Doctoral Thesis

A Formal Framework for Correct and Secure Compilation

Author:
Carmine Abate

Supervisor:
Cătălin Hriţcu

Declaration of Authorship

Signed:

Date:

Abstract

A Formal Framework for Correct and Secure Compilation

by Carmine Abate

The correctness and security of software can be guaranteed by the use of formal methods. These techniques are applied to the source code which is then compiled into executable code, a process that however may introduce bugs or vulnerabilities. To prevent such issues, formal methods can be applied to the compiler itself, but this requires rigorous and formal criteria, for both correct and secure compilation.
In this thesis, we propose a formal framework for correct and secure compilation criteria, centered on the intuition that a compiler should preserve the (security) properties of the source programs.
Security properties are modeled as predicates over execution traces and sets of execution traces, of single or multiple arities, respectively known as trace properties, hyperproperties, and relational hyperproperties.
Our compiler correctness definitions are parameterized by the class of trace properties, hyperproperties, and relational hyperproperties that are satisfied by source programs and preserved by a compiler. For secure compilation, the compiler must preserve satisfaction in presence of linked adversarial code.
Every criterion we propose comes in more equivalent formulations: one better suited for proofs, the others explicitly describe the guarantees on compiled programs.
We provide a comparison of the criteria we propose both among them and with the more established criterion of full abstraction.

Contents

List of Figures

List of Publications

List of peer-reviewed publications of the author:

[11] Carmine Abate et al. "When good components go bad: Formally secure compilation despite dynamic compromise". In: *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*. 2018, pp. 1351–1368

[8] Carmine Abate et al. "Journey beyond full abstraction: Exploring robust property preservation for secure compilation". In: *2019 IEEE 32nd Computer Security Foundations Symposium (CSF)*. IEEE. 2019, pp. 256–25615 **Distinguished paper award.**

[10] Carmine Abate et al. "Trace-relating compiler correctness and secure compilation". In: *European Symposium on Programming*. Springer. 2020, pp. 1–28

[7] Carmine Abate et al. "An Extended Account of Trace-relating Compiler Correctness and Secure Compilation". In: *ACM Transactions on Programming Languages and Systems (TOPLAS)* 43.4 (2021), pp. 1–48

[6] Carmine Abate, Matteo Busi, and Stelios Tsampas. "Fully Abstract and Robust Compilation". In: *Programming Languages and Systems*. Ed. by Hakjoo Oh. Cham: Springer International Publishing, 2021, pp. 83–101. ISBN: 978-3-030-89051-3

[9] Carmine Abate et al. "Ssprove: A foundational framework for modular cryptographic proofs in coq". In: *2021 IEEE 34th Computer Security Foundations Symposium (CSF)*. IEEE. 2021, pp. 1–15 **Distinguished paper award.**

Chapter 1

Introduction

Software bugs and vulnerabilities may result in huge pecuniary and human resources costs [125], privacy violations [46], and other security criticality [87, 75, 67]. For these reasons, past years have seen an increasing effort in research for both efficiently finding [25, 86] and preventing bugs and vulnerabilities. Prevention can be practiced via formal verification [54], adopting programming languages with expressive type systems or primitives for abstractions, e.g., modules and interfaces, and good programming disciplines [39].
The above mentioned techniques contribute to increasing the reliability of software, providing strong correctness and security guarantees but are most often limited to *source* programs: the programs that are executed are the result of a *compilation* process that can introduce afresh bugs and vulnerabilities.
To prevent incorrect behaviors or vulnerabilities in compiled programs, it's possible to apply formal methods in the design and development of the compilation chains themselves.

Proving the correctness of the compiler is an old idea [88, 81] that however has been achieved only relatively recently with the advent of CompCert [74], a formally verified compiler for the C language. CompCert led the way to the realization of other compiler verification projects [124, 84, 115], each of them establishing a relation between the semantics of a program, before and after its compilation. The semantics of programs usually capture their execution traces,i.e., sequences of events such as inputs from and outputs to an environment, and the correctness of a compiler $\downarrow$ can be expressed with the following formula:

$$\mathsf{CC} \equiv \forall \mathsf{W}\ \forall t.\ \mathsf{W}{\downarrow} \rightsquigarrow t \Rightarrow\ \mathsf{W} \rightsquigarrow t.$$

CC states that for every source program W, any trace t that can be produced by the compiled program and w.r.t the target semantics– $\mathsf{W}{\downarrow} \rightsquigarrow t$ – can also be produced by W and w.r.t the source semantics – $\mathsf{W} \rightsquigarrow t$. This fact can be proved by showing that every execution step of $\mathsf{W}{\downarrow}$ can be *simulated* by one or more execution steps of W.

Notice that CC ensures that the compiler does not introduce any unexpected behavior,i.e., the execution traces of $\mathsf{W}{\downarrow}$ are a subset of those of W. The inclusion of execution traces, is a sufficient – and necessary – condition to guarantee that the properties of (the execution traces of) source programs are *preserved* through compilation,

$$\mathsf{TP} \equiv \forall \mathsf{W}\ \forall \pi.\ \mathsf{W} \models \pi \Rightarrow \mathsf{W}{\downarrow} \models \pi,$$

where $\mathsf{W} \models \pi$ means that all the traces t that can be produced during an execution of W satisfy the property π[1].

Preservation of properties eases the reasoning of programmers, who can safely ignore the implementation details of the target language. For example, if a property is a consequence of the type of a program, the programmer does not have to bother about the target representation of that type.

At a glance: on the one hand, TP contains information the user of the compiler benefits from, on the other hand, the designer or developer of the compiler recognizes in CC the theorem to state, and the technique needed to prove it, a simulation. Therefore, the equivalence CC $\iff$ TP adds valuable information to CC and TP considered separately.

Correct compilers can still introduce vulnerabilities. D'Silva, Payer, and Song [45] propose a few scenarios in which provably correct compilers may violate security guarantees of source code, one of their example is built around the code snippet below:

```
crypt(){
key = 0xC0DE; // read key
... // work with the secure key
key = 0x0; // scrub memory
}
```

The function crypt() reads a secret key from a secure channel, performs some computations, and scrubs the key before returning to the caller. The variable key is local to crypt() and scrubbing the key avoids its persistence in memory and prevents an attacker can recover it. The assignment key = 0x0 a *"dead store"* – the key is not read after that assignment – and the elimination of this line of code is a common compiler optimization sometimes proved to be formally correct [74, 28]. Such an optimization, however, introduces a vulnerability against exploits that access values persistent in memory. More concretely, for an attacker that has access to the memory location of the key after the execution of crypt(), it's not possible to observe any difference in the execution of the function crypt() with two different keys, but after compilation – and dead store elimination – she can say exactly which key has been used in each call of crypt().

The fact that the security of crypt() can be expressed as the equivalence of any two of its executions, suggests the compiler should preserve such an equivalence. Abadi [1] first noticed that to preserve security, it's necessary to translate source observationally equivalent programs into target observationally equivalent programs.

Full abstraction requires *preservation* and *reflection* of observational equivalence: source equivalent programs must be compiled to target equivalent programs and source non equivalent programs to target non equivalent ones[2]. In the literature on fully abstract compilation equivalence of programs is usually defined with respect to the evaluation contexts of the languages that model the attackers of the language. Two programs are equivalent if they evaluate to the same values in every possible

[1] here properties are identified with their *extension*,i.e., the set of traces that satisfy the property, and a trace t satisfies a property π *iff* $t \in \pi$.

[2] reflection ensures non-triviality of the compilation and is often subsumed by compiler correctness [93], so that we mainly focus on the preservation of observational equivalence.

context [80], and a compiler is fully abstract if and only if for any two programs $\mathsf{P}_1, \mathsf{P}_2$

$$(\forall \mathsf{C_S}.\ \mathsf{C_S}[\mathsf{P}_1] \approx \mathsf{C_S}[\mathsf{P}_2]) \iff (\forall \mathbf{C_T}.\ \mathbf{C_T}[\mathsf{P}_1{\downarrow}] \approx \mathbf{C_T}[\mathsf{P}_2{\downarrow}]).$$

$\mathsf{C_S}[\mathsf{P}_1] \approx \mathsf{C_S}[\mathsf{P}_1]$ above means that P_1 or P_2 evaluate to the same values when plugged in the source context $\mathsf{C_S}$, or equivalently that $\mathsf{C_S}$ cannot distinguish the two programs. Similarly $\mathbf{C_T}[\mathsf{P}_1{\downarrow}] \approx \mathbf{C_T}[\mathsf{P}_1{\downarrow}]$ means the target context $\mathbf{C_T}$ cannot distinguish the two programs after compilation.
For the example by D'Silva, Payer, and Song [45] discussed above, a fully abstract compiler would ensure that two instances of crypt() – $\mathsf{P}_1, \mathsf{P}_2$ – that are equivalent for an attacker that can observe values persistent in memory, are still equivalent for such an attacker after compilation.

Full abstraction had a remarkable success and has been adopted to establish the preservation of confidentiality and other security properties through compilation, see e.g., [29, 27, 110, 42, 93]. It suffers however of few shortcomings [89, 61], the most worrying – highlighted by the same Abadi [1] – is that it may not preserve (logical) properties of programs. Examples from recent literature show that fully abstract compilers may not preserve properties such as data integrity and confidentiality [95, 8, 6].
Roughly speaking, fully abstract compilers may fail to preserve properties that are not captured by contextual equivalence. One might consider a coarser or a finer grained contextual equivalence to accommodate a certain property, but this approach may exclude some other property of interest and any case requires a new proof for every equivalence adopted.

Starting from the properties one is interested in, and proving the compilation chain preserve those properties, makes the guarantees given by the compiler unambiguous and explicit. For a more concrete example, the following predicate [8]

$$\mathsf{RTP} \equiv \forall \mathsf{P}\ \forall \pi.\ (\forall \mathsf{C}.\ \mathsf{C_S}[\mathsf{P}] \models \pi) \Rightarrow (\forall \mathbf{C_T}.\ \mathbf{C_T}[\mathsf{P}{\downarrow}] \models \pi).$$

states that if a program P *robustly* satisfies a property π,i.e., it satisfies π against any source context $\mathsf{C_S}$, then $\mathsf{P}{\downarrow}$ satisfies the same property against any target context $\mathbf{C_T}$.
RTP – standing for *"robust trace properties preservation"* – provides an explicit description of the guarantees given by a compiler that validates the predicate, but a direct proof of RTP imposes to consider arbitrary trace properties and arbitrary contexts, thus resulting impractical. For this reason, Abate et al. [8] showed the equivalence of RTP with

$$\mathsf{RTC} \equiv \forall \mathbf{C_T}\ \forall \mathsf{P}\ \forall t.\ \mathbf{C_T}[\mathsf{P}{\downarrow}] \rightsquigarrow t \Rightarrow \exists \mathsf{C_S}.\ \mathsf{C_S}[\mathsf{P}] \rightsquigarrow t.$$

Given the equivalence RTP $\iff$ RTC , it's possible to prove robust trace property preservation by *back-translation* of the target context $\mathbf{C_T}$ to a source context. More precisely, one has to prove that that any execution t of $\mathsf{P}{\downarrow}$ when linked with $\mathbf{C_T}$, can be simulated by executing P in some source context $\mathsf{C_S}$.

Trace properties include safety properties such as data integrity and liveness properties such as termination or guaranteed service, but cannot express relations between multiple executions such as *confidentiality*. Therefore Abate et al. [8] explored robust preservation of *hyperproperties* – that model relations between multiple executions of the same program such as confidentiality– and *relational hyperproperties* that model relations between multiple executions and of different programs, such as trace equivalence.

Trace properties, hyperproperties, and relational hyperproperties can be used to define a wide range of criteria for secure compilation such as the robust preservation of data integrity, data confidentiality, code confidentiality, or observational equivalence,i.e., fully abstract compilation itself.
At the same time, the protections to be implemented within the whole compilation chain – including compiler, linker, loader, and hardware – may be significantly different from one criterion to another. To preserve observational equivalence, it's necessary to hide all differences that target contexts can observe and that source ones cannot, while to robustly preserve data integrity, it suffices to protect the internal invariants of the program from the context. Therefore, it's often the case that a secure compilation chain for the robust preservation of data integrity can be more efficiently implemented (see [11, 48, 95]) than a chain that preserves observational equivalence or code confidentiality.

The property-free characterizations [8] such as RTC above allow to compare the different efforts required to implement the protections necessary for different criteria and suggest the technique for proving robust preservation of the compilation chain. We list a selection of them hereafter:

> Robust preservation of trace properties,e.g., data integrity or termination, is equivalent to (RTC):
>
> $$\forall \mathbf{C_T}\ \forall \mathsf{P}\ \forall t.\ \mathbf{C_T}[\mathsf{P}{\downarrow}] \rightsquigarrow t \Rightarrow \exists \mathsf{C_S}.\ \mathsf{C_S}[\mathsf{P}] \rightsquigarrow t$$
>
> Robust preservation of (subset-closed) hyperproperties,e.g., data confidentiality, is equivalent to:
>
> $$\forall \mathbf{C_T}\ \forall \mathsf{P}.\ \exists \mathsf{C_S}.\ \forall t.\ \mathbf{C_T}[\mathsf{P}{\downarrow}] \rightsquigarrow t \Rightarrow \mathsf{C_S}[\mathsf{P}] \rightsquigarrow t$$
>
> Robust preservation of (subset-closed) relational hyperproperties,e.g., code confidentiality, is equivalent to:
>
> $$\forall \mathbf{C_T}.\ \exists \mathsf{C_S}.\ \forall \mathsf{P}\ \forall t.\ \mathbf{C_T}[\mathsf{P}{\downarrow}] \rightsquigarrow t \Rightarrow \mathsf{C_S}[\mathsf{P}] \rightsquigarrow t.$$

The property-free characterizations above can be proved with back-translation techniques that sometimes adapt those developed for fully abstract compilation [8, 42]. As already mentioned, for trace properties such as data integrity (RTC) the back-translation should return a source context $\mathsf{C_S}$ that is able to simulate in the source $\mathbf{C_T}$ and only w.r.t one single execution t of the program P. For data confidentiality, the back-translation must be independent of the execution: all the executions of $\mathsf{P}{\downarrow}$ linked against $\mathbf{C_T}$ must be simulated in the same $\mathsf{C_S}$. Finally, for code confidentiality, the back-translation must be independent of the program itself: all executions of an arbitrary program linked against a target context $\mathbf{C_T}$ must be simulated by the same source context $\mathsf{C_S}$.

The equivalence of property-full and property-free criteria provide useful and valuable information also for secure compilation. On the one hand, the property-full criteria –e.g., RTP – provide an explicit description of the guarantee given by the compiler and allows programmers to reason about the security of their programs P only against source contexts $\mathsf{C_S}$, whose power is bound to respect the source type system, interfaces, and source abstractions.

On the other hand, the property-free criteria –e.g., RTC – clarify the back-translation of target contexts into source ones needed to preserve the intended security properties. Interestingly, some of the back-translations needed for robust preservation criteria such as RTC , are an adaptation of those developed for proving fully abstract compilation [42].

1.1 Outline and Contributions

This thesis builds on the following peer-reviewed papers (a complete list of publications of the author can be found in the dedicated section):

[8] Carmine Abate et al. "Journey beyond full abstraction: Exploring robust property preservation for secure compilation". In: *2019 IEEE 32nd Computer Security Foundations Symposium (CSF)*. IEEE. 2019, pp. 256–25615 **Distinguished paper award.**

[10] Carmine Abate et al. "Trace-relating compiler correctness and secure compilation". In: *European Symposium on Programming*. Springer. 2020, pp. 1–28

[7] Carmine Abate et al. "An Extended Account of Trace-relating Compiler Correctness and Secure Compilation". In: *ACM Transactions on Programming Languages and Systems (TOPLAS)* 43.4 (2021), pp. 1–48

[6] Carmine Abate, Matteo Busi, and Stelios Tsampas. "Fully Abstract and Robust Compilation". In: *Programming Languages and Systems*. Ed. by Hakjoo Oh. Cham: Springer International Publishing, 2021, pp. 83–101. ISBN: 978-3-030-89051-3

The work done in [8] aimed at exploring formal secure compilation criteria based on the notion of robust property preservation,i.e., preservation of those properties that are satisfied against arbitrary adversarial contexts. Distinct classes of properties lead to distinct criteria, that can be easier or harder to achieve in practice. For example the criterion for the robust preservation of safety properties – that include data integrity – is different and easier to achieve than the criterion for the robust preservation of hypersafety – that includes data confidentiality – or trace equivalence.
Additionally, each criterion is proved equivalent to a "property-free" formulation that clarifies which proof techniques apply. The equivalence between the "property-full" and "property-free" formulations of each secure compilation criteria, allows having both an explicit and unambiguous description of the guarantee given by the compiler (property-full) and to specify which back-translation technique should be applied for the proof.

The work done in [10, 7] aimed at studying correct and secure compilation in case source and target observations sensibly differ,i.e., cannot be put in a bijective correspondence, for example, when the values of source and target languages are far apart, or when target executions require the availability of memory while source semantics assume an infinite memory is available.
In all these scenarios source trace properties need to be *interpreted* in the target and target properties in the source. We show how to derive an interpretation of properties that corresponds to the relation between source and target observable events and traces and that leads to equivalent property-full and property-free criteria for both correct and secure compilation.

Finally, in [6] we propose an in-depth comparison between fully abstract compilation and robust preservation. In [8] it's already shown that fully abstract compilers may not robustly preserve some trace properties, and in [6] we discuss which properties are robustly preserved by a fully abstract compiler and which are not. More precisely, we show that a fully abstract compiler induces an interpretation of source properties that describes which properties are robustly satisfied by compiled programs. In the same paper, it's also proposed a formal requirement to add to fully abstract compilation to also guarantee robust preservation (of hyperproperties).

This manuscript offers a more comprehensive view of the notion of robust preservation, providing a few contributions that don't appear in the above mentioned papers. Hereafter we draw an outline of the manuscript and highlights the main improvements with respect to previous work.

Chapter 2 studies classes of properties such as *Safety* or *Liveness* in terms of their topological characterization. This allows to abstract from both the information carried by events and the structure of execution traces,e.g., finite or infinite sequences, with or without dedicated symbols for termination or divergence.
We use the topological characterization to show that relational hyperproperties are not a redundant notion as advocated by Clarkson and Schneider [33, Section 2.4][3].
Another benefit of the topological approach adopted in this manuscript is the possibility of reasoning about *separability* properties of the trace models. For a concrete example, the adoption of topologies with weaker or stronger separability for the trace models adopted by CompCert or in [8], determines the possibility of distinguishing any two distinct executions, which is particularly interesting for reasoning about silent divergence,i.e., an infinite loop that does not produce externally observable events.

Chapter 3 describes a framework – based on Galois connections – that then we use to relate properties and hyperproperties of source and target when the two languages don't share the same notion of events or traces.
The main contribution of this chapter consists of Lemma 8 and Lemma 9, which makes rigorous the intuition that the criteria from [8, 10, 7] share the same formal structure. This in turn leads to a more concise presentation than, for example, the verbose classification from [8, Fig 1], and gives chance for future improvement of the Coq mechanization accompanying [8, 10] in which property-free and property-full formulations are (first guessed and then) proved equivalent for every single criterion.

Chapter 4 explores compiler correctness in terms of preservation of trace properties, hyperproperties, and relational hyperproperties from Chapter 2.

We generalize some results from [10, 7] by removing non necessary hypotheses,e.g., Theorem 6 generalizes Theorem 3.8 from [7] and requires no hypothesis on the source and target maps used to interpret source properties in the target and target ones in the source. We rigorously prove (Lemma 15) folklore equivalences of a few criteria for compiler correctness,e.g., we show that the preservation of trace properties also ensures that of subset-closed hyperproperties and relational trace properties.
We identify (Theorem 5) sufficient conditions under which the preservation of the only safety properties implies the one of arbitrary trace properties.
Section 4.5 provides a crisper presentation of the results on the preservation of abstract noninterference from Abate et al. [10, 7]. Finally, we briefly discuss (Section 4.6)

[3] In [8, Appendix E.4] we justified the introduction of relational hyperproperties for their application in secure compilation – they lead to criteria for robust preservation that are strictly stronger than those possible considering only hyperproperties – but not as mathematical objects themselves.

how the presented framework allows defining *reflection* of classes of properties through compilation.

Chapter 5 explores secure compilation in terms of preservation of trace properties, hyperproperties, and relational hyperproperties from Chapter 2.

Section 5.4 collects the main criteria for robust preservation and discusses the relation between a few of them. In Section 5.5 we investigate the relation between the notion of robust preservation and fully abstract compilation. We come to conclusions that are similar to those of Abate, Busi, and Tsampas [6], i.e., the need for extra conditions for fully abstract compilers to achieve also robust preservation of hyperproperties. However, here, we work in a simpler setting and don't need to assume that observational equivalence coincides with trace equality.

Chapter 6 concludes and provides hints on possible lines of future work, and the various appendices provide technical details omitted in the corresponding chapters.

Chapter 2

Traces and Hyperproperties

In this chapter we describe the mathematical language and structures we use to define security properties of programs. In Section 2.1 we informally recall the theory of hyperproperties by Clarkson and Schneider [33], who consider execution traces as *infinite* sequences of states of a system. Existing verified compilers such as CompCert or CakeML adopt trace models with both finite and infinite traces, thus allowing to more naturally distinguish an execution that terminates – with a value or an error – from an infinite unproductive loop (see Section 2.5). Therefore, in Section 2.4 we study a theory that abstracts the structure of the execution traces, and define classes such as *Safety* directly as their topological characterization. The canonical definition of *Safety* e.g., by Clarkson and Schneider [33], is recovered in the appropriate trace model,i.e., of traces as infinite sequences.

2.1 Why Hyperproperties?

The goal of program verification is to increase the confidence on the fact that the executions of some program satisfy desirable properties such as termination, or do not violate some others, e.g., no memory address is accessed before being initialized. Formal proofs of similar facts require a formal language in which to express properties of programs executions. In the theory of *trace properties* [71, 72, 14, 107] the executions of a program – or system – are modeled as the sequences of the states encountered by the system during its execution (*traces*). Trace *properties* are sets of traces: every property containing all and only the traces for which the property holds. Among all trace properties there are two sub-classes of particular interest, Safety and Liveness properties, we recall their intuitive meaning hereafter.

Safety properties specify that "something bad never happens" [72]. Illustrative examples of safety properties given by Alpern and Schneider [14] are *mutual exclusion* and *deadlock freedom* where the "bad things" proscribed are the simultaneous execution of two distinct processes and the deadlock, respectively. Moreover, it is possible to regard *partial correctness* as the safety property where the bad thing is termination in a state that does not satisfy a certain post-condition, despite having started execution in a state satisfying a given pre-condition.
Safety properties can be proven with an invariance argument [73, 15], they also play a central role in runtime verification as they correspond to the properties whose violation is monitorable and can no longer be restored [103, 43].

Liveness properties specify that "something good eventually happens" [72, 14]. Illustrative examples of liveness properties given by Alpern and Schneider [14] are *starvation freedom*, *termination* and *guaranteed service*, where the "good things" prescribed are respectively progress in the execution, completion of a final instruction and the delivery of a service any time it's required.

Liveness properties allows for the prescribed "good thing" to occur after any partial execution, a proof of this fact may need a well-founded argument [15]. Monitoring liveness is, in general, not possible [43].

Interestingly, every trace property can be written as the set-theoretical intersection of a safety and a liveness properties [14], following the intuition that a system is specified once it's clear what it should and what it should not do. The classes of Safety and Liveness can therefore be intended as building blocks for all trace properties. Efficient verification techniques for these two classes allow to verify arbitrary trace properties, at least in principle.

Expressiveness of trace properties is however too limited for interesting *security* notions, as they are (the extension of) predicates on one single trace. The "good thing" prescribed by liveness properties and the "bad thing" proscribed by safety ones, refer to one single execution at a time. The well-known policy of *noninterference*[1] is not a trace properties as it requires to compare two executions of the same system of which one uses a certain information not available in the other.

Hyperproperties can specify properties of multiple executions of a system as they consists of sets of sets of traces, or sets of trace properties. Many hyperproperties are of interest for information-flow control, e.g., noninterference, observational determinism, quantitative information flow. Hyperproperties also captures properties of algebraic codes, such as maximum Hamming distance [53], or properties of multiple threads such as symmetric mutual exclusion [53], and can *quantify* the variation in outputs for a given variation of the inputs in the outputs, e.g., in robust cleanness [47, 34].

The theory of hyperproperties is a useful extension of the theory of trace properties, the "bad" and "good" things from Safety and Liveness that refer to a single partial execution can be generalized to an arbitrary – but finite – number of partial executions, leading to the definition of the classes of *Hypersafety* and *Hyperliveness*. Clarkson and Schneider [33] show that also every hyperproperty can be expressed as the intersection of a hypersafety and a hyperliveness, so that verification of arbitrary hyperproperties can be reduced to the verification for the classes of *Hypersafety* and *Hyperliveness*. Verification of a hypersafety can sometimes be reduced to verification of a safety property of the self composition of a system [68, 20], or by decomposition or partitioning of the space of traces of a program into "secret independent" branches [18]. Furthermore, monitoring [40, 52] and testing [62, 65] are possible for an interesting subset of *Hypersafety*. Verification of *Hyperliveness* seems to be a more complicated matter. Existing work in this area exploit bounded synthesis [51], and a *game theoretic* interpretation of hyperliveness [34], but they are often limited to proper subsets of *Hyperliveness*.

2.2 Why More Abstract?

The notion of *state* adopted in the model by Clarkson and Schneider [33] is left abstract in purpose, so that *"by employing rich enough notions of state, this model can encode other representations of execution"* [33, Section 2]. For example, states may encode input and output actions, such as system calls and accesses to the memory.

[1] noninterference original formulation [60, page 15]: *"what one group of users does using a certain ability has no effect on what some other group of users sees"*

On the other hand, Clarkson and Schneider [33] bound traces to be *infinite* sequences of states and termination is modeled through infinite stuttering on a same state, while existing projects for verified compilers such as CompCert [74] and CakeML [115] allow traces to be both finite and infinite sequences. This choice expresses more naturally computations that produce only a finite number of *observable* events, such as executions that terminate with a value or an error or entered in an infinite and unproductive loop, and ease reasoning on them by induction. However, what does it mean "something good eventually happens" for a computation that is terminated or failed? How should one define the classes of Safety and Liveness and how "lift" them to Hypersafety and Hyperliveness? And in general, how the choice of the trace model affect classes of trace properties and hyperproperties?

We are not the first to investigate similar questions, as for example Rosu [103] made rigorous the intuition of safety properties in the models of only infinite trace, of finite only traces and of both finite and infinite traces, providing also equivalent characterizations. Successively, Pasqua and Mastroeni [90] made rigorous also the intuition of liveness and then studied the corresponding hyperproperties for the above mentioned models. As a result, in all the three models, every arbitrary (hyper)property is the intersection of a (hyper)safety and a (hyper)liveness [90, Propositions 4, 5, 6].
All the mentioned results, and many others regarding monitorability [43], do not depend on the structure of the traces but on the structure of trace properties, that form a topological space in which Safety and Liveness are the closed and the dense sets respectively.

Adopting a purely topological point of view allows us to reason about trace properties and hyperproperties without bothering about the detail of the trace model, with a particular benefit in Chapter 4 and Chapter 5 where we are interested in compilers that preserve classes of hyperproperties defined on possibly different trace models. For sake of completeness we recall the necessary mathematical notions hereafter.

2.3 Essential Recalls of Topology

This section collects all the elementary topology that we need in the rest of the work, examples are often given in later sections, so that this section can in principle be safely skipped by a reader familiar with topology. We believe however, it may still be useful to skim the paragraph on the *lower* Vietoris topology that is defined on arbitrary subsets and not only the closed ones as customary in mathematical literature [79].

A Topological space or Topology on a set X is a subset of $\wp(X)$,i.e., $\mathcal{O} \subseteq \wp(()X)$ that is closed under arbitrary unions and finite intersection. In particular $\emptyset, X \in \mathcal{O}$. The elements of $\mathcal{O}$ are called *open sets*. A base for $\mathcal{O}$ is a collection of open sets $\mathcal{B} \subseteq \mathcal{O}$ such that every other open set can be written as (arbitrary) union of elements in $\mathcal{B}$. A subbase for $\mathcal{O}$ is a collection of open sets $\mathcal{B} \subseteq \mathcal{O}$ such that every other open set can be written as (arbitrary) union of *finite* intersections of elements in $\mathcal{B}$.

Closed and Dense sets [43]. Let X be a set and $\mathcal{O} \subseteq \wp(X)$ a topology on X.

- $C \in \wp(X)$ is *closed iff* it's complement is open, $X \setminus C \in \mathcal{O}$. It follows that the set of closed sets is closed under arbitrary intersections and finite unions. A set that is both closed and open is called *clopen*. For any $\pi \in \wp(X)$, the topological *closure* of π – written $\overline{\pi}$ – is the smallest closed set that includes π,i.e., $\overline{\pi} \equiv \bigcap \{C \mid C \texttt{ is closed } \wedge \pi \subseteq C\}$. A point $x \in X$ is a *closure point* for $\pi \subseteq X$ *iff* $\forall A \in \mathcal{O}.\ x \in A \Rightarrow \pi \cap A \neq \emptyset$, or equivalently *iff* $x \in \overline{\pi}$.

- D is *dense iff* it intersects every nonempty open set,

$$\forall A \in \mathcal{O}.\ A \neq \emptyset \Rightarrow A \cap D \neq \emptyset$$

  or equivalently if its closure coincides with the whole X, $\overline{D} = X$. We notice explicitly that Y is both closed and dense if and only if $Y = X$.

We provide examples of open, closed and dense sets in Section 2.4. We also notice explicitly that in order to define a topology it suffices to define its open sets, hence a base or a subbase of the topology, or its closed sets.

Theorem 1 (Decomposition (Theorem 1 in [14]))**.** *Let $\mathcal{O} \subseteq \wp(X)$ be a topology on X. For any $\pi \in \wp(X)$ there exists a closed set C and a dense set D such that*

$$\pi = C \cap D$$

Proof. (Sketch) The topological closure of π, namely $C = \overline{\pi}$ and $D = X \setminus (\overline{\pi} \setminus \pi)$ are respectively a closed and a dense set and their intersection is π. □

The Lower Vietoris construction lifts a topology $\mathcal{O}$ on X to a topology on $\wp(X)$.

Definition 1 (Lower Vietoris ([33, 112]))**.** Let $\mathcal{O} \subseteq X$ be a topology on X. A subbase for $\mathcal{V}_{low}(\mathcal{O})$ – the lower Vietoris topology on $\mathcal{O}$ – is

$$\mathcal{B} = \{\langle A \rangle \ \mid \ A \in \mathcal{O}\}$$

where $\langle A \rangle = \{\pi \in \wp(X) \ \mid \ \pi \cap A \neq \emptyset\}$. □

Remark 1 (Characterization of Subset Closed [33])**.** $Closed_{\subseteq}$ is the class of hyperproperties that is closed under subsets,

$$\forall H.\ H \in Closed_{\subseteq} \iff (\forall \pi.\ \pi \in H \Rightarrow (\forall \pi'.\ \pi' \subseteq \pi \Rightarrow \pi' \in H)$$

it's immedaite to show that $H \in Closed_{\subseteq}$ *iff* it is an arbitrary union of powersets of trace properties,i.e.,

$$H = \bigcup_{j \in J} \wp(\pi_j)$$

□

Remark 2 (Closed in Lower Vietoris [33])**.** Let H_C be a closed set of the $\mathcal{V}_{low}(\mathcal{O})$. Its complement is by definition an open set, $\wp(X) \setminus H_C \in \mathcal{V}_{low}(\mathcal{O})$, and therefore can be written as arbitrary union of finite intersections of elements of the subbase $\mathcal{B}$ from Definition 1,

$$\wp(X) \setminus H_C = \bigcup_{\substack{k \in \mathbb{N}, \\ i_1 \ldots i_k \in I}} \langle A_{i_1} \rangle \cap \ldots \cap \langle A_{i_k} \rangle$$

for a set of indices I such that $\forall i \in I.\ A_i \in \mathcal{O}$. From $H_C = \wp(X) \setminus (\wp(X) \setminus H_C)$, we deduce that

$$H_C = \wp(X) \setminus \bigcup_{\substack{k \in \mathbb{N}, \\ i_1 \ldots i_k \in I}} \langle A_{i_1} \rangle \cap \ldots \cap \langle A_{i_k} \rangle$$

and by De Morgan laws

$$H_C = \bigcap_{\substack{k \in \mathbb{N}, \\ i_1 \ldots i_k \in I}} (\wp(X) \setminus \langle A_{i_1} \rangle) \cup \ldots \cup (\wp(X) \setminus \langle A_{i_k} \rangle)$$

Notice that for $A \in \mathcal{O}$, $\wp(X) \setminus \langle A \rangle = \{\pi \in \wp(X) \mid \pi \cap A = \emptyset\} = \wp(X \setminus A)$, that is the set of all subsets of a closed set. It follows that H_C can be written as following

$$H_C = \bigcap_{\substack{k \in \mathbb{N}, \\ j_1 \dots j_k \in J}} \wp(C_{j_1}) \cup \ldots \cup \wp(C_{j_k}) \tag{2.1}$$

for a set of indices J such that for any $j \in J$, C_j is a closed of $\mathcal{O}$. The viceversa is also true because for a closed of $\mathcal{O}$, $\wp(C)$ is a closed of $\mathcal{V}_{low}(\mathcal{O})$ and finite union and arbitrary intersections of closed sets are closed sets. It follows that all and only the closed sets in the lower Vietoris topology can be written as in Equation (2.1). □

Remark 3 (Viteories closed are $Closed_{\subseteq}$)**.** From the characterizations above it follows immediately that every hypersafety is also subset closed. □

Definition 2 ($\mathbf{T}_1$ spaces)**.** A topological space on X $\mathcal{O} \subseteq \wp(X)$ is said to be $\mathbf{T}_1$ if one of the following equivalent conditions hold:

1. for any $x \in X, \{x\} = \cap \{A \mid A \in \mathcal{O} \wedge x \in A\}$
2. for any $x, y \in X$ such that $x \neq y$ there exist open sets $A_x, A_y \in \mathcal{O}$ such that
   - $x \in A_x$ and $y \notin A_x$ and
   - $y \in A_y$ and $x \notin A_y$

□

Definition 3 ($\mathbf{T}_2$ spaces)**.** A topological space on X $\mathcal{O} \subseteq \wp(X)$ is said to be a *Hausdorff* space or $\mathbf{T}_2$ *iff* for any $x, y \in X$ such that $x \neq y$ there exist open sets $A_x, A_y \in \mathcal{O}$ such that $x \in A_x, y \in A_y$ and $A_x \cap A_y = \emptyset$. Notice that a $\mathbf{T}_2$ space is also $\mathbf{T}_1$. □

The product of two topological spaces $\mathcal{O}(\mathrm{X})$ and $\mathcal{O}(\mathrm{Y})$ is the product in the category of topological spaces and continuous functions.

Definition 4 (Continuos function)**.** Let X, Y be two sets equipped with topologies $\mathcal{O}(X)$ and $\mathcal{O}(Y)$ respectively. A function $f : X \to Y$ is continuous – w.r.t the two topologies – *iff* the pre-image of every open set is an open set,

$$\forall A \in \mathcal{O}(Y).\ f^{-1}(A) \in \mathcal{O}(X).$$

□

Definition 5 (Product Topology)**.** The product topology of $\mathcal{O}(X)$ and $\mathcal{O}(Y)$, denoted by $\mathcal{O}(X) \times \mathcal{O}(Y)$ is the finest (i.e., smallest) topology on $X \times Y$ such that the projections $\mathtt{prj}_X : X \times Y \to X$ and $\mathtt{prj}_Y : X \times Y \to Y$ are continuous. □

2.4 Hyperproperties on Abstract Traces

As already mentioned, the set of all trace properties – for each one of the three models studied by Rosu [103] and Pasqua and Mastroeni [90] – can be equipped with a topology, where *Safety* and *Liveness* correspond to the closed and the dense sets respectively. We proved[2] the same topological characterization also for the trace

[2] See decomposition_safety_dense in the Coq development of [8]

model adopted in[8] and obtained by Theorem 1 that every trace property is the intersection of a safety and a liveness one.
Therefore, similarly to Diekert and Leucker [43], we take the topological characterizations as definition of *Safety* and *Liveness* and abstract from the structure of traces, that can now be considered elements of an arbitrary set.

Definition 6 (Abstract Traces, Properties and Hyperproperties)**.** For an arbitrary set T we call its elements traces and sometimes denote the set itself by Trace, $\wp(\mathit{Trace})$ is the set of properties of T, and $\wp(\wp(\mathit{Trace}))$ is the set of hyperproperties of T. □

Definition 7 (Abstract Safety and Liveness)**.** For a set of traces Trace and a topology $\mathcal{O} \subseteq \wp(\mathit{Trace})$, $\mathit{Safety}_{\mathcal{O}}$ is the class of the closed sets of the $\mathcal{O}$ and $\mathit{Liveness}_{\mathcal{O}}$ is the class of the dense sets of $\mathcal{O}$. When the topology $\mathcal{O}$ is clear from the context we simply write *Safety* and *Liveness*. □

We show how the canonical definitions of *Safety* an *Liveness* can be recovered when considering the Plotkin topology [33].

Example 1 (Canonical Safety and Liveness[33])**.** Let Σ be a set with at least two distinct elements, $|\Sigma| \geq 2$. A trace t is an infinite string on Σ,i.e., $\mathit{Trace} = \Sigma^{\omega}$. We denote with $m \in \Sigma^{*}$ finite prefixes of traces, and write $m \leq t$ to denote that m is a prefix of t. The Plotkin topology $\mathcal{O} \subseteq \wp(\mathit{Trace})$ is defined by its open sets being the so called *observable* trace properties: $\forall \pi \in \wp(\mathit{Trace}).\ \pi \in \mathcal{O}$ iff

$$\forall t \in \pi.\ \exists m \leq t.\ \forall t'.\ m \leq t' \Rightarrow t' \in \pi \tag{2.2}$$

A base (that is also a sub-base) for this topology is $\{\uparrow m \mid m \in \Sigma^{*}\}$ where $\uparrow m = \{t \in \Sigma^{\omega} \mid m \leq t\}$. Therefore every open set is arbitrary union of finite intersections of sets of the form $\uparrow m$.
In such a topology Safety (closed sets) and Liveness (dense sets) have the following equivalent formulation [14]:

$$\pi \in \mathit{Safety} \iff \forall t \notin \pi.\ \exists m \leq t.\ \forall t'.\ m \leq t' \Rightarrow t' \notin \pi \tag{2.3}$$

$$\pi \in \mathit{Liveness} \iff \forall m.\ \exists t \in \pi.\ m \leq t \tag{2.4}$$

□

The above formulae 2.3 and 2.4 can be used to define the classes pf *Safety* and *Liveness* also in the trace models of only finite and both finite and infinite traces [90].

In the following example we consider traces made of (finite or infinite) sequences of reduction steps in a typed lambda calculus and show that the well-known *type safety* can be written and intersection of a safety and a liveness property.

Example 2 (Progress and Preservation)**.** Consider a typed language,e.g., the simply typed lambda calculus (seee.g., these course notes https://www.cs.cmu.edu/ fp/courses/15312-f04/handouts/06-safety.pdf), and let $\mathcal{E}$ be the set of well-typed expressions of the language.
We assume that such a set includes a set $\mathcal{V}$ of values with their base types. Consider traces to be finite and infinite strings of well-typed expressions $\mathit{Trace} = \mathcal{E}^{*} \cup \mathcal{E}^{\omega}$ and we write $t \in \mathit{Trace}$ as $t = ((e_0, \tau_0), (e_1, \tau_1), \ldots (e_n, \tau_n), \ldots)$ where for every i, $\vdash e_i : \tau_i$. Notice that Trace contains all the sequences of reductions for any expression, but – together with the two trace properties we are going to define – also many *spurious* traces,e.g., $t = ((e_0, \tau_0), (e_1, \tau_1), \ldots (e_n, \tau_n), \ldots)$ with $e_i \not\rightarrow e_{i+1}$ for some i.

The well-known *type safety* is usually intended as the conjunction of progress and preservation, that we recall below and then formalize as trace properties, and type safety will be given as the set theoretical intersection of the them.

Preservation: if $\vdash e : \tau$ and $e \mapsto e'$ then $\vdash e' : \tau$

$$\pi_{Pres} = \{\ t = ((e_0, \tau_0), (e_1, \tau_1), \dots (e_n, \tau_n), \dots) \in \mathit{Trace}\ \mid\ \forall i.\ (e_i \mapsto e_{i+1}) \Rightarrow \tau_{i+1} = \tau_i\}$$

It's immediate that preservation holds iff for every well typed expression e, the sequence of its possible reductions is in π_{Pres}. Moreover $\pi_{Pres} \in \mathit{Safety}$, where the class *Safety* has the same formal definition as in Example 1. To prove this claim, let $t = ((e_0, \tau_0), (e_1, \tau_1), \dots (e_n, \tau_n), \dots) \notin \pi_{pres}$,i.e., there exists some i such that $(e_i \mapsto e_{i+1})$ and $\tau_{i+1} \neq \tau_i$. Then the prefix $m = ((e_0, \tau_0), (e_1, \tau_1), \dots (e_i, \tau_i), (e_{i+1}, \tau_{i+1})) \leq t$ is such that for any t'. $m \leq t' \Rightarrow t' \notin \pi_{Pres}$.

Progress: if $\vdash e : \tau$ then either $e \in \mathcal{V}$ or there exists e' such that $e \mapsto e'$

$$\pi_{Fin} = \{\ t = ((e_0, \tau_0), (e_1, \tau_1), \dots (e_n, \tau_n)) \in \mathcal{E}^*\ \mid\ e_n \in \mathcal{V}\}$$

$$\pi_{Inf} = \{\ t = ((e_0, \tau_0), (e_1, \tau_1), \dots (e_n, \tau_n) \dots) \in \mathcal{E}^\omega\ \mid\ \forall i.\ e_1 \mapsto e_{i+1}\}$$

and $\pi_{Prog} = \pi_{Fin} \cup \pi_{Inf}$. Once again it's immediate that progress holds iff for every well typed expression e, the sequence of its possible reductions is in π_{Prog}. We show that $\pi_{Fin} \in \mathit{Liveness}$ and deduce that also $\pi_{Prog} \in \mathit{Liveness}$ because the union of a dense set (liveness property) with another set is still a dense set (liveness property). The property $\pi_{Fin} \in \mathit{Liveness}$ because it's possible to append a final (typed) value (v, τ) to any $m \in \mathcal{E}^*$, so that $(v, \tau) \in \pi_{Fin}$.
We can conclude that a language enjoys type safety – i.e., the conjunction of progress and preservation – if it satisfies $\pi_{Pres} \cap \pi_{Prog}$ the intersection of a safety and a liveness. □

Definition 8 (Abstract Hypersafety and Hyperliveness)**.** For a set of traces *Trace* and a topology $\mathcal{O} \subseteq \wp(\mathit{Trace})$, $\mathit{Hypersafety}_{\mathcal{O}}$ is the class of the closed sets of $\mathcal{V}_{low}(\mathcal{O}) \subseteq \wp(\wp(\mathit{Trace}))$, the lower Vietoris topology on $\mathcal{O}$ and $\mathit{Hyperliveness}_{\mathcal{O}}$ is the class of the dense sets of $\mathcal{V}_{low}(\mathcal{O})$. When the topology $\mathcal{O}$ is clear from the context we simply write *Hypersafety* and *Hyperliveness*. □

The correspondence of *Hypersafety* and *Hyperliveness* with closed and dense sets, immediately implies the following.

Corollary 1. *Every hyperproperty can be written as intersection of a hypersafety and a hyperliveness.*

Definition 9 ($k{-}\mathit{Closed}_{\subseteq}$)**.** A hyperproperty $H \in \wp(\wp(\mathit{Trace}))$ is k-subset-closed or in $k{-}\mathit{Closed}_{\subseteq}$ *iff* it's violation can be witnessed by at most k traces,

$$\forall \pi \notin H.\ \exists \pi_k \subseteq \pi.\ (\pi_k \notin H \wedge |\pi_k| \leq k)$$

□

In Section 3.4 we will present the security relevant *Hypersafety* known as (abstract) noninterference, while hereafter we show the formulation of *Hypersafety* and *Hyperliveness* in the trace model from Example 1. The formulae 2.5 and 2.6 can be used to define the classes of *Hypersafety* and *Hyperliveness* also in trace model with finite only traces or finite and infinite ones.

Example 3 (Canonical Hypersafety and Hyperliveness)**.** Following Clarkson and Schneider [33] we denote with M finite sets of finite prefixes,i.e., $M \in \wp_{fin}(\Sigma^*)$ and for $\pi \in \wp(\mathit{Trace})$, we write $M \leq \pi$ for $\forall m \in M\ \exists t \in \pi.\ m \leq \pi$.
In the lower Vietoris topology on $\wp(\mathit{Trace})$ from Example 1 the open sets have the following equivalent formulation[33, Section 6] $\forall H \in \wp(\wp(\mathit{Trace}))$. $H \in \mathcal{V}_{low}(\mathcal{O})$ iff

$$\forall \pi \in H.\ \exists M \leq \pi.\ \forall \pi'.\ M \leq \pi' \Rightarrow \pi' \in H$$

In such a topology Hypersafety (closed sets) and Hyperliveness (dense sets) have the following equivalent formulation:

$$H \in \mathit{Hypersafety} \iff \forall \pi \notin H.\ \exists M \leq \pi.\ \forall \pi'.\ M \leq \pi' \Rightarrow \pi' \notin H \tag{2.5}$$

$$H \in \mathit{Hyperliveness} \iff \forall M.\ \exists \pi \in H.\ M \leq \pi \tag{2.6}$$

□

2.4.1 Relational Hyperproperties

So far we used sets of traces to specify predicates of single executions, and sets of sets of traces to specify predicates on multiple executions, but in both cases we referred to a single system. According to Clarkson and Schneider [33, Section 2.4], hyperproperties – sets of sets of traces – suffice to also relate executions of multiple systems, for example to show that two systems are equivalent, that one refines the other or to compare their efficiency. For relational specifications one can consider the *product* of two systems and express the intended relation on the executions of the two system, with a set (of sets) of pairs of execution traces (t_1, t_2), where t_1 is an execution of the first system and t_2 of the second. We extend the intuition built in [8] and argue that an approach based on products of systems is not satisfactory, for both practical and theoretical reasons.

Practical issues may arise when trying to reduce the verification of a relational property to the verification of a property of the product of two programs. Product programs is not always possible nor unique [19]. In the latter case, the *specification* of a relational property as a property of pairs of traces, may not suggest in which way two programs should be combined in a product to ease, or even just make possible, the verification.

Theoretical issues come from the fact that safety properties of the product of two systems are more expressive that the product of safety properties of each single system. We provide an example in the trace model of Clarkson and Schneider [33].

Example 4 (Relational Hypersafety are not product of Hypersafety)**.** We propose a definition for the class of $2\mathit{relHypersafety}$, that generalizes the definition of $2\mathit{relSafety}$ by Abate et al. [8]. Our definition is given for the model of Clarkson and Schneider [33], of traces as infinite sequences of elements of a set of states Σ, containing at least 2 distinct states, $|\Sigma| \geq 2$.

A set of pairs of traces properties $\mathcal{R}_H \in \wp(\mathit{Trace})^2$ is a $2\mathit{relHypersafety}$ *iff* it specifies some "bad thing", that irremediably violates a certain relation between two trace properties, never happens. Rigorously, $\mathcal{R}_H \in 2\mathit{relHypersafety}$ *iff*

$$\forall(\pi_1, \pi_2).\ (\pi_1, \pi_2) \notin \mathcal{R}_H \Rightarrow \quad \exists M_1\ \exists M_2.\ M_1 \leq \pi_1 \wedge M_2 \leq \pi_2 \wedge \forall(\pi_1', \pi_2').\ (M_1 \leq \pi_1' \wedge M_2 \leq \pi_2') \Rightarrow (\pi_1', \pi_2') \notin \mathcal{R}_H$$

Fix a finite set of finite trace prefixes $M \in \wp_{fin}(\Sigma^*)$, $M \neq \emptyset$ and consider the following relational hyperproperties, that intuitively specifies that a certain "bad thing" – specified by M – does not happen *simultaneously* for two systems,

$$\mathcal{R}_H = \{(\pi_1, \pi_2) \mid M \not\leq \pi_1 \vee M \not\leq \pi_1\}$$

- $\mathcal{R}_H \in 2relHypersafety$ indeed if $(\pi_1, \pi_2) \notin \mathcal{R}_H$ then both $\pi_1 \leq M$ and $\pi_2 \leq M$. The pair (M, M) satisfies the definition of $2relHypersafety$ given above.
- $\mathcal{R}_H$ is not the product of two hypersafety. In order to prove it, assume $\mathcal{R}_H = H_1 \times H_2$ with both $H_1, H_2 \in Hypersafety$. We show that $H_1 = H_2 = \{\pi \mid \pi \in \wp(Trace)\}$, the hyperproperty that contains all trace properties.

