On the Role of Cognizance in Responsibility

Laura Canaia and Mila Dalla Preda

Dipartimento di Informatica, University of Verona, Italy

Abstract. A novel definition of responsibility analysis based on abstract interpretation has been recently proposed [5,6]. One of the main distinguishing features of this approach with respect to the existing notions of causality in the literature, is the incorporation of the observer's viewpoint. Responsibility considers actions as responsible based on what the observer can understand of the behaviors of interest when analyzing the execution traces and concerning the specific capabilities of the observer. While the other existing approaches to causality implicitly refer to an omniscient observer who knows everything that occurred.
In this paper, we are interested in further investigating the observer's role in identifying the responsible action of a given behavior of interest.

Keywords: Abstract Interpretation · Responsibility Analysis · Program Trace Semantics · Dependency.

1 Introduction

Identifying the responsible actions or the root cause of a program behavior is crucial, particularly for program behaviors that may pose security risks. Consider, for example, static program analysis, model checking, and testing of security properties or safety-critical applications. Once a violation has been identified, it is imperative to determine its cause to guide analysts in updating the software to prevent such breaches in the future. As observed by Deng and Cousot in [5], the notion of responsibility and causality have been extensively studied across various domains such as law, artificial intelligence, statistical and quantum mechanics, biology, and social sciences; still, none of these definitions are entirely pertinent to programming languages. In particular, Deng and Cousot identify three main shortcomings in the existing approaches to causality. First, the existing concepts of causality are based on systems models that do not consider temporal information. At the same time, the order in which actions are performed plays an important role in attributing responsibility. When an action a that ensures a given behavior of interest occurs, then any other successive action should not be classified as responsible for the considered behavior (since it was already ensured by the occurrence of action a). Moreover, existing approaches to causality typically do not require that the action recognized as the cause of a given behavior has to be an action expressing the ability of entities to make choices. Being responsible should imply some free choice; an action should be identified as responsible if it takes a choice leading to the analyzed behavior.

The third drawback of the existing approaches to causality is that responsible actions are identified from the point of view of an omniscient observer who can see every detail of the program execution. Since this is not always the case, it may be interesting to relax this assumption and tune the precision at which the observer can analyze the system's executions to determine the responsible actions.

The notion of responsibility introduced by Deng and Cousot [5,6] addresses these three points. Responsibility is defined as an abstract interpretation [3,4] of program trace semantics and, therefore, considers the order in which actions are executed; it classifies as responsible actions that implement a choice, and it considers the observer's cognizance, namely the observer's capability of analyzing execution traces.

In this work, we try to understand further the implications of taking into account the observer's cognizance when analyzing responsibility with respect to a behavior of interest. In particular, we want to study the role of the observer's cognizance in determining the responsible actions. The cognizance of the observer is modeled as an equivalence relation on program traces that groups together the traces that the observer cannot distinguish. We want to investigate the effects of tuning the cognizance of the observer on the precision of the responsibility analysis, namely, on identifying the actions classified as responsible for a given behavior. In particular, we make the following contribution:

- With respect to [5,6], we weaken the assumptions that the cognizance of the observer has to satisfy. We show how the main results of [5] still hold. Interestingly, this leads to the identification of a novel notion of *free choice with respect to the cognizance of the observer*;
- We identify the class of cognizance functions that behaves as the omniscient observer when establishing the responsibility with respect to a given behavior;
- We investigate how, by tuning the cognizance relation, we may affect the responsibility analysis, namely the set of actions that are classified as responsible for a given behavior.

Since the concrete trace semantics used in the definition of responsibility is generally uncomputable, in [6], the authors present a sound over-approximation for responsibility analysis based on abstract interpretation. In this work, we focus on the definition of responsibility analysis based on the concrete trace semantics, and we start investigating the role of the cognizance of the observer on the precision of the responsibility analysis. In future work, we plan to investigate the effects of our findings on the abstract responsibility analysis.

Structure of the paper: In Section 2, we present the access control example used in [6] to provide an idea of the existing causality approaches and introduce the notion of responsibility analysis. The example will then be used along the paper to explain the obtained results and insights. In Section 3 we recall the formal definition of responsibility analysis by Deng and Cousot [6]. In the following Section 4, we present our main contribution and investigate the effects of tuning

the cognizance of the observer on the precision of the responsibility analysis. The paper ends with a discussion of possible future research directions.

2 Motivating Example

Let us recall the access control example introduced in [5,6] and reported in Fig. 1. The example is then used for briefly describing the difference between the diverse existing notions of causality and the more recent notion of responsibility.

l_1 :	$apv := 1;$	Approval: $apv > 0$ yes, $apv \leq 0$ no
l_2 :	$i_1 := [-1, 2];$	Input: 1st admin
l_3 :	$apv := (i_1 \leq 0)? - 1 : apv;$	
l_4 :	$i_2 := [-1, 2];$	Input: 2nd admin
l_5 :	$apv := (i_2 \leq 0)? - 1 : apv;$	
l_6 :	$typ := [1, 2];$	Input: system setting
l_7 :	$acs := apv \times typ;$	
l_8 :		Access fails when $acs \leq 0$

Fig. 1: Access Control Program Example [5,6]

We can interpret the program depicted in Fig. 1 as governing access to an object. Access to the object is granted if both administrators approve it. At program point l_6, the value assigned to *typ* specifies the type of access: 1 for read-only and 2 for read and write. At program points l_2 and l_4, the variables i_1 and i_2 are randomly assigned integers within the interval $[-1; 2]$, representing decisions made independently by two administrators. Here, 1 and 2 denote access approval, while 0 and -1 denote rejection. At program point l_8, access to the object is permitted only when *acs* has a strictly positive value, indicating approval by both administrators.

Let us reason on the responsibility of the behaviors corresponding to access failure. Namely, we are interested in answering the following question:

What is the cause of $acs \leq 0$ at program point l_8?

Next, we informally present how some of the main existing approaches to causality address this question, including the recently introduced responsibility analysis.

Dependency analysis: In the literature, dependency analysis [1,2,13], taint analysis [11], and program slicing [12,14] are used to identify the fragments of a program that may influence the values of variables at a specified program point. This allows us to focus on the portion of the program that influences the considered behavior (for example, the variable's value at a certain program point). When reasoning with dependencies in the access control example, we

have that the value of *acs* at program point l_8 depends on all the variables in the program. This implies that any program slicing techniques would return the whole program as the slice responsible for access failure, including actions like $apv := 1$, $apv := (i1 \leq 0)? - 1 : apv$ and $acs := apv \times typ$, which are completely deterministic and shall not be treated as responsible entities. Moreover, while it is true that the assignment $typ := [1, 2]$; influences the final value of *acs*, it is only used to specify the kind of access once the access has been guaranteed and it has nothing to do with the decision of providing or denying access to the object. Thus, while sound, slicing techniques are imprecise when reasoning on responsibility since they consider all possible dependencies.

Counterfactual causality: Counterfactual causality [8,9] establishes causation based on the following principle: An event e is deemed a cause of another event e' if and only if, had e not occurred, e' would not have happened. This criterion has the advantage of allowing us to exclude factors that are not decisive. For instance, in the access control example, counterfactual causality would exclude the action $typ := [1; 2]$ since the value of *acs* will be positive or negative regardless of the value of *typ*. However, when the inputs from both administrators are negative or zero, for example, when $i_1 = 0$ and $i_2 = 0$, then none of the input is considered responsible. This is because even if we change the input of one of the administrators, the final value of *acs* would still be 0 because of the other administrator. In this case, no action is identified as responsible for access failure, which is a shortcoming of this approach.

Actual cause: Actual cause [7,10] establishes causation based on the following principle: 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. In the access control example, the actual cause approach does not consider non-decisive factors (i.e., the assignment to *typ*). In the case when the input from both administrators is zero, namely $i_1 = 0$ and $i_2 = 0$, we have that both inputs are considered actual causes of access failure because the access failure counterfactually depends on $i_1 = 0$ (respectively, $i_2 = 0$) under the contingency where the value of i_2 (respectively, i_1) is changed to 1 or 2. However, as for the dependency analysis, the intermediate deterministic events such as $apv = -1$ and $acs = -1$ are still considered actual causes of access failure. Also, in this case, we are sound but still imprecise in identifying the root cause of access failure.

Responsibility: Even if the notion of actual cause improves the previous concepts of dependency and counterfactual causality, it misses the following three essential points in determining responsibility: (1) the temporal ordering of events/actions is not taken into account, (2) there is no requirement for the responsible entity to be free to make choices, and (3) the point of view of the observer is not taken into account and the implicit assumption is to consider an omniscient observer. The notion of responsibility analysis introduced in [5,6] addresses these issues by identifying the responsible actions according to the following principle:

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.

Thus, when considering the access control example, we have that to the cognizance of an omniscient observer, the following actions are identified as responsible when he/she analyzes a given trace:

- when analyzing an execution trace where the assignment to i_1 is such that $(i_1 \leq 0)$, action $i_1 := [-1; 2]$ is the only one to be identified as responsible for the access failure behavior. This is because the fact that the input from the 1st administrator is less or equal to 0 guarantees access failure no matter what the input from the 2nd administrator is.
- when analyzing an execution trace where the assignment to i_1 and the assignment to i_2 are such that $(i_1 > 0 \wedge i_2 \leq 0)$, the action $i_2 := [-1; 2]$ is the only one identified as responsible for the access failure behavior because it is the first action that guarantees access failure.
- when analyzing an execution trace where the assignment to i_1 and the assignment to i_2 are such that $(i_1 > 0 \wedge i_2 > 0)$, we have access success, and no action is recognized as responsible for access failure.

Let us consider a non-omniscient observer, whose cognizance does not distinguish the value of the input from the 1st administrator:

- when analyzing an execution trace where $(i_2 \leq 0)$, the action $i_2 := [-1; 2]$ is the only one 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 administrator inputs a negative value or zero no matter what the input from the 1st admin is.
- when analyzing an execution trace where $(i_2 > 0)$, the non-omniscient observer is uncertain regarding access failure or success; thus, no entity is responsible for the access failure.

In this example, responsibility analysis well identifies the actions responsible for access failure and can, for example, precisely guide the management of permissions or identify the program points to manipulate to gain access to the desired resource.

3 The Deng and Cousot Definition of Responsibility

In this section, we formalise the notion of responsibility as originally introduced by Deng and Cousot [5,6]. To this end we refer to the simple programming language considered in [6], we define its semantics as set of traces of states, we construct a lattice of program behaviors based on trace properties, we recall the definition of the observation function derived from the observer's cognizance, and of the other functions at the basis of the definition of responsibility analysis.

Transition System: Let us denote with Val the set of possible values ranged over by v, with Var the set of program variables. Let $Mem : Var \to Val$ denote the set of memory maps, ranged over by ρ, that associate values to variables. Let Loc be the set of program points ranged over by l, where l^i denotes the initial program point and l^f the final program point. Program states are denoted as $s \in \Sigma$, where $\Sigma = Loc \times Mem \cup \{\omega\}$, where ω denotes the error state. Thus, a program state $s = \langle l, \rho \rangle$ is a pair given by the program point of the next action to be executed and the current memory map. The set of initial states is $\Sigma^i = \{l^i\} \times Mem$, while the set of final states are $\Sigma^f = \{l^f\} \times Mem$. Program actions are denoted as $Action$ and ranged over by a. We denote a transition relation as $\to \in \wp(\Sigma \times Action \times \Sigma)$, such that $\langle s, a, s' \rangle \in \to$ denotes an atomic step from state s to state s' when executing action a (where $s \neq \omega$). A transition system is a pair $(\Sigma^i, \to)$ of initial states and a transition relation defined over states that models the possible state evolution.

