}[\![\mathrm{P}]\!]$ may not converge in finite time, since there may exist infinite decreasing chains in the abstract environment domain (e.g., intervals, polyhedra, octagons). To address this problem, we need a dual widening operator $\underline{\triangledown}_{\mathbb{I}} \in (\mathbb{L} \mapsto \mathcal{D}^{\sharp}_{\mathbb{M}}) \times (\mathbb{L} \mapsto \mathcal{D}^{\sharp}_{\mathbb{M}}) \mapsto (\mathbb{L} \mapsto \mathcal{D}^{\sharp}_{\mathbb{M}})$ on the abstract invariant domain, which obeys the following soundness and termination conditions:

(i) $\forall x^{\sharp}, y^{\sharp} \in (\mathbb{L} \mapsto \mathcal{D}^{\sharp}_{\mathbb{M}}).\ \dot{\gamma}_{\mathbb{M}}(x^{\sharp} \mathbin{\underline{\triangledown}_{\mathbb{I}}} y^{\sharp}) \mathrel{\dot{\subseteq}} \dot{\gamma}_{\mathbb{M}}(x^{\sharp}) \mathbin{\dot{\cap}} \dot{\gamma}_{\mathbb{M}}(y^{\sharp})$;

(ii) for any sequence $(x^{\sharp}_i)_{i\in\mathbb{N}}$, the sequence $(y^{\sharp}_i)_{i\in\mathbb{N}}$ defined as $y^{\sharp}_0 = x^{\sharp}_0$ and $\forall i \in \mathbb{N}.\ y^{\sharp}_{i+1} = y^{\sharp}_i \mathbin{\underline{\triangledown}_{\mathbb{I}}} x^{\sharp}_{i+1}$ converges in finite time.

Notice that the above soundness condition is different from the one for classic widening $\triangledown_{\mathbb{I}}$ in the forward reachability analysis.

In Section 3.5 of Reference [61], Miné has proposed a so-called "lower widening" operator $\underline{\triangledown} \in \mathcal{D}^\sharp_{\mathbb{M}} \times \mathcal{D}^\sharp_{\mathbb{M}} \mapsto \mathcal{D}^\sharp_{\mathbb{M}}$ for the interval/polyhedron/octagon domain. Correspondingly, we define $\underline{\triangledown}_{\mathbb{I}}$ as the pointwise version of $\underline{\triangledown}$ (i.e., $\mathsf{I}^\sharp \mathbin{\underline{\triangledown}_{\mathbb{I}}} \mathsf{I}^{\sharp\prime} \triangleq \lambda \ell \in \mathbb{L}.\ \mathsf{I}^\sharp(\ell) \mathbin{\underline{\triangledown}} \mathsf{I}^{\sharp\prime}(\ell)$), and it can satisfy both the soundness and the termination condition above.

(3) Under-approximating Abstract Backward Accessibility Semantics. Given a postcondition specified as $\mathsf{I}^\sharp_{\text{post}} \in \mathbb{L} \mapsto \mathcal{D}^\sharp_{\mathbb{M}}$, the corresponding concrete backward impossible failure accessibility semantics $\mathcal{S}^{\overleftarrow{if}}[\![\mathrm{P}]\!](\dot{\gamma}_{\mathbb{M}}(\mathsf{I}^\sharp_{\text{post}}))$ is the greatest fixpoint of function $\mathrm{F}^{\overleftarrow{if}}[\![\mathrm{P}]\!]$, which is less than or equal to $\dot{\gamma}_{\mathbb{M}}(\mathsf{I}^\sharp_{\text{post}})$. That is to say, $\mathcal{S}^{\overleftarrow{if}}[\![\mathrm{P}]\!](\dot{\gamma}_{\mathbb{M}}(\mathsf{I}^\sharp_{\text{post}})) = \mathrm{gfp}^{\dot{\subseteq}}_{\dot{\gamma}_{\mathbb{M}}(\mathsf{I}^\sharp_{\text{post}})} \mathrm{F}^{\overleftarrow{if}}[\![\mathrm{P}]\!]$, and it can be soundly under-approximated by the limit of a ultimately stationary sequence $(\mathsf{I}^\sharp_i)_{i\in\mathbb{N}}$, where $\mathsf{I}^\sharp_0 = \mathsf{I}^\sharp_{\text{post}}$ and $\forall i \in \mathbb{N}.\ \mathsf{I}^\sharp_{i+1} = \mathsf{I}^\sharp_i \mathbin{\underline{\triangledown}_{\mathbb{I}}} \check{\mathrm{F}}^{\overleftarrow{if}}[\![\mathrm{P}]\!](\mathsf{I}^\sharp_i)$.

$$\forall \mathsf{I}^\sharp_{\text{post}} \in \mathbb{L} \mapsto \mathcal{D}^\sharp_{\mathbb{M}}.\ \dot{\gamma}_{\mathbb{M}}\left(\lim_{\mathsf{I}^\sharp_{\text{post}}} \lambda \mathsf{I}^\sharp.\ \mathsf{I}^\sharp \mathbin{\underline{\triangledown}_{\mathbb{I}}} \check{\mathrm{F}}^{\overleftarrow{if}}[\![\mathrm{P}]\!](\mathsf{I}^\sharp)\right) \mathrel{\dot{\subseteq}} \mathrm{gfp}^{\dot{\subseteq}}_{\dot{\gamma}_{\mathbb{M}}(\mathsf{I}^\sharp_{\text{post}})} \mathrm{F}^{\overleftarrow{if}}[\![\mathrm{P}]\!]. \tag{12}$$

In the rest of this article, the under-approximating abstract backward impossible failure accessibility semantics $\check{\mathcal{S}}^{\overleftarrow{if}}[\![\mathrm{P}]\!]$ refers to the following definition, which computes a sound under-approximation of the concrete backward impossible failure accessibility semantics and can automatically infer the sufficient precondition of any given postcondition in finite time.

$$\check{\mathcal{S}}^{\overleftarrow{if}}[\![\mathrm{P}]\!] \in \left(\mathbb{L} \mapsto \mathcal{D}^\sharp_{\mathbb{M}}\right) \mapsto \left(\mathbb{L} \mapsto \mathcal{D}^\sharp_{\mathbb{M}}\right) \qquad \text{under-approximating backward IF semantics}$$

$$\check{\mathcal{S}}^{\overleftarrow{if}}[\![\mathrm{P}]\!](\mathsf{I}^\sharp_{\text{post}}) \triangleq \lim_{\mathsf{I}^\sharp_{\text{post}}} \lambda \mathsf{I}^\sharp.\ \mathsf{I}^\sharp \mathbin{\underline{\triangledown}_{\mathbb{I}}} \check{\mathrm{F}}^{\overleftarrow{if}}[\![\mathrm{P}]\!](\mathsf{I}^\sharp)$$

Example 16 (Access Control, Continued). Consider the access control program in Figure 4 again. We are interested in inferring the sufficient preconditions of the trace property "the access to o fails." Suppose the abstract environment domain $\mathcal{D}^\sharp_{\mathbb{M}}$ is chosen as the interval domain, then "the access to o fails" can be expressed by an abstract postcondition $\mathsf{I}^\sharp_{\text{post}} \in \mathbb{L} \mapsto \mathcal{D}^\sharp_{\mathbb{M}}$ such that $\mathsf{I}^\sharp_{\text{post}}(\ell_8) = acs \in [-\infty; 0]$ and $\mathsf{I}^\sharp_{\text{post}}(\ell) = \top^\sharp_{\mathbb{M}}$ for $\ell \neq \ell_8$.

Like in the Example 15, the postcondition $\mathsf{I}^\sharp_{\text{post}}$ can be refined by the intersection with the abstract forward reachability semantics $\mathcal{S}^\sharp_{\overrightarrow{ps}}[\![\mathrm{P}]\!](\mathsf{I}^\sharp_{\text{pre}})$ from the Table 2, and we get $\mathsf{I}^{\sharp\prime}_{\text{post}} = \mathsf{I}^\sharp_{\text{post}} \mathbin{\dot{\sqcap}^\sharp_{\mathbb{M}}} \mathcal{S}^\sharp_{\overrightarrow{ps}}[\![\mathrm{P}]\!](\mathsf{I}^\sharp_{\text{pre}})$, which is listed in the Table 4.

From the above refined abstract postcondition $\mathsf{I}^{\sharp\prime}_{\text{post}}$, there are two possible results of the backward impossible failure accessibility analysis $\check{\mathcal{S}}^{\overleftarrow{if}}[\![\mathrm{P}]\!](\mathsf{I}^{\sharp\prime}_{\text{post}})$, and they are, respectively, displayed in Table 5 and Table 6 (in which the interesting part is emphasized in a bold font).

The difference between these two possible results comes from the assignment "$apv := (apv \geq 1 \wedge i2 \leq 0)\ ?\ -1 : apv$" at the point ℓ_5. To guarantee that "$apv \leq 0$" at point ℓ_6, we have two possible choices: either "$apv \geq 1 \wedge i2 \leq 0$," or "$apv \leq 0$" at point ℓ_5. Since $\check{\mathcal{S}}^{\overleftarrow{if}}[\![\mathrm{P}]\!](\mathsf{I}^{\sharp\prime}_{\text{post}})$ is an under-approximation, we cannot join two cases together like in the over-approximating forward reachability analysis. Instead, we keep one case and discard the other case (e.g., the Banal analyzer adopts the former choice and produces results as in Table 5).

Alternatively, we could use the disjunctive completion [21] and maintain the abstract environment elements (i.e., sufficient preconditions) from both two tables. In this example, the disjunctive

Table 4. Refined Abstract Postcondition for "the Access to o Fails"

ℓ	$\mathsf{I}^{\sharp\prime}_{\text{post}}(\ell)$
ℓ_1	$\top^{\sharp}_{\mathbb{M}}$
ℓ_2	$apv \in [1;1] \wedge i1 \in [-\infty;\infty] \wedge i2 \in [-\infty;\infty] \wedge typ \in [-\infty;\infty] \wedge acs \in [-\infty;\infty]$
ℓ_3	$apv \in [1;1] \wedge i1 \in [-1;2] \wedge i2 \in [-\infty;\infty] \wedge typ \in [-\infty;\infty] \wedge acs \in [-\infty;\infty]$
ℓ_4	$apv \in [-1;1] \wedge i1 \in [-1;2] \wedge i2 \in [-\infty;\infty] \wedge typ \in [-\infty;\infty] \wedge acs \in [-\infty;\infty]$
ℓ_5	$apv \in [-1;1] \wedge i1 \in [-1;2] \wedge i2 \in [-1;2] \wedge typ \in [-\infty;\infty] \wedge acs \in [-\infty;\infty]$
ℓ_6	$apv \in [-1;1] \wedge i1 \in [-1;2] \wedge i2 \in [-1;2] \wedge typ \in [-\infty;\infty] \wedge acs \in [-\infty;\infty]$
ℓ_7	$apv \in [-1;1] \wedge i1 \in [-1;2] \wedge i2 \in [-1;2] \wedge typ \in [1;2] \wedge acs \in [-\infty;\infty]$
ℓ_8	$apv \in [-1;1] \wedge i1 \in [-1;2] \wedge i2 \in [-1;2] \wedge typ \in [1;2] \wedge acs \in [\mathbf{-2};\mathbf{0}]$

Table 5. The Under-approximating Abstract Backward Impossible Failure Accessibility Semantics (Option 1) for "the Access to o Fails"

ℓ	$\check{\mathcal{S}}^{\sharp}_{\overleftarrow{if}}[\![\mathrm{P}]\!](\mathsf{I}^{\sharp\prime}_{\text{post}})\ell$
ℓ_1	$\bot^{\sharp}_{\mathbb{M}}$
ℓ_2	$\bot^{\sharp}_{\mathbb{M}}$
ℓ_3	$\bot^{\sharp}_{\mathbb{M}}$
ℓ_4	$\bot^{\sharp}_{\mathbb{M}}$
ℓ_5	$apv \in [\mathbf{1};\mathbf{1}] \wedge i1 \in [-1;2] \wedge i2 \in [\mathbf{-1};\mathbf{0}] \wedge typ \in [-\infty;\infty] \wedge acs \in [-\infty;\infty]$
ℓ_6	$apv \in [\mathbf{-1};\mathbf{0}] \wedge i1 \in [-1;2] \wedge i2 \in [-1;2] \wedge typ \in [-\infty;\infty] \wedge acs \in [-\infty;\infty]$
ℓ_7	$apv \in [\mathbf{-1};\mathbf{0}] \wedge i1 \in [-1;2] \wedge i2 \in [-1;2] \wedge typ \in [1;2] \wedge acs \in [-\infty;\infty]$
ℓ_8	$apv \in [-1;1] \wedge i1 \in [-1;2] \wedge i2 \in [-1;2] \wedge typ \in [1;2] \wedge acs \in [\mathbf{-2};\mathbf{0}]$

Table 6. The Under-approximating Abstract Backward Impossible Failure Accessibility Semantics (Option 2) for "the Access to o Fails"

ℓ	$\check{\mathcal{S}}^{\sharp}_{\overleftarrow{if}}[\![\mathrm{P}]\!](\mathsf{I}^{\sharp\prime}_{\text{post}})\ell$
ℓ_1	$\bot^{\sharp}_{\mathbb{M}}$
ℓ_2	$\bot^{\sharp}_{\mathbb{M}}$
ℓ_3	$apv \in [1;1] \wedge i1 \in [\mathbf{-1};\mathbf{0}] \wedge i2 \in [-\infty;\infty] \wedge typ \in [-\infty;\infty] \wedge acs \in [-\infty;\infty]$
ℓ_4	$apv \in [\mathbf{-1};\mathbf{0}] \wedge i1 \in [-1;2] \wedge i2 \in [-\infty;\infty] \wedge typ \in [-\infty;\infty] \wedge acs \in [-\infty;\infty]$
ℓ_5	$apv \in [\mathbf{-1};\mathbf{0}] \wedge i1 \in [-1;2] \wedge i2 \in [-1;2] \wedge typ \in [-\infty;\infty] \wedge acs \in [-\infty;\infty]$
ℓ_6	$apv \in [\mathbf{-1};\mathbf{0}] \wedge i1 \in [-1;2] \wedge i2 \in [-1;2] \wedge typ \in [-\infty;\infty] \wedge acs \in [-\infty;\infty]$
ℓ_7	$apv \in [\mathbf{-1};\mathbf{0}] \wedge i1 \in [-1;2] \wedge i2 \in [-1;2] \wedge typ \in [1;2] \wedge acs \in [-\infty;\infty]$
ℓ_8	$apv \in [-1;1] \wedge i1 \in [-1;2] \wedge i2 \in [-1;2] \wedge typ \in [1;2] \wedge acs \in [\mathbf{-2};\mathbf{0}]$

completion could provide us with the exact backward impossible failure accessibility semantics and its cost is not too heavy, because we need to keep a disjunction of two abstract environment elements only at the point ℓ_5, while the abstract environment elements at other points are either the same from two tables or the bottom in one table (which can be omitted). To distinguish from the over-approximating analysis introduced later in Section 4.3.3, here we adopt the result (from the Banal analyzer) in Table 5 as an under-approximation.

Therefore, we have successfully inferred some sufficient preconditions of "the Access to o Fails": "$acs \in [-2; 0]$" at ℓ_8, "$apv \in [-1; 0]$" at ℓ_7 and ℓ_6, and "$apv \in [1; 1] \wedge i2 \in [-1; 0]$" at ℓ_5, which implies that the zero or negative value of $i2$ (i.e., the input from 2nd admin) guarantees the access failure.

4.3.3 Over-approximating Abstract Backward Impossible Failure Accessibility Analysis. Besides the under-approximating backward analysis above, we design an over-approximating abstract backward impossible failure accessibility analysis as well, which computes an over-approximation of the set of states from which all the executions must satisfy the given postcondition $\mathsf{I}^\sharp_{\text{post}}$.

Such an over-approximation is neither a sufficient precondition for $\mathsf{I}^\sharp_{\text{post}}$ to hold, nor a necessary precondition, due to the possible non-determinism of the program. Thus, it may seem to be not of practical use. However, instead of directly using such an over-approximating abstract backward impossible failure accessibility semantics in the responsibility analysis, we intend to utilize its set-complement as partitioning directives (which will be further discussed in Section 7), and it represents a set of states from which there must exist at least one concrete execution trace that fails the postcondition $\mathsf{I}^\sharp_{\text{post}}$. This may seem to be counter-intuitive at first sight, since most abstract domains (e.g., intervals, octagons, polyhedra) do not support complements. For instance, the complement of a polyhedron is a disjunction of affine inequalities, which cannot be expressed by a single polyhedron. However, it would not be a problem for our responsibility analysis, since we do not require to represent the complement set by a single abstract environment element. Instead, we could keep multiple partitioning directives at every program point. Take the complement of a polyhedron as an example: Each affine inequality (or the heuristically selected ones when the number of affine inequalities exceeds a threshold) can be used as a partitioning directive in the responsibility analysis.

In the following, we formalize the framework of over-approximating backward impossible failure accessibility analysis, which essentially corresponds to an over-approximating version of Section 3 of Reference [61] and Section 4.3.2 of this article. More precisely, it consists of the over-approximating backward transfer functions (e.g., for the Boolean tests and assignments in our simple programming language) and a narrowing operator that over-approximates meets and enforces termination.

(1) Over-approximating Abstract Backward Transfer Function. Here, we need an over-approximating abstract backward transfer function $\hat{\mathrm{F}}^\sharp_{\overleftarrow{if}}[\![\mathrm{P}]\!] \in (\mathbb{L} \mapsto \mathcal{D}^\sharp_{\mathbb{M}}) \mapsto (\mathbb{L} \mapsto \mathcal{D}^\sharp_{\mathbb{M}})$ (the symbol $\hat{}$ denotes over-approximations) that satisfies the following soundness condition (13):

$$\forall \mathsf{I}^\sharp \in \mathbb{L} \mapsto \mathcal{D}^\sharp_{\mathbb{M}}.\ \mathrm{F}_{\overleftarrow{if}}[\![\mathrm{P}]\!] \circ \dot{\gamma}_{\mathbb{M}}(\mathsf{I}^\sharp) \mathrel{\dot{\subseteq}} \dot{\gamma}_{\mathbb{M}} \circ \hat{\mathrm{F}}^\sharp_{\overleftarrow{if}}[\![\mathrm{P}]\!](\mathsf{I}^\sharp). \tag{13}$$

Like $\check{\mathrm{F}}^\sharp_{\overleftarrow{if}}[\![\mathrm{P}]\!]$ is defined by the combination of $\check{\mathrm{F}}^\sharp_{\ell\leftarrow\ell'}[\![\mathrm{P}]\!]$, the over-approximating version $\hat{\mathrm{F}}^\sharp_{\overleftarrow{if}}[\![\mathrm{P}]\!]$ can be defined by combining $\hat{\mathrm{F}}^\sharp_{\ell\leftarrow\ell'}[\![\mathrm{P}]\!] \in \mathcal{D}^\sharp_{\mathbb{M}} \mapsto \mathcal{D}^\sharp_{\mathbb{M}}$:

$$\begin{array}{lcll} \hat{\mathrm{F}}^\sharp_{\overleftarrow{if}}[\![\mathrm{P}]\!] & \in & \left(\mathbb{L} \mapsto \mathcal{D}^\sharp_{\mathbb{M}}\right) \mapsto \left(\mathbb{L} \mapsto \mathcal{D}^\sharp_{\mathbb{M}}\right) & \text{over-approximating backward IF function} \\ \hat{\mathrm{F}}^\sharp_{\overleftarrow{if}}[\![\mathrm{P}]\!]\mathsf{I}^\sharp & \triangleq & \lambda \ell \in \mathbb{L}.\ \sqcap^\sharp_{\mathbb{M}\ \ell' \in \mathbb{L}} \hat{\mathrm{F}}^\sharp_{\ell\leftarrow\ell'}[\![\mathrm{P}]\!](\mathsf{I}^\sharp(\ell')), & \end{array}$$

where the meet $\sqcap^\sharp_{\mathbb{M}}$ is exact and $\hat{\mathrm{F}}^\sharp_{\ell\leftarrow\ell'}[\![\mathrm{P}]\!]$ needs to satisfy the condition (14):

$$\forall \mathrm{M}^\sharp \in \mathcal{D}^\sharp_{\mathbb{M}}.\ \mathrm{F}_{\ell\leftarrow\ell'}[\![\mathrm{P}]\!] \circ \gamma_{\mathbb{M}}(\mathrm{M}^\sharp) \subseteq \gamma_{\mathbb{M}} \circ \hat{\mathrm{F}}^\sharp_{\ell\leftarrow\ell'}[\![\mathrm{P}]\!](\mathrm{M}^\sharp). \tag{14}$$

Similar to the definition of $\check{\mathrm{F}}^{\sharp}_{\ell\leftarrow\ell'}[\![\mathrm{P}]\!]$, here $\hat{\mathrm{F}}^{\sharp}_{\ell\leftarrow\ell'}[\![\mathrm{P}]\!]$ is defined such that: (i) if $\ell = \ell'$, then $\hat{\mathrm{F}}^{\sharp}_{\ell\leftarrow\ell'}[\![\mathrm{P}]\!](\mathrm{M}^{\sharp}) = \mathrm{M}^{\sharp}$; (ii) if $\ell \neq \ell'$ and there is not an atomic action from ℓ to ℓ', then $\hat{\mathrm{F}}^{\sharp}_{\ell\leftarrow\ell'}[\![\mathrm{P}]\!](\mathrm{M}^{\sharp}) = \top^{\sharp}_{\mathbb{M}}$; and (iii) otherwise, there is an atomic action from ℓ to ℓ', then $\hat{\mathrm{F}}^{\sharp}_{\ell\leftarrow\ell'}[\![\mathrm{P}]\!](\mathrm{M}^{\sharp})$ over-approximates the sufficient precondition that guarantees $\mathrm{M}^{\sharp}$ to hold after executing the atomic action.

Among the above three cases, (iii) is the difficult one. In the following, we take the simple programming language in Figure 2 as an example, mimic Section 3 of Reference [61], and discuss how $\hat{\mathrm{F}}^{\sharp}_{\ell\leftarrow\ell'}[\![\mathrm{P}]\!]$ is implemented for atomic actions of form ${}^{\ell_1}a^{\ell_2}$ (e.g., Boolean tests ${}^{\ell_1}b^{\ell_2}$, assignments ${}^{\ell_1}\mathcal{X} := e^{\ell_2}$), where $\hat{\mathrm{F}}^{\sharp}_{\ell_1\leftarrow\ell_2}[\![\mathrm{P}]\!](\mathrm{M}^{\sharp}) \triangleq \hat{\overleftarrow{\tau}}^{\sharp}[\![a]\!]\mathrm{M}^{\sharp}$ such that $\forall \mathrm{M}^{\sharp} \in \mathcal{D}^{\sharp}_{\mathbb{M}}.\ \overleftarrow{\tau}[\![a]\!] \circ \gamma_{\mathbb{M}}(\mathrm{M}^{\sharp}) \subseteq \gamma_{\mathbb{M}} \circ \hat{\overleftarrow{\tau}}^{\sharp}[\![a]\!](\mathrm{M}^{\sharp})$.

(1.1) Boolean tests (guards).

Affine guards. First, we consider the polyhedron domain, and the guard is of form $\vec{a}\cdot\vec{\mathcal{X}} \geq b$ such that it can be exactly represented by polyhedra. In this case, the concrete backward transfer function can be rephrased into:

$$\overleftarrow{\tau}[\![\vec{a}\cdot\vec{\mathcal{X}} \geq b]\!]\mathrm{M} = \mathrm{M} \cup \{\rho \in \mathbb{M} \mid [\![\vec{a}\cdot\vec{\mathcal{X}} \geq b]\!]\rho = \{\mathrm{ff}\}\} = \mathrm{M} \cup \{\rho \in \mathbb{M} \mid \vec{a}\cdot\vec{\rho} < b\},$$

where $\vec{\rho}$ denotes the vector of variable values in the environment ρ.

To over-approximate $\overleftarrow{\tau}[\![\vec{a}\cdot\vec{\mathcal{X}} \geq b]\!]$, we define the corresponding abstract transfer function $\hat{\overleftarrow{\tau}}^{\sharp}[\![\vec{a}\cdot\vec{\mathcal{X}} \geq b]\!] \in \mathcal{D}^{\sharp}_{\mathbb{M}} \mapsto \mathcal{D}^{\sharp}_{\mathbb{M}}$ as:

$$\hat{\overleftarrow{\tau}}^{\sharp}[\![\vec{a}\cdot\vec{\mathcal{X}} \geq b]\!]\mathrm{M}^{\sharp} \triangleq \mathrm{M}^{\sharp} \sqcup^{\sharp}_{\mathbb{M}} \vec{a}\cdot\vec{x} < b. \tag{15}$$

Since $\sqcup^{\sharp}_{\mathbb{M}}$ soundly approximates the concrete join operator $\cup$, it is easy to see the soundness condition (14) holds. Moreover, if we use the disjunctive completion [21], then both $\mathrm{M}^{\sharp}$ and $\vec{a}\cdot\vec{x} < b$ can be kept without the join, i.e., $\hat{\overleftarrow{\tau}}^{\sharp}[\![\vec{a}\cdot\vec{\mathcal{X}} \geq b]\!]\mathrm{M}^{\sharp} \triangleq \{\mathrm{M}^{\sharp},\ \vec{a}\cdot\vec{x} < b\}$, which can greatly improve the precision of analysis. When the number of disjunctive elements exceeds a threshold, we can replace them by their join. It is worth mentioning that current polyhedra abstract domain [41] supports strict constraints like $\vec{a}\cdot\vec{x} < b$. For the original polyhedra abstract domain that cannot express strict constraints, it is sound to replace $\vec{a}\cdot\vec{x} < b$ by $\vec{a}\cdot\vec{x} \leq b$ in Equation (15).

For the interval domain, the same technique can be applied to $\hat{\overleftarrow{\tau}}^{\sharp}[\![\pm\mathcal{X} \geq b]\!]$, since a box (i.e., a Cartesian products of intervals) is a special case of polyhedron. Similarly, we can handle $\hat{\overleftarrow{\tau}}^{\sharp}[\![\pm\mathcal{X} \pm y \geq b]\!]$ for the octagon domain in the same way.

Extended affine guards. For strict guards and the guards with a non-deterministic constant, the corresponding abstract transfer function is defined as:

$$\begin{aligned}
\hat{\overleftarrow{\tau}}^{\sharp}[\![\vec{a}\cdot\vec{\mathcal{X}} > b]\!]\mathrm{M}^{\sharp} &\triangleq \mathrm{M}^{\sharp} \sqcup^{\sharp}_{\mathbb{M}} \vec{a}\cdot\vec{x} \leq b \\
\hat{\overleftarrow{\tau}}^{\sharp}[\![\vec{a}\cdot\vec{\mathcal{X}} > [b;c]]\!]\mathrm{M}^{\sharp} &\triangleq \mathrm{M}^{\sharp} \sqcup^{\sharp}_{\mathbb{M}} \vec{a}\cdot\vec{x} \leq c \\
\hat{\overleftarrow{\tau}}^{\sharp}[\![\vec{a}\cdot\vec{\mathcal{X}} = [b;c]]\!]\mathrm{M}^{\sharp} &\triangleq \mathrm{M}^{\sharp} \sqcup^{\sharp}_{\mathbb{M}} \vec{a}\cdot\vec{x} < b \sqcup^{\sharp}_{\mathbb{M}} \vec{a}\cdot\vec{x} > c.
\end{aligned} \tag{16}$$

Boolean operations. For the Boolean conjunctions and disjunctions of affine guards, Section 3.2 of Reference [61] has shown that the concrete transfer function has the following property:

$$\begin{aligned}
\overleftarrow{\tau}[\![t_1 \vee t_2]\!] &= \overleftarrow{\tau}[\![t_1]\!] \cap \overleftarrow{\tau}[\![t_2]\!] \\
\overleftarrow{\tau}[\![t_1 \wedge t_2]\!] &= \overleftarrow{\tau}[\![t_1]\!] \circ \overleftarrow{\tau}[\![t_2]\!].
\end{aligned} \tag{17}$$

Since the abstract meet $\sqcap^\sharp_{\mathbb{M}}$ is exact in the interval/octagon/polyhedron domain, we can define the corresponding abstract transfer functions that are also exact:

$$\begin{aligned}\overleftarrow{\hat{\tau}}^\sharp\llbracket t_1 \vee t_2 \rrbracket &= \overleftarrow{\hat{\tau}}^\sharp\llbracket t_1 \rrbracket \sqcap^\sharp_{\mathbb{M}} \overleftarrow{\hat{\tau}}^\sharp\llbracket t_2 \rrbracket \\ \overleftarrow{\hat{\tau}}^\sharp\llbracket t_1 \wedge t_2 \rrbracket &= \overleftarrow{\hat{\tau}}^\sharp\llbracket t_1 \rrbracket \circ \overleftarrow{\hat{\tau}}^\sharp\llbracket t_2 \rrbracket.\end{aligned} \tag{18}$$

In addition, the Boolean negation of affine guard $\overleftarrow{\tau}\llbracket \neg(\vec{a}\cdot\vec{\chi} \geq b)\rrbracket$ is equivalent to $\overleftarrow{\tau}\llbracket \vec{a}\cdot\vec{\chi} < b\rrbracket$, and the negation of conjunctions or disjunctions can be eliminated by De Morgan's law.

(1.2) Projection. To reduce the backward transfer function of assignments to the backward transfer function of guards, Reference [61] introduces a projection action $\chi := [-\infty; +\infty]$, which is a special form of assignment that forgets the value of a variable. Here, we do the same and reuse the under-approximating abstract backward transfer function for projections in Reference [61], since it is proved to be exact (i.e., it is both an over-approximation and an under-approximation of the concrete transfer function):

$$\overleftarrow{\hat{\tau}}^\sharp\llbracket \chi := [-\infty; +\infty]\rrbracket \mathrm{M}^\sharp \triangleq \begin{cases} \mathrm{M}^\sharp & \text{if } \gamma_{\mathbb{M}}(\tau^\sharp\llbracket \chi := [-\infty; +\infty]\rrbracket \mathrm{M}^\sharp) = \gamma_{\mathbb{M}}(\mathrm{M}^\sharp) \\ \bot^\sharp_{\mathbb{M}} & \text{otherwise.}\end{cases} \tag{19}$$

The projection is used to model variable addition "add χ" and removal "del χ," which are not included in the language syntax but implicitly created to model assignments. Again, since the under-approximating abstract backward transfer functions for these two actions in Reference [61] are exact, we can simply reuse them:

$$\begin{aligned}\overleftarrow{\hat{\tau}}^\sharp\llbracket \text{del } \chi \rrbracket &= \tau^\sharp\llbracket \text{add } \chi \rrbracket \\ \overleftarrow{\hat{\tau}}^\sharp\llbracket \text{add } \chi \rrbracket &= \tau^\sharp\llbracket \text{del } \chi \rrbracket \circ \overleftarrow{\hat{\tau}}^\sharp\llbracket \chi := [-\infty; +\infty]\rrbracket.\end{aligned} \tag{20}$$

(1.3) Assignments.

Reduction to guards. As shown in Section 3.4 of Reference [61], assignments $\chi := e$ can be reduced to: add a temporary variable χ', then pass a guard $\chi' = e$, remove the variable χ, and rename χ' as χ. Furthermore, the backward transfer function is reduced to:

$$\overleftarrow{\tau}\llbracket \chi := e\rrbracket = \tau\llbracket \text{del } \chi'\rrbracket \circ \overleftarrow{\tau}\llbracket \chi' := [-\infty; +\infty]\rrbracket \circ \overleftarrow{\tau}\llbracket \chi' = e\rrbracket \circ \tau\llbracket \text{add } \chi\rrbracket \circ [\chi'/\chi],$$

where $[\chi'/\chi]$ represents renaming χ as χ'. Correspondingly, the over-approximating backward transfer function can be defined as:

$$\overleftarrow{\hat{\tau}}^\sharp\llbracket \chi := e\rrbracket \quad = \quad \tau^\sharp\llbracket \text{del } \chi'\rrbracket \circ \overleftarrow{\hat{\tau}}^\sharp\llbracket \chi' := [-\infty; +\infty]\rrbracket \circ \overleftarrow{\hat{\tau}}^\sharp\llbracket \chi' = e\rrbracket \circ \tau^\sharp\llbracket \text{add } \chi\rrbracket \circ [\chi'/\chi], \tag{21}$$

in which $\overleftarrow{\hat{\tau}}^\sharp\llbracket \chi' = e\rrbracket$ for the guard $\chi' = e$ is over-approximating, while $\tau^\sharp\llbracket \text{del } \chi'\rrbracket$, $\overleftarrow{\hat{\tau}}^\sharp\llbracket \chi' := [-\infty; +\infty]\rrbracket$, $\tau^\sharp\llbracket \text{add } \chi\rrbracket$ and $[\chi'/\chi]$ are exact.

Special cases of assignments. There are a few special cases such that the above general definition $\overleftarrow{\hat{\tau}}^\sharp\llbracket \chi := e\rrbracket$ can be simplified. For the case where χ is not used in the expression e, there is no need to introduce the temporary variable χ', and the corresponding $\overleftarrow{\hat{\tau}}^\sharp\llbracket \chi := e\rrbracket$ is simplified into:

$$\overleftarrow{\hat{\tau}}^\sharp\llbracket \chi := e\rrbracket \quad = \quad \overleftarrow{\hat{\tau}}^\sharp\llbracket \chi := [-\infty; +\infty]\rrbracket \circ \overleftarrow{\hat{\tau}}^\sharp\llbracket \chi = e\rrbracket. \tag{22}$$

Moreover, for purely non-deterministic assignments $\chi := [a; b]$, variable shifts $\chi := \chi + [a; b]$, and variable copies $\chi := y$, Theorem 9 of Reference [61] yields sound and exact backward transfer function, thus we can reuse them.

Another case is when the assigned expression e is invertible, i.e., there exists an expression e^{-1} that allows recovering the initial value of $\mathcal{X}$. For example, in the assignment $\mathcal{X} := \mathcal{X} + 1$, the expression $\mathcal{X} + 1$ can be inverted by $\mathcal{X} - 1$. In such a case, the backward transfer function for $\mathcal{X} := e$ can be replaced by the forward transfer function for $\mathcal{X} := e^{-1}$, i.e., $\overleftarrow{\hat{\tau}}^{\sharp}\llbracket \mathcal{X} := e \rrbracket = \tau^{\sharp}\llbracket \mathcal{X} := e^{-1} \rrbracket$, which provides a sound over-approximation.

(1.4) Expression approximation. In the above, we have discussed how to handle affine expressions in guards and assignments. As for non-affine numeric expressions, Reference [61] proposes to over-approximate arbitrary expressions by affine ones, and this is accomplished by the linearization technique [59] that performs interval arithmetic on non-linear expression parts. Similarly, here we could convert non-affine expressions into affine expressions with some non-determinism embedded in a constant interval (or constant coefficients), such that the original non-affine expressions are under-approximated. Then, by replacing the original non-affine expressions with affine ones, we can reuse the solution designed for affine expressions and correspondingly get over-approximating backward transfer functions for arbitrary guards and assignments.

(2) Narrowing. Up to now, we have discussed the design of over-approximating backward transfer function $\hat{\mathrm{F}}^{\sharp}_{\overleftarrow{if}}\llbracket \mathrm{P} \rrbracket$. By the soundness condition 10, the greatest fixpoint $\mathrm{gfp}^{\dot{\sqsubseteq}^{\sharp}_{\mathbb{M}}}_{\mathrm{I}^{\sharp}_{\mathrm{post}}} \hat{\mathrm{F}}^{\sharp}_{\overleftarrow{if}}\llbracket \mathrm{P} \rrbracket$ would be an over-approximation of the concrete backward impossible failure accessibility semantics $\mathrm{gfp}^{\dot{\sqsubseteq}}_{\dot{\gamma}_{\mathbb{M}}(\mathrm{I}^{\sharp}_{\mathrm{post}})} \mathrm{F}_{\overleftarrow{if}}\llbracket \mathrm{P} \rrbracket$. However, it is generally difficult to compute $\mathrm{gfp}^{\dot{\sqsubseteq}^{\sharp}_{\mathbb{M}}}_{\mathrm{I}^{\sharp}_{\mathrm{post}}} \hat{\mathrm{F}}^{\sharp}_{\overleftarrow{if}}\llbracket \mathrm{P} \rrbracket$, since the decreasing iteration may be infinite. In many cases, a (dual) widening is used to accelerate the convergence, but it does not apply here, since the (dual) widening makes downwards extrapolation, which may jump below the greatest fixpoint. Therefore, we propose to over-approximate a decreasing iteration by narrowing, because the narrowing can only do interpolations that prevent jumping below any fixpoint.

The narrowing operator $\triangle_{\mathbb{I}} \in (\mathbb{L} \mapsto \mathcal{D}^{\sharp}_{\mathbb{M}}) \times (\mathbb{L} \mapsto \mathcal{D}^{\sharp}_{\mathbb{M}}) \mapsto (\mathbb{L} \mapsto \mathcal{D}^{\sharp}_{\mathbb{M}})$ on the abstract invariant domain satisfies the following soundness and termination conditions:

(i) $\forall x^{\sharp}, y^{\sharp} \in (\mathbb{L} \mapsto \mathcal{D}^{\sharp}_{\mathbb{M}}).\ y^{\sharp} \mathrel{\dot{\sqsubseteq}^{\sharp}_{\mathbb{M}}} x^{\sharp} \Rightarrow y^{\sharp} \mathrel{\dot{\sqsubseteq}^{\sharp}_{\mathbb{M}}} (x^{\sharp} \triangle_{\mathbb{I}} y^{\sharp}) \mathrel{\dot{\sqsubseteq}^{\sharp}_{\mathbb{M}}} x^{\sharp}$;

(ii) for any sequence $(x^{\sharp}_i)_{i\in\mathbb{N}}$, the sequence $(y^{\sharp}_i)_{i\in\mathbb{N}}$ defined as $y^{\sharp}_0 = x^{\sharp}_0$ and $\forall i \in \mathbb{N}.\ y^{\sharp}_{i+1} = y^{\sharp}_i \triangle_{\mathbb{I}} x^{\sharp}_{i+1}$ converges in finite time.

The implementation of $\triangle_{\mathbb{I}}$ naturally follows the narrowing operator $\triangle_{\mathbb{M}} \in \mathcal{D}^{\sharp}_{\mathbb{M}} \times \mathcal{D}^{\sharp}_{\mathbb{M}} \mapsto \mathcal{D}^{\sharp}_{\mathbb{M}}$ provided by the abstract environment domain $\mathcal{D}^{\sharp}_{\mathbb{M}}$, such that $\mathrm{I}^{\sharp} \triangle_{\mathbb{I}} \mathrm{I}^{\sharp\prime} \triangleq \lambda \ell \in \mathbb{L}.\ \mathrm{I}^{\sharp}(\ell) \triangle_{\mathbb{M}} \mathrm{I}^{\sharp\prime}(\ell)$.