  Let $\pi \in \wp(Trace)$ be an arbitrary trace property, if $M \not\leq \pi$ then $(\pi, \pi') \in \mathcal{R}_H = H_1 \times H_2$ for any π', and therefore $\pi \in H_1$. If $M \leq \pi$ then $(\pi, \emptyset) \in \mathcal{R}_H = H_1 \times H_2$ as $M \not\leq \emptyset$, and $\pi \in H_1$. We showed that H_1 contains all trace properties and by symmetry also H_2 does. It follows that $\mathcal{R}_H = H_1 \times H_2$ contains all pairs of trace properties, that is clearly false.

□

The reason for which Example 4 is possible is, once again, topological. While the product of closed sets is a closed set in every topological space, it is a well-known fact [70] that closed sets of the product topology – $2relHypersafety$ – in general are not the product of two closed sets of the underlying topology,i.e., $Hypersafety$.

Definition 10 (Abstract Relational)**.** Let $Trace$ be a set and $\mathcal{O} \subseteq \wp(Trace)$ a topology on $Trace$. For a $k \in \mathbb{N} \cup \omega$:

- $\wp(Trace^k)$ is the set of $k-$relational properties of $Trace$. The set $krelSafety$ of $k-$relational safety properties is the set of closed sets in the topological product of $\mathcal{O}$ by itself, k times. The set $krelLiveness$ of $k-$relational liveness properties is the set of dense sets in the topological product of $\mathcal{O}$ by itself, k times.
- $\wp(Trace)^k$ is the set of $k-$relational hyperproperties of $Trace$. The set $krelHypersafety$ of $k-$relational hypersafety is the set of closed sets in the topological product of $\mathcal{V}_{low}(\mathcal{O})$ by itself, k times. The set $krelHyperlivenss$ of $k-$relational liveness properties is the set of dense sets in the topological product of $\mathcal{V}_{low}(\mathcal{O})$ by itself, k times.

□

2.5 Topologies for the Trace Model à la CompCert

In this section we present a few topologies for a concrete trace model to which we often refer to in the next chapters, the one adopted in the CompCert project. In CompCert[3] each event in Σ represents either:

- A system call (e.g. an input/output operation), recording the name of the system call, its parameters, and its result.
- A volatile load from a global memory location, recording the chunk and address being read and the value just read.

[3] https://compcert.org/doc/html/compcert.common.Events.html

- A volatile store to a global memory location, recording the chunk and address being written and the value stored there.
- An annotation, recording the text of the annotation and the values of the arguments.

Programs can either converge – to a value, denoted by $\bullet$, or due to an undefined behavior denoted by $\lightning$ – or diverge, denoted by $\circlearrowleft$ when no event is produced (*silent* divergence) or productively producing infinitely many events. Therefore the set of traces[4] is the following

$$\mathit{Trace} = \{m\bullet \mid m \in \Sigma^*\} + \{m\lightning \mid m \in \Sigma^*\} + \{m\circlearrowleft \mid m \in \Sigma^*\} + \Sigma^\omega$$

The (open sets of the) topologies can be defined by Equation (2.2) and similarly *Safety* and *Liveness* by the formulae formulae 2.3 and 2.4, but their expressiveness depends on what the finite prefixes allow to observe. Every choice of prefixes we present hereafter gives rise to a different topology, with more or less expressive *Safety* and *Liveness*.

Example 5 (Prefixes as lists)**.** When every prefix is a simple list of event, $m \in \Sigma^*$ they don't even allow to observe termination of a program that actually terminated. The trace property specifying that the execution did not encounter an undefined behavior is not a safety one,

$$\pi_{(\neg\lightning)}\{t \mid \forall m \in \Sigma^*.\ t \neq m\lightning\} \notin \mathit{Safety}$$

□

Example 6 (Prefixes as lists with $\bullet$ and $\lightning$)**.** This is the trace model adopted by Abate et al. [8], with a richer notion of finite prefixes, that allows to observe termination and undefined behavior when these actually occurred. Finite prefixes are drawn from the following set,

$$\Sigma^*_{\bullet,\lightning} = \Sigma^* \cup \{m\bullet \mid m \in \Sigma^*\} \cup \{m\lightning \mid m \in \Sigma^*\}$$

and the prefix relation is extended accordingly, for a trace $t \in \mathit{Trace}$,

- $m\bullet \in \Sigma^*_{\bullet,\lightning}$ is a prefix of t *iff* $t = m\bullet$ and
- $m\lightning \in \Sigma^*_{\bullet,\lightning}$ is a prefix of t *iff* $t = m\lightning$.

In this trace model,

$$\pi_{(\neg\lightning)}\{t \mid \forall m \in \Sigma^*.\ t \neq m\lightning\} \in \mathit{Safety}.$$

Interestingly, in this trace model, the dense sets,i.e., the liveness properties, must include all traces of the form $m\bullet$ and $m\lightning$[5], this because the finite prefixes $m\bullet$ and $m\lightning$ have as only continuation themselves, and liveness properties allows a continuation for any possible finite prefix. The obtained topology is not $\mathbf{T}_1$ (and hence not $\mathbf{T}_2$) because singletons of the form $\{m\circlearrowleft\}$ are not closed,i.e., they are not safety properties. □

[4] Called behaviors in CompCert `https://compcert.org/doc/html/compcert.common.Behaviors.html`

[5] see all_fin_in_all_liv in the Coq development for [8]

Example 7 (X-Prefixes)**.** A finer topology on *Trace* is given by consider the following set of "x-prefixes" (see XPrefix.v in the Coq development of [8])

$$\Sigma_X^* = \Sigma^* \cup \{m\bullet \mid m \in \Sigma^*\} \cup \{m\lightning \mid m \in \Sigma^*\} \cup \{m\circlearrowleft \mid m \in \Sigma^*\}$$

and with the prefix relation extended accordingly, for a trace $t \in \mathit{Trace}$,

- $m\bullet \in \Sigma_X^*$ is a prefix of t *iff* $t = m\bullet$,
- $m\lightning \in \Sigma_X^*$ is a prefix of t *iff* $t = m\lightning$ and
- $m\circlearrowleft \in \Sigma_X^*$ is a prefix of t *iff* $t = m\circlearrowleft$

In this trace model,

$$\pi_{(\neg\lightning)}\{t \mid \forall m \in \Sigma^*.\ t \neq m\lightning\} \in \mathit{Safety},$$

but also the trace properties that specify a program does not silently diverge is safety,

$$\pi_{(\neg\circlearrowleft)}\{t \mid \forall m \in \Sigma^*.\ t \neq m\circlearrowleft\} \in \mathit{Safety}.$$

The choice of having $m\circlearrowleft$ as a finite prefix allows us to use observe (with a finite prefix) silent divergence, when this actually happened.
Notice that the topology obtained using "x-prefixes" is $\mathbf{T}_2$, we will use this information in Section 5.4. □

2.6 Related Work

Topology has played an important role in computer science since much before the work on hyperproperties by Clarkson and Schneider [33]. Diekert and Leucker [43] show in their tutorial how the topological formulation of the notion of monitorability allows to build monitor for a class of trace properties much wider than *Safety*.

From the program logic community, Smyth [112] argues:

> *"Such concepts as "specifications", "predicate transformer", and "nondeterminism" can be greatly illuminated by being formulated in topological terms.*
>
> *. . . The removal of the restriction to flat domains should permit the development of more adequate programming logics. ".*

In particular Smyth [112] believes program logics can benefit of the use of the so-called *pointless topologies* [56, 119], and is therefore interested in more abstract space then the one we presented in this chapter. We are, instead, strongly interested in our points,i.e., execution traces, as some existing proof techniques for secure compilation strongly rely on the information collected along single execution traces or single finite prefixes [42, 93, 11, 116].

Chapter 3

Frameworks for Abstraction

The main purpose of this chapter is to provide an abstract definition of robust preservation that generalizes the ones introduced in [10, 8] to arbitrary posets. The behaviors of programs – sets of traces – are abstracted to element of arbitrary posets thus providing a unique formal definition for the robust preservation of (classes of) trace properties, hyperproperties, relational hyperproperties.
To get to such an abstract definition in Section 3.5, we first recall definitions and simple facts about *Galois connections*, that we use to relate (hyper)properties of the source language with the ones of the target.
We then recall the notion of *upper closure operator* in Section 3.3. Closure operators on a complete lattice, define a Galois connection between the lattice itself and the set of fixed points of the operator. We use closure operators in the next chapters to:

(i) define the robust preservation of *Safety*, that is defined as set of fixed points of a closure operator and

(ii) show how correct compilers preserve the notion of *abstract noninterference* by Giacobazzi and Mastroeni [58] recalled in Section 3.4.

3.1 Essential Recalls of Galois Connections

This section collects standard definition and results, and can be safely skipped by a reader familiar with the works by Cousot and Cousot [36] or Melton, Schmidt, and Strecker [78].

Definition 11 (Galois Connections (Definition 5.3.0.1 in [36]))**.**
Let $(A, \leq)$ and $(C, \sqsubseteq)$ be two posets. A pair of maps, $\alpha : C \to A$, $\gamma : A \to C$ is a Galois connection *iff* α and γ are monotone and they satisfy the following *adjunction law*: $\forall a \in A.\ \forall c \in C$

$$\alpha(c) \leq a \iff c \sqsubseteq \gamma(a)$$

A is sometimes referred to as the *abstract* domain and C as the *concrete* one. α is referred to as the *lower adjoint* and γ as the *upper adjoint*. We write $\alpha \leftrightarrows \gamma$ to denote that α and γ are respectively the lower and upper adjoint of a Galois connection. □

Lemma 1 (Characteristic Property of Galois Connections (Theorem 5.3.0.4 from [36]))**.**
Let $(A, \leq)$ and $(C, \sqsubseteq)$ be two posets. $\alpha : C \to A$ and $\gamma : A \to C$ are a Galois connection iff they are both monotone and:

i) $\forall c \in C.\ c \sqsubseteq \gamma \circ \alpha(c)$

ii) $\forall a \in A.\ \alpha \circ \gamma(a) \leq a$

Definition 12 (Insertions and Reflections (Proposition 1.2 in [78])).
Let $(A, \leq)$ and $(C, \sqsubseteq)$ be two posets and $\alpha : C \to A$ and $\gamma : A \to C$ a Galois connection.

- If i) from Lemma 1 holds with equality we say that $\alpha \leftrightarrows \gamma$ is a *reflection* and it's possible to show that this is equivalent to require surjectivity of γ (or injectivity of α).
- If ii) from Lemma 1 holds with equality we say that $\alpha \leftrightarrows \gamma$ is an *insertion* and it's possible to show that this is equivalent to require injectivity of γ (or surjectivity of α).

□

Remark 4 (Joins and Meets (Proposition 1.2 in [78])).
If $(A, \leq)$ and $(C, \sqsubseteq)$ are posets and $\alpha \leftrightarrows \gamma$, then α preserves joins and γ preserves meets. □

Lemma 2 (Uniqueness of Adjoints (Propositions 1.2 and 2.2 in [78])).
Let $(A, \leq)$ and $(C, \sqsubseteq)$ be two posets and $\alpha : C \to A$ and $\gamma : A \to C$ a Galois connection. Upper and lower adjoints uniquely determines each other, meaning that:

- *for every γ' that is an upper adjoint for α, $\gamma' = \gamma = \lambda a \in A.\ \bigcup \{c \mid \alpha(c) \leq a\}$*
- *for every α' that is a lower adjoint for γ, $\alpha' = \alpha = \lambda c \in C.\ \bigcap \{a \mid c \sqsubseteq \gamma(a)\}$*

Lemma 3 (Composition and Product of Galois Connections (Section 10 of [36])).
Composition and pointwise product of Galois connections are Galois connections. More in detail:

- *Let $(A, \leq_A)$ and $(B, \leq_B)$ and $(C, \leq_C)$ be posets, and $\alpha_1 \leftrightarrows \gamma_1$ and $\alpha_2 \leftrightarrows \gamma_2$ Galois connections between A and B and B and C respectively. The maps $\alpha_1 \circ \alpha_2 : C \to A$ and $\gamma_2 \circ \gamma_1 : A \to C$ are a Galois connection between A and C,i.e., $\alpha_1 \circ \alpha_2 \leftrightarrows \gamma_2 \circ \gamma_1$.*
- *Let $(A_1, \leq_{A_1})(A_2, \leq_{A_2}) \ldots, (A_k, \leq_{A_k})$ and $(C_1, \leq_{C_1})(C_2, \leq_{C_2}) \ldots, (C_k, \leq_{C_k})$ be posets and $\alpha_1 \leftrightarrows \gamma_1, \alpha_2 \leftrightarrows \gamma_2, \ldots, \alpha_k \leftrightarrows \gamma_k$ Galois connections. The maps*

  $$\alpha_1 \times \alpha_2 \times \ldots \times \alpha_k : (C_1 \times C_2 \times \ldots \times C_k, \dot{\leq_C}) \leftrightarrows (A_1 \times A_2 \ldots \times A_k, \dot{\leq_A}) : \gamma_1 \times \gamma_2 \ldots \times \gamma_k$$

  are a Galois connection between the pointwise products of abstract domains and the pointwise product of concrete domains (equipped with the pointwise ordering).

3.2 Relations and Galois Connections

In the scenario of Chapter 4 and Chapter 5, the execution of a program w.r.t source or target semantics may lead to the observation of events drawn by different alphabets. For an example, consider a target semantics in which programs get stuck because of a lack of some physical resource,e.g., memory in CakeML [115], and imagine that the source semantics abstract such a physical resource. It's reasonable to *relate* a target trace that corresponds to a computation that consumes all resources and gets stuck to a source trace that extends the execution as if the same resource were still

available. Therefore, Chapter 4 and Chapter 5 assume a relation between source and target traces is given. Such a relation corresponds bijectively to a Galois connection between (the complete lattices of) source and target trace properties, that we can use to interpret source properties into target ones and vice versa.

Definition 13 (Existential and Universal Images of a relation [55])**.**
Given a relation $\sim \subseteq S \times T$,

$$\sim_\exists : (\wp(S), \subseteq) \to (\wp(T), \subseteq) \qquad \sim^\forall : (\wp(T), \subseteq) \to (\wp(S), \subseteq)$$
$$\pi_S \mapsto \{t \mid \exists s \in \pi_S.\ s \sim t\} \qquad \pi_T \mapsto \{s \mid \forall t.\ s \sim t \Rightarrow t \in \pi_T\}$$

are called the *existential* and *universal* images of $\sim$, and they are the lower and upper adjoint of a Galois connection between the complete lattices of $(\wp(S), \subseteq)$ and $(\wp(T), \subseteq)$ ordered by set inclusion [55, Examples 2, (2)],i.e., $\sim_\exists \leftrightarrows \sim^\forall$. □

Theorem 2 (Relations $\cong$ Galois Connections (see Examples 2, (2) in [55]))**.**

The map that associates to each relation $\sim \subseteq S \times T$ the pair $\sim_\exists : \wp(S) \leftrightarrows \wp(T) : \sim^\forall$ is a bijection between relations on S and T and Galois connections between $\wp(S)$ and $\wp(T)$. Its inverse maps a Galois connection $\alpha : \wp(S) \leftrightarrows \wp(T) : \gamma$ to the relation $\sim^\gamma_\alpha \subseteq S \times T$, defined by

$$s \sim^\gamma_\alpha t \iff t \in \alpha(\{s\}).$$

We notice explicitly that the relation associated to a Galois connection only apparently depends exclusively on the lower adjoint α, the dependency on γ comes from uniqueness of adjoints (see Lemma 2).

Definition 14 (Converse Relation and Converse Adjoints)**.** The converse relation of $\sim \subseteq S \times T$ is the relation $\sim^\dagger \subseteq T \times S$ defined by

$$t \sim^\dagger s \iff s \sim t.$$

We call converse existential and converse universal images of $\sim$, the existential and universal images of $\sim^\dagger$, that are themselves a Galois connection by Theorem 2,

$$\sim^\dagger{}_\exists : (\wp(T), \subseteq) \leftrightarrows (\wp(S), \subseteq) : \sim^{\dagger\forall}.$$

We notice explicitly that $\sim$ coincides with the converse of its converse relation, $(\sim^\dagger)^\dagger = \sim$, $(\sim^\dagger)^\dagger{}_\exists = \sim_\exists$ and $(\sim^\dagger)^{\dagger\forall} = \sim^\forall$. □

Definition 15 (Functionality and Totality of a relation)**.**
Let $\sim \subseteq S \times T$, we say that $\sim$ is

- *$T-$functional iff* $\forall t \in T. \forall s_1, s_2 \in S.\ (s_1 \sim t \wedge s_2 \sim t) \Rightarrow s_1 = s_2$
- *S-functional* (or *T-injective*) *iff* $\forall s \in S.\ \forall t_1, t_2 \in T.\ (s \sim t_1 \wedge s \sim t_2) \Rightarrow t_1 = t_2$
- *T-total iff* $\forall t \in T.\ \exists s \in S.\ s \sim t$
- *S-total iff* $\forall s \in S.\ \exists t \in T.\ s \sim t$

□

Lemma 4 (Functionality and Converse Adjoints)**.** *Let $\alpha : \wp(S) \leftrightarrows \wp(T) : \gamma$ corresponding to $\sim \subseteq S \times T$ and let $\alpha^\dagger : \wp(T) \leftrightarrows \wp(S) : \gamma^\dagger$ be the Galois connection corresponding to the converse relation $\sim^\dagger \subseteq T \times S$. Then*

i) *If $\sim$ is T-functional then $\forall \pi_S \in \wp(S).\ \forall \pi_T \in \wp(T).\ \pi_T \subseteq \alpha(\pi_S) \Rightarrow \alpha^\dagger(\pi_T) \subseteq \pi_S$ and*

ii) *If $\sim$ is T-functional and S-total, then α is injective.*

iii) *If $\sim$ is S-functional and T-total, then γ is injective.*

iv) *If $\sim$ is S-functional (or T-injective) and T-total, then $\alpha^\dagger$ is injective.*

v) *If $\sim$ is T-functional and S-total, then $\gamma^\dagger$ is injective.*

vi) *If $\sim$ is T-functional, T-injective and T-total, then $\alpha \circ \alpha^\dagger = id$*

Proof.

i) Notice that since $\sim$ is T-functional, also $\sim^\dagger$ is. Therefore $(\alpha^\dagger)^\dagger \circ \alpha^\dagger \subseteq id$ (see [55, Examples 2, (1)]), and by $(\alpha^\dagger)^\dagger = \alpha$, $\alpha \circ \alpha^\dagger \subseteq id$.

Assume $\pi_T \subseteq \alpha(\pi_S)$, by monotonicity of $\alpha^\dagger$, $\alpha^\dagger(\pi_T) \subseteq \alpha^\dagger(\alpha(\pi_S)) \subseteq \pi_S$.

ii) Assume $\alpha(\pi_S^1) = \alpha(\pi_S^2)$ for $\pi_S^1,\ \pi_S^2 \in \wp(S)$, we show $\pi_S^1 = \pi_S^2$ by mutual inclusion. Let $s \in \pi_S^1$ and let $t \in T$ be such that $s \sim t$, that exists by S-totality. By definition of α, $t \in \alpha(\pi_S^1) = \alpha(\pi_S^2)$ which in turn means $\exists s' \in \pi_S^2.\ s' \sim t$. By T-functionality $s = s' \in \pi_S^2$, that shows $\pi_S^1 \subseteq \pi_S^2$. The other inclusion is symmetric.

iii) Assume $\gamma(\pi_T^1) = \gamma(\pi_T^2)$ for $\pi_T^1,\ \pi_T^2 \in \wp(T)$, we show $\pi_T^1 = \pi_T^2$ by mutual inclusion. Let $t \in \pi_T^1$ and let $s \in S$ be such that $s \sim t$, that exists by T-totality, and is unique by S-functionality. Therefore $s \in \gamma(\pi_T^1) = \gamma(\pi_T^2)$ which by definition of γ implies $t \in \pi_T^2$, showing $\pi_T^1 \subseteq \pi_T^2$. The other inclusion is symmetric.

iv) Assume $\alpha^\dagger(\pi_T^1) = \alpha^\dagger(\pi_T^2)$ for $\pi_T^1,\ \pi_T^2 \in \wp(T)$, we show $\pi_T^1 = \pi_T^2$ by mutual inclusion. Let $t \in \pi_T^1$ and let $s \in T$ be such that $s \sim t$, that exists by T-totality. By definition of $\alpha^\dagger$, $s \in \alpha^\dagger(\pi_T^1) = \alpha^\dagger(\pi_T^2)$, which in turn means $\exists t' \in \pi_T^2.\ s \sim t'$. By S-functional hypothesis for $\sim$, $t = t' \in \pi_T^2$, that shows $\pi_T^1 \subseteq \pi_T^2$. The other inclusion is symmetric.

v) Assume $\gamma^\dagger(\pi_S^1) = \gamma^\dagger(\pi_S^2)$ for $\pi_S^1,\ \pi_S^2 \in \wp(S)$, we show $\pi_S^1 = \pi_S^2$ by mutual inclusion. Let $s \in \pi_S^1$ and let $t \in T$ be such that $s \sim t$, that exists by S-totality, and is unique by T-functionality. Therefore $t \in \gamma^\dagger(\pi_S^1) = \gamma^\dagger(\pi_S^2)$ which by definition of $\gamma^\dagger$ implies $s \in \pi_S^2$, showing $\pi_S^1 \subseteq \pi_S^2$. The other inclusion is symmetric.

vi) Notice that also $\sim^\dagger$ is T-functional, T-injective and T-total. By functionality and totality, $\alpha^\dagger$ and $(\alpha^\dagger)^\dagger = \alpha$ are a Galois connection, with $\alpha^\dagger$ lower adjoint (see [55, Examples 2, (2)]) so that by Definition 12 it suffices to show injectivity of the lower-adjoint,i.e., $\alpha^\dagger$, that follows from *iv)* above.

□

3.3 Closure Operators

In the scenario of Chapter 4 and Chapter 5, the target interpretation of source trace properties may lose some information on the property itself, for example that the property is in monitorable,e.g., it is in *Safety*. When interested in such an information – because, for example, it's possible to monitor target safety properties – we may want to consider the monitorable (safety) target property that *best* approximate the target interpretation obtained. To this end we need the theory of closure operators, recalled hereafter.

Definition 16 (*uco*)**.** An upper closure operator on a poset $(C, \sqsubseteq)$, or $uco(C)$ is a map $\rho : C \to C$ that is:

$$\begin{array}{ll} \textit{monotone} & \forall c_1, c_2 \in C. c_1 \sqsubseteq c_2 \Rightarrow \rho(c_1) \sqsubseteq \rho(c_2) \\ \textit{idempotent} & \forall c \in C.\ \rho(\rho(c)) = \rho(c) \\ \textit{extensive} & \forall c \in C.\ c \sqsubseteq \rho(c) \end{array}$$

When it's clear from the context to which poset we refer to, we simply write *uco*. As customary, for a set X, $x \in X$, and $\rho \in uco(\wp(X), \subseteq)$, we write $\rho(x)$ for $\rho(\{x\})$. □

Remark 5 (*uco* Fixpoints [36])**.** In a complete lattice C, $\rho \in uco(C)$ is uniquely determined by the set of its fixpoints,i.e.,

$$\rho(C) \stackrel{def}{=} \{\rho(c) \mid c \in C\} = \{x \in C \mid \rho(x) = x\}.$$

□

Remark 6 (*uco* and Insertions [36])**.** For a complete lattice $(C, \sqsubseteq)$, and $\rho \in uco(C)$,

$$\rho : (C, \sqsubseteq) \leftrightarrows (\rho(C), \sqsubseteq) : id$$

is a Galois insertion between the poset C and $\rho(C) = \{\rho(c) \mid c \in C\}$ □

Remark 7. For a poset $(C, \sqsubseteq)$ and $\rho \in uco(C)$, for any $c, c' \in C$ and any $r \in \rho(C)$ then by the adjunction law for $\rho \leftrightarrows id$:

(i) $\rho(c) \sqsubseteq r \iff c \sqsubseteq r$

(ii) $\rho(c) \sqsubseteq \rho(c') \iff c \sqsubseteq \rho(c')$

□

We propose a few example of closure operator of particular interest for Chapter 4 and Chapter 5.

Example 8 (Subset Closure Operator)**.** The subset closure operator on a set X

$$\begin{array}{l} Cl_{\subseteq}^{(X)} : \wp(\wp(X)) \to \wp(\wp(X)) \\ \qquad\qquad H \mapsto \{\pi \in \wp(\wp(X)) \mid \exists h \in H.\ \pi \subseteq h\} \end{array}$$

When X is clear from the context we simply write $Cl_{\subseteq}$. Notice that the image of the subset closure operator is the class of the subset-closed hyperproperties on X,i.e.,

$$\left\{ Cl_{\subseteq}^{(X)}(H) \;\middle|\; H \in \wp(\wp(X)) \right\} = Closed_{\subseteq}^{(X)}$$

□

Example 9 (*Safe* and *Hsafe*)**.** Let *Trace* be a set and $\mathcal{O} \subseteq \wp(\mathit{Trace})$ a topology on *Trace*. *Safe* is a closure operator that maps every trace property π to the smallest safety property that includes π,i.e., its topological closure [70]

$$\begin{aligned} \mathit{Safe} : \wp(\mathit{Trace}) &\to \wp(\mathit{Trace}) \\ \pi &\mapsto \bigcap \{S \mid S \in \mathit{Safety}_{\mathcal{O}} \wedge \pi \subseteq S\} \end{aligned}$$

Notice that *Safety* is the set of fixpoints of *Safe*, therefore *Safe* is the lower adjoint of a Galois insertion between trace properties and safety properties,

$$\mathit{Safe} : \wp(\mathit{Trace}) \leftrightarrows \mathit{Safety} : \mathit{id}$$

Similarly *Hsafe* maps every hyperproperty $H \in \wp(\wp(\mathit{Trace}))$ to the smallest hypersafety that includes H,i.e., its topological closure in $\mathcal{V}_{low}(\mathcal{O})$

$$\begin{aligned} \mathit{Hsafe} : \wp(\wp(\mathit{Trace})) &\to \wp(\wp(\mathit{Trace})) \\ H &\mapsto \bigcap \{S \mid S \in \mathit{Hypersafety}_{\mathcal{O}} \wedge H \subseteq S\} \end{aligned}$$

*Hsafe*defines an insertion between $\wp(\wp(\mathit{Trace}))$ and *Hypersafety* but also between $\mathit{Closed}_{\subseteq}$ and *Hypersafety*, because every hypersafety is also subset-closed (see Remark 3),

$$\mathit{Hsafe} : \mathit{Closed}_{\subseteq} \leftrightarrows \mathit{Hypersafety} : \mathit{id}$$

□

Other interesting examples of *uco* come from equivalence relations.

Remark 8 (*uco* and *ER* [58, 36])**.** Every equivalence relation $\mathcal{R} \subseteq X \times X$ corresponds to the $\mathit{uco}(\wp(X), \subseteq)$ that associates to each subset of X the union of the $\mathcal{R}-$equivalence classes of its elements,

$$\begin{aligned} \rho_{\mathcal{R}} : (\wp(X), \subseteq) &\to (\wp(X), \subseteq) \\ S &\mapsto \bigcup_{x \in S} [x]_{\mathcal{R}} \end{aligned}$$

□

Lemma 5. *Let $\mathcal{R} \subseteq X \times X$ be an equivalence relation on X and $\rho \in \mathit{uco}(\wp(X), \subseteq)$ be its corresponding uco,i.e., $\rho = \rho_{\mathcal{R}}$. Then for any $\pi_1, \pi_2 \in \wp(X)$ such that $\rho(\pi_1) = \rho(\pi_2)$,*

$$\forall x_1 \in \pi_1.\ \exists x_2 \in \pi_2.\ \rho(x_1) = \rho(x_2)$$

Proof. Let $x_1 \in \pi_1$, by extensivity of ρ,

$$\rho(x_1) = [x_1]_{\mathcal{R}} \subseteq \rho(\pi_1) = \rho(\pi_2) = \bigcup_{x_2 \in \pi_2} [x_2]_{\mathcal{R}},$$

meaning that for some $x_2 \in \pi$, $x \in [x_2]_{\mathcal{R}}$ which in turn implies $x_1\ \mathcal{R}\ x_2$ and therefore $[x_1]_{\mathcal{R}} = [x_2]_{\mathcal{R}}$, from which the thesis follows by definition of ρ. □

Lemma 6 (Approximating *uco*)**.** *Let $(A, \leq)$ and $(C, \sqsubseteq)$ be posets and $\rho_A \in \mathit{uco}(A)$. Let $\alpha : (C, \sqsubseteq) \leftrightarrows (A, \leq) : \gamma$, be a Galois connection then $\rho_C = \gamma \circ \rho_A \circ \alpha \in \mathit{uco}(C)$.*

Proof. We need to show monotonicity, extensivity and idempotency for ρ_C.

(Monotonicity) Immediate because ρ_C is composition of monotone maps.

(Extensivity) Let $c \in C$,

$$\begin{aligned} \rho_C(c) = \gamma \circ \rho_A \circ \alpha(c) \supseteq & \quad [\text{extensivity of } \rho_A, \text{ monotonicity of } \gamma] \\ \gamma \circ \alpha(c) \supseteq c & \quad [\text{by } \textit{Lemma } 1] \end{aligned}$$

(Idempotency) By extensivity of ρ_C just shown, it suffices to show that for any $c \in C$, $\rho_C \circ \rho_C(c) \subseteq \rho_C(c)$

$$\begin{aligned} \rho_C \circ \rho_C(c) = \gamma \circ \rho_A \circ \alpha \circ \gamma \circ \rho_A \circ \alpha(c) = & \quad [\circ \text{ associative}] \\ (\gamma \circ \rho_A)((\alpha \circ \gamma)(\rho_A \circ \alpha(c))) \subseteq & \quad [\alpha \circ \gamma \subseteq id, \textit{ Lemma } 1] \\ (\gamma \circ \rho_A)(\rho_A \circ \alpha(c)) = \gamma \circ \rho_A \circ \alpha(c) = & \quad [\circ \text{ associative}, \ \rho_A \text{ idempotent}] \\ \rho_C(c) & \quad [\text{definition of } \rho_C] \end{aligned}$$

□

3.4 Abstract Noninterference

The intuitive requirement of noninterference [104] is that executing a program on *low-equivalent* inputs leads to *low-equivalent* outputs, so that the notion of noninterference itself depends on the low-equivalence relations for inputs and outputs. The correspondence between equivalence relations and a subclass of *uco* (see Remark 8) allowed Giacobazzi and Mastroeni [58] to express a wide class of notions of noninterference in the framework *abstract* noninterference. Abstract noninterference can be used to describe delcassifications of a given noninterference, relating the observational power of different attackers, the information that can be revealed – the "flow" – and the one that instead must be kept secret. We use abstract noninterference in Section 4.5 to describe the notion of (target) noninterference guaranteed when correctly compiling source noninterfering programs. Hereafter we recall the rigorous definition of abstract noninterference, already adopted in [10, 7], and that assumes its possible to distinguish inputs and outputs on source and target traces.

Assumption 1 (I/O projection)**.** For Trace there exist sets Trace_I and Trace_O and maps

$$\begin{aligned} \mathtt{prj}_I &: \mathit{Trace} \to \mathit{Trace}_I \\ \mathtt{prj}_O &: \mathit{Trace} \to \mathit{Trace}_O \end{aligned}$$

that we call input and output projections. The projections of $\pi \in \wp(\mathit{Trace})$ are

$$\begin{aligned} \mathtt{prj}_I(\pi) &= \{\mathtt{prj}_I(t) \ \mid \ t \in \pi\} \\ \mathtt{prj}_O(\pi) &= \{\mathtt{prj}_O(t) \ \mid \ t \in \pi\} \end{aligned}$$

Abstract noninterference is a hyperproperty parameterized by two closure operators[1], the first capturing low-equivalence of *inputs* and the second low-equivalence of *outputs*.

Definition 17 (Abstract Noninterference)**.** For $\phi \in \mathit{uco}(\wp(\mathit{Trace}_I))$ and $\rho \in \mathit{uco}(\wp(\mathit{Trace}_O))$,

$$\mathit{ANI}^{\rho}_{\phi} = \{\pi \in \wp(\mathit{Trace}) \ \mid \ \forall t_1, t_2 \in \pi.\ \phi(\mathtt{prj}_I(t_1)) = \phi(\mathtt{prj}_I(t_2)) \Rightarrow \rho(\mathtt{prj}_O(t_1)) = \rho(\mathtt{prj}_O(t_2))\}$$

[1] In its most general abstract noninterference requires 3 closure operators, but for our purposes 2 are sufficient.

is called abstract noninterference w.r.t. ϕ and ρ, or simply abstract noninterference when the parameters are clear from the context.

From now on, with a small abuse of notation we simply write $\phi(t)$ rather than $\phi(\mathtt{prj}_I(t))$ and $\rho(t)$ rather than $\rho(\mathtt{prj}_O(t))$, so that

$$ANI^{\rho}_{\phi} = \{\pi \in \wp(\mathit{Trace}) \mid \forall t_1, t_2 \in \pi.\ \phi(t_1) = \phi(t_2) \Rightarrow \rho(t_1) = \rho(t_2)\}$$

□

Remark 9 ($ANI^{\rho}_{\phi}$is 2-$\mathit{Closed}_{\subseteq}$)**.** Violation of ANI^{ρ}_{ϕ}requires a counterexample π consisting of at least two traces $t_1 \neq t_2$, that are low-equivalent on inputs but not low-equivalent on outputs, $\phi(t_1) = \phi(t_2) \wedge \rho(t_1) \neq \rho(t_2)$.
In models in which traces are sequences and low-equivalence is defined pointwise, the violation of low-equivalence can be decided by observing only two finite prefixes, so that $ANI^{\rho}_{\phi}$is $2-\mathit{Hypersafety}$. This is the case for example for the trace models from Alpern and Schneider [14], or the ones presented in Section 2.5. □

We propose hereafter a useful characterization of ANI^{ρ}_{ϕ} when ϕ and ρ corresponds to equivalence relations.

Lemma 7 ($ANI^{\rho}_{\phi}$for equivalence operators)**.** *Let $\phi \in uco(\wp(Trace_I))$ and $\rho \in uco(\wp(Trace_O))$ correspond to equivalence relations (see Remark 8), then $ANI^{\rho}_{\phi} =_{\pi} ANI^{\rho}_{\phi}$,*

$$_{\pi}ANI^{\rho}_{\phi} = \{\pi \in \wp(\mathit{Trace}) \mid \forall \pi_1, \pi_2 \subseteq \pi.\ \phi(\pi_1) = \phi(\pi_2) \Rightarrow \rho(\pi_1) = \rho(\pi_2)\}$$

Proof. We show mutual inclusion of ANI^{ρ}_{ϕ} and $_{\pi}ANI^{\rho}_{\phi}$.

$_{\pi}ANI^{\rho}_{\phi} \subseteq ANI^{\rho}_{\phi}$ Immediate because in the definition of $_{\pi}ANI^{\rho}_{\phi}$ subsets of π, also include singleton sets, $\pi_i = \{t_i\}$.

$ANI^{\rho}_{\phi} \subseteq {}_{\pi}ANI^{\rho}_{\phi}$ Let $\pi \in ANI^{\rho}_{\phi}$ and let $\pi_1, \pi_2 \subseteq \pi$ be such that $\phi(\pi_1) = \phi(\pi_2)$. Notice that for $i = 1, 2$, if $\phi(\pi_i) = \emptyset$ then by definition – see Remark 8 – $\pi_i = \emptyset$, so that we can assume without loss of generality that both π_1 and π_2 are non empty[2].

By Lemma 5 for any $t_1 \in \pi_1$ there is some $t_2 \in \pi_2$ such that $\phi(t_1) = \phi(t_2)$, and $t_1, t_2 \in \pi \in ANI^{\rho}_{\phi}$ implies $\rho(t_1) = \rho(t_2)$, i.e., $[t_1]_{\mathcal{R}} = [t_2]_{\mathcal{R}}$, for $\mathcal{R}$ being the equivalence relation corresponding to ρ. It follows that

$$\bigcup_{t_1 \in \pi_1} [t_1]_{\mathcal{R}} = \rho(\pi_1) \subseteq \rho(\pi_2) = \bigcup_{t_1 \in \pi_1} [t_1]_{\mathcal{R}}$$

The inclusion $\rho(\pi_2) \subseteq \rho(\pi_1)$ follows by symmetry.

□

3.5 Robust Preservation in Posets

Lemma 8 and Lemma 9 hereafter, abstract the definitions of *preservation* and *robust preservation* of classes of (relational) (hyper)properties through compilation proposed by Abate et al. [8, 10, 7]. The main benefits of are:

(i) a uniform and more compact presentation of the criteria for (robust) preservation than [8, Figure 1] or [7, Fig 4],

[2] if $\pi_1 = \pi_2 = \emptyset$ then $\rho(\pi_1) = \rho(\pi_2) = \rho(\emptyset)$.

(ii) one single proof of equivalence of the criteria, against the several ones provided in e.g., [10, 7]

(iii) a more general notion of (robust) preservation, that allows to consider programs semantics in posets different from powersets ordered by inclusion,

(iv) immediate definition of *reflection* of properties through compilation by duality (see also Section 4.6).

Lemma 8 (Preservation of Elements of a Poset)**.** *Let $\alpha : (C, \sqsubseteq) \leftrightarrows (A, \leq) : \gamma$ be a Galois connection between posets. Let $\mathscr{W}, \mathscr{W}'$ be sets, and $[\![\cdot]\!] : \mathscr{W} \to C$, $(\!|\cdot|\!) : \mathscr{W}' \to A$ and $\cdot\downarrow : \mathscr{W} \to \mathscr{W}'$ functions. The followings are equivalent:*

$$\begin{aligned}
\mathsf{CP}^{\alpha} &\equiv \forall w \in \mathscr{W}.\, \forall c \in C.\, [\![w]\!] \sqsubseteq c \Rightarrow (\!|w\downarrow|\!) \leq \alpha(c) \\
\mathsf{AP}^{\gamma} &\equiv \forall w \in \mathscr{W}.\, \forall a \in A.\, [\![w]\!] \sqsubseteq \gamma(a) \Rightarrow (\!|w\downarrow|\!) \leq a \\
\mathsf{AP}^{\alpha} &\equiv \forall w \in \mathscr{W}.\, \forall a \in A.\, \alpha([\![w]\!]) \leq a \Rightarrow (\!|w\downarrow|\!) \leq a \\
\mathsf{F}^{\gamma}_{\alpha} &\equiv \forall w \in \mathscr{W}.\, (\!|w\downarrow|\!) \leq \alpha([\![w]\!])
\end{aligned}$$

Proof.

$(\mathsf{AP}^{\gamma} \iff \mathsf{AP}^{\alpha})$ Immediate by adjunction law for $\alpha \leftrightarrows \gamma$ (Definition 11).

$(\mathsf{AP}^{\alpha} \Rightarrow \mathsf{CP}^{\alpha})$ Let $w \in \mathscr{W}$ and $c \in C$ be such that $[\![w]\!] \sqsubseteq c$. By monotonicity of α, $\alpha([\![w]\!]) \leq \alpha(c)$, with $\alpha(c) \in A$. Apply AP^{α} to w and $\alpha(c)$ and deduce $(\!|w\downarrow|\!) \leq \alpha(c)$.

$(\mathsf{CP}^{\alpha} \Rightarrow \mathsf{AP}^{\gamma})$ Let $w \in \mathscr{W}$ and $a \in A$ be such that $[\![w]\!] \sqsubseteq \gamma(a)$ with $\gamma(a) \in C$. Apply CP^{α} to w and $\gamma(a)$ and deduce $(\!|w\downarrow|\!) \leq \alpha(\gamma(a))$. By Lemma 1, $\alpha(\gamma(a)) \leq a$.

$(\mathsf{AP}^{\alpha} \Rightarrow \mathsf{F}^{\gamma}_{\alpha})$ Let $w \in \mathscr{W}$, instantiate AP^{α} with w and $\alpha([\![w]\!])$. $\alpha([\![w]\!]) \leq \alpha([\![w]\!])$ by reflexivity of $\leq$ so that $\mathsf{F}^{\gamma}_{\alpha}$ holds.

$(\mathsf{F}^{\gamma}_{\alpha} \Rightarrow \mathsf{AP}^{\alpha})$ Assume $\alpha([\![w]\!]) \leq a$ for some $w \in \mathscr{W}$ and some $a \in A$. By $\mathsf{F}^{\gamma}_{\alpha}$, $(\!|w\downarrow|\!) \leq \alpha([\![w]\!])$ and by transitivity of $\leq$, $(\!|w\downarrow|\!) \leq a$.

□

Lemma 9 (Robust Preservation of Elements of a Poset)**.** *Let $\alpha : (C, \sqsubseteq) \leftrightarrows (A, \leq) : \gamma$ be a Galois connection between a complete lattice C[3] and a poset A. Let $\mathscr{P}, \mathscr{P}', \mathscr{G}, \mathscr{G}'$ be sets, and $[\![\cdot]\!] : \mathscr{G} \times \mathscr{P} \to C$, $(\!|\cdot|\!) : \mathscr{G}' \times \mathscr{P}' \to A$ and $\cdot\downarrow : \mathscr{P} \to \mathscr{P}'$ functions. The followings are equivalent:*

$$\begin{aligned}
\mathsf{RCP}^{\alpha} &\equiv \forall p \in \mathscr{P}.\, \forall c \in C.\, (\forall g \in \mathscr{G}.\, [\![(g,p)]\!] \sqsubseteq c) \Rightarrow (\forall g' \in \mathscr{G}'.\, (\!|(g', p\downarrow)|\!) \leq \alpha(c)) \\
\mathsf{RAP}^{\gamma} &\equiv \forall p \in \mathscr{P}.\, \forall a \in A.\, (\forall g \in \mathscr{G}.\, [\![(g,p)]\!] \sqsubseteq \gamma(a)) \Rightarrow (\forall g' \in \mathscr{G}'.\, (\!|(g', p\downarrow)|\!) \leq a) \\
\mathsf{RAP}^{\alpha} &\equiv \forall p \in \mathscr{P}.\, \forall a \in A.\, (\forall g \in \mathscr{G}.\, \alpha([\![(g,p)]\!]) \leq a) \Rightarrow (\forall g' \in \mathscr{G}'.\, (\!|(g', p\downarrow)|\!) \leq a) \\
\mathsf{RF}^{\gamma}_{\alpha} &\equiv \forall p \in \mathscr{P}.\, \forall g' \in \mathscr{G}'.\, (\!|(g', p\downarrow)|\!) \leq \alpha(\bigcup_{g \in \mathscr{G}} [\![(g,p)]\!])
\end{aligned}$$

Proof.

[3] this hypothesis ensures the existence of the join appearing in the statement.

($\mathsf{RAP}^\gamma \iff \mathsf{RAP}^\alpha$) By adjunction law for $\alpha \leftrightarrows \gamma$ (Definition 11) for any $g \in \mathscr{G}$,

$$\alpha([\![(g,p)]\!]) \leq a \iff [\![(g,p)]\!] \sqsubseteq \gamma(a)$$

so that the equivalence is immediate.

($\mathsf{RAP}^\alpha \Rightarrow \mathsf{RCP}^\alpha$) Let $p \in \mathscr{P}$ and $c \in C$ be such that $\forall g \in \mathscr{G}.\ [\![(g,p)]\!] \sqsubseteq c$. By monotonicity of α, $\forall g.\ \alpha([\![(g,p)]\!]) \leq \alpha(c)$, with $\alpha(c) \in A$. Apply RAP^α to p and $\alpha(c)$ and deduce $\forall g' \in \mathscr{G}'.\ (\!|(g', p\downarrow)|\!) \leq \alpha(c)$.

($\mathsf{RCP}^\alpha \Rightarrow \mathsf{RAP}^\gamma$) Let $p \in \mathscr{P}$ and $a \in A$ be such that $\forall g \in \mathscr{G}.\ [\![(g,p)]\!] \sqsubseteq \gamma(a)$ with $\gamma(a) \in C$. Apply RCP^α to p and $\gamma(a)$ and deduce $\forall g' \in \mathscr{G}'.\ (\!|(g', p\downarrow)|\!) \leq \alpha(\gamma(a)) \leq a$ by Lemma 1.

So far we proved the equivalence of the first three criteria.

($\mathsf{RCP}^\alpha \Rightarrow \mathsf{RF}^\gamma_\alpha$) Let $p \in \mathscr{P}$. Since C is a complete lattice, there exists $\bigcup_{g\in G}[\![(g,p)]\!] \in C$ and

$$\forall g \in \mathscr{G}.\ [\![(g,p)]\!] \sqsubseteq \bigcup_{k\in G} [\![(k,p)]\!].$$

By RCP^α, $\forall g' \in \mathscr{G}'.\ (\!|(g',p)|\!) \leq \alpha(\bigcup_{k\in G}[\![(k,p)]\!])$.

($\mathsf{RF}^\gamma_\alpha \Rightarrow \mathsf{RAP}^\gamma$) Let $p \in \mathscr{P}$ and assume $\forall g \in \mathscr{G}.\ [\![(g,p)]\!] \sqsubseteq \gamma(a)$ for some $a \in A$. Since C is a complete lattice, there exists $\bigcup_{g\in G}[\![(g,p)]\!] \in C$ and

$$\bigcup_{g\in G} [\![(g,p)]\!] \sqsubseteq \gamma(a).$$

By adjunction law for $\alpha \leftrightarrows \gamma$ (Definition 11), the above is equivalent to

$$\alpha(\bigcup_{g\in G} [\![(g,p)]\!]) \leq a$$

Now by $\mathsf{RF}^\gamma_\alpha$, and transitivity of $\leq$ for any $g' \in \mathscr{G}'$

$$(\!|(g', p\downarrow)|\!) \leq a$$

that concludes the proof.

□

We notice explicitly that Lemma 9 cannot be seen as an instance of Lemma 8 because the sets $\mathscr{G}$ and $\mathscr{G}'$ can differ.

The remaining technical lemmas are used in Chapter 4 and Chapter 5 to generalize the (robust) preservation of trace properties to both hyperproperties and relational (hyper)properties.

Lemma 10 (Power Lifting of a Galois Connection)**.**
Let $\alpha : (\wp(S), \subseteq) \leftrightarrows (\wp(T), \subseteq) : \gamma$ be a Galois connection, for S and T sets and $\wp(S)$ and $\wp(T)$ ordered by set inclusion. Then

$$Cl_\subseteq(T) \circ \hat{\alpha} : (Closed^{(S)}_\subseteq, \subseteq) \leftrightarrows (Closed^{(T)}_\subseteq, \subseteq) : Cl_\subseteq(S) \circ \hat{\gamma}$$

is a Galois connection, where

$$\hat{\alpha} = \lambda H \in \mathit{Closed}_{\subseteq}^{(S)}.\ \{\alpha(\pi) \mid \pi \in H\}$$
$$\hat{\gamma} = \lambda H \in \mathit{Closed}_{\subseteq}^{(T)}.\ \{\gamma(\pi) \mid \pi \in H\}$$

Proof. Since $Cl_{\subseteq}$ is a closure operator, we only need to show monotonicity of $\hat{\alpha}$ and $\hat{\gamma}$ and the characteristic property. For the monotonicity of the adjoints, let $H_1 \subseteq H_2$, then for any π, $\pi \in H_1 \Rightarrow \pi \in H_2$, therefore $\alpha(\pi_1) \in \hat{\alpha}(H_1) \Rightarrow \hat{\alpha}(H_2)$, therefore $\hat{\alpha}$ is monotone, and with a similar argument $\hat{\gamma}$ is monotonic.
For the *i*) of Lemma 1, let $H_S \in \mathit{Closed}_{\subseteq}^{(S)}$, we need to show $H_S \subseteq Cl_{\subseteq} \circ \hat{\gamma}(Cl_{\subseteq} \circ \hat{\alpha}(H_S))$. To this end let $\pi_S \in H_S$, then

$$\begin{aligned}
&\alpha(\pi_S) \in \hat{\alpha}(H_S) \Rightarrow \\
&\alpha(\pi_S) \in Cl_{\subseteq} \circ \hat{\alpha}(H_S) \Rightarrow && [\text{by extensivity of } Cl_{\subseteq}] \\
&\gamma(\alpha(\pi_S)) \in \hat{\gamma}(Cl_{\subseteq} \circ \hat{\alpha}(H_S)) \Rightarrow && [\text{by definition of } \hat{\gamma}] \\
&\pi_S \subseteq \gamma(\alpha(\pi_S)) \wedge \gamma(\alpha(\pi_S)) \in \hat{\gamma}(Cl_{\subseteq} \circ \hat{\alpha}(H_S)) \Rightarrow && [\text{by Lemma 1 for } \alpha \leftrightarrows \gamma] \\
&\pi_S \in Cl_{\subseteq} \circ \hat{\gamma}(Cl_{\subseteq} \circ \hat{\alpha}(H_S)) && [\text{by definition of } Cl_{\subseteq}]
\end{aligned}$$

For the *ii*) of Lemma 1, let $H_T \in \mathit{Closed}_{\subseteq}^{(T)}$, we need to show $Cl_{\subseteq} \circ \hat{\alpha}(Cl_{\subseteq} \circ \hat{\gamma}(H_T)) \subseteq H_T$. To this end let $\pi_T \in Cl_{\subseteq} \circ \hat{\alpha}(Cl_{\subseteq} \circ \hat{\gamma}(H_T))$, then

$$\begin{aligned}
&\exists \pi'_T \in H_T.\ \pi_T \subseteq \alpha \circ \gamma(\pi'_T) \Rightarrow && [\text{by unfolding}] \\
&\exists \pi'_T \in H_T.\ \pi_T \subseteq \pi'_T && [\text{by Lemma 1 for } \alpha \leftrightarrows \gamma] \\
&\pi_T \in H_T && [H_T \in \mathit{Closed}_{\subseteq}^{(T)}]
\end{aligned}$$

□

Notice that in Lemma 10 we restricted domain and codomain of the adjoints to the only subset-closed hyperproperties. Actually the lift of α to arbitrary hyperproperties – $\hat{\alpha}$ – still preserves joins and therefore has a unique upper adjoint (see Lemma 2), that however not always coincides with $\hat{\gamma}$.