3.1 Program Syntax and Semantics

Syntax: We consider a simple programming language whose syntax is given by:

$$\begin{array}{lcl} Aexp & ::= & [a;b] \mid x \mid -e \mid e \; op \; e \mid b?e : e \\ Bexp & ::= & e \; comp \; e \mid b \vee b \mid b \wedge b \mid \neg b \\ Prog & ::= & stmt \\ stmt & ::= & x := e \mid \text{if } (b)\{stmt\} \text{ else } \{stmt\} \mid \text{while } (b)\{stmt\} \mid \{stmt\};\{stmt\} \end{array}$$

where e ranges over the arithmetic expressions $Aexp$ and op denotes any arithmetic operation, b ranges over the boolean expressions $Bexp$ and $comp$ denotes any comparison operation, and programs are given by sequences of statements $stmt$. The arithmetic expression $[a;b]$ denotes a random value in the interval $[a;b]$, and assignments like $x := [a;b]$ model an input value in the considered interval.

Function $[\![e]\!] : Mem \to \wp(Val \cup \{\omega\})$ denotes the semantics of arithmetic expression e, and function $[\![b]\!] : Mem \to \wp(Val \cup \{\omega\})$ denotes the semantics of boolean expression b. The definition of these semantics is standard and we omit it here. We model the behaviors of programs written in the above programming language as transition systems $P = (\Sigma^i, \to)$, where actions are either assignments $x := e$ or boolean tests b. The transition relation that relates states before and after the execution of action a at location l with successive location l' is defined as:

$$\{\langle l, \rho \rangle \to \langle l', \rho' \rangle \mid \rho, \rho' \in Mem, \rho' \in \tau[\![a]\!]\rho\} \cup \{\langle l, \rho \rangle \to \omega \mid \rho \in Mem, \omega \in \tau[\![a]\!]\rho\}$$

where $\tau : Action \times \wp(Mem) \to \wp(\Sigma \cup \{\omega\})$ is the transfer function modeling the execution of actions:

$$\tau[\![x := e]\!]M = \{\rho[x \leftarrow v] \mid \rho \in M, v \in [\![e]\!]\rho\} \cup \{\omega \mid \rho \in M, \omega \in [\![e]\!]\rho\}$$

$$\tau[\![b]\!]M = \{\rho \mid \rho \in M, true \in [\![b]\!]\rho\} \cup \{\omega \mid \rho \in M, \omega \in [\![b]\!]\rho\}$$

Trace Semantics: The semantics of a program $P = (\Sigma^i, \rightarrow)$ is a set of finite or infinite sequence of states called *traces*. Each trace models a program execution. Let Σ^* denote the set of finite or empty traces, Σ^∞ the set of infinite traces and $\Sigma^{*\infty}$ the set of empty, finite and infinite traces. A finite trace of length n is denoted by $\sigma = s_0 \dots s_{n-1}$, while $\sigma = s_0 \dots s_i \dots$ denotes an infinite trace. $\sigma_{[i]}$ refers to the i-th state in the trace σ. We denote with $|\sigma|$ the length of a trace, and the length of the empty trace ε is $|\varepsilon| = 0$, while the length of an infinite trace is ∞. The concatenation of a finite trace σ and a trace σ' is denoted as $\sigma\sigma'$. A trace σ is $\preceq$-less than or equal to another trace σ', if and only if, σ' is a prefix of σ. Formally: $\sigma \preceq \sigma' \stackrel{def}{=} |\sigma| \leq |\sigma'| \wedge \forall 0 \leq i < |\sigma| : \sigma_{[i]} = \sigma'_{[i]}$. We denote with $Pref(\mathcal{T})$ the set of prefixes of traces in $\mathcal{T} \subseteq \Sigma^{*\infty}$. Formally, $Pref \in \wp(\Sigma^{*\infty}) \mapsto \wp(\Sigma^{*\infty})$, where $Pref(\mathcal{T}) \stackrel{def}{=} \{\sigma' \in \Sigma^{+\infty} \mid \exists \sigma \in \mathcal{T}.\sigma' \preceq \sigma\}$.

Given a program $P = (\Sigma^i, \rightarrow)$, the *maximal prefix trace semantics* $[\![P]\!]^{\max} \in \wp(\Sigma^{*\infty})$ is the set of all possible *valid maximal traces*. A valid maximal trace of P starts from an initial state $s \in \Sigma^i$ and it is such that every two successive states along the trace are related by the transition relation $\rightarrow$, and either it terminates at a final state $s \in \Sigma^f$ or at the error state ω, or it does not ever terminate.

$$[\![P]\!]^{\max} \stackrel{def}{=} \{s_0 \dots s_{n-1} \in \Sigma^* \mid s_0 \in \Sigma^i \wedge \forall i \in [0, n-2].s_i \rightarrow s_{i+1} \wedge s_{n-1} \in \Sigma^f \cup \{\omega\}\} \cup \{s_0 \dots s_i \cdots \in \Sigma^\infty \mid s_0 \in \Sigma^i \wedge \forall i \in \mathbb{N}.s_i \rightarrow s_{i+1}\}$$

The *prefix trace semantics* $[\![P]\!]^{Pref} \in \wp(\Sigma^*)$ of P is the set of all possible prefix traces, which is an abstraction of maximal trace semantics via $Pref$, i.e. $[\![P]\!]^{Pref} = Pref([\![P]\!]^{\max})$. A trace is valid if $\sigma \in [\![P]\!]^{Pref}$.

3.2 Formal Definition of Responsibility Analysis

Lattice of trace properties: The behaviors of interest are modeled as *trace properties*, where a trace property is specified as the set of traces that satisfy it, denoted as $\mathcal{T} \in \wp([\![P]\!]^{\max})$. Let $\langle \mathcal{L}^{\max}, \subseteq, \top^{\max}, \bot^{\max}, \overset{\diamond}{\cup}, \overset{\diamond}{\cap} \rangle$ be the complete lattice of maximal trace properties representing the behaviors of interest. Observe that $\mathcal{L}^{\max} \in \wp(\wp([\![P]\!]^{\max}))$ is a set of behaviors of interest, each of which is represented by a maximal trace property, where $\top^{\max} = [\![P]\!]^{\max}$, $\bot^{\max} = \emptyset$, $\subseteq$ is the standard set inclusion operation, $\overset{\diamond}{\cup}$ and $\overset{\diamond}{\cap}$ are join and meet operations, which might not be the standard $\cup$ and $\cap$, since $\mathcal{L}^{\max}$is a subset of $\wp([\![P]\!]^{\max})$ but not necessarily a sublattice. Given a program P, the lattice $\mathcal{L}^{\max} \in \wp(\wp([\![P]\!]^{\max}))$ specifies the properties for which we want to analyze the responsibility.

Example: In the access control example, we can identify the following behaviors of interest that form the lattice depicted in Fig 2:

- Access Success: $AS = \{\sigma \in [\![P]\!]^{\max} \mid \exists \rho \in Mem : \sigma_{[7]} = \langle l_8, \rho \rangle \wedge \rho(acs) > 0\}$,
- Access Failure: $AF = \{\sigma \in [\![P]\!]^{\max} \mid \exists \rho \in Mem : \sigma_{[7]} = \langle l_8, \rho \rangle \wedge \rho(acs) \leq 0\}$,
- Read Only: $RO = \{\sigma \in [\![P]\!]^{\max} \mid \exists \rho \in Mem : \sigma_{[7]} = \langle l_8, \rho \rangle \wedge \rho(acs) = 1\}$,
- Read Write: $RW = \{\sigma \in [\![P]\!]^{\max} \mid \exists \rho \in Mem : \sigma_{[7]} = \langle l_8, \rho \rangle \wedge \rho(acs) = 2\}$

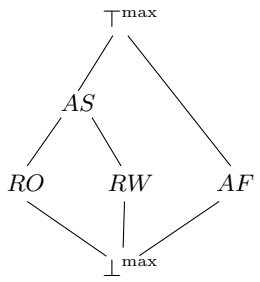

Fig. 2: Lattice of system behaviors

Prediction Abstraction: Given a maximal trace property $\mathcal{X} \in \mathcal{L}^{\max}$, the prediction abstraction $\alpha_{Pred}(\llbracket P \rrbracket^{\max}) \in \wp(\Sigma^{*\infty}) \mapsto \wp(\Sigma^{*\infty})$ returns the prefixes of maximal traces in $\mathcal{X}$ whose prolongation satisfy the property $\mathcal{X}$:

$$\alpha_{Pred}(\llbracket P \rrbracket^{\max})\mathcal{X} \stackrel{def}{=} \{\sigma \in Pref(\mathcal{X}) \mid \forall \sigma' \in \llbracket P \rrbracket^{\max}.\sigma \preceq \sigma' \ \Rightarrow \ \sigma' \in \mathcal{X}\}$$

and the corresponding concretization $\gamma_{Pred}(\llbracket P \rrbracket^{\max}) \in \wp(\Sigma^{*\infty}) \mapsto \wp(\Sigma^{*\infty})$ is:

$$\gamma_{Pred}(\llbracket P \rrbracket^{\max})\mathcal{Y} \stackrel{def}{=} \{\sigma \in \mathcal{Y} \mid \sigma \in \llbracket P \rrbracket^{\max}\} = \mathcal{Y} \cap \llbracket P \rrbracket^{\max}$$

The prediction abstraction highlights the point along the maximal trace from which a property is guaranteed to hold later in the execution. Indeed, every valid maximal trace σ' that is greater than or equal to a prefix trace σ in $\alpha_{Pred}(\llbracket P \rrbracket^{\max})\mathcal{X}$ is guaranteed to have the maximal trace property $\mathcal{X}$. Observe that in general $\alpha_{Pred}(\llbracket P \rrbracket^{\max})\mathcal{X}$ is not prefix-closed, and

$$\alpha_{Pred}(\llbracket P \rrbracket^{\max})\mathcal{X} = \mathcal{X} \cup \{\sigma \in Pref(\mathcal{X}) \mid \forall \sigma' \in \llbracket P \rrbracket^{\max}.\sigma \preceq \sigma' \ \Rightarrow \ \sigma' \in \mathcal{X}\}$$

thus $\alpha_{Pred}(\llbracket P \rrbracket^{\max})\mathcal{X}$ returns the traces of P that satisfy the property $\mathcal{X}$ enriched by all those prefixes of these traces that ensure that property $\mathcal{X}$ holds along any continuation in $\llbracket P \rrbracket^{\max}$. This is just an equivalent way of expressing property $\mathcal{X}$ in order to highlight from where $\mathcal{X}$ is ensured to hold. Indeed, we have a Galois isomorphism between maximal trace properties and prediction trace properties.