(3) Over-approximating Abstract Backward impossible failure Accessibility Semantics. Given a postcondition specified as $\mathrm{I}^{\sharp}_{\mathrm{post}} \in \mathbb{L} \mapsto \mathcal{D}^{\sharp}_{\mathbb{M}}$, the concrete backward impossible failure accessibility semantics $\mathcal{S}_{\overleftarrow{if}}\llbracket \mathrm{P} \rrbracket(\dot{\gamma}_{\mathbb{M}}(\mathrm{I}^{\sharp}_{\mathrm{post}})) = \mathrm{gfp}^{\dot{\sqsubseteq}}_{\dot{\gamma}_{\mathbb{M}}(\mathrm{I}^{\sharp}_{\mathrm{post}})} \mathrm{F}_{\overleftarrow{if}}\llbracket \mathrm{P} \rrbracket$ can be over-approximated by the limit of a ultimately stationary sequence $(\mathrm{I}^{\sharp}_i)_{i\in\mathbb{N}}$, where $\mathrm{I}^{\sharp}_0 = \mathrm{I}^{\sharp}_{\mathrm{post}}$ and $\forall i \in \mathbb{N}.\ \mathrm{I}^{\sharp}_{i+1} = \mathrm{I}^{\sharp}_i \triangle_{\mathbb{I}} \hat{\mathrm{F}}^{\sharp}_{\overleftarrow{if}}\llbracket \mathrm{P} \rrbracket(\mathrm{I}^{\sharp}_i)$:

$$\forall \mathrm{I}^{\sharp}_{\mathrm{post}} \in \mathbb{L} \mapsto \mathcal{D}^{\sharp}_{\mathbb{M}}.\ \mathrm{gfp}^{\dot{\sqsubseteq}}_{\dot{\gamma}_{\mathbb{M}}(\mathrm{I}^{\sharp}_{\mathrm{post}})} \mathrm{F}_{\overleftarrow{if}}\llbracket \mathrm{P} \rrbracket \mathrel{\dot{\sqsubseteq}} \dot{\gamma}_{\mathbb{M}} \left(\lim_{\mathrm{I}^{\sharp}_{\mathrm{post}}} \lambda \mathrm{I}^{\sharp}.\ \mathrm{I}^{\sharp} \triangle_{\mathbb{I}} \hat{\mathrm{F}}^{\sharp}_{\overleftarrow{if}}\llbracket \mathrm{P} \rrbracket\left(\mathrm{I}^{\sharp}\right) \right). \tag{23}$$

In the rest of this article, the over-approximating abstract backward impossible failure accessibility semantics $\hat{\mathcal{S}}^{\sharp}_{\overleftarrow{if}}\llbracket \mathrm{P} \rrbracket$ refers to the following definition, which gives an over-approximation of $\mathcal{S}_{\overleftarrow{if}}\llbracket \mathrm{P} \rrbracket$ and can be computed in finite time:

Table 7. Over-approximating Abstract Backward Impossible Failure Accessibility Semantics for "the Access to o Fails" with Disjunctive Completion

ℓ	$\hat{\mathcal{S}}^\sharp_{\overleftarrow{if}}[\![\mathrm{P}]\!](\mathrm{I}^{\sharp\prime}_{\mathrm{post}})\ell$
ℓ_1	$\bot^\sharp_{\mathbb{M}}$
ℓ_2	$\bot^\sharp_{\mathbb{M}}$
ℓ_3	$apv \in [1;1] \wedge i1 \in [\mathbf{-1};\mathbf{0}] \wedge i2 \in [-\infty;\infty] \wedge typ \in [-\infty;\infty] \wedge acs \in [-\infty;\infty]$
ℓ_4	$apv \in [\mathbf{-1};\mathbf{0}] \wedge i1 \in [-1;2] \wedge i2 \in [-\infty;\infty] \wedge typ \in [-\infty;\infty] \wedge acs \in [-\infty;\infty]$
ℓ_5	$\{apv \in [\mathbf{-1};\mathbf{0}] \wedge i1 \in [-1;2] \wedge i2 \in [-1;2] \wedge typ \in [-\infty;\infty] \wedge acs \in [-\infty;\infty],$ $apv \in [\mathbf{1};\mathbf{1}] \wedge i1 \in [-1;2] \wedge i2 \in [\mathbf{-1};\mathbf{0}] \wedge typ \in [-\infty;\infty] \wedge acs \in [-\infty;\infty]\}$
ℓ_6	$apv \in [\mathbf{-1};\mathbf{0}] \wedge i1 \in [-1;2] \wedge i2 \in [-1;2] \wedge typ \in [-\infty;\infty] \wedge acs \in [-\infty;\infty]$
ℓ_7	$apv \in [\mathbf{-1};\mathbf{0}] \wedge i1 \in [-1;2] \wedge i2 \in [-1;2] \wedge typ \in [1;2] \wedge acs \in [-\infty;\infty]$
ℓ_8	$apv \in [-1;1] \wedge i1 \in [-1;2] \wedge i2 \in [-1;2] \wedge typ \in [1;2] \wedge acs \in [\mathbf{-2};\mathbf{0}]$

$$\hat{\mathcal{S}}^\sharp_{\overleftarrow{if}}[\![\mathrm{P}]\!] \quad \in \quad \left(\mathbb{L} \mapsto \mathcal{D}^\sharp_{\mathbb{M}}\right) \mapsto \left(\mathbb{L} \mapsto \mathcal{D}^\sharp_{\mathbb{M}}\right) \qquad \text{over-approximating IF semantics}$$

$$\hat{\mathcal{S}}^\sharp_{\overleftarrow{if}}[\![\mathrm{P}]\!]\left(\mathrm{I}^\sharp_{\mathrm{post}}\right) \quad \triangleq \quad \lim_{\mathrm{I}^\sharp_{\mathrm{post}}} \lambda \mathrm{I}^\sharp.\ \mathrm{I}^\sharp \triangle_{\mathbb{I}} \hat{\mathrm{F}}^\sharp_{\overleftarrow{if}}[\![\mathrm{P}]\!](\mathrm{I}^\sharp).$$

In practice, the abstract environment domain $\mathcal{D}^\sharp_{\mathbb{M}}$ may not have an effective narrowing operator $\triangle_{\mathbb{M}}$, which makes the corresponding $\triangle_{\mathbb{I}}$ of no practical use. If this is the case, like in the forward reachability analysis, then we can just omit the narrowing operator and iterate the function $\hat{\mathrm{F}}^\sharp_{\overleftarrow{if}}[\![\mathrm{P}]\!]$ until the analysis result is satisfactory (typically, the number of iterations needed is quite low).

Example 17 (Access Control, Continued). Using the refined abstract postcondition $\mathrm{I}^{\sharp\prime}_{\mathrm{post}}$ from Example 16 that represents the trace property "the access to o fails," an over-approximating backward impossible failure accessibility analysis $\hat{\mathcal{S}}^\sharp_{\overleftarrow{if}}[\![\mathrm{P}]\!](\mathrm{I}^{\sharp\prime}_{\mathrm{post}})$ creates the result displayed in Table 7. Here, we adopt the disjunctive completion to gain precision, i.e., at point ℓ_5, the disjunction of two abstract environment elements are maintained, which gives the most precise backward impossible failure accessibility semantics. If the disjunctive completion is not used at point ℓ_5, then we can simply join these two abstract elements together and get the result same as abstract forward reachability semantics $\mathcal{S}^\sharp_{\overrightarrow{ps}}[\![\mathrm{P}]\!](\mathrm{I}^\sharp_{\mathrm{pre}})$ from the point ℓ_5 to the point ℓ_1, which is still sound but imprecise for the further responsibility analysis.

5 TRACE PARTITIONING

The forward reachability analysis discussed in Section 4 intends to compute an over-approximation of reachable states of the program, while the information about the execution history and concrete flow paths is lost, making the correspondingly generated over-approximating reachability semantics in some cases imprecise to determine if a behavior really occurs or not.

In References [53, 70], Mauborgne and Rival propose a trace partitioning domain, which allows partitioning traces by the history of memory and control states. Essentially, for any given transition system, they build an extended transition system by augmenting the program points (i.e., control states, labels) with partitioning tokens, which can distinguish traces by the control flow or variable values. This technique has been successfully implemented in the abstract

```
      int χ, sgn;
ℓ0 :  if (χ < 0){
ℓ1 :     sgn := -1;
ℓ2 :  }else {
ℓ3 :     sgn := 1;
ℓ4 :  }
ℓ5 :  y := χ/sgn;
ℓ6 :  ...
```

Fig. 12. Motivating example for trace partitioning.

Table 8. Abstract Forward Reachability Semantics of the Motivating Example

ℓ	$\mathcal{S}^\sharp_{\overrightarrow{ps}}[\![\mathrm{P}]\!](\mathsf{I}^\sharp_{\mathrm{pre}})\ell$
ℓ_0	$\top^\sharp_{\mathbb{M}} = \chi \in [-\infty;\infty] \wedge sgn \in [-\infty;\infty] \wedge y \in [-\infty;\infty]$
ℓ_1	$\chi \in [-\infty;-1] \wedge sgn \in [-\infty;\infty] \wedge y \in [-\infty;\infty]$
ℓ_2	$\chi \in [-\infty;-1] \wedge sgn \in [-1;-1] \wedge y \in [-\infty;\infty]$
ℓ_3	$\chi \in [0;\infty] \wedge sgn \in [-\infty;\infty] \wedge y \in [-\infty;\infty]$
ℓ_4	$\chi \in [0;\infty] \wedge sgn \in [1;1] \wedge y \in [-\infty;\infty]$
ℓ_5	$\chi \in [-\infty;\infty] \wedge sgn \in [-1;1] \wedge y \in [-\infty;\infty]$
ℓ_6	$\chi \in [-\infty;\infty] \wedge sgn \in [-1;1] \wedge y \in [-\infty;\infty]$

interpretation-based Astrée analyzer [22, 23], significantly improving the precision of analysis and reducing the execution time.

This section briefly summarizes the key idea of trace partitioning, proposes to represent elements in the trace partitioning abstract domain as trace partitioning automata, and extends the existing types of partitioning directive to include program invariants, which facilitates determining responsibility in the abstract (see Section 7). For more details about the theoretical framework and practical implementation of the trace partitioning domain, we refer to Reference [70].

5.1 The Trace Partitioning Abstract Domain

This section starts with a simple motivating example from Reference [70] and illustrates how the trace partitioning improves the precision of forward reachability analysis.

In the program of Figure 12, it is obvious that the value of sgn is either 1 or -1 at point ℓ_5, and in particular it cannot be 0 in the concrete. Therefore, dividing by sgn at point ℓ_5 is safe, and there is no possible "division by zero" error in the program. However, if we use the interval domain as the abstract environment domain, then by the over-approximating forward reachability analysis introduced in Section 4.2.2, we would get the reachability semantics (or, say, program invariants) $\mathcal{S}^\sharp_{\overrightarrow{ps}}[\![\mathrm{P}]\!](\mathsf{I}^\sharp_{\mathrm{pre}})$ listed in the Table 8, where $\mathsf{I}^\sharp_{\mathrm{pre}} \in \mathbb{L} \mapsto \mathcal{D}^\sharp_{\mathbb{M}}$ is defined such that $\mathsf{I}^\sharp_{\mathrm{pre}}(\ell_0) = \top^\sharp_{\mathbb{M}}$ and $\mathsf{I}^\sharp_{\mathrm{pre}}(\ell) = \bot^\sharp_{\mathbb{M}}$ for $\ell \neq \ell_0$. Particularly, the value of sgn at point ℓ_5 belongs to the interval $[-1;1]$, which is not precise enough to exclude the possibility of "division by zero" in the program.

An intuitive idea to solve the imprecision problem is to relate the value of sgn to the way it is computed. In this very example here, if the *true* branch of the conditional was taken, then the value of sgn at point ℓ_5 is -1; otherwise, it is 1. That is to say, we partition the set of all possible

d	::=	$\text{part}\langle \text{If}, \ell, b\rangle$	traces in the b branch of the conditional at point ℓ
	|	$\text{part}\langle \text{While}, \ell, n\rangle$	traces with exactly n iterations in the loop at point ℓ
	|	$\text{part}\langle \text{While}, \ell, > n\rangle$	traces with more than n iterations in the loop at point ℓ
	|	$\text{part}\langle \text{Val}, \ell, \chi = n\rangle$	traces such that $\chi = n$ at point ℓ
	|	$\text{part}\langle \text{Fun}, \ell, f\rangle$	traces calling function f at point ℓ
	|	$\text{part}\langle \text{None}\rangle$	void directive
t	::=	ϵ	empty stack, initial partition
	|	$d :: t'$	addition of a partitioning directive on top of t'

Fig. 13. Partitioning directives $d \in D$ and tokens $t \in T$.

concrete traces into two parts: in one partition, the *true* branch is taken; in the other partition, the *false* is taken. For each partition, the standard over-approximating forward reachability analysis can be performed, and the analysis results together would be more precise.

To generalize the idea of partitioning, Mauborgne and Rival [53, 70] propose a trace partitioning abstract domain, which is flexible and general to analyze and verify semantic properties in the same way as other classic abstract domains. In the following, we will briefly describe how to construct the trace partitioning abstract domain.

(1) Extended Transition Systems. Suppose T is a set of *partitioning tokens*, which are used to capture useful information about the history of execution and to guide trace partitioning. In practice, each partitioning token $t \in T$ is defined as a stack of *partitioning directives* that have been encountered during the execution, and all the possible partitioning directives are listed in Figure 13, each of which creates a partition as its name implies. For example, in the case of a conditional at point ℓ, by the partitioning directives $\text{part}\langle \text{If}, \ell, \text{tt}\rangle$ and $\text{part}\langle \text{If}, \ell, \text{ff}\rangle$, two partitions are created right after testing the Boolean condition, which, respectively, correspond to "true branch of the conditional at point ℓ" and "false branch of the conditional at point ℓ."

Given a set of partitioning tokens T, the *extended transition systems* are defined as transition systems over the set of program points (or, control states, labels) extended with the partitioning tokens T. More formally, let $\mathbb{L}_T \triangleq \mathbb{L} \times T$ be the set of *extended program points*, $\mathbb{S}_T \triangleq \mathbb{L}_T \times \mathbb{M}$ be the set of *extended states*, $\mathbb{S}^i_T \subseteq \mathbb{S}_T$ be the set of *extended initial states*, and $\rightarrow_T \in \wp(\mathbb{S}_T \times \mathbb{S}_T)$ be the *transition relation* among extended states. Then, we define an *extended transition system* as a tuple $\langle T, \mathbb{S}^i_T, \rightarrow_T\rangle$. In addition, a *forget function* π_τ can be defined to remove the partitioning tokens from extended program points, extended states, and transition relations, such that an extended transition system $\langle T, \mathbb{S}^i_T, \rightarrow_T\rangle$ can be transformed back into a standard transition system $\langle \mathbb{S}^i, \rightarrow\rangle$.

(2) Trace Partitioning Abstract Domain. An extended transition system $\mathrm{P}_T = \langle T, \mathbb{S}^i_T, \rightarrow_T\rangle$ is a *covering* of the original transition system $\mathrm{P} = \langle \mathbb{S}^i, \rightarrow\rangle$ if and only if every initial state $s \in \mathbb{S}^i$ has at least one corresponding initial state $s' \in \mathbb{S}^i_T$ such that $\pi_\tau(s') = s$, and every transition step in P is simulated (mimicked) by at least one transition step in P_T. Therefore, if P_T is a covering of P, then every trace in P is simulated by one or more traces in P_T. For the formal definitions of covering and partition see Section 3.2 of Reference [70].

The *trace partitioning abstract domain* $\mathbb{D}^\sharp$ is the set of tuples $\langle \mathrm{P}_T, \Phi^\sharp\rangle$, where T is a set of partitioning tokens, $\mathrm{P}_T = \langle T, \mathbb{S}^i_T, \rightarrow_T\rangle$ is a covering of the original transition system $\mathrm{P} = \langle \mathbb{S}^i, \rightarrow\rangle$, and $\Phi^\sharp \in \mathbb{L}_T \mapsto \mathcal{D}^\sharp_{\mathbb{M}}$ is a function mapping each extended program point $\langle \ell, t\rangle$ of P_T into an abstract environment in $\mathcal{D}^\sharp_{\mathbb{M}}$ that approximates the set of environments observed at point $\langle \ell, t\rangle$.

Table 9. Partitioned Forward Reachability Semantics of the Motivating Example

$\langle \ell,\ t\rangle$	$\Phi^\sharp(\langle \ell,\ t\rangle)$
$\langle \ell_0,\ \epsilon\rangle$	$x \in [-\infty;\infty] \wedge sgn \in [-\infty;\infty] \wedge y \in [-\infty;\infty]$
$\langle \ell_1,\ \text{part}\langle \text{If},\ \ell_0,\ \text{tt}\rangle\rangle$	$x \in [-\infty;-1] \wedge sgn \in [-\infty;\infty] \wedge y \in [-\infty;\infty]$
$\langle \ell_2,\ \text{part}\langle \text{If},\ \ell_0,\ \text{tt}\rangle\rangle$	$x \in [-\infty;-1] \wedge sgn \in [-1;-1] \wedge y \in [-\infty;\infty]$
$\langle \ell_3,\ \text{part}\langle \text{If},\ \ell_0,\ \text{ff}\rangle\rangle$	$x \in [0;\infty] \wedge sgn \in [-\infty;\infty] \wedge y \in [-\infty;\infty]$
$\langle \ell_4,\ \text{part}\langle \text{If},\ \ell_0,\ \text{ff}\rangle\rangle$	$x \in [0;\infty] \wedge sgn \in [1;1] \wedge y \in [-\infty;\infty]$
$\langle \ell_5,\ \text{part}\langle \text{If},\ \ell_0,\ \text{tt}\rangle\rangle$	$x \in [-\infty;-1] \wedge sgn \in [-1;-1] \wedge y \in [-\infty;\infty]$
$\langle \ell_5,\ \text{part}\langle \text{If},\ \ell_0,\ \text{ff}\rangle\rangle$	$x \in [0;\infty] \wedge sgn \in [1;1] \wedge y \in [-\infty;\infty]$
$\langle \ell_6,\ \text{part}\langle \text{If},\ \ell_0,\ \text{tt}\rangle\rangle$	$x \in [-\infty;-1] \wedge sgn \in [-1;-1] \wedge y \in [1;\infty]$
$\langle \ell_6,\ \text{part}\langle \text{If},\ \ell_0,\ \text{ff}\rangle\rangle$	$x \in [0;\infty] \wedge sgn \in [1;1] \wedge y \in [0;\infty]$

Taking the program in Figure 12 as an example, if we use partitioning directives designed for the conditional, then the forward reachability analysis with trace partitioning would construct the corresponding $\Phi^\sharp$ function, which is listed in the Table 9.

From the above table, we can see that there are two extended program points for ℓ_5: $\langle \ell_5,\ \text{part}\langle \text{If},\ \ell_0,\ \text{tt}\rangle\rangle$ and $\langle \ell_5,\ \text{part}\langle \text{If},\ \ell_0,\ \text{ff}\rangle\rangle$, and the corresponding abstract environments indicates the value of sgn is either -1 or 1, which is exactly the desired result.

However, if we have successive creations of partitions in the extended transition system, the cost would be prohibitive in practice. For instance, the partitioning of a conditional multiplies by 2 the number of partitions in the current flow, thus a series of n conditionals would lead to 2^n partitions, which brings an exponential cost. To solve this issue, we can merge partitions together when they are no longer necessary, which is implemented by popping (or removing) partitioning directives from the token. For example, at point ℓ_6 of the program in Figure 12, the partitions based on "which branch of the conditional was taken" are not expected to lead to improvement in the precision of further analysis, so we can merge those two partitions and replace the last two rows of Table 9 by $\Phi^\sharp(\langle \ell_6,\ \epsilon\rangle) = x \in [-\infty;\infty] \wedge sgn \in [-1;1] \wedge y \in [0;\infty]$, which is still more precise than the standard forward reachability analysis.

5.2 The Trace Partitioning Automata

To facilitate determining abstract responsibility on graph structures (in Section 7), this section proposes to represent the result of forward reachability analysis with trace partitioning as automata, which are called *trace partitioning automata*.

More formally, each element $\langle \mathrm{P}_T = \langle T,\ \mathbb{S}^i_T,\ \rightarrow_T\rangle,\ \Phi^\sharp\rangle$ in the trace partitioning abstract domain $\mathbb{D}^\sharp$ can be represented as an automaton $\mathcal{A} = \langle Q^i,\ \delta\rangle$, where:

- The set of initial nodes (extended initial abstract states) $Q^i = \{\langle \ell^i,\ t,\ \mathrm{M}^\sharp\rangle \in \mathbb{L}_T \times \mathcal{D}^\sharp_{\mathbb{M}} \mid \exists \rho \in \mathbb{M}.\ \langle \ell^i, t, \rho\rangle \in \mathbb{S}^i_T \wedge \mathrm{M}^\sharp = \Phi^\sharp(\langle \ell^i, t\rangle)\}$ such that every initial state $\langle \ell^i, t, \rho\rangle$ in P_T is represented by an initial node, which is associated with an abstract environment element $\mathrm{M}^\sharp = \Phi^\sharp(\langle \ell^i, t\rangle)$. By the property of $\Phi^\sharp$ in the trace partitioning abstract domain, it is guaranteed that $\rho \in \gamma_{\mathbb{M}}(\mathrm{M}^\sharp)$.
- The set of edges (extended abstract transition relations) $\delta = \{\langle \ell,\ t,\ \mathrm{M}^\sharp\rangle \rightarrow \langle \ell',\ t',\ \mathrm{M}^{\sharp\prime}\rangle \in (\mathbb{L}_T \times \mathcal{D}^\sharp_{\mathbb{M}}) \times (\mathbb{L}_T \times \mathcal{D}^\sharp_{\mathbb{M}}) \mid \exists \rho, \rho' \in \mathbb{M}.\ \langle \ell,\ t,\ \rho\rangle \rightarrow_T \langle \ell',\ t',\ \rho'\rangle \ \wedge\ \mathrm{M}^\sharp = \Phi^\sharp(\langle \ell,\ t\rangle) \wedge \mathrm{M}^{\sharp\prime} = \Phi^\sharp(\langle \ell',\ t'\rangle)\}$ such that every concrete transition relation in P_T has a corresponding edge in the automaton.

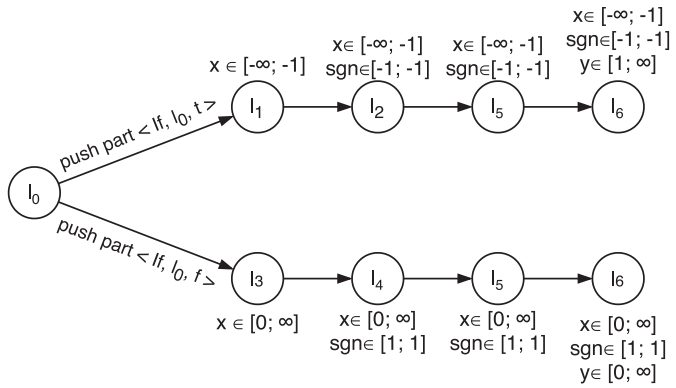

Fig. 14. Trace partitioning automaton for the motivating example without merge.

Again, consider the program in Figure 12: Its partitioned forward reachability analysis result from Table 9 can be represented as a trace partitioning automaton, which is depicted as in Figure 14. For the sake of concision, instead of explicitly drawing partitioning tokens inside the nodes, we comment some edges with "push d" such that every node after the edge has the partitioning directive d pushed into its stack of directives (i.e., its partitioning token). For instance, in Figure 14, all the nodes after the edge commented with "push part$\langle$If, ℓ_0, tt$\rangle$" has part$\langle$If, ℓ_0, tt$\rangle$ in their partitioning tokens.

In addition, to represent the merge of partitions, we comment some edges with "pop d" such that the partitioning directive d is popped from the stack of directives of every node after the edge. For instance, Figure 15 depicts the trace partitioning automaton for the program in Figure 12 with partitions merged at point ℓ_6, so the partitioning token in the node ℓ_6 is ϵ (i.e., its stack of partitioning directives is empty).

5.3 The Extension of Partitioning Directives

As illustrated in Reference [70], partitioning directives are inserted in the source code as special comments in a preprocessing phase. Specifically, among the six types of partitioning directives described in Figure 13, five of them partition traces based on the control flow, and only part$\langle$Val, ℓ, $\chi = n\rangle$ introduces a partition guided by the value of a variable x at some point ℓ. Although these partitioning directives have successfully handled a broad range of cases, there are still many cases that cannot be well coped with, and we would like to introduce a new partitioning directive to partition traces by environment properties (which are represented by abstract environment elements).

For example, to improve the precision of forward reachability analysis for the access control program in Figure 4, it is intuitive to partition traces by some environment properties that can be easily expressed by abstract environment elements in $\mathcal{D}^\sharp_{\mathbb{M}}$ (i.e., $i1 \in [-\infty, 0]$ and $i1 \in [1; \infty]$ at point ℓ_3; $apv \in [1, \infty] \wedge i2 \in [-\infty, 0]$, $apv \in [-\infty, 0]$ and $i2 \in [1, \infty]$ at point ℓ_5), and such properties (e.g., $apv \in [1, \infty] \wedge i2 \in [-\infty, 0]$) may not be expressed by directives of form part$\langle$Val, ℓ, $\chi = n\rangle$ when more than one variables are used in partitioning. Of course, the access control program can be equivalently transformed into a program with conditionals (by replacing ternary operators with conditionals), then the problem of partitioning guided by environment properties is transformed to partitioning based on the branch of conditionals. However, this is not always the case. For example, consider a simple program "$\ell_1 : z := \chi - y;\ \ell_2 :$" and suppose we are interested in whether the value of z is positive or negative at point ℓ_2, then it is of value to create partitions

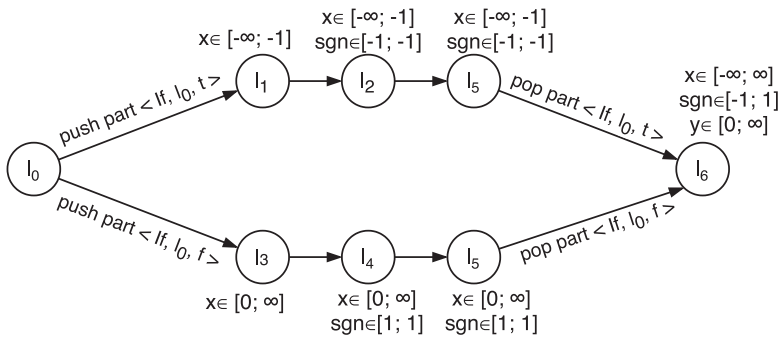

Fig. 15. Trace partitioning automaton for the motivating example with merge.

guided by $x \geq y$ and $x < y$ at point ℓ_1. Such partitions can be expressed by abstract environment elements in the polyhedra/octagons domain, but not by the existing partitioning directives.

Therefore, here we propose a new partitioning directive of the form: part$\langle$Inv, ℓ, $M^\sharp\rangle$ (where $M^\sharp$ is an abstract environment element in $\mathcal{D}^\sharp_{\mathbb{M}}$), which generates a new partition of traces such that $M^\sharp$ is satisfied at point ℓ. In practice, the trace partitioning introduced by a directive part$\langle$Inv, ℓ, $M^\sharp\rangle$ can be simply implemented by creating a new node, such that its partitioning token (i.e., the stack of partitioning directives) has this directive on the top, and the corresponding abstract environment element is the meet of $M^\sharp$ and the standard forward reachability semantics $\mathcal{S}^\sharp_{\overrightarrow{ps}}[\![P]\!](I^\sharp_{pre})\ell$ at point ℓ.

Example 18 (Access Control, Continued). In Example 14, we have discussed the standard abstract forward reachability semantics of the access control program, in which the abstract environment domain $\mathcal{D}^\sharp_{\mathbb{M}}$ is the interval domain. Now, we can gain more precision by introducing five partitioning directives: d_3 : part$\langle$Inv, ℓ_3, $i1 \in [-\infty, 0]\rangle$, d'_3 : part$\langle$Inv, ℓ_3, $i1 \in [1, \infty]\rangle$, d_5 : part$\langle$Inv, ℓ_5, $apv \in [1, \infty] \wedge i2 \in [-\infty, 0]\rangle$, d'_5 : part$\langle$Inv, ℓ_5, $apv \in [\infty, 0]\rangle$, and d''_5 : part$\langle$Inv, ℓ_5, $i2 \in [1, \infty]\rangle$, and the corresponding partitioned forward reachability semantics is listed in Table 10. For the sake of conciseness, trivial elements like "$acs \in [-\infty; \infty]$" are omitted, and the forward reachability analysis stops at invalid extended program points that are associated with $\bot^\sharp_{\mathbb{M}}$.

Compared with the standard forward reachability semantics $\mathcal{S}^\sharp_{\overrightarrow{ps}}[\![P]\!](I^\sharp_{pre})\ell$ in Example 14, the partitioned forward reachability semantics is much more precise, revealing the relation between *acs* and other variables, which is of significance in determining responsibility later.

Furthermore, the partitioned forward reachability semantics can be represented by a trace partitioning automaton in Figure 16. It is worth mentioning that, as what we have done in Figure 15, the partitions can be merged after the access check to *acs* finishes at point ℓ_8, such that all the partitioning directives pushed at point ℓ_3 or ℓ_5 can be popped from the partitioning tokens at point ℓ_8.

6 USER SPECIFICATION OF BEHAVIORS AND COGNIZANCE

In Section 3.2, the responsibility is defined as an abstraction $\alpha_R([\![P]\!]^{Max}, \mathcal{L}^{Max}, \mathbb{C}, \mathcal{B}, \mathcal{T})$, where $[\![P]\!]^{Max} \in \wp(\mathbb{S}^{*\infty})$ is the concrete maximal trace semantics, $\mathcal{L}^{Max} \in \wp(\wp(\mathbb{S}^{*\infty}))$ is a lattice of system behaviors (i.e., trace properties), $\mathbb{C} \in \mathbb{S}^{*\infty} \mapsto \wp(\mathbb{S}^{*\infty})$ is the cognizance function of a given observer, $\mathcal{B} \in \mathcal{L}^{Max}$ is the behavior whose responsibility is of interest, and $\mathcal{T} \in \wp([\![P]\!]^{Max})$ is the set of valid traces to be analyzed. Among these parameters, the maximal trace semantics $[\![P]\!]^{Max}$ is inherent in the given program P, which can be soundly over-approximated by the abstract trace partitioning automata introduced in Section 5. Meanwhile, all the other parameters indicate the objective of

Table 10. Partitioned Forward Reachability Semantics for the Access Control Program

$\langle \ell,\ t\rangle$	$\Phi^\sharp(\langle \ell,\ t\rangle)$
$\langle \ell_1,\ \epsilon\rangle$	$\top^\sharp_{\mathbb{M}}$
$\langle \ell_2,\ \epsilon\rangle$	$apv \in [1;1]$
$\langle \ell_3,\ d_3\rangle$	$apv \in [1;1] \wedge i1 \in [-1;0]$
$\langle \ell_3,\ d'_3\rangle$	$apv \in [1;1] \wedge i1 \in [1;2]$
$\langle \ell_4,\ d_3\rangle$	$apv \in [-1;-1] \wedge i1 \in [-1;0]$
$\langle \ell_4,\ d'_3\rangle$	$apv \in [1;1] \wedge i1 \in [1;2]$
$\langle \ell_5,\ d_5 :: d_3\rangle$	$\bot^\sharp_{\mathbb{M}}$
$\langle \ell_5,\ d'_5 :: d_3\rangle$	$apv \in [-1;-1] \wedge i1 \in [-1;0] \wedge i2 \in [-1;2]$
$\langle \ell_5,\ d''_5 :: d_3\rangle$	$apv \in [-1;-1] \wedge i1 \in [-1;0] \wedge i2 \in [1;2]$
$\langle \ell_5,\ d_5 :: d'_3\rangle$	$apv \in [1;1] \wedge i1 \in [1;2] \wedge i2 \in [-1;0]$
$\langle \ell_5,\ d'_5 :: d'_3\rangle$	$\bot^\sharp_{\mathbb{M}}$
$\langle \ell_5,\ d''_5 :: d'_3\rangle$	$apv \in [1;1] \wedge i1 \in [1;2] \wedge i2 \in [1;2]$
$\langle \ell_6,\ d'_5 :: d_3\rangle$	$apv \in [-1;-1] \wedge i1 \in [-1;0] \wedge i2 \in [-1;2]$
$\langle \ell_6,\ d''_5 :: d_3\rangle$	$apv \in [-1;-1] \wedge i1 \in [-1;0] \wedge i2 \in [1;2]$
$\langle \ell_6,\ d_5 :: d'_3\rangle$	$apv \in [-1;-1] \wedge i1 \in [1;2] \wedge i2 \in [-1;0]$
$\langle \ell_6,\ d''_5 :: d'_3\rangle$	$apv \in [1;1] \wedge i1 \in [1;2] \wedge i2 \in [1;2]$
$\langle \ell_7,\ d'_5 :: d_3\rangle$	$apv \in [-1;-1] \wedge i1 \in [-1;0] \wedge i2 \in [-1;2] \wedge typ \in [1;2]$
$\langle \ell_7,\ d''_5 :: d_3\rangle$	$apv \in [-1;-1] \wedge i1 \in [-1;0] \wedge i2 \in [1;2] \wedge typ \in [1;2]$
$\langle \ell_7,\ d_5 :: d'_3\rangle$	$apv \in [-1;-1] \wedge i1 \in [1;2] \wedge i2 \in [-1;0] \wedge typ \in [1;2]$
$\langle \ell_7,\ d''_5 :: d'_3\rangle$	$apv \in [1;1] \wedge i1 \in [1;2] \wedge i2 \in [1;2] \wedge typ \in [1;2]$
$\langle \ell_8,\ d'_5 :: d_3\rangle$	$apv \in [-1;-1] \wedge i1 \in [-1;0] \wedge i2 \in [-1;2] \wedge typ \in [1;2] \wedge acs \in [-2;-1]$
$\langle \ell_8,\ d''_5 :: d_3\rangle$	$apv \in [-1;-1] \wedge i1 \in [-1;0] \wedge i2 \in [1;2] \wedge typ \in [1;2] \wedge acs \in [-2;-1]$
$\langle \ell_8,\ d_5 :: d'_3\rangle$	$apv \in [-1;-1] \wedge i1 \in [1;2] \wedge i2 \in [-1;0] \wedge typ \in [1;2] \wedge acs \in [-2;-1]$
$\langle \ell_8,\ d''_5 :: d'_3\rangle$	$apv \in [1;1] \wedge i1 \in [1;2] \wedge i2 \in [1;2] \wedge typ \in [1;2] \wedge acs \in [1;2]$

responsibility analysis and can be specified only by users. However, it is difficult, if not impossible, to require users to explicitly specify system behaviors and the cognizance function in the concrete. Therefore, to implement the static responsibility analysis, the very first step is to specify $\mathcal{L}^{\mathsf{Max}}$, $\mathbb{C}$, $\mathcal{B}$, and $\mathcal{T}$ in the abstract.

For the sake of simplicity, here it is assumed that we would like to analyze all the maximal traces of P, thus $\mathcal{T} = [\![\mathrm{P}]\!]^{\mathsf{Max}}$ and there is no need for the users to explicitly designate the traces to be analyzed. As for the behavior $\mathcal{B}$ of interest, the lattice $\mathcal{L}^{\mathsf{Max}}$ of behaviors and the cognizance function $\mathbb{C}$, this section discusses how to specify them with the abstract invariant domain $\mathcal{D}^\sharp_{\mathbb{I}} = \langle \mathbb{L} \mapsto \mathcal{D}^\sharp_{\mathbb{M}}, \dot{\sqsubseteq}^\sharp_{\mathbb{M}}\rangle$ introduced in Section 4.1.3.

6.1 User Specification of Behaviors

6.1.1 The Abstract Behavior of Interest. The behavior of interest is specified by an abstract invariant $\mathcal{B}^\sharp \in \mathbb{L} \mapsto \mathcal{D}^\sharp_{\mathbb{M}}$, which associates every program point with an abstract environment element. The corresponding behavior $\mathcal{B}$ in the concrete is $[\![\mathrm{P}]\!]^{\mathsf{Max}} \cap \gamma_{\mathbb{IV}} \circ \dot{\gamma}_{\mathbb{M}}(\mathcal{B}^\sharp) = \{\sigma \in [\![\mathrm{P}]\!]^{\mathsf{Max}} \mid \forall \langle \ell, \rho\rangle \in \sigma.\ \rho \in \gamma_{\mathbb{M}}(\mathcal{B}^\sharp(\ell))\}$, i.e., the set of concrete valid maximal traces such that every state satisfies the abstract environment assigned by $\mathcal{B}^\sharp$ at the corresponding program point.

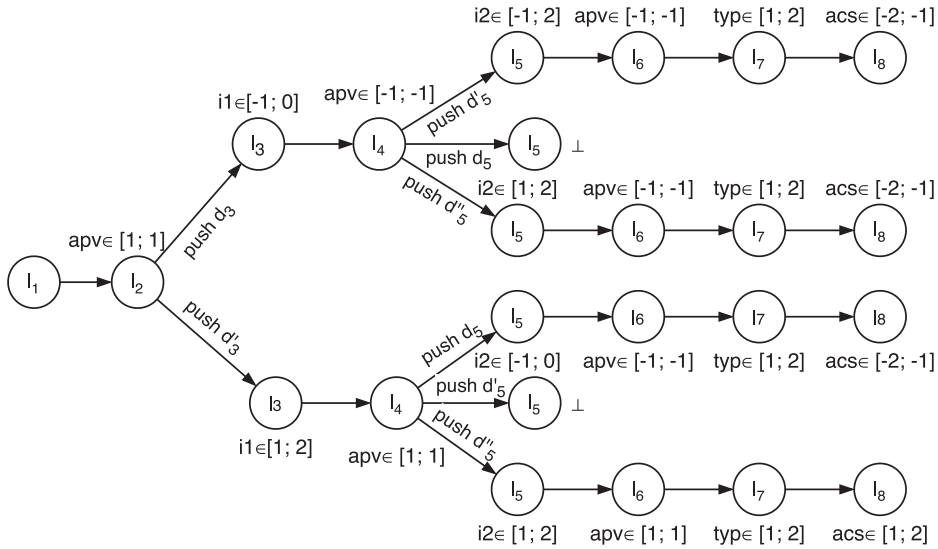

Fig. 16. Trace partitioning automaton for the access control program.

In practice, the user can explicitly designate the chosen program points with some non-trivial abstract environments from $\mathcal{D}^\sharp_{\mathbb{M}}$, while all other program points are implicitly associated with $\top^\sharp_{\mathbb{M}}$.

Example 19 (Access Control, Continued). Let us consider the access control program in Figure 4 again. There are a few behaviors that the user may be interested in: (1) If the user is interested in "the access to o fails," like the abstract postcondition $\mathsf{I}^\sharp_{\text{post}}$ defined in Example 16, then the abstract behavior $\mathcal{B}^\sharp \in \mathbb{L} \mapsto \mathcal{D}^\sharp_{\mathbb{M}}$ can be defined such that $\mathcal{B}^\sharp(\ell_8)$ is explicitly designated as "$acs \in [-\infty; 0]$," while $\mathcal{B}^\sharp(\ell)$ is implicitly assigned to $\top^\sharp_{\mathbb{M}}$ for other program points $\ell \neq \ell_8$. (2) As we have discussed in Section 3, the responsibility for the complement behavior "the access to o succeeds" is different from the one for "the access to o fails." Thus, if the user is interested in "the access to o succeeds" instead, then the abstract behavior shall be specified such that $\mathcal{B}^\sharp(\ell_8)$ is "$acs \in [1; \infty]$," while $\mathcal{B}^\sharp(\ell)$ is $\top^\sharp_{\mathbb{M}}$ for other program points $\ell \neq \ell_8$. (3) Similarly, to analyze "the read and write access to o is granted" that requires the value of acs is greater than or equal to 2 at point ℓ_8, the corresponding abstract behavior $\mathcal{B}^\sharp$ shall be specified such that $\mathcal{B}^\sharp(\ell_8)$ is "$acs \in [2; \infty]$," while $\mathcal{B}^\sharp(\ell) = \top^\sharp_{\mathbb{M}}$ for $\ell \neq \ell_8$.

It is worth mentioning that, although in the above example there is only one program point that is assigned with non-trivial abstract environment elements, in general the user can express behaviors that are related to multiple program points. However, we have to admit that the expressiveness of abstract behaviors depends on the abstract environment domain $\mathcal{D}^\sharp_{\mathbb{M}}$, and not every concrete behavior (i.e., a set of concrete traces) can be expressed by an abstract behavior. For instance, we cannot express the relation of variables by the interval domain, and it is impossible to express behaviors like "the value of x is increasing along the execution" by the numerical invariance abstract domains.

In addition, the user specified behavior $\mathcal{B}^\sharp$ is not directly used in the following backward accessibility analysis. Instead, as what we have done for $\mathsf{I}^\sharp_{\text{post}}$ in Example 16, the abstract behavior

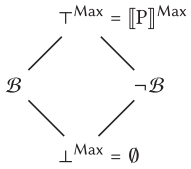

Fig. 17. The lattice $\mathcal{L}^{\mathsf{Max}}$ of behaviors in the concrete.

will be refined by the intersection with the abstract forward reachability semantics, which will be further illustrated in Section 7.

6.1.2 The Lattice of System Behaviors. For the sake of conciseness, it is assumed that the user is interested in analyzing the responsibility of only one behavior, and the corresponding lattice of behaviors in the concrete consists of four elements: the top, the bottom, the behavior of interest, and the corresponding complement behavior. However, for an abstract behavior $\mathcal{B}^\sharp \in \mathbb{L} \mapsto \mathcal{D}^\sharp_{\mathbb{M}}$, its complement may not be expressible by the abstract invariant domain $\mathbb{L} \mapsto \mathcal{D}^\sharp_{\mathbb{M}}$, since most abstract environment domains (e.g., intervals, octagons, polyhedra) do not support complements. For instance, in general the complement of an interval (e.g., $\chi \in [0; 9]$) is a disjunction of two intervals; the complement of a polyhedron is a disjunction of affine inequalities, which cannot be expressed by a polyhedron. Therefore, for any lattice of behaviors in the concrete, it may be impossible to construct the corresponding lattice of abstract behaviors, and we cannot require the user to specify such a lattice.