Lemma 11 (Lift to Full Powerset)**.** *Let $\alpha : (\wp(S), \subseteq) \leftrightarrows (\wp(T), \subseteq) : \gamma$ be a Galois connection, for S and T sets and $\wp(S)$ and $\wp(T)$ ordered by set inclusion. Then*

$$\hat{\alpha} : (\wp(\wp(S)), \subseteq) \leftrightarrows (\wp(\wp(T)), \subseteq) : \check{\gamma}$$

is a Galois connection, where

$$\hat{\alpha} = \lambda H_S \in \wp(\wp(S)).\ \{\alpha(\pi_S) \mid \pi_S \in H_S\}$$
$$\check{\gamma} = \lambda H_T \in \wp(\wp(T)).\ \bigcup \left\{ \alpha^{-1}(\pi_T) \;\middle|\; \pi_T \in H_T \right\}$$

and $\alpha^{-1}(\pi_T) = \{\pi_S \in \wp(S) \mid \alpha(\pi_S) = \pi_T\}$.

Lemma 12 (Relational Galois Connection)**.** *Let $\alpha : (\wp(S), \subseteq) \leftrightarrows (\wp(T), \subseteq) : \gamma$ be a Galois connection, for S and T sets and $\wp(S)$ and $\wp(T)$ ordered by set inclusion. Then*

$$\alpha_{r_k} : (\wp(S^k), \subseteq) \leftrightarrows (\wp(T^k), \subseteq) : \gamma_{r_k}$$

is a Galois connection, where

$$\alpha_{r_k} = \lambda s \in \wp(S^k).\ \{(t_1, t_2, \ldots, t_k) \mid \exists (s_1, s_2, \ldots, s_k) \in s.\ \forall i.\ t_i \in \alpha(\{s_i\})\}$$
$$\gamma_{r_k} = \lambda t \in \wp(T^k).\ \{(s_1, s_2, \ldots, s_k) \mid \forall (t_1, t_2, \ldots, t_k).\ (\ \forall i.\ t_i \in \alpha(\{s_i\})) \Rightarrow (t_1, t_2, \ldots, t_k) \in t\}$$

Proof. Let $\sim \subseteq S \times T$ the relation corresponding to $\alpha \leftrightarrows \gamma$ (see Theorem 2). Consider the relation $\sim^k \subseteq (S^k) \times (T^K)$ defined by:

$$(s_1, s_2, \ldots, s_k) \sim^k (t_1, t_2, \ldots, t_k) \iff \forall i = 1, \ldots, k.\ s_i \sim t_i$$

Then $\alpha_{r_k} \leftrightarrows \gamma_{r_k}$ above is the Galois connection corresponding to $\sim^k$ (see Theorem 2). □

Lemma 13 (Relational Lower Adjoint and Products)**.** *Under the same assumptions of Lemma 12, and for* $X_1, X_2, \ldots, X_k \subseteq S$*,*

$$\alpha_{r_k}(X_1 \times X_2 \times \ldots \times X_k) = \alpha(X_1) \times \alpha(X_2) \times \ldots \times \alpha(X_k),$$

Proof. We show mutual inclusion of the two sets.

"$\subseteq$" Let $(t_1, t_2, \ldots, t_k) \in \alpha_{r_k}(X_1 \times X_2 \times \ldots \times X_k)$. By definition, for any i there exists $x_i \in X_i$ such that $t_i \in \alpha(\{x_i\}) \subseteq \alpha(X_i)$, the last inclusion due to monotoniticty of α. It follows that $(t_1, t_2, \ldots, t_k) \in \alpha(X_1) \times \alpha(X_2) \times \ldots \times \alpha(X_k)$.

"$\supseteq$" Let $(t_1, t_2, \ldots, t_k) \in \alpha(X_1) \times \alpha(X_2) \times \ldots \times \alpha(X_k)$. Then for any i, $t_i \in \alpha(X_i) = \alpha(\bigcup_{x \in X_i} \{x\}) = \bigcup_{x \in X_i} \alpha(\{x\})$, the last equality due to Remark 4. Therefore, for any i, there exists some $x_i \in X_i$ such that $t_i \in \alpha(\{x_i\})$, from which the thesis follows immediately.

□

3.6 Related Work

The main inspiration for this chapter has been the seminal work by Cousot and Cousot [36] and some of their subsequent works [37, 35].

Abstractions for Correctness. The frameworks for Galois connections has been adopted to adjoin the behaviors of source and compiled programs with the goal of comparing expressiveness of theoretical languages [105] or proving some form of correctness e.g., [78, Section 3.1] and [106]. In general, abstraction frameworks are necessary to face the complexity of verification of realistic systems, so that also project such as CompCert [74] or CakeML [115] should be added to the list. For example, the memory model in CompCert is made of infinitely many blocks, that could be considered as a Galois connection between the powersets of finite and infinite sequences of memory blocks.

Abstractions for Security. Although other frameworks are available for the study of noninterference e.g., [63, 104], the one based on *uco* by Giacobazzi and Mastroeni [58] is a better fit for our theory. In the framework of abstract interpretation, security can be achieved by *incompleteness* [59, 57] but binds the attacker to be an abstract interpreter. This choice however, seems to us not expressive enough to model an attacker that can modify part of the memory during the execution of a program.

Chapter 4

Journey in Correct Compilation

In 1967 Painter [88] proves the correctness of a translation from a source language of arithmetic expressions to a target language consisting of a single address machine with an accumulator, load and store instructions but no jump. The correctness theorem states that the result of running a compiled program is to put the value of the (compiled) expression into the accumulator without affecting other registers.
A few years later, Morris [81] defines correctness of a *compile* function in terms of the existence of a *decode* function such that the diagram in Figure 4.1 commutes. The definition of compiler correctness depicted in Figure 4.1 abstracts the syntax of source and target languages, describing a relation between the source semantics of programs and target semantics of compiled programs. A similar idea shows up in may subsequent works about "well-behaved translations", see e.g., [121, 105, 117, 97].

Figure 4.1 depicts the intuition of a *refinement* [97]: the target semantics of compiled programs refines the semantics of source programs, according to the decode function that takes care of reconciling any mismatch between source and target semantics, e.g., implementation details exposed by the target but not the source. Notice that:

(i) the statement of a correctness theorem strongly depends on the refinement relation between compiled and source programs and as a consequence, correctly compiled programs may not satisfy the very same properties as in the source. Assume for example, that according to the source semantics a program that accesses to an uninitialized memory location immediately stops its execution, while in the target semantics the (compiled) program prints infinitely many times 42. When the latter behavior is intended as a refinement of the former, a correct compiler – according to such a refinement relation – may translate terminating programs into non terminating ones, or programs that satisfy the property "`never output 42`" into programs that violate such a property.

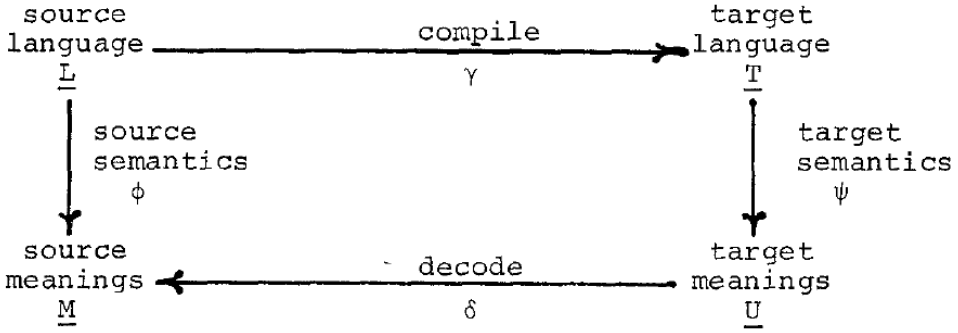

FIGURE 4.1: Definition of compiler correctness by Morris [81].

(ii) many properties, especially security ones, are known to not be preserved by refinement, see e.g., [1, 77, 33] and Chapter 1.

In this chapter we explore correct compilation terms of preservation of classes of (hyper)properties from Chapter 2. When preserving hyperproperties that are subset-closed, our criteria establish a refinement relation between the trace semantics of source and compiled programs, and stronger relations when considering classes of hyperproperties from Chapter 2 that are not included in $\mathit{Closed}_{\subseteq}$.

Moreover, our definitions ease understanding the statement of correctness as they come with equivalent formulations that explicit:

- what are the *guarantees* after compilation for a program that satisfies a certain source (hyper)property,
- what are the *obligations* to satisfy in the source, to have that a certain (hyper)property is satisfied after compilation.

4.1 Notation and Terminology

In this section we fix both notation and terminology we are going to use in the rest of the chapter. We assume that for a language the followings are given:

- a set $\mathcal{W}\mathit{hole}$, the set of (whole) programs that can be written in the language,
- a set Trace, and a topology $\mathcal{O} \subseteq \wp(\mathit{Trace})$ that describe the (trace) semantics of programs and classes of properties and hyperproperties as in Section 2.4,
- a map $\mathtt{beh}(\cdot) : \mathcal{W}\mathit{hole} \to \wp(\mathit{Trace})$ that assigns to every program its semantics. We may write $W \rightsquigarrow t$ for $t \in \mathtt{beh}(W)$.

We say that a program W satisfies:

- a trace property $\pi \in \wp(\mathit{Trace})$ and write $W \models \pi$ *iff* $\mathtt{beh}(W) \subseteq \pi$
- a hyperproperty $H \in \wp(\wp(\mathit{Trace}))$ and write $W \models H$ *iff* $\mathtt{beh}(W) \in H$

We say that programs $W_1, W_2, \ldots, W_k$ satisfy:

- a k-relational trace property $r \in \wp(\mathit{Trace}^k)$ and write $W_1, W_2, \ldots, W_k \models r$ *iff* $(\mathtt{beh}(W_1), \mathtt{beh}(W_2), \ldots, \mathtt{beh}(W_k)) \subseteq r$
- a k-relational hyperproperty $R \in \wp(\mathit{Trace})^k$ and write $W_1, W_2, \ldots, W_k \models R$ *iff* $(\mathtt{beh}(W_1), \mathtt{beh}(W_2), \ldots, \mathtt{beh}(W_k)) \in R$

When compiling from a source language to a target one, we might distinguish them by means of different colors, fonts or subscript (we follow [92]). Namely we use blue−sans−serif for the Source language and write $\cdot\downarrow : \mathsf{Source} \to \mathbf{Target}$ for a compiler from a source target language, rigorously a function $\cdot\downarrow : \mathcal{W}\mathit{hole}_{\mathsf{Source}} \to \mathcal{W}\mathit{hole}_{\mathbf{Target}}$.

4.2 Correct Compilation as Preservation of Satisfaction

When Source and **Target** share the same trace model and therefore source trace properties coincides with target trace properties, the preservation of trace properties

can be naturally expressed by requiring that every compiled program $\mathsf{W}\downarrow$ satisfies the very same properties as W, namely

$$\mathsf{TP} \equiv \forall \mathsf{W}\, \forall \pi \in \wp(\mathit{Trace}).\ \mathsf{W} \models \pi \Rightarrow \mathsf{W}\downarrow \models \pi.$$

It's immediate to show that TP holds *iff* every execution trace of $\mathsf{W}\downarrow$ w.r.t. the target semantics is also a possible execution trace of W w.r.t. the source semantics,i.e., $\mathsf{TP} \iff \mathsf{CC}$ where

$$\mathsf{CC} \equiv \forall \mathsf{W}.\forall t.\ \mathsf{W}\downarrow \rightsquigarrow t \Rightarrow \mathsf{W} \rightsquigarrow t,$$

that in turn describes a *refinement* between the behaviors of compiled programs and the behaviors of source ones,

$$\mathsf{CC} \iff \forall \mathsf{W}.\ \mathtt{beh}(\mathsf{W}\downarrow) \subseteq \mathtt{beh}(\mathsf{W}).$$

Notice that while TP specifies a relation between the trace properties satisfied by source and compiled programs, it turns out to be more convenient to prove – the equivalent – CC , for example by showing a *simulation* of the compiled program w.r.t the source one.

What discussed so far suffers of a big limitation, that is Source and $\mathbf{Target}$ must share the same trace model, so that no implementation detail or physical resource can be abstracted by the source semantics; in the diagram from Figure 4.1 the decompilation function can only be the identity.

While it's easy to express the notion of refinement in more general settings (Figure 4.1 is an option), it's not immediately clear how to generalize the relation between the trace properties satisfied by source and compiled programs expressed in TP . For that it's first of all necessary to *interpret* source trace properties into target ones and target properties into source ones, thus leading to two possible notions of preservation of trace properties.

Definition 18 (Preservation of Trace Properties)**.** For $\tau : \wp(\mathsf{Trace_S}) \to \wp(\mathbf{Trace_T})$ and $\sigma : \wp(\mathbf{Trace_T}) \to \wp(\mathsf{Trace_S})$ maps between source and target hyperproperties, we define:

$$\begin{aligned}\mathsf{TP}^{\tau} &\equiv \forall \mathsf{W}\, \forall \pi_S \in \wp(\mathsf{Trace_S}).\ \mathsf{W} \models \pi_S \Rightarrow \mathsf{W}\downarrow \models \tau(\pi_S)\\ \mathsf{TP}^{\sigma} &\equiv \forall \mathsf{W}\, \forall \pi_T \in \wp(\mathbf{Trace_T}).\ \mathsf{W} \models \sigma(\pi_T) \Rightarrow \mathsf{W}\downarrow \models \pi_T\end{aligned}$$

□

Intuitively a compiler is TP^{τ} if and only if $\tau(\pi_S)$ is the "strongest" *guarantee* –i.e., the smallest trace property – satisfied by $\mathsf{W}\downarrow$ whenever $\mathsf{W} \models \pi_S$. On the other hand a compiler is TP^{σ} if and only if $\sigma(\pi_T)$ is the "weakest" *obligation* –i.e., the largest trace property – that W has to satisfy to have $\mathsf{W}\downarrow \models \pi_T$. Therefore for a compiler that is both TP^{τ} and TP^{σ} :

(i) $\pi_S \subseteq \sigma(\tau(\pi_S))$ because both π_S and $\sigma(\tau(\pi_S))$ are obligations for $\mathsf{W}\downarrow \models \tau(\pi_S)$

(ii) $\tau(\sigma(\pi_T)) \subseteq \pi_T$ because both $\tau(\sigma(\pi_T))$ and π_T are guaranteed by $\mathsf{W} \models \sigma(\pi_T)$

Notice that conditions (i) and (ii) above matches the characteristic property of Galois connections from Lemma 1. We show that under this assumption TP^{τ} and TP^{σ} are equivalent also to another criterion, that generalizes CC and that can still be proved via a simulation see,e.g., [7, Section 4.3].

Theorem 3 (Property-Free characterization [10, 7])**.** *Let* $\tau : \wp(\mathsf{Trace_S}) \leftrightarrows \wp(\mathbf{Trace_T}) :$ σ *be a Galois connection and* $\sim \subseteq \mathit{Trace_S} \times \mathit{Trace_T}$ *its corresponding relation from Theorem 2, then*

$$\mathsf{TP}^{\tau} \iff \mathsf{TP}^{\sigma} \iff \mathsf{CC}^{\sim}$$

where

$$\mathsf{CC}^{\sim} \equiv \forall \mathsf{W}\ \forall t.\ \mathsf{W}{\downarrow} \rightsquigarrow t \Rightarrow \exists\, \mathsf{s} \sim t.\ \mathsf{W} \rightsquigarrow \mathsf{s}.$$

Proof. As a preliminary remark observe that folding the definition of $\mathtt{beh}(\cdot)$, $\mathtt{beh}(\cdot)$ and τ, $\mathsf{CC}^{\sim}$ can be equivalently written as

$$\mathsf{CC}^{\sim} \iff \forall \mathsf{W}.\ \mathtt{beh}(\mathsf{W}{\downarrow}) \subseteq \tau(\mathtt{beh}(\mathsf{W}))$$

($\mathsf{TP}^{\tau} \Rightarrow \mathsf{CC}^{\sim}$) Let W be some source program, notice that by definition $\mathsf{W} \models \mathtt{beh}(\mathsf{W})$, therefore by TP^{τ}, $\mathsf{W}{\downarrow} \models \tau(\mathtt{beh}(\mathsf{W}))$. Unfold the definition of τ and of $\models$ and deduce $\mathtt{beh}(\mathsf{W}{\downarrow}) \subseteq \tau(\mathtt{beh}(\mathsf{W}))$, hence $\mathsf{CC}^{\sim}$.

($\mathsf{CC}^{\sim} \Rightarrow \mathsf{TP}^{\sigma}$) Let W be some source program, π_S a source property and assume $\mathsf{W} \models \pi_S$. Then by definition $\mathtt{beh}(\mathsf{W}) \subseteq \pi_S$, and by monotonicity of τ, $\tau(\mathtt{beh}(\mathsf{W})) \subseteq \tau(\pi_S)$. Finally by hypothesis, $\mathtt{beh}(\mathsf{W}{\downarrow}) \subseteq \tau(\mathtt{beh}(\mathsf{W}))$, so that $\mathtt{beh}(\mathsf{W}{\downarrow}) \subseteq \tau(\pi_S)$,i.e., $\mathsf{W}{\downarrow}{\downarrow} \models \tau(\pi_S)$.

($\mathsf{TP}^{\sigma} \Rightarrow \mathsf{TP}^{\tau}$) Let W be some source program, π_S a source property and assume $\mathsf{W} \models \pi_S$, that is $\mathtt{beh}(\mathsf{W}) \subseteq \pi_S$. Recall that from Lemma 1, $\pi_S \subseteq \sigma(\tau(\pi_S))$, and deduce $\mathsf{W} \models \sigma(\tau(\pi_S))$. Apply TP^{σ} to W and $\tau(\pi_S)$ to conclude.

□

Preservation of trace properties is refinement. Indeed both TP^{τ} and TP^{σ} are equivalent to $\mathsf{CC}^{\sim}$ that in turn can be written as

$$\mathsf{CC}^{\sim} \iff \forall \mathsf{W}.\ \mathtt{beh}(\mathsf{W}{\downarrow}) \subseteq \tau(\mathtt{beh}(\mathsf{W}))$$

in which it's immediately clear that the target behaviors of every compiled program are required to be a subset of – hence refine – (the target interpretation) of source behaviors. The *decoding* in the diagram by Morris [81] from Figure 4.1 is given by $(\subseteq \circ \tau)^{-1}$.

Remark 10 (Correspondence of Traget Guarantees and Source Obligations)**.** Theorem 3 comes with several benefits that are not immediate when stating compiler correctness only as refinement.

Target Guarantees on compiled programs are explicit and their meaningfulness and strength can be easily estimated. Indeed for a $\mathsf{CC}^{\sim}$ – or TP^{τ} – compiler, if $\mathsf{W} \models \pi_S$ then $\mathsf{W}{\downarrow} \models \tau(\pi)$, thus τ describes the guarantees on compiled program and its image on a source property $\tau(\pi_S)$ can tell whether the guarantee is meaningful or vacuous,e.g., $\tau(\pi_S) = \top$.

Source Obligations to be solved are explicit. Indeed for a $\mathsf{CC}^{\sim}$ – or TP^{σ} – compiler, in order to have $\mathsf{W}{\downarrow} \models \pi_T$ it suffices to show that $\mathsf{W} \models \sigma(\pi_T)$.

To sum up, Theorem 3 defines a correspondence between source obligations – TP^{σ} – and target guarantees – TP^{τ} – given by correct compilers. Moreover $\mathsf{CC}^{\sim}$ often follows from a *forward* simulation (W simulates $\mathsf{W}{\downarrow}$) that can be "flipped" to a backward one ($\mathsf{W}{\downarrow}$ simulates W) with a general argument (see in [7, Section 4.3]). □

CompCert provides a correctness theorem[1] that expresses guarantees for any program until (possibly never) it encounters an undefined behavior [101].
We described the trace model adopted in CompCert in Section 2.5. Although the trace model is shared among all intermediate languages the relation to be considered for the correctness theorem is not the identity, but rather the following $\sim \subseteq \mathit{Trace} \times \mathit{Trace}$

$$\mathsf{s} \sim t \iff \mathsf{s} = t \vee (\exists m \leq t.\ \mathsf{s} = m\lightning).$$

$\mathsf{CC}^{\sim}$ for the above $\sim$ is the statement of correctness for CompCert, see also [7, Section 4.1].

CakeML target semantics set a bound on the target memory that can be allocated during executions, such a bound is abstracted away by the source semantics. Compiler correctness is expressed by requiring that compiled programs simulates source ones as long as they don't hit the limit of available memory. Source and target traces can therefore be modeled as following

$$\begin{aligned} \mathit{Trace}_S &= \Sigma^* \cup \Sigma^\omega \\ \mathit{Trace}_T &= \Sigma^* \cup \{m\mathbb{E} \mid m \in \Sigma^*\} \cup \Sigma^\omega \end{aligned}$$

where Σ is a common alphabet of events and $\cdot\mathbb{E} \notin \Sigma$ is a trace terminating event denoting all the memory available has been exhausted. The compiler correctness theorem of CakeML (see also [7, Section 4.2]) can be obtained by instantiating $\mathsf{CC}^{\sim}$ with the following relation $\sim \subseteq \mathit{Trace} \times \mathit{Trace}$:

$$\mathsf{s} \sim t \iff \mathsf{s} = t \vee (\exists m.\ m \leq \mathsf{s}.\ t = m\mathbb{E}).$$

We conclude with a technical remark on another possible formulation of $\mathsf{CC}^{\sim}$ that will be useful in the next section.

Remark 11 ($\mathsf{CC}^{\sim}$ equivalent formulations)**.** Let $\tau : \wp(\mathsf{Trace_S}) \leftrightarrows \wp(\mathbf{Trace_T}) : \sigma$ be a Galois connection and $\sim \subseteq \mathit{Trace}_S \times \mathit{Trace}_T$ its corresponding relation from Theorem 2, then the followings are equivalent:

$$\mathsf{CC}^{\sim} \equiv \forall \mathsf{W}\ \forall t.\ \mathsf{W}{\downarrow} \rightsquigarrow t \Rightarrow \exists\, \mathsf{s} \sim t.\ \mathsf{W} \rightsquigarrow \mathsf{s}$$

$$\forall \mathsf{W}.\ \mathtt{beh}(\mathsf{W}{\downarrow}) \subseteq \tau(\mathtt{beh}(\mathsf{W})) \tag{4.1}$$

$$\forall \mathsf{W}\ \forall \pi_T \in \wp(\mathbf{Trace_T}).\ \mathtt{beh}(\mathsf{W}{\downarrow}) \cap \pi_T \neq \emptyset \Rightarrow \tau(\mathtt{beh}(\mathsf{W})) \cap \pi_T \neq \emptyset \tag{4.2}$$

□

Proof. The first equivalence is immediate by definition of τ. Equation (4.1) trivially implies Equation (4.2). To show also the vice versa, assume $t \in \mathtt{beh}(\mathsf{W}{\downarrow})$, instantiate Equation (4.2) with $\{t\}$ and deduce $t \in \tau(\mathtt{beh}(\mathsf{W}))$. □

4.3 Preservation of Particular Classes

We now focus on the preservation of particular classes of trace properties,e.g., *Safety*, but before that, let us give a deeper look at Theorem 3 and its proof that strongly relies on the properties of the Galois connection $\tau \leftrightarrows \sigma$, but very little on the relation $\sim \subseteq \mathit{Trace}_S \times \mathit{Trace}_T$ that is only used to "fold" the definition of τ in $\mathsf{CC}^{\sim}$.

[1] Stated at the top of the CompCert file `driver/Complements.v` and discussed by Regehr [101].

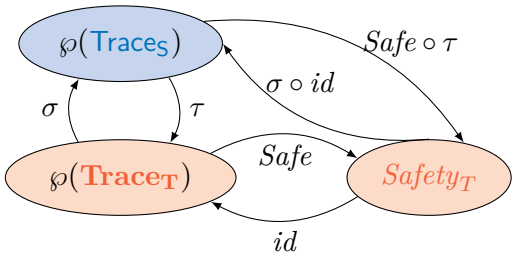

FIGURE 4.2: ([7, Fig. 2]) a Galois connection between $\wp(\mathsf{Trace_S})$ and $Safety_T$, from $\tau : \wp(\mathsf{Trace_S}) \leftrightarrows \wp(\mathbf{Trace_T}) : \sigma$.

The reader might have conjectured that a result similar to Theorem 3 is possible anytime a Galois connection is given, not necessarily the one corresponding to the trace relation. Actually, Lemma 8 states and prove such a result, and when instantiated with the Galois connection $\tau \leftrightarrows \sigma$, $[\![\cdot]\!] = \mathtt{beh}(\cdot)$, $(\!|\cdot|\!) = \mathtt{beh}(\cdot)$, $\cdot\downarrow = \cdot\downarrow$ one has that TP^{τ} and TP^{σ} are equivalent to

$$\forall \mathsf{W}\ \mathtt{beh}(\mathsf{W}\downarrow) \subseteq \tau(\mathtt{beh}(\mathsf{W}))$$

that is just another formulation of $\mathsf{CC}^{\sim}$ (see Remark 11). Lemma 8 clarifies how to define – and in a few equivalent ways – the preservation of classes of properties from Abate et al. [10]. The definition requires a Galois connection between the classes of interests, and when interested in the class of *Safety* we can obtain a Galois connection between target safety properties and source trace properties from $\tau : \wp(\mathsf{Trace_S}) \leftrightarrows \wp(\mathbf{Trace_T}) : \sigma$, as shown in Figure 4.2 ([7, Fig. 2]). It suffices to pre-compose τ with the closure operator *Safe* that defines *Safety* (see Example 9) and post-compose σ with the inclusion of $Safety_T$ in $\wp(\mathbf{Trace_T})$,i.e., the identity.

Vice versa, $\tau \circ id : Safety_S \leftrightarrows \wp(\mathbf{Trace_T}) : Safe \circ \sigma$ in general is not a Galois connection, so that criteria such as

$$\mathtt{buggy}_{\tau} \equiv \forall \mathsf{W}\ \forall \pi_S \in Safety_S.\ \mathsf{W} \models \pi_S \Rightarrow \mathsf{W}\downarrow \models \tau(\pi_S)$$
$$\mathtt{buggy}_{\sigma} \equiv \forall \mathsf{W}\ \forall \pi_T \in \wp(\mathbf{Trace_T}).\ \mathsf{W} \models Safe \circ \sigma(\pi_T) \Rightarrow \mathsf{W}\downarrow \models \pi_T$$

are in general not equivalent, and the correspondence between source obligations and target guarantees from Remark 10 is irremediably lost, as shown in the following example.

Example 10 (Buggy definitions)**.** Let Source and **Target** languages be the same simple imperative language with `WHILE` loops and with natural numbers as possible inputs and outputs. Let the compiler be the identity and source and target traces be (possibly infinite) sequences of inputs and outputs, and the topology the one from Example 1. Let the relation $\sim$ be the identity, so that both τ and σ are the identity too. The above $\mathtt{buggy}_{\tau}$ and $\mathtt{buggy}_{\sigma}$ are not equivalent because the first is (trivially) true for any W and any π_S and the second does not hold for the liveness trace property "eventually `OUTPUT 42`", $\pi = \{t \mid \exists m.\ m \cdot (\mathtt{OUTPUT\ 42}) \leq t\}$, and the program

$$\mathsf{W} = \mathtt{WHILE\ true\ OUTPUT\ 0}.$$

Notice indeed that $\mathtt{beh}(\mathsf{W}) = \mathtt{beh}(\mathsf{W}\downarrow) = \{(\mathtt{OUTPUT\ 0})^{\omega}\}$ and recall that the closure of a liveness property is the set of all traces $Safe(\pi) = Trace$ (liveness properties are

dense sets, see Section 2.3), so that the premises of the $\mathtt{buggy}_\sigma$ are true – $\mathsf{W} \models \mathit{Trace}$ – but its conclusion are false,i.e., $\mathsf{W}{\downarrow} \not\models \pi$. □

We now propose the "right" definitions of preservation of safety properties, obtained by application of Lemma 8 to the Galois connection $\mathit{Safe} \circ \tau \leftrightarrows \sigma$[2].

Theorem 4 (Preservation of *Safety* Properties)**.** *Let $\tau : \wp(\mathbf{Trace_S}) \leftrightarrows \wp(\mathbf{Trace_T}) : \sigma$ a Galois connection between trace properties. The followings are equivalent:*

$$\begin{aligned}
\mathsf{TP}^{\mathit{Safe}\circ\tau} &\equiv \forall \mathsf{W}\ \forall \pi_S \in \wp(\mathbf{Trace_S}).\ \mathsf{W} \models \pi_S \Rightarrow \mathsf{W}{\downarrow} \models \mathit{Safe} \circ \tau(\pi_S)\\
\mathsf{SP}^{\sigma} &\equiv \forall \mathsf{W}\ \forall \pi_T \in \mathit{Safety}_T.\ \mathsf{W} \models \sigma(\pi_T) \Rightarrow \mathsf{W}{\downarrow} \models \pi_T\\
\mathsf{F}^{\sigma}_{\mathit{Safe}\circ\tau} &\equiv \forall \mathsf{W}.\ \mathtt{beh}(\mathsf{W}{\downarrow}) \subseteq \mathit{Safe} \circ \tau(\mathtt{beh}(\mathsf{W}))
\end{aligned}$$

where for $\pi_T \in \wp(\mathbf{Trace_T})$, $\mathit{Safe}(\pi_T)$ is the best approximation of π_T in Safety,i.e., the smallest safety property that includes π_T (see Example 9).

Proof. Apply Lemma 8 with:

- $(C, \sqsubseteq) = (\wp(\mathbf{Trace_S}), \subseteq), (A, \leq) = (\mathit{Safety}_T, \subseteq)$
- the Galois connection $\mathit{Safe} \circ \tau : \wp(\mathbf{Trace_S}) \leftrightarrows \mathit{Safety}_T : \sigma$
- $\mathscr{W} = \mathit{Whole}, [\![\cdot]\!] : \mathit{Whole} \to \wp(\mathbf{Trace_S}) = \mathtt{beh}(\cdot)$
- $\mathscr{W}' = \mathit{Whole}\ (\!|\cdot|\!) : \mathit{Whole} \to \mathit{Safety}_T = \mathit{Safe} \circ \mathtt{beh}(\cdot)$
- $\downarrow : \mathscr{W} \to \mathscr{W}' = \cdot{\downarrow}$

and obtain the followings are equivalent:

$$\begin{aligned}
\mathsf{CP}^{\alpha} &\equiv \forall \mathsf{W}\ \forall \pi_S \in \wp(\mathbf{Trace_S}).\ \mathtt{beh}(\mathsf{W}) \subseteq \pi_S \Rightarrow \mathit{Safe} \circ \mathtt{beh}(\mathsf{W}{\downarrow}) \subseteq \mathit{Safe} \circ \tau(\pi_S)\\
\mathsf{AP}^{\gamma} &\equiv \forall \mathsf{W}\ \forall \pi_T \in \mathit{Safety}_T.\ \mathtt{beh}(\mathsf{W}) \subseteq \sigma(\pi_T) \Rightarrow \mathit{Safe} \circ \mathtt{beh}(\mathsf{W}{\downarrow}) \subseteq \pi_T\\
\mathsf{F}^{\gamma}_{\alpha} &\equiv \forall \mathsf{W}.\ \mathit{Safe} \circ \mathtt{beh}(\mathsf{W}{\downarrow}) \subseteq \mathit{Safe} \circ \tau(\mathtt{beh}(\mathsf{W}))
\end{aligned}$$

By Remark 7 for $\mathit{Safe} \in \mathit{uco}(\wp(\mathbf{Trace_T}))$,

$$\mathsf{CP}^{\alpha} \iff \mathsf{TP}^{\mathit{Safe}\circ\tau}\ ,\ \mathsf{AP}^{\gamma} \iff \mathsf{SP}^{\sigma}\ ,\ \mathsf{F}^{\gamma}_{\alpha} \iff \mathsf{F}^{\sigma}_{\mathit{Safe}\circ\tau},$$

that concludes the proof. □

We propose hereafter a characterization of $\mathsf{F}^{\sigma}_{\mathit{Safe}\circ\tau}$ from Theorem 4 that helps us understanding how the preservation of *Safety* can be proved in the most common trace models,e.g., those from Example 1 and Section 2.5.

Lemma 14 (Criterion for *Safety* preservation)**.** *Let $\tau : \wp(\mathbf{Trace_S}) \leftrightarrows \wp(\mathbf{Trace_T}) : \sigma$ a Galois connection between trace properties. The preservation of Safety properties, in any of the formulation of Theorem 4, is equivalent to*

$$\mathsf{SC}^{\sim} \equiv \forall \mathsf{W}\ \forall A_T \in \mathcal{O}_T.\ \mathtt{beh}(\mathsf{W}{\downarrow}) \cap A_T \neq \emptyset \Rightarrow \tau(\mathtt{beh}(\mathsf{W})) \cap A_T \neq \emptyset.$$

Proof. First of all, recall that for sets $X \subseteq Z$ and $Y \subseteq Z$, $X \subseteq Y \iff X \cap (Z \setminus Y) = \emptyset$, so that $\mathsf{SC}^{\sim}$ is equivalent to

$$\begin{aligned}
&\forall \mathsf{W}.\ \forall A_T \in \mathcal{O}_T.\\
&\qquad \mathtt{beh}(\mathsf{W}{\downarrow}) \not\subseteq (\mathit{Trace}_T \setminus A_T) \Rightarrow \tau(\mathtt{beh}(\mathsf{W})) \not\subseteq (\mathit{Trace}_T \setminus A_T).
\end{aligned} \tag{4.3}$$

[2]Formally we should write "$\sigma \circ \mathit{id}$" or "$\sigma_{|\mathit{Safety}_T}$", but by an innocuous abuse of notation we still denote the upper adjoint by σ.

Recall moreover, that a property is safety *iff* it is the set-theoretic complement of an open set and vice versa. So that Equation (4.3) is equivalent to

$$\forall \mathsf{W}.\ \forall \pi_T \in \mathit{Safety}_T.\ \mathtt{beh}(\mathsf{W}\downarrow) \not\subseteq \pi_T \Rightarrow \tau(\mathtt{beh}(\mathsf{W})) \not\subseteq \pi_T \tag{4.4}$$

and by contraposition to

$$\forall \mathsf{W}\ \forall \pi_T \in \mathit{Safety}_T.\ \tau(\mathtt{beh}(\mathsf{W})) \subseteq \pi_T \Rightarrow \mathtt{beh}(\mathsf{W}\downarrow) \subseteq \pi_T \tag{4.5}$$

Finally by Remark 7, to

$$\forall \mathsf{W}\ \forall \pi_T \in \mathit{Safety}_T.\ \mathit{Safe} \circ \tau(\mathtt{beh}(\mathsf{W})) \subseteq \pi_T \Rightarrow \mathit{Safe}(\mathtt{beh}(\mathsf{W}\downarrow)) \subseteq \pi_T \tag{4.6}$$

This shows that $\mathsf{SC}^{\sim}$ is equivalent to AP^{α} from Lemma 8 (instantiated with the same parameters as in the proof of Theorem 4), that in turn is equivalent to $\mathsf{AP}^{\gamma} \iff \mathsf{CP}^{\alpha} \iff \mathsf{F}^{\gamma}_{\alpha}$ and (se proof of Theorem 4) to the preservation of *Safety* in any of the formulation of Theorem 4. □

Example 11 (Preservation of *Safety* in practice)**.** We already noticed that in the most common trace models, such as the one from Clarkson and Schneider [33] or those presented in Section 2.5, *Safety* is defined by the following formula

$$\pi \in \mathit{Safety} \iff \forall t \notin \pi.\ \exists m \leq t.\ \forall t'.\ m \leq t' \Rightarrow t' \notin \pi.$$

We show that if $\tau \leftrightarrows \sigma$ is the Galois connection corresponding to a certain $\sim \subseteq \mathit{Trace}_S \times \mathit{Trace}_T$ (see Theorem 2), then $\mathsf{SC}^{\sim} \iff$

$$\forall \mathsf{W}\ \forall \mathbf{m}.\ \mathsf{W}\downarrow \rightsquigarrow^* \mathbf{m} \Rightarrow \exists t \geq \mathbf{m}.\ \exists\, \mathtt{s} \sim t. \mathsf{W} \rightsquigarrow \mathtt{s} \tag{4.7}$$

where $\mathsf{W}\downarrow \rightsquigarrow^* \mathbf{m}$ stands for $\exists t \geq m.\ \mathsf{W}\downarrow \rightsquigarrow t$.

(Proof.) For $''\Rightarrow''$ notice that if $\mathsf{W}\downarrow \rightsquigarrow^* \mathbf{m}$ then $\mathtt{beh}(\mathsf{W}\downarrow) \cap \{t \mid m \leq t\} \neq \emptyset$ and $\{t \mid m \leq t\}$ is a (basic) open set (see Example 1). By $\mathsf{SC}^{\sim}$ deduce $\tau(\mathtt{beh}(\mathsf{W})) \cap \{t \mid m \leq t\} \neq \emptyset$ that unfolds to $\exists t.\ (\exists \mathtt{s}.\ \mathtt{s} \sim t \wedge \mathtt{s} \in \mathtt{beh}(\mathsf{W})) \wedge t \in \{t' \mid m \leq t'\}$, from which Equation (4.7) follows immediately.
For $''\Leftarrow''$ assume that $\tau(\mathtt{beh}(\mathsf{W})) \cap A_T \neq \emptyset$ for some program W and some open set $A_T \in \mathcal{O}_T$. Recall that A_T is made of arbitrary union of sets of the form $\{t \mid \exists m \in F.\ m \leq t\}$, where F is a set of finite prefixes. It follows that $\mathsf{W}\downarrow \rightsquigarrow^* m$ and by Equation (4.7) $\exists t \geq m.\ \exists \mathtt{s} \sim t.\ \mathsf{W} \rightsquigarrow \mathtt{s}$ that implies $\tau(\mathtt{beh}(\mathsf{W})) \cap \{t \mid \exists m \in F.\ m \leq t\} \neq \emptyset$ and therefore $\tau(\mathtt{beh}(\mathsf{W})) \cap A_T \neq \emptyset$, that concludes the proof. □

We explicitly notice that when the Source and **Target** languages share the same trace model and $\sim$ is the identity,

$$\mathsf{SC}^{=} \iff \forall \mathsf{W}\ \forall m.\ \mathsf{W}\downarrow \rightsquigarrow^* m \Rightarrow \mathsf{W} \rightsquigarrow^* m.$$

Therefore in trace models such as Example 11, the preservation of safety properties ($\mathsf{SC}^{=}$) can be proved by showing that the $\mathsf{W}\downarrow$ simulates W for an arbitrary but *finite* number of steps. With a coinductive argument, and assuming the trace semantics have no internal nondeterminism, it's possible to show that the simulation actually holds for infinitely many steps, so that the preservation of *Safety* suffices to actually preserve arbitrary trace properties. The reason for this is that in absence of internal nondeterminism, if $t \notin \mathtt{beh}(\mathsf{W})$, then, t has a finite prefix that is not produced by any execution of W,i.e., $\mathtt{beh}(\mathsf{W})$ is a safety property.

Motivated by the above considerations, we abstract the hypothesis on absence of internal nondeterminism to what we call *"safe semantics"*, and show that it gives the equivalence of the preservation of *Safety* and the preservation of arbitrary trace properties.

Theorem 5 ($\mathsf{SC}^{\sim}$ and safe semantics $\Rightarrow$ $\mathsf{CC}^{\sim}$ in $\mathbf{T}_1$ spaces.)**.** *Let* $\tau : \wp(\mathsf{Trace_S}) \leftrightarrows \wp(\mathbf{Trace_T}) : \sigma$ *a Galois connection between trace properties. For* $\sim \subseteq \mathit{Trace}_S \times \mathit{Trace}_T$ *is the relation corresponding to* $\tau \leftrightarrows \sigma$ *(see Definition 13),* $\mathsf{SC}^{\sim} \Rightarrow \mathsf{CC}^{\sim}$ *, under the assumption*

$$\forall \mathsf{W}.\ \tau(\mathtt{beh}(\mathsf{W})) \in \mathit{Safety}_T$$

Proof. Recall that

$$\begin{aligned} \mathsf{CC}^{\sim} &\iff \forall \mathsf{W}.\ \mathtt{beh}(\mathsf{W}{\downarrow}) \subseteq \tau(\mathtt{beh}(\mathsf{W})) \\ \mathsf{SC}^{\sim} &\iff \forall \mathsf{W}.\ \mathtt{beh}(\mathsf{W}{\downarrow}) \subseteq \mathit{Safe} \circ \tau(\mathtt{beh}(\mathsf{W})) \end{aligned}$$

The thesis follows immediately from the fact that for any W,

$$\tau(\mathtt{beh}(\mathsf{W})) \in \mathit{Safety}_T \iff (\mathit{Safe} \circ \tau(\mathtt{beh}(\mathsf{W})) = \tau(\mathtt{beh}(\mathsf{W})))$$

□

On the Preservation of Liveness Properties. Differently to *Safety*, definitions of preservation of *Liveness* sensibly differ depending on the trace model adopted. In the trace models by Clarkson and Schneider [33], every trace property can be written as the intersection of two liveness properties[14, Theorem 2], so that the preservation of *Liveness* suffices for the preservation of arbitrary trace properties. In the trace model by Abate et al. [8], *Liveness* has a minimum element[3], namely the set of all finite traces $L_{min} = \{t \mid \exists m \in \Sigma^*.\ t = m\bullet \vee t = m\lightning\}$ and the map

$$\begin{aligned} \rho : \wp(\mathbf{Trace_T}) &\to \mathit{Liveness}_T \\ \pi_T &\mapsto \pi_T \cup L_{min} \end{aligned}$$

can be used as closure operator[4] to define a Galois connection $\rho \circ \tau \leftrightarrows \sigma$, and hence the preservation of liveness properties, similarly to *Safe* in Theorem 4.

On the Preservation of Arbitrary Hyperproperties. So far we defined compiler correctness as preservation of (classes of) trace properties, but it's well known that these classes cannot express notions such as confidentiality, observational determinism [77, 33], that specify *relations* between two or more executions of the same program. For a notion of compiler correctness that also preserves relations between multiple executions of programs, we extend the theory presented so far easily extend to arbitrary *hyperproperties*, this can be done by lifting galois connections as shown in Lemma 11.

Theorem 6 (Preservation of Arbitrary Hyperproperties)**.** *Let* $\sim \subseteq \mathit{Trace}_S \times \mathit{Trace}_T$ *and let* $\tau : \wp(\mathsf{Trace_S}) \leftrightarrows \wp(\mathbf{Trace_T}) : \sigma$ *the Galois connection corresponding to* $\sim$ *(see Theorem 2). Denote with* $\tau : \wp(\wp(\mathsf{Trace_S})) \leftrightarrows \wp(\wp(\mathbf{Trace_T})) : \check{\sigma}$ *the lift of*

[3] See Lemma `all_fin_in_all_liv`

[4] see Dense closure operator

$\tau \leftrightarrows \sigma$ to arbitrary hyperproperties (see Lemma 11). The followings are equivalent:

$$\begin{aligned}
\mathsf{HP}^{\tau} &\equiv \forall \mathsf{W}\ \forall H_S \in \wp(\wp(\mathsf{Trace_S})).\ \mathsf{W} \models H_S \Rightarrow \mathsf{W}{\downarrow} \models \tau(H_S) \\
\mathsf{HP}^{\check{\sigma}} &\equiv \forall \mathsf{W}\ \forall H_T \in \wp(\wp(\mathbf{Trace_T})).\ \mathsf{W} \models \check{\sigma}(H_T) \Rightarrow \mathsf{W}{\downarrow} \models H_T \\
\mathsf{HC}^{\sim} &\equiv \forall \mathsf{W}.\ \mathit{beh}(\mathsf{W}{\downarrow}) = \tau(\mathit{beh}(\mathsf{W}))
\end{aligned}$$

Proof. In Lemma 8, let $\mathscr{W} = \mathcal{W}hole$, $\mathscr{W}' = \mathcal{W}hole$ and $\cdot{\downarrow} = \cdot{\downarrow}$. Let the abstract domain be $\wp(\wp(\mathbf{Trace_T}))$, the concrete one $\wp(\wp(\mathsf{Trace_S}))$,

$$\begin{aligned}
\alpha &= \lambda H_S.\ \{\tau(\pi_S) \mid \pi_S \in H_S\} \\
\gamma &= \lambda H_T.\ \bigcup \left\{ \alpha^{-1}(\pi_T) \;\middle|\; \pi_T \in H_T \right\}
\end{aligned}$$

that coincide with $\tau \leftrightarrows \check{\sigma}$ (see Lemma 11). Finally let $[\![\cdot]\!] = \{\mathtt{beh}(\cdot)\}$ and $(\!|\cdot|\!) = \{\mathtt{beh}(\cdot)\}$, so that HP^{τ} and $\mathsf{HP}^{\check{\sigma}}$ are equivalent to

$$\forall \mathsf{W}.\ \{\mathtt{beh}(\mathsf{W}{\downarrow})\} \subseteq \tau(\{\mathtt{beh}(\mathsf{W})\})$$

that in turn is equivalent to $\mathsf{HC}^{\sim}$. □

Previous versions of the above theorem (see Abate et al. [7, Theorem 3.8]), adopt – rather than $\check{\sigma}$ – the map $\lambda H_T \in \wp(\wp(\mathbf{Trace_T}))$. $\{\sigma(\pi_T) \mid \pi_T \in H_T\}$, that however is not always the upper adjoint of (the lift of) τ. As a consequence some implications needed the extra hypothesis of $\tau \leftrightarrows \sigma$ being an insertion or a reflection.

4.4 The Atlas of Correct Compilation

Thanks to Lemma 8, we can define criteria for the preservation of each one of the classes of properties, hyperproperties and relational hyperproperties from Section 2.4. Lemma 15 shows however that the criteria presented so far already capture the preservation of most of the classes of interests. For example, $\mathsf{CC}^{\sim}$ – and its equivalent criteria – suffice to ensure also the preservation of subset-closed hyperproperties (known to be preserved by refinement).