Example: When considering the access control example we have that:

$$\alpha_{Pred}(\llbracket P \rrbracket^{\max})AF = \left\{ \sigma \in \llbracket P \rrbracket^{Pref} \;\middle|\; \begin{array}{l} \exists \rho_1 \in Mem, v \in \{-1, 0\}, v' \in \{1, 2\}. \\ \langle l_1, \rho_1 \rangle \langle l_2, \rho_2 = \rho_1[apv \leftarrow 1] \rangle \langle l_3, \rho_3 = \rho_1[i_1 \leftarrow v] \rangle \preceq \sigma \\ \vee \\ \langle l_1, \rho_1 \rangle \langle l_2, \rho_2 = \rho_1[apv \leftarrow 1] \rangle \langle l_3, \rho_3 = \rho_1[i_1 \leftarrow v'] \rangle \\ \langle l_4, \rho_4 = \rho_3 \rangle \langle l_5, \rho_5 = \rho_4[i_2 \leftarrow v] \rangle \preceq \sigma \end{array} \right\}$$

which precisely captures the fact that access failure is guaranteed when the input from the 1st administrator is negative or zero, or when the input from the 1st

administrator is positive and the input from the 2nd administrator is negative or zero. Analogously, we have that access success is guaranteed when both inputs from the two administrators are positive:

$$\alpha_{Pred}(\llbracket P \rrbracket^{\max})AS = \left\{ \sigma \in \llbracket P \rrbracket^{Pref} \;\middle|\; \begin{array}{l} \exists \rho_1 \in Mem, v, v' \in \{1,2\}. \\ \langle l_1, \rho_1 \rangle \langle l_2, \rho_2 = \rho_1[apv \leftarrow 1] \rangle \langle l_3, \rho_3 = \rho_1[i_1 \leftarrow v] \rangle \\ \langle l_4, \rho_4 = \rho_3 \rangle \langle l_5, \rho_5 = \rho_4[i_2 \leftarrow v'] \rangle \preceq \sigma \end{array} \right\}$$

Inquiry Function: The inquiry function models what an observer can learn by observing a prefix trace of P as regarding the properties of interest expressed by $\mathcal{L}^{\max}$. More specifically, the inquiry function $\mathbb{I}$ maps every trace $\sigma \in \Sigma^{*\infty}$ to the strongest maximal trace property in $\mathcal{L}^{\max}$ that σ can guarantee. Formally we have that:

$$\mathbb{I} \in \wp(\Sigma^{*\infty}) \mapsto \wp(\wp(\Sigma^{*\infty})) \mapsto \Sigma^{*\infty} \mapsto \wp(\Sigma^{*\infty})$$
$$\mathbb{I}(\llbracket P \rrbracket^{\max}, \mathcal{L}^{\max}, \sigma) \stackrel{def}{=} \overset{\diamond}{\bigcap} \{\mathcal{T} \in \mathcal{L}^{\max} \mid \sigma \in \alpha_{Pred}(\llbracket P \rrbracket^{\max})\mathcal{T}\}$$

Note that, when $\mathbb{I}(\llbracket P \rrbracket^{\max}, \mathcal{L}^{\max}, \sigma) = \mathcal{B}$ then σ guarantees the satisfaction of $\mathcal{B}$, namely for any trace $\sigma' \in \llbracket P \rrbracket^{\max}$ such that $\sigma \preceq \sigma'$ it holds that $\sigma' \in \mathcal{B}$. Moreover, we can observe that the inquiry function cannot learn from invalid traces namely for any invalid trace $\sigma \notin \llbracket P \rrbracket^{Pref}$, we have that $\mathbb{I}(\llbracket P \rrbracket^{\max}, \mathcal{L}^{\max}, \sigma) = \bot^{\max}$. Another interesting property of the inquiry function, is that the longer σ is, the stronger is the property that the inquiry function can guarantee.

Example: When considering the access control example we have that: the prefix trace that terminates before the input of the 1st administrator cannot guarantee any interesting property in $\mathcal{L}^{\max}$:

$$\mathbb{I}(\llbracket P \rrbracket^{\max}, \mathcal{L}^{\max}, \langle l_1, \rho_1 \rangle \langle l_2, \rho_2 = \rho_1[apv \leftarrow 1] \rangle) = \top^{\max}$$

While the prefix trace that terminates after the input from the 1st administrator when this input is negative or zero guarantees access failure:

$$\mathbb{I}(\llbracket P \rrbracket^{\max}, \mathcal{L}^{\max}, \langle l_1, \rho_1 \rangle \langle l_2, \rho_2 = \rho_1[apv \leftarrow 1] \rangle \langle l_3, \rho_3 = \rho_2[i_1 \leftarrow 0] \rangle) = AF$$

The prefix trace that terminates after the input from the 1st administrator when this input is positive cannot conclude anything regarding access success or failure:

$$\mathbb{I}(\llbracket P \rrbracket^{\max}, \mathcal{L}^{\max}, \langle l_1, \rho_1 \rangle \langle l_2, \rho_2 = \rho_1[apv \leftarrow 1] \rangle \langle l_3, \rho_3 = \rho_2[i_1 \leftarrow 1] \rangle) = \top^{\max}$$

The prefix trace that terminates after the input from the 2nd administrator when $i_1 > 0$ and $i_2 \leq 0$ guarantees access failure:

$$\mathbb{I}(\llbracket P \rrbracket^{\max}, \mathcal{L}^{\max}, \langle l_1, \rho_1 \rangle \langle l_2, \rho_2 = \rho_1[apv \leftarrow 1] \rangle \langle l_3, \rho_3 = \rho_2[i_1 \leftarrow 1] \rangle \langle l_4, \rho_4 = \rho_3 \rangle, \langle l_5, \rho_5 = \rho_4[i_2 \leftarrow -1] \rangle) = AF$$

To conclude the prefix trace that terminates after the input from the 2nd administrator when $i_1 > 0$ and $i_2 > 0$ guarantees access success:

$$\mathbb{I}(\llbracket P \rrbracket^{\max}, \mathcal{L}^{\max}, \langle l_1, \rho_1 \rangle \langle l_2, \rho_2 = \rho_1[apv \leftarrow 1] \rangle \langle l_3, \rho_3 = \rho_2[i_1 \leftarrow 1] \rangle \langle l_4, \rho_4 = \rho_3 \rangle, \langle l_5, \rho_5 = \rho_4[i_2 \leftarrow 1] \rangle) = AS$$

Cognizance Function: The observer point of view, namely what the observer can learn from the analysis of program executions, is modeled by the *cognizance relation* $\mathbb{C} \in \Sigma^{*\infty} \mapsto \wp(\Sigma^{*\infty})$ that groups together the traces that the observer cannot distinguish.

$$\mathbb{C}(\sigma) = [\sigma]_{\mathbb{C}} \stackrel{def}{=} \{\sigma' \in \Sigma^{*\infty} \mid \text{ the observer cannot distinguish } \sigma \text{ from } \sigma'\}$$

Where $[\sigma]_{\mathbb{C}}$ and $\mathbb{C}(\sigma)$ denote the equivalence class of the traces equivalent to σ with respect to $\mathbb{C}$. We write $\mathbb{C}_1 \sqsubseteq \mathbb{C}_2$ when the equivalence relation $\mathbb{C}_1$ is a refinement of the equivalence relation $\mathbb{C}_2$, namely when $\forall \sigma \in \Sigma^{*\infty} : [\sigma]_{\mathbb{C}_1} \subseteq [\sigma]_{\mathbb{C}_2}$, or equivalently $\mathbb{C}_1(\sigma) \subseteq \mathbb{C}_2(\sigma)$. We denote with $\mathbb{C}_o$ what we call the omniscient observer, namely the observer that distinguishes every execution trace: $\forall \sigma \in \Sigma^{*\infty} : [\sigma]_{\mathbb{C}_o} = \mathbb{C}_o(\sigma) = \{\sigma\}$, which means that no ambiguity is inserted by the observer. In [6] the authors make the following assumptions on the cognizance of the observer:

(A1) The cognizance of a trace $\sigma\sigma'$ is the concatenation of cognizance of σ and σ', i.e, $\forall \sigma, \sigma' \in \Sigma^{*\infty}.\mathbb{C}(\sigma\sigma') = \{\pi\pi' \mid \pi \in \mathbb{C}(\sigma) \wedge \pi' \in \mathbb{C}(\sigma')\}$, and $\mathbb{C}(\epsilon) = \{\epsilon\}$.

(A2) Given an invalid trace, the cognizance function would not return a valid trace, i.e., $\forall \sigma \in \Sigma^{*\infty}.\sigma \notin \llbracket P \rrbracket^{Pref} \Rightarrow \mathbb{C}(\sigma) \cap \llbracket P \rrbracket^{Pref} = \emptyset$.

Example: If we consider a non-omniscient observer $\mathbb{C}$ that is unaware of the input from the 1st administrator, we have that he/she is going to group together the prefixes with the different values for the input from the first administrator:

$\mathbb{C}(\langle l_1, \rho_1 \rangle \langle l_2, \rho_2 = \rho_1[apv \leftarrow 1] \rangle \langle l_3, \rho_3 = \rho_2[i_1 \leftarrow 1] \rangle) =$
$\{\langle l_1, \rho_1 \rangle \langle l_2, \rho_2 = \rho_1[apv \leftarrow 1] \rangle \langle l_3, \rho_3 = \rho_2[i_1 \leftarrow v] \rangle \mid v \in \{-1, 0, 1, 2\}\}$

Observation Function: The idea is that the observation function formalizes what an observer can learn from a trace by taking into account its cognizance. As expected, given a trace σ the observation function applies the inquiry function on each trace in $\mathbb{C}(\sigma)$, and then returns the join of the set of maximal trace properties obtained. Formally, we have that:

$$\mathbb{O} \in \wp(\Sigma^{*\infty}) \mapsto \wp(\wp(\Sigma^{*\infty})) \mapsto (\Sigma^{*\infty} \mapsto \wp(\Sigma^{*\infty})) \mapsto \Sigma^{*\infty} \mapsto \wp(\Sigma^{*\infty})$$
$$\mathbb{O}(\llbracket P \rrbracket^{\max}, \mathcal{L}^{\max}, \mathbb{C}, \sigma) \stackrel{def}{=} \overset{\diamond}{\cup} \{\mathbb{I}(\llbracket P \rrbracket^{\max}, \mathcal{L}^{\max}, \sigma') \mid \sigma' \in \mathbb{C}(\sigma)\}$$

Note that, when considering the omniscient observer the observation function corresponds to the inquiry function. As for the inquiry function we have that the observation function cannot learn from an invalid trace. Indeed, for any $\sigma \notin \llbracket P \rrbracket^{Pref}$, we have that $\mathbb{O}(\llbracket P \rrbracket^{\max}, \mathcal{L}^{\max}, \mathbb{C}, \sigma) = \bot^{\max}$, since every trace $\sigma' \in \mathbb{C}(\sigma)$ is invalid by (A2). Moreover, the longer σ is, the stronger is the property that the observation function can return.