Fortunately, the abstract responsibility analysis introduced later in Section 7 does not directly use the lattice of abstract behaviors, and it is sufficient to provide only the abstract behavior $\mathcal{B}^\sharp$ of interest to the analyzer. Nevertheless, to prove the soundness of abstract responsibility analysis for a given abstract behavior $\mathcal{B}^\sharp$, the corresponding lattice $\mathcal{L}^{\mathsf{Max}}$ of behaviors in the concrete can be easily built as in Figure 17.

More precisely, the lattice $\mathcal{L}^{\mathsf{Max}}$ of concrete behaviors consists of four elements: the top $\top^{\mathsf{Max}}$ is the maximal trace semantics $[\![\mathrm{P}]\!]^{\mathsf{Max}}$, the bottom $\bot^{\mathsf{Max}}$ is the empty set, the behavior $\mathcal{B} = [\![\mathrm{P}]\!]^{\mathsf{Max}} \cap \gamma_{\mathbb{TV}} \circ \dot{\gamma}_{\mathbb{M}}(\mathcal{B}^\sharp) = \{\sigma \in [\![\mathrm{P}]\!]^{\mathsf{Max}} \mid \forall \langle \ell, \rho\rangle \in \sigma.\ \rho \in \gamma_{\mathbb{M}}(\mathcal{B}^\sharp(\ell))\}$, and the complement behavior $\neg\mathcal{B} = [\![\mathrm{P}]\!]^{\mathsf{Max}} \backslash \mathcal{B} = \{\sigma \in [\![\mathrm{P}]\!]^{\mathsf{Max}} \mid \exists \langle \ell, \rho\rangle \in \sigma.\ \rho \notin \gamma_{\mathbb{M}}(\mathcal{B}^\sharp(\ell))\}$ is the set of valid maximal traces, in each of which there exists at least one state that does not satisfy the abstract environment assigned by $\mathcal{B}^\sharp$.

Example 20 (Access Control, Continued). For the access control program, its maximal trace semantics $[\![\mathrm{P}]\!]^{\mathsf{Max}}$ is given in example 1. If the behavior of interest is "the access to o fails" (i.e., $\mathcal{B}^\sharp$ is specified such that $\mathcal{B}^\sharp(\ell_8) = \mathit{acs} \in [-\infty; 0]$" and $\mathcal{B}^\sharp(\ell) = \top^\sharp_{\mathbb{M}}$ for $\ell \neq \ell_8$), then the corresponding concrete behavior $\mathcal{B} = \{\sigma \in [\![\mathrm{P}]\!]^{\mathsf{Max}} \mid \exists \rho \in \mathbb{M}.\ \sigma_{[7]} = \langle \ell_8, \rho\rangle \wedge \rho(\mathit{acs}) \leq 0\}$, and the complement behavior $\neg\mathcal{B} = \{\sigma \in [\![\mathrm{P}]\!]^{\mathsf{Max}} \mid \exists \rho \in \mathbb{M}.\ \sigma_{[7]} = \langle \ell_8, \rho\rangle \wedge \rho(\mathit{acs}) > 0\}$. Together with the empty set, the four elements form the lattice of behaviors in the concrete.

6.2 User Specification of the Cognizance

In the concrete, the cognizance function $\mathbb{C} \in \mathbb{S}^{*\infty} \mapsto \wp(\mathbb{S}^{*\infty})$ of an observer essentially maps a trace σ to an equivalence class $\mathbb{C}(\sigma)$ of traces such that every trace in $\mathbb{C}(\sigma)$ is equivalent (or, say, indistinguishable) to σ according to the cognizance of that observer. Specially for an omniscient observer, every trace is distinguishable from other traces, thus the equivalence class for each trace is a singleton (i.e., $\forall \sigma \in \mathbb{S}^{*\infty}.\ \mathbb{C}_o(\sigma) = \{\sigma\}$). However, it is infeasible to require users to directly

provide a cognizance function or an equivalence relation on concrete traces, hence the cognizance function needs to be specified in the abstract instead.

6.2.1 The Abstract Cognizance Function. Formally, the *abstract cognizance function* $\mathbb{C}^\sharp \in \mathbb{L} \mapsto \wp(\mathcal{D}^\sharp_{\mathbb{M}})$ is defined as a function mapping the program point to a set of *cognizance directives* d_c, each of which is an element of the numerical abstract domain $\mathcal{D}^\sharp_{\mathbb{M}}$. It is important to note that, although both the abstract environment element $\mathrm{M}^\sharp$ and the cognizance directive d_c are from the same abstract domain $\mathcal{D}^\sharp_{\mathbb{M}}$ (e.g., intervals, octagons, polyhedra), their meanings in the concrete are completely different: $\mathrm{M}^\sharp$ represents a set of concrete environments that satisfy a certain property, while d_c essentially defines an equivalence relation on concrete environments, which is further used to define an equivalence relation on concrete traces.

To start with, we give several examples of the abstract cognizance functions and explain their concrete meanings in an informal but intuitive way, while its formal concretization back to the concrete cognizance function is defined later in this section.

(i) Consider an abstract cognizance function $\mathbb{C}^\sharp$ such that $\mathbb{C}^\sharp(\ell) = \{\chi \in [-\infty; \infty]\}$. When $\chi \in [-\infty; \infty]$ is treated as an abstract environment $\mathrm{M}^\sharp$, then it represents a set of concrete environments, i.e., $\gamma_{\mathbb{M}}(\mathrm{M}^\sharp) = \mathbb{M}$, which does not provide any non-trivial information. In contrary, if we take $\chi \in [-\infty; \infty]$ as a cognizance directive, then it actually defines an equivalence relation on environments, such that two environments are equivalent even if their values of χ are different, as long as the values of any other variable (e.g., z) are the same in those two environments. Thus, such an abstract cognizance function $\mathbb{C}^\sharp$ indicates that the observer does not know the value of χ at the program point ℓ, but the value of any other variable.

(ii) For another abstract cognizance function such that $\mathbb{C}^\sharp(\ell) = \{\chi \in [-\infty; -1], \chi \in [0; \infty]\}$, there are two cognizance directives assigned to point ℓ. Take $\chi \in [0; \infty]$ as an example, it does not mean the value of χ is positive or zero at point ℓ. Instead, it means that any two environments ρ and ρ' at point ℓ are equivalent (or indistinguishable) if and only if the value of χ in both ρ and ρ' are positive or zero (e.g., $\rho(\chi) = 0$ and $\rho'(\chi) = 5$), and the values of any other variable are the same. Similarly, $\chi \in [-\infty; -1]$, as a cognizance directive, means that two environments ρ and ρ' at ℓ are equivalent, as long as their values of χ are negative (e.g., $\rho(\chi) = -2$ and $\rho'(\chi) = -5$) and the values of any other variable are the same. Together, the abstract cognizance function $\mathbb{C}^\sharp$ indicates that the observer does not know the exact value of χ at point ℓ, but only the sign of χ (i.e., positive or zero, or negative), as well as the exact value of other variables.

(iii) The numerical abstract domain used in previous two examples is the interval domain, and now we consider another example with octagon/polyhedron domains. Suppose the abstract cognizance function $\mathbb{C}^\sharp$ is specified such that $\mathbb{C}^\sharp(\ell) = \{\chi \leq y, y < \chi\}$, then the cognizance directive $\chi \leq y$ (respectively, $y < \chi$) means that two environments ρ and ρ' are equivalent if and only if $\rho(\chi) \leq \rho(y)$ and $\rho'(\chi) \leq \rho'(y)$ (respectively, $\rho(y) < \rho(\chi)$ and $\rho'(y) < \rho'(\chi)$) hold, and the values of any other variable in ρ and ρ' are the same. That is to say, the observer does not know the exact value of χ and y at point ℓ, but the relation between χ and y, as well as the exact value of other variables.

In the following, we formalize the equivalence relations introduced by the abstract cognizance function and define the concretization from the abstract cognizance $\mathbb{C}^\sharp \in \mathbb{L} \mapsto \wp(\mathcal{D}^\sharp_{\mathbb{M}})$ back to the corresponding concrete cognizance $\mathbb{C} \in \mathbb{S}^{*\infty} \mapsto \wp(\mathbb{S}^{*\infty})$.

Equivalence Relation on Environments. Suppose $\mathcal{D}^\sharp_{\mathbb{M}}$ is a numerical abstract domain (e.g., intervals, octagons, polyhedra). For any cognizance directive $d_c \in \mathcal{D}^\sharp_{\mathbb{M}}$, let $\mathrm{vars}(d_c)$ be the set of variables used in d_c. For instance, $\mathrm{vars}(\chi \in [-\infty; \infty]) = \{\chi\}$, and $\mathrm{vars}(\chi \leq y) = \{\chi, y\}$. Then, for every

cognizance directive $d_c \in \mathcal{D}^\sharp_{\mathbb{M}}$, we can define an equivalence relation $\overset{d_c}{\sim}$ on concrete environments as follows:

$$\begin{array}{lcll} \overset{d_c}{\sim} & \in & \wp(\mathbb{M} \times \mathbb{M}) & \text{equivalence relation on environments} \\ \rho \overset{d_c}{\sim} \rho' & \Leftrightarrow & \rho = \rho' \vee (\rho \in \gamma_{\mathbb{M}}(d_c) \wedge \rho' \in \gamma_{\mathbb{M}}(d_c) \wedge \forall \chi \in \mathbb{X} \backslash \mathrm{vars}(d_c).\, \rho(\chi) = \rho'(\chi)). & \end{array}$$

That is to say, two environments are equivalent (indistinguishable) according to a cognizance directive d_c if and only if either they are equal to each other, or both of them belong to $\gamma_{\mathbb{M}}(d_c)$ and the values of any variable not used in d_c are the same.

For example, suppose the set of all variables in a program is $\mathbb{X} = \{\chi, y\}$, and the cognizance directive d_c is $\chi \in [0; \infty]$ from the interval domain such that $\mathrm{vars}(d_c) = \{\chi\}$. Let $[\chi \mapsto v, y \mapsto v']$ be an environment such that the value of χ is v and the value of y is v'. Then, we have $[\chi \mapsto 0, y \mapsto 1] \overset{d_c}{\sim} [\chi \mapsto 5, y \mapsto 1]$, since the values of χ in both environments are positive or zero, and the values of y are the same in those two environments. Besides, $[\chi \mapsto -1, y \mapsto 1] \overset{d_c}{\not\sim} [\chi \mapsto 5, y \mapsto 1]$ and $[\chi \mapsto -1, y \mapsto 1] \overset{d_c}{\not\sim} [\chi \mapsto -2, y \mapsto 1]$, since χ is negative in at least one environment; and $[\chi \mapsto 0, y \mapsto 1] \overset{d_c}{\not\sim} [\chi \mapsto 5, y \mapsto 2]$, because the values of y in those two environments are different.

Specially, for the cognizance directive $\bot^\sharp_{\mathbb{M}} \in \mathcal{D}^\sharp_{\mathbb{M}}$, the set of used variables $\mathrm{vars}(\bot^\sharp_{\mathbb{M}})$ is empty, thus two environments cannot be equivalent unless they are equal. Thus, the special cognizance directive $\bot^\sharp_{\mathbb{M}}$ indicates that every concrete environment is distinguishable from each other.

Equivalence Relation on Traces. Given an abstract cognizance function $\mathbb{C}^\sharp \in \mathbb{L} \mapsto \wp(\mathcal{D}^\sharp_{\mathbb{M}})$, an equivalence relation $\overset{\mathbb{C}^\sharp}{\sim}$ on concrete traces can be defined as follows (where $|\sigma|$ denotes the length of σ, and it is ∞ when the trace σ is infinite):

$$\begin{array}{lcll} \overset{\mathbb{C}^\sharp}{\sim} & \in & \wp(\mathbb{S}^{*\infty} \times \mathbb{S}^{*\infty}) & \text{equivalence relation on traces} \\ \sigma \overset{\mathbb{C}^\sharp}{\sim} \sigma' & \Leftrightarrow & |\sigma| = |\sigma'| \wedge \forall i \in [0, |\sigma|).\, (\sigma_{[i]} = \langle \ell,\ \rho\rangle \wedge \sigma'_{[i]} = \langle \ell',\ \rho'\rangle) & \\ & & \qquad \Rightarrow (\ell = \ell' \wedge (\exists d_c \in \mathbb{C}^\sharp(\ell).\, \rho \overset{d_c}{\sim} \rho')). & \end{array}$$

That is to say, two concrete traces are equivalent (indistinguishable) according to the abstract cognizance $\mathbb{C}^\sharp$ if and only if they are of the same length and have the same control flow, and the environments at the same location are equivalent according to some cognizance directive assigned to that point.

For instance, suppose the set of all variables in a program is $\mathbb{X} = \{\chi, y\}$, and the abstract cognizance function $\mathbb{C}^\sharp$ is defined such that $\mathbb{C}^\sharp(\ell_1) = \{\bot^\sharp_{\mathbb{M}}\}$ and $\mathbb{C}^\sharp(\ell_2) = \{\chi \in [0; \infty]\}$. Then, a trace $\langle \ell_1,\ [\chi \mapsto -1, y \mapsto 1]\rangle \to \langle \ell_2,\ [\chi \mapsto 0, y \mapsto 1]\rangle$ is equivalent to another trace $\langle \ell_1,\ [\chi \mapsto -1, y \mapsto 1]\rangle \to \langle \ell_2,\ [\chi \mapsto 5, y \mapsto 1]\rangle$, because the two traces have the same control flow, the two environments at point ℓ_1 are equal, and the two environments at point ℓ_2 are equivalent according to the cognizance directive $\chi \in [0; \infty]$.

Concretization to the Concrete Cognizance Function. Using the equivalence relation $\overset{\mathbb{C}^\sharp}{\sim}$ introduced by the abstraction $\mathbb{C}^\sharp$, we can define the concretization function:

$$\begin{array}{rcll} \gamma_{\mathbb{C}} & \in & \left(\mathbb{L} \mapsto \wp\left(\mathcal{D}^{\sharp}_{\mathbb{M}}\right)\right) \mapsto (\mathbb{S}^{*\infty} \mapsto \wp(\mathbb{S}^{*\infty})) & \text{cognizance concretization} \\ \gamma_{\mathbb{C}}(\mathbb{C}^{\sharp}) & \triangleq & \lambda\sigma \in \mathbb{S}^{*\infty}.\ [\sigma]_{\underset{\sim}{\mathbb{C}^{\sharp}}} & \\ & = & \lambda\sigma \in \mathbb{S}^{*\infty}.\ \{\sigma' \in \mathbb{S}^{*\infty} \mid \sigma \overset{\mathbb{C}^{\sharp}}{\sim} \sigma'\}. & \end{array}$$

According to the above definition, for any abstract cognizance function $\mathbb{C}^{\sharp}$, the corresponding concrete cognizance function maps a trace σ to its equivalence class $[\sigma]_{\underset{\sim}{\mathbb{C}^{\sharp}}}$, i.e., the set of traces that are $\overset{\mathbb{C}^{\sharp}}{\sim}$ equivalent to σ.

Here, we have to admit that, compared with the concrete cognizance function that could map a trace to an arbitrary set of traces, the expressiveness of our abstract cognizance function is restricted: Only traces with the same control flow can be specified as equivalent in the abstract, but it is expressive enough to cover many interesting cases. An alternative way to specify the abstract cognizance function is to use abstract relational invariants, which could express relational properties about two executions of a single program on different inputs [3, 4].

6.2.2 Validating Partitioning Directives with Cognizance. As discussed in Section 5, the program's trace semantics is soundly over-approximated by trace partitioning automata, which can be computed by the abstract forward (possible success) reachability analysis with trace partitioning. Hence, every valid maximal trace is represented by at least one (and possibly more than one) paths in the automaton, and every path in the automaton represents a set of concrete traces, which may include invalid concrete trace due to the over-approximation.

To implement the cognizance function $\mathbb{C}^{\sharp}$ in the abstract responsibility analysis, the key is to guarantee that: For any two concrete traces σ and σ' that are equivalent (indistinguishable) to each other according to $\mathbb{C}^{\sharp}$ (i.e., $\sigma \overset{\mathbb{C}^{\sharp}}{\sim} \sigma'$), they must be represented by the same path in the trace partitioning automaton.

Since the structure of trace partitioning automata is decided by the partitioning directives, we need to make sure that during the execution of any two equivalent traces, every time when a partitioning directive d_p is encountered, the two traces must belong to the same partition (i.e., both of them are in the partition generated by d_p, or neither of them are in the partition generated by d_p). If such a property holds, then the partitioning directive d_p is said to be *valid with respect to the cognizance* $\mathbb{C}^{\sharp}$ and will be used to generate trace partitioning automata; otherwise, it is *invalid* and will be either removed or revised before it is used in generating trace partitioning automata.

By the definition of $\overset{\mathbb{C}^{\sharp}}{\sim}$, equivalent traces are assumed to have the same control flow. Thus, for all partitioning directives related with the control states (e.g., a partitioning directive part⟨If, ℓ, b⟩ that partitions traces by the branch of a conditional), every two equivalent traces are ensured to belong to the same partition. That is to say, for any cognizance function, all the control-state-related partitioning directives are valid. Therefore, when implementing the cognizance function in the abstract responsibility analysis, we only need to check the validity of partitioning directives related with memory states (i.e., environments) that is of the form "part⟨Inv, ℓ, $\mathrm{M}^{\sharp}$⟩," while the directive of the form "part⟨Val, ℓ, $\chi = n$⟩" can be treated as a special case of "part⟨Inv, ℓ, $\mathrm{M}^{\sharp}$⟩."

In this section, we formalize the validity of a partitioning directive d_p with respect to a given cognizance directive d_c and propose a sound approach to check the validity in the abstract.

(1) The Definition of Validity of Partitioning Directives. As explained above, all the control-state-related partitioning directives in Figure 13 are always valid, and here we only need to consider the validity of partitioning directives that are related with environments. Intuitively, a partitioning

directive $d_p = \text{part}\langle \text{Inv}, \ell, \mathrm{M}_p^\sharp \rangle$ creates a partition at point ℓ such that the environment property $\mathrm{M}_p^\sharp$ holds, and this partition is valid if and only if it does not partition any equivalence class of environments into two separate parts. That is to say, every equivalence class of environments must be either a subset of $\gamma_{\mathbb{M}}(\mathrm{M}_p^\sharp)$ or completely disjoint from $\gamma_{\mathbb{M}}(\mathrm{M}_p^\sharp)$.

Definition 2. A partitioning directive $d_p = \text{part}\langle \text{Inv}, \ell, \mathrm{M}_p^\sharp \rangle$ is valid with respect to a cognizance directive $d_c \in \mathcal{D}_{\mathbb{M}}^\sharp$ if and only if

$$\forall \rho \in \mathbb{M}.\ [\rho]_{\underset{\sim}{d_c}} \subseteq \gamma_{\mathbb{M}}\left(\mathrm{M}_p^\sharp\right) \ \vee \ [\rho]_{\underset{\sim}{d_c}} \cap \gamma_{\mathbb{M}}\left(\mathrm{M}_p^\sharp\right) = \emptyset, \tag{24}$$

where $[\rho]_{\underset{\sim}{d_c}} = \{\rho' \in \mathbb{M} \mid \rho \overset{d_c}{\sim} \rho'\}$ and $\rho \overset{d_c}{\sim} \rho' \Leftrightarrow \rho = \rho' \vee (\rho \in \gamma_{\mathbb{M}}(d_c) \wedge \rho' \in \gamma_{\mathbb{M}}(d_c) \wedge \forall \chi \in \mathbb{X} \backslash \text{vars}(d_c).\ \rho(\chi) = \rho'(\chi))$.

For example, for a cognizance directive $d_c = \chi \in [0; \infty]$, the partitioning directives $\text{part}\langle \text{Inv}, \ell, \chi \in [-1; \infty] \rangle$, $\text{part}\langle \text{Inv}, \ell, \chi \in [-\infty; -2] \rangle$, and $\text{part}\langle \text{Inv}, \ell, y \in [1; \infty] \rangle$ are valid, since none of these partitioning directives would partition any equivalence class incurred by $\overset{d_c}{\sim}$. Meanwhile, partitioning directives $\text{part}\langle \text{Inv}, \ell, \chi \in [1; \infty] \rangle$ and $\text{part}\langle \text{Inv}, \ell, \chi \in [-9; 9] \rangle$ are invalid: For example, $[\chi \mapsto 0, y \mapsto 1]$ is equivalent to $[\chi \mapsto 5, y \mapsto 1]$ according to $\overset{d_c}{\sim}$, but $[\chi \mapsto 5, y \mapsto 1]$ belongs to the partition generated by $\text{part}\langle \text{Inv}, \ell, \chi \in [1; \infty] \rangle$ while $[\chi \mapsto 0, y \mapsto 1]$ does not belong to it.

In practice, it is difficult or even impossible to directly check if the condition (24) holds or not, since the number of equivalence classes (or, say, the size of quotient set of $\mathbb{M}$ by the equivalence relation $\overset{d_c}{\sim}$) is huge, making the cost of directly checking the condition (24) prohibitive. In the following, we try to transfer (24) into equivalent forms, which are easier to check in practice.

By the definition of $\overset{d_c}{\sim}$, it is trivial that: For every environment $\rho \in \mathbb{M} \backslash \gamma_{\mathbb{M}}(d_c)$, its equivalence class $[\rho]_{\underset{\sim}{d_c}} = \{\rho\}$. Since a singleton is either a subset of another set or completely disjoint from that set, the condition "$[\rho]_{\underset{\sim}{d_c}} \subseteq \gamma_{\mathbb{M}}(\mathrm{M}_p^\sharp) \ \vee \ [\rho]_{\underset{\sim}{d_c}} \cap \gamma_{\mathbb{M}}(\mathrm{M}_p^\sharp) = \emptyset$" trivially holds for every partitioning directive $\text{part}\langle \text{Inv}, \ell, \mathrm{M}_p^\sharp \rangle$ where $\mathrm{M}_p^\sharp \in \mathcal{D}_{\mathbb{M}}^\sharp$.

Therefore, the condition (24) is equivalent to the the following simplified one:

$$\forall \rho \in \gamma_{\mathbb{M}}(d_c).\ [\rho]_{\underset{\sim}{d_c}} \subseteq \gamma_{\mathbb{M}}\left(\mathrm{M}_p^\sharp\right) \ \vee \ [\rho]_{\underset{\sim}{d_c}} \cap \gamma_{\mathbb{M}}\left(\mathrm{M}_p^\sharp\right) = \emptyset. \tag{25}$$

Compared with checking the condition (24), the cost of checking the condition (25) is lower: Instead of checking the quotient set of $\mathbb{M}$ by $\overset{d_c}{\sim}$, now we need to check only the quotient set of $\gamma_{\mathbb{M}}(d_c)$ by $\overset{d_c}{\sim}$, whose size is reduced.

Further Refinement on the Definition. First, it is not hard to find that, for any abstract environment element $\mathrm{M}_p^\sharp \in \mathcal{D}_{\mathbb{M}}^\sharp$, we have:

$$\forall \rho \in \mathbb{M}.\ \left(\forall \chi \in \text{vars}\left(\mathrm{M}_p^\sharp\right).\ \rho(\chi) = \rho'(\chi)\right) \Rightarrow \left(\rho \in \gamma_{\mathbb{M}}\left(\mathrm{M}_p^\sharp\right) \Leftrightarrow \rho' \in \gamma_{\mathbb{M}}\left(\mathrm{M}_p^\sharp\right)\right). \tag{26}$$

The intuition is that, whether an environment belongs to $\gamma_{\mathbb{M}}(\mathrm{M}_p^\sharp)$ or not (or, say, whether an environment property $\mathrm{M}_p^\sharp$ holds or not) is not affected by the value of variables that are not used in $\mathrm{M}_p^\sharp$. For example, suppose $\mathrm{M}_p^\sharp = \chi \in [0; \infty]$ (where trivial constraints like "$y \in [-\infty; \infty]$" are assumed to be omitted in the abstract environment element), then whether an environment belongs to $\gamma_{\mathbb{M}}(\mathrm{M}_p^\sharp)$ or not is solely decided by the value of χ. Hence, if two environments have the same value of χ and may have different values of other variables, then both of them or neither of them belong to $\gamma_{\mathbb{M}}(\mathrm{M}_p^\sharp)$.

Second, given a cognizance directive $d_c \in \mathcal{D}^\sharp_{\mathbb{M}}$ and a partitioning directive $d_p = \text{part}\langle \text{Inv}, \ell, \mathrm{M}^\sharp_p\rangle$ where $\mathrm{M}^\sharp_p \in \mathcal{D}^\sharp_{\mathbb{M}}$, we define a new equivalence relation $\sim_{\mathrm{M}^\sharp_p \backslash d_c}$ on environments:

$$\sim_{\mathrm{M}^\sharp_p \backslash d_c} \quad \in \quad \wp(\mathbb{M} \times \mathbb{M}) \qquad \text{equivalence relation on environments}$$

$$\rho \sim_{\mathrm{M}^\sharp_p \backslash d_c} \rho' \quad \Leftrightarrow \quad \rho = \rho' \vee (\rho \in \gamma_{\mathbb{M}}(d_c) \wedge \rho' \in \gamma_{\mathbb{M}}(d_c) \wedge \forall \chi \in \text{vars}(\mathrm{M}^\sharp_p) \backslash \text{vars}(d_c).\ \rho(\chi) = \rho'(\chi)).$$

It is obvious that the size of each equivalence class by $\sim_{\mathrm{M}^\sharp_p \backslash d_c}$ is greater than $\overset{d_c}{\sim}$:

$$\forall \rho \in \mathbb{M}.\ [\rho]_{\overset{d_c}{\sim}} \subseteq [\rho]_{\sim_{\mathrm{M}^\sharp_p \backslash d_c}} \tag{27}$$

where $[\rho]_{\overset{d_c}{\sim}} = \{\rho' \in \mathbb{M} \mid \rho \overset{d_c}{\sim} \rho'\}$ and $[\rho]_{\sim_{\mathrm{M}^\sharp_p \backslash d_c}} = \{\rho' \in \mathbb{M} \mid \rho \sim_{\mathrm{M}^\sharp_p \backslash d_c} \rho'\}$.

Corollary 8. $\forall \rho \in \mathbb{M}.\ \forall \rho' \in [\rho]_{\sim_{\mathrm{M}^\sharp_p \backslash d_c}}.\ \exists \rho'' \in [\rho]_{\overset{d_c}{\sim}}.\ \forall \chi \in \mathit{vars}(\mathrm{M}^\sharp_p).\ \rho'(\chi) = \rho''(\chi)$.

Last, using the Corollary 8 and (26), we can prove that the condition (25) is equivalent to the condition (28), and get the following lemma:

Lemma 4. *A partitioning directive* $\mathrm{d}_p = \mathit{part}\langle \mathit{Inv}, \ell, \mathrm{M}^\sharp_p\rangle$ *is valid with respect to a cognizance directive* $\mathrm{d}_c \in \mathcal{D}^\sharp_{\mathbb{M}}$ *if and only if*

$$\forall \rho \in \gamma_{\mathbb{M}}(\mathrm{d}_c).\ [\rho]_{\sim_{\mathrm{M}^\sharp_p \backslash \mathrm{d}_c}} \subseteq \gamma_{\mathbb{M}}\left(\mathrm{M}^\sharp_p\right) \ \vee \ [\rho]_{\sim_{\mathrm{M}^\sharp_p \backslash \mathrm{d}_c}} \cap \gamma_{\mathbb{M}}\left(\mathrm{M}^\sharp_p\right) = \emptyset, \tag{28}$$

where $[\rho]_{\sim_{\mathrm{M}^\sharp_p \backslash \mathrm{d}_c}} = \{\rho' \in \mathbb{M} \mid \rho \sim_{\mathrm{M}^\sharp_p \backslash \mathrm{d}_c} \rho'\}$ *and* $\rho \sim_{\mathrm{M}^\sharp_p \backslash \mathrm{d}_c} \rho' \Leftrightarrow \rho = \rho' \vee (\rho \in \gamma_{\mathbb{M}}(\mathrm{d}_c) \wedge \rho' \in \gamma_{\mathbb{M}}(\mathrm{d}_c) \wedge \forall \chi \in \mathit{vars}(\mathrm{M}^\sharp_p) \backslash \mathit{vars}(\mathrm{d}_c).\ \rho(\chi) = \rho'(\chi))$.

Intuitively, compared with checking the condition (25), the cost of checking (28) is further reduced. Essentially, for both conditions, we need to partition the set of environments $\gamma_{\mathbb{M}}(d_c)$ into equivalence classes and check if there exists any equivalence class that overlaps with $\gamma_{\mathbb{M}}(\mathrm{M}^\sharp_p)$. Since the definition of $\sim_{\mathrm{M}^\sharp_p \backslash d_c}$ is looser than $\overset{d_c}{\sim}$, the size of each equivalence class created by $\sim_{\mathrm{M}^\sharp_p \backslash d_c}$ is larger, thus the number of equivalence classes that need to be checked is smaller.

(2) Checking the Validity of Partitioning Directives in the Abstract. Although we have formally defined the validity of a partitioning directive $d_p = \text{part}\langle \text{Inv}, \ell, \mathrm{M}^\sharp_p\rangle$ with respect to a cognizance directive $d_c \in \mathcal{D}^\sharp_{\mathbb{M}}$, it is impractical to directly use those definitions to check the validity of partitioning directives, since it requires to compare sets of environments in the concrete. The objective of this section is to propose a sound checking approach, which guarantees that if a partitioning directive is determined as valid in the abstract, then it is indeed valid in the concrete.

To begin with, we consider the abstract environment domains $\mathcal{D}^\sharp_{\mathbb{M}}$ that have a Galois connection with the concrete environment domain, i.e., $\langle \wp(\mathbb{M}), \subseteq\rangle \underset{\alpha_{\mathbb{M}}}{\overset{\gamma_{\mathbb{M}}}{\leftrightarrows}} \langle \mathcal{D}^\sharp_{\mathbb{M}}, \sqsubseteq^\sharp_{\mathbb{M}}\rangle$. Such abstract domains include but are not limited to the interval domain and the octagon domain. In this case, we have: $\alpha_{\mathbb{M}}([\rho]_{\sim_{\mathrm{M}^\sharp_p \backslash d_c}}) \sqsubseteq^\sharp_{\mathbb{M}} \mathrm{M}^\sharp_p \Leftrightarrow [\rho]_{\sim_{\mathrm{M}^\sharp_p \backslash d_c}} \subseteq \gamma_{\mathbb{M}}(\mathrm{M}^\sharp_p)$ and $\alpha_{\mathbb{M}}([\rho]_{\sim_{\mathrm{M}^\sharp_p \backslash d_c}}) \sqcap^\sharp_{\mathbb{M}} \mathrm{M}^\sharp_p = \bot^\sharp_{\mathbb{M}} \Rightarrow [\rho]_{\sim_{\mathrm{M}^\sharp_p \backslash d_c}} \cap \gamma_{\mathbb{M}}(\mathrm{M}^\sharp_p) = \emptyset$. Therefore, we can infer a sufficient condition for (28) to hold:

$$\forall \rho \in \gamma_{\mathbb{M}}(d_c).\ \alpha_{\mathbb{M}}\left([\rho]_{\sim_{\mathrm{M}^\sharp_p \backslash d_c}}\right) \sqsubseteq^\sharp_{\mathbb{M}} \mathrm{M}^\sharp_p \ \vee \ \alpha_{\mathbb{M}}\left([\rho]_{\sim_{\mathrm{M}^\sharp_p \backslash d_c}}\right) \sqcap^\sharp_{\mathbb{M}} \mathrm{M}^\sharp_p = \bot^\sharp_{\mathbb{M}}. \tag{29}$$

More generally, for abstract domains (e.g., the polyhedron domain) that do not have a corresponding abstraction function $\alpha_{\mathbb{M}} \in \wp(\mathbb{M}) \mapsto \mathcal{D}^\sharp_{\mathbb{M}}$, we can use the following condition instead as a sufficient condition to check (28):

$$\forall \rho \in \gamma_{\mathbb{M}}(d_c).\ \exists d'_c \in \mathcal{D}^\sharp_{\mathbb{M}}.\ [\rho]_{\sim_{\mathrm{M}^\sharp_p \backslash d_c}} \subseteq \gamma_{\mathbb{M}}(d'_c) \wedge \left(d'_c \sqsubseteq^\sharp_{\mathbb{M}} \mathrm{M}^\sharp_p \ \vee\ d'_c \sqcap^\sharp_{\mathbb{M}} \mathrm{M}^\sharp_p = \bot^\sharp_{\mathbb{M}}\right). \tag{30}$$

Now the question is: How to find $d'_c \in \mathcal{D}^\sharp_{\mathbb{M}}$ such that $[\rho]_{\sim_{\mathrm{M}^\sharp_p \backslash d_c}} \subseteq \gamma_{\mathbb{M}}(d'_c)$? Suppose $\mathcal{D}^\sharp_{\mathbb{M}}$ is a classic numerical domain (including intervals, octagons, and polyhedra), $\mathrm{vars}(\mathrm{M}^\sharp_p) \backslash \mathrm{vars}(d_c) = \{\mathcal{X}_1, \ldots, \mathcal{X}_n\}$ that are denoted as $\vec{\mathcal{X}}$. Then, for any environment $\rho \in \gamma_{\mathbb{M}}(d_c)$, its equivalence class $[\rho]_{\sim_{\mathrm{M}^\sharp_p \backslash d_c}}$ can be soundly over-approximated by $d'_c = d_c \sqcap^\sharp_{\mathbb{M}} (\vec{\mathcal{X}} = \vec{v})$, where $\vec{v}$ is the values of $\vec{\mathcal{X}}$ in ρ. Therefore, we can infer another sufficient condition for (28) to hold, which is more convenient to check.

Lemma 5. *A partitioning directive* $\mathrm{d}_p = \mathit{part}\langle \mathit{Inv},\ \ell,\ \mathrm{M}^\sharp_p\rangle$ *is valid with respect to a cognizance directive* $\mathrm{d}_c \in \mathcal{D}^\sharp_{\mathbb{M}}$ *if*

$$\forall \vec{v} \in \mathbb{V}^n.\ \mathrm{d}'_c = \left(\mathrm{d}_c \sqcap^\sharp_{\mathbb{M}} \vec{\mathcal{X}} = \vec{v}\right) \wedge \left(\mathrm{d}'_c \sqsubseteq^\sharp_{\mathbb{M}} \mathrm{M}^\sharp_p \vee \mathrm{d}'_c \sqcap^\sharp_{\mathbb{M}} \mathrm{M}^\sharp_p = \bot^\sharp_{\mathbb{M}}\right), \tag{31}$$

where $\vec{\mathcal{X}} = \mathit{vars}(\mathrm{M}^\sharp_p) \backslash \mathit{vars}(\mathrm{d}_c) = \{\mathcal{X}_1, \ldots, \mathcal{X}_n\}$.

More specifically, $\vec{\mathcal{X}} = \vec{v}$ is expressed as "$\mathcal{X}_1 \in [v_1; v_1] \wedge \ldots \wedge \mathcal{X}_n \in [v_n; v_n]$" in the interval domain, and as "$\mathcal{X}_1 \le v_1 \wedge -\mathcal{X}_1 \le -v_1 \wedge \ldots \wedge \mathcal{X}_n \le v_n \wedge -\mathcal{X}_n \le -v_n$" in the octagon/polyhedron domain.

However, directly checking the condition (31) is still costly, thus we discuss a few special cases that are common and easy to check in practice:

(S1) If $\mathrm{vars}(\mathrm{M}^\sharp_p) \cap \mathrm{vars}(d_c) = \emptyset$ (or, say, $\mathrm{M}^\sharp_p$ and d_c do not have commonly used variables, e.g., $\mathrm{M}^\sharp_p = \mathcal{X} \le 1$ and $d_c = y \le 0$): In this case, $\vec{\mathcal{X}} = \mathrm{vars}(\mathrm{M}^\sharp_p) \backslash \mathrm{vars}(d_c) = \mathrm{vars}(\mathrm{M}^\sharp_p)$ includes all the variables used in $\mathrm{M}^\sharp_p$, hence we have $\forall \vec{v} \in \mathbb{V}^n.\ (\vec{\mathcal{X}} = \vec{v} \sqsubseteq^\sharp_{\mathbb{M}} \mathrm{M}^\sharp_p) \vee (\vec{\mathcal{X}} = \vec{v} \sqcap^\sharp_{\mathbb{M}} \mathrm{M}^\sharp_p = \bot^\sharp_{\mathbb{M}})$. Since $(d_c \sqcap^\sharp_{\mathbb{M}} \vec{\mathcal{X}} = \vec{v}) \sqsubseteq^\sharp_{\mathbb{M}} \vec{\mathcal{X}} = \vec{v}$, condition (31) always holds. Thus, if $\mathrm{vars}(\mathrm{M}^\sharp_p) \cap \mathrm{vars}(d_c) = \emptyset$, then the partitioning directive $d_p = \mathrm{part}\langle \mathrm{Inv},\ \ell,\ \mathrm{M}^\sharp_p\rangle$ is valid with respect to $d_c \in \mathcal{D}^\sharp_{\mathbb{M}}$.

(S2) If $\mathrm{vars}(\mathrm{M}^\sharp_p) \backslash \mathrm{vars}(d_c) = \emptyset$ (or, say, every variable used in $\mathrm{M}^\sharp_p$ is also used in d_c, e.g., $\mathrm{M}^\sharp_p = \mathcal{X} \le 1$ and $d_c = \mathcal{X} \le 0 \wedge y \le 0$): In this case, $\vec{\mathcal{X}} = \mathrm{vars}(\mathrm{M}^\sharp_p) \backslash \mathrm{vars}(d_c) = \emptyset$, hence $d'_c = d_c$ in the condition (31). It is obvious that the condition (31) is equivalent to $d_c \sqsubseteq^\sharp_{\mathbb{M}} \mathrm{M}^\sharp_p \vee d_c \sqcap^\sharp_{\mathbb{M}} \mathrm{M}^\sharp_p = \bot^\sharp_{\mathbb{M}}$. Thus, when $\mathrm{vars}(\mathrm{M}^\sharp_p) \backslash \mathrm{vars}(d_c) = \emptyset$, the partitioning directive $d_p = \mathrm{part}\langle \mathrm{Inv},\ \ell,\ \mathrm{M}^\sharp_p\rangle$ is valid with respect to the cognizance directive $d_c \in \mathcal{D}^\sharp_{\mathbb{M}}$ if and only if $d_c \sqsubseteq^\sharp_{\mathbb{M}} \mathrm{M}^\sharp_p \vee d_c \sqcap^\sharp_{\mathbb{M}} \mathrm{M}^\sharp_p = \bot^\sharp_{\mathbb{M}}$ holds.

(S3) If $d_c \sqsubseteq^\sharp_{\mathbb{M}} \mathrm{M}^\sharp_p \vee d_c \sqcap^\sharp_{\mathbb{M}} \mathrm{M}^\sharp_p = \bot^\sharp_{\mathbb{M}}$ holds: Since $d'_c = (d_c \sqcap^\sharp_{\mathbb{M}} \vec{\mathcal{X}} = \vec{v}) \sqsubseteq^\sharp_{\mathbb{M}} d_c$, we always have $d'_c \sqsubseteq^\sharp_{\mathbb{M}} \mathrm{M}^\sharp_p \vee d'_c \sqcap^\sharp_{\mathbb{M}} \mathrm{M}^\sharp_p = \bot^\sharp_{\mathbb{M}}$. Thus, if $d_c \sqsubseteq^\sharp_{\mathbb{M}} \mathrm{M}^\sharp_p \vee d_c \sqcap^\sharp_{\mathbb{M}} \mathrm{M}^\sharp_p = \bot^\sharp_{\mathbb{M}}$, then the partitioning directive $d_p = \mathrm{part}\langle \mathrm{Inv},\ \ell,\ \mathrm{M}^\sharp_p\rangle$ is valid with respect to the cognizance directive $d_c \in \mathcal{D}^\sharp_{\mathbb{M}}$.