Lemma 15 (Preservation of $\mathit{Closed}_{\subseteq}$ and $\wp(\mathit{Trace}^k)$ from $\wp(\mathit{Trace})$)**.**
Let $\tau : \wp(\mathsf{Trace_S}) \leftrightarrows \wp(\mathbf{Trace_T}) : \sigma$ be a Galois connection between trace properties. For simplicity denote still with τ and σ both their lift to subset closed hyperproperties (see Lemma 10) and to relational trace properties (see Lemma 12). The followings hold:

i) TP^{τ} *implies*

$$\mathsf{H}_{\subseteq}\mathsf{P}^{\tau} \equiv \forall \mathsf{W}\ \forall H_S \in \mathit{Closed}_{\subseteq}.\ \mathsf{W} \models H_S \Rightarrow \mathsf{W}{\downarrow} \models \mathit{Cl}_{\subseteq} \circ \tau(H_S)$$

ii) TP^{σ} *implies*

$$\mathsf{H}_{\subseteq}\mathsf{P}^{\sigma} \equiv \forall \mathsf{W}\ \forall H_T \in \mathit{Closed}_{\subseteq}.\ \mathsf{W} \models \mathit{Cl}_{\subseteq} \circ \sigma(H_T) \Rightarrow \mathsf{W}{\downarrow} \models H_T$$

iii) TP^{τ} *implies*

$$\begin{aligned}
k\mathsf{TP}^{\tau} \equiv \forall \mathsf{W}_1 \mathsf{W}_2 \ldots \mathsf{W}_k\ \forall r_S \in \wp(\mathit{Trace_S}^k).& \\
(\mathsf{W}_1, \mathsf{W}_2, \ldots, \mathsf{W}_k) \models r_S \Rightarrow (\mathsf{W}_1{\downarrow}, \mathsf{W}_2{\downarrow}, \ldots, \mathsf{W}_k{\downarrow}) &\models \tau(r_S)
\end{aligned}$$

iv) TP^{τ} *implies*

$$k\mathsf{TP}^{\sigma} \equiv \forall \mathtt{W_1 W_2 \ldots W_k}\ \forall r_T \in \wp(\mathit{Trace_T}^k).\ (\mathtt{W_1}, \mathtt{W_2}, \ldots, \mathtt{W_k}) \models \sigma(r_S) \Rightarrow (\mathtt{W_1}{\downarrow}, \mathtt{W_2}{\downarrow}, \ldots, \mathtt{W_k}{\downarrow}) \models r_T$$

Proof.

i) and *ii)* can be proved as following. In Lemma 8, let $\mathscr{W} = \mathcal{W}\mathit{hole}$, $\mathscr{W}' = \mathcal{W}\mathit{hole}$ and $\cdot{\downarrow} = \cdot{\downarrow}$. Let the abstract domain be $\mathit{Closed}_{\subseteq}^{(\mathit{Trace_T})}$, the concrete one $\mathit{Closed}_{\subseteq}^{(\mathit{Trace_S})}$, $\alpha = \mathit{Cl}_{\subseteq} \circ \tau$ and $\gamma = \mathit{Cl}_{\subseteq} \circ \sigma$, that form a Galois connection for Lemma 10.

Finally let $[\![\cdot]\!] = \mathit{Cl}_{\subseteq}(\{\mathtt{beh}(\cdot)\})$ and $(\!|\cdot|\!) = \mathit{Cl}_{\subseteq}(\{\mathtt{beh}(\cdot)\})$, so that $\mathsf{H}_{\subseteq}\mathsf{P}^{\tau}$ and $\mathsf{H}_{\subseteq}\mathsf{P}^{\sigma}$ are equivalent to

$$\forall \mathtt{W}.\ \mathit{Cl}_{\subseteq}\{\mathtt{beh}(\mathtt{W}{\downarrow})\} \subseteq \mathit{Cl}_{\subseteq} \circ \tau(\{\mathtt{beh}(\mathtt{W})\}) \tag{4.8}$$

that in turns is equivalent to $\forall \mathtt{W}.\mathtt{beh}(\mathtt{W}{\downarrow}) \subseteq \tau(\mathtt{beh}(\mathtt{W}))$,i.e., $\mathsf{CC}^{\sim}$ (see Remark 11).

iii) and *iv)* can be proved as following. In Lemma 8 let $\mathscr{W} = \mathcal{W}\mathit{hole}^k$, $\mathscr{W}' = \mathcal{W}\mathit{hole}^k$ and $\cdot{\downarrow} = \lambda(\mathtt{W_1}, \mathtt{W_2}, \ldots, \mathtt{W_k}).(\mathtt{W_1}{\downarrow}, \mathtt{W_2}{\downarrow}, \ldots.\mathtt{W_k}{\downarrow})$. Let the abstract domain be $\wp(\mathit{Trace_T}^k)$, the concrete one $\wp(\mathit{Trace_S}^k)$, and

$$\alpha = \lambda r_S \in \wp(\mathit{Trace_S}^k).\ \{(t_1, t_2, \ldots, t_k) \mid \exists(\mathtt{s_1}, \mathtt{s_2}, \ldots, \mathtt{s_k}) \in r_S.\ \forall i.\ t_i \in \tau(\{\mathtt{s_i}\})\}$$
$$\gamma = \lambda r_T \in \wp(\mathit{Trace_T}^k).\ \{(\mathtt{s_1}, \mathtt{s_2}, \ldots, \mathtt{s_k}) \mid \forall(t_1, t_2, \ldots, t_k).\ (\ \forall i.\ t_i \in \tau(\{\mathtt{s_i}\})) \Rightarrow (t_1, t_2, \ldots, t_k) \in r_T\}$$

that form a Galois connection for Lemma 12. Finally let

$$[\![\cdot]\!] = \lambda(\mathtt{W_1}, \mathtt{W_2}, \ldots, \mathtt{W_k}).\ \mathtt{beh}(\mathtt{W_1}) \times \mathtt{beh}(\mathtt{W_2}) \times \ldots \times \mathtt{beh}(\mathtt{W_k})$$
$$(\!|\cdot|\!) = \lambda(\mathtt{W_1}, \mathtt{W_2}, \ldots.\mathtt{W_k}).\ \mathtt{beh}(\mathtt{W_1}) \times \mathtt{beh}(\mathtt{W_2}) \times \ldots \times \mathtt{beh}(\mathtt{W_k})$$

so that $k\mathsf{TP}^{\tau}$ and $k\mathsf{TP}^{\sigma}$ are equivalent to

$$\forall(\mathtt{W_1}, \mathtt{W_2}, \ldots, \mathtt{W_k}).\ \mathtt{beh}(\mathtt{W_1}{\downarrow}) \times \mathtt{beh}(\mathtt{W_2}{\downarrow}) \times \ldots \times \mathtt{beh}(\mathtt{W_k}{\downarrow}) \subseteq \alpha(\mathtt{beh}(\mathtt{W_1}) \times \mathtt{beh}(\mathtt{W_2}) \times \ldots \times \mathtt{beh}(\mathtt{W_k}))$$

and by Lemma 13, $\alpha(\mathtt{beh}(\mathtt{W_1}) \times \mathtt{beh}(\mathtt{W_2}) \times \ldots \times \mathtt{beh}(\mathtt{W_k})) = \tau(\mathtt{beh}(\mathtt{W_1})) \times \tau(\mathtt{beh}(\mathtt{W_2})) \ldots \times \tau(\mathtt{beh}(\mathtt{W_k}))$. The thesis follows from application (k times) of $\mathsf{CC}^{\sim} \iff \forall \mathtt{W}.\mathtt{beh}(\mathtt{W}{\downarrow}) \subseteq \tau(\mathtt{beh}(\mathtt{W}))$ (see Remark 11).

□

Corollary 2 (Preservation of *Hypersafety* and *krelSafety*)**.** *Under the hypothesis of Theorem 5,* $\mathsf{SC}^{\sim}$ *(or one of its equivalent formulations from Theorem 4) suffices to preserve also Hypersafety and krelSafety.*

To sum up, the preservation of trace properties suffices to ensure also the preservation of relational properties, subset-closed hyperproperties and hypersafety, and under some assumption on the (target interpretation of the) source semantics it is a consequence of the preservation of *Safety*. It is also the case that the preservation of subset-closed or safety relational hyperproperties follows from $\mathsf{CC}^{\sim}$ (by combining the arguments used in the various points of Lemma 15), while to preserve arbitrary

relational hyperproperties it suffices to preserve arbitrary hyperproperties (same argument as points (iii) and (iv) in Lemma 15 but for $\mathsf{HC}^{\sim}$).
We therefore believe the following diagram exhaustively captures most of the interesting criteria for compiler correctness, and order them according to their relative strength. All the definitions in the diagram are characterized by a correspondence between source obligations and target guarantees as discussed in Remark 10.

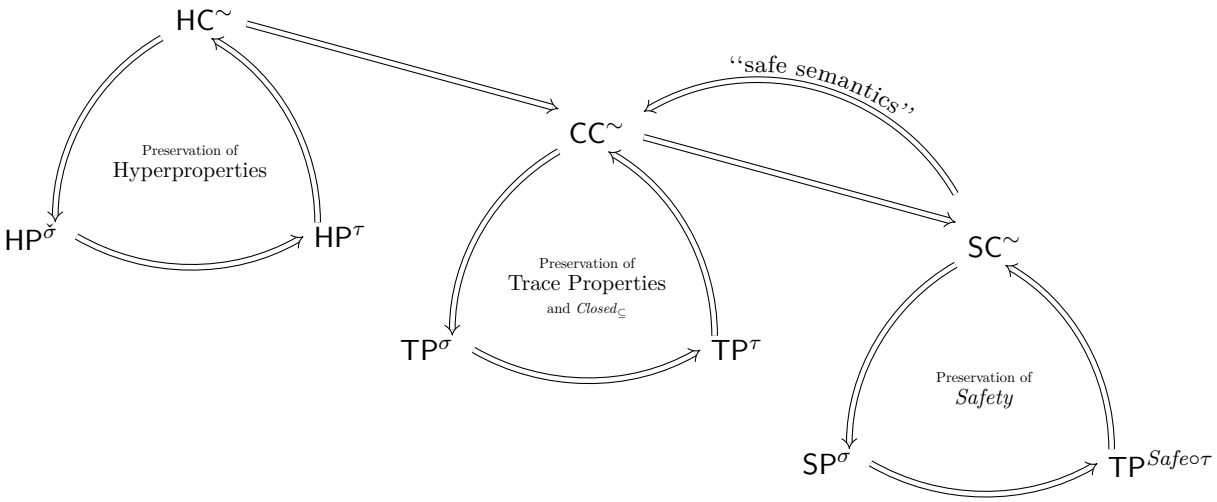

FIGURE 4.3: Main criteria for correct compilation and relations between them [7, Fig. 3]

4.5 Correct Compilers and Noninterference

In Section 4.4 we showed that correct ($\mathsf{CC}^{\sim}$) compilers preserve subset-closed hyperproperties. This means that if W satisfies a subset-closed hyperproperty $H_S \in Closed_{\subseteq}$ then $\mathsf{W}{\downarrow}$ still satisfies a subset-closed hyperproperty $Cl_{\subseteq} \circ \tau(H_S) \in Closed_{\subseteq}$, but not that H_S and $Cl_{\subseteq} \circ \tau(H_S)$ are *the same* specification. For example, if $\mathsf{W} \models ANI^{\rho_S}_{\phi_S}$ we cannot a priori conclude that $\mathsf{W}{\downarrow}$ satisfies some non-trivial target abstract noninterference,i.e., that $Cl_{\subseteq} \circ \tau(H_S) = ANI^{\rho_T}_{\phi_T}$ for some ϕ_T and ρ_T.
In this section we show that under mild conditions on the relation $\sim$ source programs that satisfy noninterference are compiled to target programs that still satisfy noninterference. Our results capture the intuition that if $\mathsf{W} \models ANI^{\rho_S}_{\phi_S}$ then the hyperproperty satisfied by $\mathsf{W}{\downarrow}$ – $Cl_{\subseteq} \circ \tau(ANI^{\rho_S}_{\phi_S})$ – specifies a *declassification* of target noninterference that "matches" $ANI^{\rho_S}_{\phi_S}$. Dually, the obligation to solve in order to have $\mathsf{W}{\downarrow} \models ANI^{\rho_T}_{\phi_T}$ is still a source noninterference that approximates $ANI^{\rho_T}_{\phi_T}$. The results presented in this section are a cleaner and more concise presentation of [7, Section 5].
For this section to Assumption 1, we also add the assumption that $\sim \subseteq Trace_S \times Trace_T$ consists of a relation between source and target inputs and a relation between source and target outputs.

Assumption 2. There exists $\sim_I \subseteq \mathsf{Trace_I} \times \mathbf{Trace_I}$ and $\sim_O \subseteq \mathsf{Trace_O} \times \mathbf{Trace_O}$ and for any s, t

$$\mathsf{s} \sim t \iff \mathtt{prj}_I(\mathsf{s}) \sim_I \mathtt{prj}_I(t) \wedge \mathtt{prj}_O(\mathsf{s}) \sim_O \mathtt{prj}_O(t)$$

With a little abuse of notation we still write τ for τ_I (τ_O) the existential images of $\sim_I$ ($\sim_O$) and σ for σ_I (σ_O) the universal images of $\sim_I$ ($\sim_O$) from Definition 13, and similarly for the converse relations and adjoints from Definition 14. Furthermore we simply say $\sim$ is S-functional on inputs to mean $\sim_I$ is $\mathsf{Trace_O}$-functional and $\sim$ is $\mathbf{T}$-total on outputs to mean $\sim_O$ is $\mathbf{Trace_O}$-total. Similarly we simply say σ is injective on inputs to mean σ_I is injective.

Theorem 7 (Compiling $ANI^{\rho_S}_{\phi_S}$)**.** *Assume* $\cdot\!\downarrow : \mathsf{Source} \to \mathbf{Target}$ *is* $\mathsf{CC}^{\sim}$ *, for a relation* $\sim \subseteq \mathit{Trace}_S \times \mathit{Trace}_T$ *that is* $\mathbf{T}$*-functional on inputs,* S*-total on inputs. Assume that* $\mathsf{W} \models ANI^{\rho_S}_{\phi_S}$ *for* ϕ_S *and* ρ_S *corresponding to equivalence relations. Then* $\mathsf{W}\!\downarrow \models ANI^{\rho_T}_{\phi_T}$*, where*

$$\phi_T = \sigma^\dagger \circ \phi_S \circ \tau^\dagger$$
$$\rho_T = \sigma^\dagger \circ \rho_S \circ \tau^\dagger$$

Proof. First of all notice that ϕ_T and ρ_T are *uco* by Lemma 6. By $\mathsf{CC}^{\sim}$ and Lemma 7, it suffices to show that for any $\pi_T, \pi'_T \subseteq \tau(\mathtt{beh}(\mathsf{W}))$, if $\phi_T(\pi_T) = \phi_T(\pi'_T)$ then $\rho_T(\pi_T) = \rho_T(\pi'_T)$.

$\phi_T(\pi_T) = \phi_T(\pi'_T) \Rightarrow$ [definition of ϕ_T]
$\sigma^\dagger \circ \phi_S \circ \tau^\dagger(\pi_T) = \sigma^\dagger \circ \phi_S \circ \tau^\dagger(\pi'_T) \Rightarrow$ [$\sigma^\dagger$ injective on inputs (*Lemma 4*)]
$\phi_S(\tau^\dagger(\pi_T)) = \phi_S(\tau^\dagger(\pi'_T)) \Rightarrow$ [$\tau^\dagger(\pi_T), \tau^\dagger(\pi'_T) \subseteq \mathtt{beh}(\mathsf{W})$ (*Lemma 4*), $\mathsf{W} \models ANI^{\rho_S}_{\phi_S}$]
$\rho_S(\tau^\dagger(\pi_T)) = \rho_S(\tau^\dagger(\pi'_T)) \Rightarrow$ [$\sigma^\dagger$ is a function]
$\sigma^\dagger \circ \rho_S \circ \tau^\dagger(\pi_T) = \sigma^\dagger \circ \rho_S \circ \tau^\dagger(\pi'_T) \Rightarrow$ [definition of ρ_T]
$\rho_T(\pi_T) = \rho_T(\pi'_T)$

□

Notice that in Theorem 7, injectivity of $\sigma^\dagger$ ensures that $\phi_S = \tau^\dagger \circ \phi_T \circ \sigma^\dagger$, sometimes called the *best correct approximation* of ϕ_T [36]. On the other hand, ρ_S is also a correct approximation of ρ_T, and becomes optimal under the assumption of $\mathbf{T}$-functionality on outputs and S-totality on outputs. In other terms Theorem 7 captures the intuition that with a correct compiler, if a source attacker ϕ_S cannot observe any interference on source programs, then a target attacker ρ_T with "the same" observational power as ϕ_S, cannot observe any interference on compiled programs.

The next result explains in which circumstances satisfaction of target noninterference follows from satisfaction of source noninterference and correct compilation.

Theorem 8 (De-compiling $ANI^{\rho_T}_{\phi_T}$)**.** *Assume* $\cdot\!\downarrow : \mathsf{Source} \to \mathbf{Target}$ *is* $\mathsf{CC}^{\sim}$ *, for a relation* $\sim \subseteq \mathit{Trace}_S \times \mathit{Trace}_T$ *that is* $\mathbf{T}$*-functional, injective and total on inputs,* S*-functional on outputs and* $\mathbf{T}$*-total on outputs. In order to show that* $\mathsf{W}\!\downarrow \models ANI^{\rho_T}_{\phi_T}$*, for* ρ_T *and* ϕ_T *corresponding to equivalence relations, it suffices to show that* $\mathsf{W} \models ANI^{\rho_S}_{\phi_S}$ *where*

$$\phi_S = \sigma \circ \phi_T \circ \tau$$
$$\rho_S = \sigma \circ \rho_T \circ \tau$$

Proof. First of all notice that ϕ_S and ρ_S are *uco* by Lemma 6. By $\mathsf{CC}^{\sim}$ and Lemma 7, it suffices to show that for any $\pi_T, \pi'_T \subseteq \tau(\mathtt{beh}(\mathsf{W}))$, if $\phi_T(\pi_T) = \phi_T(\pi'_T)$ then

$\rho_T(\pi_T) = \rho_T(\pi'_T)$.

$\phi_T(\pi_T) = \phi_T(\pi'_T) \Rightarrow$ [$\tau \circ \tau^\dagger = id$ (*Lemma* 4)]
$\phi_T(\tau \circ \tau^\dagger(\pi_T)) = \phi_T(\tau \circ \tau^\dagger(\pi'_T)) \Rightarrow$ [σ is a function]
$\sigma \circ \phi_T \circ \tau \circ (\tau^\dagger(\pi_T)) = \sigma \circ \phi_T \circ \tau(\tau^\dagger(\pi'_T)) \Rightarrow$ [definition of ϕ_S]
$\phi_S(\tau^\dagger(\pi_T)) = \phi_S(\tau^\dagger(\pi'_T)) \Rightarrow$ [$\tau^\dagger(\pi_T), \tau^\dagger(\pi'_T) \subseteq \mathtt{beh}(\mathsf{W})$ (*Lemma* 4), $\mathsf{W} \models ANI^{\rho_S}_{\phi_S}$]
$\rho_S(\tau^\dagger(\pi_T)) = \rho_S(\tau^\dagger(\pi'_T)) \Rightarrow$ [definition of ρ_S]
$\sigma \circ \rho_T \circ \tau \circ (\tau^\dagger(\pi_T)) = \sigma \circ \rho_T \circ \tau(\tau^\dagger(\pi'_T)) \Rightarrow$ [σ injective on outputs (*Lemma* 4)]
$\phi_T \circ \tau \circ (\tau^\dagger(\pi_T)) = \phi_T \circ \tau(\tau^\dagger(\pi'_T)) \Rightarrow$ [$\tau \circ \tau^\dagger = id$ (*Lemma* 4)]
$\rho_T(\pi_T) = \rho_T(\pi'_T)$

□

Notice that in Theorem 7, injectivity of σ on outputs ensures that $\rho_T = \tau \circ \rho_S \circ \sigma$,i.e., ρ_T is the *best correct approximation* of ρ_S. On the other hand, ϕ_S is also a correct approximation of ϕ_T, and becomes optimal under the assumption of S-functionality on inputs. In other terms Theorem 8 captures the intuition that with a correct compiler, if the observations of the target adversary ρ_T correspond to the observations of a source attacker ρ_S, target noninterference on compiled programs is guaranteed by noninterference of source ones, for which the literautre provides a wide range of verification techniques,e.g., [76, 19, 20, 68].

On some Noninterference Preserving Compilers.

Compilers that preserve instances of abstract noninterference already exist in the literature,e.g., [21, 22, 82, 108]. The preservation of noninterference sometimes follows from a simple simulation,i.e., a proof of $\mathsf{CC}^{\sim}$, with $\sim$ that satisfies the hypothesis of Theorem 7. For example (see [7, Section 5.7] for a more detailed discussion) for many of the compilation steps in the variant of CompCert by Barthe et al. [22], the preservation of *cryptographic constant time* – an instance of ANI^{ρ}_{ϕ}– can be proved by exhibiting a simulation, that is a proof of $\mathsf{CC}^{\sim}$.
In one compilation step however, the relation $\sim \subseteq Trace_S \times Trace_T$ does not satisfy the hypothesis of Theorem 7 and preservation of constant time requires a more involved proof, a "cube-shaped" simulation (see [22, Fig. 9]).

4.6 Property Reflection by Converse Adjoints

The theory we presented generalize compiler correctness – intended as refinement – to the preservation of properties and hyperproperties, also those that are not preserved by refinement.
The relation between the behaviors of $\mathsf{W}\downarrow$ and those of W specified by $\mathsf{CC}^{\sim}$, ensures that W and $\mathsf{W}\downarrow$ satisfy the same properties in the sense of TP^{τ} and TP^{σ} . It's interesting to investigate what relation holds between the trace properties satisfied by $\mathsf{W}\downarrow$ and the one satisfied by W if the we "swap" the relation between the behaviors of source and compiled programs,i.e., if the following holds

$$\mathsf{FC}^{\sim} \equiv \forall \mathsf{W}\ \forall \mathsf{s}.\ \mathsf{W} \rightsquigarrow \mathsf{s} \Rightarrow \exists\, t.\ \mathsf{s} \sim t \wedge \mathsf{W}\downarrow \rightsquigarrow t.$$

$\mathsf{FC}^{\sim}$ ensures the *reflection* of trace properties, that can be easily defined in the presented framework. For a relation $\sim \subseteq \mathit{Trace}_S \times \mathit{Trace}_T$, we can consider its converse

$$\sim^{\dagger} \subseteq \mathit{Trace}_T \times \mathit{Trace}_S, \quad t \sim^{\dagger} \mathsf{s} \iff \mathsf{s} \sim t$$

and the corresponding Galois connection (see Definition 14).

Theorem 9 (Reflection of Trace Properties)**.** *Let* $\sim \subseteq \mathit{Trace}_S \times \mathit{Trace}_T$ *and* $\tau^{\dagger} : \wp(\mathbf{Trace_T}) \leftrightarrows \wp(\mathsf{Trace_S}) : \sigma^{\dagger}$ *the Galois connection corresponding to* $\sim^{\dagger} \subseteq \mathit{Trace}_T \times \mathit{Trace}_S$*, the converse of* $\sim$ *(see Definition 14), then the followings are equivalent:*

$$\begin{aligned}
\mathsf{TR}^{\tau^{\dagger}} &\equiv \forall \mathsf{W}\ \forall \pi_T \in \wp(\mathbf{Trace_T}).\ \mathsf{W}{\downarrow} \models \pi_T \Rightarrow \mathsf{W} \models \tau^{\dagger}(\pi_T) \\
\mathsf{TR}^{\sigma^{\dagger}} &\equiv \forall \mathsf{W}\ \forall \pi_S \in \wp(\mathsf{Trace_S}).\ \mathsf{W}{\downarrow} \models \sigma^{\dagger}(\pi_S) \Rightarrow \mathsf{W} \models \pi_S \\
\mathsf{FC}^{\sim} &\equiv \forall \mathsf{W}\ \forall \mathsf{s}.\ \mathsf{W} \rightsquigarrow \mathsf{s} \Rightarrow \exists\, t.\ \mathsf{s} \sim t \wedge \mathsf{W}{\downarrow} \rightsquigarrow t
\end{aligned}$$

Proof. In Lemma 8, let $\mathscr{W} = \mathscr{W}' = \mathcal{W}\mathit{hole}$, and $\cdot{\downarrow} = \lambda W.\, W$. Let the abstract domain be $\wp(\mathsf{Trace_S})$, the concrete one $\wp(\wp(\mathbf{Trace_T}))$, $[\![\mathsf{W}]\!] = \mathtt{beh}(\mathsf{W}{\downarrow})$ and $(\!|\mathsf{W}|\!) = \mathtt{beh}(\mathsf{W})$. Finally let $\alpha = \tau^{\dagger}$ and $\gamma = \sigma^{\dagger}$ so that $\mathsf{TR}^{\tau^{\dagger}}$ and $\mathsf{TR}^{\sigma^{\dagger}}$ are equivalent to

$$\forall \mathsf{W}.\mathtt{beh}(\mathsf{W}) \subseteq \tau^{\dagger}(\mathtt{beh}(\mathsf{W}{\downarrow}))$$

that corresponds to $\mathsf{FC}^{\sim}$ by unfolding the definition of $\tau^{\dagger}$. □

The definitions of the other possible criteria for the reflection of classes of properties are straightforward.

We believe that a theory of property reflection may find application in generalizing – by reflecting only some properties – the notion of reflection in translations by Sabry and Wadler [105], and we expect proof for $\mathsf{FC}^{\sim}$ could be given by showing that that the source program simulates the target one.

4.7 Related Work

This chapter mainly tackled two problems: the definition of compiler correctness and the understanding of correctness itself.

The definition of correctness has been investigated for the first time by Painter [88], and then by Morris [81] who first abstracted away the syntax of the source and target languages. Hur and Dreyer [64] define correctness of a translation from ML to assembly in terms of a logical relation between a common algebraic model for the two languages, that can be seen as an instance of our framework recalling that what we call *Trace* in this thesis is an arbitrary set (see Definition 6).
In formal verification of compilers such as CompCert [74] and CakeML [115], correctness is usually expressed as a refinement of behaviors and proved by exhibiting a simulation relation w.r.t the small-step operational semantics of the two languages.

The problem of preservation of noninterference is usually approached "operationally",i.e., by showing a relation between the small step operational semantics,see e.g., [21, 22, 82, 108]. We don't see in this a limitation of our framework, as the small step and big step semantics – that may better play the role of $\mathtt{beh}(\cdot)$ – are related by Galois connections [35].

Xia et al. [123] proved the correctness of a translation between `IMP` and `ASM` – an imperative and an assembly like language respectively – by showing an "equivalence

of traces" that we can be seen as an instance of $\mathsf{HC}^{\sim}$ for $\sim$ being the equivalence relation defined by Xia et al. [123].

The problem of understanding the statement of a compiler correctness theorem, has been approached by both Patterson and Ahmed [97] and Abate et al. [10], and then in this thesis.

- Patterson and Ahmed [97] provide a framework to express (compositional) compiler correctness through a refinement relation. By investigating such a relation, a reviewer or a user of a verified compiler can assess the strength of the statement of the compiler correctness theorem.
- Abate et al. [10] (see also [7]) argue that the investigation of the only refinement relation cannot be satisfactory; complex relations can convince a human reviewer and still provide no guarantee at all on compiled programs (see [7, Introduction]). Therefore they (and us) propose a framework in which the refinement relation comes with an explicit description of the guarantees after compilation and the source obligations (see Remark 10). Compositional compiler correctness can be intended as a particular case of the theory of the next chapter, where target contexts only consist of compiled source ones.

Translation validation aims at establishing correctness of every single translation, or run of a compiler [100]. The original idea by Pnueli, Siegel, and Singerman [100] consists in comparing source and compiled programs by means of an analyzer that either produces a counterexample witnessing a behavior of the compiled program that was not possible in the source, or a proof script witnessing the compiled program *correctly* implements the source one. Correctness is formulated by Pnueli, Siegel, and Singerman [100] as a *refinement* relation between the labeled transition systems of source and compiled program.
The compiler correctness criteria presented in this chapter can be used also as translation validation criteria by specializing them with the program of interest. For example, $\mathsf{RSC}^{\sim}(\mathsf{W})$ can be used to show that the compiler preserve all safety properties of the single program W.

Chapter 5

Journey in Secure Compilation

In Chapter 4 we studied the preservation through compilation of classes of hyperproperties. The preservation of the class of arbitrary trace properties and subset-closed hyperproperties is equivalent to say that the behaviors of compiled programs are a subsets of the behaviors of source programs. Preservation of arbitrary – also non subset-closed – hyperproperties, may give strictly stronger guarantees including the preservation of security notions such as some possibilistic information flow policies, that are not preserved by usual refinement [33].

The theory presented in Chapter 4 excludes scenarios in which programs interact with third party applications, external libraries, web browsers, or servers. Vulnerable or malicious third party applications may lead to *low-level* attacks that were not possible in the source, even if the compilation was correct [8, 99, 45]. Imagine, for example, to compile a high-assurance crypto library and link it with a server, that includes unsafe code,e.g., written in an unsafe language such as C/C++. Despite the guarantees given by the library itself e.g., that the private key of the server is not leaked [41], the third party application may be victim of a buffer overflow attack, and allow an adversary to access the key in memory [46].
In order to prevent the exploitation of vulnerabilities similar to the one above, compiled program must be forced to respect buffers boundaries, and to do so, compiler, linker, OS and hardware should implement the *abstractions* for memory safety available in the source language [4]. If source abstractions are successfully implemented there is no room for new vulnerabilities on compiled programs.

Intuition: *Security of a compilation chain is obtained if compiled programs are no more vulnerable than source ones.*

There are multiple ways of making rigorous or formal the intuition above, we focus on two that models *attackers as contexts* of the language [93, 1]:

> *Robust preservation:* hyperproperties satisfied when executing a program in an arbitrary source context, must be satisfied when executing the compiled program in an arbitrary target context [8],
>
> *Full abstraction,* the preservation and reflection of *observational* or *contextual equivalence* [1]. Source programs are indistinguishable when executing them in any source context *iff* after compilation they are indistinguishable when executing them in any target context.

Robust preservation has been originally introduced for hyperproperties of *externally* observable events, such as inputs and outputs, but in several scenarios it's useful to reason w.r.t. a more abstract trace model (see e.g., [8, Section 6.4]), therefore, in Section 5.2 we study robust preservation of abstract classes of hyperproperties from Chapter 2. This choice makes also possible to relate robust preservation and fully abstract compilation, that instead refer only to observations contexts can make and

non necessarily external to the language. In Section 5.5 we reach conclusions similar to Abate, Busi, and Tsampas [6] about the robustness of fully abstract compilers, but without the need of prior knowledge on Mathematical Operational Semantics, making the result accessible to an audience non familiar with the language of category theory.

5.1 Notation and Terminology

Here we fix the notation and terminology we are going to use in the rest of the chapter. We assume that for a language the followings are given:

- a set $\mathcal{P}rg$, the set of programs that can be written in the language,
- a set $\mathcal{C}tx$, the set of contexts with which a program can be linked,
- a set Trace, and a topology $\mathcal{O} \subseteq \wp(\mathit{Trace})$ that describe the (trace) semantics of programs (linked against contexts) and classes of properties and hyperproperties as in Section 2.4,
- a map $\mathtt{beh}(\cdot) : \mathcal{P}rg \times \mathcal{C}tx \to \wp(\mathit{Trace})$ that assigns to every program P its semantics in a given context C, that we denote by $\mathtt{beh}(C[P])$[1].

  We write $C[P] \rightsquigarrow t$ for $t \in \mathtt{beh}(C[P])$.

We say that a program P robustly satisfies:

- a trace property $\pi \in \wp(\mathit{Trace})$ and write $\forall C.\ C[P] \models \pi$ *iff*

$$\forall C.\ \mathtt{beh}(C[P]) \subseteq \pi$$

- a hyperproperty $H \in \wp(\wp(\mathit{Trace}))$ and write $\forall C.\ C[W] \models H$ *iff*

$$\forall C.\ \mathtt{beh}(C[P]) \in H$$

We say that programs $P_1, P_2, \ldots, P_k$ robustly satisfy:

- a k-relational trace property $r \in \wp(\mathit{Trace}^k)$ and write $\forall C.\ C[P_1], C[P_2], \ldots, C[P_k] \models r$ *iff*

$$\forall C.\ (\mathtt{beh}(C[P_1]), \mathtt{beh}(C[P_2]), \ldots, \mathtt{beh}(C[P_k])) \subseteq r$$

- a k-relational hyperproperty $R \in \wp(\mathit{Trace})^k$ and write $\forall C.\ C[P_1], C[P_2], \ldots, C[P_k] \models R$ *iff*

$$\forall C.\ (\mathtt{beh}(C[P_1]), \mathtt{beh}(C[P_2]), \ldots, \mathtt{beh}(C[P_k])) \in R$$

When compiling from a source language to a target one, we might distinguish them by means of different colors, fonts or subscripts (we follow [92]). Namely we use blue−sans−serif for the Source language and write $\cdot\downarrow : \mathsf{Source} \to \mathbf{Target}$ for a compiler from a source target language, rigorously a function $\cdot\downarrow : \mathcal{P}rg \to \mathcal{P}rg$.

[1] Modeling failure in linking a program and a context can be done by setting $\mathtt{beh}(C[P]) = \emptyset$, or adding a partial function for the linking see e.g., [8, Appendix G].

5.2 Secure Compilation as Robust Preservation

In the theory presented in this section, we require a compiler to preserve the satisfaction of trace properties against arbitrary, adversarial contexts. Such a notion is called preservation of robust satisfaction or robust preservation of trace properties and is formally different from the refinement or inclusion of traces from Chapter 4. For robust preservation, every single trace $\mathsf{P}{\downarrow}$ in any target contexts refines some trace P in some source context. From the point of view of security, this can be intended as requiring that any target attack – described by a single execution trace – can be simulated in the source.

Theorem 10 (Robust Preservation of Trace Properties)**.** *Let* $\tau : \wp(\mathsf{Trace_S}) \leftrightarrows \wp(\mathbf{Trace_T}) : \sigma$ *a Galois connection between trace properties. The followings are equivalent:*

$$\begin{aligned}
\mathsf{RTP}^{\tau} &\equiv \forall \mathsf{P}\ \forall \pi_S \in \wp(\mathsf{Trace_S}).\ \forall \mathsf{C}.\ \mathsf{C_S}[\mathsf{P}] \models \pi_S \Rightarrow \forall \mathbf{C_T}.\ \mathbf{C_T}[\mathsf{P}{\downarrow}] \models \tau(\pi_S) \\
\mathsf{RTP}^{\sigma} &\equiv \forall \mathsf{P}\ \forall \pi_T \in \wp(\mathbf{Trace_T}).\ \forall \mathsf{C}.\ \mathsf{C_S}[\mathsf{P}] \models \sigma(\pi_T) \Rightarrow \forall \mathbf{C_T}.\ \mathbf{C_T}[\mathsf{P}{\downarrow}] \models \pi_T \\
\mathsf{RTC}^{\sim} &\equiv \forall \mathsf{P}\ \forall \mathbf{C_T}\ \forall t.\ \mathbf{C_T}[\mathsf{P}{\downarrow}] \rightsquigarrow t \Rightarrow \exists \mathsf{C_S}.\ \exists\, \mathsf{s} \sim t.\ \mathsf{C_S}[\mathsf{P}] \rightsquigarrow \mathsf{s}
\end{aligned}$$

Proof. Apply Lemma 9 with:

- $(C, \sqsubseteq) = (\wp(\mathsf{Trace_S}), \subseteq), (A, \leq) = (\wp(\mathbf{Trace_T}), \subseteq)$
- the Galois connection $\tau : \wp(\mathsf{Trace_S}) \leftrightarrows \wp(\mathbf{Trace_T}) : \sigma$
- $\mathscr{P} = \mathcal{P}rg$, $\mathscr{G} = \mathcal{C}tx$, $[\![\cdot]\!] : \mathcal{C}tx \times \mathcal{P}rg \to \wp(\mathsf{Trace_S}) = \lambda(\mathsf{C_S}, \mathsf{P}).\ \mathtt{beh}(\mathsf{C_S}[\mathsf{P}])$
- $\mathscr{P}' = \mathcal{P}rg$, $\mathscr{G}' = \mathcal{C}tx$, $(\!|\cdot|\!) : \mathcal{C}tx \times \mathcal{P}rg \to \wp(\mathbf{Trace_T}) = \lambda(\mathbf{C_T}, \mathbf{Q}).\ \mathtt{beh}(\mathbf{C_T}[\mathbf{Q}])$
- $\downarrow : \mathcal{P}rg \to \mathcal{P}rg = \cdot{\downarrow}$

It follows that $\mathsf{RTP}^{\tau} \iff \mathsf{RTP}^{\sigma}$ and they are equivalent to

$$\mathsf{RF}^{\sigma}_{\tau} \equiv \forall \mathsf{P}\ \forall \mathbf{C_T}.\ \mathtt{beh}(\mathbf{C_T}[\mathsf{P}{\downarrow}]) \subseteq \tau(\bigcup_{\mathsf{C_S}} \mathtt{beh}(\mathsf{C_S}[\mathsf{P}]))$$

For $\mathsf{RF}^{\sigma}_{\tau} \iff \mathsf{RTC}^{\sim}$ notice that τ preserves joins (see Remark 4) and hence

$$\tau(\bigcup_{\mathsf{C_S}} \mathtt{beh}(\mathsf{C_S}[\mathsf{P}])) = \bigcup_{\mathsf{C_S}} \tau(\mathtt{beh}(\mathsf{C_S}[\mathsf{P}])),$$

from which the thesis follows immediately. □

Remark 12 (Correspondence of Traget Guarantees and Source Obligations)**.** Theorem 10 comes with similar benefits as those discussed in Remark 10.

Target Guarantees are explicit and their meaningfulness and strength can be easily estimated. Indeed, due to the equivalence $\mathsf{RTC}^{\sim} \iff \mathsf{RTP}^{\tau}$, the map τ describes the guarantees on compiled program and against arbitrary target contexts. Its image on a source property – $\tau(\pi_S)$ – defines whether the guarantee is meaningful or vacuous,e.g., $\tau(\pi_S) = \top$.

Source Obligations to be solved are explicit. Indeed, due to the equivalence $\mathsf{RTC}^{\sim} \iff \mathsf{RTP}^{\sigma}$, it suffices to show $\sigma(\pi_T)$ is robustly satisfied in the source, to have π_T robustly satisfied after compilation. □

5.3 Robust Preservation of Particular Classes

Similarly to Section 4.3 we define the robust preservation of *Safety* but in contrast to the results of Section 4.3 we notice a deeper gap between the robust preservation of arbitrary trace properties and of safety ones (Section 5.3.1).

Theorem 11 (Robust Preservation of *Safety* Properties)**.** *Let* $\tau : \wp(\mathsf{Trace_S}) \leftrightarrows \wp(\mathbf{Trace_T}) : \sigma$ *be a Galois connection between trace properties. The followings are equivalent:*

$$\mathsf{RTP}^{Safe \circ \tau} \equiv \forall \mathsf{P}\ \forall \pi_S \in \wp(\mathsf{Trace_S}).\ \forall \mathsf{C}.\ \mathsf{C_S}[\mathsf{P}] \models \pi_S \Rightarrow \forall \mathbf{C_T}.\ \mathbf{C_T}[\mathsf{P}\downarrow] \models Safe \circ \tau(\pi_S)$$
$$\mathsf{RSP}^{\sigma} \equiv \forall \mathsf{P}\ \forall \pi_T \in Safety_T.\ \forall \mathsf{C}.\ \mathsf{C_S}[\mathsf{P}] \models \sigma(\pi_T) \Rightarrow \forall \mathbf{C_T}.\ \mathbf{C_T}[\mathsf{P}\downarrow] \models \pi_T$$
$$\mathsf{RF}^{\sigma}_{Safe \circ \tau} \equiv \forall \mathsf{P}\ \forall \mathbf{C_T}.\ \mathit{beh}(\mathbf{C_T}[\mathsf{P}\downarrow]) \subseteq Safe \circ \tau(\bigcup_{\mathsf{C_S}} \mathit{beh}(\mathsf{C_S}[\mathsf{P}]))$$

where for $\pi_T \in \wp(\mathbf{Trace_T})$, $Safe(\pi_T)$ *is the best approximation of* π_T *in Safety,i.e., the smallest safety property that includes* π_T *(see Example 9).*

Proof. Apply Lemma 9 with:

- $(C, \sqsubseteq) = (\wp(\mathsf{Trace_S}), \subseteq), (A, \leq) = (Safety_T, \subseteq)$
- the Galois connection $Safe \circ \tau : \wp(\mathsf{Trace_S}) \leftrightarrows Safety_T : \sigma$
- $\mathscr{P} = \mathcal{P}rg$, $\mathscr{G} = \mathcal{C}tx$, $[\![\cdot]\!] : \mathcal{C}tx \times \mathcal{P}rg \to \wp(\mathsf{Trace_S}) = \lambda(\mathsf{C_S}, \mathsf{P}).\ \mathtt{beh}(\mathsf{C_S}[\mathsf{P}])$
- $\mathscr{P}' = \mathcal{P}rg$, $\mathscr{G}' = \mathcal{C}tx$, $(\!|\cdot|\!) : \mathcal{C}tx \times \mathcal{P}rg \to Safety_T = \lambda(\mathbf{C_T}, \mathbf{Q}).\ Safe(\mathtt{beh}(\mathbf{C_T}[\mathbf{Q}]))$
- $\downarrow : \mathcal{P}rg \to \mathcal{P}rg = \cdot\downarrow$

and obtain the followings are equivalent:

$$\begin{aligned}
\mathsf{RCP}^{\alpha} &\equiv \forall \mathsf{P}\ \forall \pi_S \in \wp(\mathsf{Trace_S}).\\
&\quad (\forall \mathsf{C_S}.\ \mathtt{beh}(\mathsf{C_S}[\mathsf{P}]) \subseteq \pi_S) \Rightarrow (\forall \mathbf{C_T}.\ Safe(\mathtt{beh}(\mathbf{C_T}[\mathsf{P}\downarrow])) \subseteq Safe \circ \tau(\pi_S))\\
\mathsf{RAP}^{\gamma} &\equiv \forall \mathsf{P}\ \forall \pi_T \in Safety_T.\\
&\quad (\forall \mathsf{C_S}.\ \mathtt{beh}(\mathsf{C_S}[\mathsf{P}]) \subseteq \sigma(\pi_T) \Rightarrow (\forall \mathbf{C_T}.\ Safe \circ \mathtt{beh}(\mathbf{C_T}[\mathsf{P}\downarrow]) \subseteq \pi_T\\
\mathsf{RF}^{\gamma}_{\alpha} &\equiv \forall \mathsf{P}\ \forall \mathbf{C_T}.\ Safe \circ \mathtt{beh}(\mathbf{C_T}[\mathsf{P}\downarrow]) \subseteq Safe \circ \tau(\bigcup_{\mathsf{C_S}} \mathtt{beh}(\mathsf{C_S}[\mathsf{P}]))
\end{aligned}$$

Finally notice that by Remark 7,

$$(\forall \mathbf{C_T}.\ Safe(\mathtt{beh}(\mathbf{C_T}[\mathsf{P}\downarrow])) \subseteq Safe \circ \tau(\pi_S)) \iff (\forall \mathbf{C_T}.\ (\mathtt{beh}(\mathbf{C_T}[\mathsf{P}\downarrow])) \subseteq Safe \circ \tau(\pi_S)),$$

so that $\mathsf{RCP}^{\alpha} \iff \mathsf{RTP}^{Safe \circ \tau}$. Similarly, $\mathsf{RSP}^{\sigma} \iff \mathsf{RAP}^{\gamma}$ and $\mathsf{RF}^{\sigma}_{Safe \circ \tau} \iff \mathsf{RF}^{\gamma}_{\alpha}$, that concludes the proof. □

Similarly to Section 4.3 we provide a characterization that we use to describe the robust preservation of *Safety* in the most common trace models,e.g., those from Example 1 and Section 2.5.

Lemma 16 (Criterion for robust preservation of *Safety*)**.** *Let* $\tau : \wp(\mathsf{Trace_S}) \leftrightarrows \wp(\mathbf{Trace_T}) : \sigma$ *be a Galois connection between trace properties. The robust preservation of Safety properties, in any formulation of Theorem 11, is equivalent to*

$$\mathsf{RSC}^{\sim} \equiv \forall \mathsf{P}\ \forall \mathbf{C_T}\ \forall A_T \in \mathcal{O}_T.\ \mathit{beh}(\mathbf{C_T}[\mathsf{P}\downarrow]) \cap A_T \neq \emptyset \Rightarrow \exists \mathsf{C_S}.\ \tau(\mathit{beh}(\mathsf{C_S}[\mathsf{P}])) \cap A_T \neq \emptyset$$

Proof. First of all notice that by basic symbols manipulation $\mathsf{RSC}^{\sim}$ is equivalent to

$$\forall \mathsf{P}\ \forall A_T \in \mathcal{O}_T.\\ (\exists \mathbf{C_T}.\ \mathtt{beh}(\mathbf{C_T}[\mathsf{P}\downarrow]) \cap A_T \neq \emptyset) \Rightarrow (\exists \mathsf{C_S}.\ \tau(\mathtt{beh}(\mathsf{C_S}[\mathsf{P}]) \cap A_T \neq \emptyset)$$

and by contraposition to

$$\forall \mathsf{P}\ \forall A_T \in \mathcal{O}_T.\\ (\forall \mathsf{C_S}.\ \tau(\mathtt{beh}(\mathsf{C_S}[\mathsf{P}]) \cap A_T = \emptyset) \Rightarrow (\forall \mathbf{C_T}.\ \mathtt{beh}(\mathbf{C_T}[\mathsf{P}\downarrow]) \cap A_T = \emptyset)$$

Now notice that for any open set A_T and any set X, $X \cap A_T = \emptyset$ is equivalent to say that $X \subseteq \mathit{Trace}_T \setminus A_T$, which in turn is in Safety_T (see Definition 7). Therefore $\mathsf{RSC}^{\sim}$ is equivalent to

$$\forall \mathsf{P}\ \forall \pi_T \in \mathit{Safety}_T.\\ (\forall \mathsf{C_S}.\ \tau(\mathtt{beh}(\mathsf{C_S}[\mathsf{P}])) \subseteq \pi_T) \Rightarrow (\forall \mathbf{C_T}.\ \mathtt{beh}(\mathbf{C_T}[\mathsf{P}\downarrow]) \subseteq \pi_T) \tag{5.1}$$

Notice that, by the adjunction law Definition 11 between $\tau \leftrightarrows \sigma$, for every $\mathsf{C_S}$

$$\tau(\mathtt{beh}(\mathsf{C_S}[\mathsf{P}])) \subseteq \pi_T \iff \mathtt{beh}(\mathsf{C_S}[\mathsf{P}]) \subseteq \sigma(\pi_T)$$

and that by Remark 7

$$(\forall \mathbf{C_T}.\ \mathtt{beh}(\mathbf{C_T}[\mathsf{P}\downarrow]) \subseteq \pi_T) \iff (\forall \mathbf{C_T}.\ \mathit{Safe}(\mathtt{beh}(\mathbf{C_T}[\mathsf{P}\downarrow])) \subseteq \pi_T)$$

so that $\mathsf{RSC}^{\sim}$ is equivalent to

$$\forall \mathsf{P}\ \forall \pi_T \in \mathit{Safety}_T.\\ (\forall \mathsf{C_S}.\mathtt{beh}(\mathsf{C_S}[\mathsf{P}]) \subseteq \sigma(\pi_T)) \Rightarrow (\forall \mathbf{C_T}.\ \mathit{Safe}(\mathtt{beh}(\mathbf{C_T}[\mathsf{P}\downarrow])) \subseteq \pi_T).$$

This shows that that $\mathsf{RSC}^{\sim}$ is equivalent to RAP^{α} from Lemma 9 (instantiated with the same parameters as in the proof of Theorem 4), that in turn is equivalent to $\mathsf{RAP}^{\gamma} \iff \mathsf{RCP}^{\alpha} \iff \mathsf{RF}^{\gamma}_{\alpha}$ and (see proof of Theorem 11) to the robust preservation of *Safety* in any of the formulation of Theorem 11. □

Example 12 (Robust Preservation of *Safety* in practice)**.** Generalizing Example 11, in trace models such as the one from Clarkson and Schneider [33] or those presented in Section 2.5[2], $\mathsf{RSC}^{\sim}$ has the following shape

$$\forall \mathsf{P}\ \forall \mathbf{C_T}\ \forall \mathbf{m}.\ \mathbf{C_T}[\mathsf{P}\downarrow] \rightsquigarrow^* \mathbf{m} \Rightarrow \exists \mathsf{C_S}\ \exists t \geq \mathbf{m}\ \exists\ \mathsf{s} \sim t.\mathsf{C_s}[\mathsf{P}] \rightsquigarrow \mathsf{s} \tag{5.2}$$

where $\mathbf{C_T}[\mathsf{P}\downarrow] \rightsquigarrow^* \mathbf{m}$ stands for $\exists t \geq .\mathbf{C_T}[\mathsf{P}\downarrow] \rightsquigarrow t$. In case $\mathit{Trace}_S = \mathit{Trace}_T$ (and $\mathcal{O}_S = \mathcal{O}_T$) and $\sim$ is the identity relation,

$$\mathsf{RSC}^{=} \equiv \forall \mathsf{P}\ \forall \mathbf{C_T} \forall m.\ \mathbf{C_T}[\mathsf{P}\downarrow] \rightsquigarrow^* m \Rightarrow \exists \mathsf{C_s}.\ \mathsf{C_S}[\mathsf{P}] \rightsquigarrow^* m$$

for which the literature provides already a few proof techniques [11, 116, 95]. □

5.3.1 Robust Preservation of Safety vs Arbitrary Properties

Differently to what shown in Section 4.3, the equivalence of the robust preservation of *Safety* and that of arbitrary trace properties requires strong assumptions. The

[2] where $\pi \in \mathit{Safety} \iff \forall t \notin \pi.\ \exists m \leq t.\ \forall t'.\ m \leq t' \Rightarrow t' \notin \pi$.

equivalence of $\mathsf{RSC}^{\sim}$ and $\mathsf{RTC}^{\sim}$ means the equality of

$$\bigcup_{\mathsf{C_S}} \tau(\mathtt{beh}(\mathsf{C_S}[\mathsf{P}])) = \mathit{Safe}(\bigcup_{\mathsf{C_S}} \tau(\mathtt{beh}(\mathsf{C_S}[\mathsf{P}]))) \tag{5.3}$$

that is

$$\bigcup_{\mathsf{C_S}} \tau(\mathtt{beh}(\mathsf{C_S}[\mathsf{P}])) \in \mathit{Safety}_T.$$

Assuming the (target interpretation of the) behaviors of P are a safety property in any $\mathsf{C_S}$ (a generalization of the hypothesis of Theorem 5), does not suffices for Equation (5.3) to hold, as the class of *Safety* is closed under *finite*, but not *arbitrary* union.