Example: Consider an observer with cognizance $\mathbb{C}$ that is not aware of the input from the 1st administrator. In this case, even if the 1st administrator inputs a negative value the observer cannot be sure of the access failure behavior:

$$\mathbb{O}([\![P]\!]^{\max}, \mathcal{L}^{\max}, \mathbb{C}, \langle l_1, \rho_1\rangle\langle l_2, \rho_2 = \rho_1[apv \leftarrow 1]\rangle\langle l_3, \rho_3 = \rho_2[i_1 \leftarrow -1]\rangle) = \overset{\diamond}{\bigcup} \{\mathbb{I}([\![P]\!]^{\max}, \mathcal{L}^{\max}, \langle l_1, \rho_1\rangle\langle l_2, \rho_2 = \rho_1[apv \leftarrow 1]\rangle\langle l_3, \rho_3 = \rho_2[i_1 \leftarrow v]\rangle \mid v \in \{-1, 0, 1, 2\}\} = AF \overset{\diamond}{\cup} AF \overset{\diamond}{\cup} \top^{\max} \overset{\diamond}{\cup} \top^{\max} = \top^{\max}$$

while the omniscient observer when considering the same trace can conclude that it will exhibit an access failure behavior:

$$\mathbb{O}([\![P]\!]^{\max}, \mathcal{L}^{\max}, \mathbb{C}_o, \langle l_1, \rho_1\rangle\langle l_2, \rho_2 = \rho_1[apv \leftarrow 1]\rangle\langle l_3, \rho_3 = \rho_2[i_1 \leftarrow -1]\rangle) = AF.$$

Responsibility: We can now formally define the responsibility abstraction:

$$\alpha_R \in \wp(\Sigma^{*\infty}) \mapsto \wp(\wp(\Sigma^{*\infty})) \mapsto (\Sigma^{*\infty} \mapsto \wp(\Sigma^{*\infty})) \mapsto \wp(\Sigma^{*\infty}) \mapsto \wp(\Sigma^{*\infty}) \mapsto (\Sigma^* \times (\Sigma \times \Sigma) \times \Sigma^{*\infty})$$

$$\alpha_R([\![P]\!]^{\max}, \mathcal{L}^{\max}, \mathbb{C}, \mathcal{B}, \mathcal{T}) \overset{def}{=} \left\{ \langle \sigma_H, \tau_R, \sigma_F\rangle \;\middle|\; \begin{array}{l} \sigma_H\tau_R\sigma_F \in \mathcal{T} \\ \wedge \emptyset \subset \mathbb{O}([\![P]\!]^{\max}, \mathcal{L}^{\max}, \mathbb{C}, \sigma_H\tau_R) \subseteq \mathcal{B} \\ \wedge \mathbb{O}([\![P]\!]^{\max}, \mathcal{L}^{\max}, \mathbb{C}, \sigma_H) \not\subseteq \mathcal{B} \end{array} \right\}$$

Where $\mathcal{B} \in \mathcal{L}^{\max}$ is the behavior whose responsibility is of interest, and $\mathcal{T} \in \mathcal{L}^{\max}$ is the set of traces that we want to analyze in order to identify the actions that are responsible for behavior $\mathcal{B}$. Every trace $\sigma \in \mathcal{T}$ is split into three parts $\sigma = \sigma_H\tau_R\sigma_F$, where $\sigma_H = s_0 \dots s_{r-1} \in \Sigma^*$ represents the history, the transition $\tau_R = s_{r-1} \overset{a_R}{\rightarrow} s_r$ represents the action *responsible* of behavior $\mathcal{B}$, and $\sigma_F = s_r \cdots \in \Sigma^{*\infty}$ represents the future. The responsibility abstraction requires that σ_H does not guarantee $\mathcal{B}$ by saying that $\mathbb{O}([\![P]\!]^{\max}, \mathcal{L}^{\max}, \mathbb{C}, \sigma_H) \not\subseteq \mathcal{B}$, while $\sigma_H\tau_R$ guarantees a behavior that is at least as strong as $\mathcal{B}$ and it is not $\bot^{\max}$. Therefore, to the cognizance $\mathbb{C}$ of a given observer, the transition $\tau_R = s_{r-1} \overset{a_R}{\rightarrow} s_r$ (or, say, the action a_R) is said to be responsible for the behavior $\mathcal{B}$ in the trace $\sigma_H\tau_R\sigma_F$, since it is the first action that makes the observer sure of the fact that behavior $\mathcal{B}$ will be guaranteed by any continuation of the trace. Since $\alpha_R([\![P]\!]^{\max}, \mathcal{L}^{\max}, \mathbb{C}, \mathcal{B})$ preserves joins on analyzed traces $\mathcal{T}$, we have a Galois connection.

It is possible to prove that for a cognizance that satisfies assumptions A1 and A2, if τ_R is recognized as responsible for a behavior $\mathcal{B}$ in a valid trace $\sigma_H\tau_R\sigma_F$, then $\sigma_H\tau_R$ guarantees the occurrence of behavior $\mathcal{B}$, and there must exist another valid prefix trace $\sigma_H\tau'_R$ such that the behavior $\mathcal{B}$ is not guaranteed. This means that a responsible action must correspond to an action that expresses a free choice, for which there exists another possible continuation τ'_R that leads to a trace that does not satisfy $\mathcal{B}$. Moreover, the so defined notion of responsibility

abstraction takes into account both the temporal ordering of actions and the observer's cognizance as shown in the example below. This makes it clear how this novel notion of responsibility improves with respect to the causality notions present in the literature.

Example: Considering the access control example and the omniscient observer that analyzes all the possible traces. We have that the actions responsible for AF are $i_1 := [-1; 2]$ for those traces where $i_1 \leq 0$ and $i_2 := [-1; 2]$ for those traces where $i_1 > 0$ and $i_2 \leq 0$; while the action responsible for AS is $i_2 := [-1; 2]$ for those traces where $i_1 > 0$ and $i_2 > 0$, since we need to observe both inputs in order to be sure of access success. While, if we consider a non-omniscient observer that cannot see the value of the input from the 1st administrator, nor the value assumed by *apv* at l_3 we have that the only responsible action for AF is $i_2 := [-1; 2]$ when $i_2 \leq 0$, and no action is responsible for AS.

4 Main Contribution: The Role of Cognizance in Responsibility

In this section, we investigate the relation between the cognizance of the observer and the capability of the observer in identifying the responsible actions.

4.1 Relaxing Assumption A2 on Cognizance

First of all we observe that assumption A2 on the cognizance of the observer, means that the cognizance relation cannot confuse valid and invalid traces as shown by the following result (proof in the Appendix).

Lemma 1. *Assumption A2 implies that* $\sigma \in [\![P]\!]^{Pref} \Rightarrow \mathbb{C}(\sigma) \subseteq [\![P]\!]^{Pref}$.

Assumption A2, together with assumption A1 that defines the equivalence relation $\mathbb{C}$ on traces as actually an equivalence relation on states may be restrictive. For example, a cognizance $\mathbb{C}$ that does not observe the input value from the 1st administrator and that could observe the value assigned to *apv* at program point l_3 does not satisfy the two assumptions above. In fact, since the considered $\mathbb{C}$ does not distinguish the value of i_1 we have that, when the observer considers a prefix trace that terminates after the input 0 from the 1st administrator, the cognizance does not distinguish the input value and sees every possible value of i_1 as possible:

$$\mathbb{C}(\langle l_1, \rho_1\rangle\langle l_2, \rho_2 = \rho_1[apv \leftarrow 1]\rangle\langle l_3, \rho_3 = \rho_2[i_1 \leftarrow 0]\rangle) =$$

$$\{\langle l_1, \rho_1\rangle\langle l_2, \rho_2 = \rho_1[apv \leftarrow 1]\rangle\langle l_3, \rho_3 = \rho_2[i_1 \leftarrow v]\rangle \mid v \in \{-1, 0, 1, 2\}\}$$

Moreover, the considered cognizance $\mathbb{C}$ precisely sees the values assigned to *apv* at program point l_3, therefore:

$$\mathbb{C}(\langle l_4, \rho_4 = \rho_3[apv \leftarrow -1]\rangle) = \langle l_4, \rho_4 = \rho_3[apv \leftarrow -1]\rangle$$

Thus, by combining the two traces according to assumption A1 we have that:

$$\mathbb{C}(\langle l_1, \rho_1\rangle\langle l_2, \rho_2 = \rho_1[apv \leftarrow 1]\rangle\langle l_3, \rho_3 = \rho_2[i_1 \leftarrow 0]\rangle\langle l_4, \rho_4 = \rho_3[apv \leftarrow -1]\rangle) =$$

$$\{\langle l_1, \rho_1\rangle\langle l_2, \rho_2 = \rho_1[apv \leftarrow 1]\rangle\langle l_3, \rho_3 = \rho_2[i_1 \leftarrow v]\rangle\langle l_4, \rho_4 = \rho_3[apv \leftarrow -1]\rangle \mid v \in \{-1, 0, 1, 2\}\}$$

that includes also the two invalid traces:

$$\{\langle l_1, \rho_1\rangle\langle l_2, \rho_2 = \rho_1[apv \leftarrow 1]\rangle\langle l_3, \rho_3 = \rho_2[i_1 \leftarrow v]\rangle\langle l_4, \rho_4 = \rho_3[apv \leftarrow -1]\rangle \mid v \in \{1, 2\}\}$$

One may argue that an observer who does not know the value assigned to i_1 at program point l_2 should not be able to know the exact value of apv at program point l_3. But this means that we imagine that the observer's uncertainty in distinguishing the exact value of i_1 is propagated along the computation, and we do not allow the observer to see the effects of the specific value assumed by i_1 later on in the execution. While it may be reasonable, for example, for the observer to realize that the access has been denied at program point l_8 even if the input from the 2nd administrator is positive. We can imagine an observer that can distinguish some aspects of the real computation but not all. Such an observer should be modeled with cognizance that satisfies assumption A1 but not necessarily assumption A2. Therefore, in the rest of this section, we relax the assumption $A2$ and allow the observer to confuse valid and invalid traces. We believe that most of the results regarding the concrete responsibility analysis in [6] are still valid since when a trace is invalid, nothing can be learned from the inquiry function, and therefore, the invalid trace is somehow discarded. Moreover, the main result proved in Theorem 1 of [6] regarding the concrete responsibility analysis is still valid with a minimum adjustment that introduces the interesting concept of free choice with respect to an observer, as proved by the following result (proof in the Appendix).

Theorem 1. *For the cognizance $\mathbb{C}$ of an observer, if τ_R is responsible for a behavior $\mathcal{B}$ in a valid trace $\sigma_H\tau_R\sigma_F$, then:*

1. *$\sigma_H\tau_R$ guarantees the occurrence of the behavior $\mathcal{B}$,*
2. *there exists a valid trace $\sigma's \in [\![P]\!]^{Pref}$ such that $\sigma' \in \mathbb{C}(\sigma_H)$ and $\sigma's$ does not guarantee the behavior $\mathcal{B}$.*

Example: Let us consider the access control example, and assume that we are analyzing all the prefix traces in $[\![P]\!]^{Pref}$ with respect to the cognizance $\mathbb{C}$ of the observer. Assume that $\mathbb{C}$ does not distinguish the input value from the 1st administrator, while $\mathbb{C}$ can observe the value assigned to apv at program point l_3 we have that:

– We have one action that is responsible for AS, and it is $i_2 := [-1, 2]$ when i_2 assumes a positive value and apv at program point l_3 has been assigned 1;

– We have two actions that are classified as responsible of AF. The first one is $apv := (i_1 \leq 0)? - 1 : apv$ when apv is assigned value -1. Even if we do not know the value assumed by i_1 we can observe the effects that it has on this assignment. The second responsible action is $i_2 := [-1; 2]$ when it assumes value -1 or 0 and apv at program point l_3 has been assigned to 1.