To sum up the lemma 5 and special cases (S1–S3), now we propose a sound approach to check if a partitioning directive $d_p = \mathrm{part}\langle \mathrm{Inv},\ \ell,\ \mathrm{M}^\sharp_p\rangle$ is valid with respect to the cognizance directive $d_c \in \mathcal{D}^\sharp_{\mathbb{M}}$ and formalize it as a function $\mathrm{isValid}_d \in \mathcal{D}^\sharp_{\mathbb{M}} \mapsto (\mathcal{D}^\sharp_{\mathbb{M}} \mapsto \mathbb{B})$ such that $\mathrm{isValid}_d(d_c, \mathrm{M}^\sharp_p)$ returns whether the partitioning directive is valid or not. Observe that $\mathrm{isValid}_d(d_c, \mathrm{M}^\sharp_p)$ returns false in Case (S2), since the condition for Case (S3) is already checked and it is false.

bool isValid$_d(d_c, \mathrm{M}_p^\sharp)$ {
  If (vars($\mathrm{M}_p^\sharp$) ∩ vars(d_c) = ∅) return true; // Case (S1)
  If ($d_c \sqsubseteq_{\mathbb{M}}^\sharp \mathrm{M}_p^\sharp \vee d_c \sqcap_{\mathbb{M}}^\sharp \mathrm{M}_p^\sharp = \bot_{\mathbb{M}}^\sharp$) return true; // Case (S3)
  If (vars($\mathrm{M}_p^\sharp$)\vars(d_c) = ∅) return false; // Case (S2)
  Check (31) and return the result. // Lemma 5
}

By the proof of Lemma 5 and the explanation of (S1–S3), we know that the above approach is sound. More precisely, if the function isValid$_d(d_c, \mathrm{M}_p^\sharp)$ returns true, then the partitioning directive $d_p = \mathrm{part}\langle \mathrm{Inv},\ \ell,\ \mathrm{M}_p^\sharp\rangle$ must be valid with respect to the cognizance directive $d_c \in \mathcal{D}_{\mathbb{M}}^\sharp$ in the concrete (def. 2).

Special Case of the Interval Domain. In particular (but not necessary), the implementation of isValid$_d(d_c, \mathrm{M}_p^\sharp)$ can be further simplified if the abstract environment domain $\mathcal{D}_{\mathbb{M}}^\sharp$ is the interval domain. Since the interval domain cannot express the relation among variables and every element in the interval domain is simply a conjunction of interval constraints on a set of variables, for any $\mathrm{M}_p^\sharp, d_c \in \mathcal{D}_{\mathbb{M}}^\sharp$, the constraints in $\mathrm{M}_p^\sharp$ can be split into two parts: $\mathrm{M}_p^\sharp|_{\mathrm{vars}(\mathrm{M}_p^\sharp)\setminus\mathrm{vars}(d_c)}$ denotes the constraints on the variables in vars($\mathrm{M}_p^\sharp$)\vars(d_c), and $\mathrm{M}_p^\sharp|_{\mathrm{vars}(\mathrm{M}_p^\sharp)\cap\mathrm{vars}(d_c)}$ denotes the constraints on the variables in vars($\mathrm{M}_p^\sharp$) ∩ vars(d_c). When the set vars($\mathrm{M}_p^\sharp$)\vars(d_c) (respectively, vars($\mathrm{M}_p^\sharp$) ∩ vars(d_c)) is empty, $\mathrm{M}_p^\sharp|_{\mathrm{vars}(\mathrm{M}_p^\sharp)\setminus\mathrm{vars}(d_c)}$ (respectively, $\mathrm{M}_p^\sharp|_{\mathrm{vars}(\mathrm{M}_p^\sharp)\cap\mathrm{vars}(d_c)}$) denotes $\top_{\mathbb{M}}^\sharp$. Then, condition (31) is equivalent to: $\forall \vec{v} \in \mathbb{V}^n.\ (d_c \sqsubseteq_{\mathbb{M}}^\sharp \mathrm{M}_p^\sharp|_{\mathrm{vars}(\mathrm{M}_p^\sharp)\cap\mathrm{vars}(d_c)} \wedge \vec{\chi} = \vec{v} \sqsubseteq_{\mathbb{M}}^\sharp \mathrm{M}_p^\sharp|_{\mathrm{vars}(\mathrm{M}_p^\sharp)\setminus\mathrm{vars}(d_c)}) \vee (d_c \sqcap_{\mathbb{M}}^\sharp \mathrm{M}_p^\sharp|_{\mathrm{vars}(\mathrm{M}_p^\sharp)\cap\mathrm{vars}(d_c)} = \bot_{\mathbb{M}}^\sharp \wedge \vec{\chi} = \vec{v} \sqcap_{\mathbb{M}}^\sharp \mathrm{M}_p^\sharp|_{\mathrm{vars}(\mathrm{M}_p^\sharp)\setminus\mathrm{vars}(d_c)} = \bot_{\mathbb{M}}^\sharp)$.

Since $(\vec{\chi} = \vec{v} \sqsubseteq_{\mathbb{M}}^\sharp \mathrm{M}_p^\sharp|_{\mathrm{vars}(\mathrm{M}_p^\sharp)\setminus\mathrm{vars}(d_c)}) \vee (\vec{\chi} = \vec{v} \sqcap_{\mathbb{M}}^\sharp \mathrm{M}_p^\sharp|_{\mathrm{vars}(\mathrm{M}_p^\sharp)\setminus\mathrm{vars}(d_c)} = \bot_{\mathbb{M}}^\sharp)$ always hold in the above condition, the condition (31) is equivalent to:

$$d_c \sqsubseteq_{\mathbb{M}}^\sharp \mathrm{M}_p^\sharp|_{\mathrm{vars}(\mathrm{M}_p^\sharp)\cap\mathrm{vars}(d_c)} \vee d_c \sqcap_{\mathbb{M}}^\sharp \mathrm{M}_p^\sharp|_{\mathrm{vars}(\mathrm{M}_p^\sharp)\cap\mathrm{vars}(d_c)} = \bot_{\mathbb{M}}^\sharp, \tag{32}$$

and the implementation of isValid$_d(d_c, \mathrm{M}_p^\sharp)$ for the interval domain could be simplified into checking if the condition (32) holds.

For example, if $d_c = \chi \in [0;\infty] \wedge y \in [0;\infty]$ and $d_p = \mathrm{part}\langle \mathrm{Inv},\ell, \mathrm{M}_p^\sharp = y \in [-5;5] \wedge z \in [-5;5]\rangle$, then $\mathrm{M}_p^\sharp|_{\mathrm{vars}(\mathrm{M}_p^\sharp)\cap\mathrm{vars}(d_c)} = y \in [-5;5]$, since vars($\mathrm{M}_p^\sharp$) ∩ vars($d_c$) = {$y$}. It is not hard to see that $d_c \not\sqsubseteq_{\mathbb{M}}^\sharp y \in [-5;5]$ and $d_c \sqcap_{\mathbb{M}}^\sharp y \in [-5;5] = \chi \in [0;\infty] \wedge y \in [0;5] \neq \bot_{\mathbb{M}}^\sharp$, thus the condition (32) does not hold, and d_p is invalid with respect to d_c.

Example 21 (Checking the Validity of Partitioning Directives). Here, we give some examples of checking the validity of partitioning directive d_p with respect to some cognizance directive d_c by the approach proposed in this section.

(i) $d_c = \chi \leq -1$ and $d_p = \mathrm{part}\langle \mathrm{Inv},\ \ell,\ \mathrm{M}_p^\sharp = y \leq 0\rangle$: d_c indicates that the observer does not know the exact value of χ if it is negative, and d_p would like to generate a partition such that the value of y is less than 0. Since vars($\mathrm{M}_p^\sharp$) ∩ vars(d_c) = ∅ (Case (S1)) holds, the partitioning directive d_p is valid with respect to d_c.

(ii) $d_c = \chi \leq -1$ and $d_p = \mathrm{part}\langle \mathrm{Inv},\ \ell,\ \mathrm{M}_p^\sharp = \chi \leq 0\rangle$: d_c indicates that the observer does not know the exact value of χ if it is negative, and d_p intends to generate a partition such that the

value of χ is less than 0. It is obvious that $d_c \sqsubseteq^\sharp_{\mathbb{M}} \mathrm{M}^\sharp_p$, thus this is case (S3) and the partitioning directive d_p is valid.

(iii) $d_c = \chi \leq 0$ and $d_p = \mathrm{part}\langle \mathrm{Inv},\ \ell,\ \mathrm{M}^\sharp_p = \chi \leq -1\rangle$: d_c indicates that the observer does not know the exact value of χ if it is negative or zero, and d_p intends to generate a partition such that the value of χ is negative. In this example, $\mathrm{vars}(\mathrm{M}^\sharp_p)\backslash\mathrm{vars}(d_c) = \emptyset$, and it is easy to see that $d_c \not\sqsubseteq^\sharp_{\mathbb{M}} \mathrm{M}^\sharp_p$ and $d_c \sqcap^\sharp_{\mathbb{M}} \mathrm{M}^\sharp_p \neq \bot^\sharp_{\mathbb{M}}$, thus this is case (S2) and the partitioning directive d_p is invalid with respect to d_c.

(iv) $d_c = \chi \leq y$ and $d_p = \mathrm{part}\langle \mathrm{Inv},\ \ell,\ \mathrm{M}^\sharp_p = \chi \leq z\rangle$: d_c indicates that the observer does not know the exact value of χ and y when $\chi \leq y$, but knows the relation between χ and y; and d_p would like to generate a partition such that $\chi \leq z$. It is not hard to see that none of (S1–S3) holds in this example, thus we need to directly check the condition (31), which is $\forall v \in \mathbb{V}.\ d'_c = (\chi \leq y \wedge z = v) \wedge (d'_c \sqsubseteq^\sharp_{\mathbb{M}} \chi \leq z \vee d'_c \sqcap^\sharp_{\mathbb{M}} \chi \leq z = \bot^\sharp_{\mathbb{M}})$. Such a condition does not hold: for example, if $v = 0$, then $d'_c = (\chi \leq y \wedge z = 0)$, hence we have $d_c \not\sqsubseteq^\sharp_{\mathbb{M}} \chi \leq z$ and $d_c \sqcap^\sharp_{\mathbb{M}} \chi \leq z = (\chi \leq y \wedge z = 0 \wedge \chi \leq 0) \neq \bot^\sharp_{\mathbb{M}}$. Therefore, the partitioning directive d_p is invalid with respect to d_c.

(v) $d_c = \chi \leq y \wedge y \leq z$ and $d_p = \mathrm{part}\langle \mathrm{Inv},\ \ell,\ \mathrm{M}^\sharp_p = z < \chi \wedge w \leq 0\rangle$: In this example, $\mathrm{vars}(\mathrm{M}^\sharp_p)\backslash\mathrm{vars}(d_c) = \{w\}$, and none of (S1–S3) holds in this example, thus we need to directly check the condition (31), which is always true, because $\forall v \in \mathbb{V}.\ (\chi \leq y \wedge y \leq z \wedge w = v) \sqcap^\sharp_{\mathbb{M}} (z < \chi \wedge w \leq 0) = \bot^\sharp_{\mathbb{M}}$. Therefore, the partitioning directive d_p is valid with respect to d_c.

(3) Checking the Validity of a Partition Function in the Abstract. Up to now, we have discussed how to check if a single partitioning directive is valid with respect to a cognizance directive. For a program, the user specifies an abstract cognizance function $\mathbb{C}^\sharp \in \mathbb{L} \mapsto \wp(\mathcal{D}^\sharp_{\mathbb{M}})$, and there are typically more than one partitioning directive of the form $\mathrm{part}\langle \mathrm{Inv},\ \ell,\ \mathrm{M}^\sharp_p\rangle$, hence we need to check the validity of all these partitioning directives with respect to the whole cognizance function. For the sake of clarity, here we rephrase the set of partitioning directives of the form $\mathrm{part}\langle \mathrm{Inv},\ \ell,\ \mathrm{M}^\sharp_p\rangle$ as a *partition function* $\mathbb{P}^\sharp \in \mathbb{L} \mapsto \wp(\mathcal{D}^\sharp_{\mathbb{M}})$, such that $\forall \ell \in \mathbb{L}.\ \forall \mathrm{M}^\sharp_p \in \mathbb{P}^\sharp(\ell).\ \mathrm{part}\langle \mathrm{Inv},\ \ell,\ \mathrm{M}^\sharp_p\rangle$ is a partitioning directive in the program.

Formally, here we define a function $\mathrm{isValid}_{\mathbb{P}}$ that checks if a partition function $\mathbb{P}^\sharp$ is valid with respect to a cognizance function $\mathbb{C}^\sharp$:

$$\mathrm{isValid}_{\mathbb{P}} \in \left(\mathbb{L} \mapsto \wp\left(\mathcal{D}^\sharp_{\mathbb{M}}\right)\right) \mapsto \left(\left(\mathbb{L} \mapsto \wp\left(\mathcal{D}^\sharp_{\mathbb{M}}\right)\right) \mapsto \mathbb{B}\right) \qquad \text{Validity of Partition}$$

$$\mathrm{isValid}_{\mathbb{P}}(\mathbb{C}^\sharp, \mathbb{P}^\sharp) \triangleq \begin{cases} \text{true} & \text{if } \forall \ell \in \mathbb{L}.\ \forall d_c \in \mathbb{C}^\sharp(\ell), \mathrm{M}^\sharp_p \in \mathbb{P}^\sharp(\ell).\ \mathrm{isValid}_d\left(d_c, \mathrm{M}^\sharp_p\right) \\ \text{false} & \text{otherwise.} \end{cases}$$

That is to say, a partition function $\mathbb{P}^\sharp$ is valid with respect to an abstract cognizance function $\mathbb{C}^\sharp$ if and only if, at each program point ℓ, every partitioning directive specified by $\mathbb{P}^\sharp$ is valid with respect to every cognizance directive assigned by $\mathbb{C}^\sharp$.

Recall that the partitioning directives are designed to create new partitions when constructing trace partitioning automata, and the sole purpose of checking if a partition function $\mathbb{P}^\sharp$ is valid with respect to an abstract cognizance function $\mathbb{C}^\sharp$ is to ensure that any two indistinguishable traces would not be partitioned into different partitions, thus are always represented by the same path in the constructed trace partitioning automaton.

Theorem 2. *If the partition function $\mathbb{P}^\sharp$ is valid with respect to the cognizance function $\mathbb{C}^\sharp$, then every two indistinguishable traces $\sigma \overset{\mathbb{C}^\sharp}{\sim} \sigma'$ must belong to the same partition created by $\mathbb{P}^\sharp$ at every program point along the execution.*

Formally, $\forall \mathbb{C}^\sharp, \mathbb{P}^\sharp \in \mathbb{L} \mapsto \wp(\mathcal{D}^\sharp_{\mathbb{M}})$. $isValid_{\mathbb{P}}(\mathbb{C}^\sharp, \mathbb{P}^\sharp) \Rightarrow (\forall \sigma, \sigma' \in \mathbb{S}^{*\infty}.\ \sigma \overset{\mathbb{C}^\sharp}{\sim} \sigma' \Rightarrow (\forall i \in [0, |\sigma|).\ \exists \ell \in \mathbb{L}, \rho, \rho' \in \mathbb{M}.\ \sigma_{[i]} = \langle \ell,\ \rho\rangle \wedge \sigma'_{[i]} = \langle \ell,\ \rho'\rangle \wedge \forall \mathrm{M}^\sharp_p \in \mathbb{P}^\sharp(\ell).\ \rho \in \gamma_{\mathbb{M}}(\mathrm{M}^\sharp_p) \Leftrightarrow \rho' \in \gamma_{\mathbb{M}}(\mathrm{M}^\sharp_p)))$.

6.2.3 Revising Partitioning Directives to Be Valid. In the previous sections, we have introduced the method to check the validity of partitioning directives (or, say, the partition function), while the approach of creating partitioning directives will be discussed in Section 7. Intuitively, we could create partitioning directives based on the information provided by the cognizance function, such that the created partitioning directives are always valid. For example, if the cognizance function $\mathbb{C}^\sharp(\ell) = \{\chi < 0, \chi \geq 0\}$ indicates that the observer knows the sign of χ at point ℓ, but not the exact value of χ. It is intuitive to create two partitions according to the sign of χ at point ℓ: part⟨Inv, ℓ, $\chi < 0$⟩ and part⟨Inv, ℓ, $\chi \geq 0$⟩, both of which can be simply proved to be valid with respect to $\mathbb{C}^\sharp$. However, this is not always the case, and we may want to create partitioning directives based on some other criteria, which may bring us invalid partitioning directives. Thus, a missing part here is: What shall we do if a certain partitioning directive $d_p = \mathrm{part}\langle \mathrm{Inv},\ \ell,\ \mathrm{M}^\sharp_p\rangle$ (or $\mathrm{M}^\sharp_p$ for short) is found invalid with respect to a cognizance directive d_c at point ℓ (i.e., $\mathrm{isValid}_d(d_c, \mathrm{M}^\sharp_p) = \mathrm{false}$)?

Obviously, we can simply discard the partitioning directive d_p, and the correspondingly constructed trace partitioning automaton is still guaranteed to be sound. However, this may incur the loss of precision in the forward reachability analysis, which further affects the result of abstract responsibility analysis.

Alternatively, we can retrieve the validity by revising $\mathrm{M}^\sharp_p$. By the definition of $\mathrm{isValid}_d(d_c, \mathrm{M}^\sharp_p)$, we know that $d_c \sqsubseteq^\sharp_{\mathbb{M}} \mathrm{M}^\sharp_p \vee d_c \sqcap^\sharp_{\mathbb{M}} \mathrm{M}^\sharp_p = \bot^\sharp_{\mathbb{M}}$ does not hold. That is to say, $d_c \not\sqsubseteq^\sharp_{\mathbb{M}} \mathrm{M}^\sharp_p$ and $d_c \sqcap^\sharp_{\mathbb{M}} \mathrm{M}^\sharp_p \neq \bot^\sharp_{\mathbb{M}}$, thus there are two possible cases:

(1) $\mathrm{M}^\sharp_p \sqsubset^\sharp_{\mathbb{M}} d_c$: $\mathrm{M}^\sharp_p$ is strictly less than d_c, or $\gamma_{\mathbb{M}}(\mathrm{M}^\sharp_p) \subsetneq \gamma_{\mathbb{M}}(d_c)$. In this case, we can just use d_c as a new partitioning directive to replace $\mathrm{M}^\sharp_p$, i.e., we define $\mathrm{M}^{\sharp\prime}_p = d_c$, and $\mathrm{M}^{\sharp\prime}_p$ is obviously valid with respect to d_c. For example, $d_p = \mathrm{part}\langle \mathrm{Inv},\ \ell,\ \mathrm{M}^\sharp_p = \chi < 0\rangle$ is invalid with respect to $d_c = \chi \leq 0$, and we can replace it by a new partitioning directive part⟨Inv, ℓ, $\mathrm{M}^\sharp_p = \chi \leq 0$⟩, which is trivially valid.

(2) $\mathrm{M}^\sharp_p \not\sqsubseteq^\sharp_{\mathbb{M}} d_c \wedge d_c \not\sqsubseteq^\sharp_{\mathbb{M}} \mathrm{M}^\sharp_p \wedge d_c \sqcap^\sharp_{\mathbb{M}} \mathrm{M}^\sharp_p \neq \bot^\sharp_{\mathbb{M}}$: $\mathrm{M}^\sharp_p$ overlaps with d_c, and they are incomparable. In this case, there are two possible ways to create new partitioning directives to replace $\mathrm{M}^\sharp_p$:

(a) Define a new partitioning directive $\mathrm{M}^{\sharp\prime}_p = \mathrm{M}^\sharp_p \sqcup^\sharp_{\mathbb{M}} d_c$.

Obviously, $\mathrm{M}^{\sharp\prime}_p$ is valid with respect to d_c, since $d_c \sqsubseteq^\sharp_{\mathbb{M}} \mathrm{M}^{\sharp\prime}_p$. For example, $d_p = \mathrm{part}\langle \mathrm{Inv},\ \ell,\ \mathrm{M}^\sharp_p = \chi \in [1; \infty]\rangle$ is invalid with respect to the cognizance directive $d_c = \chi \in [0; 1]$ that indicates the observer cannot distinguish the value 0 and 1 of χ, then we can replace it by a new partitioning directive part⟨Inv, ℓ, $\mathrm{M}^\sharp_p = \chi \in [0; \infty]\rangle$. It is worth mentioning that convex abstract domains (e.g., polyhedra) cannot exactly represent unions, which must be over-approximated (e.g., the convex hull for polyhedra). If the incurred loss of precision is unacceptable and we need the exact union, then we could use the disjunctive

completion as a new partitioning directive, although it may be costly and does not scale well.

(b) Or, we split $\mathrm{M}_p^\sharp \sqcup_{\mathbb{M}}^\sharp d_c$ by defining two new partitioning directives: $\mathrm{M}_p^{\sharp\prime} = d_c$ and $\mathrm{M}_p^{\sharp\prime\prime} = \mathrm{M}_p^\sharp \sqcap_{\mathbb{M}}^\sharp \neg d_c$.

It is not hard to see that $d_c \sqsubseteq_{\mathbb{M}}^\sharp \mathrm{M}_p^{\sharp\prime}$ and $d_c \sqcap_{\mathbb{M}}^\sharp \mathrm{M}_p^{\sharp\prime\prime} = \bot_{\mathbb{M}}^\sharp$, thus these two new partitioning directives are valid. Specially, $\mathrm{M}_p^{\sharp\prime\prime}$ under-approximates $\mathrm{M}_p^\sharp$, thus the correspondingly created partition preserves the desired property of partitioning by $\mathrm{M}_p^\sharp$. However, the classic numerical domains (such as intervals, octagons, polyhedra) do not support the complement operation $\neg$, e.g., the complement of a polyhedron is a disjunction of affine inequalities, thus $\mathrm{M}_p^\sharp \sqcap_{\mathbb{M}}^\sharp \neg d_c$ may not be expressed by a single element in $\mathcal{D}_{\mathbb{M}}^\sharp$. If this happens, then instead of defining a single partitioning directive to represent $\mathrm{M}_p^\sharp \sqcap_{\mathbb{M}}^\sharp \neg d_c$, we define a list of partitioning directives, each of which is a conjunction of $\mathrm{M}_p^\sharp$ and an affine inequality from $\neg d_c$. For example, $d_p = \mathrm{part}\langle \mathrm{Inv},\ell,\ \mathrm{M}_p^\sharp = \chi \leq 10\rangle$ is invalid with respect to the cognizance directive $d_c = \chi \geq 0 \wedge y > 0$. The complement of d_c is the disjunction of $\chi < 0$ and $y \leq 0$. Thus, we create three new partitioning directives: $\mathrm{M}_p^{\sharp\prime} = d_c = \chi \geq 0 \wedge y > 0$, $\mathrm{M}_p^{\sharp\prime\prime} = \mathrm{M}_p^\sharp \sqcap_{\mathbb{M}}^\sharp \chi < 0 = \chi < 0$, and $\mathrm{M}_p^{\sharp\prime\prime\prime} = \mathrm{M}_p^\sharp \sqcap_{\mathbb{M}}^\sharp y \leq 0 = \chi \leq 10 \wedge y \leq 0$. In addition, when the number of affine inequality from $\neg d_c$ is large, we could heuristically select part of them to create new partitioning directives, reducing the cost incurred by trace partitioning without harm to the soundness.

To sum up, in this section, we have discussed the user specification of system behaviors and cognizance in the abstract, proposed a sound approach to check if the partitioning directives are valid with respect to the user specified cognizance, and sketched some possible methods to retrieve the validity for invalid partitioning directives.

7 ABSTRACT RESPONSIBILITY ANALYSIS

The concrete responsibility analysis $\alpha_R(\llbracket \mathrm{P} \rrbracket^{\mathrm{Max}}, \mathcal{L}^{\mathrm{Max}}, \mathbb{C}, \mathcal{B}, \mathcal{T})$ proposed in Section 3.2 is undecidable, and an implementation of it has to abstract sets of finite or infinite traces involved in $\llbracket \mathrm{P} \rrbracket^{\mathrm{Max}}$, $\mathcal{L}^{\mathrm{Max}}$, $\mathbb{C}$, $\mathcal{B}$, and $\mathcal{T}$. Up to now, we have discussed the abstraction of maximal trace semantics $\llbracket \mathrm{P} \rrbracket^{\mathrm{Max}}$ by trace partitioning automata that are constructed by over-approximating forward reachability analysis (Section 4.2) with trace partitioning (Section 5), the abstraction of system behaviors $\mathcal{B}$ by abstract invariants (Section 6.1), and the abstraction of cognizance $\mathbb{C}$ by abstract cognizance function (Section 6.2). Moreover, it is assumed that the lattice of behaviors $\mathcal{L}^{\mathrm{Max}}$ consists of only $\mathcal{B}$ and its complement (besides the top and bottom), and the set of traces to be analyzed $\mathcal{T}$ is the whole maximal trace semantics, thus all the components in responsibility analysis have been abstracted.

In this section, we propose the framework of responsibility analysis in the abstract, which essentially consists of an iteration of forward (possible success) reachability analysis with trace partitioning and backward impossible failure accessibility analysis. In addition, this abstract responsibility analysis is proved to be sound.

7.1 The Framework of Abstract Responsibility Analysis

As shown in Figure 18, given a program P with the user-specified behavior of interest $\mathcal{B}^\sharp \in \mathbb{L} \mapsto \mathcal{D}_{\mathbb{M}}^\sharp$ and abstract cognizance $\mathbb{C}^\sharp \in \mathbb{L} \mapsto \wp(\mathcal{D}_{\mathbb{M}}^\sharp)$, the abstract responsibility analysis can determine the responsible entities in P that are potentially responsible for $\mathcal{B}^\sharp$ to the cognizance of $\mathbb{C}^\sharp$.

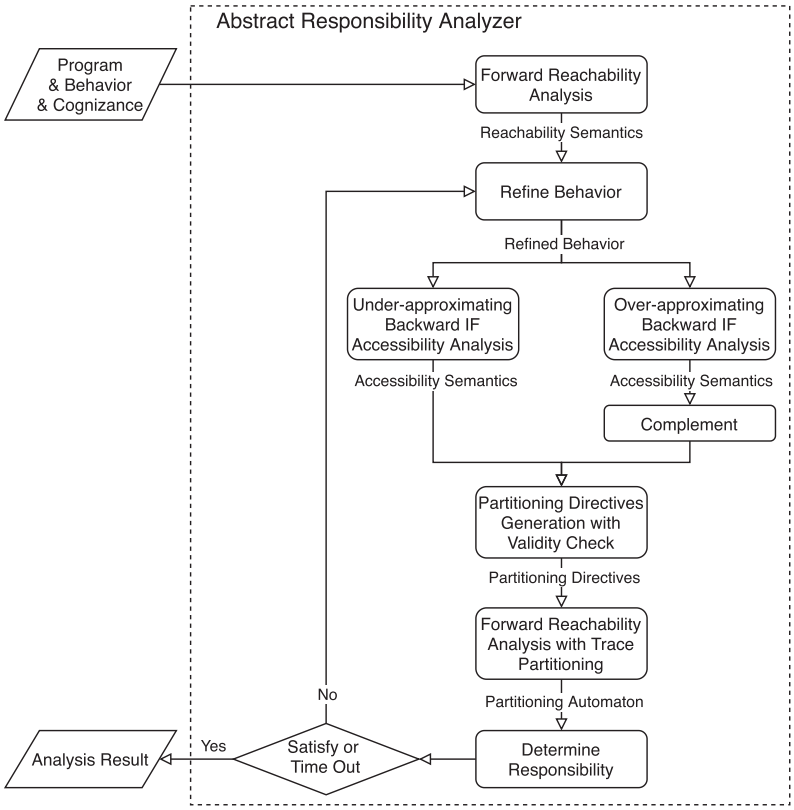

Fig. 18. Trace framework of abstract responsibility analysis.

More precisely, the abstract responsibility analysis starts with a forward reachability analysis $\mathcal{S}^{\sharp}_{\overrightarrow{ps}}[\![\mathrm{P}]\!]$, which produces an over-approximation of the program's reachability semantics. Then, after refining the behavior of interest $\mathcal{B}^{\sharp}$ by the intersection with the computed reachability semantics, we perform in parallel both an under-approximating backward impossible failure accessibility analysis $\check{\mathcal{S}}^{\sharp}_{\overleftarrow{if}}[\![\mathrm{P}]\!](\mathcal{B}^{\sharp})$ and an over-approximating backward impossible failure accessibility analysis $\hat{\mathcal{S}}^{\sharp}_{\overleftarrow{if}}[\![\mathrm{P}]\!](\mathcal{B}^{\sharp})$, and the correspondingly computed accessibility semantics (or its complement) are transformed into partitioning directives of form "$d_p = \mathrm{part}\langle \mathrm{Inv},\ \ell,\ \mathrm{M}^{\sharp}_p\rangle$." Further, using the partitioning directives that are valid with respect to $\mathbb{C}^{\sharp}$, a new round of forward reachability analysis is conducted, which computes a refined reachability semantics and a trace partitioning automaton. In such an automaton, nodes created by partitioning directives from the complement of $\hat{\mathcal{S}}^{\sharp}_{\overleftarrow{if}}[\![\mathrm{P}]\!](\mathcal{B}^{\sharp})$ are marked as left bounds, while nodes created by partitioning directives from $\check{\mathcal{S}}^{\sharp}_{\overleftarrow{if}}[\![\mathrm{P}]\!](\mathcal{B}^{\sharp})$ are marked as right bounds. It follows that, along each path in the automaton, the responsible entities must be located after the left bounds (if any) and before the right bounds (if any). Thus, at this point we can determine responsible entities in the trace partitioning automaton and stop if we are satisfied with the results or the cost exceeds the pre-specified threshold. Otherwise, we start a new round of backward-forward analysis with the behavior $\mathcal{B}^{\sharp}$ that is refined again

by the new reachability semantics, which may improve the precision of responsibility analysis result.

It is not hard to see that most components in this framework of abstract responsibility analysis have been discussed in previous sections. In the rest of this section, we summarize these components and illustrate how they collaborate to determine responsibility in the abstract.

7.2 The Preprocessing Phase

The abstract responsibility analysis starts with a preprocessing phase, in which the user specifies the behavior of interest and the observer's cognizance, and a preliminary forward reachability analysis is conducted to compute the reachability semantics.

The abstract behavior $\mathcal{B}^\sharp$ and the abstract cognizance $\mathbb{C}^\sharp$ have been elaborated in Section 6, and the over-approximating forward reachability analysis $\mathcal{S}^\sharp_{\overrightarrow{ps}}[\![\mathrm{P}]\!](\mathsf{I}^\sharp_{\mathsf{pre}})$ has been formalized in Section 4.2, thus here we reuse them and supplement with some practical tips that could facilitate the coming analysis phases.

For any program P to be analyzed, we insert a dummy initial program point ℓ_0 followed by a dummy action that does not affect the program execution (e.g., skip) in front of P, such that the variable initialization action at the beginning of program execution is explicitly mimicked by this dummy action. Therefore, when the dummy initial action is determined as responsible for a behavior $\mathcal{B}^\sharp$, it means that whether $\mathcal{B}^\sharp$ occurs or not may be decided by the variable initialization.

In addition, for the over-approximating forward reachability analysis $\mathcal{S}^\sharp_{\overrightarrow{ps}}[\![\mathrm{P}]\!](\mathsf{I}^\sharp_{\mathsf{pre}})$, the abstract precondition $\mathsf{I}^\sharp_{\mathsf{pre}} \in \mathbb{L} \mapsto \mathcal{D}^\sharp_{\mathbb{M}}$ is always defined such that the abstract environment element for the dummy initial point is the top (i.e., $\mathsf{I}^\sharp_{\mathsf{pre}}(\ell_0) = \top^\sharp_{\mathbb{M}}$) and it is the bottom for all other program points (i.e., $\mathsf{I}^\sharp_{\mathsf{pre}}(\ell) = \bot^\sharp_{\mathbb{M}}$ for $\ell \neq \ell_0$). Moreover, the precision of this forward reachability analysis can be improved by trace partitioning, which is optional. Although until this step we have not obtained any partitioning directives that are related with memory states and of form "part⟨Inv, ℓ, $\mathrm{M}^\sharp_p$⟩," we can still conduct the trace partitioning by partitioning directives related with the control flow (e.g., part⟨If, ℓ, b⟩ and part⟨While, ℓ, n⟩), which can be derived as in the preprocessing phase of Reference [70].

Example 22 (Access Control, Continued). For the access control program in Figure 4, we insert a dummy initial point ℓ_0 as well as a dummy action before the point ℓ_1, which has no affect on the result of forward reachability analysis in this phase. Suppose the user is interested in the behavior "the access to o fails," then the user can specify the abstract behavior $\mathcal{B}^\sharp \in \mathbb{L} \mapsto \mathcal{D}^\sharp_{\mathbb{M}}$ such that $\mathcal{B}^\sharp(\ell_8) = \mathit{acs} \in [-\infty; 0]$, while $\mathcal{B}^\sharp(\ell) = \top^\sharp_{\mathbb{M}}$ for other program points $\ell \neq \ell_8$. Here, we consider two types of observers: an omniscient observer, whose abstract cognizance function $\mathbb{C}^\sharp_o \in \mathbb{L} \mapsto \wp(\mathcal{D}^\sharp_{\mathbb{M}})$ is specified such that $\mathbb{C}^\sharp_o(\ell) = \{\bot^\sharp_{\mathbb{M}}\}$ for every program point $\ell \in \mathbb{L}$; and an observer that does not know the input of the 1st admin, and the corresponding abstract cognizance function $\mathbb{C}^\sharp \in \mathbb{L} \mapsto \wp(\mathcal{D}^\sharp_{\mathbb{M}})$ is defined such that, if $\ell \in \{\ell_0, \ell_1, \ell_2\}$, then $\mathbb{C}^\sharp(\ell) = \{\bot^\sharp_{\mathbb{M}}\}$, otherwise $\mathbb{C}^\sharp(\ell) = \{\mathit{i1} \in [-1; 2]\}$.

Since there is no conditional or while loop in the access control program, we do the forward reachability analysis without trace partitioning, and the corresponding forward reachability semantics $\mathcal{S}^\sharp_{\overrightarrow{ps}}[\![\mathrm{P}]\!](\mathsf{I}^\sharp_{\mathsf{pre}})$ is listed in Table 11 (which is almost the same as the Table 2). For the sake of clarity and to be consistent with the analysis result from the Interproc analyzer [40], the trivial constraints like $\mathit{acs} \in [-\infty; \infty]$ are omitted in the table.

Table 11. Abstract Forward Reachability Semantics for the Access Control Program

ℓ	$\mathcal{S}^\sharp_{\overrightarrow{ps}}[\![\mathrm{P}]\!](\mathrm{I}^\sharp_{\mathrm{pre}})\ell$
ℓ_0	$\top^\sharp_{\mathbb{M}}$
ℓ_1	$\top^\sharp_{\mathbb{M}}$
ℓ_2	$apv \in [1;1]$
ℓ_3	$apv \in [1;1] \wedge i1 \in [-1;2]$
ℓ_4	$apv \in [-1;1] \wedge i1 \in [-1;2]$
ℓ_5	$apv \in [-1;1] \wedge i1 \in [-1;2] \wedge i2 \in [-1;2]$
ℓ_6	$apv \in [-1;1] \wedge i1 \in [-1;2] \wedge i2 \in [-1;2]$
ℓ_7	$apv \in [-1;1] \wedge i1 \in [-1;2] \wedge i2 \in [-1;2] \wedge typ \in [1;2]$
ℓ_8	$apv \in [-1;1] \wedge i1 \in [-1;2] \wedge i2 \in [-1;2] \wedge typ \in [1;2] \wedge acs \in [-2;2]$

7.3 The Backward Analysis Phase

The objective of this backward analysis phase is to create partitioning directives of the form "part⟨Inv, ℓ, $\mathrm{M}^\sharp_p$⟩" that can either guarantee that the behavior $\mathcal{B}^\sharp$ always hold or guarantee the existence of at least one execution trace that fails behavior $\mathcal{B}^\sharp$.

7.3.1 Behavior Refinement with Reachability Semantics. After completing a forward reachability analysis, the first step is to refine the behavior $\mathcal{B}^\sharp$ of interest by the intersection with the computed reachability semantics $\mathcal{S}^\sharp_{\overrightarrow{ps}}[\![\mathrm{P}]\!](\mathrm{I}^\sharp_{\mathrm{pre}})$.

If $\mathcal{S}^\sharp_{\overrightarrow{ps}}[\![\mathrm{P}]\!](\mathrm{I}^\sharp_{\mathrm{pre}})$ is computed without trace partitioning, then the intersection of $\mathcal{S}^\sharp_{\overrightarrow{ps}}[\![\mathrm{P}]\!](\mathrm{I}^\sharp_{\mathrm{pre}}) \in \mathbb{L} \mapsto \mathcal{D}^\sharp_{\mathbb{M}}$ with $\mathcal{B}^\sharp \in \mathbb{L} \mapsto \mathcal{D}^\sharp_{\mathbb{M}}$ is simply the pointwise meet $\sqcap^\sharp_{\mathbb{M}}$ of abstract environments, and the refined behavior is $\mathcal{S}^\sharp_{\overrightarrow{ps}}[\![\mathrm{P}]\!](\mathrm{I}^\sharp_{\mathrm{pre}}) \dot{\sqcap}^\sharp_{\mathbb{M}} \mathcal{B}^\sharp$. However, if the trace partitioning is involved in the forward reachability analysis, then $\mathcal{S}^\sharp_{\overrightarrow{ps}}[\![\mathrm{P}]\!](\mathrm{I}^\sharp_{\mathrm{pre}}) \in \mathbb{L}_T \mapsto \mathcal{D}^\sharp_{\mathbb{M}} \triangleq \mathbb{L} \times T \mapsto \mathcal{D}^\sharp_{\mathbb{M}}$ (where T is the set of partitioning tokens) has to be transformed into the form $\mathbb{L} \mapsto \mathcal{D}^\sharp_{\mathbb{M}}$ before its intersection with $\mathcal{B}^\sharp$.

There are possibly several ways to do so: (1) A naive method is to apply to $\mathcal{S}^\sharp_{\overrightarrow{ps}}[\![\mathrm{P}]\!](\mathrm{I}^\sharp_{\mathrm{pre}})$ the forget function π_τ, which is defined in the trace partitioning abstract domain (Section 5.1) to remove partitioning tokens from extended program points, such that abstract environments at the same point with different partitioning tokens are joined together. Formally, we construct $\mathrm{I}^\sharp \in \mathbb{L} \mapsto \mathcal{D}^\sharp_{\mathbb{M}}$ such that $\forall \ell \in \mathbb{L}.\ \mathrm{I}^\sharp(\ell) = \sqcup^\sharp_{\mathbb{M}}\{\mathcal{S}^\sharp_{\overrightarrow{ps}}[\![\mathrm{P}]\!](\mathrm{I}^\sharp_{\mathrm{pre}})\langle \ell,\ t\rangle \mid t \in T\}$, and the refined behavior would be $\mathrm{I}^\sharp \dot{\sqcap}^\sharp_{\mathbb{M}} \mathcal{B}^\sharp$. (2) The naive method can be improved if we do the intersection with $\mathcal{B}^\sharp$ before joining the abstract environments together. Formally, the refined behavior is $\mathcal{B}^{\sharp\prime} \in \mathbb{L} \mapsto \mathcal{D}^\sharp_{\mathbb{M}}$ such that $\mathcal{B}^{\sharp\prime}(\ell) = \sqcup^\sharp_{\mathbb{M}}\{\mathcal{S}^\sharp_{\overrightarrow{ps}}[\![\mathrm{P}]\!](\mathrm{I}^\sharp_{\mathrm{pre}})\langle \ell,\ t\rangle \sqcap^\sharp_{\mathbb{M}} \mathcal{B}(\ell) \mid t \in T\}$. (3) Another alternative method is to use multiple behaviors to represent the intersection of $\mathcal{B}^\sharp$ with $\mathcal{S}^\sharp_{\overrightarrow{ps}}[\![\mathrm{P}]\!](\mathrm{I}^\sharp_{\mathrm{pre}})$. More specifically, $\mathcal{S}^\sharp_{\overrightarrow{ps}}[\![\mathrm{P}]\!](\mathrm{I}^\sharp_{\mathrm{pre}})$ can be equivalently viewed as a trace partitioning automaton, thus for each path in this automaton we can do an intersection with $\mathcal{B}^\sharp$ and construct a new behavior if the path is still valid (i.e., no node is attached with $\bot^\sharp_{\mathbb{M}}$). This method is the most precise one for refining the behavior of interest,

but the cost of introducing multiple behaviors to the following backward analysis is prohibitive, hence it is not adopted in this article.