More concretely, the problem for the CompCert trace model and for the relation $\sim$ being the identity, correspond to the equivalence to

$$\mathsf{RSC}^{=} \equiv \forall \mathsf{P}\ \forall \mathbf{C_T} \forall m.\ \mathbf{C_T}[\mathsf{P}\downarrow] \rightsquigarrow^* m \Rightarrow \exists \mathsf{C_s}.\ \mathsf{C_S}[\mathsf{P}] \rightsquigarrow^* m$$
$$\mathsf{RTC}^{=} \equiv \forall \mathsf{P}\ \forall \mathbf{C_T} \forall t.\ \mathbf{C_T}[\mathsf{P}\downarrow] \rightsquigarrow t \Rightarrow \exists \mathsf{C_s}.\ \mathsf{C_S}[\mathsf{P}] \rightsquigarrow t$$

where the m above may be a simple list of events $l \in \Sigma^*$, or $l\bullet$, $l\lightning$, $l\circlearrowleft$ depending on the topology chosen (see Section 2.5). If $\mathbf{C_T}[\mathsf{P}\downarrow] \rightsquigarrow t \in \Sigma^\omega$, $\mathsf{RSC}^{=}$ ensures the existence of a family of source contexts $\left\{ \mathsf{C_S}^{(m)} \;\middle|\; m \leq t \right\}$, such that

$$\forall m \leq t.\ \mathsf{C_S}^{(m)}[\mathsf{P}] \rightsquigarrow^* m.$$

This however, does not suffice to conclude the whole trace t can be produced by P in one of the $\mathsf{C_S}^{(m)}$ above, as better exemplified by the following example.

Example 13 (From appendix E of [8])**.** Let Source and **Target** languages be the same simple imperative language with `WHILE` loops and with natural numbers as possible inputs and outputs. Let the compiler be the identity and consider the CompCert trace model with the topology of Example 6[3].
Contexts simply impose the "fuel" available for the execution of a program and in the source $\mathcal{C}tx = \mathbb{N}$ while in the target $\mathcal{C}tx = \mathbb{N} \cup \{\omega\}$ where ω means no bound is set, so that the program can possibly run forever.
$\mathsf{RSC}^{=}$ is clearly satisfied by the identity compiler, but $\mathsf{RTC}^{=}$ fails on

$$\mathsf{P} = \texttt{WHILE true OUTPUT 42;}$$

and the target context $\mathbf{C_T} = \omega$. □

Nevertheless, there are scenarios in which Equation (5.3) holds (under the same assumptions as in Section 4.3) and the robust preservation of *Safety* is equivalent to that of arbitrary trace properties.

Remark 13 (Preserving Robust Safety of Processes)**.** In process calculi adopted for communication protocols, attackers can be modeled as processes and composed in parallel with the process defining a protocol,see e.g., [2]. With a single global channel fixed, it's possible to obtain the most powerful attacker of Dolev and Yao [44] – *DY* below – by parallel composition of all the primitives of the languge, operating on the same global channel. Proving security of a protocol against such an attacker suffices

[3] for the topology of Example 7 one just need to extend the contexts with a boolean that when true allows the program to silently diverge.

to guarantee *robustness*, i.e., security against any attacker of the language (see [126] for some examples).

In Example 14 we recall the syntax of the applied π-calculus by Abadi, Blanchet, and Fournet [2, Section 2.1] and show a possible most powerful adversary for a simple process. Hereafter we assume Source is some process calculus, and for a protocol P and an attacker C, let $\mathtt{beh}(\mathsf{C}[\mathsf{P}])$ denote the trace semantics of $\mathsf{C} \mid \mathsf{P}$. Let DY be the most powerful source attacker, then

$$\bigcup_{\mathsf{C_S}} \tau(\mathtt{beh}(\mathsf{C_S}[\mathsf{P}])) = \tau(\mathtt{beh}(\mathsf{DY}[\mathsf{P}])) \tag{5.4}$$

If $\tau(\mathtt{beh}(\mathsf{DY}[\mathsf{P}])) \in \mathit{Safety}_T$ then Equation (5.3) holds and the robust preservation of *Safety* is equivalent to the robut preservation of arbitrary trace properties. We discuss potential consequences of this fact in Section 6.1. □

Example 14 (Applied π-calculus and most powerful attacker)**.**
The syntax of the applied π-calculus assumes a finite set Σ of function symbols,e.g., f, `enc`, `pair`, each with its own arity (arity 0 for a constant symbol). Function symbols are defined by an equational theory or by means of rewrite rules,e.g., encryption and decryption functions to be used to model symmetric encryption ([2, Section 3]) can be defined by binary symbols `enc`and `dec`satisfying the equation

$$\mathtt{dec}(\mathtt{enc}(m, k), k) = m,$$

where m plays the role of a plaintext and k of a (symmetric) key.

Terms are defined by the following grammar

$L, M, N, T, U, V ::=$	terms
$a, b, c, \ldots, k, \ldots, m, n, \ldots, s$	name
x, y, z	variable
$f(M_1, M_2, \ldots, M_l)$	function application

where $f \in \Sigma$ and has arity l.

Plain Processes are defined by the following grammar, where u, v, w range both on names and variables

$P, Q, R ::=$	plain processes
$\mathbf{0}$	null process
$P \mid Q$	parallel composition
$!P$	replication
$\nu n.P$	name restriction (“new”)
if $M = N$ then P else Q	conditional
$u(x).P$	message input
$\bar{u}\langle N\rangle.P$	message output

Finally, *extended* processes are defined by the following grammar

$A, B, C ::=$	extended processes
P	plain process
$A \mid B$	parallel composition
$\nu n.A$	name restriction
$\nu x.A$	variable restriction
$\{M/x\}$	active substitution

The following process ([2, page 12])

$$\nu k.\bar{a}\langle \mathtt{enc}(M,k)\rangle$$

sends over the message M encrypted with a fresh key k over a.

By assuming a single global channel (g below), the Dolev-Yao attacker for the process above can reroute, discard and forge messages and apply function symbols such as `enc`and `dec`, and execute the protocol an unbounded number of times. Therefore the following models a most powerful attacker for the process above

$$\begin{aligned} DY = (&\bar{g}.\mathbf{0} \\ &\mid !(\nu x.\bar{g}\langle x\rangle.\mathbf{0}) \\ &\mid !(g(x).\bar{g}\langle x\rangle.\mathbf{0}) \\ &\mid !(g(x).g(y).\bar{g}\langle \mathtt{enc}(x,y)\rangle.\bar{g}\langle \mathtt{dec}(x,y)\rangle) \\ &\mid !(\bar{g}\langle x\rangle)) \end{aligned}$$

The interested reader can refer to [126, Section 4] for processes that define a key exchange protocol based on Diffie-Hellman and corresponding Dolev-Yao adversary. □

5.4 The Atlas of Secure Compilation

In Section 4.4 we argued that the interesting notions of preservation for a compiler can be restricted essentially to: the preservation of *Safety*, of arbitrary trace properties and of arbitrary hyperproperties. The introduction of contexts in the theory comes with a finer distinction between criteria that express robust preservation of distinct classes of (relational) (hyper)properties. In particular it's possible to create examples showing that the robust preservation of subset-closed hyperproperties or of relational trace properties are strictly stronger than the one of trace properties (see [8, Appendix E]).
The reason for this is that for the robust preservation of trace properties for any trace t such that $\mathbf{C_T}[\mathsf{P}\downarrow] \rightsquigarrow t$, it's required to exhibit a source context $\mathsf{C_S}$ and a source trace $\mathsf{s} \sim t$, such that $\mathsf{C_S}[\mathsf{P}] \rightsquigarrow \mathsf{s}$, while in the robust preservation of $Closed_{\subseteq}$ below, the source context $\mathsf{C_S}$ must be the same for all the target trace produced by $\mathbf{C_T}[\mathsf{P}\downarrow]$.
A similar reasoning suggest why the robust preservation of relational properties is strictly stronger than the robust preservation of trace properties. Hereafter we explicit some of the criteria in their property-free formulation, the interested reader will find the complete taxonomy in Appendix A.

Robust Preservation of subset-closed hyperproperties – $Closed_{\subseteq}$

$$\mathsf{RH}_{\subseteq}\mathsf{C}^{\sim} \iff \forall \mathsf{P}\ \forall \mathbf{C_T}.\ \exists \mathsf{C_S}.\ \mathtt{beh}(\mathbf{C_T}[\mathsf{P}\downarrow]) \subseteq \tau(\mathtt{beh}(\mathsf{C_S}[\mathsf{P}]))$$

Robust Preservation of arbitrary hyperproperties – $\wp(\wp(Trace))$

$$\mathsf{RHC}^{\sim} \iff \forall \mathsf{P}\ \forall \mathbf{C_T}.\ \exists \mathsf{C_S}.\ \mathtt{beh}(\mathbf{C_T}[\mathsf{P}\downarrow]) = \tau(\mathtt{beh}(\mathsf{C_S}[\mathsf{P}]))$$

Robust Preservation of relational trace properties $\wp(Trace^k)$

$$k\mathsf{RTC}^{\sim} \iff \forall \mathsf{P_1}, \mathsf{P_2}, \ldots \mathsf{P_k}\ \forall \mathbf{C_T}\ \forall t_1, t_2, \ldots, t_k.(\forall i = 1 \ldots k.\ \mathbf{C_T}[\mathsf{P_i}\downarrow] \rightsquigarrow t_i) \Rightarrow \exists \mathsf{C_S}\ \exists \mathsf{s_1}, \mathsf{s_2}, \ldots, \mathsf{s_k}.(\forall i = 1 \ldots k.\ \mathsf{s_i} \sim t_i \wedge \mathsf{C_S}[\mathsf{P_i}] \rightsquigarrow \mathsf{s_i})$$

Robust Preservation of relational hyperproperties – $\wp(Trace)^k$

$$k\mathsf{RHC}^{\sim} \iff \forall \mathsf{P_1}, \mathsf{P_2}, \ldots \mathsf{P_k}\ \forall \mathbf{C_T}.\ \exists \mathsf{C_S}.\ \forall i = 1, 2, \ldots, k\ \ \mathtt{beh}(\mathbf{C_T}[\mathsf{P_i}\downarrow]) = \tau(\mathtt{beh}(\mathsf{C_S}[\mathsf{P_i}]))$$

The criterion for *Hypersafety*, comes with a property-free formulation that is formally similar to the one for *Safety* – $\mathsf{RSC}^{\sim}$ – but involves the open sets of the lower Vietoris topology from Definition 1,

Robust Preservation of *Hypersafety* $\iff \forall \mathsf{P}\ \forall \mathbf{C_T}\ \forall A_T \in \mathcal{V}_{low}(\mathcal{O}_T).$
$\{\mathtt{beh}(\mathbf{C_T}[\mathsf{P}\downarrow])\} \in A_T \Rightarrow \exists \mathsf{C_S}.\ \tau(\{\mathtt{beh}(\mathsf{C_S}[\mathsf{P}])\}) \in A_T.$

In the most common application where the trace model is the one from Section 2.5 and open sets are defined by finite sets of finite prefixes M, it assumes the formulation given by Abate et al. [8] (for the relation between source and target traces being the identity),

$$\forall \mathsf{P}\ \forall \mathbf{C_T}\ \forall M.\ M \leq \mathtt{beh}(\mathbf{C_T}[\mathsf{P}\downarrow]) \Rightarrow \exists \mathsf{C_S}.\ M \leq \mathtt{beh}(\mathsf{C_S}[\mathsf{P}])$$

Figure 5.1 exhaustively captures most of the interesting criteria for secure compilation, and order them according to their relative strength, all the missing detail are in Appendix A.

5.4.1 Language Features that Simplify the Hierarchy

The hierarchy in Figure 5.1 may be sensibly simplified under specific features of the languages that may allow contexts to read the code of the programs they are linked with (*reflection*) or to make non-deterministic choices.

Reflection *"is the ability of an agent to reason not only introspectively, about its self and internal thought processes, but also externally, about its behaviour and situation in the world"* [111, Section 2]. In programming languages this implies that a context can get some information on the program it is linked with, and in languages like Lisp contexts can even access to the code of the program they are linked with [111]. This feature is sometimes called *full reflection* and is known to be cause of degeneration of observational equivalence to syntactic equality [120, 50].
Notice that full reflection in the Source language allows a source context $\mathsf{C_S}$ to behave differently when linked with a program $\mathsf{P_1}$ rather than $\mathsf{P_2}$. In particular if $\mathsf{C_1}[\mathsf{P_1}] \rightsquigarrow \mathsf{s_1}$ and $\mathsf{C_2}[\mathsf{P_2}] \rightsquigarrow \mathsf{s_2}$, then full reflection allows to write a context $\mathsf{C_S}$ such that linked with $\mathsf{P_1}$ may lead to the trace $\mathsf{s_1}$ and linked with $\mathsf{P_2}$ to the trace $\mathsf{s_2}$,

$$\mathsf{C_S}[\mathsf{P_1}] \rightsquigarrow \mathsf{s_1} \wedge \mathsf{C_S}[\mathsf{P_2}] \rightsquigarrow \mathsf{s_2}.$$

This in turn suffices to prove the equivalence of (the property-free characterization for) the robust preservation of trace properties and relational trace properties ($k = 2$ below)

$$\mathsf{RTC}^{\sim} \equiv \forall \mathsf{P}\, \forall \mathbf{C_T}\, \forall t.\ \mathbf{C_T}[\mathsf{P}{\downarrow}] \rightsquigarrow t \Rightarrow \exists \mathsf{C_S}.\ \exists\, \mathsf{s} \sim t.\ \mathsf{C_S}[\mathsf{P}] \rightsquigarrow \mathsf{s}$$

$$\begin{aligned} k\mathsf{RTC}^{\sim} \equiv\ & \forall \mathsf{P_1}, \mathsf{P_2} \forall \mathbf{C_T}\, \forall t_1, t_2.(\mathbf{C_T}[\mathsf{P_i}{\downarrow}] \rightsquigarrow t_i,\ i = 1, 2) \Rightarrow \\ & \exists \mathsf{C_S}\ \exists \mathsf{s_1}, \mathsf{s_2}.(\mathsf{s_i} \sim t_i \wedge \mathsf{C_S}[\mathsf{P_i}] \rightsquigarrow \mathsf{s_i},\ i = 1, 2). \end{aligned}$$

In Appendix B.1 we use a similar argument to show that for a Source language with full reflection the robust preservation of relational trace properties and subset-closed hyperproperties is equivalent to the robust preservation of the robust preservation of trace properties, and a similar result holds for *Safety* and *krelHypersafety*.

Nondeterministic Composition of Contexts can be obtained for example in process calculi by means of parallel composition. For given $\mathsf{C_1}, \mathsf{C_2}$ and P the behavior of P linked with $\mathsf{C_1} \mid \mathsf{C_2}$ over approximate the behavior of P in each of the two contexts,

$$\mathtt{beh}((\mathsf{C_1} \mid \mathsf{C_2})[\mathsf{P}]) \supseteq \mathtt{beh}(\mathsf{C_1}[\mathsf{P}]) \cup \mathtt{beh}(\mathsf{C_2}[\mathsf{P}]).$$

Once again this suffices to prove the equivalence of (the property-free characterization for) the robust preservation of trace properties and relational trace properties, and similar results for *Safety* (see Appendix B.2 for more detail).

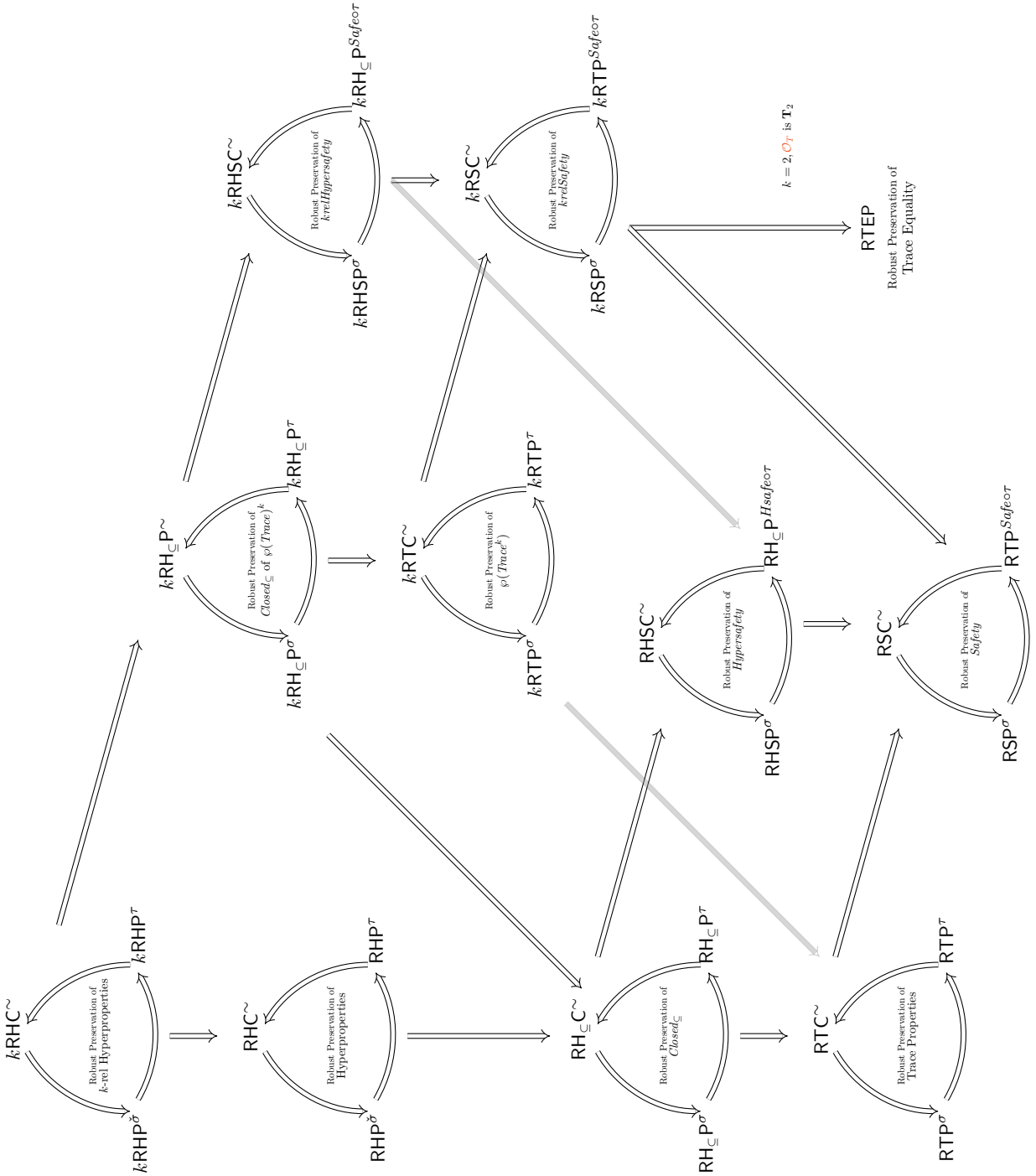

FIGURE 5.1: Main criteria for secure compilation and relations between them [7, Fig. 4]

5.4.2 Preservation of Trace Equality

Equality has a privileged role among relations, and similarly *trace equality* has a privileged role among relational hyperproperties. Under mild assumptions indeed, trace equality coincides with *observational equivalence* [49], and therefore its robust preservation can be considered to be (the security relevant implication of) fully abstract compilation [8, 93, 117], often elected as *the golden standard* for secure compilation and that we discuss in more detail in Section 5.5.
Abate et al. [8] showed that the preservation of trace equality – RTEP – in general does not imply any of the robust criteria, and does not ensure the robust preservation of security relevant notions, such as data integrity nor confidentiality. On the other hand RTEP follows immediately from the robust preservation of relational hyperproperties – kRHC$^{\sim}$ for $k = 2$ –, as trace equality is a relational hyperproperty of arity 2.
More interestingly, under mild assumptions, RTEP follows from the robust preservation of a class of relations that does not include trace equality, nor any other proper relation on sets of traces (relational hyperproperty), but only relation between single traces. Abate et al. [8, Theorem 5.1] show indeed that for a compiler between determinate languages[4], RTEP follows from the robust preservation of relational safety, kRSC$^{\sim}$, $k = 2$. They also need to assume *input totality* of target programs[5], and that the silent divergence of a program is reported on a finite prefix of the corresponding execution trace[6]. In order to understand the need for the mentioned assumptions we propose a

Sketch of the proof by Abate et al. [8, Theorem 5.1]

Assuming kRSC$^{\sim}$ for $k = 2$ and $\sim$ being the identity, i.e.,

$$\forall \mathsf{P}_1\mathsf{P}_2\ \forall \mathbf{C_T}\ \forall x_1x_2.\ \mathbf{C_T}[\mathsf{P}_1{\downarrow}] \rightsquigarrow^* x_1 \wedge \mathbf{C_T}[\mathsf{P}_2{\downarrow}] \rightsquigarrow^* x_2 \Rightarrow \exists \mathsf{C_S}.\ \mathsf{C_S}[\mathsf{P}_1] \rightsquigarrow^* x_1 \wedge \mathsf{C_S}[\mathsf{P}_2] \rightsquigarrow^* x_2$$

it is shown the contrapositive of RTEP, i.e.,

$$\forall \mathsf{P}_1\mathsf{P}_2\ \forall \mathbf{C_T}.\ \mathtt{beh}(\mathbf{C_T}[\mathsf{P}_1{\downarrow}]) \neq \mathtt{beh}(\mathbf{C_T}[\mathsf{P}_2{\downarrow}]) \Rightarrow \exists \mathsf{C_S}.\ \mathtt{beh}(\mathsf{C_S}[\mathsf{P}_1]) \neq \mathtt{beh}(\mathsf{C_S}[\mathsf{P}_2]).$$

Notice that $\mathtt{beh}(\mathbf{C_T}[\mathsf{P}_1{\downarrow}]) \neq \mathtt{beh}(\mathbf{C_T}[\mathsf{P}_2{\downarrow}])$ implies the existence of traces $t_1 \neq t_2$ such that $\mathbf{C_T}[\mathsf{P}_i{\downarrow}] \rightsquigarrow t_i$ but $\mathbf{C_T}[\mathsf{P}_i{\downarrow}] \not\rightsquigarrow t_j$ for $i \neq j$. In what follows every bullet shows which hypothesis is used for the rest of the proof:

- $t_1 \neq t_2$ can be witnessed with a pair of finite x-prefixes $x_1 \leq t_1,\ x_2 \leq t_2$ such that $x_1 \neq x_2$, therefore the trace relation

  $$r = \{(t, t') \mid \neg(x_1 \leq t \wedge x_2 \leq t')\}$$

  is in $2relSafety$, and since it is violated in a target context, it is also violated in some source context $\mathsf{C_S}$ (by kRSC$^{\sim}$), so that $\mathsf{C_S}[\mathsf{P}_1] \rightsquigarrow^* x_1$ and $\mathsf{C_S}[\mathsf{P}_2] \rightsquigarrow^* x_2$ or (vicervesa), but to conclude one still need to show it's not the case the same program has two execution with the two x-prefixes, e.g., $\mathsf{C_S}[\mathsf{P}_1] \rightsquigarrow^* x_1, x_2$.

[4] absence of internal nondeterminism, model by requiring that if two execution traces differ, then they differ for the first time on an input.

[5] programs cannot reject any input, still they can diverge or terminates or encounter an undefined behavior when a bad input is given.

[6] this is possible by using *x-prefixes* and the topology from Example 7.

- Notice that the first event on which x_1, x_2 above differ is not an input (by target input totality) and therefore
- x_1, x_2 cannot be produced by the same source program (by source determinacy), thus concluding the proof.

Our topological approach to classes of (relational) hyperproperties from Chapter 2, allows to express the first point above as a *separability* requirement on the trace model,i.e., the hypothesis $\mathcal{O}$ is $\mathbf{T}_2$, and (a condition weaker than the conjunction of) the remaining two[7] as following.

Definition 19 (Abstract Determinacy)**.** A language is determined w.r.t a trace model $(Trace, \mathcal{O})$ *iff* the behaviors of programs cannot meet disjoint non empty open sets,i.e., *iff* $\forall C \in \mathcal{C}tx.\ \forall P \in \mathcal{P}rg.\ \forall A_1 A_2 \in \mathcal{O}$ such that $A_1 \cap A_2 = \emptyset$ and $A_1, A_2 \neq \emptyset$,

$$\mathtt{beh}(C[P]) \cap A_1 \neq \emptyset \Rightarrow \mathtt{beh}(C[P]) \cap A_2 = \emptyset.$$

□

The following theorem generalizes the results from Abate et al. [8, Theorem 5.1],

Theorem 12 (Robust Preservation of *krelSafety* and Trace Equality)**.** *Assume* $Trace_S = Trace_T, \mathcal{O}_S = \mathcal{O}_T$ *and* $\sim \subseteq Trace_S \times Trace_T$ *is the identity, hence the corresponding* τ *and* σ *are the identity too.*

If the target topology $\mathcal{O}_T$ *is* $\mathbf{T}_2$ *(Definition 3) and the source language is determined according to Definition 19, then* $\mathsf{2RSC}^{=} \Rightarrow \mathsf{RTEP}$ *where*

$$\begin{aligned}\mathsf{2RSC}^{=} \equiv \forall \mathsf{P}_1\mathsf{P}_2\ \forall \mathbf{C_T}\ \forall A \in \mathcal{O} \times \mathcal{O}.\\ beh(\mathbf{C_T}[\mathsf{P}_1\downarrow]) \times beh(\mathbf{C_T}[\mathsf{P}_2\downarrow]) \cap A \neq \emptyset \Rightarrow\\ \exists \mathsf{C_S}.\ beh(\mathsf{C_S}[\mathsf{P}_1]) \times beh(\mathsf{C_S}[\mathsf{P}_1]) \cap A \neq \emptyset\end{aligned}$$

and

$$\begin{aligned}\mathsf{RTEP} \equiv \forall \mathsf{P}_1\mathsf{P}_2.\ (\forall \mathsf{C_S}.\ beh(\mathsf{C_S}[\mathsf{P}_1]) = beh(\mathsf{C_S}[\mathsf{P}_2])) \Rightarrow\\ (\forall \mathbf{C_T}.\ beh(\mathbf{C_T}[\mathsf{P}_1\downarrow]) = beh(\mathbf{C_T}[\mathsf{P}_2\downarrow]))\end{aligned}$$

Proof. We show the contrapositive of RTEP,

$$\begin{aligned}\forall \mathsf{P}_1\mathsf{P}_2\ \forall \mathbf{C_T}.\ \mathtt{beh}(\mathbf{C_T}[\mathsf{P}_1\downarrow]) \neq \mathtt{beh}(\mathbf{C_T}[\mathsf{P}_2\downarrow]) \Rightarrow\\ \exists \mathsf{C_S}.\ \mathtt{beh}(\mathsf{C_S}[\mathsf{P}_1]) \neq \mathtt{beh}(\mathsf{C_S}[\mathsf{P}_2]).\end{aligned}$$

Assume $\mathsf{2RSC}^{=}$ and assume $\mathtt{beh}(\mathbf{C_T}[\mathsf{P}_1\downarrow]) \neq \mathtt{beh}(\mathbf{C_T}[\mathsf{P}_2\downarrow])$. Without loss of generality we can assume there exist $t_1 \neq t_2$ such that $\mathbf{C_T}[\mathsf{P}_i\downarrow] \rightsquigarrow t_i$ and $\mathbf{C_T}[\mathsf{P}_i\downarrow] \not\rightsquigarrow t_j$ for $i \neq j$. Since the topology is $\mathbf{T}_2$, there exist open sets $A_1, A_2 \in \mathcal{O}$ such that $t_1 \in A_1,\ t_2 \in A_2$ and $A_1 \cap A_2 = \emptyset$.

Apply $\mathsf{2RSC}^{=}$ to $A = A_1 \times A_2 \in \mathcal{O} \times \mathcal{O}$ and deduce there exists $\mathsf{C_S}$ such that

$$\mathtt{beh}(\mathsf{C_S}[\mathsf{P}_1]) \times \mathtt{beh}(\mathsf{C_S}[\mathsf{P}_2]) \cap A_1 \times A_2 \neq \emptyset$$

and hence $\mathtt{beh}(\mathsf{C_S}[\mathsf{P_i}]) \cap A_i \neq \emptyset,\ i = 1, 2$. Finally by abstract determinacy

$$\mathtt{beh}(\mathsf{C_S}[\mathsf{P}_1]) \cap A_2 = \emptyset \text{ and } \mathtt{beh}(\mathsf{C_S}[\mathsf{P}_2]) \cap A_1 = \emptyset,$$

[7] Notice that x_1 and x_2 indeed define disjoint open sets

that implies $\mathtt{beh}(\mathtt{C_S}[\mathtt{P_1}]) \neq \mathtt{beh}(\mathtt{C_S}[\mathtt{P_2}])$ and concludes the proof. □

In the next section we compare robust criteria from Figure 5.1 with the preservation of observational equivalence. We do not approximate observational equivalence with equality of traces of *externally observable* events as done by Abate et al. [8], but rather instantiate the robust criteria presented so far with a more abstract notion of traces that collect the observables of the language itself,e.g., the values of the language.

5.5 Fully Abstract Compilation

We already discussed in Chapter 2 that the main motivation for the introduction of hyperproperties was the impossibility of specifying relations between multiple executions (or multiple systems), and therefore for reasoning about security notions such as confidentiality.
Rather than adopting the language of hyperproperties, confidentiality of communication protocols, can be defined as an *equivalence* to an ideal protocol known to enjoy confidentiality [32]. The equivalence ensures that no attacker can *observe* any difference with the ideal, hence secure protocol. A similar equivalence-based approach is possible for defining privacy [3], strong secrecy [24], or authentication [5].
Compilers $\cdot\downarrow : \mathtt{Source} \to \mathtt{Target}$ that translate equivalent source expressions to equivalent target expressions (*preservation* of the equivalence) and non-equivalent source expressions to non-equivalent target expressions (*reflection* of the equivalence) are called *fully abstract* [1].

Given the above premises, it's legit to expect that fully abstract compilers ensure that source and translated expressions enjoy the same security properties defined through equivalences. It is worth however, to be pedantic on a few points:

- different security notions may be defined by means of multiple equivalences that coincide only under extra hypothesis [32, 49], and even in this last case,
- what is guaranteed on compiled expressions for all the other trace properties and hyperproperties (robustly) satisfied in by source expressions? (see [6]).

In the rest of this section, we focus on contextual or *observational* equivalence, that is the most commonly adopted in the literature on fully abstract compilation, see e.g., [1, 27, 93, 31, 96, 80, 42, 110]. Two expressions of a language are observationally equivalent *iff* they lead to the same *observable* results, and in any context of the language.

The following example shows that, when a security property is not captured by observational equivalence there is no guarantee it will be (robustly) preserved by a fully abstract compiler[8],

Example 15 (Fully Abstract Compilation $\not\Rightarrow$ Robust *Safety* Preservation)**.** Let the observables of $\mathtt{Source}$ and $\mathtt{Target}$ be natural numbers and source (target) programs to be functions from booleans (natural numbers) to natural numbers,

$$\mathcal{P}rg = \mathbb{B} \to \mathbb{N}$$
$$\mathcal{P}rg = \mathbb{N} \to \mathbb{N}$$

Both source and target contexts can call a program (a function) on a certain boolean or natural number and observe the result; observational equivalence is simply pointwise equality of functions (same outputs on same inputs).

[8]The following example adapts [6, Example 1] and hence also the example from [8, Appendix E.5] to the formalism used in Section 5.5.1.

For coherence with the formalism in the next section notice that observables can be seen as particular programs,i.e., constant functions and contexts can be modeled as functions on programs, contexts transform a program in a constant function,

$$\mathsf{C_S} \in \mathcal{C}tx \iff \exists b \in \mathbb{B}.\ \forall \mathsf{P} : \mathbb{B} \to \mathbb{N}.\ \mathsf{C_S}[\mathsf{P}] = \lambda\_.\ \mathsf{P}(b)$$
$$\mathbf{C_T} \in \mathcal{C}tx \iff \exists n \in \mathbb{N}.\ \forall \mathbf{Q} : \mathbb{N} \to \mathbb{N}.\ \mathbf{C_T}[\mathbf{Q}] = \lambda\_.\ \mathbf{Q}(n)$$

Identify $\mathbb{B}$ with $\{0,1\} \subseteq \mathbb{N}$ and consider a compiler that translates a function to "the same" function when restricted to $\{0,1\}$ and returns a default value – playing the role of a bug – for the other natural numbers,

$$\mathsf{P}{\downarrow} = \lambda n \in \mathbb{N}.\ \begin{cases} \mathsf{P}(\texttt{true}) & \text{if } n = 0 \\ \mathsf{P}(\texttt{false}) & \text{if } n = 1 \\ 42 & \text{otherwise} \end{cases}$$

The above compiler preserves and reflects observational equivalence, but does not robustly preserve the safety property "never observe 42". □

Nevertheless the literature is rich of examples of fully abstract compiler that robustly preserve interesting hyperproperties such as noninterference [27], isolation [31], well-bracketed control flow and local state encapsulation [110].

Abate, Busi, and Tsampas [6] investigated the relation between fully abstract and robust compilation in the framework of *Mathematical Operational Semantics*. In particular they provide a sufficient condition that makes fully abstract compilers also robustly preserve arbitrary hyperproperties, under the assumption that observational equivalence coincides with bisimilarity.

In the next section we focus on the canonical definition of observational equivalence by Mitchell [80], the canonical model of contexts as "programs with a hole", and without the burden of the categorical machinery coming from the Mathematical Operational Semantics framework.

5.5.1 When Fully Abstract Compilers are Robust?

In the scenario we consider in this section we mainly follow Mitchell [80, Section 3], for the modeling of Source and **Target** languages. The programs of each language are expressions of the language, contexts are intended as "programs with a hole" and modeled as total functions that transform programs into programs[9], in particular we assume the existence of the *identity* context for any language.

The *observables* of a language are a subset of the expressions of the language, $Obs \subseteq \mathcal{P}rg$ that intuitively correspond to the "printable values of the language" (see [80, Section 3]). Every expression evaluates to a (possibly empty) set of observables according to a relation $\xrightarrow{\texttt{eval}} \subseteq \mathcal{P}rg \times \wp(Obs)$, with the convention that every observable $a \in Obs$ evaluates only to itself,i.e., $a \xrightarrow{\texttt{eval}} \{a\}$.

Definition 20 (Observational equivalence)**.** Two programs P_1, P_2 are observationally equivalent *iff* in the same context they evaluate to the same observables,

$$P_1 \overset{\texttt{obs}}{=} P_2 \iff (\forall C \in \mathcal{C}tx\ \forall A \in \wp(Obs).\ C[P_1] \xrightarrow{\texttt{eval}} A \iff C[P_2] \xrightarrow{\texttt{eval}} A)$$

Observational equivalence is an equivalence relation. □

[9] there is no need to consider partial function as partiality is given the evaluation relation.

Remark 14 ($\overset{\texttt{obs}}{=}$ is syntactic equality on Obs)**.** If $a_1, a_2 \in Obs$ and $a_1 \overset{\texttt{obs}}{=} a_2$, then they evaluate to the same observable in any context, in particular in the identity context,

$$a_1 \xrightarrow{\texttt{eval}} \{a\} \iff a_2 \xrightarrow{\texttt{eval}} \{a\}$$

where however they evaluate to themselves,i.e., $a_1 = a = a_2$. □

In this model, Source and Target languages are tuples

$$\mathsf{Source} = (\mathcal{P}rg,\ \mathcal{C}tx \subseteq (\mathcal{P}rg \to \mathcal{P}rg),\ Obs_S \subseteq \mathcal{P}rg, \xrightarrow{\mathsf{eval}} \subseteq \mathcal{P}rg \times \wp(Obs_S))$$
$$\mathbf{Target} = (\mathcal{P}rg,\ \mathcal{C}tx \subseteq (\mathcal{P}rg \to \mathcal{P}rg),\ Obs_T \subseteq \mathcal{P}rg, \xrightarrow{\mathsf{eval}} \subseteq \mathcal{P}rg \times \wp(Obs_T))$$

notice that linking in both languages is function application and a compiler is a function $\cdot\!\downarrow : \mathcal{P}rg \to \mathcal{P}rg$.

The following definition generalizes the notion of *abstraction-preserving reduction* by Mitchell [80]. To help her intuition, the reader can imagine a language reduction as a logical relation between target and source expressions, that relates target observationally equivalent terms to source observationally equivalent ones.

Definition 21 (Language Reduction)**.** Given a compiler $\cdot\!\downarrow : \mathsf{Source} \to \mathbf{Target}$, a reduction from Target to Source w.r.t. $\cdot\!\downarrow$ is a relation $\vartheta \subseteq \mathbf{Target} \times \mathsf{Source}$ such that

(i) ϑ is total on target contexts and compiled programs,

$$\forall \mathbf{C_T}\ \forall \mathsf{P}.\ \exists \mathsf{W}.\ \mathbf{C_T}[\mathsf{P}\!\downarrow]\ \vartheta\ \mathsf{W}$$

(ii) ϑ is compositional,i.e.,

$$\forall \mathbf{C_T}\ \forall \mathsf{P}\ \forall \mathsf{W}.\ \mathbf{C_T}[\mathsf{P}\!\downarrow]\ \vartheta\ \mathsf{W} \Rightarrow \exists \mathsf{C_S} \exists \mathsf{P'}.\ (\mathbf{C_T}\ \vartheta\ \mathsf{C_S}) \wedge (\mathsf{P}\!\downarrow\ \vartheta\ \mathsf{P'}) \wedge \mathsf{W} \overset{\texttt{obs}}{=} \mathsf{C_S}[\mathsf{P'}].$$

(iii) ϑ preserves and reflect observational equivalence,

$$\forall \mathsf{P}_1, \mathsf{P}_2.\ \mathsf{P}_1 \overset{\texttt{obs}}{=} \mathsf{P}_2 \iff (\forall \mathbf{Q_1}\ \vartheta\ \mathsf{P}_1.\ \forall \mathbf{Q_2}\ \vartheta\ \mathsf{P}_2.\ \mathbf{Q_1} \overset{\texttt{obs}}{=} \mathbf{Q_2})$$

When ϑ is a total function, the definition coincides with the one of abstraction-preserving reduction by Mitchell [80]. □

Example 16 (ϑ from the pre-image of a compiler)**.** Let Source and Target be as in Example 15, but this time identify $\mathbb{B}$ with $\mathbb{N}/\equiv_2$, the equivalence classes of natural numbers modulo 2, and modify the compiler accordingly to

$$\mathsf{P}\!\downarrow = \lambda n \in \mathbb{N}.\ \begin{cases} \mathsf{P}(\texttt{true}) & \text{if } n \equiv 0\ (\texttt{mod}\ 2) \\ \mathsf{P}(\texttt{false}) & \text{if } n \equiv 1\ (\texttt{mod}\ 2) \end{cases}$$

Define $\vartheta \subseteq \mathbf{Target} \times \mathsf{Source}$ as following:

- relate every target observable,i.e., a natural number (a constant function $\mathbb{N} \to \mathbb{N}$), to itself (a constant function $\mathbb{B} \to \mathbb{N}$), $\forall \mathbf{n} \in \mathbb{N}.\ \mathbf{n}\ \vartheta\ \mathsf{n}$

- for any $\mathsf{P}, \mathsf{P}{\downarrow}\ \vartheta\ \mathsf{P}$
- let $\mathbf{C_T} \in \mathcal{C}tx$ and $n \in \mathbb{N}$ be such that

$$\forall \mathbf{Q} : \mathbb{N} \to \mathbb{N}.\ \mathbf{C_T}[\mathbf{Q}] = \lambda\_.\ \mathbf{Q}(n)$$

then $\mathbf{C_T}\ \vartheta\ \mathsf{C_S}$, where $\mathsf{C_S}$ corresponds to n (`mod` 2),i.e.,

$$\forall \mathsf{P} : \mathbb{B} \to \mathbb{N}.\ \mathsf{C_S}[\mathsf{P}] = \lambda\_.\ \mathsf{P}(n\ (\texttt{mod}\ 2))$$

□

We propose a bunch of technical lemmas and remarks useful for the main theorem of this section.

Lemma 17 (ϑ is $\mathbf{T}$-functional on $\overset{\mathsf{obs}}{=}$ equivalence classes)**.** *Let* $\vartheta \subseteq \mathbf{Target} \times \mathsf{Source}$ *be a language reduction as in Definition 21, then*

$$\forall \mathsf{P}.\ \forall \mathbf{Q}\mathbf{Q}'.\ \mathbf{Q}\ \vartheta\ \mathsf{P} \wedge \mathbf{Q}'\ \vartheta\ \mathsf{P} \Rightarrow \mathbf{Q} \overset{\mathsf{obs}}{=} \mathbf{Q}'$$

Proof. From reflexivity of $\overset{\mathsf{obs}}{=}$ we have $\mathsf{P} \overset{\mathsf{obs}}{=} \mathsf{P}$ so that the thesis follows from (iii) of Definition 21. □

Lemma 18 (ϑ is $\mathbf{T}$-injective on $\overset{\mathsf{obs}}{=}$ equivalence classes)**.** *Let* $\vartheta \subseteq \mathbf{Target} \times \mathsf{Source}$ *be a language reduction as in Definition 21, then*

$$\forall \mathsf{P}\mathsf{P}'.\ \forall \mathbf{Q}.\ \mathbf{Q}\ \vartheta\ \mathsf{P} \wedge \mathbf{Q}\ \vartheta\ \mathsf{P}' \Rightarrow \mathsf{P} \overset{\mathsf{obs}}{=} \mathsf{P}'$$

Proof. We show the thesis by applying (iii) from Definition 21. Indeed let $\mathbf{Q_1}\ \vartheta\ \mathsf{P}$, $\mathbf{Q_2}\ \vartheta\ \mathsf{P}'$, then by Lemma 17 applied to $\mathsf{P}, \mathbf{Q}, \mathbf{Q_1}$, we deduce $\mathbf{Q_1} \overset{\mathsf{obs}}{=} \mathbf{Q}$. Similarly applying Lemma 17 to $\mathsf{P}', \mathbf{Q}, \mathbf{Q_2}$ we deduce $\mathbf{Q_2} \overset{\mathsf{obs}}{=} \mathbf{Q}$, so that the thesis follows by transitivity of observational equivalence. □

Remark 15 (left inverse of ϑ on Obs)**.** By Lemma 17 and Lemma 18, ϑ is a total injective function on the set of expressions quotiented w.r.t. observational equivalence. Moreover, observational equivalence on observables is syntactic equality (see Remark 14).

If ϑ relates target observables only with source ones,i.e.,

$$\forall\ A_T \in \wp(Obs_T).\ \vartheta(A_T) \in Obs_S,$$

where $\vartheta(A_T) = \{\mathsf{a} \mid \exists a \in A_T.\ a\ \vartheta\ \mathsf{a}\}$, then the restriction of ϑ to the set of target observables is a total injective function between source and target observables,

$$\vartheta_{|Obs_T} : Obs_T \to Obs_S.$$

Injectivity of $\vartheta_{|Obs_T}$ ensures the existence of a left inverse. □

We can finally state our definition of fully abstract compilation and show in Lemma 19 that it strengthen the usual definition of preservation and reflection of observational equivalence.

Definition 22 (Fully Abstract Compilation)**.** A compiler $\cdot{\downarrow} : \mathsf{Source} \to \mathbf{Target}$ is FAC *iff* there exists a reduction $\vartheta \subseteq \mathbf{Target} \times \mathsf{Source}$ that includes the pre-image of the compiler,i.e., $\forall\ \mathsf{P}.\ \mathsf{P}{\downarrow}\ \vartheta\ \mathsf{P}$. □

Lemma 19 (Reductions preserve and reflect contextual equivalence)**.** *A* FAC *compiler* $\cdot\downarrow : \mathsf{Source} \to \mathbf{Target}$ *preserves and reflect observational equivalence,i.e.,*

$$\forall \mathsf{P}_1\mathsf{P}_2.\ (\mathsf{P}_1 \overset{\mathbf{obs}}{=} \mathsf{P}_2) \iff (\mathsf{P}_1\downarrow \overset{\mathbf{obs}}{=} \mathsf{P}_2\downarrow)$$

Proof. Let $\mathsf{P}_1, \mathsf{P}_2$ be two source programs. By hypothesis there exists a language reduction $\vartheta \subseteq \mathbf{Target} \times \mathsf{Source}$ that includes the pre-image of $\cdot\downarrow$. In particular $\mathsf{P}_1\downarrow\ \vartheta\ \mathsf{P}_1$ and $\mathsf{P}_2\downarrow\ \vartheta\ \mathsf{P}_2$.

For preservation, assume $\mathsf{P}_1 \overset{\mathbf{obs}}{=} \mathsf{P}_2$. By (iii) from Definition 21 it follows that $\mathsf{P}_1\downarrow \overset{\mathbf{obs}}{=} \mathsf{P}_2\downarrow$.

For reflection, assume $\mathsf{P}_1\downarrow \overset{\mathbf{obs}}{=} \mathsf{P}_2\downarrow$. Then by Lemma 17, every $\mathbf{Q_i}\ \vartheta\ \mathsf{P_i}$ is observationally equivalent to $\mathsf{P_i}\downarrow, i = 1, 2$, and by transitivity of observational equivalence, $\mathbf{Q_1} \overset{\mathbf{obs}}{=} \mathbf{Q_2}$. The thesis follows from (iii) of Definition 21. □

We can finally answer to the question in the title of the current section, and show a relation between the target observables to which $\mathbf{C_T}[\mathsf{P}\downarrow]$ evaluates and the one P can evaluate in some source context $\mathsf{C_S}$ (Theorem 13). Then in Theorem 14 we show how this result can be intended as the robust preservation of (hyper)properties of observables.

Theorem 13 (A sufficient condition for the robustness of FAC)**.** *Let* $\vartheta \subseteq \mathbf{Target} \times \mathsf{Source}$ *be a reduction for* $\cdot\downarrow : \mathsf{Source} \to \mathbf{Target}$ *such that:*

1. $\forall\ \mathsf{P}.\ \mathsf{P}\downarrow\ \vartheta\ \mathsf{P}$*,i.e.,* ϑ *makes the compiler* FAC *,*
2. ϑ *relates target observables only with source ones,i.e.,*

$$\forall\ A_T \in \wp(Obs_T).\ \vartheta(A_T) \in Obs_S,$$

   where $\vartheta(A_T) = \{\mathsf{a} \mid \exists a \in A_T.\ a\ \vartheta\ \mathsf{a}\}$ *,*

3. *target programs simulate related source ones,i.e.,*

$$\forall A_T \in \wp(Obs_T)\ \forall \mathbf{W}\ \forall \mathsf{W}.\ \mathbf{W}\ \vartheta\ \mathsf{W} \Rightarrow \mathbf{W} \overset{\mathbf{eval}}{\longrightarrow} A_T \Rightarrow \mathsf{W} \overset{\mathsf{eval}}{\longrightarrow} \vartheta(A_T)$$

   Notice that the above is well posed because of hypothesis 2.