We can see that in this example the action $apv := (i_1 \leq 0)? - 1 : apv$ is the first point in the program execution where the observer can see the effects of the value assumed by the input i_1 at location l_2. Even if the conditional assignment at l_3 does not correspond to a free choice it is a *free choice with respect to the cognizance of the observer* that becomes here aware of the effects of the input i_1. We believe that this notion of being a free choice with respect to the cognizance of the observer is more general and has implications that should be better investigated.

4.2 The Observer's Cognizance and the Precision of Responsibility Analysis

Let us try to better understand when the observer of the attacker does not influence the precision of the responsibility analysis. In [6] it is noted that when considering the omniscient observer there is no loss of precision in identifying the responsible actions between the observation function and the inquiry function. Let us see how this actually holds for a larger class of cognizance relations.

We define the equivalence relation $\equiv_{\mathbb{I}} \in \Sigma^{*\infty} \mapsto \wp(\Sigma^{*\infty})$ induced by the inquiry function $\mathbb{I}$, where the equivalence class of a trace $\sigma \in [\![P]\!]^{Pref}$ is defined as:

$$[\sigma]_{\mathbb{I}} = \{\sigma' \in [\![P]\!]^{Pref} \mid \mathbb{I}([\![P]\!]^{\max}, \mathcal{L}^{\max}, \sigma') = \mathbb{I}([\![P]\!]^{\max}, \mathcal{L}^{\max}, \sigma)\}$$

Namely the equivalence class of a trace σ with respect to the equivalence relation $\equiv_{\mathbb{I}}$ is given by all those traces from which the inquiry function can guarantee the property in $\mathcal{L}^{\max}$ that can be guaranteed by observing σ. Thus, $\equiv_{\mathbb{I}}$ groups together all those traces from which the inquiry function can ensure the same behavior in the lattice $\mathcal{L}^{\max}$. Namely $\equiv_{\mathbb{I}}$ groups together all those traces that are equivalent with respect to $\mathbb{I}$. This allows us to generalize what observed in [6] and prove that the observation function and the inquiry function coincide whenever the equivalence relation representing the cognizance of the observer is at least as precise as the equivalence relation induced by the inquiry function, as stated by the following result.

Lemma 2. *If* $\mathbb{C} \sqsubseteq \equiv_{\mathbb{I}}$ *then* $\mathbb{I}([\![P]\!]^{\max}, \mathcal{L}^{\max}, \sigma) = \mathbb{O}([\![P]\!]^{\max}, \mathcal{L}^{\max}, \mathbb{C}, \sigma)$

Thus, any cognizance $\mathbb{C}$ that is at least as precise as $\equiv_{\mathbb{I}}$, namely such that $\mathbb{C} \sqsubseteq \equiv_{\mathbb{I}}$, does not influence the responsibility analysis and behaves as the omniscient observer $\mathbb{C}_o$ in terms of the actions that it recognizes as responsible. This because when $\mathbb{C} \sqsubseteq \equiv_{\mathbb{I}}$ it means that the observer can precisely compute the inquiry function and therefore it is able to recognize as responsible exactly the actions that are classified as responsible by the omniscient observer.

Corollary 1. *If* $\mathbb{C} \sqsubseteq \equiv_{\mathbb{I}}$ *then*

$$\alpha_R([\![P]\!]^{\max}, \mathcal{L}^{\max}, \mathbb{C}, \mathcal{B}, \mathcal{T}) = \alpha_R([\![P]\!]^{\max}, \mathcal{L}^{\max}, \mathbb{C}_o, \mathcal{B}, \mathcal{T})$$

Example: In the access control example consider an attacker that does not distinguish between the values -1 and 0, and between the values 1 and 2 as possible inputs from the 1st administrator. So we have:

$$\mathbb{C}(\langle l_1, \rho_1\rangle\langle l_2, \rho_2 = \rho_1[apv \leftarrow 1]\rangle\langle l_3, \rho_3 = \rho_2[i_1 \leftarrow -1]\rangle) =$$
$$\{\langle l_1, \rho_1\rangle\langle l_2, \rho_2 = \rho_1[apv \leftarrow 1]\rangle\langle l_3, \rho_3 = \rho_2[i_1 \leftarrow v]\rangle \mid v \in \{-1, 0\}\}$$
$$\mathbb{C}(\langle l_1, \rho_1\rangle\langle l_2, \rho_2 = \rho_1[apv \leftarrow 1]\rangle\langle l_3, \rho_3 = \rho_2[i_1 \leftarrow 1]\rangle) =$$
$$\{\langle l_1, \rho_1\rangle\langle l_2, \rho_2 = \rho_1[apv \leftarrow 1]\rangle\langle l_3, \rho_3 = \rho_2[i_1 \leftarrow v]\rangle \mid v \in \{1, 2\}\}$$

of course this cognizance is in accordance with what can be learned by the inquiry function. Indeed, we have that

$$\mathbb{I}([\![P]\!]^{\max}, \mathcal{L}^{\max}, \langle l_1, \rho_1\rangle\langle l_2, \rho_2 = \rho_1[apv \leftarrow 1]\rangle\langle l_3, \rho_3 = \rho_2[i_1 \leftarrow -1]\rangle) =$$
$$\mathbb{I}([\![P]\!]^{\max}, \mathcal{L}^{\max}, \langle l_1, \rho_1\rangle\langle l_2, \rho_2 = \rho_1[apv \leftarrow 1]\rangle\langle l_3, \rho_3 = \rho_2[i_1 \leftarrow 0]\rangle) = AF$$

and

$$\mathbb{I}([\![P]\!]^{\max}, \mathcal{L}^{\max}, \langle l_1, \rho_1\rangle\langle l_2, \rho_2 = \rho_1[apv \leftarrow 1]\rangle\langle l_3, \rho_3 = \rho_2[i_1 \leftarrow 1]\rangle) =$$
$$\mathbb{I}([\![P]\!]^{\max}, \mathcal{L}^{\max}, \langle l_1, \rho_1\rangle\langle l_2, \rho_2 = \rho_1[apv \leftarrow 1]\rangle\langle l_3, \rho_3 = \rho_2[i_1 \leftarrow 2]\rangle) = \top^{\max}$$

And we have that what the attacker can conclude when the input from the 1st administrator is -1 is AF, as for the omniscient observer:

$$\mathbb{O}([\![P]\!]^{\max}, \mathcal{L}^{\max}, \mathbb{C}, \langle l_1, \rho_1\rangle\langle l_2, \rho_2 = \rho_1[apv \leftarrow 1]\rangle\langle l_3, \rho_3 = \rho_2[i_1 \leftarrow -1]\rangle) = AF$$
$$\mathbb{O}([\![P]\!]^{\max}, \mathcal{L}^{\max}, \mathbb{C}_o, \langle l_1, \rho_1\rangle\langle l_2, \rho_2 = \rho_1[apv \leftarrow 1]\rangle\langle l_3, \rho_3 = \rho_2[i_1 \leftarrow -1]\rangle) = AF$$

4.3 Tuning the Cognizance of the Observer

We would like to understand if and how by tuning the precision of the cognizance of the observer we influence the results of responsibility analysis. To this end we consider two cognizance relations $\mathbb{C}_1$ and $\mathbb{C}_2$, where $\mathbb{C}_1$ is a refinement of $\mathbb{C}_2$, namely the two cognizance are comparable and $\mathbb{C}_1$ is more precise than $\mathbb{C}_2$. In this case it is possible to prove that when considering a trace σ the observer with a more precise cognizance can derive more precise information regarding the predicted behavior that the trace guarantees.

Lemma 3. *If* $\mathbb{C}_1 \sqsubseteq \mathbb{C}_2$ *then* $\mathbb{O}([\![P]\!]^{\max}, \mathcal{L}^{\max}, \mathbb{C}_1, \sigma) \subseteq \mathbb{O}([\![P]\!]^{\max}, \mathcal{L}^{\max}, \mathbb{C}_2, \sigma)$

The above Lemma (whose proof is in the Appendix) states that a more concrete observer can be more precise in identifying the property ensured by a considered trace.

Example: When considering the access control example we have that the non-omniscient observer $\mathbb{C}$ that does not see the input from the 1st administrator cannot conclude that a trace that terminates with a negative input from the 1st administrator leads to AF, while this can be observed by an omniscient observer:

$$\mathbb{O}(\llbracket P \rrbracket^{\max}, \mathcal{L}^{\max}, \mathbb{C}_o, \langle l_1, \rho_1 \rangle \langle l_2, \rho_2 = \rho_1[apv \leftarrow 1] \rangle \langle l_3, \rho_3 = \rho_2[i_1 \leftarrow -1] \rangle) = AF$$

$$\mathbb{O}(\llbracket P \rrbracket^{\max}, \mathcal{L}^{\max}, \mathbb{C}, \langle l_1, \rho_1 \rangle \langle l_2, \rho_2 = \rho_1[apv \leftarrow 1] \rangle \langle l_3, \rho_3 = \rho_2[i_1 \leftarrow -1] \rangle) = \top^{\max}$$

However, when considering the actions that are recognized as responsible of a given behavior from observers with a comparable cognizance we have that the set of responsible actions are not related and the two observers will, in general, identify different actions as responsible. So by tuning the precision of the cognizance of the observer we have different actions that are seen as responsible, and are those actions that implement the free choice with respect to the cognizance of the observer. Thus, in general we have that if $\mathbb{C}_1 \sqsubseteq \mathbb{C}_2$ then the set of actions classified as responsible of a behavior $\mathcal{B}$ by $\mathbb{C}_1$ is not a subset of the actions classified as responsible by $\mathbb{C}_2$:

$$\{\tau_R \mid \sigma_H \tau_R \sigma_F \in \alpha_R(\llbracket P \rrbracket^{\max}, \mathcal{L}^{\max}, \mathbb{C}_2, \mathcal{B}, \llbracket P \rrbracket^{Pref})\} \not\subseteq$$

$$\{\tau_R \mid \sigma_H \tau_R \sigma_F \in \alpha_R(\llbracket P \rrbracket^{\max}, \mathcal{L}^{\max}, \mathbb{C}_1, \mathcal{B}, \llbracket P \rrbracket^{Pref})\}$$

as shown by the following example.