Example 23 (Access Control, Continued). Following Example 22, the trace partitioning is not used in the forward reachability analysis, hence the abstract behavior $\mathcal{B}^\sharp$ can be refined simply by the pointwise meet $\dot{\sqcap}^\sharp_{\mathbb{M}}$ with $\mathcal{S}^\sharp_{\overrightarrow{ps}}[\![\mathrm{P}]\!](\mathrm{I}^\sharp_{\mathrm{pre}})$ from Table 11. For the sake of conciseness, the refined behavior is still called $\mathcal{B}^\sharp$, thus we have: $\mathcal{B}^\sharp(\ell_8) = apv \in [-1;1] \wedge i1 \in [-1;2] \wedge i2 \in [-1;2] \wedge typ \in [1;2] \wedge acs \in [-2;0]$, and $\mathcal{B}^\sharp(\ell) = \mathcal{S}^\sharp_{\overrightarrow{ps}}[\![\mathrm{P}]\!](\mathrm{I}^\sharp_{\mathrm{pre}})\ell$ for program points ℓ other than ℓ_8.

7.3.2 Under-approximating/Over-approximating Backward Impossible Failure Accessibility Analysis. Using the under-approximating backward impossible failure accessibility analysis given in Section 4.3.2, we get $\check{\mathcal{S}}^\sharp_{\overleftarrow{if}}[\![\mathrm{P}]\!](\mathcal{B}^\sharp) \in \mathbb{L} \mapsto \mathcal{D}^\sharp_{\mathbb{M}}$ such that, for every program point ℓ, $\check{\mathcal{S}}^\sharp_{\overleftarrow{if}}[\![\mathrm{P}]\!](\mathcal{B}^\sharp)\ell$ is an under-approximation of the weakest sufficient precondition for $\mathcal{B}^\sharp$. Since an under-approximation of the weakest sufficient precondition is still a sufficient condition, every concrete valid trace that begins from a state $\langle \ell,\ \rho\rangle$ such that $\rho \in \gamma_{\mathbb{M}}(\check{\mathcal{S}}^\sharp_{\overleftarrow{if}}[\![\mathrm{P}]\!](\mathcal{B}^\sharp)\ell)$ must satisfy the behavior $\mathcal{B}^\sharp$.

Similarly, using the over-approximating backward impossible failure accessibility analysis formalized in Section 4.3.3, we get $\hat{\mathcal{S}}^\sharp_{\overleftarrow{if}}[\![\mathrm{P}]\!](\mathrm{I}^{\sharp\prime}_{\mathrm{post}}) \in \mathbb{L} \mapsto \mathcal{D}^\sharp_{\mathbb{M}}$ such that, for every program point ℓ, $\hat{\mathcal{S}}^\sharp_{\overleftarrow{if}}[\![\mathrm{P}]\!](\mathcal{B}^\sharp)\ell$ over-approximates the weakest sufficient precondition for $\mathcal{B}^\sharp$. Since an over-approximation of the weakest sufficient precondition is not necessarily a sufficient condition, $\hat{\mathcal{S}}^\sharp_{\overleftarrow{if}}[\![\mathrm{P}]\!](\mathcal{B}^\sharp)\ell$ does not guarantee that the occurrence of behavior $\mathcal{B}^\sharp$. However, it is guaranteed that, if all the concrete valid traces that begin from a state $\langle \ell,\ \rho\rangle$ have the behavior $\mathcal{B}^\sharp$, then ρ must satisfy the environment property $\hat{\mathcal{S}}^\sharp_{\overleftarrow{if}}[\![\mathrm{P}]\!](\mathcal{B}^\sharp)\ell$ (i.e., $\rho \in \gamma_{\mathbb{M}}(\hat{\mathcal{S}}^\sharp_{\overleftarrow{if}}[\![\mathrm{P}]\!](\mathcal{B}^\sharp)\ell)$). That is to say, from a state $\langle \ell,\ \rho\rangle$ such that ρ does not satisfy $\hat{\mathcal{S}}^\sharp_{\overleftarrow{if}}[\![\mathrm{P}]\!](\mathcal{B}^\sharp)\ell$ (i.e., $\rho \notin \gamma_{\mathbb{M}}(\hat{\mathcal{S}}^\sharp_{\overleftarrow{if}}[\![\mathrm{P}]\!](\mathcal{B}^\sharp)\ell)$), there must exist at least one concrete valid trace that fails the behavior $\mathcal{B}^\sharp$.

To make use of the environments that do not satisfy $\hat{\mathcal{S}}^\sharp_{\overleftarrow{if}}[\![\mathrm{P}]\!](\mathcal{B}^\sharp)\ell$, we compute the complement of $\hat{\mathcal{S}}^\sharp_{\overleftarrow{if}}[\![\mathrm{P}]\!](\mathcal{B}^\sharp)$, or even better, $\mathcal{S}^\sharp_{\overrightarrow{ps}}[\![\mathrm{P}]\!](\mathrm{I}^\sharp_{\mathrm{pre}}) \backslash \hat{\mathcal{S}}^\sharp_{\overleftarrow{if}}[\![\mathrm{P}]\!](\mathcal{B}^\sharp)$ (i.e., $\mathcal{S}^\sharp_{\overrightarrow{ps}}[\![\mathrm{P}]\!](\mathrm{I}^\sharp_{\mathrm{pre}}) \dot{\sqcap}^\sharp_{\mathbb{M}} (\dot{\neg}\hat{\mathcal{S}}^\sharp_{\overleftarrow{if}}[\![\mathrm{P}]\!](\mathcal{B}^\sharp))$), such that invalid environments are excluded. Yet, most abstract environment domains do not directly support the complement operation, including the classic numerical domains (such as the interval, octagon, and polyhedron domain). For example, the complement of a polyhedron is a disjunction of affine inequalities. Nevertheless, similar to the disjunctive completion, we can define the complement of $\hat{\mathcal{S}}^\sharp_{\overleftarrow{if}}[\![\mathrm{P}]\!](\mathcal{B}^\sharp) \in \mathbb{L} \mapsto \mathcal{D}^\sharp_{\mathbb{M}}$ as $\dot{\neg}\hat{\mathcal{S}}^\sharp_{\overleftarrow{if}}[\![\mathrm{P}]\!](\mathcal{B}^\sharp) \in \mathbb{L} \mapsto \wp(\mathcal{D}^\sharp_{\mathbb{M}})$, such that each abstract environment element in $\dot{\neg}\hat{\mathcal{S}}^\sharp_{\overleftarrow{if}}[\![\mathrm{P}]\!](\mathcal{B}^\sharp)\ell$ represents an affine inequality in the disjunction at the point ℓ.

It is worth mentioning that the number of affine inequalities in the complement of some abstract environment element from $\mathcal{D}^\sharp_{\mathbb{M}}$ may be large, especially for polyhedra. However, it is safe to remove part of these affine inequalities and keep only the heuristically selected ones, without any harm to the soundness of abstract responsibility analysis.

Table 12. Under-approximating Backward IF Accessibility Semantics for $\mathcal{B}^\sharp$

ℓ	$\check{\mathcal{S}}^\sharp_{\overleftarrow{if}}[\![\mathrm{P}]\!](\mathcal{B}^\sharp)\ell$
ℓ_0	$\bot^\sharp_{\mathbb{M}}$
ℓ_1	$\bot^\sharp_{\mathbb{M}}$
ℓ_2	$\bot^\sharp_{\mathbb{M}}$
ℓ_3	$\bot^\sharp_{\mathbb{M}}$
ℓ_4	$\bot^\sharp_{\mathbb{M}}$
ℓ_5	$apv \in [\mathbf{1};\mathbf{1}] \wedge i1 \in [-1;2] \wedge i2 \in [\mathbf{-1};\mathbf{0}]$
ℓ_6	$apv \in [\mathbf{-1};\mathbf{0}] \wedge i1 \in [-1;2] \wedge i2 \in [-1;2]$
ℓ_7	$apv \in [\mathbf{-1};\mathbf{0}] \wedge i1 \in [-1;2] \wedge i2 \in [-1;2] \wedge typ \in [1;2]$
ℓ_8	$apv \in [-1;1] \wedge i1 \in [-1;2] \wedge i2 \in [-1;2] \wedge typ \in [1;2] \wedge acs \in [\mathbf{-2};\mathbf{0}]$

Table 13. Over-approximating Backward IF Accessibility Semantics for $\mathcal{B}^\sharp$ with Disjunctive Completion

ℓ	$\hat{\mathcal{S}}^\sharp_{\overleftarrow{if}}[\![\mathrm{P}]\!](\mathcal{B}^\sharp)\ell$
ℓ_0	$\bot^\sharp_{\mathbb{M}}$
ℓ_1	$\bot^\sharp_{\mathbb{M}}$
ℓ_2	$\bot^\sharp_{\mathbb{M}}$
ℓ_3	$apv \in [1;1] \wedge i1 \in [\mathbf{-1};\mathbf{0}]$
ℓ_4	$apv \in [\mathbf{-1};\mathbf{0}] \wedge i1 \in [-1;2]$
ℓ_5	$\{apv \in [\mathbf{-1};\mathbf{0}] \wedge i1 \in [-1;2] \wedge i2 \in [-1;2],\ apv \in [\mathbf{1};\mathbf{1}] \wedge i1 \in [-1;2] \wedge i2 \in [\mathbf{-1};\mathbf{0}]\}$
ℓ_6	$apv \in [\mathbf{-1};\mathbf{0}] \wedge i1 \in [-1;2] \wedge i2 \in [-1;2]$
ℓ_7	$apv \in [\mathbf{-1};\mathbf{0}] \wedge i1 \in [-1;2] \wedge i2 \in [-1;2] \wedge typ \in [1;2]$
ℓ_8	$apv \in [-1;1] \wedge i1 \in [-1;2] \wedge i2 \in [-1;2] \wedge typ \in [1;2] \wedge acs \in [\mathbf{-2};\mathbf{0}]$

Example 24 (Access Control, Continued). Following Example 23, we conduct an under-approximating backward impossible failure accessibility analysis on $\mathcal{B}^\sharp$, and the corresponding result $\check{\mathcal{S}}^\sharp_{\overleftarrow{if}}[\![\mathrm{P}]\!](\mathcal{B}^\sharp)$ is listed in Table 12. Similarly, the result of the over-approximating backward impossible failure accessibility analysis on $\mathcal{B}^\sharp$ is listed in Table 13. Notice that we have adopted the disjunctive completion in $\check{\mathcal{S}}^\sharp_{\overleftarrow{if}}[\![\mathrm{P}]\!](\mathcal{B}^\sharp)\ell_5$ to gain the precision, otherwise it would be $apv \in [-1;1] \wedge i1 \in [-1;2] \wedge i2 \in [-1;2]$ instead, which is equal to $\mathcal{S}^\sharp_{\overrightarrow{ps}}[\![\mathrm{P}]\!](\mathrm{I}^\sharp_{\mathrm{pre}})\ell$.

Furthermore, the complement of $\hat{\mathcal{S}}^\sharp_{\overleftarrow{if}}[\![\mathrm{P}]\!](\mathcal{B}^\sharp)$ is listed in Table 14. Notice that, instead of simply using the direct complement of $\hat{\mathcal{S}}^\sharp_{\overleftarrow{if}}[\![\mathrm{P}]\!](\mathcal{B}^\sharp)$, here we adopt $\mathcal{S}^\sharp_{\overrightarrow{ps}}[\![\mathrm{P}]\!](\mathrm{I}^\sharp_{\mathrm{pre}}) \dot{\setminus} \hat{\mathcal{S}}^\sharp_{\overleftarrow{if}}[\![\mathrm{P}]\!](\mathcal{B}^\sharp)$, or, say, $\mathcal{S}^\sharp_{\overrightarrow{ps}}[\![\mathrm{P}]\!](\mathrm{I}^\sharp_{\mathrm{pre}}) \dot{\sqcap}^\sharp_{\mathbb{M}} (\dot{\neg} \hat{\mathcal{S}}^\sharp_{\overleftarrow{if}}[\![\mathrm{P}]\!](\mathcal{B}^\sharp))$, such that invalid environments would not be included. For example, at point ℓ_2, the direct complement of $\hat{\mathcal{S}}^\sharp_{\overleftarrow{if}}[\![\mathrm{P}]\!](\mathcal{B}^\sharp)\ell_2 = \bot^\sharp_{\mathbb{M}}$ is $\top^\sharp_{\mathbb{M}}$. After the meet $\sqcap^\sharp_{\mathbb{M}}$ with the reachability semantics $\mathcal{S}^\sharp_{\overrightarrow{ps}}[\![\mathrm{P}]\!](\mathrm{I}^\sharp_{\mathrm{pre}})\ell_2$, we can get the more precise $apv \in [1;1]$. Similarly, at point ℓ_3, the direct complement of $\hat{\mathcal{S}}^\sharp_{\overleftarrow{if}}[\![\mathrm{P}]\!](\mathcal{B}^\sharp)\ell_3 = apv \in [1;1] \wedge i1 \in [-1;0]$ is the

Table 14. The Complement of Over-approximating Backward IF Accessibility Semantics for $\mathcal{B}^\sharp$

ℓ	$\mathcal{S}^\sharp_{\overrightarrow{ps}}[\![\mathrm{P}]\!](\mathrm{I}^\sharp_{\mathrm{pre}})\ell \backslash \hat{\mathcal{S}}^\sharp_{\overleftarrow{if}}[\![\mathrm{P}]\!](\mathcal{B}^\sharp)\ell$
ℓ_0	$\{\top^\sharp_{\mathbb{M}}\}$
ℓ_1	$\{\top^\sharp_{\mathbb{M}}\}$
ℓ_2	$\{apv \in [1;1]\}$
ℓ_3	$\{apv \in [1;1] \wedge i1 \in [\mathbf{1};\mathbf{2}]\}$
ℓ_4	$\{apv \in [\mathbf{1};\mathbf{1}] \wedge i1 \in [-1;2]\}$
ℓ_5	$\{apv \in [\mathbf{1};\mathbf{1}] \wedge i1 \in [-1;2] \wedge i2 \in [\mathbf{1};\mathbf{2}]\}$
ℓ_6	$\{apv \in [\mathbf{1};\mathbf{1}] \wedge i1 \in [-1;2] \wedge i2 \in [-1;2]\}$
ℓ_7	$\{apv \in [\mathbf{1};\mathbf{1}] \wedge i1 \in [-1;2] \wedge i2 \in [-1;2] \wedge typ \in [1;2]\}$
ℓ_8	$\{apv \in [-1;1] \wedge i1 \in [-1;2] \wedge i2 \in [-1;2] \wedge typ \in [1;2] \wedge acs \in [\mathbf{1};\mathbf{2}]\}$

disjunction of $\{apv \in [-\infty;0], apv \in [2;\infty], i1 \in [-\infty;-2], i1 \in [1;\infty]\}$, most of which are invalid (or, say, unreachable in the concrete). After the meet with the reachability semantics $\mathcal{S}^\sharp_{\overrightarrow{ps}}[\![\mathrm{P}]\!](\mathrm{I}^\sharp_{\mathrm{pre}})\ell_3 = apv \in [1;1] \wedge i1 \in [-1;2]$, it is refined to $\{apv \in [1;1] \wedge i1 \in [1;2]\}$, which is much more precise than direct complement.

7.3.3 Partitioning Directives Generation with Validity Check. Using the under-approximating backward impossible failure accessibility semantics $\check{\mathcal{S}}^\sharp_{\overleftarrow{if}}[\![\mathrm{P}]\!](\mathcal{B}^\sharp)$ and the complement of over-approximating backward impossible failure accessibility semantics $\mathcal{S}^\sharp_{\overrightarrow{ps}}[\![\mathrm{P}]\!](\mathrm{I}^\sharp_{\mathrm{pre}})\backslash\hat{\mathcal{S}}^\sharp_{\overleftarrow{if}}[\![\mathrm{P}]\!](\mathcal{B}^\sharp)$, this step aims at constructing a partition function $\mathbb{P}^\sharp \in \mathbb{L} \mapsto \wp(\mathcal{D}^\sharp_{\mathbb{M}})$, such that $\forall \ell \in \mathbb{L}.\ \forall \mathrm{M}^\sharp_p \in \mathbb{P}^\sharp(\ell)$. $\mathrm{part}\langle \mathrm{Inv},\ \ell,\ \mathrm{M}^\sharp_p\rangle$ is a partitioning directive that is valid with respect to the specified cognizance function $\mathbb{C}^\sharp$ and will be used in the next round of forward reachability analysis. Sometimes, a partitioning directive $d_p = \mathrm{part}\langle \mathrm{Inv},\ \ell,\ \mathrm{M}^\sharp_p\rangle$ is called as $\mathrm{M}^\sharp_p$ for short, when the program point ℓ is known from the context.

More specifically, here we design four types of partitioning directives that are based on the environments, and accordingly the partition function $\mathbb{P}^\sharp$ can be split into four parts:

(1) Right-bound partitioning directives. For any point ℓ, $\check{\mathcal{S}}^\sharp_{\overleftarrow{if}}[\![\mathrm{P}]\!](\mathcal{B}^\sharp)\ell$ is a right-bound partitioning directive, if it is not $\bot^\sharp_{\mathbb{M}}$ and is valid with respect to all cognizance directives assigned to ℓ. Formally, we define the right-bound partition function $\mathbb{P}^\sharp_R$:

$$\mathbb{P}^\sharp_R \in \mathbb{L} \mapsto \wp\left(\mathcal{D}^\sharp_{\mathbb{M}}\right) \qquad \text{right-bound partition function}$$

$$\mathbb{P}^\sharp_R(\ell) \triangleq \left\{ \mathrm{M}^\sharp_p \mid \mathrm{M}^\sharp_p = \check{\mathcal{S}}^\sharp_{\overleftarrow{if}}[\![\mathrm{P}]\!](\mathcal{B}^\sharp)\ell \wedge \mathrm{M}^\sharp_p \neq \bot^\sharp_{\mathbb{M}} \wedge \forall d_c \in \mathbb{C}^\sharp(\ell).\ \mathrm{isValid}_d\left(d_c, \mathrm{M}^\sharp_p\right) \right\}.$$

By the definition of $\check{\mathcal{S}}^\sharp_{\overleftarrow{if}}[\![\mathrm{P}]\!](\mathcal{B}^\sharp)$, the partitions generated by right-bound partitioning directives during the next forward reachability analysis would guarantee the occurrence of $\mathcal{B}^\sharp$.

In addition, the time cost of forward reachability analysis with trace partitioning greatly depends on the number of created partitions, while typically $\check{\mathcal{S}}^\sharp_{\overleftarrow{if}}[\![\mathrm{P}]\!](\mathcal{B}^\sharp)$ contains redundant elements in

consecutive program points, thus adopting every element in $\check{\mathcal{S}}^{\sharp}_{\overleftarrow{if}}[\![\mathrm{P}]\!](\mathcal{B}^{\sharp})$ as partitioning directives may bring unnecessary burden to the forward reachability analysis without benefits in improving the precision. Therefore, in practice, we can keep the partitioning directives only for the program points of importance (e.g., the points immediately after external inputs) and discard the rest of them.

(2) Left-bound partitioning directives. Similar to the generation of right-bound partitioning directives from $\check{\mathcal{S}}^{\sharp}_{\overleftarrow{if}}[\![\mathrm{P}]\!](\mathcal{B}^{\sharp})$, the left-bound partitioning directives are derived from the complement of $\hat{\mathcal{S}}^{\sharp}_{\overleftarrow{if}}[\![\mathrm{P}]\!](\mathcal{B}^{\sharp})$ (i.e., $\mathcal{S}^{\sharp}_{\overrightarrow{ps}}[\![\mathrm{P}]\!](\mathrm{I}^{\sharp}_{\mathrm{pre}})\backslash\hat{\mathcal{S}}^{\sharp}_{\overleftarrow{if}}[\![\mathrm{P}]\!](\mathcal{B}^{\sharp})$). Specifically, for any point ℓ, every element in $\mathcal{S}^{\sharp}_{\overrightarrow{ps}}[\![\mathrm{P}]\!](\mathrm{I}^{\sharp}_{\mathrm{pre}})\ell\backslash\hat{\mathcal{S}}^{\sharp}_{\overleftarrow{if}}[\![\mathrm{P}]\!](\mathcal{B}^{\sharp})\ell$ is a left-bound partitioning directive if it is not $\bot^{\sharp}_{\mathbb{M}}$ and is valid with respect to all cognizance directives assigned to ℓ. Formally, the left-bound partition function $\mathbb{P}^{\sharp}_{L}$ is:

$$\mathbb{P}^{\sharp}_{L} \in \mathbb{L} \mapsto \wp\left(\mathcal{D}^{\sharp}_{\mathbb{M}}\right) \qquad \text{left-bound partition function}$$

$$\mathbb{P}^{\sharp}_{L}(\ell) \triangleq \left\{ \mathrm{M}^{\sharp}_{p} \mid \mathrm{M}^{\sharp}_{p} \in \mathcal{S}^{\sharp}_{\overrightarrow{ps}}[\![\mathrm{P}]\!](\mathrm{I}^{\sharp}_{\mathrm{pre}})\ell\backslash\hat{\mathcal{S}}^{\sharp}_{\overleftarrow{if}}[\![\mathrm{P}]\!](\mathcal{B}^{\sharp})\ell \wedge \mathrm{M}^{\sharp}_{p} \neq \bot^{\sharp}_{\mathbb{M}} \wedge \forall d_c \in \mathbb{C}^{\sharp}(\ell).\ \mathrm{isValid}_{d}\left(d_c, \mathrm{M}^{\sharp}_{p}\right) \right\}.$$

By the definition of $\hat{\mathcal{S}}^{\sharp}_{\overleftarrow{if}}[\![\mathrm{P}]\!](\mathcal{B}^{\sharp})$, we know that from every partition generated by the left-bound partitioning directive, there must exist at least one concrete valid trace that fails the behavior $\mathcal{B}^{\sharp}$. Moreover, similar to the right-bound partitioning directives, it is of practical use to keep the left-bound partitioning directives only for selected program points and discard the rest.

(3) Dual-right-bound partitioning directives. Intuitively, the left-bound partitioning directives can determine the points from which there is still possibility to fail $\mathcal{B}^{\sharp}$, and the right-bound partitioning directives are used to determine the points at which $\mathcal{B}^{\sharp}$ is guaranteed and the responsibility analysis could stop. Besides these two types of partitioning directive, the responsibility analysis would benefit from another type of partitioning directive, which are used to determine the points at which the behavior $\mathcal{B}^{\sharp}$ is guaranteed to fail and the responsibility analysis can also stop. Such partitioning directives are called dual-right-bound partitioning directives, which marks the finishing point for responsibility analysis on the traces failing $\mathcal{B}^{\sharp}$; without such partitioning directives, the responsibility analysis may last much longer than necessary.

Theoretically, the dual-right-bound partitioning directives can be derived from backward impossible failure accessibility analyses for the complements of $\mathcal{B}$, but the cost of doing so would be prohibitive and the analysis results overlaps with the left-bound partitioning directives. In practice, to mark the finishing point of responsibility analysis on the traces failing $\mathcal{B}^{\sharp}$, we can simply use the complements of $\mathcal{B}^{\sharp}$, or more precisely, $\mathcal{S}^{\sharp}_{\overrightarrow{ps}}[\![\mathrm{P}]\!](\mathrm{I}^{\sharp}_{\mathrm{pre}})\backslash\mathcal{B}^{\sharp}$. Specifically, for every point ℓ of interest where the original behavior (before the refinement) $\mathcal{B}^{\sharp}(\ell) \neq \top^{\sharp}_{\mathbb{M}}$, we compute the complement of $\mathcal{B}^{\sharp}(\ell)$, which is represented by the disjunctive completion (e.g., a disjunction of affine inequalities for polyhedral), do the meet with $\mathcal{S}^{\sharp}_{\overrightarrow{ps}}[\![\mathrm{P}]\!](\mathrm{I}^{\sharp}_{\mathrm{pre}})$ for every element in the disjunction, and collect the valid ones in the dual-right-bound partition function $\mathbb{P}^{\sharp}_{\tilde{R}}$. Usually, there are not many dual-right-bound partitioning directives, since the original behavior $\mathcal{B}^{\sharp}$ is specified $\top^{\sharp}_{\mathbb{M}}$ at most program points.

$$\mathbb{P}^\sharp_{\tilde{R}} \quad \in \quad \mathbb{L} \mapsto \wp\left(\mathcal{D}^\sharp_{\mathbb{M}}\right) \qquad \text{dual-right-bound partition function}$$

$$\mathbb{P}^\sharp_{\tilde{R}}(\ell) \quad \triangleq \quad \left\{ \mathrm{M}^\sharp_p \mid \mathrm{M}^\sharp_p \in \mathcal{S}^\sharp_{\overrightarrow{ps}}[\![\mathrm{P}]\!](\mathrm{I}^\sharp_{\mathrm{pre}})\ell \backslash \mathcal{B}^\sharp(\ell) \wedge \mathrm{M}^\sharp_p \neq \bot^\sharp_{\mathbb{M}} \wedge \forall d_c \in \mathbb{C}^\sharp(\ell).\ \mathrm{isValid}_d\left(d_c, \mathrm{M}^\sharp_p\right) \right\}.$$

(4) No-bound partitioning directives. To make sure that the trace partitioning automaton (or, say, the extended transition system in Reference [70]) constructed by the partitioning directives introduced above is a covering of the original transition system (i.e., every transition in the original transition system is simulated by at least one transition in the trace partitioning automaton), we introduce some complementary partitioning directives to ensure every reachable state is covered by at least one partition. Such partitioning directives are called no-bound partitioning directives, and we define a no-bound partition function $\mathbb{P}^\sharp_o \in \mathbb{L} \mapsto \wp(\mathcal{D}^\sharp_{\mathbb{M}})$.

Formally, it is required that: $\forall \ell \in \mathbb{L}.\ \cup \{\gamma_{\mathbb{M}}(\mathrm{M}^\sharp_p) \mid \mathrm{M}^\sharp_p \in \mathbb{P}^\sharp_R(\ell) \cup \mathbb{P}^\sharp_L(\ell) \cup \mathbb{P}^\sharp_{\tilde{R}}(\ell) \cup \mathbb{P}^\sharp_o(\ell)\} \supseteq \gamma_{\mathbb{M}}(\mathcal{S}^\sharp_{\overrightarrow{ps}}[\![\mathrm{P}]\!](\mathrm{I}^\sharp_{\mathrm{pre}})\ell)$, where $\mathcal{S}^\sharp_{\overrightarrow{ps}}[\![\mathrm{P}]\!](\mathrm{I}^\sharp_{\mathrm{pre}})\ell$ over-approximates the set of all reachable concrete environments at point ℓ. Ideally, the no-bound partitioning directives $\mathbb{P}^\sharp_o(\ell)$ can be computed by the subtraction of $\mathbb{P}^\sharp_R(\ell) \cup \mathbb{P}^\sharp_L(\ell) \cup \mathbb{P}^\sharp_{\tilde{R}}(\ell)$ from $\mathcal{S}^\sharp_{\overrightarrow{ps}}[\![\mathrm{P}]\!](\mathrm{I}^\sharp_{\mathrm{pre}})\ell$. However, in some cases it may be difficult to do such a subtraction operation. If this happens, then it is always safe to define $\mathbb{P}^\sharp_o(\ell) = \mathcal{S}^\sharp_{\overrightarrow{ps}}[\![\mathrm{P}]\!](\mathrm{I}^\sharp_{\mathrm{pre}})\ell$, or even, $\mathbb{P}^\sharp_o(\ell) = \top^\sharp_{\mathbb{M}}$, which are guaranteed to be valid with respect to any cognizance function.

Combining the above four types of partitioning directives together, we can get a partition function $\mathbb{P}^\sharp \in \mathbb{L} \mapsto \wp(\mathcal{D}^\sharp_{\mathbb{M}})$ such that $\mathbb{P}^\sharp(\ell) \triangleq \mathbb{P}^\sharp_R(\ell) \cup \mathbb{P}^\sharp_L(\ell) \cup \mathbb{P}^\sharp_{\tilde{R}}(\ell) \cup \mathbb{P}^\sharp_o(\ell)$. For every program point ℓ, every partitioning directive in $\mathbb{P}^\sharp(\ell)$ is valid with respect to every cognizance directive d_c in $\mathbb{C}^\sharp(\ell)$, thus by the definition of $\mathrm{isValid}_{\mathbb{P}}$, the partition function $\mathbb{P}^\sharp$ is valid with respect to the cognizance function $\mathbb{C}^\sharp$. Besides, it is assumed that $\mathbb{P}^\sharp(\ell_0) = \emptyset$ for the dummy initial point ℓ_0, such that the correspondingly constructed trace partitioning automaton has only one initial node.

Example 25 (Access Control, Continued). Using the backward analysis result $\check{\mathcal{S}}^\sharp_{\overleftarrow{if}}[\![\mathrm{P}]\!](\mathcal{B}^\sharp)$ and $\mathcal{S}^\sharp_{\overrightarrow{ps}}[\![\mathrm{P}]\!](\mathrm{I}^\sharp_{\mathrm{pre}}) \backslash \hat{\mathcal{S}}^\sharp_{\overleftarrow{if}}[\![\mathrm{P}]\!](\mathcal{B}^\sharp)$ from Example 24, here we generate partitioning directives for two different cognizance functions that are specified in Example 22.

(1) Consider the omniscient cognizance function $\mathbb{C}^\sharp_o$ such that $\mathbb{C}^\sharp_o(\ell) = \{\bot^\sharp_{\mathbb{M}}\}$ for every point $\ell \in \mathbb{L}$. In this case, every partitioning directive is trivially valid with respect to $\mathbb{C}^\sharp_o$, and the corresponding partition function $\mathbb{P}^\sharp$ is displayed in Table 15. As mentioned before, the partition function $\mathbb{P}^\sharp$ may keep the partitioning directives only for the selected program points of importance, and in this example such program points include: ℓ_1 that is immediately after the variable initialization action (i.e., the dummy initial action); ℓ_3, ℓ_5, and ℓ_7 that are immediately after external inputs; and ℓ_8 that is specified with the behavior $\mathcal{B}^\sharp$ of interest. Meanwhile, the partitioning directives at other points (ℓ_2, ℓ_4, and ℓ_6) are optional and would not affect the final result of abstract responsibility analysis, thus are omitted here.

Taking the point ℓ_5 as an example, the forward reachability semantics $\mathcal{S}^\sharp_{\overrightarrow{ps}}[\![\mathrm{P}]\!](\mathrm{I}^\sharp_{\mathrm{pre}})\ell_5 = apv \in [-1;1] \wedge i1 \in [-1;2] \wedge i2 \in [-1;2]$ is partitioned into three parts: the right-bound partitioning directives $\mathbb{P}^\sharp_R(\ell_5) = \{apv \in [1;1] \wedge i1 \in [-1;2] \wedge i2 \in [-1;0]\}$ that guarantee "the access to o fails"; the left-bound partitioning directives $\mathbb{P}^\sharp_L(\ell_5) = \{apv \in [1;1] \wedge i1 \in [-1;2] \wedge i2 \in [1;2]\}$, which ensures there exists at least one valid concrete trace such that "the access to o succeeds"; the no-bound

Table 15. The Partition Function for the Omnisicent Cognizance

ℓ	$\mathbb{P}^\sharp_R(\ell)$	$\mathbb{P}^\sharp_L(\ell)$	$\mathbb{P}^\sharp_{\tilde{R}}(\ell)$	$\mathbb{P}^\sharp_o(\ell)$
ℓ_0	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$
ℓ_1	$\emptyset$	$\{\top^\sharp_{\mathbb{M}}\}$	$\emptyset$	$\emptyset$
ℓ_2	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$
ℓ_3	$\emptyset$	$\{apv \in [1;1] \wedge i1 \in [\mathbf{1};\mathbf{2}]\}$	$\emptyset$	$\{apv \in [1;1] \wedge i1 \in [\mathbf{-1};\mathbf{0}]\}$
ℓ_4	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$
ℓ_5	$\{apv \in [\mathbf{1};\mathbf{1}] \wedge i1 \in [-1;2] \wedge i2 \in [\mathbf{-1};\mathbf{0}]\}$	$\{apv \in [\mathbf{1};\mathbf{1}] \wedge i1 \in [-1;2] \wedge i2 \in [\mathbf{1};\mathbf{2}]\}$	$\emptyset$	$\{apv \in [\mathbf{-1};\mathbf{0}] \wedge i1 \in [-1;2] \wedge i2 \in [\mathbf{-1};\mathbf{2}]\}$
ℓ_6	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$
ℓ_7	$apv \in [\mathbf{-1};\mathbf{0}] \wedge i1 \in [-1;2] \wedge i2 \in [-1;2] \wedge typ \in [1;2]$	$\{apv \in [\mathbf{1};\mathbf{1}] \wedge i1 \in [-1;2] \wedge i2 \in [-1;2] \wedge typ \in [1;2]\}$	$\emptyset$	$\emptyset$
ℓ_8	$\{apv \in [-1;1] \wedge i1 \in [-1;2] \wedge i2 \in [-1;2] \wedge typ \in [1;2] \wedge acs \in [\mathbf{-2};\mathbf{0}]\}$	$\{apv \in [-1;1] \wedge i1 \in [-1;2] \wedge i2 \in [-1;2] \wedge typ \in [1;2] \wedge acs \in [\mathbf{1};\mathbf{2}]\}$	$\mathbb{P}^\sharp_L(\ell_8)$	$\emptyset$

partitioning directives $\mathbb{P}^\sharp_o(\ell_5) = \{apv \in [-1;0] \wedge i1 \in [-1;2] \wedge i2 \in [-1;2]\}$ are complementary to the two other types of partitioning directives such that every reachable environment at point ℓ_5 is covered. Although $\mathbb{P}^\sharp_o(\ell_5)$ actually guarantee "the access to o fails" in this very example, we cannot take advantage of this information, since typically the no-bound partitioning directives cannot guarantee anything.

(2) Consider a non-omniscient cognizance function $\mathbb{C}^\sharp$ such that, if $\ell \in \{\ell_0, \ell_1, \ell_2\}$, then $\mathbb{C}^\sharp(\ell) = \{\bot^\sharp_{\mathbb{M}}\}$, otherwise $\mathbb{C}^\sharp(\ell) = \{i1 \in [-1;2]\}$. In this case, every partitioning directive from Table 15 needs to be checked with respect to the cognizance. Since the abstract environment domain is the interval domain, checking the validity of partitioning directives is quite easy by the condition (32), and we can find that only the partitioning directives at point ℓ_3 are invalid. Take $\mathrm{M}^\sharp_p = apv \in [1;1] \wedge i1 \in [1;2] \in \mathbb{P}^\sharp_L(\ell_3)$ as an example, the only cognizance directive at ℓ_3 is $d_c = i1 \in [-1;2] \in \mathbb{C}^\sharp(\ell_3)$, thus $\mathrm{M}^\sharp_p|_{\mathrm{vars}(\mathrm{M}^\sharp_p) \cap \mathrm{vars}(d_c)} = i1 \in [1;2]$, and it is obvious the condition (32) does not hold. Similarly, the partitioning directive in $\mathbb{P}^\sharp_o(\ell_3)$ is found invalid. Therefore, after removing the invalid partitioning directives at ℓ_3, we get the partition function as in Table 16, which is valid with respect to $\mathbb{C}^\sharp$.

7.4 The Forward Analysis Phase

The objective of this forward analysis phase is to construct a trace partitioning automaton with the partitioning directives from the last phase, mark left bounds and right bounds of responsibility in the automaton, and determine responsible entities.

(1) Trace Partitioning Automaton Generation. Using the partitioning directives from the backward analysis phase (i.e., $\{\mathrm{part}\langle \mathrm{Inv}, \ell, \mathrm{M}^\sharp_p\rangle \mid \ell \in \mathbb{L} \wedge \mathrm{M}^\sharp_p \in \mathbb{P}^\sharp(\ell)\}$) and optionally the partitioning directives based on the control flow (e.g., $\mathrm{part}\langle \mathrm{If}, \ell, b\rangle$), we perform an over-approximating forward reachability analysis with trace partitioning (Section 5), compute the refined forward reachability semantics, and construct a trace partitioning automaton. Specially, the nodes generated by left-bound partitioning directives are marked as "*left-bound nodes*" in the automaton, the nodes

Table 16. The Partition Function for the Non-omniscient Cognizance

ℓ	$\mathbb{P}^\sharp_R(\ell)$	$\mathbb{P}^\sharp_L(\ell)$	$\mathbb{P}^\sharp_{\tilde{R}}(\ell)$	$\mathbb{P}^\sharp_o(\ell)$
ℓ_0	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$
ℓ_1	$\emptyset$	$\{\top^\sharp_{\mathbb{M}}\}$	$\emptyset$	$\emptyset$
ℓ_2	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$
ℓ_3	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$
ℓ_4	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$
ℓ_5	$\{apv \in [\mathbf{1};\mathbf{1}] \wedge i1 \in [-1;2] \wedge i2 \in [-\mathbf{1};\mathbf{0}]\}$	$\{apv \in [\mathbf{1};\mathbf{1}] \wedge i1 \in [-1;2] \wedge i2 \in [\mathbf{1};\mathbf{2}]\}$	$\emptyset$	$\{apv \in [-\mathbf{1};\mathbf{0}] \wedge i1 \in [-1;2] \wedge i2 \in [-\mathbf{1};\mathbf{2}]\}$
ℓ_6	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$
ℓ_7	$apv \in [-\mathbf{1};\mathbf{0}] \wedge i1 \in [-1;2] \wedge i2 \in [-1;2] \wedge typ \in [1;2]$	$\{apv \in [\mathbf{1};\mathbf{1}] \wedge i1 \in [-1;2] \wedge i2 \in [-1;2] \wedge typ \in [1;2]\}$	$\emptyset$	$\emptyset$
ℓ_8	$\{apv \in [-1;1] \wedge i1 \in [-1;2] \wedge i2 \in [-1;2] \wedge typ \in [1;2] \wedge acs \in [-\mathbf{2};\mathbf{0}]\}$	$\{apv \in [-1;1] \wedge i1 \in [-1;2] \wedge i2 \in [-1;2] \wedge typ \in [1;2] \wedge acs \in [\mathbf{1};\mathbf{2}]\}$	$\mathbb{P}^\sharp_L(\ell_8)$	$\emptyset$

generated by right-bound partitioning directives are marked as "*right-bound nodes*," and the nodes generated by dual-right-bound partitioning directives are marked as "*dual-right-bound nodes*."

Furthermore, after the forward reachability analysis with trace partitioning completes, we can improve the constructed automaton by propagating the right-bounds or dual-right-bounds: For any node in the automaton that is not marked as any bound, if all its successors are marked as right-bound nodes (respectively, dual-right-bound nodes), then we mark this node as a right-bound node (respectively, a dual-right-bound node) as well.

(2) Determining Responsible Entities. Now, we can determine the responsibility in the generated trace partitioning automaton. The intuition is: Every path in the automaton represents a set of concrete traces; if a path contains a dual-right-bound node, then the path does not have the behavior $\mathcal{B}^\sharp$, hence there is no responsible entity along this path; otherwise, the responsible entities are the edges (i.e., actions), which are located after the left-bound nodes (if any) and before the right-bound nodes (if any) along the path. That is to say, for any path that does not contain a dual-right-bound node, all the actions between the rightmost left-bound node (if any) and the left-most right-bound node (if any) are potentially responsible for the behavior $\mathcal{B}^\sharp$. Specially, if there is neither a left-bound node nor a right-bound node along a certain path, then the analysis is not precise enough and every action along that path would be determined as potentially responsible.

Since only the actions with free choices can be possibly responsible for a behavior, we can further restrict the potentially responsible entities to the actions such as external inputs, random number generation, and variable initialization (which is mimicked as the dummy initial action).

In addition, we do not only find potentially responsible entities, but also get some hints on when these entities are actually responsible, and this is the so called "responsible under the condition." Suppose an edge $\langle \ell,\ t,\ \mathrm{M}^\sharp\rangle \to \langle \ell',\ d_p :: t,\ \mathrm{M}^{\sharp\prime}\rangle$ in the automaton is found potentially responsible, it means that the action a from ℓ to ℓ' (which can be retrieved from the program source code) is potentially responsible under the condition that the partitioning token t holds at point ℓ and the action a satisfies the partitioning directive d_p. For example, for the access control program in Figure 4, the edge $\langle \ell_4,\ t,\ \mathrm{M}^\sharp\rangle \to \langle \ell_5,\ d_p :: t,\ \mathrm{M}^{\sharp\prime}\rangle$ is determined responsible, where $t = apv \in$

$[1;1] \wedge i1 \in [1;2]$ and $d_p = apv \in [1;1] \wedge i1 \in [-1;2] \wedge i2 \in [-1;0]$. Instead of claiming that the action $i2 := [-1;2]$ is responsible for "the access to o fails" in all executions, we state that $i2 := [-1;2]$ is responsible under the condition that: The partitioning token $apv \in [1;1] \wedge i1 \in [1;2]$ holds at ℓ_4, and the action $i2 := [-1;2]$ satisfies the new partitioning directive $apv \in [1;1] \wedge i1 \in [-1;2] \wedge i2 \in [-1;0]$. That is to say, the input from 2nd admin is responsible for the behavior "the access to o fails" if the input from 1st admin is positive and the input from 2nd admin is negative or zero, which is much more informative than simply claiming the input from 2nd admin is responsible.