The following holds

$$\forall \mathsf{P}\ \forall \mathbf{C_T}.\ \exists \mathsf{C_S}.\ \forall A_T \in \wp(Obs_T).\ \mathbf{C_T}[\mathsf{P}\downarrow] \overset{\mathbf{eval}}{\longrightarrow} A_T \Rightarrow \mathsf{C_S}[\mathsf{P}] \overset{\mathsf{eval}}{\longrightarrow} \vartheta(A_T). \tag{5.5}$$

Proof. By (i) of Definition 21 there exists W such that

$$\mathbf{C_T}[\mathsf{P}\downarrow]\ \vartheta\ \mathsf{W}. \tag{5.6}$$

By (ii) of Definition 21 such a W is observationally equivalent to the linking of some P' and some $\mathsf{C_S}$ related by ϑ to $\mathsf{P}\downarrow$ and $\mathbf{C_T}$ respectively, i.e.,

$$\exists \mathsf{C_S}\ \exists \mathsf{P}'.\ (\mathbf{C_T}\ \vartheta\ \mathsf{C_S}) \wedge (\mathsf{P}\downarrow\ \vartheta\ \mathsf{P}') \wedge \mathsf{W} \overset{\mathbf{obs}}{=} \mathsf{C_S}[\mathsf{P}'].$$

Since $\mathsf{P}{\downarrow}$ is related to both P' and P, by Lemma 18, $\mathsf{P}' \overset{\mathtt{obs}}{=} \mathsf{P}$ so that, by definition of observational equivalence, for any $A_T \in Obs_T$,

$$\mathsf{C_S}[\mathsf{P}] \xrightarrow{\mathsf{eval}} \vartheta(A_T) \iff \mathsf{C_S}[\mathsf{P}'] \xrightarrow{\mathsf{eval}} \vartheta(A_T) \iff \mathsf{W} \xrightarrow{\mathsf{eval}} \vartheta(A_T). \tag{5.7}$$

Finally assume $\mathbf{C_T}[\mathsf{P}{\downarrow}] \xrightarrow{\mathsf{eval}} A_T$. Then by by hypothesis 3, and Equation (5.6), we have $\mathsf{W} \xrightarrow{\mathsf{eval}} \vartheta(A_T)$, and therefore the thesis by Equation (5.7). □

Remark 16. The reduction from Example 16 satisfies the hypothesis of Theorem 13. □

The reader might have noticed similarities between the conclusions of Theorem 13 and the criterion for the robust preservation of hyperproperties – $\mathsf{RHC}^{\sim}$– from Section 5.4. Indeed, by elementary manipulations, it's possible to write Equation (5.5) as

$$\forall \mathsf{P}\ \forall \mathbf{C_T}.\ \exists \mathsf{C_S}.\ \vartheta(\xrightarrow{\mathsf{eval}} \mathbf{C_T}[\mathsf{P}{\downarrow}]) = (\xrightarrow{\mathsf{eval}} \mathsf{C_S}[\mathsf{P}]) \tag{5.8}$$

where $\xrightarrow{\mathtt{eval}} W$ denotes $A \in \wp(Obs)$ such that $W \xrightarrow{\mathtt{eval}} A$.

Moreover by Remark 15, ϑ has a left inverse (on Obs), that with a little abuse of notation we denote by $\vartheta^{-1} : Obs_S \to Obs_T$. Therefore Equation (5.5) is equivalent to

$$\forall \mathsf{P}\ \forall \mathbf{C_T}.\ \exists \mathsf{C_S}.\ (\xrightarrow{\mathsf{eval}} \mathbf{C_T}[\mathsf{P}{\downarrow}]) = \vartheta^{-1}(\xrightarrow{\mathsf{eval}} \mathsf{C_S}[\mathsf{P}]) \tag{5.9}$$

We explicitly notice that ϑ^{-1} is in general the *upper* adjoint of ϑ, while the formula for robust preservation of hyperproperties involves a *lower* adjoint (see Lemma 9). This means that Equation (5.9) in general *does not* fit in the framework proposed so far[10], and we cannot conclude *a priori* it is equivalent to the robust preservation of hyperproperties (of Obs). Still we can prove it is a sufficient condition for it.

Theorem 14 (FAC implies the robust preservation of $\wp(\wp(Obs))$)**.** *Let $\vartheta \subseteq \mathbf{Target} \times \mathsf{Source}$ be a reduction for $\cdot{\downarrow} : \mathsf{Source} \to \mathbf{Target}$ that satisfies the same hypothesis as Theorem 13. The following holds:*

$$\forall \mathsf{P}\ \forall H_T \in \wp(\wp(Obs_T)).\ (\forall \mathsf{C_S}.\ (\xrightarrow{\mathsf{eval}} \mathsf{C_S}[\mathsf{P}]) \in \vartheta(H_T)) \Rightarrow (\forall \mathsf{C_T}.\ (\xrightarrow{\mathsf{eval}} \mathbf{C_T}[\mathsf{P}{\downarrow}]) \in H_T)$$

where $\vartheta(H_T) = \{\vartheta(A_T) \mid A_T \in H_T\}$.

Proof. Let $\mathsf{P} \in \mathcal{P}rg$ and $H_T \in \wp(\wp(Obs_T))$ and assume $\forall \mathsf{C_S}.\ (\xrightarrow{\mathsf{eval}} \mathsf{C_S}[\mathsf{P}]) \in \vartheta(H_T))$, that means

$$\forall \mathsf{C_S}.\ \exists A_T \in H_T.\ (\xrightarrow{\mathsf{eval}} \mathsf{C_S}[\mathsf{P}]) = \vartheta(A_T) \tag{5.10}$$

For any $\mathbf{C_T}$, by Equation (5.8) there is some $\mathsf{C_S}$ such that

$$(\xrightarrow{\mathsf{eval}} \mathsf{C_S}[\mathsf{P}]) = \vartheta(\xrightarrow{\mathsf{eval}} \mathbf{C_T}[\mathsf{P}{\downarrow}]),$$

that together with Equation (5.10) implies

$$\exists A_T \in H_T.\ \vartheta(\xrightarrow{\mathsf{eval}} \mathbf{C_T}[\mathsf{P}{\downarrow}]) = \vartheta(A_T),$$

[10] It fits in the framework if ϑ is also surjective.

from which by injectivity (on observables) of ϑ (see Remark 15), it follows

$$\exists A_T \in H_T.\ (\xrightarrow{\mathbf{eval}} \mathbf{C_T}[\mathsf{P}\downarrow]) = (A_T),$$

i.e., $(\xrightarrow{\mathbf{eval}} \mathbf{C_T}[\mathsf{P}\downarrow]) \in H_T$, that concludes the proof. □

Notes on the Robustness of Fully Abstract Compilers. Theorem 13 and Theorem 14 refer to the observables of Source and **Target** that – following Mitchell [80] – are a subset of the expressions of the two languages. Some of the technical lemmas involved in the presented theorems, strongly rely on this choice, therefore the results presented in this section show robustness of fully abstract compilers only for the – quite restrictive – scenario in which $Trace = Obs \subseteq \mathcal{P}rg$, and not for traces composed of events *external* to expressions of the language, such as inputs and outputs. For the latter trace model, there are indeed multiple examples of compilers that preserve and reflect observational equivalence, and don't enjoy any notion of robustness among those of Section 5.4, see e.g., [8, 6, 93].
Another, less worrying, limitation is the injectivity of ϑ, that highlights a fact already known to Parrow [89] and Gorla and Nestmann [61]: the observational equivalence of **Target** should be coarser than the observational equivalence of Source , and as a consequence it must be possible to encode (injectively) all the observables of **Target** in the observables of Source .

5.6 More Comparative Work

In this section we expose previous work in which the author contributed at comparing fully abstract compilation with the notion of robust preservation [6]. The motivation of the work of Abate, Busi, and Tsampas [6] has to be found in the already discussed examples showing that fully abstract compilers in general do not preserve data integrity nor confidentiality. Notice that similar examples do not explain *what* hyperproperties are robustly preserved by fully abstract compilers, that is the starting point of the query of Abate, Busi, and Tsampas [6], who for every fully abstract compiler, provide a map $\tilde{\tau} : \wp(\wp(\mathsf{Trace_S})) \to \wp(\wp(\mathbf{Trace_T}))$ that describes the guarantees given by the compiler, in the same way discussed in previous sections: if P robustly satisfies H, then P↓ robustly satisfies $\tilde{\tau}(\mathsf{H})$ [6, Theorem 1]. Despite its *optimality*, $\tilde{\tau}$ necessitates of several information on the compiler itself, making the actual evaluation of $\tilde{\tau}(\mathsf{H})$ prohibitive, if not impossible. In order to overcome this issue and have an immediate and easy evaluation of $\tilde{\tau}(\mathsf{H})$, Abate, Busi, and Tsampas [6] adopt the framework of Mathematical Operational Semantics framework (MOS) [118] and express both fully abstract and robust compilation within it.

5.6.1 Overview of Mathematical Operational Semantics

The main idea of MOS consists of describing the semantics of programming languages, or systems in general, by relating the evaluation of syntactic terms and the events exposed during such an evaluation. This relation is expressed formally as a *distributive law*, i.e., natural transformation[11] relating:

- a functor $\Sigma : \mathtt{Set} \to \mathtt{Set}$ that represents the algebraic signature of the language and thus acts as an abstract description of its syntax,

[11] for our puropouses it suffices to consider functors on the category `Set` of sets and total functions [66]

- a functor $B : \mathtt{Set} \to \mathtt{Set}$ that describes the behaviors of the language in terms of its observable events, e.g., the behavior of a non-deterministic labeled transition system can be modeled by $BX = \wp(X)^{\Delta}$, where Δ is a set of labels [122]);

More interestingly, a wide class of distributive laws corresponds to the so called GSOS rules [66], collections of formal rules that can be used to specify small-step operational semantics for many programming languges from the literature [122].
Before providing a concrete example we show how to retrieve programs and their execution traces when defining the semantics of a language for the distributive laws we are going to consider later on, that are natural transformations $\rho : \Sigma(\mathbb{I} \times B) \Longrightarrow B\Sigma^*$, where $\mathbb{I}$ is the identity functor Σ^* is the monad freely generated by Σ.

Remark 17 (Programs and Behaviors in MOS)**.**
Assume a syntax functor $\Sigma : \mathtt{Set} \to \mathtt{Set}$, and a behavior functor $B : \mathtt{Set} \to \mathtt{Set}$ are given.

The syntax funcotr has an initial algebra $A = \Sigma^*\emptyset$ that defines the set of closed terms of the language [66].

The behavior functor B has a final coalgebra $Z = B^{\infty}\top$ that defines the set of all possible behaviors [66].

A distributive law $\rho : \Sigma(\mathbb{I} \times B) \Longrightarrow B\Sigma^*$ induces a map $f : A \to Z$ that assignes to every closed term a behavior [66], playing the same role as $\mathtt{beh}(\cdot)$ in the MOS framework.

□

In the following example we provide a formal semantics to a source and a target languages by means of GSOS rules, and show what are the underlying syntax and behavior functors.

Example 17 ((Example 2 from [6]))**.** We consider two simple `WHILE` languages with a mutable state. A state $s \in S = (Var \to \mathbb{N})$ assigns to every variable a natural number. Variables are partioned in *high* or private or *low* or public, $Var = Var_H \uplus Var_L$. The only difference between the two languages consist in the set of contexts, that in the source contain the only identity, and in the target also $\lceil \cdot \rceil$, that can observe some internal information and report it externally (see **bang1**, **bang2** rules in Figure 5.2).
Assume $\oplus$ and U are binary and unary operators (resp.) on natural numbers, the following grammars generate the set of expressions

$$\langle \mathcal{E} \rangle ::= n \in \mathbb{N} \mid v \in Var \mid \langle \mathcal{E} \rangle \oplus \langle \mathcal{E} \rangle \mid U\langle \mathcal{E} \rangle$$

The following grammars generate the set of source and target programs and contexts

$$\langle \mathsf{P} \rangle ::= \mathsf{skip} \mid v := \mathcal{E} \mid \langle \mathsf{P} \rangle;\langle \mathsf{P} \rangle \mid \mathtt{WHILE}\ \ \mathcal{E}\ \langle \mathsf{P} \rangle$$

$$\langle \mathsf{C_S} \rangle ::= [\ \cdot\ ]$$

$$\langle \mathbf{P} \rangle ::= \mathbf{skip} \mid v := \mathcal{E} \mid \langle \mathbf{P} \rangle;\langle \mathbf{P} \rangle \mid \mathtt{WHILE}\ \ \mathcal{E}\ \langle \mathbf{P} \rangle$$

$$\langle \mathbf{C_T} \rangle ::= [\ \cdot\ ] \mid \lceil\ \cdot\ \rceil$$

The rules in Figure 5.2 define a small-step operational semantics for source and target languages, $\mathcal{H}, \mathbf{!} \notin S$. Syntax and behavior functors are respectively:

$$\frac{}{s, \mathsf{skip} \to s, \checkmark}\ \mathsf{skip}$$

$$\frac{v \in Var_H \quad s(v) \neq [e]_s}{s, v := e \to^{\mathcal{H}} s_{[v \leftarrow [e]_s]}, \checkmark}\ \mathsf{asnH} \qquad \frac{v \in Var_L \quad e \cap Var_H = \emptyset}{s, v := e \to s_{[v \leftarrow [e]_s]}, \checkmark}\ \mathsf{asnL}$$

$$\frac{s, \mathsf{P} \to s', \checkmark}{s, \mathsf{P}; \mathsf{Q} \to s', \mathsf{Q}}\ \mathsf{seq1} \qquad \frac{s, \mathsf{P} \to s', \mathsf{P}'}{s, \mathsf{P}; \mathsf{Q} \to s', \mathsf{P}'; \mathsf{Q}}\ \mathsf{seq2}$$

$$\frac{[e]_s = 0}{s, \mathtt{WHILE}\ \ e\ \mathsf{p} \to s, \mathsf{skip}}\ \mathsf{while1} \qquad \frac{[e]_s \neq 0}{s, \mathtt{WHILE}\ \ e\ \mathsf{p} \to s, \mathsf{p}; \mathtt{WHILE}\ \ e\ \mathsf{p}}\ \mathsf{while2}$$

$$\frac{s, \mathbf{P} \to^{\mathcal{H}} s', \checkmark}{s, \lceil \mathbf{P} \rceil \to^{!} s', \checkmark}\ \mathbf{bang1} \qquad \frac{s, \mathbf{P} \to^{\mathcal{H}} s', \mathbf{P}'}{s, \lceil \mathbf{P} \rceil \to^{!} s', \mathbf{P}'}\ \mathbf{bang2}$$

FIGURE 5.2: Operational semantics for the two languages. The rules for the target extend the one of the source with **bang1** and **bang2** Fig. 4., Fig.5. from [6]

$$\begin{aligned}
\Sigma\ X &\triangleq \top \uplus (\mathbb{N} \times E) \uplus (X \times X) \uplus (E \times X) \\
\mathsf{B}\ X &\triangleq (S \times (\mathtt{Maybe}\ \mathcal{H}) \times (X \uplus \checkmark))^S \\
\mathbf{\Sigma}\ X &\triangleq \Sigma\ X \uplus X \\
\mathbf{B}\ X &\triangleq (S \times (\mathtt{Maybe}\ (\mathcal{H} \uplus \mathbf{!})) \times (X \uplus \checkmark))^S
\end{aligned}$$

where E is the set of all expressions generated by the grammar $\mathcal{E}$, `Maybe` is the (functor underlying the) well-known maybe monad[12], and the extra information in the traget syntax functor models the target context that is not present in the source.

Source closed programs are given by $\mathsf{A} = \Sigma^*\emptyset$, that means they are freely generated by the grammar generated by the union of the grammars for partial programs and cotnexts,i.e., $\langle \mathsf{P} \rangle | \langle \mathsf{C_S} \rangle$.

Source behaviors $\mathsf{Z} = \mathsf{B}^\infty \top$ are functions that when applied to a state (possibly paired with other information) return a new state and another function.

Finally we give a taste of the distibutive law corresponding to the rules from Figure 5.2. The rules of seq1 and seq2 for sequential composition correspond to the

[12] https://en.wikibooks.org/wiki/Haskell/Understanding_monads/Maybe

following component of the distributive law for the source[13], $\rho_S : \Sigma(\mathbb{I} \times \mathsf{B}) \Longrightarrow \mathsf{B}\Sigma^*$

$$(\mathsf{W}_1, \mathsf{b}) \;;\; (\mathsf{W}_2, \mathsf{g}) \mapsto \lambda\, s. \begin{cases} (s', \delta, \mathsf{W}'; \mathsf{W}_2) & \text{if } \mathsf{b}(s) = (s', \delta, \mathsf{W}') \\ (s', \delta, \mathsf{W}_2) & \text{if } \mathsf{b}(s) = (s', \delta, \checkmark) \end{cases}$$

$\mathsf{W}_1,\ \mathsf{W}_2 \in X$ with X a generic set of terms, can be programs, contexts or programs within a context, and $\mathsf{f}, \mathsf{g} \in \mathsf{B}X$. The image of ρ_S is an element of $\mathsf{B}\Sigma^*X = (S \times (\texttt{Maybe}\ \mathcal{H}) \times (\Sigma^*X \uplus \checkmark)))^S$, depending on whether W_1 transitions to a term W' (applying seq2), or terminates with $\checkmark$ (applying seq1). □

Remark 18 (More concrete traces)**.** The set $\mathsf{Z} = \mathsf{B}^\infty\top$ is in bijective correspondence with the non-empty subsets of the set of the sequences of states possibly exposing also the symbol $\mathcal{H}$.

$$\mathit{Trace}_S = \{m\checkmark \;\mid\; m \in (S \cup \{\mathcal{H}\})^\omega\} \cup \{t \;\mid\; t \in (S \cup \{\mathcal{H}\})^\omega\}$$

and similarly $\mathbf{Z} = \mathbf{B}^\infty\top$ is in bijection with non-empty subsets of

$$\mathit{Trace}_T = \{m\checkmark \;\mid\; m \in (S \cup \{\mathcal{H}, \mathbf{!}\})^\omega\} \cup \{t \;\mid\; t \in (S \cup \{\mathcal{H}, \mathbf{!}\})^\omega\}\,.$$

An immediate benefit of this fact it the possibility of identifying hyperproperties on sequences of states and on functions. □

5.6.2 Fully Abstract Compilation in MOS

In MOS, *well-behaved* translations can be expressed as relations between the distributive law of the source and that of the target [121]. It has been observed by Tsampas et al. [117] that it's actually possible to recover fully abstract compilation as the following relation called "map of distributive laws."

Definition 23 (MoDL [117])**.** A map of distributive laws between $\rho_S : \Sigma(\mathbb{I} \times \mathsf{B}) \Longrightarrow \mathsf{B}\Sigma^*$ and $\rho_T : \mathbf{\Sigma}(\mathbb{I} \times \mathbf{B}) \Longrightarrow \mathbf{B}\mathbf{\Sigma}^*$ is a pair of natural transformations $s : \Sigma \Longrightarrow \mathbf{\Sigma}^*$ and $b : \mathsf{B} \Longrightarrow \mathbf{B}$ such that the following diagram commutes,

$$\begin{array}{ccc} \Sigma(\mathbb{I} \times \mathsf{B}) & \xrightarrow{\rho_S} & \mathsf{B}\Sigma^* \\ {\scriptstyle s^* \circ \Sigma(id \times b)}\downarrow & & \downarrow{\scriptstyle b \circ \mathsf{B}s^*} \\ \mathbf{\Sigma}(\mathbb{I} \times \mathbf{B}) & \xrightarrow{\rho_T} & \mathbf{B}\mathbf{\Sigma}^* \end{array}$$

where $s^* : \Sigma^* \Longrightarrow \mathbf{\Sigma}^*$ extends $s : \Sigma \Longrightarrow \mathbf{\Sigma}^*$ to a morphism of free monads, i.e., to terms of arbitrary depth via structural induction. □

Remark 19 (Compilers and property mappings from a MoDL)**.** The natural transfomation $s : \Sigma \Longrightarrow \mathbf{\Sigma}^*$ induces a map between $\mathsf{A} = \Sigma^*\emptyset$ and $\mathbf{A} = \mathbf{\Sigma}^*\emptyset$ [66], that in turn are the closed terms of source and target languages, see Remark 17. Therefore the map

$$s^*_\emptyset : \mathsf{A} \to \mathbf{A}$$

plays the role of a compiler in the MOS framework.

Similarly, the natural transformaltion $b : \mathsf{B} \Longrightarrow \mathbf{B}$ induces a map between $\mathsf{Z} = \mathsf{B}^\infty\top$ and $\mathbf{Z} = \mathbf{B}^\infty\top$ [66], that in turn are the possible source and target behaviors Remark 17. Therefore the map

$$b^\infty_\top : \mathsf{Z} \to \mathbf{Z}$$

[13] Σ^* is the freely generated monad over Σ and $\mathbb{I}$ the identity functor

can be used to interpret source hyperproperties intro target ones. □

Theorem 15 (MoDL as fully abstract compilers [117])**.**
If (s,b) is a MoDL *for* $\rho_S : \Sigma(\mathbb{I} \times \mathsf{B}) \Longrightarrow \mathsf{B}\Sigma^*$ *and* $\rho_T : \mathbf{\Sigma}(\mathbb{I} \times \mathbf{B}) \Longrightarrow \mathbf{B}\mathbf{\Sigma}^*$*, then the compiler* $\cdot\downarrow = s^*_\emptyset$ *preserves bisimilarity. As a consequence, when bisimilarity coincides with observational equivalence, e.g., in absence of internal nondeterminism [49],* MoDL *defines a fully abstract compiler.*

Example 18. For the languages in Example 17, consider the map s that define an embedding of the source terms in the target. Let b the natural transformation that (informally speaking) associates to every trace the trace itself and a copy of it where the occurences of $\mathcal{H}$ are repalced with $(\mathcal{H}, \mathbf{!})$.
The maps (s, b) are a MoDL, and the identity compiler $\cdot\downarrow = s^*_\emptyset = id$ preserves observational equivalence (by Theorem 15), but it is not robust as it does not preserve the robust satisfaction of noninterference[6, Example 2]. More precisely, the target guarantee on compiled programs that robustly satisfy source noninteference is strictly weaker than target noninterference. □

Example 18 is due to the following difference between bisimilarity and low-equivalence that in turn defines noninterference (see [6, Example 2] for all the detail):

- bisimilarity requires the (pointwise) equality of both states and occurrences of $\mathcal{H}$ and $\mathbf{!}$, this last one only in the target. In particular, both high and low variables must match in bisimilar programs and this fact holds in the source if and only if it holds in the target;
- low-equivalence ignores the extra symbol $\mathcal{H}$ and requires only low variables to match in the source. In the target instead, also the occurences of $\mathbf{!}$ must match. As a consequence the assignment of a high variable $v{:=}42, v \in Var_H$, is or is not reported with a $\mathbf{!}$ on the trace in the target context $\lceil v{:=}42 \rceil$ depending on $s(v) \neq 42$ or $s(v) = 42$, for s being the state right before the assignement.

5.6.3 Making Fully Abstract Compilers Robust

MoDL specifies a relation between the semantics of source programs with the semantics of compiled one, that ensures the behaviors of any two source programs are modified *uniformly* by the transformation b. Notice this implies the robust preservation of (some) relational hyperproperties but says nothing on the guarantees on a single compiled program against arbitrary target contexts.

In order to reason explicitely on target contexts Abate, Busi, and Tsampas [6] select inside the syntax functor the constructs $\mathcal{C}$ for the terms that are contexts,i.e., $\Sigma = \mathcal{C} \uplus \mathcal{T}$, and assume the distibutive law $\rho : \Sigma(\mathbb{I} \times B) \Longrightarrow B\Sigma^*$ can be expressed as

$$\rho = [B\ in_1 \circ \rho^1,\ \rho^2],$$

for natural transformations $\rho^1 : \mathcal{C}(\mathbb{I} \times B) \Longrightarrow B\mathcal{C}^*$ and $\rho^2 : \mathcal{T}(\mathbb{I} \times B) \Longrightarrow B\mathcal{T}^*$.

These mild assumptions allow to specify how compiled programs should behave in arbitrary target contexts, in a way that resemble the universal quantification in the property-free formulations of the criteria from Section 5.4.

Definition 24 (MMoDL [6])**.** A many layers map of distributive laws (MMoDL) between $\rho_S : \Sigma(\mathbb{I} \times \mathsf{B}) \Longrightarrow \mathsf{B}\Sigma^*$ and $\rho_T : \mathbf{\Sigma}(\mathbb{I} \times \mathbf{B}) \Longrightarrow \mathbf{B}\mathbf{\Sigma}^*$ is given by natural transformations $b : \mathsf{B} \Longrightarrow \mathbf{B}$ and $t : \mathcal{C} \Longrightarrow \mathcal{C}^*$ making the following diagram commute:

$$\begin{array}{ccccccc}
\mathcal{C}\Sigma(\mathbb{I}\times \mathsf{B}) & \xrightarrow{\mathcal{C}^*(\Sigma\ \mathtt{prj}_1,\ \rho_S)} & \mathcal{C}(\mathbb{I}\times \mathsf{B})\Sigma^* & \xrightarrow{t} & \mathcal{C}^*(\mathbb{I}\times \mathsf{B})\Sigma^* & \xrightarrow{{\rho_S}^1} & \mathsf{B}\mathcal{C}^*\Sigma^* \\
\downarrow{\scriptstyle \mathcal{C}^*(\Sigma\ \mathtt{prj}_1,\ \rho_S)} & & & & & & \downarrow{\scriptstyle b} \\
\mathcal{C}(\mathbb{I}\times \mathsf{B})\Sigma^* & \xrightarrow{\mathcal{C}(\mathbb{I}\times b)} & \mathcal{C}(\mathbb{I}\times \mathbf{B})\Sigma^* & \xrightarrow{{\rho_T}^1} & \mathbf{B}\mathcal{C}^*\Sigma^* & \xrightarrow{\mathbf{B}t^*} & \mathbf{B}\mathcal{C}^*\Sigma^*
\end{array}$$

□

The top-left object, $\mathcal{C}\Sigma(\mathbb{I}\times \mathsf{B})$, represents a target context linked with some source term. In both paths, the inner source terms are initially evaluated w.r.t. the source semantics ρ_S, then

> In the upper path, the target context is *back-translated* [42] by t, executed w.r.t. the source semantics (${\rho_S}^1$), and finally the resulting behavior translated back to the target via b.
>
> In the lower path, first the resulting behavior is translated through $\mathcal{C}(\mathbb{I}\times b)$, then the target context executes w.r.t. the target semantics (${\rho_T}^1$), and the behavior of the target context is interpreted in the source via $\mathbf{B}t^*$.

Before showing that MMoDL actually ensures robustness, we need to expressed robust compilation within the MOS framework.

Robust Preservation in MOS. Recall that $A = \Sigma_\emptyset^* = (\mathcal{C}_\emptyset \uplus \mathcal{T}_\emptyset)^*$ and $Z = B^\infty\top$ are respectively the programs of a language and their possible behaviors Remark 17. A compiler $\cdot\!\downarrow : \mathsf{A} \to \mathbf{A}$ is RHP^τ in MOS and w.r.t an interpretation of source behaviors into target ones $\tau : \mathsf{Z} \to \mathbf{Z}$, *iff* there exists $bk : \mathcal{C}_\emptyset^* \times \mathcal{T}_\emptyset^* \to \mathcal{C}_\emptyset^*$ such that

$$\tau \circ \mathsf{f} \circ bk = \mathbf{f} \circ (id \times \cdot\!\downarrow). \tag{5.11}$$

The reader will recognize in Equation (5.11) the property-free formulation of RHP^τ from Section 5.4,

$$\begin{array}{l}\forall \mathsf{P} \in \mathcal{T}^*\ \forall \mathbf{C_T} \in \mathcal{C}_\emptyset^*.\ \exists \mathsf{C_S} \in \mathcal{C}_\emptyset^*. \\ \qquad \mathtt{beh}(\mathbf{C_T}[\mathsf{P}\!\downarrow]) = \tau(\mathtt{beh}(\mathsf{C_S}[\mathsf{P}]))\end{array}$$

where $\mathcal{T}_\emptyset^* = \mathcal{P}rg$, $\mathcal{C}_\emptyset^* = \mathcal{C}tx$, $\mathcal{C}_\emptyset^* = \mathcal{C}tx$ and $\mathsf{C_S}$ is given by the back-translation function bk,i.e., $\mathsf{C_S} = bk(\mathbf{C_T}, \mathsf{P})$.

Theorem 16 (Fully abstract and Robust (Theorem 2 in [6]))**.** *Let $s : \Sigma \Longrightarrow \mathbf{\Sigma}^*$, $b : \mathsf{B} \Longrightarrow \mathbf{B}$ and $t : \mathcal{C} \Longrightarrow \mathcal{C}$ such that (s, b) and (t, b) are (respectively) a MoDL and a MMoDL from $\rho_S : \Sigma(\mathbb{I}\times \mathsf{B}) \Longrightarrow \mathsf{B}\Sigma^*$ to $\rho_T : \mathbf{\Sigma}(\mathbb{I}\times \mathbf{B}) \Longrightarrow \mathbf{B}\mathbf{\Sigma}^*$.*
The compiler $\cdot\!\downarrow = s_\emptyset^*$ preserves bisimilarity (by Theorem 15) and is RHP^τ for $\tau = b_\top^\infty$, as $bk = t_\emptyset^* \times id$ satisfies Equation (5.11).

Finally we show which extra protection must be taken for compiled programs from Example 18 to ensure the compiler is also a MMoDL and therefore robust, this time for τ that maps source noninterference to a hyperproperty that implies target noninterference.

Example 19 (Example 3 in [6])**.** The intuition to protect against the target context $\lceil\cdot\rceil$ consists in *sand-boxing* programs when linked within that context. Formally the target language is extended with a constructor that sandboxes variable assignements[14],

$$\langle \mathbf{P} \rangle ::= \mathbf{skip} \mid v := \mathcal{E} \mid \langle \mathbf{P} \rangle;\langle \mathbf{P} \rangle \mid \mathtt{WHILE}\ \ \mathcal{E}\ \langle \mathbf{P} \rangle \mid \lfloor v := \mathcal{E} \rfloor$$

[14] alternatively the constructor could be implemented externally to the language, e.g., in hardware

the semantics of sandboxing is defined by the following two extra rules [6, Fig. 3]

$$\frac{s, \mathbf{P} \to^{\mathcal{H}} s', \checkmark}{s, \lfloor \mathbf{P} \rfloor \to s', \checkmark}\ \mathbf{sb1} \qquad \frac{s, \mathbf{P} \to^{\mathcal{H}} s', \mathbf{P}'}{s, \lfloor \mathbf{P} \rfloor \to s', \mathbf{P}'}\ \mathbf{sb2}$$

With these adjustements Abate, Busi, and Tsampas [6] show that the compiler that wraps assignments into sandboxed assignements and the embedding of source behaviors in the target are both a MoDL and a MMoDL, moreover the target interpretation of source noninterfrence is a subset of target noninterference,i.e., the translation ensures robust preservation of noninterference. □

Conjecture 1 (MMoDL and the Atlas from Figure 5.1)**.** We conjecture that MMoDL is strictly weaker than any relational criterion in Figure 5.1. The reason is that the back-translation functions that can be obtained by a natural transformation $t : \mathcal{C} \Longrightarrow \mathsf{C}^*$, – e.g., $bk = t^*_\emptyset \times id$ in Theorem 16 – may associate two different source contexts for $\mathsf{P}_1 \neq \mathsf{P}_2$, in contrast with the requirement of, for example, $k\mathsf{RHC}^\sim$.

5.6.4 Some Final Remarks

In this section and in Section 5.5 we discussed two possible approaches for strengthening the notion of preservation of observational equivalence, to have that fully abstract compilers are also robust. Both appraoches come with pros and cons:

- The approach chosen in Section 5.5 binds observables to the only "printable values" of two languages. We believe this makes it tricky to specifying conditions of initial or intermediate phases of the execution, such as noninterference. For a concrete example, Busi et al. [31] prove preservation of noninterference by showing a fully abstract translation, but the secret to be protected is in the code of the programs to be compiled.

  On the other hand, the notion of observational equivalence considered is the canonical one, and can therefore be adopted to establish whether fully abstract translations from the literature are also robust.

- The use of MOS allows to consider hyperproperties defined starting from an arbitrary functor, that may include states, I/O and sequences of them. On the other hand:

  – MoDL preserve bisimiliarity, hence not the canonical observational equivalence

  – the correspondence between distributive laws and formal rules allows to express the semantics of languages with a reasonable effort, lifting such a correspondence so that MoDL and MMoDL can be expressed as a relation between source and target rules is the only hope to make MoDL and MMoDL scale to more interesting languages, but at the best of our knowledge this approach is still under investigation.

Finally we notice some similarities between the hypothesis required on ϑ in Theorem 14 and Definition 24. In both cases, a relation between target terms and back-translated one is expressed, and such a relation must be shown to be stable under evaluation of terms, "step by step" in the MOS framework, more loosely in Section 5.5.

5.7 On Existing Proof Techniques

Although developing proof techniques for secure compilation is not the goal of this work, for sake of completeness we give an overview of the existing literature regarding such a topic.

Proving a compiler is fully abstract consists in showing preservation and reflection of contextual equivalence. Reflection is often subsumed by correctness [93], while preservation is more commonly proved by showing that a target context $\mathbf{C_T}$, able to distinguish $\mathsf{P}_1{\downarrow}$ and $\mathsf{P}_2{\downarrow}$, can be *back-translated* to a source context $\mathsf{C_S}$ that can distinguish P_1 and P_2. Back-translation of target contexts has been achieved in the literature by means of:

- *logical relations.* These kind of relations intuitively lift a basic or ground relation according to the operations of a mathematical structure or the constructs of a language,e.g., two functions are related if they return related results on related arguments or inputs. Logical relations are a flexible and powerful tool to prove properties of a language (see [109] for a gentle introduction) and can be extended to relate terms of different languages in order to prove well-behavedness of translations or encodings [93, 91]. However, when source and target languages have sensibly different constructors, it can be necessary to interpret the two languages in a common algebraic structure, and there define the logical relation itself [64].

- *embedding* the type of back-translated target contexts in a *universal type* [85]. This allows to formally back-translate target contexts that in principle don't have an exact source counterpart, but that on source values behave like a source context. This intuition is expressed as an adjunction composed of a projection map from the universal type to a source one and an embedding into the universal type.

- information collected on execution *traces*,e.g., [42]. A similar technique usually adopts traces exposing relevant information on the interaction between program and context, and such that trace equivalence properly implement contextual or observational equivalence. An important advantage of trace-based techniques is that the violation of contextual equivalence is witnessed by a pair of finite prefixes (as it is a $2relHypersafety$), so that it's sound to back-translate a target context $\mathbf{C_T}$ to a source one $\mathsf{C_S}$ that only *approximates* $\mathbf{C_T}$ for a finite amount of steps [42].

Proof techniques for robust compilation. Similarly to fully abstract compilation, a proof of robust compilation requires the back-translation of a traget context $\mathbf{C_T}$ into a source one, $\mathsf{C_S}$. In contrast to fully abstract compilation, the requirement on the back-translated context is not to have the same "distinguishing power" as $\mathbf{C_T}$, but depends on the family of trace properties or hyperproperties to be robustly preserved. Let us recall hereafter the property-free characterization of few criteria from Figure 5.1 to exaplain what the back-translation should guarantee. For

simplicity we assume $\sim$ to be the identity (therefore τ is the identity too).

$$\mathsf{RTC}^{\sim} \equiv \forall \mathsf{P}\ \forall \mathbf{C_T}\ \forall t.\ \mathbf{C_T}[\mathsf{P}\downarrow] \rightsquigarrow t \Rightarrow \exists \mathsf{C_S}.\ \mathsf{C_S}[\mathsf{P}] \rightsquigarrow t$$

$$\mathsf{RHC}^{\sim} \equiv \forall \mathsf{P}\ \forall \mathbf{C_T}.\ \exists \mathsf{C_S}.\ \mathtt{beh}(\mathbf{C_T}[\mathsf{P}\downarrow]) = \mathtt{beh}(\mathsf{C_S}[\mathsf{P}])$$

$$k\mathsf{RHC}^{\sim} \equiv \forall \mathbf{C_T}.\ \exists \mathsf{C_S}. \forall \mathsf{P}.\ \mathtt{beh}(\mathbf{C_T}[\mathsf{P}\downarrow]) = \mathtt{beh}(\mathsf{C_S}[\mathsf{P}])$$

For the preservation of trace properties – $\mathsf{RTC}^{\sim}$ – the back-translated context $\mathsf{C_S}$ depends on P, $\mathbf{C_T}$ and on the trace t that it is required to be exposed in the source. As a consequence, for the same program P and the same context $\mathbf{C_T}$, distinct execution traces $t_1 \neq t_2$ may be simulated by distinct source contexts, while for the robust preservation of hyperproperties – $\mathsf{RHC}^{\sim}$– the back-translated source context must be the same for any trace. Notice that the back-translation needed for $\mathsf{RHC}^{\sim}$can sitll use information on the source program P while the one for $k\mathsf{RHC}^{\sim}$ can only use information on the target context $\mathbf{C_T}$ to produce a source context that "simulates" $\mathbf{C_T}$ on every source program.

To sum up: "the stronger the theorem the harder the proof", as the back-translation is required to produce more precise source contexts and with less information. For this reason, the easiest back-translation to achieve is the one for proving robust preservation of *Safety*,i.e.,

$$\mathsf{RSC}^{\sim} \equiv \forall \mathsf{P}\ \forall \mathbf{C_T}\ \forall m.\ \mathbf{C_T}[\mathsf{P}\downarrow] \rightsquigarrow^* m \Rightarrow \exists \mathsf{C_S}.\ \mathsf{C_S}[\mathsf{P}] \rightsquigarrow^* m$$

where the context $\mathsf{C_S}$ is required to simulate the target one only for a finite number of steps (enough to expose the prefix m) and for one single execution. If traces – and hence prefixes – record enough information about the control flow of program executions e.g., events are of the form "the context called function f of the program", then a suitable $\mathsf{C_S}$ can be easily built so to expose a finite list of events when linked with P and executed w.r.t. the source semantics. When traces do not record enough information, it's possbile to consider an instrumented trace semantics that is informative enough, and then "erase" the extra information via a Galois insertion [11, 48]. More interestringly, the idea illustrated above can be generalized to finite sets of finite prefixes, thus providing an efficient proof technique for the robust preservation of *k-Hypersafety* such as noninterference and *krelSafety* (for k finite)[8, Section 6.4].

The Categorical approach of MoDL proposed by Tsampas et al. [117] for achieving fully abstract compilation, mainly relies on the fact that bisimiliarity is a congruence, thus allowing to drop the universal quantification on contexts. On the other hand, for robust satisfaction one cannot avoid the universal quantification on target contexts as many relations are simply not congruences, therefore MoDL must be strenghtened with MMoDL. While the examples in [117, 6] involve relatively simple languages so that proofs of naturality of the morphisms involved can be given, it seems hopeless to prove MoDL and MMoDL from their definitions for more interesting languages. As discussed in Section 5.6.4, a better strategy would consist in lifting the correspondence between distributive laws and GSOS rules to a correspondence between maps of distributive laws (MoDL) and relations between GSOS rules of the source and the target.

5.8 Related Work

Abadi [1] first propose the use of full abstraction for estimating security in translations by comparing attackers observations on source and compiled programs. Fully abstract compilation has been successfully adopted for showing the (robust) preservation of a particular hyperproperty e.g., noninterference [27], isolation [31], or few of them at once e.g., well-bracketed control flow and local state isolation [110], that can be captured by the same notion of observational equivalence.

Abate et al. [8] showed that fully abstract compilers don't robustly preserve *arbitrary* or large classes of trace properties e.g., *Safety*. They key idea for building a counterexample is to exclude some safety property from the notion of equivalence preserved by the compiler (see e.g., Example 15). Abate, Busi, and Tsampas [6, Section 3] show that indeed the guarantee – given by a fully abstract compiler – on a arbitrary hyperproperty can be trivial. They provide a condition to make fully abstract compilers also robust, that we discussed extensively in Section 5.6.

Patrignani and Garg [95] proposed *robustly safe compilation* i.e., robust preservation of *Safety*, as criterion for secure compilation with efficient proof techniques. A variant of robustly safe compilation is adopted by Abate et al. [11] to define robust preservation of *Safety* for translations between unsafe languages,i.e., languages in which programs can encounter an undefined behavior.

The idea of Abate et al. [8] is that a "one-size fits all" definition of secure compilation is too restrictive, and different notions of security of source programs,e.g., data integrity and data confidentiality, may be preserved under different conditions. They therefore define a family of criteria, each one ensuring the robust preservation of a certain class of hyperproperties, and show that for many interesting classes, efficient trace-based proof techniques are possible (see [8, Section 6] and [116]), sometimes borrowing ideas from trace-based proof techniques for fully abstract compilation see e.g., [42]. The two subsequent works [10, 7] generalize the idea to allow source and target observables as well as trace model to differ.

Finally, Paykin et al. [98] show how to apply the theory of robust preservation to the study of *weird machines* – a computational model for exploits – and Busi [29, Chapter 5] discusses how to apply the theory of robust preservation to translation validation.

Chapter 6

Conclusions

This manuscript presented formal frameworks for correct and secure compilation. Our definitions of correct and secure compilation come with an explicit description of the class of security properties preserved through compilation, thus allowing to establish if, and in what sense, the translation of a program is correct and secure.

Program properties have been classified from a topological point of view, as to abstract away detail on the particular execution model adopted. Properties of source and target programs have been related within the framework of Galois connections. The core idea of our theory is captured by Lemma 8 and Lemma 9 that define preservation and robust preservation for program semantics defined over arbitrary posets, and hence also the topological classes of *Safety*, *Hypersafety* or classes of relational properties.

Chapter 4 exposed our framework for correct compilation, intended as preservation of classes of properties, hyperproperties, and relational hyperproperties, and that generalizes the more common definition of compiler correctness as refinement or inclusion of traces. We also showed that preserving safety properties – a proper sub-class of all trace properties – often suffices for the preservation of properties that are also non safety,e.g., liveness ones.

Chapter 5 exposed our framework for secure compilation criteria, intended as preservation of classes of properties and against arbitrary, hence adversarial, target code. We compared our criteria with the notion of fully abstract compilation, discussed why this may fail to preserve integrity or confidentiality of data, and we provided a sufficient condition for it to achieve a form of robustness.

6.1 Future Work

We discuss hereafter a few lines of possible future work, some already under investigation by collaborators of the author.

Robust verification in source high-level languages can be approached both with fully automatic techniques [69] and with interactive ones. The SSProve framework [9] provides sound rules for interactive proofs of robust satisfaction of relational properties for cryptographic games against distinguishers. This kind of attackers can interact with programs only by calling functions (implemented and exposed by programs) on some input and observing the result, in particular the memory accessible by the attacker and the one of the programs are disjoint. We believe SSProve can in principle support a more powerful attacker model, at the price of extending its program logic with rules from separation logic [102].

Unsafe languages are subject to low-level attacks that can sometimes be mitigated by compartmentalization techniques. It's reasonable to imagine compartmentalization schemes such as software fault isolation, or capabilities to be part of secure compilation chains. Both secure compilation criteria and techniques for implementing compartmentalization have been recently proposed,see e.g., [110, 11, 48] but formal proofs for these compilers are still incomplete due to their complexity, even for simple languages (see e.g., [11, 48]).

Efficient proof techniques are therefore necessary to formal verification. Currently, the most efficient proof techniques available are those for compilers that extend CompCert,e.g., [22, 114, 113], even though they are not robust in the sense of Chapter 5. The technique proposed by Abate et al. [11] assumes correctness and states it similarly to CompCert, therefore the complexity of the proof is due to the back-translation of target contexts, and can be approached as explained in Section 5.7.
Providing efficient proof techniques was not the goal of Abate, Busi, and Tsampas [6] (see also Section 5.6), nevertheless we believe – and the last author of [6] is currently working at it – both MoDL and MMoDL can be expressed as relations between formal rules describing the operational semantics of source and target languages. These kind of rules in turn, seem amenable to formalization in a proof assistant.

Translation validation can be intended both as a valid alternative to formal verification and as a first milestone towards a formally verified compiler. Translation validation can show correctness and security of the translation of a family of programs or for a family of target contexts, while providing hints on the incorrectness or insecurity of the translations of other programs. While translation validation for correctness exists since a long time (see [100]), its development for security properties is less mature: the work by Namjoshi and Tabajara [83] does not allow to validate robust preservation as in Chapter 5, while the proposal by Busi, Degano, and Galletta [30] requires some extra assumptions for its soundness (see [29]).
The main challenge in the development of translation validation techniques for the robust criteria from Figure 5.1 is embodied by the symbols "$\exists\mathsf{C_S}$" common to all the property-free characterizations. Solving the obligation given by such an existential quantifier needs an approach that is sensibly different to the one originally developed by Pnueli, Siegel, and Singerman [100]. We believe a possible technique may rely on the synthesis of a source context from a hyperproperty [26] that approximates the behavior of a compiled program in a certain target context,i.e., $\mathtt{beh}(\mathbf{C_T}[\mathsf{P}{\downarrow}])$.

Probabilistic languages come with primitives for random sampling elements of a set according to certain distributions and are of particular interest for machine learning, statistical applications and for the development of cryptography. At the best of our knowledge, verified compilers for languages with probabilistic primitives come with proofs of functional correctness e.g., [12, 13], still it's not clear weather probability distributions of programs are preserved.

In Appendix C we sketch a possible approach to study translations between probabilistic languages. We build on the work by Cousot and Monerau [38] who show how to extend abstract interpretation to probabilistic languages, and the notion of Galois connections to probabilistic spaces. This in turn allows to define the probability that a program satisfies a hyperproperty (called "probability of a program property" by Cousot and Monerau [38]), and it's easy to show that a compiler that preserves

hyperproperties according to one of the equivalent definitions from Theorem 6, also preserves the probability of their satisfaction.
We believe such a straightforward extension of the framework presented in this thesis may be further extended to establish whether compilation – e.g., from the monadic language from [9] – preserved the *advantage* of an adversary in a cryptographic game. What discussed in Appendix C is however not yet mature as:

(i) it's still not clear how to formulate some statements common to cryptographic applications expressing that "ciphertexts are indistinguishable from random",

(ii) it's not possible to have a "probabilistic" statement for secure compilation such as the one by Bartoletti and Zunino [23]. Programs written in BitML are translated to the language for bitcoin smart contracts, with a overwhelming probability that a target attack can be simulated in the source.

Languages for smart contracts such as BitML [23] or Move comes with relatively simple constructs, and it seems there is a increasing interest and effort in both robust verification [94, 17, 16] and secure compilation for them [23].

Extraction of programs from high-level modeling languages or process calculi represents an interesting field for the application of the theory presented in this thesis. The parallel composition of contexts can model attackers with nondeterministic behaviors, and we discussed in Section 5.4.1 how similar features of the source language allows to achieve strong notions of robust preservation from – the easier to achieve – robust preservation of *Safety*.

Expressiveness of languages is compared by means of *reductions* between the two languages. The notion of reduction originally proposed for comparing expressiveness of Turing machines, can be extended to other languages as proposed by Mitchell [80]. Roughly speaking, a fully abstract translation – satisfying some homomorphism conditions – between two languages ensures that source programs can compute the same functions (on source values) as the target ones (on target values) [80, Proposition 3.4]. Patrignani, Martin, and Devriese [96] provide a fully abstract translation between lambda calculi with iso and equi recursive types, thus showing their equi-expressiveness.
On the other hand, Gorla and Nestmann [61], discuss examples of fully abstract translations between languages clearly not equi-expressive. The most surprising of these examples is the existence of a fully abstract translation between Turing machines and deterministic finite automata. In [6, Appendix C] it's explained how trace-based criteria can be used to rule out such a bad *false positive* result of equi-expressiveness. Supported by a correspondence between open sets of certain topologies and computability notions [112], we believe the hierarchies from Figure 4.3 and Figure 5.1 may find application in the comparison of expressiveness between languages.

Appendix A

Atlas of Secure Compilation in Detail

In this appendix we show how to recover one by one all the criteria collected in Figure 5.1.