Example: Let us consider the access control example, and assume that we are analyzing the responsibility for AF and AS of all the prefix traces in $\llbracket P \rrbracket^{Pref}$. Let us consider the following cognizance of the observer:

- $\mathbb{C}_1$ that does not distinguish the value of i_1 given in input at program point l_2, and does not distinguish the value assigned to apv at program point l_3;
- $\mathbb{C}_2$ that does not distinguish the value of i_1 given in input at program point l_2, and distinguishes the value assigned to apv at program point l_3;
- $\mathbb{C}_3$ that does not distinguish the value of i_2 given in input at program point l_4, and does not distinguish the value assigned to apv at program point l_5;
- $\mathbb{C}_4$ that does not distinguish the value of i_2 given in input at program point l_4, and distinguishes the value assigned to apv at program point l_5;

Observe that $\mathbb{C}_o \sqsubseteq \mathbb{C}_2 \sqsubseteq \mathbb{C}_1$ and $\mathbb{C}_o \sqsubseteq \mathbb{C}_4 \sqsubseteq \mathbb{C}_3$. In the following table we report the responsible actions when analyzing the set of all prefixes traces $\llbracket P \rrbracket^{Pref}$:

	AF	AS
$\mathbb{C}_o$ omniscient	$i_1 := [1;2]$ if $i_1 \leq 0$ $i_2 := [1;2]$ if $i_1 > 0 \wedge i_2 \leq 0$	$i_2 := [1;2]$ if $i_1 > 0 \wedge i_2 > 0$
$\mathbb{C}_1$ no i_1 no apv at l_3	$i_2 := [1;2]$ if $i_2 \leq 0$	none
$\mathbb{C}_2$ no i_1	$i_2 := [1;2]$ if $apv = 1 \wedge i_2 \leq 0$ $apv := (i_1 \leq 0)? -1 : apv$ if $apv = -1$	$i_2 := [1;2]$ if $apv = 1 \wedge i_2 > 0$
$\mathbb{C}_3$ no i_2 no apv at l_5	$i_1 := [1;2]$ if $i_1 \leq 0$	none
$\mathbb{C}_4$ no i_2	$i_1 := [1;2]$ if $i_1 \leq 0$ $apv := (i_2 \leq 0)? -1 : apv$ if $i_1 > 0 \wedge apv = -1$	$apv := (i_2 \leq 0)? -1 : apv$ if $i_1 > 0 \wedge apv = 1$

As discussed earlier when the observer has cognizance $\mathbb{C}_1$ and cannot see the value of i_1 and neither the value assumed by apv at program point l_3, then the action $i_2 := [1;2]$ corresponding to the input of the 2nd administrator is responsible of access failure when the value for i_2 is negative or zero. While when the observer has cognizance $\mathbb{C}_2$ that cannot see the value of i_1, while the observer can precisely distinguish the value assumed by apv at program point l_3, then both the actions $i_2 := [1;2]$ and $apv := (i_1 \leq 0)? -1 : apv$ are responsible of access failure, respectively when the value for i_2 is negative or zero and when the value assigned to apv is -1. In particular, we can observe that in this case the action $apv := (i_1 \leq 0)? -1 : apv$ was not considered responsible for access failure by the omniscient observer that can see the value assumed by i_1 after the free choice involved in the input from the 1st administrator. The observer with cognizance $\mathbb{C}_2$ does not see the value assumed by i_1, but he/she can see the effects of the specific input on the value assigned to apv at program point l_3. Thus, even if the assignment at l_3 is fully determined by the previous actions and has no free choice, it is still true that from the point of the view of an observer with cognizance $\mathbb{C}_2$, the action $apv := (i_1 \leq 0)? -1 : apv$ represents a sort of free choice and is therefore classified as responsible of access failure.

5 Examples

In this section, we present two examples. The first example focuses on the password timing attack, demonstrating how relaxing assumption A2 can reveal the influence of unseen variables, offering valuable insights. The second example illustrates how adjusting the observer's cognizance can shift the responsibility for a specific behavior along the trace.

Password Timing Attack: A password timing attack occurs when an adversary attempts to derive information on the correct password by analyzing the authentication process, focusing on variations in response times. In particular, the attacker can learn how many characters have been correctly guessed. The

```
l1 :  pwd := [p0, p2, ..., pn−1];                    Array storing the secret password
l2 :  guess := array[n][1, INT_MAX];                 Guessed password from an attacker
l3 :  t := 0;
l4 :  while(t ≠ n){
l5 :      t := (pwd[t] ≠ guess[t])? break : t + 1; }
l6 :  check := (t = n)? 1 : 0;                       Login success when check = 1
l7 :
```

Fig. 3: Password Timing Attack Example

example in Fig. 3 refers to this specific scenario. At program point l_1 the password is initialized to a secret value and stored in the array *pwd* of length n. With a slight abuse of notation, the instruction at program point l_2 represents a random assignment to an array of n elements, where each element assumes a random value in the interval $[1, \text{INT\_MAX}]$. Hence, at program point l_2 the attacker guesses a possible password and the value is stored in the array *guess* of n elements. At program point l_3, the counter variable t is initialized to 0. The while loop at program points l_4 and l_5 iteratively compares the t-th element of the guessed password with the t-th element of the secret password. If they match, t is incremented and the next element is considered. This process continues until a mismatch is found or the end of the array is reached. When a mismatched is found variable t keeps its current value and we exit the loop. The instruction at program point l_6 assigns value 1 to variable *check* when the attacker succeeded in guessing the password and 0 otherwise. Let us analyze the responsibility of access success or failure.

Trace semantics: For the program in Fig. 3 the maximal traces either terminate after n successful comparisons between the elements of the guessed password and the corresponding elements of the secret password, or they terminate as as soon as there is a mismatch between an element of the guessed password and the corresponding element of the secret password (starting from the first element of the array). For exmple, the following trace corresponds to the case where we have match between guessed and secret password, where $pwd = guess = [1, 2, 3]$ and $n = 3$. For improving the readability we have highlighted the loop iterations and used the apex on the memory map to indicate the iteration number:

$$\begin{array}{ll} & \langle l_1, \rho_1\rangle\langle l_2, \rho_2 = \rho_1[pwd \leftarrow [1,2,3]]\rangle\langle l_3, \rho_3 = \rho_2[guess \leftarrow [1,2,3]]\rangle \\ \text{1st iteration} & \langle l_4, \rho_4^1 = \rho_3[t \leftarrow 0]\rangle\langle l_5, \rho_5^1 = \rho_4^1\rangle \\ \text{2nd iteration} & \langle l_4, \rho_4^2 = \rho_5^1[t \leftarrow 1]\rangle\langle l_5, \rho_5^2 = \rho_4^2\rangle \\ \text{3rd iteration} & \langle l_4, \rho_4^3 = \rho_5^2[t \leftarrow 2]\rangle\langle l_5, \rho_5^3 = \rho_4^3\rangle \\ \text{guard false} & \langle l_4, \rho_4^4 = \rho_5^3[t \leftarrow 3]\rangle\langle l_6, \rho_6 = \rho_4^4\rangle\langle l_7, \rho_7 = \rho_6[check \leftarrow 1]\rangle \end{array}$$

The following maximal trace corresponds to a mismatched where $pwd = [1, 2, 3]$ and $guess = [1, 2, 5]$:

$\langle l_1, \rho_1\rangle\langle l_2, \rho_2 = \rho_1[pwd \leftarrow [1,2,3]]\rangle\langle l_3, \rho_3 = \rho_2[guess \leftarrow [1,2,5]]\rangle$
1st iteration $\langle l_4, \rho_4^1 = \rho_3[t \leftarrow 0]\rangle\langle l_5, \rho_5^1 = \rho_4^1\rangle$
2nd iteration $\langle l_4, \rho_4^2 = \rho_5^1[t \leftarrow 1]\rangle\langle l_5, \rho_5^2 = \rho_4^2\rangle$
3rd iteration $\langle l_4, \rho_4^3 = \rho_5^2[t \leftarrow 2]\rangle\langle l_5, \rho_5^3 = \rho_4^3\rangle$
mismatch $\langle l_6, \rho_6 = \rho_5^3\rangle\langle l_7, \rho_7 = \rho_6[check \leftarrow 0]\rangle$

Lattice of Trace Properties: We consider two behavioral properties corresponding to Authentication Success (AS) and Authentication Failure (AF), where:

- $AS = \{\sigma_i \dots \sigma_f \in [\![P]\!]^{\max} \mid \sigma_f = \langle l_7, \rho_f\rangle \wedge \rho_f(check) = 1\}$
- $AF = \{\sigma_i \dots \sigma_f \in [\![P]\!]^{\max} \mid \sigma_f = \langle l_7, \rho_f\rangle \wedge \rho_f(check) = 0\}$

Hence, the lattice of trace properties is $\mathcal{L}^{\max} = \{\emptyset, AS, AF, [\![P]\!]^{\max}\}$.

Cognizance: We analyze the responsibility of AF with respect to three different cognizance of the observer:

1. Omniscient Observer $\mathbb{C}_o$, in this case the observation function behaves as follows:
   - $\mathbb{O}([\![P]\!]^{\max}, \mathcal{L}^{\max}, \mathbb{C}_o, \langle l_1, \rho_1\rangle\langle l_2, \rho_2 = \rho_1[pwd \leftarrow [p_0, \dots, p_{n-1}]]\rangle) = \top^{\max}$;
   - $\mathbb{O}([\![P]\!]^{\max}, \mathcal{L}^{\max}, \mathbb{C}_o, \langle l_1, \rho_1\rangle\langle l_2, \rho_2 = \rho_1[pwd \leftarrow [p_0, \dots, p_{n-1}]]\rangle\langle l_3, \rho_3 = \rho_2[guess \leftarrow [g_0, \dots, g_{n-1}]]\rangle) = AS$ if $p_i = g_i$ for $i \in [0, n-1]$;
   - $\mathbb{O}([\![P]\!]^{\max}, \mathcal{L}^{\max}, \mathbb{C}_o, \langle l_1, \rho_1\rangle\langle l_2, \rho_2 = \rho_1[pwd \leftarrow [p_0, \dots, p_{n-1}]]\rangle\langle l_3, \rho_3 = \rho_2[guess \leftarrow [g_0, \dots, g_{n-1}]]\rangle) = AF$ if $\exists i \in [0, n-1]$ such that $p_i \neq g_i$.

   The omniscient observer identifies as responsible of either AF or AS the input at program point l_2.
2. Observer who cannot see the value of the secret password pwd, whose cognizance is denoted by $\mathbb{C}_1$. In this case the observation function behaves as follows:
   - $\mathbb{O}([\![P]\!]^{\max}, \mathcal{L}^{\max}, \mathbb{C}_1, \langle l_1, \rho_1\rangle\langle l_2, \rho_2 = \rho_1[pwd \leftarrow [p_0, \dots, p_{n-1}]]\rangle\langle l_3, \rho_3 = \rho_2[guess \leftarrow [g_0, \dots, g_{n-1}]]\rangle\langle l_4, \rho_4^1 = \rho_3[t \leftarrow 0]\rangle) = \top^{\max}$. The observer cannot see the value of the secret password and he/she cannot know whether the guessed password corresponds to the secret password or not.
   - The observation function returns AS when the instruction at program point l_5 assigns n at variable t, which means that all the elements of the secret and guessed password match.
   - The observation function returns AF at program point l_5 when the value of variable t is not updated and we exit the loop with $t \neq n$. This means that we have a mismatch when comparing the t-th element of the secret and guessed password.

   Following this reasoning, when considering an observer that does not see the value pwd, the responsibility analysis will identify the assignment of the value n to variable t at program point l_5 as responsible for the AS behavior. This action is also the one responsible of the AF behavior when the value of variable t is not updated and we exit the loop with $t \neq n$. This means, that for any maximal trace we have that the action responsible of either

AS or AF is the one corresponding to the execution of the instruction at program point l_5 in the last iteration of the while loop in the considered trace. Note that this is the point where the observer with cognizance $\mathbb{C}_1$ realizes whether the guessed password matches or not the secret password. Namely this instruction is a free choice with respect to cognizance $\mathbb{C}_1$.