(3) Termination or a New Round of Analysis. Up until this step, we have already successfully inferred some information about the responsible entities. If such an analysis result is satisfactory or the time and costs exceeds the prespecified threshold, then we could terminate the analyzer and return the found responsible entities to the user. Otherwise, if the precision of forward reachability semantics $\mathcal{S}^{\sharp}_{\overrightarrow{ps}}[\![\mathrm{P}]\!](\mathrm{I}^{\sharp}_{\mathrm{pre}})$ is improved in the last forward analysis phase, then we could start a new round of backward accessibility analysis (Section 7.3) followed by the forward reachability analysis (Section 7.4) to seek for more precise responsibility analysis results.

Intuitively, in the new round of analysis, using the refined behavior of interest (possibly with the disjunctive completion), the backward impossible failure accessibility analysis is expected to be more precise, which creates more partitioning directives to construct a refined automaton and further improves the responsibility analysis result. The extreme case is that we create as many partitioning directives as possible and construct the most precise trace partitioning automaton, such that every path in the automaton represents a single concrete valid trace. From such a trace partitioning automaton, we can get exactly the same analysis result as the concrete responsibility analysis (Section 3), yet the time cost is in general prohibitive.

Example 26 (Access Control, Continued). Following Example 25, we conduct a forward reachability analysis with trace partitioning, construct the trace partitioning automaton, and determine responsible entities. Since the partition function varies for different cognizance functions, the corresponding constructed trace partitioning automata are different.

(1) First, consider the omniscient cognizance function $\mathbb{C}^{\sharp}_{o}$. In this case, we adopt the partitioning directives from the partition function in the Table 15, and the correspondingly constructed trace partitioning automaton is in Figure 19, in which various types of nodes are represented by different circles.

Since there is at most one element in $\mathbb{P}^{\sharp}_{L}(\ell)$ for every program point ℓ, we simply use the notation $d_{L(\ell)}$ for short to refer the partitioning directive part$\langle$Inv, ℓ, $\mathrm{M}^{\sharp}\rangle$, where $\mathrm{M}^{\sharp}$ is the only element in $\mathbb{P}^{\sharp}_{L}(\ell)$. For instance, $d_{L(\ell_3)}$ refers to the partitioning directive part$\langle$Inv, ℓ, $apv \in [1;1] \wedge i1 \in [\mathbf{1};\mathbf{2}]\rangle$, where $apv \in [1;1] \wedge i1 \in [\mathbf{1};\mathbf{2}] \in \mathbb{P}^{\sharp}_{L}(\ell_3)$. Similarly, we use the notations $d_{R(\ell)}$, $d_{\tilde{R}(\ell)}$ and $d_{o(\ell)}$ to refer the partitioning directives from $\mathbb{P}^{\sharp}_{R}(\ell)$, $\mathbb{P}^{\sharp}_{\tilde{R}}(\ell)$, and $\mathbb{P}^{\sharp}_{o}(\ell)$. Besides, for the sake of conciseness, instead of explicitly drawing partitioning tokens inside the nodes of the automaton, we label some edges with a partitioning directive d such that every node after the edge has the partitioning directive d pushed into its stack of directives (i.e., the partitioning token). For instance, for the node at point ℓ_3 with double dashed circles in upper path of the automaton, its partitioning token is "$d_{L(\ell_3)} :: d_{L(\ell_1)}$."

Furthermore, the automaton in Figure 19 can be refined by removing the invalid node whose associated abstract environment element is $\bot^{\sharp}_{\mathbb{M}}$ (i.e., it is unreachable) and propagating right-bound

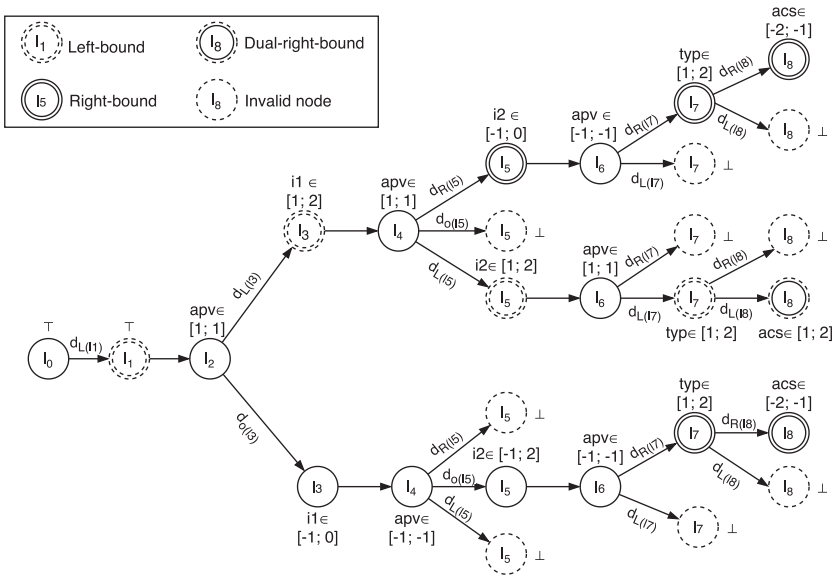

Fig. 19. Trace partitioning automaton for the omniscient cognizance.

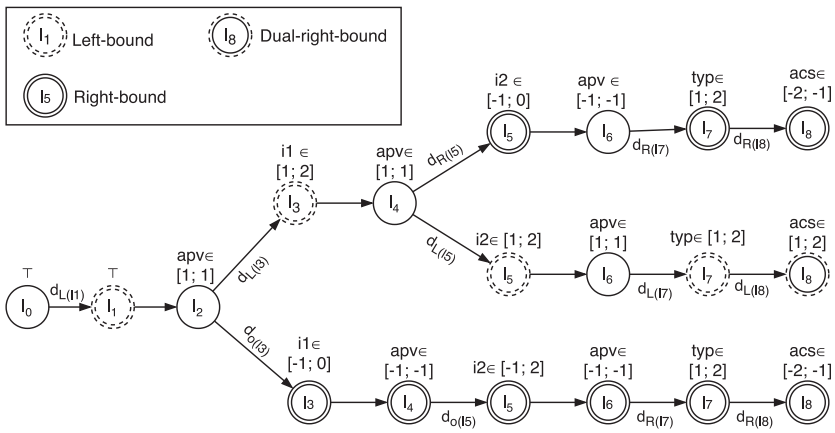

Fig. 20. The refined trace partitioning automaton for the omniscient cognizance.

nodes, and we get a simpler trace partitioning automaton as in Figure 20. For example, the node at point ℓ_5 created by $d_{o(\ell_5)}$ in the upper path is invalid and can be removed; the node at point ℓ_6 in the lower path has only one valid successor that is marked as a right-bound node, thus we mark the node at point ℓ_6 also as a right-bound node, as well as its predecessors.

From the above automaton, we can clearly see that there are three maximal paths from ℓ_0 to ℓ_8 in the automaton, which over-approximate all the concrete valid traces of the program.

For the upper path, the rightmost left-bound node is at ℓ_3 and the leftmost right-bound node is at ℓ_5, thus the responsible entities must be located between ℓ_3 and ℓ_5. Since the action "$apv := (i1 \leq 0)\ ?\ -1 : apv$" has no free choice, only the action "$i2 := [-1; 2]$" is determined responsible under the condition: $apv \in [1; 1] \wedge i1 \in [1; 2]$ hold at point ℓ_4, and the action "$i2 := [-1; 2]$" satisfies

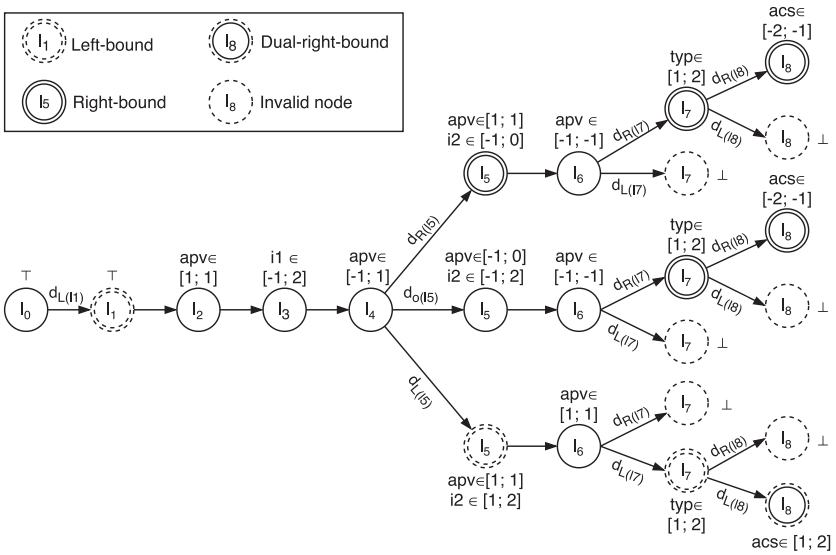

Fig. 21. The trace partitioning automaton for a non-omniscient cognizance.

$apv \in [1;1] \wedge i1 \in [-1;2] \wedge i2 \in [1;2]$. It indicates that the input from 2nd admin is responsible if the input from 1st admin is positive and the input from 2nd admin is negative or zero.

For the path in the middle, the node at ℓ_8 is marked as a dual-right-bound node, which means that every concrete trace represented by this path does not have the behavior "the access to o fails." Thus, there is no responsible entity along this path.

For the lower path, the rightmost left-bound node is at ℓ_1 and the leftmost right-bound node is at ℓ_3. Since the action "$apv := 1$" from ℓ_1 to ℓ_2 has no free choice, only the action $i1 := [-1;2]$ from ℓ_2 to ℓ_3 is determined responsible, under the condition that $apv \in [1;1] \wedge i1 \in [-1;0]$ holds at point ℓ_3. That is to say, the input from 1st admin is responsible if it is -1 or 0.

To sum up, for the omniscient cognizance in the access control program example, the abstract responsibility analysis finds that the input from 1st admin or 2nd admin is potentially responsible for "the access to o fails" under certain conditions, while other actions with free choice (e.g., the variable initialization and the input from system settings) are not responsible. This analysis result is almost as precise as the concrete responsibility analysis, thus there is no need to conduct a new round of analysis and we can terminate the analysis here.

(2) Second, consider the trace partitioning automaton constructed for the non-omniscient cognizance function $\mathbb{C}^\sharp$ such that the observer does not know the input of 1st admin (i.e., the observer cannot distinguish the value of $i1$ in the interval $[-1;2]$). In this case, we adopt the partitioning directives from the partition function defined in Table 16, and the correspondingly constructed trace partitioning automaton is in Figure 21, in which we represent various types of nodes by different circles as in Figure 19.

Compared with the trace partitioning automaton for the omniscient cognizance, we do not have the partitioning directives at point ℓ_3, while the partitioning directives at other points are still valid and preserved. After removing the invalid nodes and propagating right-bound nodes, the refined trace partitioning automaton is in Figure 22.

Similar to the automaton in Figure 20, there are three maximal paths in the refined automaton for the non-omniscient cognizance, which over-approximate the concrete valid traces.

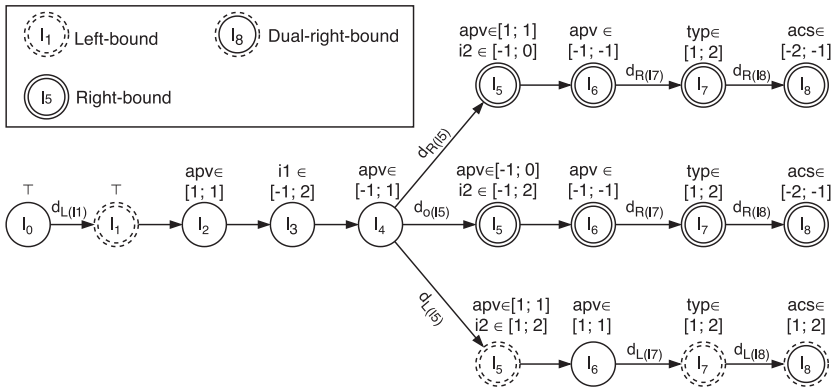

Fig. 22. The refined trace partitioning automaton for a non-omniscient cognizance.

For the lower path, the node at ℓ_8 is marked as a dual-right-bound node, which means that the behavior of interest does not hold, thus there is no responsible entity along the path.

In contrast, along both the upper path and the middle path, the rightmost left-bound node is at point ℓ_1 and the leftmost right-bound node is at point ℓ_5, thus the responsible entities must be located between ℓ_1 and ℓ_5. After filtering out the actions without free choices, we would determine both "$i1 := [-1; 2]$" and "$i2 := [-1; 2]$" potentially responsible for the behavior. More precisely, "$i1 := [-1; 2]$" is responsible under no condition, while "$i2 := [-1; 2]$" is responsible under the condition that the partitioning directive $apv \in [1; 1] \wedge i1 \in [-1; 2] \wedge i2 \in [-1; 0]$ or $apv \in [-1; 0] \wedge i1 \in [-1; 2] \wedge i2 \in [-1; 2]$ holds at point ℓ_5.

To sum up, for the non-omniscient cognizance such that the observer does not know the input from 1st admin, the abstract responsibility analysis find both the input from 1st admin and the input from 2nd admin are potentially responsible for the behavior "the access to o fails" in every execution where the behavior occurs. Compared with the concrete responsibility analysis that determines only the input from 2nd admin responsible, this abstract analysis result is less precise, but it is still sound, since every entity that is responsible in the concrete is also found responsible in the abstract.

7.5 The Soundness of Abstract Responsibility Analysis

In this section, we prove that the abstract responsibility analysis introduced in Section 7.1 is sound with respect to the concrete responsibility analysis defined in Section 3.2.4.

Theorem 3. *Every entity that is responsible in the concrete must be found responsible in the abstract responsibility analysis.*

Proof. Given a program P along with the user specified behavior of interest $\mathcal{B}^\sharp \in \mathbb{L} \mapsto \mathcal{D}^\sharp_{\mathbb{M}}$ and cognizance function $\mathbb{C}^\sharp \in \mathbb{L} \mapsto \wp(\mathcal{D}^\sharp_{\mathbb{M}})$, the corresponding concrete behavior of interest $\mathcal{B}$ and lattice of concrete behaviors $\mathcal{L}^{\mathsf{Max}}$ are formalized in Section 6.1, as well as the concrete cognizance function $\mathbb{C}$ in Section 6.2. Suppose that the behavior $\mathcal{B}$ holds in a valid concrete trace σ of P, and a concrete transition $\tau = \langle \ell,\ \rho\rangle \xrightarrow{a} \langle \ell',\ \rho'\rangle$ (in which a may be omitted and can be retrieved from the source code) in σ is found responsible for $\mathcal{B}$ by the Definition 5 of concrete responsibility analysis (i.e., the trace σ is splitted into $\sigma = \sigma_{\mathrm{H}}\tau\sigma_{\mathrm{F}}$ such that $\emptyset \subsetneq \mathbb{O}([\![\mathrm{P}]\!]^{\mathsf{Max}}, \mathcal{L}^{\mathsf{Max}}, \mathbb{C}, \sigma_{\mathrm{H}}\tau) \subseteq \mathcal{B} \wedge \mathbb{O}([\![\mathrm{P}]\!]^{\mathsf{Max}}, \mathcal{L}^{\mathsf{Max}}, \mathbb{C}, \sigma_{\mathrm{H}}) \not\subseteq \mathcal{B}$), then we would like to prove that the action a must be found responsible in the abstract responsibility analysis.

Since the trace partitioning automaton constructed in the abstract responsibility analysis is a covering of the concrete trace semantics of P, every valid concrete trace is simulated by at least one path in the automaton. Let $\sigma^\sharp$ be a path in the trace partitioning automaton that simulates the concrete trace σ and $\tau^\sharp = \langle \ell,\ t,\ \mathrm{M}^\sharp\rangle \xrightarrow{a} \langle \ell',\ t',\ \mathrm{M}^{\sharp\prime}\rangle \in (\mathbb{L}_T \times \mathcal{D}^\sharp_{\mathbb{M}}) \times (\mathbb{L}_T \times \mathcal{D}^\sharp_{\mathbb{M}})$ be the edge on the path $\sigma^\sharp$ that represents the transition τ. Thus, we need to prove that $\tau^\sharp$ must be found responsible in the abstract responsibility analysis.

To start with, we prove that there is no dual-right-bound node along the abstract path $\sigma^\sharp$ by contradiction. Assume there is a dual-right-bound node at point $\ell_{\tilde{R}}$ on $\sigma^\sharp$, which is created by a dual-right-bound partitioning directive part$\langle$Inv, $\ell_{\tilde{R}}, \mathrm{M}^\sharp_{\tilde{R}}\rangle$. By the definition of dual-right-bound partition function, we know that $\mathrm{M}^\sharp_{\tilde{R}}$ guarantees the complement of $\mathcal{B}^\sharp$ (i.e., $\mathrm{M}^\sharp_{\tilde{R}} \in \mathcal{S}^\sharp_{\overrightarrow{ps}}[\![\mathrm{P}]\!](\mathrm{I}^\sharp_{\mathrm{pre}})\ell_{\tilde{R}} \backslash \mathcal{B}^\sharp(\ell_{\tilde{R}})$), thus all the concrete traces represented by $\sigma^\sharp$ must fail the behavior $\mathcal{B}$ at point $\ell_{\tilde{R}}$. This contradicts with our assumption that σ is simulated by $\sigma^\sharp$ and the behavior $\mathcal{B}$ holds in σ. Thus, along the path $\sigma^\sharp$, there is no dual-right-bound node, and all the edges between the rightmost left-bound-node (if any) and the leftmost right-bound-node are determined potentially responsible by the abstract responsibility analysis. Specially, if there is no left-bound-node or right-bound-node along $\sigma^\sharp$, every edge is determined potentially responsible, which obviously includes $\tau^\sharp$.

Furthermore, we prove that if there is a left-bound node along $\sigma^\sharp$, then $\tau^\sharp$ must be located after that node. Assume that there is a left-bound node along $\sigma^\sharp$, which is created by a left-bound partitioning directive part$\langle$Inv, ℓ_L, $\mathrm{M}^\sharp_L\rangle$, and a concrete state $s_L = \langle \ell_L,\ \rho_L\rangle$ on σ is represented by the left-bound node. By the definition of the left-bound partition function, the abstract environment $\mathrm{M}^\sharp_L$ is from the complement of the over-approximating backward impossible failure semantics for the behavior $\mathcal{B}^\sharp$ (i.e., $\mathrm{M}^\sharp_L \in \mathcal{S}^\sharp_{\overrightarrow{ps}}[\![\mathrm{P}]\!](\mathrm{I}^\sharp_{\mathrm{pre}})\ell_L \backslash \hat{\mathcal{S}}^\sharp_{\overleftarrow{if}}[\![\mathrm{P}]\!](\mathcal{B}^\sharp)\ell_L$). That is to say, from every concrete state that is represented by the left-bound node, there must exist at least one valid concrete trace that fails the behavior $\mathcal{B}$. If we split σ into σ' and σ'' such that σ' ends with s_L while σ'' begins with s_L, then it is obvious that σ' cannot guarantee the occurrence of $\mathcal{B}$, thus $\mathbb{I}([\![\mathrm{P}]\!]^{\mathrm{Max}}, \mathcal{L}^{\mathrm{Max}}, \sigma') \not\subseteq \mathcal{B}$. Since $\mathbb{I}([\![\mathrm{P}]\!]^{\mathrm{Max}}, \mathcal{L}^{\mathrm{Max}}, \sigma') \subseteq \mathbb{O}([\![\mathrm{P}]\!]^{\mathrm{Max}}, \mathcal{L}^{\mathrm{Max}}, \mathbb{C}, \sigma')$, we have $\mathbb{O}([\![\mathrm{P}]\!]^{\mathrm{Max}}, \mathcal{L}^{\mathrm{Max}}, \mathbb{C}, \sigma') \not\subseteq \mathcal{B}$. As $\mathbb{O}([\![\mathrm{P}]\!]^{\mathrm{Max}}, \mathcal{L}^{\mathrm{Max}}, \mathbb{C}, \sigma_{\mathrm{H}}\tau) \subseteq \mathcal{B}$ and the observation function $\mathbb{O}$ is decreasing (Lemma 3), we find that $\sigma_{\mathrm{H}}\tau$ must be greater (longer) than σ'. Therefore, the responsible transition τ must be located after the state s_L, and accordingly the edge $\tau^\sharp$ must be located after the left-bound node.

Last, we prove that if there is a right-bound node along $\sigma^\sharp$, then $\tau^\sharp$ must be located before that node. Assume that there is a right-bound node along $\sigma^\sharp$, which is created by a right-bound partitioning directive part$\langle$Inv, $\ell_R, \mathrm{M}^\sharp_R\rangle$, and a concrete state $s_R = \langle \ell_R, \rho_R\rangle$ on σ is represented by the right-bound node. By the definition of the right-bound partition function, the abstract environment $\mathrm{M}^\sharp_R$ is from the under-approximating backward impossible failure semantics for the behavior $\mathcal{B}^\sharp$ (i.e., $\mathrm{M}^\sharp_R = \check{\mathcal{S}}^\sharp_{\overleftarrow{if}}[\![\mathrm{P}]\!](\mathcal{B}^\sharp)\ell_R$). That is to say, every concrete trace starting from the states represented by the right-bound node is guaranteed to have the behavior $\mathcal{B}$. If we split σ into σ' and σ'' such that σ' ends with s_R while σ'' begins with s_R, then it is easy to know $\mathcal{B}$ holds in σ' (since $\mathcal{B}$ holds in the whole trace σ), and every trace with the prefix σ' is guaranteed to have the behavior $\mathcal{B}$. Hence, we get $\mathbb{I}([\![\mathrm{P}]\!]^{\mathrm{Max}}, \mathcal{L}^{\mathrm{Max}}, \sigma') \subseteq \mathcal{B}$. Now we consider the traces that are equivalent to σ' according to the cognizance $\mathbb{C}^\sharp$. For any trace σ'_e such that $\sigma'_e \overset{\mathbb{C}^\sharp}{\sim} \sigma'$, the behavior $\mathcal{B}$ must hold during the execution of σ'_e (since $\mathbb{O}([\![\mathrm{P}]\!]^{\mathrm{Max}}, \mathcal{L}^{\mathrm{Max}}, \mathbb{C}, \sigma_{\mathrm{H}}\tau) \subseteq \mathcal{B}$). By Theorem 2, σ'_e must be represented by the same path as σ' in the automaton, thus the last state in σ'_e is also represented by the same right-bound node

in the automaton. Thus, every trace with the prefix σ'_e is guaranteed to have the behavior $\mathcal{B}$, which implies that $\mathbb{I}([\![\mathrm{P}]\!]^{\mathsf{Max}}, \mathcal{L}^{\mathsf{Max}}, \sigma'_e) \subseteq \mathcal{B}$. By the definition of $\mathbb{O}$, we get $\mathbb{O}([\![\mathrm{P}]\!]^{\mathsf{Max}}, \mathcal{L}^{\mathsf{Max}}, \mathbb{C}, \sigma') \subseteq \mathcal{B}$. Since the observation function $\mathbb{O}$ is decreasing (Lemma 3) and $\mathbb{O}([\![\mathrm{P}]\!]^{\mathsf{Max}}, \mathcal{L}^{\mathsf{Max}}, \mathbb{C}, \sigma_{\mathrm{H}}) \not\subseteq \mathcal{B}$, it is easy to see that σ' is strictly greater (longer) than σ_{H}. Therefore, the responsible transition τ must be located before the state s_R, and accordingly the edge $\tau^\sharp$ must be located before the right-bound node.

To sum up, we have proved that for every concrete trace σ with a responsible entity that is a transition $\tau = \langle \ell, \rho\rangle \xrightarrow{a} \langle \ell', \rho'\rangle$, there exists an abstract path $\sigma^\sharp$ with an edge $\tau^\sharp = \langle \ell, t, \mathrm{M}^\sharp\rangle \xrightarrow{a} \langle \ell', t', \mathrm{M}^{\sharp\prime}\rangle$ in the corresponding trace partitioning automaton, and the edge $\tau^\sharp$ must be located after all the left-bound nodes (if any) and before all the right-bound nodes (if any) on the path $\sigma^\sharp$. Thus, by the abstract responsibility analysis designed in Section 7.1, the edge $\tau^\sharp$ must be determined responsible for $\mathcal{B}^\sharp$. □

8 RELATED WORK

Definition of Causality and Responsibility. Hume [38] is the first one to specify causation by counterfactual dependence [54]. The best-known counterfactual theory of causation is proposed by Lewis [51], which defines causation as a transitive closure of counterfactual dependencies. Halpern and Pearl [34, 35, 63] define actual causality based on SEM and extend counterfactual dependency to allow "contingent dependency." Chockler and Halpern [13] define responsibility to have a quantitative measure of the relevance between causes and effects and define blame to consider the epistemic state of an agent. Their application of actual causality, responsibility, and blame is mainly on artificial intelligence.

Our definition of responsibility also adopts the idea of counterfactual dependence in the sense that, suppose an event σ_{R} is said to be responsible for behavior $\mathcal{B}$ in the trace $\sigma_{\mathrm{H}}\sigma_{\mathrm{R}}$, there must exist another event σ'_{R} such that, if σ_{R} is replaced by σ'_{R}, then $\mathcal{B}$ is not guaranteed (by Lemma 1).

A "naive" definition of *causality* [51, 52] based on counterfactual dependency could exclude non-decisive factors (e.g., the wind in this example) from the analysis result. This definition proposed by Lewis adopts an alternative world semantics and determines causality relations according to a criterion: An event e is a cause of the occurrence of another event e' if and only if, were e not to occur, e' would not happen. The testing of this condition hinges upon the availability of alternative worlds. For instance, in the conjunctive scenario of this forest fire example, we can infer that the forest would not be burnt down in an alternative world where the arsonist A does not drop a lit match, thus the arsonist A is causal for the forest fire; yet, in the alternative world where there is no wind, the forest would still be burnt down, hence the wind is not a cause of the forest fire. However, the counterfactual causality may be too strict in some circumstances such that no cause could be found. Taking the disjunctive scenario of forest fire as an example, in the alternative world where one arsonist A (respectively, B) does not drop a lit match, the forest would still have been burnt down due to the other arsonist B (respectively, A), hence neither of these two arsonists would be determined as the cause of forest fire. Thus, it may be inappropriate to directly adopt the idea of counterfactual dependency in the responsibility analysis.

Error Cause Localization. Classic program analysis techniques, e.g., dependency analysis [1, 12, 73] and program slicing [2, 46, 74, 75], are useful in detecting the code that may be relevant to errors, but fail to localize the cause of error.

In recent years, there are many papers [8, 32, 33, 42, 43, 65–67] on fault localization for counterexample traces, and most of them compare multiple traces produced by a model checker and build a heuristic metric to localize the point from which error traces separate from correct traces. Other related papers include error diagnosis by abductive/backward inference [27], tracking down bugs

by dynamic invariant detection [36]. Actual causality is applied to explain counterexamples from model checker [10] and estimate the coverage of specification [14]. Besides, there are researches on analyzing causes of specific security issues; e.g., King et al. [45] employ a blame dependency graph to explain the source of information flow violation and generate a program slice as the error report.

Compared to the above techniques, this article succeeds to formally define the cause or responsibility, and the proposed responsibility analysis, which does not require a counterexample from the model checker, is sound, scalable, and generic to cope with various problems.

9 CONCLUSION

This article formally defines responsibility as an abstraction of trace semantics. Typically, the responsibility analysis consists of four steps: collect the trace semantics, build a lattice of system behaviors of interest, create an observation function for each observer, and apply the responsibility abstraction on analyzed traces. Compared to current dependency and causality analysis methods, the responsibility analysis is demonstrated to be more generic and precise in a few examples. In addition, a sound framework of abstract responsibility analysis is proposed, which is based on trace partitioning automata constructed by the iteration of over-approximating forward reachability analysis with trace partitioning and under-approximating/over-approximating backward impossible failure accessibility analysis. It is guaranteed that actions that are not found responsible in the abstract analysis are definitely not responsible in the concrete.

We hope this article has successfully demonstrated that the responsibility analysis constitutes a worthy avenue of research. In the future, there are a number of directions that deserve further exploration.

Analysis of Probabilistic Programs. The definition of responsibility proposed in this article can be extended to probabilistic programming languages such that the degree of responsibility of each responsible entity can be quantified, which is similar to the degree of blame designed to quantify actual causality [13]. More precisely, instead of identifying a single responsible entity for each specific trace as in (5), we can collect all the potentially responsible entities for the whole system and assign each of them with a probability of being responsible for the behavior of interest.

Generalization of Abstract Analysis. The framework of abstract responsibility analysis can be applied to new abstract domains other than the classic numeric domains discussed in this article, such that we can analyze the responsibility of more behaviors (which cannot be expressed by intervals, octagons, or polyhedra). The main challenges are expected to come from designing a sound under-approximating backward impossible failure accessibility analysis for the new abstract domain. In addition, we suggest specifying the abstract cognizance function by abstract relational invariants [3, 4] that can directly express relational properties about two executions of the program, such that we do not have the restrictions that two equivalent traces must be of the same length and have the same control flow.

Alternative Definitions of Responsibility. In the philosophy literature, there is a protracted controversy concerning the meaning of responsibility. Just like the law varies from one nation to another, there cannot exist a perfect universal rule of defining responsibility [28] that deals well with all scenarios. In our current definition (5), whether a transition τ_{R} (or, say, the corresponding action a_{R}) is responsible or not in the trace $\sigma_{\mathrm{H}}\tau_{\mathrm{R}}\sigma_{\mathrm{F}}$ solely depends on its history σ_{H}, while its future σ_{F} has no impact on deciding the responsibility. For instance, in the forest fire example, whether an arsonist A is responsible or not solely depends on if there is another arsonist that already dropped a lit match before A or not. This definition of responsibility is quite intuitive and works in many scenarios, but not necessarily all scenarios. In some scenarios, the future part σ_{F} may also need to be taken into account for determining responsibility. We wish to design a lattice of responsibility

definitions, each of which adopts a distinct rule of defining responsibility, and for every specific scenario there is at least one definition from the lattice that can handle it well.

APPENDIX

A APPENDED PROOFS

A.1 Proof of Galois Isomorphism (1)

$$\langle \wp([\![\mathrm{P}]\!]^{\mathrm{Max}}),\ \subseteq\rangle \underset{\alpha_{\mathrm{Pred}}([\![\mathrm{P}]\!]^{\mathrm{Max}})}{\overset{\gamma_{\mathrm{Pred}}([\![\mathrm{P}]\!]^{\mathrm{Max}})}{\leftleftarrows}} \langle \bar{\alpha}_{\mathrm{Pred}}\{[\![\mathrm{P}]\!]^{\mathrm{Max}}\}(\wp([\![\mathrm{P}]\!]^{\mathrm{Max}})),\ \subseteq\rangle$$

Proof. First, we prove that $\alpha_{\mathrm{Pred}}([\![\mathrm{P}]\!]^{\mathrm{Max}})$ and $\gamma_{\mathrm{Pred}}([\![\mathrm{P}]\!]^{\mathrm{Max}})$ are increasing.

– $\mathcal{X} \subseteq \mathcal{X}'$

$\Rightarrow \forall \sigma' \in [\![\mathrm{P}]\!]^{\mathrm{Max}}.\ (\sigma' \in \mathcal{X}) \Rightarrow (\sigma' \in \mathcal{X}')$ ⟅def. $\subseteq$⟆

$\Rightarrow \forall \sigma \in \mathbb{S}^{*\infty}, \sigma' \in [\![\mathrm{P}]\!]^{\mathrm{Max}}.\ (\neg(\sigma \preceq \sigma') \vee (\sigma' \in \mathcal{X})) \Rightarrow (\neg(\sigma \preceq \sigma') \vee (\sigma' \in \mathcal{X}'))$ ⟅def. $\vee$⟆

$\Rightarrow \{\sigma \in \mathbb{S}^{*\infty} \mid \forall \sigma' \in [\![\mathrm{P}]\!]^{\mathrm{Max}}.\ \neg(\sigma \preceq \sigma') \vee (\sigma' \in \mathcal{X})\} \subseteq$
$\{\sigma \in \mathbb{S}^{*\infty} \mid \forall \sigma' \in [\![\mathrm{P}]\!]^{\mathrm{Max}}.\ \neg(\sigma \preceq \sigma') \vee (\sigma' \in \mathcal{X}')\}$ ⟅def. $\subseteq$⟆

$\Rightarrow \{\sigma \in \mathbb{S}^{*\infty} \mid \forall \sigma' \in [\![\mathrm{P}]\!]^{\mathrm{Max}}.\ \sigma \preceq \sigma' \Rightarrow \sigma' \in \mathcal{X}\} \subseteq$
$\{\sigma \in \mathbb{S}^{*\infty} \mid \forall \sigma' \in [\![\mathrm{P}]\!]^{\mathrm{Max}}.\ \sigma \preceq \sigma' \Rightarrow \sigma' \in \mathcal{X}'\}$ ⟅def. $\Rightarrow$⟆

$\Rightarrow (\mathrm{Pref}(\mathcal{X}) \cap \{\sigma \mid \forall \sigma' \in [\![\mathrm{P}]\!]^{\mathrm{Max}}.\ \sigma \preceq \sigma' \Rightarrow \sigma' \in \mathcal{X}\}) \subseteq$
$(\mathrm{Pref}(\mathcal{X}') \cap \{\sigma \mid \forall \sigma' \in [\![\mathrm{P}]\!]^{\mathrm{Max}}.\ \sigma \preceq \sigma' \Rightarrow \sigma' \in \mathcal{X}'\})$ ⟅def. $\cap$ and Pref is increasing⟆

$\Rightarrow \{\sigma \in \mathrm{Pref}(\mathcal{X}) \mid \forall \sigma' \in [\![\mathrm{P}]\!]^{\mathrm{Max}}.\ \sigma \preceq \sigma' \Rightarrow \sigma' \in \mathcal{X}\} \subseteq$
$\{\sigma \in \mathrm{Pref}(\mathcal{X}') \mid \forall \sigma' \in [\![\mathrm{P}]\!]^{\mathrm{Max}}.\ \sigma \preceq \sigma' \Rightarrow \sigma' \in \mathcal{X}'\}$ ⟅def. $\cap$⟆

$\Rightarrow \alpha_{\mathrm{Pred}}([\![\mathrm{P}]\!]^{\mathrm{Max}})\mathcal{X} \subseteq \alpha_{\mathrm{Pred}}([\![\mathrm{P}]\!]^{\mathrm{Max}})\mathcal{X}'$ ⟅def. $\alpha_{\mathrm{Pred}}([\![\mathrm{P}]\!]^{\mathrm{Max}})$⟆

– $\mathcal{Y} \subseteq \mathcal{Y}'$

$\Rightarrow (\mathcal{Y} \cap [\![\mathrm{P}]\!]^{\mathrm{Max}}) \subseteq (\mathcal{Y}' \cap [\![\mathrm{P}]\!]^{\mathrm{Max}})$ ⟅def. $\cap$⟆

$\Rightarrow \gamma_{\mathrm{Pred}}([\![\mathrm{P}]\!]^{\mathrm{Max}})\mathcal{Y} \subseteq \gamma_{\mathrm{Pred}}([\![\mathrm{P}]\!]^{\mathrm{Max}})\mathcal{Y}'$ ⟅def. $\gamma_{\mathrm{Pred}}([\![\mathrm{P}]\!]^{\mathrm{Max}})$⟆

Then, we prove that $\gamma_{\mathrm{Pred}}([\![\mathrm{P}]\!]^{\mathrm{Max}}) \circ \alpha_{\mathrm{Pred}}([\![\mathrm{P}]\!]^{\mathrm{Max}})$ and $\alpha_{\mathrm{Pred}}([\![\mathrm{P}]\!]^{\mathrm{Max}}) \circ \gamma_{\mathrm{Pred}}([\![\mathrm{P}]\!]^{\mathrm{Max}})$ are identity functions.

– $\gamma_{\mathrm{Pred}}([\![\mathrm{P}]\!]^{\mathrm{Max}}) \circ \alpha_{\mathrm{Pred}}([\![\mathrm{P}]\!]^{\mathrm{Max}})\mathcal{X}$

$= \gamma_{\mathrm{Pred}}([\![\mathrm{P}]\!]^{\mathrm{Max}})(\{\sigma \in \mathrm{Pref}(\mathcal{X}) \mid \forall \sigma' \in [\![\mathrm{P}]\!]^{\mathrm{Max}}.\ \sigma \preceq \sigma' \Rightarrow \sigma' \in \mathcal{X}\})$ ⟅def. α_{Pred}⟆

$= \gamma_{\mathrm{Pred}}([\![\mathrm{P}]\!]^{\mathrm{Max}})(\mathcal{X} \cup \{\sigma \in \mathrm{Pref}(\mathcal{X}) \backslash [\![\mathrm{P}]\!]^{\mathrm{Max}} \mid \forall \sigma' \in [\![\mathrm{P}]\!]^{\mathrm{Max}}.\ \sigma \preceq \sigma' \Rightarrow \sigma' \in \mathcal{X}\})$
⟅$\mathcal{X} = \mathrm{Pref}(\mathcal{X}) \cap [\![\mathrm{P}]\!]^{\mathrm{Max}}$, since $\mathcal{X} \in \wp([\![\mathrm{P}]\!]^{\mathrm{Max}})$⟆

$= [\![\mathrm{P}]\!]^{\mathrm{Max}} \cap (\mathcal{X} \cup \{\sigma \in \mathrm{Pref}(\mathcal{X}) \backslash [\![\mathrm{P}]\!]^{\mathrm{Max}} \mid \forall \sigma' \in [\![\mathrm{P}]\!]^{\mathrm{Max}}.\ \sigma \preceq \sigma' \Rightarrow \sigma' \in \mathcal{X}\})$
⟅def. γ_{Pred}⟆

$= [\![\mathrm{P}]\!]^{\mathrm{Max}} \cap \mathcal{X}$ ⟅$[\![\mathrm{P}]\!]^{\mathrm{Max}} \cap (\mathrm{Pref}(\mathcal{X}) \backslash [\![\mathrm{P}]\!]^{\mathrm{Max}}) = \emptyset$⟆

$= \mathcal{X}$ ⟅$\mathcal{X} \in \wp([\![\mathrm{P}]\!]^{\mathrm{Max}})$⟆

– $\alpha_{\mathrm{Pred}}([\![\mathrm{P}]\!]^{\mathrm{Max}}) \circ \gamma_{\mathrm{Pred}}([\![\mathrm{P}]\!]^{\mathrm{Max}})\mathcal{Y}$

$= \alpha_{\mathrm{Pred}}([\![\mathrm{P}]\!]^{\mathrm{Max}}) \circ \gamma_{\mathrm{Pred}}([\![\mathrm{P}]\!]^{\mathrm{Max}}) \circ \alpha_{\mathrm{Pred}}([\![\mathrm{P}]\!]^{\mathrm{Max}})\mathcal{X}'$

$$\wr \mathcal{Y} \in \bar{\alpha}_{\mathsf{Pred}}\{[\![\mathrm{P}]\!]^{\mathsf{Max}}\}(\wp([\![\mathrm{P}]\!]^{\mathsf{Max}})), \text{ thus } \exists \mathcal{X}'.\ \mathcal{Y} = \alpha_{\mathsf{Pred}}([\![\mathrm{P}]\!]^{\mathsf{Max}})\mathcal{X}' \wr$$

$$= \alpha_{\mathsf{Pred}}([\![\mathrm{P}]\!]^{\mathsf{Max}})\mathcal{X}' \qquad \wr \gamma_{\mathsf{Pred}}([\![\mathrm{P}]\!]^{\mathsf{Max}}) \circ \alpha_{\mathsf{Pred}}([\![\mathrm{P}]\!]^{\mathsf{Max}})\mathcal{X}' = \mathcal{X}' \wr$$

$$= \mathcal{Y} \qquad \wr \text{by the assumption } \mathcal{Y} = \alpha_{\mathsf{Pred}}([\![\mathrm{P}]\!]^{\mathsf{Max}})\mathcal{X}' \wr$$

– By the four properties proved above, $\alpha_{\mathsf{Pred}}([\![\mathrm{P}]\!]^{\mathsf{Max}})$ and $\gamma_{\mathsf{Pred}}([\![\mathrm{P}]\!]^{\mathsf{Max}})$ form a Galois isomorphism. □

A.2 Proofs of Corollary 1 and 2

Corollary 1. *Given the semantics $[\![\mathrm{P}]\!]^{\mathsf{Max}}$ and lattice $\mathcal{L}^{\mathsf{Max}}$ of system behaviors, for any maximal trace property $\mathcal{T} \in \mathcal{L}^{\mathsf{Max}}$, if a trace σ belongs to the prediction trace property that corresponds to $\mathcal{T}$, then every valid trace greater than σ belongs to that prediction trace property too. I.e., $\forall \mathcal{T} \in \mathcal{L}^{\mathsf{Max}}.\ \forall \sigma, \sigma' \in [\![\mathrm{P}]\!]^{\mathsf{Pref}}.\ (\sigma \in \alpha_{\mathsf{Pred}}([\![\mathrm{P}]\!]^{\mathsf{Max}})\mathcal{T} \wedge \sigma \preceq \sigma') \Rightarrow \sigma' \in \alpha_{\mathsf{Pred}}([\![\mathrm{P}]\!]^{\mathsf{Max}})\mathcal{T}$.*

Proof. Proof by contradiction. We assume $\exists \mathcal{T} \in \mathcal{L}^{\mathsf{Max}}.\ \exists \sigma, \sigma' \in [\![\mathrm{P}]\!]^{\mathsf{Pref}}.\ \sigma \in \alpha_{\mathsf{Pred}}([\![\mathrm{P}]\!]^{\mathsf{Max}})\mathcal{T} \wedge \sigma \preceq \sigma' \wedge \sigma' \notin \alpha_{\mathsf{Pred}}([\![\mathrm{P}]\!]^{\mathsf{Max}})\mathcal{T}$. By the definition of prediction abstraction, $\alpha_{\mathsf{Pred}}([\![\mathrm{P}]\!]^{\mathsf{Max}})\mathcal{T} = \{\sigma \in \mathsf{Pref}(\mathcal{T}) \mid \forall \sigma' \in [\![\mathrm{P}]\!]^{\mathsf{Max}}.\ \sigma \preceq \sigma' \Rightarrow \sigma' \in \mathcal{T}\}$. There are two possibilities for $\sigma' \notin \alpha_{\mathsf{Pred}}([\![\mathrm{P}]\!]^{\mathsf{Max}})\mathcal{T}$:

(1) $\sigma' \notin \mathsf{Pref}(\mathcal{T})$, hence every maximal trace greater than σ' does not belong to $\mathcal{T}$;

(2) $\exists \sigma'' \in [\![\mathrm{P}]\!]^{\mathsf{Max}}.\ \sigma' \preceq \sigma'' \wedge \sigma'' \notin \mathcal{T}$.