A.1 Robust Preservation of Classes of Trace Properties

Theorem 17 (Robust Preservation of Trace Properties (detailed))**.**
Let $\tau : \wp(\mathsf{Trace_S}) \leftrightarrows \wp(\mathbf{Trace_T}) : \sigma$ *a Galois connection between trace properties. The followings are equivalent:*

$$\mathsf{RTP}^{\tau} \equiv \forall \mathsf{P}\ \forall \pi_S \in \wp(\mathsf{Trace_S}).\ \forall \mathsf{C}.\ \mathsf{C_S}[\mathsf{P}] \models \pi_S \Rightarrow \forall \mathbf{C_T}.\ \mathbf{C_T}[\mathsf{P}\downarrow] \models \tau(\pi_S)$$

$$\mathsf{RTP}^{\sigma} \equiv \forall \mathsf{P}\ \forall \pi_T \in \wp(\mathbf{Trace_T}).\ \forall \mathsf{C}.\ \mathsf{C_S}[\mathsf{P}] \models \sigma(\pi_T) \Rightarrow \forall \mathbf{C_T}.\ \mathbf{C_T}[\mathsf{P}\downarrow] \models \pi_T$$

$$\mathsf{RF}^{\sigma}_{\tau} \equiv \forall \mathsf{P}\ \forall \mathbf{C_T}.\ \mathit{beh}(\mathbf{C_T}[\mathsf{P}\downarrow]) \subseteq \tau(\bigcup_{\mathsf{C_S}} \mathit{beh}(\mathsf{C_S}[\mathsf{P}]))$$

$$\mathsf{RTC}^{\sim} \equiv \forall \mathsf{P}\ \forall \mathbf{C_T}\ \forall t.\ \mathbf{C_T}[\mathsf{P}\downarrow] \rightsquigarrow t \Rightarrow \exists \mathsf{C_S}.\ \exists\, \mathsf{s} \sim t.\ \mathsf{C_S}[\mathsf{P}] \rightsquigarrow \mathsf{s}$$

where $\sim \subseteq \mathit{Trace_S} \times \mathit{Trace_T}$ *is the relation corresponding to* $\tau \leftrightarrows \sigma$ *(see Definition 13).*

Proof. For the equivalence of the first three criteria apply Lemma 9 with:

- $(C, \sqsubseteq) = (\wp(\mathsf{Trace_S}), \subseteq), (A, \leq) = (\wp(\mathbf{Trace_T}), \subseteq)$
- the Galois connection $\tau : \wp(\mathsf{Trace_S}) \leftrightarrows \wp(\mathbf{Trace_T}) : \sigma$
- $\mathscr{P} = \mathcal{P}rg$, $\mathscr{G} = \mathcal{C}tx$, $[\![\cdot]\!] : \mathcal{C}tx \times \mathcal{P}rg \to \wp(\mathsf{Trace_S}) = \lambda(\mathsf{C_S}, \mathsf{P}).\ \mathtt{beh}(\mathsf{C_S}[\mathsf{P}])$
- $\mathscr{P}' = \mathcal{P}rg$, $\mathscr{G}' = \mathcal{C}tx$, $(\!|\cdot|\!) : \mathcal{C}tx \times \mathcal{P}rg \to \wp(\mathbf{Trace_T}) = \lambda(\mathbf{C_T}, \mathbf{Q}).\ \mathtt{beh}(\mathbf{C_T}[\mathbf{Q}])$
- $\downarrow : \mathcal{P}rg \to \mathcal{P}rg = \cdot\downarrow$

Finally we show the equivalence $\mathsf{RF}^{\sigma}_{\tau} \iff \mathsf{RTC}^{\sim}$. Recall that τ preserves joins (see Remark 4),i.e.,

$$\tau(\bigcup_{\mathsf{C_S}} \mathtt{beh}(\mathsf{C_S}[\mathsf{P}])) = \bigcup_{\mathsf{C_S}} \tau(\mathtt{beh}(\mathsf{C_S}[\mathsf{P}])),$$

from which the thesis follows by unfolding the definition of τ. □

Theorem 18 (Robust Preservation of *Safety* Properties (detailed))**.**
Let $\tau : \wp(\mathsf{Trace_S}) \leftrightarrows \wp(\mathbf{Trace_T}) : \sigma$ *be a Galois connection between trace properties. The followings are equivalent:*

$$\mathsf{RTP}^{Safe\circ\tau} \equiv \forall \mathsf{P}\ \forall \pi_S \in \wp(\mathsf{Trace_S}).\ \forall \mathsf{C}.\ \mathsf{C_S}[\mathsf{P}] \models \pi_S \Rightarrow \forall \mathbf{C_T}.\ \mathbf{C_T}[\mathsf{P}\downarrow] \models Safe \circ \tau(\pi_S)$$
$$\mathsf{RSP}^{\sigma} \equiv \forall \mathsf{P}\ \forall \pi_T \in Safety_T.\ \forall \mathsf{C}.\ \mathsf{C_S}[\mathsf{P}] \models \sigma(\pi_T) \Rightarrow \forall \mathbf{C_T}.\ \mathbf{C_T}[\mathsf{P}\downarrow] \models \pi_T$$
$$\mathsf{RF}^{\sigma}_{Safe\circ\tau} \equiv \forall \mathsf{P}\ \forall \mathbf{C_T}.\ \mathit{beh}(\mathbf{C_T}[\mathsf{P}\downarrow]) \subseteq Safe \circ \tau(\bigcup_{\mathsf{C_S}} \mathit{beh}(\mathsf{C_S}[\mathsf{P}]))$$
$$\mathsf{RSC}^{\sim} \equiv \forall \mathsf{P}\ \forall \mathbf{C_T}\ \forall A_T \in \mathcal{O}_T.\ \mathit{beh}(\mathbf{C_T}[\mathsf{P}\downarrow]) \cap A_T \neq \emptyset \Rightarrow \exists \mathsf{C_S}.\ \tau(\mathit{beh}(\mathsf{C_S}[\mathsf{P}])) \cap A_T \neq \emptyset$$

where for $\pi_T \in \wp(\mathbf{Trace_T})$, $Safe(\pi_T)$ *is the best approximation of* π_T *in Safety,i.e., the smallest safety property that includes* π_T *(see Example 9).*

Proof. Apply Lemma 9 with:

- $(C, \sqsubseteq) = (\wp(\mathsf{Trace_S}), \subseteq), (A, \leq) = (Safety_T, \subseteq)$
- the Galois connection $Safe \circ \tau : \wp(\mathsf{Trace_S}) \leftrightarrows Safety_T : \sigma$
- $\mathscr{P} = \mathcal{P}rg$, $\mathscr{G} = \mathcal{C}tx$, $[\![\cdot]\!] : \mathcal{C}tx \times \mathcal{P}rg \to \wp(\mathsf{Trace_S}) = \lambda(\mathsf{C_S}, \mathsf{P}).\ \mathtt{beh}(\mathsf{C_S}[\mathsf{P}])$
- $\mathscr{P}' = \mathcal{P}rg$, $\mathscr{G}' = \mathcal{C}tx$, $(\!|\cdot|\!) : \mathcal{C}tx \times \mathcal{P}rg \to Safety_T = \lambda(\mathbf{C_T}, \mathbf{Q}).\ Safe(\mathtt{beh}(\mathbf{C_T}[\mathbf{Q}]))$
- $\downarrow : \mathcal{P}rg \to \mathcal{P}rg = \cdot\downarrow$

and obtain the followings are equivalent:

$$\begin{aligned}
\mathsf{RCP}^{\alpha} &\equiv \forall \mathsf{P}\ \forall \pi_S \in \wp(\mathsf{Trace_S}).\\
&\quad (\forall \mathsf{C_S}.\ \mathtt{beh}(\mathsf{C_S}[\mathsf{P}]) \subseteq \pi_S) \Rightarrow (\forall \mathbf{C_T}.\ Safe(\mathtt{beh}(\mathbf{C_T}[\mathsf{P}\downarrow])) \subseteq Safe \circ \tau(\pi_S))\\
\mathsf{RAP}^{\alpha} &\equiv \forall \mathsf{P}\ \forall \pi_T \in Safety_T.\\
&\quad (\forall \mathsf{C_S}.\ Safe \circ \tau(\mathtt{beh}(\mathsf{C_S}[\mathsf{P}])) \subseteq \pi_T \Rightarrow (\forall \mathbf{C_T}.\ Safe \circ \mathtt{beh}(\mathbf{C_T}[\mathsf{P}\downarrow]) \subseteq \pi_T\\
\mathsf{RAP}^{\gamma} &\equiv \forall \mathsf{P}\ \forall \pi_T \in Safety_T.\\
&\quad (\forall \mathsf{C_S}.\ \mathtt{beh}(\mathsf{C_S}[\mathsf{P}]) \subseteq \sigma(\pi_T) \Rightarrow (\forall \mathbf{C_T}.\ Safe \circ \mathtt{beh}(\mathbf{C_T}[\mathsf{P}\downarrow]) \subseteq \pi_T\\
\mathsf{RF}^{\gamma}_{\alpha} &\equiv \forall \mathsf{P}\ \forall \mathbf{C_T}.\ Safe \circ \mathtt{beh}(\mathbf{C_T}[\mathsf{P}\downarrow]) \subseteq Safe \circ \tau(\bigcup_{\mathsf{C_S}} \mathtt{beh}(\mathsf{C_S}[\mathsf{P}]))
\end{aligned}$$

Notice that by Remark 7,

$$(\forall \mathbf{C_T}.\ Safe(\mathtt{beh}(\mathbf{C_T}[\mathsf{P}\downarrow])) \subseteq Safe \circ \tau(\pi_S)) \iff (\forall \mathbf{C_T}.\ (\mathtt{beh}(\mathbf{C_T}[\mathsf{P}\downarrow])) \subseteq Safe \circ \tau(\pi_S)),$$

so that $\mathsf{RCP}^{\alpha} \iff \mathsf{RTP}^{Safe\circ\tau}$. Similarly, $\mathsf{RSP}^{\sigma} \iff \mathsf{RAP}^{\gamma}$ and $\mathsf{RF}^{\sigma}_{Safe\circ\tau} \iff \mathsf{RF}^{\gamma}_{\alpha}$, and $\mathsf{RAP}^{\alpha} \iff \mathsf{RSC}^{\sim}$ as shown in Lemma 16. □

A.2 Robust Preservation of Classes of Hyperproperties

Theorem 19 (Robust Preservation of *Hypersafety* (detailed))**.**
Let $\tau : \wp(\mathsf{Trace_S}) \leftrightarrows \wp(\mathbf{Trace_T}) : \sigma$ *be a Galois connection between trace properties. Let*

$$Cl_{\subseteq} \circ \tau : Closed_{\subseteq}(\wp(\wp(\mathsf{Trace_S}))) \leftrightarrows Closed_{\subseteq}(\wp(\wp(\mathbf{Trace_T}))) : Cl_{\subseteq} \circ \sigma$$

be the lifting to subset-closed hyperproperties as in Lemma 10. The followings are equivalent:

$$\begin{aligned}
\mathsf{RH}_{\subseteq}\mathsf{P}^{Hsafe \circ \tau} &\equiv \forall \mathsf{P}\ \forall H_S \in Closed_{\subseteq}.\ \forall \mathsf{C}.\ \mathsf{C_S}[\mathsf{P}] \models H_S \Rightarrow \forall \mathbf{C_T}.\ \mathbf{C_T}[\mathsf{P}\!\downarrow] \models Hsafe \circ \tau(H_S) \\
\mathsf{RHSP}^{\sigma} &\equiv \forall \mathsf{P}\ \forall H_T \in Hypersafety.\ \forall \mathsf{C}.\ \mathsf{C_S}[\mathsf{P}] \models \sigma(H_T) \Rightarrow \forall \mathbf{C_T}.\ \mathbf{C_T}[\mathsf{P}\!\downarrow] \models H_T \\
\mathsf{RF}^{\sigma}_{Hsafe \circ \tau} &\equiv \forall \mathsf{P}\ \forall \mathbf{C_T}.\ \mathtt{beh}(\mathbf{C_T}[\mathsf{P}\!\downarrow]) \in Hsafe(\{\tau(\mathtt{beh}(\mathsf{C_S}[\mathsf{P}])) \mid \mathsf{C_S} \in \mathcal{C}tx\}) \\
\mathsf{RHSC}^{\sim} &\equiv \forall \mathsf{P}\ \forall \mathbf{C_T}\ \forall A_T \in \mathcal{V}_{low}(\mathcal{O}_T). \\
&\qquad Cl_{\subseteq}(\{\mathtt{beh}(\mathbf{C_T}[\mathsf{P}\!\downarrow])\}) \cap A_T \neq \emptyset \Rightarrow \\
&\qquad \exists \mathsf{C_S}.\ Cl_{\subseteq} \circ \tau(\{\mathtt{beh}(\mathsf{C_S}[\mathsf{P}])\}) \cap A_T \neq \emptyset
\end{aligned}$$

where for $H_T \in \wp(\wp(\mathbf{Trace_T}))$, $Hsafe(H_T)$ *is the best approximation of* H_T *in Hypersafety,i.e., the smallest hypersafety that includes* H_T *(see Example 9).*

Proof. Apply Lemma 9 with:

- $(C, \sqsubseteq) = (Closed_{\subseteq}(\wp(\wp(\mathsf{Trace_S}))), \subseteq), (A, \leq) = (Hypersafety_T, \subseteq)$
- the Galois connection $Hsafe \circ \tau : Closed_{\subseteq}(\wp(\wp(\mathsf{Trace_S}))) \leftrightarrows Hypersafety_T : \sigma$
- $\mathscr{P} = \mathcal{P}rg$, $\mathscr{G} = \mathcal{C}tx$, $[\![\cdot]\!] : \mathcal{C}tx \times \mathcal{P}rg \to Closed_{\subseteq}(\wp(\wp(\mathsf{Trace_S}))) = \lambda(\mathsf{C_S}, \mathsf{P}).\ Cl_{\subseteq}(\{\mathtt{beh}(\mathsf{C_S}[\mathsf{P}])\})$.
- $\mathscr{P}' = \mathcal{P}rg$, $\mathscr{G}' = \mathcal{C}tx$, $(\!|\cdot|\!) : \mathcal{C}tx \times \mathcal{P}rg \to Hypersafety_T = \lambda(\mathbf{C_T}, \mathbf{Q}).\ Hsafe(\{\mathtt{beh}(\mathbf{C_T}[\mathbf{Q}])\})$
- $\downarrow : \mathcal{P}rg \to \mathcal{P}rg = \cdot\!\downarrow$

and obtain the followings are equivalent:

$$\begin{aligned}
\mathsf{RCP}^{\alpha} &\equiv \forall \mathsf{P}\ \forall H_S \in Closed_{\subseteq}(\wp(\wp(\mathsf{Trace_S}))). \\
&\qquad (\forall \mathsf{C_S}.\ Cl_{\subseteq}(\{\mathtt{beh}(\mathsf{C_S}[\mathsf{P}])\} \subseteq H_S) \Rightarrow (\forall \mathbf{C_T}.\ Hsafe(\{\mathtt{beh}(\mathbf{C_T}[\mathsf{P}\!\downarrow])\}) \subseteq Hsafe \circ \tau(H_S)) \\
\mathsf{RAP}^{\alpha} &\equiv \forall \mathsf{P}\ \forall H_T \in Hypersafety_T. \\
&\qquad (\forall \mathsf{C_S}.\ Hsafe \circ \tau(\{\mathtt{beh}(\mathsf{C_S}[\mathsf{P}])\}) \subseteq H_T \Rightarrow (\forall \mathbf{C_T}.\ Hsafe(\{\mathtt{beh}(\mathbf{C_T}[\mathsf{P}\!\downarrow])\}) \subseteq H_T \\
\mathsf{RAP}^{\gamma} &\equiv \forall \mathsf{P}\ \forall H_T \in Hypersafety_T. \\
&\qquad (\forall \mathsf{C_S}.\ Cl_{\subseteq}(\{\mathtt{beh}(\mathsf{C_S}[\mathsf{P}])\}) \subseteq \sigma(H_T) \Rightarrow (\forall \mathbf{C_T}.\ Hsafe(\{\mathtt{beh}(\mathbf{C_T}[\mathsf{P}\!\downarrow])\}) \subseteq H_T \\
\mathsf{RF}^{\gamma}_{\alpha} &\equiv \forall \mathsf{P}\ \forall \mathbf{C_T}.\ Hsafe(\{\mathtt{beh}(\mathbf{C_T}[\mathsf{P}\!\downarrow])\}) \subseteq Hsafe \circ \tau(\bigcup_{\mathsf{C_S}} Cl_{\subseteq}(\{\mathtt{beh}(\mathsf{C_S}[\mathsf{P}])\}))
\end{aligned}$$

Notice that by Remark 7,

$$\begin{aligned}
(\forall \mathbf{C_T}.\ Hsafe(\{\mathtt{beh}(\mathbf{C_T}[\mathsf{P}\!\downarrow])\}) \subseteq Hsafe \circ \tau(H_S)) &\iff \\
(\forall \mathbf{C_T}.\ Cl_{\subseteq}(\{\mathtt{beh}(\mathbf{C_T}[\mathsf{P}\!\downarrow])\}) \subseteq Hsafe \circ \tau(H_S)) &\iff \\
(\forall \mathbf{C_T}.\ \mathtt{beh}(\mathbf{C_T}[\mathsf{P}\!\downarrow]) \in Hsafe \circ \tau(H_S))
\end{aligned}$$

so that $\mathsf{RCP}^{\alpha} \iff \mathsf{RH}_{\subseteq}\mathsf{P}^{Hsafe \circ \tau}$ and similarly, $\mathsf{RHSP}^{\sigma} \iff \mathsf{RAP}^{\gamma}$.
For $\mathsf{RF}^{\sigma}_{Hsafe \circ \tau} \iff \mathsf{RF}^{\gamma}_{\alpha}$ simply notice that by $Hypersafety \subseteq Closed_{\subseteq}$ the application of $Cl_{\subseteq}$ can be omitted in $\mathsf{RF}^{\gamma}_{\alpha}$ above.

Finally for the equivalence recall that violation of a hypersafety, corrsponds with intersection of it's complement, that is an open set $A_T \in \mathcal{V}_{low}(\mathcal{O}_T)$, so that $\mathsf{RHSC}^{\sim}$ is equivalent to the contraposition of RAP^{α} . □

Theorem 20 (Robust Preservation of $Closed_{\subseteq}$ (detailed))**.**
Let $\tau : \wp(\mathsf{Trace_S}) \leftrightarrows \wp(\mathbf{Trace_T}) : \sigma$ *be a Galois connection between trace properties. Let*

$$Cl_{\subseteq} \circ \tau : Closed_{\subseteq}(\wp(\wp(\mathsf{Trace_S}))) \leftrightarrows Closed_{\subseteq}(\wp(\wp(\mathbf{Trace_T}))) : Cl_{\subseteq} \circ \sigma$$

be the lifting to subset-closed hyperproperties as in Lemma 10. The followings are equivalent:

$$\begin{aligned}
\mathsf{RH}_{\subseteq}\mathsf{P}^{\tau} &\equiv \forall \mathsf{P}\ \forall H_S \in Closed_{\subseteq}.\ \forall \mathsf{C}.\ \mathsf{C_S}[\mathsf{W}] \models H_S \Rightarrow \forall \mathbf{C_T}.\ \mathbf{C_T}[\mathsf{P}\downarrow] \models Cl_{\subseteq} \circ \tau(H_S) \\
\mathsf{RH}_{\subseteq}\mathsf{P}^{\sigma} &\equiv \forall \mathsf{P}\ \forall H_T \in Closed_{\subseteq}.\ \forall \mathsf{C}.\ \mathsf{C_S}[\mathsf{P}] \models Cl_{\subseteq} \circ \sigma(H_T) \Rightarrow \forall \mathbf{C_T}.\ \mathbf{C_T}[\mathsf{P}\downarrow] \models H_T \\
\mathsf{RF}^{Cl_{\subseteq}\circ\sigma}_{Cl_{\subseteq}\circ\tau} &\equiv \forall \mathsf{P}\ \forall \mathbf{C_T}.\ \mathit{beh}(\mathbf{C_T}[\mathsf{P}\downarrow]) \in\ Cl_{\subseteq}(\{\tau(\mathit{beh}(\mathsf{C_S}[\mathsf{P}]))\ \mid\ \mathsf{C_S} \in \mathcal{C}tx\}) \\
\mathsf{RH}_{\subseteq}\mathsf{C}^{\sim} &\equiv \forall \mathsf{P}\ \forall \mathbf{C_T}\ \exists \mathsf{C_S}.\ \mathit{beh}(\mathbf{C_T}[\mathsf{P}\downarrow]) \subseteq \tau(\mathit{beh}(\mathsf{C_S}[\mathsf{P}]))
\end{aligned}$$

Proof. Apply Lemma 9 with:

- $(C, \sqsubseteq) = (Closed_{\subseteq}(\wp(\wp(\mathsf{Trace_S}))), \subseteq), (A, \leq) = (Closed_{\subseteq}(\wp(\wp(\mathbf{Trace_T}))), \subseteq)$
- the Galois connection $Cl_{\subseteq} \circ \tau : Closed_{\subseteq}(\wp(\wp(\mathsf{Trace_S}))) \leftrightarrows Closed_{\subseteq}(\wp(\wp(\mathbf{Trace_T}))) : Cl_{\subseteq} \circ \sigma$
- $\mathscr{P} = \mathcal{P}rg$, $\mathscr{G} = \mathcal{C}tx$, $[\![\cdot]\!] : \mathcal{C}tx \times \mathcal{P}rg \to Closed_{\subseteq}(\wp(\wp(\mathsf{Trace_S}))) = \lambda(\mathsf{C_S}, \mathsf{P}).\ Cl_{\subseteq}(\{\mathtt{beh}(\mathsf{C_S}[\mathsf{P}])\})$.
- $\mathscr{P}' = \mathcal{P}rg$, $\mathscr{G}' = \mathcal{C}tx$, $(\!|\cdot|\!) : \mathcal{C}tx \times \mathcal{P}rg \to Closed_{\subseteq}(\wp(\wp(\mathbf{Trace_T}))) = \lambda(\mathbf{C_T}, \mathbf{Q}).\ Cl_{\subseteq}(\{\mathtt{beh}(\mathbf{C_T}[\mathbf{Q}])$
- $\downarrow : \mathcal{P}rg \to \mathcal{P}rg = \cdot\downarrow$

and obtain the followings are equivalent:

$$\begin{aligned}
\mathsf{RCP}^{\alpha} &\equiv \forall \mathsf{P}\ \forall H_S \in Closed_{\subseteq}(\wp(\wp(\mathsf{Trace_S}))). \\
&\quad (\forall \mathsf{C_S}.\ Cl_{\subseteq}(\{\mathtt{beh}(\mathsf{C_S}[\mathsf{P}])\} \subseteq H_S) \Rightarrow (\forall \mathbf{C_T}.\ Cl_{\subseteq}(\{\mathtt{beh}(\mathbf{C_T}[\mathsf{P}\downarrow])\}) \subseteq Cl_{\subseteq} \circ \tau(H_S)) \\
\mathsf{RAP}^{\alpha} &\equiv \forall \mathsf{P}\ \forall H_T \in Closed_{\subseteq}\wp(\wp(\mathbf{Trace_T})). \\
&\quad (\forall \mathsf{C_S}.\ Cl_{\subseteq} \circ \tau(\{\mathtt{beh}(\mathsf{C_S}[\mathsf{P}])\}) \subseteq H_T \Rightarrow (\forall \mathbf{C_T}.\ Cl_{\subseteq}(\{\mathtt{beh}(\mathbf{C_T}[\mathsf{P}\downarrow])\}) \subseteq H_T \\
\mathsf{RAP}^{\gamma} &\equiv \forall \mathsf{P}\ \forall H_T \in Closed_{\subseteq}\wp(\wp(\mathbf{Trace_T})). \\
&\quad (\forall \mathsf{C_S}.\ Cl_{\subseteq}(\{\mathtt{beh}(\mathsf{C_S}[\mathsf{P}])\}) \subseteq \sigma(H_T) \Rightarrow (\forall \mathbf{C_T}.\ Cl_{\subseteq}(\{\mathtt{beh}(\mathbf{C_T}[\mathsf{P}\downarrow])\}) \subseteq H_T \\
\mathsf{RF}^{\gamma}_{\alpha} &\equiv \forall \mathsf{P}\ \forall \mathbf{C_T}.\ Cl_{\subseteq}(\{\mathtt{beh}(\mathbf{C_T}[\mathsf{P}\downarrow])\}) \subseteq Cl_{\subseteq} \circ \tau(\bigcup_{\mathsf{C_S}} Cl_{\subseteq}(\{\mathtt{beh}(\mathsf{C_S}[\mathsf{P}])\}))
\end{aligned}$$

Notice that by Remark 7,

$$\begin{aligned}
(\forall \mathbf{C_T}.\ Cl_{\subseteq}(\{\mathtt{beh}(\mathbf{C_T}[\mathsf{P}\downarrow])\}) \subseteq Cl_{\subseteq} \circ \tau(H_S)) &\iff \\
(\forall \mathbf{C_T}.\ \{\mathtt{beh}(\mathbf{C_T}[\mathsf{P}\downarrow])\} \subseteq Cl_{\subseteq} \circ \tau(H_S)) &\iff \\
(\forall \mathbf{C_T}.\ \mathtt{beh}(\mathbf{C_T}[\mathsf{P}\downarrow]) \in Cl_{\subseteq} \circ \tau(H_S))
\end{aligned}$$

so that $\mathsf{RCP}^{\alpha} \iff \mathsf{RH}_{\subseteq}\mathsf{P}^{\tau}$, $\mathsf{RH}_{\subseteq}\mathsf{P}^{\sigma} \iff \mathsf{RAP}^{\gamma}$, $\mathsf{RF}^{\sigma}_{Cl_{\subseteq}\circ\tau} \iff \mathsf{RF}^{\gamma}_{\alpha}$ and $\mathsf{RH}_{\subseteq}\mathsf{C}^{\sim} \iff \mathsf{RF}^{\gamma}_{\alpha}$. □

Theorem 21 (Robust Preservation of Hyperproperties (detailed))**.** *Let* $\tau : \wp(\mathsf{Trace_S}) \leftrightarrows \wp(\mathbf{Trace_T}) : \sigma$ *be a Galois connection between trace properties. Let*

$$\tau : \wp(\wp(\mathsf{Trace_S})) \leftrightarrows \wp(\wp(\mathbf{Trace_T})) : \breve{\sigma}$$

be the lifting to hyperproperties as in Lemma 11. The followings are equivalent:

$$\begin{aligned}
\mathsf{RHP}^{\tau} &\equiv \forall \mathsf{P}\ \forall H_S \in \wp(\wp(\mathsf{Trace_S}))\text{.}\ \forall \mathsf{C}\text{.}\ \mathsf{C_S}[\mathsf{P}] \models H_S \Rightarrow \forall \mathbf{C_T}\text{.}\ \mathbf{C_T}[\mathsf{P}\!\downarrow] \models \tau(H_S)\\
\mathsf{RHP}^{\breve{\sigma}} &\equiv \forall \mathsf{P}\ \forall H_T \in \wp(\wp(\mathbf{Trace_T}))\text{.}\ \forall \mathsf{C}\text{.}\ \mathsf{C_S}[\mathsf{P}] \models \sigma(H_T) \Rightarrow \forall \mathbf{C_T}\text{.}\ \mathbf{C_T}[\mathsf{P}\!\downarrow] \models H_T\\
\mathsf{RF}^{\breve{\sigma}}_{\tau} &\equiv \forall \mathsf{P}\ \forall \mathbf{C_T}\text{.}\ \mathit{beh}(\mathbf{C_T}[\mathsf{P}\!\downarrow]) \in \{\tau(\mathit{beh}(\mathsf{C_S}[\mathsf{P}])) \mid \mathsf{C_S} \in \mathcal{C}tx\}\\
\mathsf{RHC}^{\sim} &\equiv \forall \mathsf{P}\ \forall \mathbf{C_T}\ \exists \mathsf{C_S}\text{.}\ \mathit{beh}(\mathbf{C_T}[\mathsf{P}\!\downarrow]) = \tau(\mathit{beh}(\mathsf{C_S}[\mathsf{P}]))
\end{aligned}$$

Proof. Apply Lemma 9 with:

- $(C, \sqsubseteq) = (\wp(\wp(\mathsf{Trace_S})), \subseteq), (A, \leq) = (\wp(\wp(\mathbf{Trace_T})), \subseteq)$
- the Galois connection $\tau : \wp(\wp(\mathsf{Trace_S})) \leftrightarrows \wp(\wp(\mathbf{Trace_T})) : \breve{\sigma}$
- $\mathscr{P} = \mathcal{P}rg$, $\mathscr{G} = \mathcal{C}tx$, $[\![\cdot]\!] : \mathcal{C}tx \times \mathcal{P}rg \to \wp(\wp(\mathsf{Trace_S})) = \lambda(\mathsf{C_S}, \mathsf{P}).\{\mathtt{beh}(\mathsf{C_S}[\mathsf{P}])\}$
- $\mathscr{P}' = \mathcal{P}rg$, $\mathscr{G}' = \mathcal{C}tx$, $(\!|\cdot|\!) : \mathcal{C}tx \times \mathcal{P}rg \to \mathit{Closed}_{\subseteq}(\wp(\wp(\mathbf{Trace_T}))) = \lambda(\mathbf{C_T}, \mathbf{Q}).\ \{\mathtt{beh}(\mathbf{C_T}[\mathbf{Q}])\}$
- $\downarrow : \mathcal{P}rg \to \mathcal{P}rg = \cdot\!\downarrow$

and obtain the followings are equivalent:

$$\begin{aligned}
\mathsf{RCP}^{\alpha} &\equiv \forall \mathsf{P}\ \forall H_S \in \wp(\wp(\mathsf{Trace_S}))\text{.}\\
&\quad (\forall \mathsf{C_S}\text{.}\ \{\mathtt{beh}(\mathsf{C_S}[\mathsf{P}])\} \subseteq H_S) \Rightarrow (\forall \mathbf{C_T}\text{.}\ \{\mathtt{beh}(\mathbf{C_T}[\mathsf{P}\!\downarrow])\} \subseteq \tau(H_S))\\
\mathsf{RAP}^{\alpha} &\equiv \forall \mathsf{P}\ \forall \wp(\wp(\mathbf{Trace_T}))\text{.}\\
&\quad (\forall \mathsf{C_S}\text{.}\ \tau(\{\mathtt{beh}(\mathsf{C_S}[\mathsf{P}])\}) \subseteq H_T \Rightarrow (\forall \mathbf{C_T}\text{.}\ \{\mathtt{beh}(\mathbf{C_T}[\mathsf{P}\!\downarrow])\} \subseteq H_T\\
\mathsf{RAP}^{\gamma} &\equiv \forall \mathsf{P}\ \forall H_T \in \wp(\wp(\mathbf{Trace_T}))\text{.}\\
&\quad (\forall \mathsf{C_S}\text{.}\ \{\mathtt{beh}(\mathsf{C_S}[\mathsf{P}])\} \subseteq \breve{\sigma}(H_T) \Rightarrow (\forall \mathbf{C_T}\text{.}\ \{\mathtt{beh}(\mathbf{C_T}[\mathsf{P}\!\downarrow])\} \subseteq H_T\\
\mathsf{RF}^{\gamma}_{\alpha} &\equiv \forall \mathsf{P}\ \forall \mathbf{C_T}\text{.}\ \{\mathtt{beh}(\mathbf{C_T}[\mathsf{P}\!\downarrow])\} \subseteq \tau(\bigcup_{\mathsf{C_S}} \{\mathtt{beh}(\mathsf{C_S}[\mathsf{P}])\})
\end{aligned}$$

It follows immedaitely that $\mathsf{RCP}^{\alpha} \iff \mathsf{RHP}^{\tau}$, $\mathsf{RHP}^{\breve{\sigma}} \iff \mathsf{RAP}^{\gamma}$, $\mathsf{RF}^{\breve{\sigma}}_{\tau} \iff \mathsf{RF}^{\gamma}_{\alpha}$ and $\mathsf{RHC}^{\sim} \iff \mathsf{RF}^{\gamma}_{\alpha}$. □

A.3 Robust Preservation of Classes of Relational Trace Properties

Theorem 22 (Robust Preservation of Relational trace properties (detailed))**.** *Let* $\tau : \wp(\mathsf{Trace_S}) \leftrightarrows \wp(\mathbf{Trace_T}) : \sigma$ *a Galois connection between trace properties, and let* $\sim \subseteq \mathit{Trace_S} \times \mathit{Trace_T}$ *be its corresponding the relation (see Definition 13). For any* $k \in \mathbb{N} \cup \{\omega\}$*, for simplicity denote still by*

$$\tau : \wp(\mathit{Trace_S}^k) \leftrightarrows \wp(\mathit{Trace_T}^k) : \sigma$$

the relational Galois connection for $\tau : \wp(\mathsf{Trace_S}) \leftrightarrows \wp(\mathbf{Trace_T}) : \sigma$ *from Lemma 12. The followings are equivalent:*

$$\begin{aligned} k\mathsf{RTP}^{\tau} \equiv\ & \forall \mathsf{P_1}, \mathsf{P_2}, \ldots \mathsf{P_k}\ \forall \pi_S \in \wp(\mathit{Trace_S}^k). \\ & (\forall \mathsf{C_S}.\ \mathsf{C_S}[\mathsf{P_1}], \mathsf{C_S}[\mathsf{P_2}], \ldots, \mathsf{C_S}[\mathsf{P_k}] \models \pi_S) \Rightarrow \\ & (\forall \mathbf{C_T}.\ \mathbf{C_T}[\mathsf{P_1}\!\downarrow], \mathbf{C_T}[\mathsf{P_2}\!\downarrow], \ldots, \mathbf{C_T}[\mathsf{P_k}\!\downarrow] \models \tau(\pi_S)) \end{aligned}$$

$$\begin{aligned} k\mathsf{RTP}^{\sigma} \equiv\ & \forall \mathsf{P_1}, \mathsf{P_2}, \ldots \mathsf{P_k}\ \forall \pi_T \in \wp(\mathit{Trace_T}^k). \\ & (\forall \mathsf{C_S}.\ \mathsf{C_S}[\mathsf{P_1}], \mathsf{C_S}[\mathsf{P_2}], \ldots, \mathsf{C_S}[\mathsf{P_k}] \models \sigma(\pi_T)) \Rightarrow \\ & (\forall \mathbf{C_T}.\ \mathbf{C_T}[\mathsf{P_1}\!\downarrow], \mathbf{C_T}[\mathsf{P_2}\!\downarrow], \ldots, \mathbf{C_T}[\mathsf{P_k}\!\downarrow] \models \pi_T) \end{aligned}$$

$$\begin{aligned} k\mathsf{RF}^{\sigma}_{\tau} \equiv\ & \forall \mathsf{P_1}, \mathsf{P_2}, \ldots \mathsf{P_k}\ \forall \mathbf{C_T}. \\ & \mathit{beh}(\mathbf{C_T}[\mathsf{P_1}\!\downarrow]) \times \mathit{beh}(\mathbf{C_T}[\mathsf{P_1}\!\downarrow]) \times \ldots \times \mathit{beh}(\mathbf{C_T}[\mathsf{P_k}\!\downarrow]) \subseteq \\ & \tau(\bigcup_{\mathsf{C_S}} \mathit{beh}(\mathsf{C_S}[\mathsf{P_1}]) \times \mathit{beh}(\mathsf{C_S}[\mathsf{P_2}]) \times \ldots \times \mathit{beh}(\mathsf{C_S}[\mathsf{P_k}])) \end{aligned}$$

$$\begin{aligned} k\mathsf{RTC}^{\sim} \equiv\ & \forall \mathsf{P_1}, \mathsf{P_2}, \ldots \mathsf{P_k}\ \forall \mathbf{C_T}\ \forall t_1, t_2, \ldots, t_k. \\ & (\forall i = 1 \ldots k.\ \mathbf{C_T}[\mathsf{P_i}\!\downarrow] \rightsquigarrow t_i) \Rightarrow \exists \mathsf{C_S}\ \exists \mathsf{s_1}, \mathsf{s_2}, \ldots, \mathsf{s_k}. \\ & (\forall i = 1 \ldots k.\ \mathsf{s_i} \sim t_i \wedge \mathsf{C_S}[\mathsf{P_i}] \rightsquigarrow \mathsf{s_i}) \end{aligned}$$

Proof. Apply Lemma 9 with:

- $(C, \sqsubseteq) = (\wp(\mathit{Trace_S}^k), \subseteq), (A, \leq) = (\wp(\mathit{Trace_T}^k), \subseteq)$
- the Galois connection $\tau : \wp(\mathit{Trace_S}^k) \leftrightarrows \wp(\mathit{Trace_T}^k) : \sigma$
- $\mathscr{P} = \mathcal{P}\mathit{rg}^k$, $\mathscr{G} = \mathcal{C}\mathit{tx}$,

  $[\![\cdot]\!] : \mathcal{C}\mathit{tx} \times \mathcal{P}\mathit{rg}^k \to \wp(\mathit{Trace_S}^k) = \lambda(\mathsf{C_S}, (\mathsf{P_1}, \mathsf{P_2}, \ldots, \mathsf{P_k})).$
  $\mathsf{beh}(\mathsf{C_S}[\mathsf{P_1}]) \times \mathsf{beh}(\mathsf{C_S}[\mathsf{P_2}]) \times \ldots \times \mathsf{beh}(\mathsf{C_S}[\mathsf{P_k}])$

- $\mathscr{P}' = \mathcal{P}\mathit{rg}$, $\mathscr{G}' = \mathcal{C}\mathit{tx}$,

  $(\!|\cdot|\!) : \mathcal{C}\mathit{tx} \times \mathcal{P}\mathit{rg} \to \wp(\mathit{Trace_T}^k) = \lambda(\mathbf{C_T}, (\mathbf{Q_1}, \mathbf{Q_2}, \ldots, \mathbf{Q_k})).$
  $\mathsf{beh}(\mathbf{C_T}[\mathbf{Q_1}]) \times \mathsf{beh}(\mathbf{C_T}[\mathbf{Q_2}]) \times \ldots \times \mathsf{beh}(\mathbf{C_T}[\mathbf{Q_1}])$

- $\downarrow : \mathcal{P}\mathit{rg}^k \to \mathcal{P}\mathit{rg}^k = \cdot\!\downarrow^k$

and obtain the followings are equivalent:

$$\begin{aligned}\mathsf{RCP}^{\alpha} \equiv\ & \forall \mathsf{P}_1\mathsf{P}_2\ldots\mathsf{P}_k\ \forall \pi_S \in \wp(\mathit{Trace}_S{}^k).\\ & (\forall \mathsf{C}_\mathsf{S}.\ \mathtt{beh}(\mathsf{C}_\mathsf{S}[\mathsf{P}_1]) \times \mathtt{beh}(\mathsf{C}_\mathsf{S}[\mathsf{P}_2]) \times \ldots \times \mathtt{beh}(\mathsf{C}_\mathsf{S}[\mathsf{P}_k]) \subseteq \pi_S) \Rightarrow\\ & (\forall \mathbf{C_T}.\ \mathtt{beh}(\mathbf{C_T}[\mathsf{P}_1{\downarrow}]) \times \mathtt{beh}(\mathbf{C_T}[\mathsf{P}_2{\downarrow}]) \times \ldots \times \mathtt{beh}(\mathbf{C_T}[\mathsf{P}_k{\downarrow}]) \subseteq \tau(\pi_S))\end{aligned}$$

$$\begin{aligned}\mathsf{RAP}^{\alpha} \equiv\ & \forall \mathsf{P}_1\mathsf{P}_2\ldots\mathsf{P}_k\ \forall \pi_T \in \wp(\mathit{Trace}_T{}^k).\\ & (\forall \mathsf{C}_\mathsf{S}.\ \tau(\mathtt{beh}(\mathsf{C}_\mathsf{S}[\mathsf{P}_1]) \times \mathtt{beh}(\mathsf{C}_\mathsf{S}[\mathsf{P}_2]) \times \ldots \times \mathtt{beh}(\mathsf{C}_\mathsf{S}[\mathsf{P}_k])) \subseteq \pi_T) \Rightarrow\\ & (\forall \mathbf{C_T}.\ \mathtt{beh}(\mathbf{C_T}[\mathsf{P}_1{\downarrow}]) \times \mathtt{beh}(\mathbf{C_T}[\mathsf{P}_2{\downarrow}]) \times \ldots \times \mathtt{beh}(\mathbf{C_T}[\mathsf{P}_k{\downarrow}]) \subseteq \pi_T)\end{aligned}$$

$$\begin{aligned}\mathsf{RAP}^{\gamma} \equiv\ & \forall \mathsf{P}_1\mathsf{P}_2\ldots\mathsf{P}_k\ \forall \pi_T \in \wp(\mathit{Trace}_T{}^k).\\ & (\forall \mathsf{C}_\mathsf{S}.\ \mathtt{beh}(\mathsf{C}_\mathsf{S}[\mathsf{P}_1]) \times \mathtt{beh}(\mathsf{C}_\mathsf{S}[\mathsf{P}_2]) \times \ldots \times \mathtt{beh}(\mathsf{C}_\mathsf{S}[\mathsf{P}_k]) \subseteq \sigma(\pi_T)) \Rightarrow\\ & (\forall \mathbf{C_T}.\ \mathtt{beh}(\mathbf{C_T}[\mathsf{P}_1{\downarrow}]) \times \mathtt{beh}(\mathbf{C_T}[\mathsf{P}_2{\downarrow}]) \times \ldots \times \mathtt{beh}(\mathbf{C_T}[\mathsf{P}_k{\downarrow}]) \subseteq \pi_T)\end{aligned}$$

$$\begin{aligned}\mathsf{RF}^{\gamma}_{\alpha} \equiv\ & \forall \mathsf{P}_1, \mathsf{P}_2, \ldots \mathsf{P}_k\ \forall \mathbf{C_T}.\\ & \mathtt{beh}(\mathbf{C_T}[\mathsf{P}_1{\downarrow}]) \times \mathtt{beh}(\mathbf{C_T}[\mathsf{P}_1{\downarrow}]) \times \ldots \times \mathtt{beh}(\mathbf{C_T}[\mathsf{P}_k{\downarrow}]) \subseteq\\ & \tau(\bigcup_{\mathsf{C}_\mathsf{S}} \mathtt{beh}(\mathsf{C}_\mathsf{S}[\mathsf{P}_1]) \times \mathtt{beh}(\mathsf{C}_\mathsf{S}[\mathsf{P}_2]) \times \ldots \times \mathtt{beh}(\mathsf{C}_\mathsf{S}[\mathsf{P}_k]))\end{aligned}$$

It follows immediately that $\mathsf{RCP}^{\alpha} \iff k\mathsf{RTP}^{\tau}$, $k\mathsf{RTP}^{\sigma} \iff \mathsf{RAP}^{\gamma}$, $\mathsf{RF}^{\sigma}_{\tau} \iff \mathsf{RF}^{\gamma}_{\alpha}$. Finally $\mathsf{RTC}^{\sim} \iff \mathsf{RF}^{\gamma}_{\alpha}$, because

$$\begin{aligned}&\tau(\bigcup_{\mathsf{C}_\mathsf{S}} \mathtt{beh}(\mathsf{C}_\mathsf{S}[\mathsf{P}_1]) \times \mathtt{beh}(\mathsf{C}_\mathsf{S}[\mathsf{P}_2]) \times \ldots \times \mathtt{beh}(\mathsf{C}_\mathsf{S}[\mathsf{P}_k])) = && \text{[by Remark 4]}\\ &\bigcup_{\mathsf{C}_\mathsf{S}} \tau(\mathtt{beh}(\mathsf{C}_\mathsf{S}[\mathsf{P}_1]) \times \mathtt{beh}(\mathsf{C}_\mathsf{S}[\mathsf{P}_2]) \times \ldots \times \mathtt{beh}(\mathsf{C}_\mathsf{S}[\mathsf{P}_k])) = && \text{[by Lemma 13]}\\ &\bigcup_{\mathsf{C}_\mathsf{S}} \tau(\mathtt{beh}(\mathsf{C}_\mathsf{S}[\mathsf{P}_1])) \times \tau(\mathtt{beh}(\mathsf{C}_\mathsf{S}[\mathsf{P}_2])) \times \ldots \times \tau(\mathtt{beh}(\mathsf{C}_\mathsf{S}[\mathsf{P}_k]))\end{aligned}$$

and the thesis follows by unfolding the definition of $\tau : \wp(\mathsf{Trace_S}) \to \wp(\mathbf{Trace_T})$ pointwise. □

Theorem 23 (Robust Preservation of *krelSafety* (detailed))**.** *Let* $\tau : \wp(\mathsf{Trace_S}) \leftrightarrows \wp(\mathbf{Trace_T}) : \sigma$ *a Galois connection between trace properties, and let* $\sim \subseteq \mathit{Trace}_S \times \mathit{Trace}_T$ *be its corresponding the relation (see Definition 13). For any* $k \in \mathbb{N} \cup \{\omega\}$*, for simplicity denote still by*

$$\mathit{Safe} \circ \tau : \wp(\mathit{Trace}_S{}^k) \leftrightarrows \mathit{krelSafety}_T : \sigma$$

the composition of the relational Galois connection for $\tau : \wp(\mathsf{Trace_S}) \leftrightarrows \wp(\mathbf{Trace_T}) : \sigma$ *from Lemma 12 and the colsure operator Safe that maps every target relational trace property to the smallest k-relational safety property that includes it.*

The followings are equivalent:

$$
\begin{aligned}
k\mathsf{RTP}^{Safe\circ\tau} \equiv\ & \forall \mathsf{P}_1, \mathsf{P}_2, \ldots \mathsf{P}_k\ \forall \pi_S \in \wp(Trace_S{}^k).\\
& (\forall \mathsf{C_S}.\ \mathsf{C_S}[\mathsf{P}_1], \mathsf{C_S}[\mathsf{P}_2], \ldots, \mathsf{C_S}[\mathsf{P}_k] \models \pi_S) \Rightarrow\\
& (\forall \mathbf{C_T}.\ \mathbf{C_T}[\mathsf{P}_1{\downarrow}], \mathbf{C_T}[\mathsf{P}_2{\downarrow}], \ldots, \mathbf{C_T}[\mathsf{P}_k{\downarrow}] \models Safe \circ \tau(\pi_S))
\end{aligned}
$$

$$
\begin{aligned}
k\mathsf{RSP}^{\sigma} \equiv\ & \forall \mathsf{P}_1, \mathsf{P}_2, \ldots \mathsf{P}_k\ \forall \pi_T \in krelSafety.\\
& (\forall \mathsf{C_S}.\ \mathsf{C_S}[\mathsf{P}_1], \mathsf{C_S}[\mathsf{P}_2], \ldots, \mathsf{C_S}[\mathsf{P}_k] \models \sigma(\pi_T)) \Rightarrow\\
& (\forall \mathbf{C_T}.\ \mathbf{C_T}[\mathsf{P}_1{\downarrow}], \mathbf{C_T}[\mathsf{P}_2{\downarrow}], \ldots, \mathbf{C_T}[\mathsf{P}_k{\downarrow}] \models \pi_T)
\end{aligned}
$$

$$
\begin{aligned}
k\mathsf{RF}^{\sigma}_{Safe\circ\tau} \equiv\ & \forall \mathsf{P}_1, \mathsf{P}_2, \ldots \mathsf{P}_k\ \forall \mathbf{C_T}.\\
& beh(\mathbf{C_T}[\mathsf{P}_1{\downarrow}]) \times beh(\mathbf{C_T}[\mathsf{P}_1{\downarrow}]) \times \ldots \times beh(\mathbf{C_T}[\mathsf{P}_k{\downarrow}]) \subseteq\\
& Safe \circ \tau(\bigcup_{\mathsf{C_S}} beh(\mathsf{C_S}[\mathsf{P}_1]) \times beh(\mathsf{C_S}[\mathsf{P}_2]) \times \ldots \times beh(\mathsf{C_S}[\mathsf{P}_k]))
\end{aligned}
$$

$$
\begin{aligned}
k\mathsf{RSC}^{\sim} \equiv\ & \forall \mathsf{P}_1, \mathsf{P}_2, \ldots \mathsf{P}_k\ \forall \mathbf{C_T}\ \forall A_T \in \mathcal{O}_T{}^k.\\
& beh(\mathbf{C_T}[\mathsf{P}_1{\downarrow}]) \times beh(\mathbf{C_T}[\mathsf{P}_2{\downarrow}]) \times \ldots \times beh(\mathbf{C_T}[\mathsf{P}_k{\downarrow}]) \cap A_T \neq \emptyset \Rightarrow\\
& \tau(beh(\mathsf{C_S}[\mathsf{P}_1]) \times beh(\mathsf{C_S}[\mathsf{P}_2]) \times \ldots \times beh(\mathsf{C_S}[\mathsf{P}_k])) \cap A_T \neq \emptyset
\end{aligned}
$$

Proof. Apply Lemma 9 with:

- $(C, \sqsubseteq) = (\wp(Trace_S{}^k), \subseteq), (A, \leq) = (krelSafety_T, \subseteq)$
- the Galois connection $Safe \circ \tau : \wp(Trace_S{}^k) \leftrightarrows krelSafety_T : \sigma$
- $\mathscr{P} = \mathcal{P}rg^k$, $\mathscr{G} = \mathcal{C}tx$,

  $[\![\cdot]\!] : \mathcal{C}tx \times \mathcal{P}rg^k \to \wp(Trace_S{}^k) = \lambda(\mathsf{C_S}, (\mathsf{P}_1, \mathsf{P}_2, \ldots, \mathsf{P}_k)).$
  $\mathtt{beh}(\mathsf{C_S}[\mathsf{P}_1]) \times \mathtt{beh}(\mathsf{C_S}[\mathsf{P}_2]) \times \ldots \times \mathtt{beh}(\mathsf{C_S}[\mathsf{P}_k])$