3. Observer who cannot distinguish the value of *pwd* nor the value of the variable t, whose cognizance is denoted by $\mathbb{C}_2$. In this case, the observation function will map all traces to $\top^{\max}$ except for the ones with the assignment to *check*, as it is the only distinguishable value in the entire computation. In this case no action is recognized as responsible of the value of variable *check*.

In this scenario it becomes evident that a less refined cognizance corresponds to a higher level of security. Imagine the observer being able to make multiple attempts to guess the correct password to log into the system. The observer with cognizance $\mathbb{C}_1$ is able to see the value of the counter variable t and he/she can use this information to understand the number of correctly guessed password elements. The second observer with cognizance $\mathbb{C}_2$, on the other hand, does not have any information about the value of variable t. As a result, he/she cannot get any useful information from the execution while attempting to log into the system. The difference in the results of responsibility analysis when considering cognizance $\mathbb{C}_1$ and $\mathbb{C}_2$ highlights the benefits that a potential attacker can have when knowing the value of variable t, and therefore suggests to hide such a value in order to prevent the disclosure of important information to a potential attacker.

l_1 :	$balance := 100\$;$	Initial value of bank account
l_2 :	$i_1 := [20\$; 100\$];$	First user's input
l_3 :	$i_2 := [20\$; 100\$];$	Second user's input
l_4 :	$out_1 := i_1 + i_2;$	
l_5 :	$i_3 := [20\$; 100\$];$	Third user's input
l_6 :	$out_2 := i_3 + out_1;$	
l_7 :	$balance := (balance - out_2 > 0)?\ balance - out_2 : -1;$	$balance = -1$ alert
l_8 :		

Fig. 4: Negative or zero balance example

Zero or Negative Balance: Let us consider a scenario where three users have access to the same bank account whose initial balance is of 100$ and each user can withdraw an amount of money in the interval $[20\$; 100\$]$. We are interested in studying the actions responsible of the positive value of the variable *balance*, or of the alert value of -1 at program point l_7. Formally we consider the following trace properties:

- Positive Balance $PB = \{\sigma_i \dots \sigma_f \in [\![P]\!]^{\max} \mid \sigma_f = \langle l_8, \rho_f \rangle,\ \rho_f(balance) > 0\}$

– Alert $NZB = \{\sigma_i \dots \sigma_f \in [\![P]\!]^{\max} \mid \sigma_f = \langle l_8, \rho_f \rangle,\ \rho_f(balance) \leq 0\}$

Let us consider the maximal trace σ where each user withdraws 60\$:

$\sigma = \langle l_1, \rho_1 \rangle \langle l_2, \rho_2 = \rho_1[balance \leftarrow 100\$] \rangle \langle l_3, \rho_3 = \rho_2[i_1 \leftarrow 60\$] \rangle \langle l_4, \rho_4 = \rho_3[i_2 \leftarrow 60\$] \rangle \langle l_5, \rho_5 = \rho_4[out_1 \leftarrow 120\$] \rangle \langle l_6, \rho_6 = \rho_5[i_3 \leftarrow 60\$] \rangle \langle l_7, \rho_7 = \rho_6[out_2 \leftarrow 180\$] \rangle \langle l_8, \rho_8 = \rho_7[balance = -1] \rangle$

Let us studying the responsibility of NZB of this specific trace σ with respect to three different cognizance of the observer:

1. $\mathbb{C}_o$: the omniscient observer sees as responsible of NZB the input of the first user. If the first user withdraws 60\$ we can be sure that at the end the variable *balance* will have vale -1, no matter what the other users withdraw. For this reason the assignment at program point l_2 is identified as the responsible action.
2. $\mathbb{C}_1$: Observer who cannot see the value of i_1. In this case the action responsible of NZB is the input of the second observer, namely the assignment at program point l_3. Once again if the second user withdraws 60\$ we can be sure that at the end the variable *balance* will have vale -1, no matter what the other users withdraw.
3. $\mathbb{C}_2$: Observer who cannot see the value of i_1 and of i_2. In this case the action responsible of NZB is the assignment to the variable out_1 at program point l_4. If the sum of what has been withdrawn by the first and the second user is 120\$ than we can be sure that at the end the variable *balance* will have vale -1, no matter what the third user does.
4. $\mathbb{C}_3$ Observer who cannot see the value of i_1, of i_2 and of out_1. In this case the action responsible of NZB is the input of the third observer, namely the assignment at program point l_5. Once again if the third user withdraws 60\$ we can be sure that at the end the variable *balance* will have vale -1, no matter what the other users withdraw.

		Responsible action of NZB
$\mathbb{C}_o$	sees everything	$i_1 := [20\$; 100\$];$
$\mathbb{C}_1$	no i_1	$i_2 := [20\$; 100\$];$
$\mathbb{C}_2$	no i_1 and i_2	$out_1 := i_1 + i_2;$
$\mathbb{C}_3$	no i_1, i_2 and out_1	$i_3 := [20\$; 100\$];$

In this case we can see that in our setting there is a difference between the observers with cognizance $\mathbb{C}_2$ and $\mathbb{C}_3$ when analyzing the trace where all the users withdraw 60\$. The more refined observer $\mathbb{C}_2$ that can see the value of out_1 can realize that the action responsible of the alert is something that has happened before the input of the third user, even if he/she is not able to precisely know who is the responsible among the first and second user. This is possible by observing that the assignment to out_1 at program point l_4 is a free-choice with respect to the cognizance $\mathbb{C}_2$ that does not see the precise value withdrawn by the first two users but can see the sum of such amounts. The observer with cognizance $\mathbb{C}_2$ sees the action at program point l_4 as responsible of NZB, since it is the first point along the trace where the observer sees the effects of the

amounts withdrawn by the first two users and, when considering trace σ above, this is enough for ensuring the *NZB* behavior. On the other hand, the observer with cognizance $\mathbb{C}_3$ that does not see the input of the first and second user and neither the sum of such values, identifies as responsible of the *NZB* behavior the input of the third user.

In this example, it is evident that adjusting the observer's cognizance on the trace where all users withdraw 60$, leads to shifting the responsibility of *NZB* along the trace.

6 Discussion

In this work we have further investigated the notion of responsibility analysis introduced by Deng and Cousot in [5,6]. The main novelties of responsibility analysis with respect to the previous work on causality is that an action is classified as responsible for a given behavior if it implements free choice and with respect to the cognizance of the observer who is analyzing the responsibility. We agree that these two important aspects have to be considered when identifying the root cause of a behavior. For this reason, in this work, we have further investigated these two aspects. We have studied the effects of tuning the precision of the cognizance of the observer on the set of actions that the observer classifies as responsible. Interestingly, when the cognizance of the observer loses precision, it is not that responsibility analysis loses precision in the sense that it classifies more actions as potentially responsible. Instead, when the cognizance of the observer loses precision, we have that different actions are classified as responsible. By reasoning on the examples, this seems to be related to the fact that observers with different cognizance see different actions as implementing a free choice. Thus, it seems that being an action that realizes a free choice may vary with respect to the cognizance of the observer. We believe that this opens interesting prospects that require further investigation.

We believe that the interplay between the notions of responsibility, free choice and cognizance of the observer still needs to be investigated in order to better understand the possible implications of this kind of analysis. Indeed, these notions are relevant in many aspects of program analysis. Once an error, or an unwanted behavior is identified responsibility analysis can drive the analysis in fixing the identified problem. But also being able to understand the actions responsible of a given output is becoming more and more important in modern systems.

Acknowledgement. This work was partially supported by Air Force Office of Scientific Research under award number FA9550-23-1-0544 and by the project SERICS (PE00000014) under the MUR National Recovery and Resilience Plan funded by the European Union - NextGenerationEU.

References

1. Abadi, M., Banerjee, A., Heintze, N., Riecke, J.G.: A core calculus of dependency. In: Appel, A.W., Aiken, A. (eds.) POPL '99, Proceedings of the 26th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, San Antonio, TX, USA, January 20-22, 1999. pp. 147–160. ACM (1999)
2. Cheney, J., Ahmed, A., Acar, U.A.: Provenance as dependency analysis. CoRR **abs/0708.2173** (2007)
3. Cousot, P., Cousot, R.: Abstract interpretation: A unified lattice model for static analysis of programs by construction or approximation of fixpoints. In: Graham, R.M., Harrison, M.A., Sethi, R. (eds.) Proceedings of the 4th ACM Symposium on Principles of Programming Languages, Los Angeles, California, USA, January 1977. pp. 238–252. ACM (1977). https://doi.org/10.1145/512950.512973
4. Cousot, P., Cousot, R.: Systematic design of program analysis frameworks. In: Aho, A.V., Zilles, S.N., Rosen, B.K. (eds.) Proceedings of the 6th ACM Symposium on Principles of Programming Languages, San Antonio, Texas, USA, January 1979. pp. 269–282. ACM Press (1979). https://doi.org/10.1145/567752.567778
5. Deng, C., Cousot, P.: Responsibility analysis by abstract interpretation. In: Chang, B.E. (ed.) Static Analysis - 26th International Symposium, SAS 2019, Porto, Portugal, October 8-11, 2019, Proceedings. Lecture Notes in Computer Science, vol. 11822, pp. 368–388. Springer (2019)
6. Deng, C., Cousot, P.: The systematic design of responsibility analysis by abstract interpretation. ACM Trans. Program. Lang. Syst. **44**(1), 3:1–3:90 (2022)
7. Halpern, J.Y., Pearl, J.: Causes and explanations: A structural-model approach: Part 1: Causes. Brit. J. Philos. Sci **56**(4), 843?887 (2005)
8. Lewis, D.: Causation. J. Philoso. **70**(17) (1973)
9. Lewis, D.: Counterfactuals. John Wiley & Sons (2013)
10. Pearl, J.: Causality: Models, Reasoning and Inference (2nd ed.). Cambridge University Press (2013)
11. Pistoia, M., Flynn, R.J., Koved, L., Sreedhar, V.C.: Interprocedural analysis for privileged code placement and tainted variable detection. In: Black, A.P. (ed.) ECOOP 2005 - Object-Oriented Programming, 19th European Conference, Glasgow, UK, July 25-29, 2005, Proceedings. Lecture Notes in Computer Science, vol. 3586, pp. 362–386. Springer (2005)
12. Rival, X.: Understanding the origin of alarms in astrée. In: International Static Analysis Symposium. pp. 303–319. Springer (2005)
13. Urban, C., Müller, P.: An abstract interpretation framework for input data usage. In: Programming Languages and Systems: 27th European Symposium on Programming, ESOP 2018, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2018, Thessaloniki, Greece, April 14-20, 2018, Proceedings 27. pp. 683–710. Springer (2018)
14. Weiser, M.: Program slicing. IEEE Transactions on Software Engineering **SE-10**(4), 352–357 (1984)

7 Appendix

Lemma 1 Assumption A2 implies that $\sigma \in [\![P]\!]^{Pref} \Rightarrow \mathbb{C}(\sigma) \subseteq [\![P]\!]^{Pref}$.