Both cases imply that there is a maximal trace $\sigma'' \in [\![\mathrm{P}]\!]^{\mathsf{Max}}$ such that $\sigma \preceq \sigma' \preceq \sigma'' \wedge \sigma'' \notin \mathcal{T}$, which contradicts with the assumption of $\sigma \in \alpha_{\mathsf{Pred}}([\![\mathrm{P}]\!]^{\mathsf{Max}})\mathcal{T}$. □

Corollary 2. *Given the semantics $[\![\mathrm{P}]\!]^{\mathsf{Max}}$ and the lattice $\mathcal{L}^{\mathsf{Max}}$ of system behaviors, for any maximal trace property $\mathcal{T} \in \mathcal{L}^{\mathsf{Max}}$ and any valid prefix trace π that is not maximal, if every valid prefix trace πs that concatenates π with a new event s belongs to the prediction trace property $\alpha_{\mathsf{Pred}}([\![\mathrm{P}]\!]^{\mathsf{Max}})\mathcal{T}$, then π belongs to $\alpha_{\mathsf{Pred}}([\![\mathrm{P}]\!]^{\mathsf{Max}})\mathcal{T}$ too. More formally, $\forall \mathcal{T} \in \mathcal{L}^{\mathsf{Max}}.\ \forall \pi \in [\![\mathrm{P}]\!]^{\mathsf{Pref}} \setminus [\![\mathrm{P}]\!]^{\mathsf{Max}}.\ (\forall s \in \mathbb{S}.\ \pi s \in [\![\mathrm{P}]\!]^{\mathsf{Pref}} \Rightarrow \pi s \in \alpha_{\mathsf{Pred}}([\![\mathrm{P}]\!]^{\mathsf{Max}})\mathcal{T}) \Rightarrow \pi \in \alpha_{\mathsf{Pred}}([\![\mathrm{P}]\!]^{\mathsf{Max}})\mathcal{T}$.*

Proof. Prove by contradiction, and here we assume that $\exists \mathcal{T} \in \mathcal{L}^{\mathsf{Max}}.\ \exists \pi \in [\![\mathrm{P}]\!]^{\mathsf{Pref}} \setminus [\![\mathrm{P}]\!]^{\mathsf{Max}}.\ (\forall s \in \mathbb{S}.\ \pi s \in [\![\mathrm{P}]\!]^{\mathsf{Pref}} \Rightarrow \pi s \in \alpha_{\mathsf{Pred}}([\![\mathrm{P}]\!]^{\mathsf{Max}})\mathcal{T}) \wedge \pi \notin \alpha_{\mathsf{Pred}}([\![\mathrm{P}]\!]^{\mathsf{Max}})\mathcal{T}$. According to the definition that $\alpha_{\mathsf{Pred}}([\![\mathrm{P}]\!]^{\mathsf{Max}})\mathcal{T} = \{\sigma \in \mathsf{Pref}(\mathcal{T}) \mid \forall \sigma' \in [\![\mathrm{P}]\!]^{\mathsf{Max}}.\ \sigma \preceq \sigma' \Rightarrow \sigma' \in \mathcal{T}\}$, there are two possibilities to have $\pi \notin \alpha_{\mathsf{Pred}}([\![\mathrm{P}]\!]^{\mathsf{Max}})\mathcal{T}$:

(1) $\pi \notin \mathsf{Pref}(\mathcal{T})$. This implies that $\forall s \in \mathbb{S}.\ \pi s \in [\![\mathrm{P}]\!]^{\mathsf{Pref}} \Rightarrow \pi s \notin \mathsf{Pref}(\mathcal{T})$, which further implies that $\forall s \in \mathbb{S}.\ \pi s \in [\![\mathrm{P}]\!]^{\mathsf{Pref}} \Rightarrow \pi s \notin \alpha_{\mathsf{Pred}}([\![\mathrm{P}]\!]^{\mathsf{Max}})\mathcal{T}$. Since $\pi \in [\![\mathrm{P}]\!]^{\mathsf{Pref}} \setminus [\![\mathrm{P}]\!]^{\mathsf{Max}}$, there must exist at least one s such that $\pi s \in [\![\mathrm{P}]\!]^{\mathsf{Pref}} \wedge \pi s \notin \alpha_{\mathsf{Pred}}([\![\mathrm{P}]\!]^{\mathsf{Max}})\mathcal{T}$.

(2) There is a maximal trace $\sigma' \in [\![\mathrm{P}]\!]^{\mathsf{Max}}$ such that $\pi \prec \sigma' \wedge \sigma' \notin \mathcal{T}$. Take $s = \sigma'_{[|\pi|]}$, then $\pi s \in [\![\mathrm{P}]\!]^{\mathsf{Pref}} \wedge \pi s \preceq \sigma' \wedge \sigma' \notin \mathcal{T}$ holds, which implies $\pi s \in [\![\mathrm{P}]\!]^{\mathsf{Pref}} \wedge \pi s \notin \alpha_{\mathsf{Pred}}([\![\mathrm{P}]\!]^{\mathsf{Max}})\mathcal{T}$.

Both two cases find that $\exists s \in \mathbb{S}.\ \pi s \in [\![\mathrm{P}]\!]^{\mathsf{Pref}} \wedge \pi s \notin \alpha_{\mathsf{Pred}}([\![\mathrm{P}]\!]^{\mathsf{Max}})\mathcal{T}$, which contradicts with the assumption $\forall s \in \mathbb{S}.\ \pi s \in [\![\mathrm{P}]\!]^{\mathsf{Pref}} \Rightarrow \pi s \in \alpha_{\mathsf{Pred}}([\![\mathrm{P}]\!]^{\mathsf{Max}})\mathcal{T}$. □

A.3 Proof of Lemma 2 and Corollary 4

Lemma 2. *Given the semantics $[\![\mathrm{P}]\!]^{\mathsf{Max}}$ and lattice $\mathcal{L}^{\mathsf{Max}}$ of system behaviors, the inquiry function $\mathbb{I}([\![\mathrm{P}]\!]^{\mathsf{Max}}, \mathcal{L}^{\mathsf{Max}})$ is decreasing on the inquired trace σ: The greater (longer) σ is, the stronger property it can guarantee. I.e., $\forall \sigma, \sigma' \in \mathbb{S}^{*\infty}.\ \sigma \preceq \sigma' \Rightarrow \mathbb{I}([\![\mathrm{P}]\!]^{\mathsf{Max}}, \mathcal{L}^{\mathsf{Max}}, \sigma) \supseteq \mathbb{I}([\![\mathrm{P}]\!]^{\mathsf{Max}}, \mathcal{L}^{\mathsf{Max}}, \sigma')$.*

Proof. First, if σ is invalid (i.e., $\sigma \notin [\![\mathrm{P}]\!]^{\mathsf{Pref}}$), then every trace σ' that is greater than σ must also be invalid (i.e., $\sigma' \notin [\![\mathrm{P}]\!]^{\mathsf{Pref}}$), hence it is obvious that $\mathbb{I}([\![\mathrm{P}]\!]^{\mathsf{Max}}, \mathcal{L}^{\mathsf{Max}}, \sigma) = \mathbb{I}([\![\mathrm{P}]\!]^{\mathsf{Max}}, \mathcal{L}^{\mathsf{Max}}, \sigma') = \bot^{\mathsf{Max}}$.

Second, if σ' is invalid ($\sigma' \notin [\![\mathrm{P}]\!]^{\mathsf{Pref}}$), then $\mathbb{I}([\![\mathrm{P}]\!]^{\mathsf{Max}}, \mathcal{L}^{\mathsf{Max}}, \sigma') = \bot^{\mathsf{Max}}$, hence $\mathbb{I}([\![\mathrm{P}]\!]^{\mathsf{Max}}, \mathcal{L}^{\mathsf{Max}}, \sigma) \supseteq \bot^{\mathsf{Max}} = \mathbb{I}([\![\mathrm{P}]\!]^{\mathsf{Max}}, \mathcal{L}^{\mathsf{Max}}, \sigma')$.

Last, if both σ and σ' are valid($\sigma, \sigma' \in [\![\mathrm{P}]\!]^{\mathrm{Pref}}$), then we have

$$
\begin{aligned}
&\sigma \preceq \sigma' \\
\Rightarrow\ & \forall \mathcal{T} \in \mathcal{L}^{\mathrm{Max}}.\ \sigma \in \alpha_{\mathrm{Pred}}([\![\mathrm{P}]\!]^{\mathrm{Max}})\mathcal{T} \Rightarrow \sigma' \in \alpha_{\mathrm{Pred}}([\![\mathrm{P}]\!]^{\mathrm{Max}})\mathcal{T} && \text{corollary 1} \\
\Rightarrow\ & \{\mathcal{T} \in \mathcal{L}^{\mathrm{Max}} \mid \sigma \in \alpha_{\mathrm{Pred}}([\![\mathrm{P}]\!]^{\mathrm{Max}})\mathcal{T}\} \subseteq \{\mathcal{T} \in \mathcal{L}^{\mathrm{Max}} \mid \sigma' \in \alpha_{\mathrm{Pred}}([\![\mathrm{P}]\!]^{\mathrm{Max}})\mathcal{T}\} && \text{def. } \Rightarrow \\
\Rightarrow\ & \cap\{\mathcal{T} \in \mathcal{L}^{\mathrm{Max}} \mid \sigma \in \alpha_{\mathrm{Pred}}([\![\mathrm{P}]\!]^{\mathrm{Max}})\mathcal{T}\} \supseteq \cap\{\mathcal{T} \in \mathcal{L}^{\mathrm{Max}} \mid \sigma' \in \alpha_{\mathrm{Pred}}([\![\mathrm{P}]\!]^{\mathrm{Max}})\mathcal{T}\} && \text{def. } \cap \\
\Rightarrow\ & \mathbb{I}([\![\mathrm{P}]\!]^{\mathrm{Max}}, \mathcal{L}^{\mathrm{Max}}, \sigma) \supseteq \mathbb{I}([\![\mathrm{P}]\!]^{\mathrm{Max}}, \mathcal{L}^{\mathrm{Max}}, \sigma'). && \text{def. } \mathbb{I}
\end{aligned}
$$

To sum up the three cases above, we prove that $\mathbb{I}([\![\mathrm{P}]\!]^{\mathrm{Max}}, \mathcal{L}^{\mathrm{Max}})$ is decreasing. □

Corollary 4. *Given the semantics $[\![\mathrm{P}]\!]^{\mathrm{Max}}$ and lattice $\mathcal{L}^{\mathrm{Max}}$ of behaviors, $\forall \sigma \in [\![\mathrm{P}]\!]^{\mathrm{Pref}} \backslash [\![\mathrm{P}]\!]^{\mathrm{Max}}$. $\mathbb{I}([\![\mathrm{P}]\!]^{\mathrm{Max}}, \mathcal{L}^{\mathrm{Max}}, \sigma) = \bigcup_{s \in \mathbb{S}} \mathbb{I}([\![\mathrm{P}]\!]^{\mathrm{Max}}, \mathcal{L}^{\mathrm{Max}}, \sigma s) = \bigcup_{\sigma s \in [\![\mathrm{P}]\!]^{\mathrm{Pref}}} \mathbb{I}([\![\mathrm{P}]\!]^{\mathrm{Max}}, \mathcal{L}^{\mathrm{Max}}, \sigma s)$.*

Proof. First, it is easy to see that $\cup\{\mathbb{I}([\![\mathrm{P}]\!]^{\mathrm{Max}}, \mathcal{L}^{\mathrm{Max}}, \sigma s) \mid s \in \mathbb{S}\} = (\cup\{\mathbb{I}([\![\mathrm{P}]\!]^{\mathrm{Max}}, \mathcal{L}^{\mathrm{Max}}, \sigma s) \mid \sigma s \in [\![\mathrm{P}]\!]^{\mathrm{Pref}}\}) \cup (\cup\{\mathbb{I}([\![\mathrm{P}]\!]^{\mathrm{Max}}, \mathcal{L}^{\mathrm{Max}}, \sigma s) \mid \sigma s \notin [\![\mathrm{P}]\!]^{\mathrm{Pref}}\}) = (\cup\{\mathbb{I}([\![\mathrm{P}]\!]^{\mathrm{Max}}, \mathcal{L}^{\mathrm{Max}}, \sigma s) \mid \sigma s \in [\![\mathrm{P}]\!]^{\mathrm{Pref}}\}) \cup \bot^{\mathrm{Max}} = \cup\{\mathbb{I}([\![\mathrm{P}]\!]^{\mathrm{Max}}, \mathcal{L}^{\mathrm{Max}}, \sigma s) \mid \sigma s \in [\![\mathrm{P}]\!]^{\mathrm{Pref}}\}$.

Second, we prove $\cup\{\mathbb{I}([\![\mathrm{P}]\!]^{\mathrm{Max}}, \mathcal{L}^{\mathrm{Max}}, \sigma s) \mid \sigma s \in [\![\mathrm{P}]\!]^{\mathrm{Pref}}\} = \mathbb{I}([\![\mathrm{P}]\!]^{\mathrm{Max}}, \mathcal{L}^{\mathrm{Max}}, \sigma)$ in two steps: (1) by Lemma 2, it is proved that $\forall \sigma, \sigma s \in \mathbb{S}^{*\infty}$. $\mathbb{I}([\![\mathrm{P}]\!]^{\mathrm{Max}}, \mathcal{L}^{\mathrm{Max}}, \sigma) \supseteq \mathbb{I}([\![\mathrm{P}]\!]^{\mathrm{Max}}, \mathcal{L}^{\mathrm{Max}}, \sigma s)$, thus $\mathbb{I}([\![\mathrm{P}]\!]^{\mathrm{Max}}, \mathcal{L}^{\mathrm{Max}}, \sigma) \supseteq \cup\{\mathbb{I}([\![\mathrm{P}]\!]^{\mathrm{Max}}, \mathcal{L}^{\mathrm{Max}}, \sigma s) \mid \sigma s \in [\![\mathrm{P}]\!]^{\mathrm{Pref}}\}$. (2) assume $\mathbb{I}([\![\mathrm{P}]\!]^{\mathrm{Max}}, \mathcal{L}^{\mathrm{Max}}, \sigma) \supsetneq \cup\{\mathbb{I}([\![\mathrm{P}]\!]^{\mathrm{Max}}, \mathcal{L}^{\mathrm{Max}}, \sigma s) \mid \sigma s \in [\![\mathrm{P}]\!]^{\mathrm{Pref}}\} = \mathcal{T}$. By the definition of $\mathbb{I}$ in (2), we know that $\sigma \notin \alpha_{\mathrm{Pred}}([\![\mathrm{P}]\!]^{\mathrm{Max}})\mathcal{T}$ and $\forall \sigma s \in [\![\mathrm{P}]\!]^{\mathrm{Pref}}$. $\sigma s \in \alpha_{\mathrm{Pred}}([\![\mathrm{P}]\!]^{\mathrm{Max}})\mathcal{T}$, which is impossible by Corollary 2. Thus, by contradiction, $\mathbb{I}([\![\mathrm{P}]\!]^{\mathrm{Max}}, \mathcal{L}^{\mathrm{Max}}, \sigma) = \cup\{\mathbb{I}([\![\mathrm{P}]\!]^{\mathrm{Max}}, \mathcal{L}^{\mathrm{Max}}, \sigma s) \mid \sigma s \in [\![\mathrm{P}]\!]^{\mathrm{Pref}}\}$. □

A.4 Proofs of Corollary 6, 7, and Lemma 3

Corollary 6. *Given the semantics $[\![\mathrm{P}]\!]^{\mathrm{Max}}$ and lattice $\mathcal{L}^{\mathrm{Max}}$ of system behaviors, for any observer with cognizance $\mathbb{C}$, if the corresponding observation function maps a trace σ to a maximal trace property $\mathcal{T} \in \mathcal{L}^{\mathrm{Max}}$, then σ guarantees the satisfaction of property $\mathcal{T}$ (i.e., every valid maximal trace that is greater than or equal to σ is guaranteed to have property $\mathcal{T}$).*

Proof. Suppose $\mathbb{I}([\![\mathrm{P}]\!]^{\mathrm{Max}}, \mathcal{L}^{\mathrm{Max}}, \sigma) = \mathcal{T}'$. By Corollary 3, σ guarantees the property $\mathcal{T}'$, i.e., every valid maximal trace that is greater than or equal to σ belongs to $\mathcal{T}'$.

In addition, since the cognizance is extensive (i.e., $\sigma \in \mathbb{C}(\sigma)$), then from the definition of observation function in (4), we know that $\mathcal{T} = \mathbb{O}([\![\mathrm{P}]\!]^{\mathrm{Max}}, \mathcal{L}^{\mathrm{Max}}, \mathbb{C}, \sigma) = \cup\{\mathbb{I}([\![\mathrm{P}]\!]^{\mathrm{Max}}, \mathcal{L}^{\mathrm{Max}}, \sigma') \mid \sigma' \in \mathbb{C}(\sigma)\} \supseteq \mathbb{I}([\![\mathrm{P}]\!]^{\mathrm{Max}}, \mathcal{L}^{\mathrm{Max}}, \sigma) = \mathcal{T}'$. Therefore, every valid maximal trace that is greater than or equal to σ belongs to $\mathcal{T}$. That is to say, σ guarantees the satisfaction of property $\mathcal{T}$. □

Corollary 7. *Given the semantics $[\![\mathrm{P}]\!]^{\mathrm{Max}}$, the lattice $\mathcal{L}^{\mathrm{Max}}$ of system behaviors and the cognizance function $\mathbb{C}$, we have: $\forall \sigma \in [\![\mathrm{P}]\!]^{\mathrm{Pref}} \backslash [\![\mathrm{P}]\!]^{\mathrm{Max}}$. $\mathbb{O}([\![\mathrm{P}]\!]^{\mathrm{Max}}, \mathcal{L}^{\mathrm{Max}}, \mathbb{C}, \sigma) = \bigcup_{s \in \mathbb{S}} \mathbb{O}([\![\mathrm{P}]\!]^{\mathrm{Max}}, \mathcal{L}^{\mathrm{Max}}, \mathbb{C}, \sigma s) = \bigcup_{\sigma s \in [\![\mathrm{P}]\!]^{\mathrm{Pref}}} \mathbb{O}([\![\mathrm{P}]\!]^{\mathrm{Max}}, \mathcal{L}^{\mathrm{Max}}, \mathbb{C}, \sigma s)$.*

Proof. We start the proof from the right side:

$$
\begin{aligned}
& \bigcup_{\sigma s \in [\![\mathrm{P}]\!]^{\mathrm{Pref}}} \mathbb{O}([\![\mathrm{P}]\!]^{\mathrm{Max}}, \mathcal{L}^{\mathrm{Max}}, \mathbb{C}, \sigma s) \\
=\ & \Big(\bigcup_{\sigma s \in [\![\mathrm{P}]\!]^{\mathrm{Pref}}} \mathbb{O}([\![\mathrm{P}]\!]^{\mathrm{Max}}, \mathcal{L}^{\mathrm{Max}}, \mathbb{C}, \sigma s)\Big) \cup \bot^{\mathrm{Max}} && \text{def.} \bot^{\mathrm{Max}} \\
=\ & \bigcup_{\sigma s \in [\![\mathrm{P}]\!]^{\mathrm{Pref}}} \mathbb{O}([\![\mathrm{P}]\!]^{\mathrm{Max}}, \mathcal{L}^{\mathrm{Max}}, \mathbb{C}, \sigma s) \cup \bigcup_{\sigma s \notin [\![\mathrm{P}]\!]^{\mathrm{Pref}}} \mathbb{O}([\![\mathrm{P}]\!]^{\mathrm{Max}}, \mathcal{L}^{\mathrm{Max}}, \mathbb{C}, \sigma s) && \text{def. } \mathbb{O} \\
=\ & \bigcup_{s \in \mathbb{S}} \mathbb{O}([\![\mathrm{P}]\!]^{\mathrm{Max}}, \mathcal{L}^{\mathrm{Max}}, \mathbb{C}, \sigma s) && \text{merge two cases}
\end{aligned}
$$

$$
\begin{aligned}
&= \uplus\{\mathbb{I}([\![\mathrm{P}]\!]^{\mathrm{Max}}, \mathcal{L}^{\mathrm{Max}}, \pi) \mid \pi \in \mathbb{C}(\sigma s) \wedge s \in \mathbb{S}\} && \text{⟅def. } \mathbb{O}\text{⟆}\\
&= \uplus\{\mathbb{I}([\![\mathrm{P}]\!]^{\mathrm{Max}}, \mathcal{L}^{\mathrm{Max}}, \sigma'\sigma'') \mid \sigma'\sigma'' \in \mathbb{C}(\sigma s) \wedge s \in \mathbb{S}\} && \text{⟅replace } \pi \text{ with } \sigma'\sigma''\text{⟆}\\
&= \uplus\{\mathbb{I}([\![\mathrm{P}]\!]^{\mathrm{Max}}, \mathcal{L}^{\mathrm{Max}}, \sigma'\sigma'') \mid \sigma' \in \mathbb{C}(\sigma) \wedge \sigma'' \in \mathbb{C}(s) \wedge s \in \mathbb{S}\} && \text{⟅assumption (A1)⟆}\\
&= (\uplus\{\mathbb{I}([\![\mathrm{P}]\!]^{\mathrm{Max}}, \mathcal{L}^{\mathrm{Max}}, \sigma'\sigma'') \mid \sigma' \in \mathbb{C}(\sigma) \wedge \sigma'' \in \mathbb{C}(s) \wedge s \in \mathbb{S} \wedge |\sigma''| = 1\}) \uplus\\
&\quad (\uplus\{\mathbb{I}([\![\mathrm{P}]\!]^{\mathrm{Max}}, \mathcal{L}^{\mathrm{Max}}, \sigma'\sigma'') \mid \sigma' \in \mathbb{C}(\sigma) \wedge \sigma'' \in \mathbb{C}(s) \wedge s \in \mathbb{S} \wedge |\sigma''| = 0\}) \uplus\\
&\quad (\uplus\{\mathbb{I}([\![\mathrm{P}]\!]^{\mathrm{Max}}, \mathcal{L}^{\mathrm{Max}}, \sigma'\sigma'') \mid \sigma' \in \mathbb{C}(\sigma) \wedge \sigma'' \in \mathbb{C}(s) \wedge s \in \mathbb{S} \wedge |\sigma''| > 1\}).
\end{aligned}
$$

In the above, the formula is split into three cases by the length of σ''. The first case:

$$
\begin{aligned}
&\uplus\{\mathbb{I}([\![\mathrm{P}]\!]^{\mathrm{Max}}, \mathcal{L}^{\mathrm{Max}}, \sigma'\sigma'') \mid \sigma' \in \mathbb{C}(\sigma) \wedge \sigma'' \in \mathbb{C}(s) \wedge s \in \mathbb{S} \wedge |\sigma''| = 1\}\\
&= \uplus\{\mathbb{I}([\![\mathrm{P}]\!]^{\mathrm{Max}}, \mathcal{L}^{\mathrm{Max}}, \sigma' s) \mid \sigma' \in \mathbb{C}(\sigma) \wedge s \in \mathbb{S}\} && \text{⟅corollary 5⟆}\\
&= \uplus\{\mathbb{I}([\![\mathrm{P}]\!]^{\mathrm{Max}}, \mathcal{L}^{\mathrm{Max}}, \sigma') \mid \sigma' \in \mathbb{C}(\sigma)\} && \text{⟅corollary 4⟆}\\
&= \mathbb{O}([\![\mathrm{P}]\!]^{\mathrm{Max}}, \mathcal{L}^{\mathrm{Max}}, \mathbb{C}, \sigma). && \text{⟅def. } \mathbb{O}\text{⟆}
\end{aligned}
$$

The second case: If there is $s \in \mathbb{S}$ such that $\varepsilon \in \mathbb{C}(s)$, then $\uplus\{\mathbb{I}([\![\mathrm{P}]\!]^{\mathrm{Max}}, \mathcal{L}^{\mathrm{Max}}, \sigma'\sigma'') \mid \sigma' \in \mathbb{C}(\sigma) \wedge \sigma'' \in \mathbb{C}(s) \wedge s \in \mathbb{S} \wedge |\sigma''| = 0\} = \uplus\{\mathbb{I}([\![\mathrm{P}]\!]^{\mathrm{Max}}, \mathcal{L}^{\mathrm{Max}}, \sigma') \mid \sigma' \in \mathbb{C}(\sigma)\} = \mathbb{O}([\![\mathrm{P}]\!]^{\mathrm{Max}}, \mathcal{L}^{\mathrm{Max}}, \mathbb{C}, \sigma)$. Otherwise, it is an empty set.

The third case:

$$
\begin{aligned}
&\uplus\{\mathbb{I}([\![\mathrm{P}]\!]^{\mathrm{Max}}, \mathcal{L}^{\mathrm{Max}}, \sigma'\sigma'') \mid \sigma' \in \mathbb{C}(\sigma) \wedge \sigma'' \in \mathbb{C}(s) \wedge s \in \mathbb{S} \wedge |\sigma''| > 1\}\\
&\subseteq \uplus\{\mathbb{I}([\![\mathrm{P}]\!]^{\mathrm{Max}}, \mathcal{L}^{\mathrm{Max}}, \sigma') \mid \sigma' \in \mathbb{C}(\sigma) \wedge \sigma'' \in \mathbb{C}(s) \wedge s \in \mathbb{S} \wedge |\sigma''| > 1\} && \text{⟅lemma 2⟆}\\
&\subseteq \uplus\{\mathbb{I}([\![\mathrm{P}]\!]^{\mathrm{Max}}, \mathcal{L}^{\mathrm{Max}}, \sigma') \mid \sigma' \in \mathbb{C}(\sigma)\} && \text{⟅def. } \uplus\text{⟆}\\
&= \mathbb{O}([\![\mathrm{P}]\!]^{\mathrm{Max}}, \mathcal{L}^{\mathrm{Max}}, \mathbb{C}, \sigma). && \text{⟅def. } \mathbb{O}\text{⟆}
\end{aligned}
$$

Joining the above three cases together, we have proved that $\biguplus_{\sigma s \in [\![\mathrm{P}]\!]^{\mathrm{Pref}}} \mathbb{O}([\![\mathrm{P}]\!]^{\mathrm{Max}}, \mathcal{L}^{\mathrm{Max}}, \mathbb{C}, \sigma s) = \mathbb{O}([\![\mathrm{P}]\!]^{\mathrm{Max}}, \mathcal{L}^{\mathrm{Max}}, \mathbb{C}, \sigma)$. □

Lemma 3. *Given the semantics $[\![\mathrm{P}]\!]^{\mathrm{Max}}$, lattice $\mathcal{L}^{\mathrm{Max}}$ of system behaviors and cognizance function $\mathbb{C}$, the observation function $\mathbb{O}([\![\mathrm{P}]\!]^{\mathrm{Max}}, \mathcal{L}^{\mathrm{Max}}, \mathbb{C})$ is decreasing on the observed trace σ: The greater (longer) σ is, the stronger property it can observe. I.e., $\forall \sigma, \sigma' \in \mathbb{S}^{*\infty}.\ \sigma \preceq \sigma' \Rightarrow \mathbb{O}([\![\mathrm{P}]\!]^{\mathrm{Max}}, \mathcal{L}^{\mathrm{Max}}, \mathbb{C}, \sigma) \supseteq \mathbb{O}([\![\mathrm{P}]\!]^{\mathrm{Max}}, \mathcal{L}^{\mathrm{Max}}, \mathbb{C}, \sigma')$.*

Proof. We only need to consider the case where $\sigma \prec \sigma'$. First, if σ is invalid (i.e., $\sigma \notin [\![\mathrm{P}]\!]^{\mathrm{Pref}}$), then every trace σ' that is greater than σ must also be invalid (i.e., $\sigma' \notin [\![\mathrm{P}]\!]^{\mathrm{Pref}}$), hence we have $\mathbb{O}([\![\mathrm{P}]\!]^{\mathrm{Max}}, \mathcal{L}^{\mathrm{Max}}, \mathbb{C}, \sigma) = \mathbb{O}([\![\mathrm{P}]\!]^{\mathrm{Max}}, \mathcal{L}^{\mathrm{Max}}, \mathbb{C}, \sigma') = \bot^{\mathrm{Max}}$.

Second, if $\sigma' \notin [\![\mathrm{P}]\!]^{\mathrm{Pref}}$, then we have $\mathbb{O}([\![\mathrm{P}]\!]^{\mathrm{Max}}, \mathcal{L}^{\mathrm{Max}}, \mathbb{C}, \sigma') = \bot^{\mathrm{Max}}$. Hence, it is trivial to find $\mathbb{O}([\![\mathrm{P}]\!]^{\mathrm{Max}}, \mathcal{L}^{\mathrm{Max}}, \mathbb{C}, \sigma) \supseteq \bot^{\mathrm{Max}} = \mathbb{O}([\![\mathrm{P}]\!]^{\mathrm{Max}}, \mathcal{L}^{\mathrm{Max}}, \mathbb{C}, \sigma')$.

Last, if $\sigma, \sigma' \in [\![\mathrm{P}]\!]^{\mathrm{Pref}}$, then σ must be a valid non-maximal trace, i.e., $\sigma \in [\![\mathrm{P}]\!]^{\mathrm{Pref}} \setminus [\![\mathrm{P}]\!]^{\mathrm{Max}}$. From Corollary 7, it is easy to see $\forall s \in \mathbb{S}.\ \mathbb{O}([\![\mathrm{P}]\!]^{\mathrm{Max}}, \mathcal{L}^{\mathrm{Max}}, \mathbb{C}, \sigma) \supseteq \mathbb{O}([\![\mathrm{P}]\!]^{\mathrm{Max}}, \mathcal{L}^{\mathrm{Max}}, \mathbb{C}, \sigma s)$. Since σ' is greater than σ(or, say, σ' is a prolongation of σ with states), then by the transitivity of $\supseteq$, it is not hard to see that $\mathbb{O}([\![\mathrm{P}]\!]^{\mathrm{Max}}, \mathcal{L}^{\mathrm{Max}}, \mathbb{C}, \sigma) \supseteq \mathbb{O}([\![\mathrm{P}]\!]^{\mathrm{Max}}, \mathcal{L}^{\mathrm{Max}}, \mathbb{C}, \sigma')$. □

A.5 Proof of Theorem 1

Theorem 1. *If τ_{R} is said to be responsible for a behavior $\mathcal{B}$ in a valid trace $\sigma_{\mathrm{H}}\tau_{\mathrm{R}}\sigma_{\mathrm{F}}$, then $\sigma_{\mathrm{H}}\tau_{\mathrm{R}}$ guarantees the occurrence of behavior $\mathcal{B}$, and there must exist another valid prefix trace $\sigma_{\mathrm{H}}\tau'_{\mathrm{R}}$ such that the behavior $\mathcal{B}$ is not guaranteed.*

Proof. First, from the definition of responsibility, we know $\mathbb{O}(\mathcal{S}^{\mathsf{Max}}, \mathcal{L}^{\mathsf{Max}}, \mathbb{C}, \sigma_{\mathrm{H}}\tau_{\mathrm{R}}) \subseteq \mathcal{B}$. By Corollary 6, $\sigma_{\mathrm{H}}\tau_{\mathrm{R}}$ guarantees the satisfaction of $\mathbb{O}(\mathcal{S}^{\mathsf{Max}}, \mathcal{L}^{\mathsf{Max}}, \mathbb{C}, \sigma_{\mathrm{H}}\tau_{\mathrm{R}})$, which is at least as strong as $\mathcal{B}$. Thus, the occurrence of behavior $\mathcal{B}$ is guaranteed.

Second, we prove by contradiction. Assume that every valid trace $\sigma_{\mathrm{H}}\tau'_{\mathrm{R}}$ guarantees the occurrence of behavior $\mathcal{B}$ (i.e., $\forall \sigma_{\mathrm{H}}\tau'_{\mathrm{R}} \in \mathcal{S}^{\mathsf{Pref}}.\ \mathbb{O}(\mathcal{S}^{\mathsf{Max}}, \mathcal{L}^{\mathsf{Max}}, \mathbb{C}, \sigma_{\mathrm{H}}\tau'_{\mathrm{R}}) \subseteq \mathcal{B}$). By Corollary 7, we can prove that $\mathbb{O}(\mathcal{S}^{\mathsf{Max}}, \mathcal{L}^{\mathsf{Max}}, \mathbb{C}, \sigma_{\mathrm{H}}) \subseteq \mathcal{B}$, which contradicts with the condition $\mathbb{O}([\![\mathrm{P}]\!]^{\mathsf{Max}}, \mathcal{L}^{\mathsf{Max}}, \mathbb{C}, \sigma_{\mathrm{H}}) \not\subseteq \mathcal{B}$ for τ_{R} to be responsible for the behavior $\mathcal{B}$. □

A.6 Proof of Galois Connection (6)

$$\langle \wp(\mathbb{S}^{*\infty}),\ \subseteq \rangle \underset{\alpha_{\mathbb{IV}}}{\overset{\gamma_{\mathbb{IV}}}{\leftrightarrows}} \langle \mathbb{L} \mapsto \wp(\mathbb{M}),\ \dot{\subseteq} \rangle.$$

Proof. For any $\mathcal{T} \in \wp(\mathbb{S}^{*\infty})$ and $\mathsf{I} \in \mathbb{L} \mapsto \wp(\mathbb{M})$, we can prove that:

$$\begin{aligned}
&\alpha_{\mathbb{IV}}(\mathcal{T}) \mathrel{\dot{\subseteq}} \mathsf{I} \\
\Leftrightarrow\ & \forall \ell \in \mathbb{L}.\ \alpha_{\mathbb{IV}}(\mathcal{T})\ell \subseteq \mathsf{I}(\ell) && \text{〖def. } \dot{\subseteq}\text{〗} \\
\Leftrightarrow\ & \forall \ell \in \mathbb{L}.\ \{\rho \in \mathbb{M} \mid \exists \sigma \in \mathcal{T}.\ \langle \ell,\ \rho\rangle \in \sigma\} \subseteq \mathsf{I}(\ell) && \text{〖def. } \alpha_{\mathbb{IV}}(S)\ell\text{〗} \\
\Leftrightarrow\ & \forall \ell \in \mathbb{L}.\ \forall \rho \in \mathbb{M}.\ (\exists \sigma \in \mathcal{T}.\ \langle \ell,\ \rho\rangle \in \sigma) \Rightarrow \rho \in \mathsf{I}(\ell) && \text{〖def. } \subseteq\text{〗} \\
\Leftrightarrow\ & \forall \ell \in \mathbb{L}.\ \forall \rho \in \mathbb{M}.\ \forall \sigma \in \mathbb{S}^{*\infty}.\ (\sigma \in \mathcal{T} \wedge \langle \ell,\ \rho\rangle \in \sigma) \Rightarrow \rho \in \mathsf{I}(\ell) && \text{〖def. } \exists\text{〗} \\
\Leftrightarrow\ & \forall \sigma \in \mathcal{T}.\ \forall \langle \ell,\ \rho\rangle \in \sigma.\ \rho \in \mathsf{I}(\ell) && \text{〖def. } \forall \text{ and } \Rightarrow\text{〗} \\
\Leftrightarrow\ & \mathcal{T} \subseteq \{\sigma \in \mathbb{S}^{*\infty} \mid \forall \langle \ell,\ \rho\rangle \in \sigma.\ \rho \in \mathsf{I}(\ell)\} && \text{〖def. } \subseteq\text{〗} \\
\Leftrightarrow\ & \mathcal{T} \subseteq \gamma_{\mathbb{IV}}(\mathsf{I}). && \text{〖def. } \gamma_{\mathbb{IV}}\text{〗}
\end{aligned}$$

By the above property, we have proved that $\alpha_{\mathbb{IV}}$ and $\gamma_{\mathbb{IV}}$ form a Galois connection. □

A.7 Proof of the Soundness Condition (7)

The abstract function $\hat{\mathrm{F}}^{\sharp}_{\overrightarrow{ps}}[\![\mathrm{P}]\!]$ obeys the following soundness condition ($\hat{\mathrm{F}}^{\sharp}_{\overrightarrow{ps}}[\![\mathrm{P}]\!]$):

$$\forall \mathsf{I}^{\sharp} \in \mathbb{L} \mapsto \mathcal{D}^{\sharp}_{\mathbb{M}}.\ \mathrm{F}_{\overrightarrow{ps}}[\![\mathrm{P}]\!] \circ \dot{\gamma}_{\mathbb{M}}(\mathsf{I}^{\sharp}) \mathrel{\dot{\subseteq}} \dot{\gamma}_{\mathbb{M}} \circ \hat{\mathrm{F}}^{\sharp}_{\overrightarrow{ps}}[\![\mathrm{P}]\!](\mathsf{I}^{\sharp}).$$