Proof. A2 states that $\forall \sigma \in \Sigma^{*\infty}.\sigma \notin [\![P]\!]^{Pref} \Rightarrow \mathbb{C}(\sigma) \cap [\![P]\!]^{Pref} = \emptyset$. Note that also the converse holds. Assume that $\mathbb{C}(\sigma) \cap [\![P]\!]^{Pref} = \emptyset$, this means that $\forall \sigma' \in \mathbb{C}(\sigma)$ we have that $\sigma' \notin [\![P]\!]^{Pref}$ and this implies that $\sigma \notin [\![P]\!]^{Pref}$ which contradicts the hypothesis. Thus, we have that $\sigma \notin [\![P]\!]^{Pref} \Leftrightarrow \mathbb{C}(\sigma) \cap [\![P]\!]^{Pref} = \emptyset$. Given $\sigma \in [\![P]\!]^{Pref}$, assume that $\mathbb{C}(\sigma) \not\subseteq [\![P]\!]^{Pref}$. This means that $\exists \sigma' \in \mathbb{C}(\sigma) : \sigma' \notin [\![P]\!]^{Pref}$. From what we have just proved this means that $\mathbb{C}(\sigma') \cap [\![P]\!]^{Pref} = \emptyset$ and since $\sigma \in \mathbb{C}(\sigma')$ we get to the contradiction that $\sigma \notin [\![P]\!]^{Pref}$.

Theorem 1 For the cognizance $\mathbb{C}$ of an attacker, if τ_R is responsible for a behavior $\mathcal{B}$ in a valid trace $\sigma_H \tau_R \sigma_F$, then:

1. $\sigma_H \tau_R$ guarantees the occurrence of the behavior $\mathcal{B}$,
2. there exists a valid trace $\sigma' s \in [\![P]\!]^{Pref}$ such that $\sigma' \in \mathbb{C}(\sigma_H)$ and $\sigma' s$ does not guarantee the behavior $\mathcal{B}$.

Proof.
1. By definition, we have that $\emptyset \subset \mathbb{O}([\![P]\!]^{\max}, \mathcal{L}^{\max}, \mathbb{C}, \sigma_H \tau_R) \subseteq \mathcal{B}$. This is true iff $\overset{\diamond}{\cup} \{\mathbb{I}([\![P]\!]^{\max}, \mathcal{L}^{\max}, \sigma') \mid \sigma' \in \mathbb{C}(\sigma_H \tau_R)\} \subseteq \mathcal{B}\}$, namely if $\forall \sigma' \in \mathbb{C}(\sigma_H \tau_R)$ it holds that $\mathbb{I}([\![P]\!]^{\max}, \mathcal{L}^{\max}, \sigma') \subseteq \mathcal{B}$. In particular, it holds for $\sigma_H \tau_R$, and this means that $\sigma_H \tau_R \in \alpha_{Pred}([\![P]\!]^{\max})\mathcal{B}$ and therefore $\sigma_H \tau_R$ guarantees the behavior $\mathcal{B}$.
2. By definition, we have that $\mathbb{O}([\![P]\!]^{\max}, \mathcal{L}^{\max}, \mathbb{C}, \sigma_H) \not\subseteq \mathcal{B}$. This means that $\overset{\diamond}{\cup} \{\mathbb{I}([\![P]\!]^{\max}, \mathcal{L}^{\max}, \sigma') \mid \sigma' \in \mathbb{C}(\sigma_H) \not\subseteq \mathcal{B}\}$. This means that $\exists \sigma' \in \mathbb{C}(\sigma_H)$ such that $\mathbb{I}([\![P]\!]^{\max}, \mathcal{L}^{\max}, \sigma') \not\subseteq \mathcal{B}$. This σ' is such that $\sigma' \in [\![P]\!]^{Pref}$, since if $\sigma' \notin [\![P]\!]^{Pref}$ then $\mathbb{I}([\![P]\!]^{\max}, \mathcal{L}^{\max}, \sigma') = \bot^{\max} \subseteq \mathcal{B}$. Thus, σ' is valid: $\sigma' \in [\![P]\!]^{Pref}$. Moreover, (from what observed on the inquiry function and proved in Corollary 4 of [6]) it has to hold that $\mathbb{I}([\![P]\!]^{\max}, \mathcal{L}^{\max}, \sigma') = \overset{\diamond}{\cup} \{\mathbb{I}([\![P]\!]^{\max}, \mathcal{L}^{\max}, \sigma' s) \mid \sigma' s \in [\![P]\!]^{Pref}\} \not\subseteq \mathcal{B}$. Namely, $\exists \sigma' s \in [\![P]\!]^{Pref}$ such that $\mathbb{I}([\![P]\!]^{\max}, \mathcal{L}^{\max}, \sigma' s) \not\subseteq \mathcal{B}$. Which means that this valid trace $\sigma' s \notin \alpha_{Pred}([\![P]\!]^{\max})\mathcal{B}$ and therefore does not guarantee the behavior $\mathcal{B}$.

Lemma 2 If $\mathbb{C} \sqsubseteq \equiv_{\mathbb{I}}$ then $\mathbb{I}([\![P]\!]^{\max}, \mathcal{L}^{\max}, \sigma) = \mathbb{O}([\![P]\!]^{\max}, \mathcal{L}^{\max}, \mathbb{C}, \sigma)$

Proof. By definition, $\mathbb{O}([\![P]\!]^{\max}, \mathcal{L}^{\max}, \mathbb{C}, \sigma) = \overset{\diamond}{\cup} \{\mathbb{I}([\![P]\!]^{\max}, \mathcal{L}^{\max}, \sigma') \mid \sigma' \in \mathbb{C}(\sigma)\}$. Thus, it is enough to show that if $\mathbb{C} \sqsubseteq \equiv_{\mathbb{I}}$ then $\forall \sigma' \in \mathbb{C}(\sigma)$ it holds that $\mathbb{I}([\![P]\!]^{\max}, \mathcal{L}^{\max}, \sigma) = \mathbb{I}([\![P]\!]^{\max}, \mathcal{L}^{\max}, \sigma')$. If $\mathbb{C} \sqsubseteq \equiv_{\mathbb{I}}$ then $\forall \sigma \in [\![P]\!]^{Pref} : [\sigma]_{\mathbb{C}} \subseteq [\sigma]_{\mathbb{I}}$. Therefore, if $\mathbb{C}(\sigma) = \mathbb{C}(\sigma')$ then $\mathbb{I}([\![P]\!]^{\max}, \mathcal{L}^{\max}, \sigma) = \mathbb{I}([\![P]\!]^{\max}, \mathcal{L}^{\max}, \sigma')$ and this implies that $\forall \sigma' \in \mathbb{C}(\sigma)$, then $\mathbb{I}([\![P]\!]^{\max}, \mathcal{L}^{\max}, \sigma) = \mathbb{I}([\![P]\!]^{\max}, \mathcal{L}^{\max}, \sigma')$, therefore $\overset{\diamond}{\cup} \{\mathbb{I}([\![P]\!]^{\max}, \mathcal{L}^{\max}, \sigma') \mid \sigma' \in \mathbb{C}(\sigma)\} = \mathbb{I}([\![P]\!]^{\max}, \mathcal{L}^{\max}, \sigma)$.

Corollary 1 If $\mathbb{C} \sqsubseteq \equiv_{\mathbb{I}}$ then

$$\alpha_R([\![P]\!]^{\max}, \mathcal{L}^{\max}, \mathbb{C}, \mathcal{B}, \mathcal{T}) = \alpha_R([\![P]\!]^{\max}, \mathcal{L}^{\max}, \mathbb{C}_o, \mathcal{B}, \mathcal{T})$$

Proof.

$$\alpha_R([\![P]\!]^{\max}, \mathcal{L}^{\max}, \mathbb{C}, \mathcal{B}, \mathcal{T}) = \left\{ \langle \sigma_H, \tau_R, \sigma_F \rangle \;\middle|\; \begin{array}{l} \sigma_H \tau_R \sigma_F \in [\![P]\!]^{Pref} \\ \emptyset \subset \mathbb{O}([\![P]\!]^{\max}, \mathcal{L}^{\max}, \mathbb{C}, \sigma_H \tau_R) \subseteq \mathcal{B} \\ \mathbb{O}([\![P]\!]^{\max}, \mathcal{L}^{\max}, \mathbb{C}, \sigma_H) \not\subseteq \mathcal{B} \end{array} \right\}$$

[From Lemma 2]

$$= \left\{ \langle \sigma_H, \tau_R, \sigma_F \rangle \;\middle|\; \begin{array}{l} \sigma_H \tau_R \sigma_F \in [\![P]\!]^{Pref} \\ \emptyset \subset \mathbb{I}([\![P]\!]^{\max}, \mathcal{L}^{\max}, \sigma_H \tau_R) \subseteq \mathcal{B} \\ \mathbb{I}([\![P]\!]^{\max}, \mathcal{L}^{\max}, \sigma_H) \not\subseteq \mathcal{B} \end{array} \right\}$$

[From Definition of $\mathbb{C}_o$]

$$= \left\{ \langle \sigma_H, \tau_R, \sigma_F \rangle \;\middle|\; \begin{array}{l} \sigma_H \tau_R \sigma_F \in [\![P]\!]^{Pref} \\ \emptyset \subset \mathbb{O}([\![P]\!]^{\max}, \mathcal{L}^{\max}, \mathbb{C}_o, \sigma_H \tau_R) \subseteq \mathcal{B} \\ \mathbb{O}([\![P]\!]^{\max}, \mathcal{L}^{\max}, \mathbb{C}_o, \sigma_H) \not\subseteq \mathcal{B} \end{array} \right\}$$

$$= \alpha_R([\![P]\!]^{\max}, \mathcal{L}^{\max}, \mathbb{C}_o, \mathcal{B}, \mathcal{T})$$

Lemma 3 If $\mathbb{C}_1 \sqsubseteq \mathbb{C}_2$ then $\mathbb{O}([\![P]\!]^{\max}, \mathcal{L}^{\max}, \mathbb{C}_1, \sigma) \subseteq \mathbb{O}([\![P]\!]^{\max}, \mathcal{L}^{\max}, \mathbb{C}_2, \sigma)$

Proof. The hypothesis $\mathbb{C}_1 \sqsubseteq \mathbb{C}_2$ means that $\forall \sigma \in \Sigma^{*\infty} : \mathbb{C}_1(\sigma) \subseteq \mathbb{C}_2(\sigma)$. Hence:

$$\begin{aligned} \mathbb{O}([\![P]\!]^{\max}, \mathcal{L}^{\max}, \mathbb{C}_1, \sigma) &= \mathop{\hat{\cup}} \{\mathbb{I}([\![P]\!]^{\max}, \mathcal{L}^{\max}, \sigma') \mid \sigma' \in \mathbb{C}_1(\sigma)\} \\ &\quad \text{[Since } \mathbb{C}_1(\sigma) \subseteq \mathbb{C}_2(\sigma)\text{]} \\ &\subseteq \mathop{\hat{\cup}} \{\mathbb{I}([\![P]\!]^{\max}, \mathcal{L}^{\max}, \sigma') \mid \sigma' \in \mathbb{C}_2(\sigma)\} \\ &= \mathbb{O}([\![P]\!]^{\max}, \mathcal{L}^{\max}, \mathbb{C}_2, \sigma) \end{aligned}$$