Proof. We start with the soundness of abstract atomic transfer function $\hat{\mathrm{F}}^{\sharp}_{\ell\to\ell'}[\![\mathrm{P}]\!]$:

$$\begin{aligned}
&\forall \ell, \ell' \in \mathbb{L}.\ \forall \mathrm{M}^{\sharp} \in \mathcal{D}^{\sharp}_{\mathbb{M}}.\ \mathrm{F}_{\ell\to\ell'}[\![\mathrm{P}]\!] \circ \gamma_{\mathbb{M}}(\mathrm{M}^{\sharp}) \subseteq \gamma_{\mathbb{M}} \circ \hat{\mathrm{F}}^{\sharp}_{\ell\to\ell'}[\![\mathrm{P}]\!](\mathrm{M}^{\sharp}) \\
\Rightarrow\ & \forall \ell, \ell' \in \mathbb{L}.\ \forall \mathsf{I}^{\sharp} \in \mathbb{L} \mapsto \mathcal{D}^{\sharp}_{\mathbb{M}}.\ \mathrm{F}_{\ell\to\ell'}[\![\mathrm{P}]\!] \circ \gamma_{\mathbb{M}}(\mathsf{I}^{\sharp}(\ell)) \subseteq \gamma_{\mathbb{M}} \circ \hat{\mathrm{F}}^{\sharp}_{\ell\to\ell'}[\![\mathrm{P}]\!](\mathsf{I}^{\sharp}(\ell)) \\
& && \text{〖by replacing } \mathrm{M}^{\sharp} \text{ with } \mathsf{I}^{\sharp}(\ell)\text{〗} \\
\Rightarrow\ & \forall \ell' \in \mathbb{L}.\ \forall \mathsf{I}^{\sharp} \in \mathbb{L} \mapsto \mathcal{D}^{\sharp}_{\mathbb{M}}.\ \cup_{\ell\in\mathbb{L}} \mathrm{F}_{\ell\to\ell'}[\![\mathrm{P}]\!](\gamma_{\mathbb{M}}(\mathsf{I}^{\sharp}(\ell))) \subseteq \gamma_{\mathbb{M}}(\sqcup^{\sharp}_{\mathbb{M}\,\ell\in\mathbb{L}} \hat{\mathrm{F}}^{\sharp}_{\ell\to\ell'}[\![\mathrm{P}]\!](\mathsf{I}^{\sharp}(\ell))) \\
& && \text{〖join on } \ell \in \mathbb{L}\text{, and } \sqcup^{\sharp}_{\mathbb{M}} \text{ soundly approximates } \cup\text{〗} \\
\Rightarrow\ & \forall \ell' \in \mathbb{L}.\ \forall \mathsf{I}^{\sharp} \in \mathbb{L} \mapsto \mathcal{D}^{\sharp}_{\mathbb{M}}.\ (\mathrm{F}_{\overrightarrow{ps}}[\![\mathrm{P}]\!] \circ \dot{\gamma}_{\mathbb{M}}(\mathsf{I}^{\sharp}))(\ell') \mathrel{\dot{\subseteq}} (\dot{\gamma}_{\mathbb{M}} \circ \hat{\mathrm{F}}^{\sharp}_{\overrightarrow{ps}}[\![\mathrm{P}]\!](\mathsf{I}^{\sharp}))(\ell') \\
& && \text{〖def. } \mathrm{F}_{\overrightarrow{ps}}[\![\mathrm{P}]\!] \text{ and } \hat{\mathrm{F}}^{\sharp}_{\overrightarrow{ps}}[\![\mathrm{P}]\!]\text{〗} \\
\Rightarrow\ & \forall \mathsf{I}^{\sharp} \in \mathbb{L} \mapsto \mathcal{D}^{\sharp}_{\mathbb{M}}.\ \mathrm{F}_{\overrightarrow{ps}}[\![\mathrm{P}]\!] \circ \dot{\gamma}_{\mathbb{M}}(\mathsf{I}^{\sharp}) \mathrel{\dot{\subseteq}} \dot{\gamma}_{\mathbb{M}} \circ \hat{\mathrm{F}}^{\sharp}_{\overrightarrow{ps}}[\![\mathrm{P}]\!](\mathsf{I}^{\sharp}). && \text{〖def. } \dot{\subseteq}\text{〗}
\end{aligned}$$

□

A.8 Proof of Galois Connection (9)

$$\langle \mathbb{L} \mapsto \wp(\mathbb{M}),\ \dot{\subseteq} \rangle \underset{\mathcal{S}^{\overrightarrow{ps}}[\![\mathrm{P}]\!]}{\overset{\mathcal{S}^{\overleftarrow{if}}[\![\mathrm{P}]\!]}{\leftrightarrows}} \langle \mathbb{L} \mapsto \wp(\mathbb{M}),\ \dot{\subseteq} \rangle$$

Proof. For any $\mathsf{I}_{\mathsf{pre}}, \mathsf{I}_{\mathsf{post}} \in \mathbb{L} \mapsto \wp(\mathbb{M})$, we can prove that:

$$\begin{aligned}
&\mathcal{S}^{\overrightarrow{ps}}[\![\mathrm{P}]\!](\mathsf{I}_{\mathsf{pre}}) \mathrel{\dot{\subseteq}} \mathsf{I}_{\mathsf{post}} \\
\Leftrightarrow\ & \forall \ell' \in \mathbb{L}.\ \mathcal{S}^{\overrightarrow{ps}}[\![\mathrm{P}]\!](\mathsf{I}_{\mathsf{pre}})\ell' \subseteq \mathsf{I}_{\mathsf{post}}(\ell') && \text{(def. } \dot{\subseteq}\text{)} \\
\Leftrightarrow\ & \forall \ell' \in \mathbb{L}.\ \{\rho' \in \mathbb{M} \mid \exists \sigma \in \mathbb{S}^*, \ell \in \mathbb{L}, \rho \in \mathsf{I}_{\mathsf{pre}}(\ell).\ \langle \ell,\ \rho\rangle\sigma\langle \ell',\ \rho'\rangle \in [\![\mathrm{P}]\!]^{\mathsf{lt}}\} \subseteq \mathsf{I}_{\mathsf{post}}(\ell') \\
& && \text{(def. } \mathcal{S}^{\overrightarrow{ps}}[\![\mathrm{P}]\!]\text{)} \\
\Leftrightarrow\ & \forall \ell' \in \mathbb{L}.\ \{\rho' \in \mathbb{M} \mid \neg(\forall \sigma \in \mathbb{S}^*, \ell \in \mathbb{L}, \rho \in \mathsf{I}_{\mathsf{pre}}(\ell).\ \langle \ell,\ \rho\rangle\sigma\langle \ell',\ \rho'\rangle \notin [\![\mathrm{P}]\!]^{\mathsf{lt}})\} \subseteq \{\rho' \in \mathbb{M} \mid \rho' \in \mathsf{I}_{\mathsf{post}}(\ell')\} && \text{(def. } \exists \text{ and } \forall\text{)} \\
\Leftrightarrow\ & \forall \ell' \in \mathbb{L}.\ \{\rho' \in \mathbb{M} \mid \forall \sigma \in \mathbb{S}^*, \ell \in \mathbb{L}, \rho \in \mathsf{I}_{\mathsf{pre}}(\ell).\ \langle \ell,\ \rho\rangle\sigma\langle \ell',\ \rho'\rangle \notin [\![\mathrm{P}]\!]^{\mathsf{lt}}\} \cup \{\rho' \in \mathbb{M} \mid \rho' \in \mathsf{I}_{\mathsf{post}}(\ell')\} = \mathbb{M} && \text{(def. } \neg \text{ and } \cup\text{)} \\
\Leftrightarrow\ & \forall \ell' \in \mathbb{L}.\ \forall \rho' \in \mathbb{M}.\ (\forall \sigma \in \mathbb{S}^*, \ell \in \mathbb{L}, \rho \in \mathsf{I}_{\mathsf{pre}}(\ell).\ \langle \ell,\ \rho\rangle\sigma\langle \ell',\ \rho'\rangle \notin [\![\mathrm{P}]\!]^{\mathsf{lt}}) \vee \rho' \in \mathsf{I}_{\mathsf{post}}(\ell') && \text{(def. } \vee\text{)} \\
\Leftrightarrow\ & \forall \ell \in \mathbb{L}.\ \forall \rho \in \mathsf{I}_{\mathsf{pre}}(\ell).\ \forall \sigma \in \mathbb{S}^*, \ell' \in \mathbb{L}, \rho' \in \mathbb{M}.\ \langle \ell,\ \rho\rangle\sigma\langle \ell',\ \rho'\rangle \in [\![\mathrm{P}]\!]^{\mathsf{lt}} \Rightarrow \rho' \in \mathsf{I}_{\mathsf{post}}(\ell') && \text{(def. } \Rightarrow\text{)} \\
\Leftrightarrow\ & \forall \ell \in \mathbb{L}.\ \mathsf{I}_{\mathsf{pre}}(\ell) \subseteq \{\rho \in \mathbb{M} \mid \forall \sigma \in \mathbb{S}^*, \ell' \in \mathbb{L}, \rho' \in \mathbb{M}.\ \langle \ell,\ \rho\rangle\sigma\langle \ell',\ \rho'\rangle \in [\![\mathrm{P}]\!]^{\mathsf{lt}} \Rightarrow \rho' \in \mathsf{I}_{\mathsf{post}}(\ell')\} \\
& && \text{(def. } \subseteq\text{)} \\
\Leftrightarrow\ & \forall \ell \in \mathbb{L}.\ \mathsf{I}_{\mathsf{pre}}(\ell) \subseteq \mathcal{S}^{\overleftarrow{if}}[\![\mathrm{P}]\!](\mathsf{I}_{\mathsf{post}})\ell && \text{(def. } \mathcal{S}^{\overleftarrow{if}}[\![\mathrm{P}]\!]\text{)} \\
\Leftrightarrow\ & \mathsf{I}_{\mathsf{pre}} \mathrel{\dot{\subseteq}} \mathcal{S}^{\overleftarrow{if}}[\![\mathrm{P}]\!](\mathsf{I}_{\mathsf{post}}). && \text{(def. } \dot{\subseteq}\text{)}
\end{aligned}$$

Thus, the forward (possible success) reachability semantics $\mathcal{S}^{\overrightarrow{ps}}[\![\mathrm{P}]\!]$ and the backward impossible failure accessibility semantics $\mathcal{S}^{\overleftarrow{if}}[\![\mathrm{P}]\!]$ form a Galois connection. □

A.9 Proof of Corollary 8, Lemma 4, and Theorem 2

Corollary 8. $\forall \rho \in \mathbb{M}.\ \forall \rho' \in [\rho]_{\sim_{\mathrm{M}_p^\sharp \backslash d_c}}.\ \exists \rho'' \in [\rho]_{d_c \atop \sim}.\ \forall \chi \in \mathit{vars}(\mathrm{M}_p^\sharp).\ \rho'(\chi) = \rho''(\chi)$.

Proof. The key is to prove there exists an environment ρ'' in $[\rho]_{d_c \atop \sim}$ that satisfies all the requirements. Here, we construct $\rho'' = \rho[\forall \chi \in \mathrm{vars}(\mathrm{M}_p^\sharp) \cup \mathrm{vars}(d_c).\ \chi \mapsto \rho'(\chi)]$ such that (i) $\forall \chi \in \mathrm{vars}(\mathrm{M}_p^\sharp) \cup \mathrm{vars}(d_c).\ \rho'(\chi) = \rho''(\chi)$ and (ii) $\forall \chi \in \mathbb{X} \backslash (\mathrm{vars}(\mathrm{M}_p^\sharp) \cup \mathrm{vars}(d_c)).\ \rho(\chi) = \rho''(\chi)$. Thus, the constructed environment ρ'' satisfies the requirement $\forall \chi \in \mathrm{vars}(\mathrm{M}_p^\sharp).\ \rho'(\chi) = \rho''(\chi)$.

Now, we only need to prove that $\rho'' \in [\rho]_{d_c \atop \sim}$. Since $\rho' \in [\rho]_{\sim_{\mathrm{M}_p^\sharp \backslash d_c}}$, by the definition of $\sim_{\mathrm{M}_p^\sharp \backslash d_c}$, we know that there are two possible cases: The first case is $\rho = \rho'$, then ρ'' is also equal to ρ, which makes $\rho'' \in [\rho]_{d_c \atop \sim}$ trivial; the second case is $\rho \in \gamma_{\mathbb{M}}(d_c) \wedge \rho' \in \gamma_{\mathbb{M}}(d_c)$ and (iii) $\forall \chi \in \mathrm{vars}(\mathrm{M}_p^\sharp) \backslash \mathrm{vars}(d_c).\ \rho(\chi) = \rho'(\chi)$. Since $\forall \chi \in \mathrm{vars}(d_c).\ \rho'(\chi) = \rho''(\chi)$, by (26) we can prove that $\rho'' \in \gamma_{\mathbb{M}}(d_c)$ holds. Moreover, combining (i) and (iii) together, we get $\forall \chi \in \mathrm{vars}(\mathrm{M}_p^\sharp) \backslash \mathrm{vars}(d_c).\ \rho(\chi) = \rho''(\chi)$, which further implies that $\forall \chi \in \mathbb{X} \backslash \mathrm{vars}(d_c).\ \rho(\chi) = \rho''(\chi)$. By the definition of $\overset{d_c}{\sim}$, we have proved that $\rho'' \in [\rho]_{d_c \atop \sim}$. □

Lemma 4. *A partitioning directive* $\mathrm{d}_p = \mathit{part}\langle \mathit{Inv}, \ell, \mathrm{M}_p^\sharp \rangle$ *is valid with respect to a cognizance directive* $\mathrm{d}_c \in \mathcal{D}_{\mathbb{M}}^\sharp$ *if and only if*

$$\forall \rho \in \gamma_{\mathbb{M}}(\mathrm{d}_c).\ [\rho]_{\sim_{\mathrm{M}_p^\sharp \backslash \mathrm{d}_c}} \subseteq \gamma_{\mathbb{M}}(\mathrm{M}_p^\sharp)\ \vee\ [\rho]_{\sim_{\mathrm{M}_p^\sharp \backslash \mathrm{d}_c}} \cap \gamma_{\mathbb{M}}(\mathrm{M}_p^\sharp) = \emptyset,$$

where $[\rho]_{\sim_{\mathrm{M}_p^\sharp \backslash \mathrm{d}_c}} = \{\rho' \in \mathbb{M} \mid \rho \sim_{\mathrm{M}_p^\sharp \backslash \mathrm{d}_c} \rho'\}$ *and* $\rho \sim_{\mathrm{M}_p^\sharp \backslash \mathrm{d}_c} \rho' \Leftrightarrow \rho = \rho' \vee (\rho \in \gamma_{\mathbb{M}}(\mathrm{d}_c) \wedge \rho' \in \gamma_{\mathbb{M}}(\mathrm{d}_c) \wedge \forall \chi \in \mathit{vars}(\mathrm{M}_p^\sharp) \backslash \mathit{vars}(\mathrm{d}_c).\ \rho(\chi) = \rho'(\chi))$.

Proof. To prove that the condition (25) is equivalent to the condition (28), we first need to show that: $\forall d_c, \mathrm{M}_p^\sharp \in \mathcal{D}_{\mathbb{M}}^\sharp.\ \forall \rho \in \mathbb{M}.\ [\rho]_{\underset{\sim}{d_c}} \subseteq \gamma_{\mathbb{M}}(\mathrm{M}_p^\sharp) \Leftrightarrow [\rho]_{\sim_{\mathrm{M}_p^\sharp \backslash d_c}} \subseteq \gamma_{\mathbb{M}}(\mathrm{M}_p^\sharp)$. The proof of this statement from right to left is trivial, since $[\rho]_{\underset{\sim}{d_c}} \subseteq [\rho]_{\sim_{\mathrm{M}_p^\sharp \backslash d_c}}$ (27). Here, we consider the opposite direction: $[\rho]_{\underset{\sim}{d_c}} \subseteq \gamma_{\mathbb{M}}(\mathrm{M}_p^\sharp) \Rightarrow [\rho]_{\sim_{\mathrm{M}_p^\sharp \backslash d_c}} \subseteq \gamma_{\mathbb{M}}(\mathrm{M}_p^\sharp)$. By Corollary (8), we have $\forall \rho' \in [\rho]_{\sim_{\mathrm{M}_p^\sharp \backslash d_c}}.\ \exists \rho'' \in [\rho]_{\underset{\sim}{d_c}}.\ \forall \chi \in \mathrm{vars}(\mathrm{M}_p^\sharp).\ \rho'(\chi) = \rho''(\chi)$. Since the assumption $[\rho]_{\underset{\sim}{d_c}} \subseteq \gamma_{\mathbb{M}}(\mathrm{M}_p^\sharp)$ implies that $\rho'' \in \gamma_{\mathbb{M}}(\mathrm{M}_p^\sharp)$, then by (26), we prove that $\rho' \in \gamma_{\mathbb{M}}(\mathrm{M}_p^\sharp)$, which implies that $[\rho]_{\sim_{\mathrm{M}_p^\sharp \backslash d_c}} \subseteq \gamma_{\mathbb{M}}(\mathrm{M}_p^\sharp)$.

Similarly, we can prove that $[\rho]_{\underset{\sim}{d_c}} \cap \gamma_{\mathbb{M}}(\mathrm{M}_p^\sharp) = \emptyset \Leftrightarrow [\rho]_{\sim_{\mathrm{M}_p^\sharp \backslash d_c}} \cap \gamma_{\mathbb{M}}(\mathrm{M}_p^\sharp) = \emptyset$.

Together, we have proved that $\forall d_c, \mathrm{M}_p^\sharp \in \mathcal{D}_{\mathbb{M}}^\sharp.\ \forall \rho \in \mathbb{M}.\ ([\rho]_{\underset{\sim}{d_c}} \subseteq \gamma_{\mathbb{M}}(\mathrm{M}_p^\sharp) \vee [\rho]_{\underset{\sim}{d_c}} \cap \gamma_{\mathbb{M}}(\mathrm{M}_p^\sharp) = \emptyset) \Leftrightarrow ([\rho]_{\sim_{\mathrm{M}_p^\sharp \backslash d_c}} \subseteq \gamma_{\mathbb{M}}(\mathrm{M}_p^\sharp) \vee [\rho]_{\sim_{\mathrm{M}_p^\sharp \backslash d_c}} \cap \gamma_{\mathbb{M}}(\mathrm{M}_p^\sharp) = \emptyset)$. □

Theorem 2. *If the partition function* $\mathbb{P}^\sharp$ *is valid with respect to the cognizance function* $\mathbb{C}^\sharp$*, then every two indistinguishable traces* $\sigma \overset{\mathbb{C}^\sharp}{\sim} \sigma'$ *must belong to the same partition created by* $\mathbb{P}^\sharp$ *at every program point along the execution.*

Formally, $\forall \mathbb{C}^\sharp, \mathbb{P}^\sharp \in \mathbb{L} \mapsto \wp(\mathcal{D}_{\mathbb{M}}^\sharp).\ \mathit{isValid}_{\mathbb{P}}(\mathbb{C}^\sharp, \mathbb{P}^\sharp) \Rightarrow (\forall \sigma, \sigma' \in \mathbb{S}^{*\infty}.\ \sigma \overset{\mathbb{C}^\sharp}{\sim} \sigma' \Rightarrow (\forall i \in [0, |\sigma|).\ \exists \mathit{l} \in \mathbb{L}, \rho, \rho' \in \mathbb{M}.\ \sigma_{[i]} = \langle \mathit{l},\ \rho \rangle \wedge \sigma'_{[i]} = \langle \mathit{l},\ \rho' \rangle \wedge \forall \mathrm{M}_p^\sharp \in \mathbb{P}^\sharp(\mathit{l}).\ \rho \in \gamma_{\mathbb{M}}(\mathrm{M}_p^\sharp) \Leftrightarrow \rho' \in \gamma_{\mathbb{M}}(\mathrm{M}_p^\sharp)))$.

Proof. The contraposition of this theorem states that, suppose $\mathbb{P}^\sharp$ is valid with respect to $\mathbb{C}^\sharp$, if two traces do not belong to the same partition created by $\mathbb{P}^\sharp$ at some program point along the execution, then these two traces cannot be equivalent according to $\mathbb{C}^\sharp$. More formally, $\forall \mathbb{C}^\sharp, \mathbb{P}^\sharp \in \mathbb{L} \mapsto \wp(\mathcal{D}_{\mathbb{M}}^\sharp).\ \mathrm{isValid}_{\mathbb{P}}(\mathbb{C}^\sharp, \mathbb{P}^\sharp) \Rightarrow (\forall \sigma, \sigma' \in \mathbb{S}^{*\infty}.\ (\exists i \in [0, |\sigma|), \mathit{l} \in \mathbb{L}, \rho, \rho' \in \mathbb{M}, \mathrm{M}_p^\sharp \in \mathbb{P}^\sharp(\mathit{l}).\ \sigma_{[i]} = \langle \mathit{l},\ \rho \rangle \wedge \sigma'_{[i]} = \langle \mathit{l},\ \rho' \rangle \wedge \rho \in \gamma_{\mathbb{M}}(\mathrm{M}_p^\sharp) \wedge \rho' \notin \gamma_{\mathbb{M}}(\mathrm{M}_p^\sharp)) \Rightarrow \sigma \overset{\mathbb{C}^\sharp}{\not\sim} \sigma')$.

Here, we prove by contradiction. Assume that there exist two traces $\sigma \overset{\mathbb{C}^\sharp}{\sim} \sigma'$ such that they do not belong to the same partition at some location i, i.e., $\mathrm{M}_p^\sharp \in \mathbb{P}^\sharp(\mathit{l}) \wedge \sigma_{[i]} = \langle \mathit{l},\ \rho \rangle \wedge \sigma'_{[i]} = \langle \mathit{l},\ \rho' \rangle \wedge \rho \in \gamma_{\mathbb{M}}(\mathrm{M}_p^\sharp) \wedge \rho' \notin \gamma_{\mathbb{M}}(\mathrm{M}_p^\sharp)$.

Since $\sigma \overset{\mathbb{C}^\sharp}{\sim} \sigma'$, there must exist some $d_c \in \mathbb{C}^\sharp(\mathit{l})$ such that $\rho \overset{d_c}{\sim} \rho'$. By the definition of $\overset{d_c}{\sim}$, there are two possible cases:

(1) $\rho = \rho'$. In this case, it is impossible to have $\rho \in \gamma_{\mathbb{M}}(\mathrm{M}_p^\sharp) \wedge \rho' \notin \gamma_{\mathbb{M}}(\mathrm{M}_p^\sharp)$, which simply introduces a contradiction.

(2) $\rho \in \gamma_{\mathbb{M}}(d_c) \wedge \rho' \in \gamma_{\mathbb{M}}(d_c) \wedge \forall \chi \in \mathbb{X} \backslash \mathrm{vars}(d_c).\ \rho(\chi) = \rho'(\chi)$. Since $\mathrm{isValid}_{\mathbb{P}}(\mathbb{C}^\sharp, \mathbb{P}^\sharp)$ is true, we know $\mathrm{isValid}_d(d_c, \mathrm{M}_p^\sharp)$ must hold, which further implies $[\rho]_{\underset{\sim}{d_c}} \subseteq \gamma_{\mathbb{M}}(\mathrm{M}_p^\sharp)\ \vee\ [\rho]_{\underset{\sim}{d_c}} \cap \gamma_{\mathbb{M}}(\mathrm{M}_p^\sharp) = \emptyset$ (24). By the assumption $\rho \in \gamma_{\mathbb{M}}(\mathrm{M}_p^\sharp)$ and the fact that $\rho \in [\rho]_{\underset{\sim}{d_c}}$, we know $[\rho]_{\underset{\sim}{d_c}} \cap \gamma_{\mathbb{M}}(\mathrm{M}_p^\sharp) \neq \emptyset$, thus

$[\rho]_{\underset{\sim}{d_c}} \subseteq \gamma_{\mathbb{M}}(\mathrm{M}_p^\sharp)$ must hold. Since $\rho \overset{d_c}{\sim} \rho'$, we have $\rho' \in [\rho]_{\underset{\sim}{d_c}}$, thus $\rho' \in \gamma_{\mathbb{M}}(\mathrm{M}_p^\sharp)$, which contradicts with the assumption $\rho' \notin \gamma_{\mathbb{M}}(\mathrm{M}_p^\sharp)$. □

ACKNOWLEDGMENTS

Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the National Science Foundation.

REFERENCES

[1] Martín Abadi, Anindya Banerjee, Nevin Heintze, and Jon G. Riecke. 1999. A core calculus of dependency. In *POPL*. ACM, 147–160.
[2] Hiralal Agrawal and Joseph Robert Horgan. 1990. Dynamic Program Slicing. In *PLDI*. ACM, 246–256.
[3] Alejandro Aguirre, Gilles Barthe, Marco Gaboardi, Deepak Garg, and Pierre-Yves Strub. 2017. A relational logic for higher-order programs. *Proc. ACM Program. Lang.* 1, ICFP (2017), 21:1–21:29.
[4] Alejandro Aguirre, Gilles Barthe, Marco Gaboardi, Deepak Garg, and Pierre-Yves Strub. 2019. A relational logic for higher-order programs. *J. Funct. Program.* 29 (2019), e16. DOI: https://doi.org/10.1017/S0956796819000145
[5] Roberto Bagnara, Patricia M. Hill, and Enea Zaffanella. 2008. The Parma Polyhedra Library: Toward a complete set of numerical abstractions for the analysis and verification of hardware and software systems. *Sci. Comput. Program.* 72, 1-2 (2008), 3–21. DOI: https://doi.org/10.1016/j.scico.2007.08.001
[6] Roberto Bagnara, Elisa Ricci, Enea Zaffanella, and Patricia M. Hill. 2002. Possibly not closed convex polyhedra and the Parma Polyhedra Library. In *Static Analysis, 9th International Symposium, SAS 2002, Madrid, Spain, September 17–20, 2002, Proceedings (Lecture Notes in Computer Science, Vol. 2477)*, Manuel V. Hermenegildo and Germán Puebla (Eds.). Springer, 213–229. DOI: https://doi.org/10.1007/3-540-45789-5_17
[7] Alexey Bakhirkin, Josh Berdine, and Nir Piterman. 2014. Backward analysis via over-approximate abstraction and under-approximate subtraction. In *Static Analysis - 21st International Symposium, SAS 2014, Munich, Germany, September 11–13, 2014. Proceedings (Lecture Notes in Computer Science, Vol. 8723)*, Markus Müller-Olm and Helmut Seidl (Eds.). Springer, 34–50. DOI: https://doi.org/10.1007/978-3-319-10936-7_3
[8] Thomas Ball, Mayur Naik, and Sriram K. Rajamani. 2003. From symptom to cause: Localizing errors in counterexample traces. In *POPL*. ACM, 97–105.
[9] Adrian Beer, Stephan Heidinger, Uwe Kühne, Florian Leitner-Fischer, and Stefan Leue. 2015. Symbolic causality checking using bounded model checking. In *Model Checking Software - 22nd International Symposium, SPIN 2015, Stellenbosch, South Africa, August 24–26, 2015, Proceedings (Lecture Notes in Computer Science, Vol. 9232)*, Bernd Fischer and Jaco Geldenhuys (Eds.). Springer, 203–221. DOI: https://doi.org/10.1007/978-3-319-23404-5_14
[10] Ilan Beer, Shoham Ben-David, Hana Chockler, Avigail Orni, and Richard J. Trefler. 2012. Explaining counterexamples using causality. *Form. Meth. Syst. Des.* 40, 1 (2012), 20–40.
[11] Bryant Chen, Judea Pearl, and Elias Bareinboim. 2016. Incorporating knowledge into structural equation models using auxiliary variables. In *IJCAI*. IJCAI/AAAI Press, 3577–3583.
[12] James Cheney, Amal Ahmed, and Umut A. Acar. 2011. Provenance as dependency analysis. *Math. Struct. Comput. Sci.* 21, 6 (2011), 1301–1337.
[13] Hana Chockler and Joseph Y. Halpern. 2004. Responsibility and blame: A structural-model approach. *J. Artif. Intell. Res.* 22 (2004), 93–115.
[14] Hana Chockler, Joseph Y. Halpern, and Orna Kupferman. 2008. What causes a system to satisfy a specification? *ACM Trans. Comput. Log.* 9, 3 (2008), 20:1–20:26.
[15] Westland J. Christopher. 2015. *Structural Equation Models, from Paths to Networks*. Springer.
[16] Michael R. Clarkson and Fred B. Schneider. 2008. Hyperproperties. In *CSF*. IEEE Computer Society, 51–65. DOI: https://doi.org/10.1109/CSF.2008.7
[17] Patrick Cousot. 2019. Abstract semantic dependency. In *Static Analysis - 26th International Symposium, SAS 2019, Porto, Portugal, October 8–11, 2019, Proceedings (Lecture Notes in Computer Science, Vol. 11822)*, Bor-Yuh Evan Chang (Ed.). Springer, 389–410. DOI: https://doi.org/10.1007/978-3-030-32304-2_19
[18] Patrick Cousot. 2021. *Principles of Abstract Interpretation*. The MIT Press.
[19] Patrick Cousot and Radhia Cousot. 1976. Static determination of dynamic properties of programs. In *Proceedings of the Second International Symposium on Programming*. Dunod, Paris, France, 106–130.
[20] Patrick Cousot and Radhia Cousot. 1977. Abstract Interpretation: A unified lattice model for static analysis of programs by construction or approximation of fixpoints. In *POPL*. ACM, 238–252.
[21] Patrick Cousot and Radhia Cousot. 1979. Systematic design of program analysis frameworks. In *POPL*. ACM Press, 269–282.

[22] Patrick Cousot, Radhia Cousot, Jérôme Feret, Laurent Mauborgne, Antoine Miné, David Monniaux, and Xavier Rival. 2005. The ASTREÉ analyzer. In *Programming Languages and Systems, 14th European Symposium on Programming,ESOP 2005, Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2005, Edinburgh, UK, April 4–8, 2005, Proceedings (Lecture Notes in Computer Science, Vol. 3444)*, Shmuel Sagiv (Ed.). Springer, 21–30. DOI: https://doi.org/10.1007/978-3-540-31987-0_3

[23] Patrick Cousot, Radhia Cousot, Jérôme Feret, Laurent Mauborgne, Antoine Miné, and Xavier Rival. 2009. Why does Astrée scale up?*Form. Meth. Syst. Des.* 35, 3 (2009), 229–264. DOI: https://doi.org/10.1007/s10703-009-0089-6

[24] Patrick Cousot and Nicolas Halbwachs. 1978. Automatic discovery of linear restraints among variables of a program. In *Conference Record of the Fifth Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*. ACM Press, New York, NY, 84–97.

[25] Chaoqiang Deng and Patrick Cousot. 2019. Responsibility analysis by abstract interpretation. In *Static Analysis - 26th International Symposium, SAS 2019, Porto, Portugal, October 8–11, 2019, Proceedings (Lecture Notes in Computer Science, Vol. 11822)*, Bor-Yuh Evan Chang (Ed.). Springer, 368–388. DOI: https://doi.org/10.1007/978-3-030-32304-2_18

[26] Chaoqiang Deng and Patrick Cousot. 2019. Responsibility analysis by abstract interpretation. Retrieved from http://arxiv.org/abs/1907.08251.

[27] Isil Dillig, Thomas Dillig, and Alex Aiken. 2012. Automated error diagnosis using abductive inference. In *PLDI*. ACM, 181–192.

[28] Henry Frankel. 1976. Harré on causation. *Philos. Sci.* 43, 4 (Dec. 1976), 560–569.

[29] Jonathan Frankle, Sunoo Park, Daniel Shaar, Shafi Goldwasser, and Daniel J. Weitzner. 2018. Practical accountability of secret processes. In *USENIX Security Symposium*. USENIX Association, 657–674.

[30] Roberto Giacobazzi and Isabella Mastroeni. 2018. Abstract non-interference: A unifying framework for weakening information-flow. *ACM Trans. Priv. Secur.* 21, 2 (2018), 9:1–9:31.

[31] Joseph A. Goguen and José Meseguer. 1982. Security policies and security models. In *IEEE Symposium on Security and Privacy*. IEEE Computer Society, 11–20.

[32] Andreas Griesmayer, Stefan Staber, and Roderick Bloem. 2007. Automated fault localization for C programs. *Electr. Notes Theor. Comput. Sci.* 174, 4 (2007), 95–111.

[33] Alex Groce, Sagar Chaki, Daniel Kroening, and Ofer Strichman. 2006. Error explanation with distance metrics. *Int. J. Softw. Tools Technol. Transf.* 8, 3 (2006), 229–247.

[34] Joseph Y. Halpern and Judea Pearl. 2001. Causes and explanations: A structural-model approach: Part 1: Causes. In *UAI*. Morgan Kaufmann, 194–202.

[35] Joseph Y. Halpern and Judea Pearl. 2005. Causes and explanations: A structural-model approach. Part I: Causes. *Brit. J. Philos. Sci.* 56, 4 (2005), 843–887.

[36] Sudheendra Hangal and Monica S. Lam. 2002. Tracking down software bugs using automatic anomaly detection. In *ICSE*. ACM, 291–301.

[37] Thomas A. Henzinger, Anna Lukina, and Christian Schilling. 2019. Outside the Box: Abstraction-Based Monitoring of Neural Networks. *arXiv preprint arXiv:1911.09032* (2019).

[38] David Hume. 1748. *An Enquiry Concerning Human Understanding*. A. Millar, London. Retrieved from http://www.davidhume.org/texts/ehu.html.

[39] Radha Jagadeesan, Alan Jeffrey, Corin Pitcher, and James Riely. 2009. Towards a theory of accountability and audit. In *ESORICS (Lecture Notes in Computer Science, Vol. 5789)*. Springer, 152–167.

[40] Bertrand Jeannet. 2009.*The Interproc Analyzer*. Retrieved from http://pop-art.inrialpes.fr/interproc/interprocweb.cgi.

[41] Bertrand Jeannet and Antoine Miné. 2009. Apron: A library of numerical abstract domains for static analysis. In *Computer-aided Verification, 21st International Conference, CAV 2009, Grenoble, France, June 26–July 2, 2009. Proceedings (Lecture Notes in Computer Science, Vol. 5643)*, Ahmed Bouajjani and Oded Maler (Eds.). Springer, 661–667. DOI: https://doi.org/10.1007/978-3-642-02658-4_52

[42] HoonSang Jin, Kavita Ravi, and Fabio Somenzi. 2002. Fate and free will in error traces. In *TACAS (Lecture Notes in Computer Science, Vol. 2280)*. Springer, 445–459.

[43] Manu Jose and Rupak Majumdar. 2011. Cause clue clauses: Error localization using maximum satisfiability. In *PLDI*. ACM, 437–446.

[44] Bishoksan Kafle, John P. Gallagher, Graeme Gange, Peter Schachte, Harald Søndergaard, and Peter J. Stuckey. 2018. An iterative approach to precondition inference using constrained Horn clauses. *Theor. Pract. Log. Program.* 18, 3–4 (2018), 553–570. DOI: https://doi.org/10.1017/S1471068418000091

[45] Dave King, Trent Jaeger, Somesh Jha, and Sanjit A. Seshia. 2008. Effective blame for information-flow violations. In *SIGSOFT FSE*. ACM, 250–260.

[46] Bogdan Korel and Juergen Rilling. 1998. Dynamic program slicing methods. *Inf. Softw. Technol.* 40, 11–12 (1998), 647–659.

[47] Matthias Kuntz, Florian Leitner-Fischer, and Stefan Leue. 2011. From probabilistic counterexamples via causality to fault trees. In *Computer Safety, Reliability, and Security - 30th International Conference, SAFECOMP 2011, Naples, Italy, September 19–22, 2011. Proceedings (Lecture Notes in Computer Science, Vol. 6894)*, Francesco Flammini, Sandro Bologna, and Valeria Vittorini (Eds.). Springer, 71–84. DOI: https://doi.org/10.1007/978-3-642-24270-0_6

[48] Janusz Laski and William Stanley. 2009. *Program Dependencies*. Springer London, 125–142. DOI: https://doi.org/10.1007/978-1-84882-240-5_6

[49] Florian Leitner-Fischer and Stefan Leue. 2013. Causality checking for complex system models. In *Verification, Model Checking, and Abstract Interpretation, 14th International Conference, VMCAI 2013, Rome, Italy, January 20–22, 2013. Proceedings (Lecture Notes in Computer Science, Vol. 7737)*, Roberto Giacobazzi, Josh Berdine, and Isabella Mastroeni (Eds.). Springer, 248–267. DOI: https://doi.org/10.1007/978-3-642-35873-9_16

[50] Tal Lev-Ami, Mooly Sagiv, Thomas Reps, and Sumit Gulwani. 2007. *Backward Analysis for Inferring Quantified Preconditions*. Tr-2007-12-01, Tel Aviv University.

[51] David Lewis. 1973. Causation. *J. Philos.* 70, 17 (1973), 556–567.

[52] David Lewis. 2013. *Counterfactuals*. John Wiley & Sons.

[53] Laurent Mauborgne and Xavier Rival. 2005. Trace partitioning in abstract interpretation-based static analyzers. In *ESOP (Lecture Notes in Computer Science, Vol. 3444)*. Springer, 5–20.

[54] Peter Menzies. 2017. Counterfactual theories of causation. In *the Stanford Encyclopedia of Philosophy* (winter 2017 ed.), Edward N. Zalta (Ed.). Metaphysics Research Lab, Stanford University.

[55] Antoine Miné. 2012.*The Banal Static Analyzer Prototype*. Retrieved from https://www-apr.lip6.fr/~mine/banal/.

[56] Antoine Miné. 2001. A new numerical abstract domain based on difference-bound matrices. In *Programs as Data Objects, Second Symposium, PADO 2001, Aarhus, Denmark, May 21–23, 2001, Proceedings (Lecture Notes in Computer Science, Vol. 2053)*, Olivier Danvy and Andrzej Filinski (Eds.). Springer, 155–172. DOI: https://doi.org/10.1007/3-540-44978-7_10

[57] Antoine Miné. 2001. The octagon abstract domain. In *Proceedings of the Eighth Working Conference on Reverse Engineering, WCRE'01, Stuttgart, Germany, October 2–5, 2001*, Elizabeth Burd, Peter Aiken, and Rainer Koschke (Eds.). IEEE Computer Society, 310. DOI: https://doi.org/10.1109/WCRE.2001.957836

[58] Antoine Miné. 2006. The octagon abstract domain. *High. Order Symb. Comput.* 19, 1 (2006), 31–100. DOI: https://doi.org/10.1007/s10990-006-8609-1

[59] Antoine Miné. 2006. Symbolic methods to enhance the precision of numerical abstract domains. In *Verification, Model Checking, and Abstract Interpretation, 7th International Conference, VMCAI 2006, Charleston, SC, USA, January 8–10, 2006, Proceedings (Lecture Notes in Computer Science, Vol. 3855)*, E. Allen Emerson and Kedar S. Namjoshi (Eds.). Springer, 348–363. DOI: https://doi.org/10.1007/11609773_23

[60] Antoine Miné. 2012. Inferring sufficient conditions with backward polyhedral under-approximations. *Electron. Notes Theor. Comput. Sci.* 287 (2012), 89–100. DOI: https://doi.org/10.1016/j.entcs.2012.09.009

[61] Antoine Miné. 2014. Backward under-approximations in numeric abstract domains to automatically infer sufficient program conditions. *Sci. Comput. Program.* 93 (2014), 154–182. DOI: https://doi.org/10.1016/j.scico.2013.09.014

[62] Duong Nguyen Que. 2010. *Robust and Generic Abstract Domain for Static Program Analyses: The Polyhedral Case*. Ph.D. Dissertation. Paris, ENMP.

[63] Judea Pearl. 2013. *Causality: Models, Reasoning and Inference* (2nd ed.). Cambridge University Press.

[64] Marco Pistoia, Robert J. Flynn, Larry Koved, and Vugranam C. Sreedhar. 2005. Interprocedural analysis for privileged code placement and tainted variable detection. In *ECOOP (Lecture Notes in Computer Science, Vol. 3586)*. Springer, 362–386.

[65] Dawei Qi, Abhik Roychoudhury, Zhenkai Liang, and Kapil Vaswani. 2009. Darwin: An approach for debugging evolving programs. In *ESEC/SIGSOFT FSE*. ACM, 33–42.

[66] Kavita Ravi and Fabio Somenzi. 2004. Minimal assignments for bounded model checking. In *TACAS (Lecture Notes in Computer Science, Vol. 2988)*. Springer, 31–45.

[67] Manos Renieris and Steven P. Reiss. 2003. Fault localization with nearest neighbor queries. In *ASE*. IEEE Computer Society, 30–39.

[68] Jacques Riguet. 1948. Relations binaires, fermetures, correspondances de Galois. *Bulletin de la S.M.F., Tome 76* (1948), 114–155.

[69] Xavier Rival. 2005. Understanding the origin of alarms in Astrée. In *SAS (Lecture Notes in Computer Science, Vol. 3672)*. Springer, 303–319.

[70] Xavier Rival and Laurent Mauborgne. 2007. The trace partitioning abstract domain. *ACM Trans. Program. Lang. Syst.* 29, 5 (2007), 26.

[71] Xavier Rival and Kwangkeun Yi. 2020. *Introduction to Static Analysis*. The MIT Press.

[72] Alfred Tarski et al. 1955. A lattice-theoretical fixpoint theorem and its applications. *Pacif. J. Math.* 5, 2 (1955), 285–309.

[73] Caterina Urban and Peter Müller. 2018. An abstract interpretation framework for input data usage. In *ESOP (Lecture Notes in Computer Science, Vol. 10801)*. Springer, 683–710.
[74] Mark Weiser. 1981. Program slicing. In *ICSE*. IEEE Computer Society, 439–449.
[75] Mark Weiser. 1984. Program slicing. *IEEE Trans. Softw. Eng.* 10, 4 (1984), 352–357.
[76] Daniel J. Weitzner, Harold Abelson, Tim Berners-Lee, Joan Feigenbaum, James A. Hendler, and Gerald J. Sussman. 2008. Information accountability. *Commun. ACM* 51, 6 (2008), 82–87.

Received March 2021; revised September 2021; accepted September 2021