- $\mathscr{P}' = \mathcal{P}rg$, $\mathscr{G}' = \mathcal{C}tx$,

  $(\!|\cdot|\!) : \mathcal{C}tx \times \mathcal{P}rg \to \wp(Trace_T{}^k) = \lambda(\mathbf{C_T}, (\mathbf{Q_1}, \mathbf{Q_2}, \ldots, \mathbf{Q_k})).$
  $Safe(\mathtt{beh}(\mathbf{C_T}[\mathbf{Q_1}]) \times \mathtt{beh}(\mathbf{C_T}[\mathbf{Q_2}]) \times \ldots \times \mathtt{beh}(\mathbf{C_T}[\mathbf{Q_1}]))$

- $\downarrow : \mathcal{P}rg^k \to \mathcal{P}rg^k = \cdot{\downarrow}^k$

and obtain the followings are equivalent:

$$\begin{aligned}\mathsf{RCP}^{\alpha} \equiv\ & \forall \mathsf{P}_1 \mathsf{P}_2 \ldots \mathsf{P}_k\ \forall \pi_S \in \wp(\mathit{Trace}_S{}^k).\\ & (\forall \mathsf{C_S}.\ \mathtt{beh}(\mathsf{C_S}[\mathsf{P}_1]) \times \mathtt{beh}(\mathsf{C_S}[\mathsf{P}_2]) \times \ldots \times \mathtt{beh}(\mathsf{C_S}[\mathsf{P}_k]) \subseteq \pi_S) \Rightarrow\\ & (\forall \mathbf{C_T}.\ \mathit{Safe}(\mathtt{beh}(\mathbf{C_T}[\mathsf{P}_1\!\downarrow]) \times \mathtt{beh}(\mathbf{C_T}[\mathsf{P}_2\!\downarrow]) \times \ldots \times \mathtt{beh}(\mathbf{C_T}[\mathsf{P}_k\!\downarrow])) \subseteq \mathit{Safe} \circ \tau(\pi_S))\end{aligned}$$

$$\begin{aligned}\mathsf{RAP}^{\alpha} \equiv\ & \forall \mathsf{P}_1 \mathsf{P}_2 \ldots \mathsf{P}_k\ \forall \pi_T \in \mathit{krelSafety}_T.\\ & (\forall \mathsf{C_S}.\ \mathit{Safe} \circ \tau(\mathtt{beh}(\mathsf{C_S}[\mathsf{P}_1]) \times \mathtt{beh}(\mathsf{C_S}[\mathsf{P}_2]) \times \ldots \times \mathtt{beh}(\mathsf{C_S}[\mathsf{P}_k])) \subseteq \pi_T) \Rightarrow\\ & (\forall \mathbf{C_T}.\ \mathit{Safe}(\mathtt{beh}(\mathbf{C_T}[\mathsf{P}_1\!\downarrow]) \times \mathtt{beh}(\mathbf{C_T}[\mathsf{P}_2\!\downarrow]) \times \ldots \times \mathtt{beh}(\mathbf{C_T}[\mathsf{P}_k\!\downarrow]) \subseteq \pi_T))\end{aligned}$$

$$\begin{aligned}\mathsf{RAP}^{\gamma} \equiv\ & \forall \mathsf{P}_1 \mathsf{P}_2 \ldots \mathsf{P}_k\ \forall \pi_T \in \mathit{krelSafety}_T.\\ & (\forall \mathsf{C_S}.\ \mathtt{beh}(\mathsf{C_S}[\mathsf{P}_1]) \times \mathtt{beh}(\mathsf{C_S}[\mathsf{P}_2]) \times \ldots \times \mathtt{beh}(\mathsf{C_S}[\mathsf{P}_k]) \subseteq \sigma(\pi_T)) \Rightarrow\\ & (\forall \mathbf{C_T}.\ \mathit{Safe}(\mathtt{beh}(\mathbf{C_T}[\mathsf{P}_1\!\downarrow]) \times \mathtt{beh}(\mathbf{C_T}[\mathsf{P}_2\!\downarrow]) \times \ldots \times \mathtt{beh}(\mathbf{C_T}[\mathsf{P}_k\!\downarrow]) \subseteq \pi_T))\end{aligned}$$

$$\begin{aligned}\mathsf{RF}^{\gamma}_{\alpha} \equiv\ & \forall \mathsf{P}_1, \mathsf{P}_2, \ldots \mathsf{P}_k\ \forall \mathbf{C_T}.\\ & \mathit{Safe}(\mathtt{beh}(\mathbf{C_T}[\mathsf{P}_1\!\downarrow]) \times \mathtt{beh}(\mathbf{C_T}[\mathsf{P}_1\!\downarrow]) \times \ldots \times \mathtt{beh}(\mathbf{C_T}[\mathsf{P}_k\!\downarrow])) \subseteq\\ & \mathit{Safe} \circ \tau(\bigcup_{\mathsf{C_S}} \mathtt{beh}(\mathsf{C_S}[\mathsf{P}_1]) \times \mathtt{beh}(\mathsf{C_S}[\mathsf{P}_2]) \times \ldots \times \mathtt{beh}(\mathsf{C_S}[\mathsf{P}_k])))\end{aligned}$$

From Remark 7, it follows immediately that $\mathsf{RCP}^{\alpha} \iff k\mathsf{RTP}^{\mathit{Safe}\circ\tau}$, $k\mathsf{RSP}^{\sigma} \iff \mathsf{RAP}^{\gamma}$, $\mathsf{RF}^{\sigma}_{\mathit{Safe}\tau} \iff \mathsf{RF}^{\gamma}_{\alpha}$. Finally $k\mathsf{RSC}^{\sim} \iff \mathsf{RF}^{\gamma}_{\alpha}$, with the same argument as in Lemma 16. □

A.4 Robust Preservation of Classes of Relational Hyperproperties

Theorem 24 (Robust Preservation of *krelHypersafety* (detailed))**.**
Let $\tau : \wp(\mathsf{Trace_S}) \leftrightarrows \wp(\mathbf{Trace_T}) : \sigma$ *a Galois connection between trace properties, and let* $\sim \subseteq \mathit{Trace}_S \times \mathit{Trace}_T$ *be its corresponding the relation (see Definition 13). For any* $k \in \mathbb{N} \cup \{\omega\}$*, for simplicity denote still by*

$$\mathit{Hsafe} \circ \tau : \mathit{Closed}_{\subseteq}(\wp(\mathit{Trace}_S{}^k)) \leftrightarrows \mathit{krelSafety}_T : \mathit{Cl}_{\subseteq} \circ \sigma$$

the Galois connection between source relational hyperproperties closed under subsets and target realtional hypersafety (of the same arity), obtained by lifting (see Lemma 10) the relational Galois connection for $\tau : \wp(\mathsf{Trace_S}) \leftrightarrows \wp(\mathbf{Trace_T}) : \sigma$ *from Lemma 12 and then composing with the colsure operator Safe that maps every target k-relational hyperproperty to the smallest k-relational hypersafety that includes it.*
The followings are equivalent:

$$\begin{aligned} k\mathsf{RH}_{\subseteq}\mathsf{P}^{\mathit{Safe}\circ\tau} \equiv\ & \forall \mathsf{P_1}, \mathsf{P_2}, \ldots \mathsf{P_k}\ \forall H_S \in \mathit{Closed}_{\subseteq}(\wp(\mathsf{Trace_S})^k). \\ & (\forall \mathsf{C_S}.\ \mathsf{C_S}[\mathsf{P_1}], \mathsf{C_S}[\mathsf{P_2}], \ldots, \mathsf{C_S}[\mathsf{P_k}] \models H_S) \Rightarrow \\ & (\forall \mathbf{C_T}.\ \mathbf{C_T}[\mathsf{P_1}\!\downarrow], \mathbf{C_T}[\mathsf{P_2}\!\downarrow], \ldots, \mathbf{C_T}[\mathsf{P_k}\!\downarrow] \models \mathit{Hsafe} \circ \tau(H_S)) \end{aligned}$$

$$\begin{aligned} k\mathsf{RHSP}^{\sigma} \equiv\ & \forall \mathsf{P_1}, \mathsf{P_2}, \ldots \mathsf{P_k}\ \forall H_T \in \mathit{krelHypersafety}_T. \\ & (\forall \mathsf{C_S}.\ \mathsf{C_S}[\mathsf{P_1}], \mathsf{C_S}[\mathsf{P_2}], \ldots, \mathsf{C_S}[\mathsf{P_k}] \models \sigma(H_T)) \Rightarrow \\ & (\forall \mathbf{C_T}.\ \mathbf{C_T}[\mathsf{P_1}\!\downarrow], \mathbf{C_T}[\mathsf{P_2}\!\downarrow], \ldots, \mathbf{C_T}[\mathsf{P_k}\!\downarrow] \models H_T) \end{aligned}$$

$$\begin{aligned} k\mathsf{RF}^{\sigma}_{\mathit{Hsafe}\circ\tau} \equiv\ & \forall \mathsf{P_1}, \mathsf{P_2}, \ldots \mathsf{P_k}\ \forall \mathbf{C_T}. \\ & \mathit{beh}(\mathbf{C_T}[\mathsf{P_1}\!\downarrow]) \times \mathit{beh}(\mathbf{C_T}[\mathsf{P_1}\!\downarrow]) \times \ldots \times \mathit{beh}(\mathbf{C_T}[\mathsf{P_k}\!\downarrow]) \in \\ & \mathit{Hsafe} \circ \tau(\{\mathsf{C_S}\,\mathit{beh}(\mathsf{C_S}[\mathsf{P_1}]) \times \mathit{beh}(\mathsf{C_S}[\mathsf{P_2}]) \times \ldots \times \mathit{beh}(\mathsf{C_S}[\mathsf{P_k}]) \mid \mathsf{C_S} \in \mathcal{C}tx\}) \end{aligned}$$

$$\begin{aligned} k\mathsf{RHSC}^{\sim} \equiv\ & \forall \mathsf{P_1}, \mathsf{P_2}, \ldots \mathsf{P_k}\ \forall \mathbf{C_T}\ \forall A_T \in (\mathcal{V}_{\mathit{low}}(\mathcal{O}_T))^k. \\ & \mathit{Cl}_{\subseteq}(\{\mathit{beh}(\mathbf{C_T}[\mathsf{P_1}\!\downarrow]) \times \mathit{beh}(\mathbf{C_T}[\mathsf{P_2}\!\downarrow]) \times \ldots \times \mathit{beh}(\mathbf{C_T}[\mathsf{P_k}\!\downarrow])\}) \cap A_T \neq \emptyset \Rightarrow \\ & \exists \mathsf{C_S}.\mathit{Cl}_{\subseteq} \circ \tau(\{\mathit{beh}(\mathsf{C_S}[\mathsf{P_1}]) \times \mathit{beh}(\mathsf{C_S}[\mathsf{P_2}]) \times \ldots \times \mathit{beh}(\mathsf{C_S}[\mathsf{P_k}])\}) \cap A_T \neq \emptyset \end{aligned}$$

Proof. Apply Lemma 9 with:

- $(C, \sqsubseteq) = (\mathit{Closed}_{\subseteq}(\wp(\mathit{Trace}_S{}^k)), \subseteq), (A, \leq) = (\mathit{krelHypersafety}_T, \subseteq)$
- the Galois connection $\mathit{Hsafe} \circ \tau : \mathit{Closed}_{\subseteq}(\wp(\mathit{Trace}_S{}^k)) \leftrightarrows \mathit{krelHypersafety}_T : \mathit{Cl}_{\subseteq}\sigma$
- $\mathscr{P} = \mathcal{P}rg^k$, $\mathscr{G} = \mathcal{C}tx$,

  $[\![\cdot]\!] : \mathcal{C}tx \times \mathcal{P}rg^k \to \wp(\mathit{Trace}_S{}^k) = \lambda(\mathsf{C_S}, (\mathsf{P_1}, \mathsf{P_2}, \ldots, \mathsf{P_k})).$
  $\mathit{Cl}_{\subseteq}(\{\mathsf{beh}(\mathsf{C_S}[\mathsf{P_1}]) \times \mathsf{beh}(\mathsf{C_S}[\mathsf{P_2}]) \times \ldots \times \mathsf{beh}(\mathsf{C_S}[\mathsf{P_k}])\})$
- $\mathscr{P}' = \mathcal{P}rg$, $\mathscr{G}' = \mathcal{C}tx$,

  $(\!|\cdot|\!) : \mathcal{C}tx \times \mathcal{P}rg \to \wp(\mathit{Trace}_T{}^k) = \lambda(\mathbf{C_T}, (\mathbf{Q_1}, \mathbf{Q_2}, \ldots, \mathbf{Q_k})).$
  $\mathit{Hsafe}(\{\mathsf{beh}(\mathbf{C_T}[\mathbf{Q_1}]) \times \mathsf{beh}(\mathbf{C_T}[\mathbf{Q_2}]) \times \ldots \times \mathsf{beh}(\mathbf{C_T}[\mathbf{Q_1}])\})$

- $\downarrow : \mathcal{P}rg^k \to \mathcal{P}rg^k = \cdot\downarrow^k$

and obtain the followings are equivalent:

$$\begin{aligned}\mathsf{RCP}^{\alpha} \equiv\ & \forall \mathsf{P}_1 \mathsf{P}_2 \ldots \mathsf{P}_k\ \forall H_S \in \mathit{Closed}_{\subseteq}(\wp(\mathsf{Trace_S})^k).\\ & (\forall \mathsf{C_S}.\ \mathit{Cl}_{\subseteq}(\{\mathtt{beh}(\mathsf{C_S}[\mathsf{P}_1]) \times \mathtt{beh}(\mathsf{C_S}[\mathsf{P}_2]) \times \ldots \times \mathtt{beh}(\mathsf{C_S}[\mathsf{P}_k])\}) \subseteq \pi_S) \Rightarrow\\ & (\forall \mathbf{C_T}.\ \mathit{Hsafe}(\{\mathtt{beh}(\mathbf{C_T}[\mathsf{P}_1\downarrow]) \times \mathtt{beh}(\mathbf{C_T}[\mathsf{P}_2\downarrow]) \times \ldots \times \mathtt{beh}(\mathbf{C_T}[\mathsf{P}_k\downarrow])\}) \subseteq \mathit{Hsafe} \circ \tau(H_S))\end{aligned}$$

$$\begin{aligned}\mathsf{RAP}^{\alpha} \equiv\ & \forall \mathsf{P}_1 \mathsf{P}_2 \ldots \mathsf{P}_k\ \forall \pi_T \in \mathit{krelHypersafety}_T.\\ & (\forall \mathsf{C_S}.\ \mathit{Hsafe} \circ \tau(\mathit{Cl}_{\subseteq}\{\mathtt{beh}(\mathsf{C_S}[\mathsf{P}_1]) \times \mathtt{beh}(\mathsf{C_S}[\mathsf{P}_2]) \times \ldots \times \mathtt{beh}(\mathsf{C_S}[\mathsf{P}_k])\}) \subseteq H_T) \Rightarrow\\ & (\forall \mathbf{C_T}.\ \mathit{Hsafe}(\{\mathtt{beh}(\mathbf{C_T}[\mathsf{P}_1\downarrow]) \times \mathtt{beh}(\mathbf{C_T}[\mathsf{P}_2\downarrow]) \times \ldots \times \mathtt{beh}(\mathbf{C_T}[\mathsf{P}_k\downarrow]) \subseteq \pi_T\}))\end{aligned}$$

$$\begin{aligned}\mathsf{RAP}^{\gamma} \equiv\ & \forall \mathsf{P}_1 \mathsf{P}_2 \ldots \mathsf{P}_k\ \forall \pi_T \in \mathit{krelHypersafety}_T.\\ & (\forall \mathsf{C_S}.\ \mathit{Cl}_{\subseteq}(\{\mathtt{beh}(\mathsf{C_S}[\mathsf{P}_1]) \times \mathtt{beh}(\mathsf{C_S}[\mathsf{P}_2]) \times \ldots \times \mathtt{beh}(\mathsf{C_S}[\mathsf{P}_k]) \subseteq \mathit{Cl}_{\subseteq} \circ \sigma(H_T)\})) \Rightarrow\\ & (\forall \mathbf{C_T}.\ \mathit{Hsafe}(\{\mathtt{beh}(\mathbf{C_T}[\mathsf{P}_1\downarrow]) \times \mathtt{beh}(\mathbf{C_T}[\mathsf{P}_2\downarrow]) \times \ldots \times \mathtt{beh}(\mathbf{C_T}[\mathsf{P}_k\downarrow]) \subseteq H_T\}))\end{aligned}$$

$$\begin{aligned}\mathsf{RF}^{\gamma}_{\alpha} \equiv\ & \forall \mathsf{P}_1, \mathsf{P}_2, \ldots \mathsf{P}_k\ \forall \mathbf{C_T}.\\ & \mathit{Hsafe}(\{\mathtt{beh}(\mathbf{C_T}[\mathsf{P}_1\downarrow]) \times \mathtt{beh}(\mathbf{C_T}[\mathsf{P}_1\downarrow]) \times \ldots \times \mathtt{beh}(\mathbf{C_T}[\mathsf{P}_k\downarrow])\}) \subseteq\\ & \mathit{Hsafe} \circ \tau(\{\mathtt{beh}(\mathsf{C_S}[\mathsf{P}_1]) \times \mathtt{beh}(\mathsf{C_S}[\mathsf{P}_2]) \times \ldots \times \mathtt{beh}(\mathsf{C_S}[\mathsf{P}_k]) \mid \mathsf{C_S} \in \mathcal{C}tx\}))\end{aligned}$$

With the same argument as in Theorem 19, one can show that $\mathsf{RCP}^{\alpha} \iff k\mathsf{RH}_{\subseteq}\mathsf{P}^{\mathit{Safe} \circ \tau}$, $k\mathsf{RHSP}^{\sigma} \iff \mathsf{RAP}^{\gamma}$, $\mathsf{RF}^{\sigma}_{\mathit{Hsafe} \circ \tau} \iff \mathsf{RF}^{\gamma}_{\alpha}$ and $k\mathsf{RHSC}^{\sim} \iff \mathsf{RF}^{\gamma}_{\alpha}$. □

Theorem 25 (Robust Preservation of $\mathit{Closed}_{\subseteq}$ of $\wp(\mathit{Trace})^k$ (detailed))**.** *Let* $\tau : \wp(\mathsf{Trace_S}) \leftrightarrows \wp(\mathbf{Trace_T}) : \sigma$ *a Galois connection between trace properties, and let* $\sim \subseteq \mathit{Trace}_S \times \mathit{Trace}_T$ *be its corresponding the relation (see Definition 13). For any* $k \in \mathbb{N} \cup \{\omega\}$*, for simplicity denote still by*

$$\mathit{Cl}_{\subseteq} \circ \tau : \mathit{Closed}_{\subseteq}(\wp(\mathit{Trace}_S{}^k)) \leftrightarrows \mathit{Closed}_{\subseteq}(\wp(\mathit{Trace}_T{}^k)) : \mathit{Cl}_{\subseteq} \circ \sigma$$

the Galois connection between source and target relational hyperproperties closed under subsets (of the same arity), obtained by lifting (see Lemma 10) the relational Galois connection for $\tau : \wp(\mathsf{Trace_S}) \leftrightarrows \wp(\mathbf{Trace_T}) : \sigma$ *from Lemma 12.*

The followings are equivalent:

$$
\begin{aligned}
k\mathsf{RH}_{\subseteq}\mathsf{P}^{\tau} \equiv\ & \forall \mathsf{P}_1, \mathsf{P}_2, \ldots \mathsf{P}_k\ \forall H_S \in \mathit{Closed}_{\subseteq}(\wp(\mathsf{Trace_S})^k). \\
& (\forall \mathsf{C_S}.\ \mathsf{C_S}[\mathsf{P}_1], \mathsf{C_S}[\mathsf{P}_2], \ldots, \mathsf{C_S}[\mathsf{P}_k] \models H_S) \Rightarrow \\
& (\forall \mathbf{C_T}.\ \mathbf{C_T}[\mathsf{P}_1\!\downarrow], \mathbf{C_T}[\mathsf{P}_2\!\downarrow], \ldots, \mathbf{C_T}[\mathsf{P}_k\!\downarrow] \models \mathit{Cl}_{\subseteq} \circ \tau(H_S))
\end{aligned}
$$

$$
\begin{aligned}
k\mathsf{RH}_{\subseteq}\mathsf{P}^{\sigma} \equiv\ & \forall \mathsf{P}_1, \mathsf{P}_2, \ldots \mathsf{P}_k\ \forall H_T \in \mathit{Closed}_{\subseteq}(\wp(\mathbf{Trace_T})^k). \\
& (\forall \mathsf{C_S}.\ \mathsf{C_S}[\mathsf{P}_1], \mathsf{C_S}[\mathsf{P}_2], \ldots, \mathsf{C_S}[\mathsf{P}_k] \models \mathit{Cl}_{\subseteq} \circ \sigma(H_T)) \Rightarrow \\
& (\forall \mathbf{C_T}.\ \mathbf{C_T}[\mathsf{P}_1\!\downarrow], \mathbf{C_T}[\mathsf{P}_2\!\downarrow], \ldots, \mathbf{C_T}[\mathsf{P}_k\!\downarrow] \models H_T)
\end{aligned}
$$

$$
\begin{aligned}
k\mathsf{RF}^{\mathit{Cl}_{\subseteq}\circ\sigma}_{\mathit{Cl}_{\subseteq}\circ\tau} \equiv\ & \forall \mathsf{P}_1, \mathsf{P}_2, \ldots \mathsf{P}_k\ \forall \mathbf{C_T}. \\
& \mathit{beh}(\mathbf{C_T}[\mathsf{P}_1\!\downarrow]) \times \mathit{beh}(\mathbf{C_T}[\mathsf{P}_1\!\downarrow]) \times \ldots \times \mathit{beh}(\mathbf{C_T}[\mathsf{P}_k\!\downarrow]) \in \\
& \mathit{Cl}_{\subseteq} \circ \tau(\{\mathit{beh}(\mathsf{C_S}[\mathsf{P}_1]) \times \mathit{beh}(\mathsf{C_S}[\mathsf{P}_2]) \times \ldots \times \mathit{beh}(\mathsf{C_S}[\mathsf{P}_k]) \mid \mathsf{C_S} \in \mathcal{C}tx\})
\end{aligned}
$$

$$
\begin{aligned}
k\mathsf{RH}_{\subseteq}\mathsf{P}^{\sim} \equiv\ & \forall \mathsf{P}_1, \mathsf{P}_2, \ldots \mathsf{P}_k\ \forall \mathbf{C_T}\ \exists \mathsf{C_S}. \\
& (\forall i = 1 \ldots k.\ \mathit{beh}(\mathbf{C_T}[\mathsf{P}_i\!\downarrow]) \subseteq \tau(\mathit{beh}(\mathsf{C_S}[\mathsf{P}])))
\end{aligned}
$$

Proof. Apply Lemma 9 with:

- $(C, \sqsubseteq) = (\mathit{Closed}_{\subseteq}(\wp(\mathit{Trace_S}^k)), \subseteq), (A, \leq) = (\mathit{Closed}_{\subseteq}(\wp(\mathit{Trace_T}^k)), \subseteq)$
- the Galois connection $\mathit{Cl}_{\subseteq} \circ \tau : \mathit{Closed}_{\subseteq}(\wp(\mathit{Trace_S}^k)) \leftrightarrows \mathit{Closed}_{\subseteq}(\wp(\mathit{Trace_S}^k)) : \mathit{Cl}_{\subseteq} \circ \sigma$
- $\mathscr{P} = \mathcal{P}rg^k$, $\mathscr{G} = \mathcal{C}tx$,

  $[\![\cdot]\!] : \mathcal{C}tx \times \mathcal{P}rg^k \to \wp(\mathit{Trace_S}^k) = \lambda(\mathsf{C_S}, (\mathsf{P}_1, \mathsf{P}_2, \ldots, \mathsf{P}_k)).$
  $\mathit{Cl}_{\subseteq}(\{\mathsf{beh}(\mathsf{C_S}[\mathsf{P}_1]) \times \mathsf{beh}(\mathsf{C_S}[\mathsf{P}_2]) \times \ldots \times \mathsf{beh}(\mathsf{C_S}[\mathsf{P}_k])\})$

- $\mathscr{P}' = \mathcal{P}rg$, $\mathscr{G}' = \mathcal{C}tx$,

  $(\!|\cdot|\!) : \mathcal{C}tx \times \mathcal{P}rg \to \wp(\mathit{Trace_T}^k) = \lambda(\mathbf{C_T}, (\mathbf{Q_1}, \mathbf{Q_2}, \ldots, \mathbf{Q_k})).$
  $\mathit{Cl}_{\subseteq}(\{\mathbf{beh}(\mathbf{C_T}[\mathbf{Q_1}]) \times \mathbf{beh}(\mathbf{C_T}[\mathbf{Q_2}]) \times \ldots \times \mathbf{beh}(\mathbf{C_T}[\mathbf{Q_1}])\})$

- $\downarrow : \mathcal{P}rg^k \to \mathcal{P}rg^k = \cdot\!\downarrow^k$

and obtain the followings are equivalent:

$$\begin{aligned}\mathsf{RCP}^{\alpha} \equiv\ & \forall \mathsf{P}_1\mathsf{P}_2\ldots\mathsf{P}_k\ \forall H_S \in \mathit{Closed}_{\subseteq}(\wp(\mathsf{Trace_S})^k).\\ & (\forall \mathsf{C_S}.\ \mathit{Cl}_{\subseteq}(\{\mathtt{beh}(\mathsf{C_S}[\mathsf{P}_1]) \times \mathtt{beh}(\mathsf{C_S}[\mathsf{P}_2]) \times \ldots \times \mathtt{beh}(\mathsf{C_S}[\mathsf{P}_k])\}) \subseteq \pi_S) \Rightarrow\\ & (\forall \mathbf{C_T}.\ \mathit{Cl}_{\subseteq}(\{\mathtt{beh}(\mathbf{C_T}[\mathsf{P}_1\!\downarrow]) \times \mathtt{beh}(\mathbf{C_T}[\mathsf{P}_2\!\downarrow]) \times \ldots \times \mathtt{beh}(\mathbf{C_T}[\mathsf{P}_k\!\downarrow])\}) \subseteq \mathit{Cl}_{\subseteq} \circ \tau(H_S))\end{aligned}$$

$$\begin{aligned}\mathsf{RAP}^{\alpha} \equiv\ & \forall \mathsf{P}_1\mathsf{P}_2\ldots\mathsf{P}_k\ \forall \pi_T \in \mathit{Closed}_{\subseteq}(\wp(\mathbf{Trace_T})^k).\\ & (\forall \mathsf{C_S}.\ \mathit{Cl}_{\subseteq} \circ \tau(\mathit{Cl}_{\subseteq}\{\mathtt{beh}(\mathsf{C_S}[\mathsf{P}_1]) \times \mathtt{beh}(\mathsf{C_S}[\mathsf{P}_2]) \times \ldots \times \mathtt{beh}(\mathsf{C_S}[\mathsf{P}_k])\}) \subseteq H_T) \Rightarrow\\ & (\forall \mathbf{C_T}.\ \mathit{Cl}_{\subseteq}(\{\mathtt{beh}(\mathbf{C_T}[\mathsf{P}_1\!\downarrow]) \times \mathtt{beh}(\mathbf{C_T}[\mathsf{P}_2\!\downarrow]) \times \ldots \times \mathtt{beh}(\mathbf{C_T}[\mathsf{P}_k\!\downarrow]) \subseteq H_T\}))\end{aligned}$$

$$\begin{aligned}\mathsf{RAP}^{\gamma} \equiv\ & \forall \mathsf{P}_1\mathsf{P}_2\ldots\mathsf{P}_k\ \forall \pi_T \in \mathit{Closed}_{\subseteq}(\wp(\mathbf{Trace_T})^k).\\ & (\forall \mathsf{C_S}.\ \mathit{Cl}_{\subseteq}(\{\mathtt{beh}(\mathsf{C_S}[\mathsf{P}_1]) \times \mathtt{beh}(\mathsf{C_S}[\mathsf{P}_2]) \times \ldots \times \mathtt{beh}(\mathsf{C_S}[\mathsf{P}_k]) \subseteq \sigma(H_T)\})) \Rightarrow\\ & (\forall \mathbf{C_T}.\ \mathit{Cl}_{\subseteq}(\{\mathtt{beh}(\mathbf{C_T}[\mathsf{P}_1\!\downarrow]) \times \mathtt{beh}(\mathbf{C_T}[\mathsf{P}_2\!\downarrow]) \times \ldots \times \mathtt{beh}(\mathbf{C_T}[\mathsf{P}_k\!\downarrow]) \subseteq H_T\}))\end{aligned}$$

$$\begin{aligned}\mathsf{RF}^{\gamma}_{\alpha} \equiv\ & \forall \mathsf{P}_1, \mathsf{P}_2, \ldots \mathsf{P}_k\ \forall \mathbf{C_T}.\\ & \mathit{Cl}_{\subseteq}(\{\mathtt{beh}(\mathbf{C_T}[\mathsf{P}_1\!\downarrow]) \times \mathtt{beh}(\mathbf{C_T}[\mathsf{P}_1\!\downarrow]) \times \ldots \times \mathtt{beh}(\mathbf{C_T}[\mathsf{P}_k\!\downarrow])\}) \subseteq\\ & \mathit{Cl}_{\subseteq} \circ \tau(\{\mathtt{beh}(\mathsf{C_S}[\mathsf{P}_1]) \times \mathtt{beh}(\mathsf{C_S}[\mathsf{P}_2]) \times \ldots \times \mathtt{beh}(\mathsf{C_S}[\mathsf{P}_k]) \mid \mathsf{C_S} \in \mathcal{C}tx\}))\end{aligned}$$

With the same argument as in Theorem 20, one can show that $\mathsf{RCP}^{\alpha} \iff k\mathsf{RH}_{\subseteq}\mathsf{P}^{\tau}$, $k\mathsf{RH}_{\subseteq}\mathsf{P}^{\sigma} \iff \mathsf{RAP}^{\gamma}$, $\mathsf{RF}^{\mathit{Cl}_{\subseteq}\circ\sigma}_{\mathit{Cl}_{\subseteq}\circ\tau} \iff \mathsf{RF}^{\gamma}_{\alpha}$ and $k\mathsf{RH}_{\subseteq}\mathsf{P}^{\sim} \iff \mathsf{RF}^{\gamma}_{\alpha}$. □

Theorem 26 (Robust Preservation of Relational Hyperproperties (detailed))**.** *Let* $\tau : \wp(\mathsf{Trace_S}) \leftrightarrows \wp(\mathbf{Trace_T}) : \sigma$ *a Galois connection between trace properties, and let* $\sim \subseteq \mathit{Trace}_S \times \mathit{Trace}_T$ *be its corresponding the relation (see Definition 13). For any* $k \in \mathbb{N} \cup \{\omega\}$*, for simplicity denote still by*

$$\tau : \wp(\mathit{Trace}_S{}^k) \leftrightarrows \wp(\mathit{Trace}_T{}^k) : \check{\sigma}$$

the Galois connection between source and target relational hyperproperties (of the same arity), obtained by lifting (see Lemma 11) the relational Galois connection for $\tau : \wp(\mathsf{Trace_S}) \leftrightarrows \wp(\mathbf{Trace_T}) : \sigma$ *from Lemma 12.*

The followings are equivalent:

$$
\begin{aligned}
k\mathsf{RHP}^{\tau} \equiv\ & \forall \mathsf{P}_1, \mathsf{P}_2, \ldots \mathsf{P}_k\ \forall H_S \in \wp(\mathsf{Trace_S})^k. \\
& (\forall \mathsf{C_S}.\ \mathsf{C_S}[\mathsf{P}_1], \mathsf{C_S}[\mathsf{P}_2], \ldots, \mathsf{C_S}[\mathsf{P}_k] \models H_S) \Rightarrow \\
& (\forall \mathbf{C_T}.\ \mathbf{C_T}[\mathsf{P}_1\!\downarrow], \mathbf{C_T}[\mathsf{P}_2\!\downarrow], \ldots, \mathbf{C_T}[\mathsf{P}_k\!\downarrow] \models Cl_{\subseteq} \circ \tau(H_S))
\end{aligned}
$$

$$
\begin{aligned}
k\mathsf{RH}_{\subseteq}\mathsf{P}^{\sigma} \equiv\ & \forall \mathsf{P}_1, \mathsf{P}_2, \ldots \mathsf{P}_k\ \forall H_T \in \wp(\mathbf{Trace_T})^k. \\
& (\forall \mathsf{C_S}.\ \mathsf{C_S}[\mathsf{P}_1], \mathsf{C_S}[\mathsf{P}_2], \ldots, \mathsf{C_S}[\mathsf{P}_k] \models \check{\sigma}(H_T)) \Rightarrow \\
& (\forall \mathbf{C_T}.\ \mathbf{C_T}[\mathsf{P}_1\!\downarrow], \mathbf{C_T}[\mathsf{P}_2\!\downarrow], \ldots, \mathbf{C_T}[\mathsf{P}_k\!\downarrow] \models H_T)
\end{aligned}
$$

$$
\begin{aligned}
k\mathsf{RF}^{\check{\sigma}}_{\tau} \equiv\ & \forall \mathsf{P}_1, \mathsf{P}_2, \ldots \mathsf{P}_k\ \forall \mathbf{C_T}. \\
& \mathit{beh}(\mathbf{C_T}[\mathsf{P}_1\!\downarrow]) \times \mathit{beh}(\mathbf{C_T}[\mathsf{P}_1\!\downarrow]) \times \ldots \times \mathit{beh}(\mathbf{C_T}[\mathsf{P}_k\!\downarrow]) \in \\
& \tau(\{\mathit{beh}(\mathsf{C_S}[\mathsf{P}_1]) \times \mathit{beh}(\mathsf{C_S}[\mathsf{P}_2]) \times \ldots \times \mathit{beh}(\mathsf{C_S}[\mathsf{P}_k]) \ \mid\ \mathsf{C_S} \in \mathcal{C}tx\})
\end{aligned}
$$

$$
\begin{aligned}
k\mathsf{RHC}^{\sim} \equiv\ & \forall \mathsf{P}_1, \mathsf{P}_2, \ldots \mathsf{P}_k\ \forall \mathbf{C_T}\ \exists \mathsf{C_S}. \\
& (\forall i = 1 \ldots k.\ \mathit{beh}(\mathbf{C_T}[\mathsf{P}_i\!\downarrow]) = \tau(\mathit{beh}(\mathsf{C_S}[\mathsf{P}])))
\end{aligned}
$$

Proof. Apply Lemma 9 with:

- $(C, \sqsubseteq) = (\wp(\mathit{Trace_S}^k), \subseteq), (A, \leq) = (\wp(\mathit{Trace_T}^k), \subseteq)$
- the Galois connection $\tau : \wp(\mathit{Trace_S}^k) \leftrightarrows \wp(\mathit{Trace_S}^k) : \check{\sigma}$
- $\mathscr{P} = \mathcal{P}rg^k$, $\mathscr{G} = \mathcal{C}tx$,

  $[\![\cdot]\!] : \mathcal{C}tx \times \mathcal{P}rg^k \to \wp(\mathit{Trace_S}^k) = \lambda(\mathsf{C_S}, (\mathsf{P}_1, \mathsf{P}_2, \ldots, \mathsf{P}_k)).$
  $\{\mathtt{beh}(\mathsf{C_S}[\mathsf{P}_1]) \times \mathtt{beh}(\mathsf{C_S}[\mathsf{P}_2]) \times \ldots \times \mathtt{beh}(\mathsf{C_S}[\mathsf{P}_k])\}$

- $\mathscr{P}' = \mathcal{P}rg$, $\mathscr{G}' = \mathcal{C}tx$,

  $(\!|\cdot|\!) : \mathcal{C}tx \times \mathcal{P}rg \to \wp(\mathit{Trace_T}^k) = \lambda(\mathbf{C_T}, (\mathbf{Q_1}, \mathbf{Q_2}, \ldots, \mathbf{Q_k})).$
  $\{\mathtt{beh}(\mathbf{C_T}[\mathbf{Q_1}]) \times \mathtt{beh}(\mathbf{C_T}[\mathbf{Q_2}]) \times \ldots \times \mathtt{beh}(\mathbf{C_T}[\mathbf{Q_1}])\}$

- $\downarrow : \mathcal{P}rg^k \to \mathcal{P}rg^k = \cdot\!\downarrow^k$

and obtain the followings are equivalent:

$$\begin{aligned}\mathsf{RCP}^{\alpha} \equiv\ & \forall \mathsf{P}_1\mathsf{P}_2\ldots\mathsf{P}_k\ \forall H_S \in \wp(\mathsf{Trace}_{\mathsf{S}})^k.\\ & (\forall \mathsf{C}_{\mathsf{S}}.\ \{\mathtt{beh}(\mathsf{C}_{\mathsf{S}}[\mathsf{P}_1]) \times \mathtt{beh}(\mathsf{C}_{\mathsf{S}}[\mathsf{P}_2]) \times \ldots \times \mathtt{beh}(\mathsf{C}_{\mathsf{S}}[\mathsf{P}_k])\} \subseteq \pi_S) \Rightarrow\\ & (\forall \mathbf{C_T}.\ \{\mathtt{beh}(\mathbf{C_T}[\mathsf{P}_1{\downarrow}]) \times \mathtt{beh}(\mathbf{C_T}[\mathsf{P}_2{\downarrow}]) \times \ldots \times \mathtt{beh}(\mathbf{C_T}[\mathsf{P}_k{\downarrow}])\} \subseteq \tau(H_S))\end{aligned}$$

$$\begin{aligned}\mathsf{RAP}^{\alpha} \equiv\ & \forall \mathsf{P}_1\mathsf{P}_2\ldots\mathsf{P}_k\ \forall \pi_T \in \wp(\mathbf{Trace_T})^k.\\ & (\forall \mathsf{C}_{\mathsf{S}}.\ \tau(\mathit{Cl}_{\subseteq}\{\mathtt{beh}(\mathsf{C}_{\mathsf{S}}[\mathsf{P}_1]) \times \mathtt{beh}(\mathsf{C}_{\mathsf{S}}[\mathsf{P}_2]) \times \ldots \times \mathtt{beh}(\mathsf{C}_{\mathsf{S}}[\mathsf{P}_k])\} \subseteq H_T) \Rightarrow\\ & (\forall \mathbf{C_T}.\ \{\mathtt{beh}(\mathbf{C_T}[\mathsf{P}_1{\downarrow}]) \times \mathtt{beh}(\mathbf{C_T}[\mathsf{P}_2{\downarrow}]) \times \ldots \times \mathtt{beh}(\mathbf{C_T}[\mathsf{P}_k{\downarrow}]) \subseteq H_T\})\end{aligned}$$

$$\begin{aligned}\mathsf{RAP}^{\gamma} \equiv\ & \forall \mathsf{P}_1\mathsf{P}_2\ldots\mathsf{P}_k\ \forall \pi_T \in \wp(\mathbf{Trace_T})^k.\\ & (\forall \mathsf{C}_{\mathsf{S}}.\ \{\mathtt{beh}(\mathsf{C}_{\mathsf{S}}[\mathsf{P}_1]) \times \mathtt{beh}(\mathsf{C}_{\mathsf{S}}[\mathsf{P}_2]) \times \ldots \times \mathtt{beh}(\mathsf{C}_{\mathsf{S}}[\mathsf{P}_k]) \subseteq \check{\sigma}(H_T)\}) \Rightarrow\\ & (\forall \mathbf{C_T}.\ \{\mathtt{beh}(\mathbf{C_T}[\mathsf{P}_1{\downarrow}]) \times \mathtt{beh}(\mathbf{C_T}[\mathsf{P}_2{\downarrow}]) \times \ldots \times \mathtt{beh}(\mathbf{C_T}[\mathsf{P}_k{\downarrow}]) \subseteq H_T\})\end{aligned}$$

$$\begin{aligned}\mathsf{RF}^{\gamma}_{\alpha} \equiv\ & \forall \mathsf{P}_1, \mathsf{P}_2, \ldots \mathsf{P}_k\ \forall \mathbf{C_T}.\\ & \{\mathtt{beh}(\mathbf{C_T}[\mathsf{P}_1{\downarrow}]) \times \mathtt{beh}(\mathbf{C_T}[\mathsf{P}_1{\downarrow}]) \times \ldots \times \mathtt{beh}(\mathbf{C_T}[\mathsf{P}_k{\downarrow}])\} \subseteq\\ & \tau(\{\mathtt{beh}(\mathsf{C}_{\mathsf{S}}[\mathsf{P}_1]) \times \mathtt{beh}(\mathsf{C}_{\mathsf{S}}[\mathsf{P}_2]) \times \ldots \times \mathtt{beh}(\mathsf{C}_{\mathsf{S}}[\mathsf{P}_k]) \mid \mathsf{C}_{\mathsf{S}} \in \mathcal{C}tx\})\end{aligned}$$

With the same argument as in Theorem 21, one can show that $\mathsf{RCP}^{\alpha} \iff k\mathsf{RHP}^{\tau}$, $k\mathsf{RHP}^{\check{\sigma}} \iff \mathsf{RAP}^{\gamma}$, $\mathsf{RF}^{\check{\sigma}}_{\tau} \iff \mathsf{RF}^{\gamma}_{\alpha}$ and $k\mathsf{RHC}^{\sim} \iff \mathsf{RF}^{\gamma}_{\alpha}$. □

Appendix B

Some Collapses in the Atlas of Secure Compilation

In this appendix we show how features of the languages involved may simplify the hierarchy of secure compilation criteria from Section 5.4.

B.1 Full Reflection

We consider a Source language that is *reflexive*, i.e., it allows to reason about itself [111]. We show that if a source context can access to the code of the programs it is linked with and behave accordingly, then the robust preservation of a class of hyperproperties suffices to the robust preservation of the correspoding calss of relational hyperproperties.

We model the intuition of reflection, by assuming an operator $\otimes : \wp(\mathcal{C}tx) \setminus \{\emptyset\} \to \mathcal{C}tx$ that composes a non empty family of contexts into a single context that can behave as any of the contexts of the family. We may write $\underset{i \in I}{\otimes} C_i$ rather than $\otimes \{C_i \mid i \in I\}$ for the composition of a family of contexts indexed in I.

Assumption 3. We assume that $\otimes : \wp(\mathcal{C}tx) \setminus \{\emptyset\} \to \mathcal{C}tx$ respects the following condition

$$\forall I \neq \emptyset.\ \forall \{C_i\}_{i \in I}\ \forall \{P_i\}_{i \in I}.\ C = \underset{i \in I}{\otimes} C_i \in \mathcal{C}tx \wedge$$
$$\forall i \in I.\ \mathtt{beh}(C[P_i]) = \mathtt{beh}(C_i[P_i])$$

Under Assumption 3 for the Source language, the hierachy from Figure 5.1 reduces to Figure B.1 and that we prove with the lemmas below.

Lemma 20. *Under Assumption 3 for* Source , $\mathsf{RTC}^{\sim} \Rightarrow \mathsf{RH}_{\subseteq}\mathsf{C}^{\sim}$.

Proof. Recall that

$$\mathsf{RTC}^{\sim} \equiv \forall \mathsf{P}\ \forall \mathbf{C_T}\ \forall t.\ \mathbf{C_T}[\mathsf{P}{\downarrow}] \rightsquigarrow t \Rightarrow \exists \mathsf{C_S}.\ \exists\, \mathsf{s} \sim t.\ \mathsf{C_S}[\mathsf{P}] \rightsquigarrow \mathsf{s}$$
$$\mathsf{RH}_{\subseteq}\mathsf{C}^{\sim} \equiv \forall \mathsf{P}\ \forall \mathbf{C_T}\ \exists \mathsf{C_S}.\ \mathtt{beh}(\mathbf{C_T}[\mathsf{P}{\downarrow}]) \subseteq \tau(\mathtt{beh}(\mathsf{C_S}[\mathsf{P}]))$$

For P and $\mathbf{C_T}$, we can compose by Assumption 3, the family of source contexts given by $\mathsf{RTC}^{\sim}$ on any target trace t such that $\mathbf{C_T}[\mathsf{P}{\downarrow}] \rightsquigarrow t$. The context

$$\mathsf{C_S} = \otimes \left\{ \mathsf{C_S}^{(t)} \;\middle|\; t \in \mathit{Trace}_T \wedge \mathbf{C_T}[\mathsf{P}{\downarrow}] \rightsquigarrow t \Rightarrow \exists \mathsf{s} \sim t.\ \mathsf{C_S}^{(t)}[\mathsf{P}] \rightsquigarrow \mathsf{s} \right\}$$

satisfies $\mathsf{RH}_{\subseteq}\mathsf{C}^{\sim}$. □

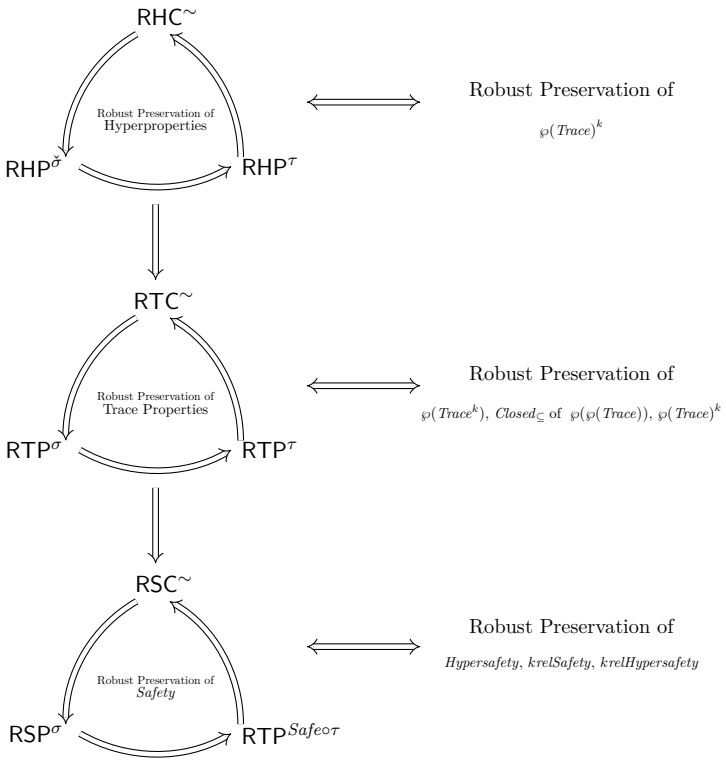

FIGURE B.1: Robust Criteria under Assumption 3 for Source

Lemma 21. *Under Assumption 3 for* Source *, for any* $k \in \mathbb{N} \cup \{\omega\}$, $\mathsf{RTC}^{\sim} \Rightarrow k\mathsf{RTC}^{\sim}$.

Proof. Recall that

$$\begin{aligned}
&\mathsf{RTC}^{\sim} \equiv \forall \mathsf{P}\ \forall \mathbf{C_T}\ \forall t.\ \mathbf{C_T}[\mathsf{P}\downarrow] \rightsquigarrow t \Rightarrow \exists \mathsf{C_S}.\ \exists\, \mathsf{s} \sim t.\ \mathsf{C_S}[\mathsf{P}] \rightsquigarrow \mathsf{s} \\
&k\mathsf{RTC}^{\sim} \equiv \forall \mathsf{P}_1, \mathsf{P}_2, \ldots \mathsf{P}_k\ \forall \mathbf{C_T}\ \forall t_1, t_2, \ldots, t_k. \\
&\quad (\forall i = 1 \ldots k.\ \mathbf{C_T}[\mathsf{P}_i\downarrow] \rightsquigarrow t_i) \Rightarrow \exists \mathsf{C_S}\ \exists \mathsf{s}_1, \mathsf{s}_2, \ldots, \mathsf{s}_k. \\
&\quad (\forall i = 1 \ldots k.\ \mathsf{s}_i \sim t_i \wedge \mathsf{C_S}[\mathsf{P}_i] \rightsquigarrow \mathsf{s}_i)
\end{aligned}$$

For $\mathsf{P}_1, \mathsf{P}_2, \ldots \mathsf{P}_k$ and $\mathbf{C_T}$, we can compose by Assumption 3, the family of source contexts given by $\mathsf{RTC}^{\sim}$. The context

$$\mathsf{C_S} = \otimes \left\{ \mathsf{C_S}^{(i)} \;\middle|\; i = 1 \ldots k \wedge \mathbf{C_T}[\mathsf{P}_i\downarrow] \rightsquigarrow t_i \Rightarrow \exists \mathsf{s}_i \sim t_i.\ \mathsf{C_S}^{(i)}[\mathsf{P}_i] \rightsquigarrow \mathsf{s}_i \right\}$$

satisfies $k\mathsf{RTC}^{\sim}$. □

Lemma 22. *Under Assumption 3 for* Source *, for any* $k \in \mathbb{N} \cup \{\omega\}$, $\mathsf{RTC}^{\sim} \Rightarrow k\mathsf{RH}_{\subseteq}\mathsf{P}^{\sim}$.

Proof. By Lemma 21 it suffices to show that $k\mathsf{RTC}^{\sim} \Rightarrow k\mathsf{RH}_{\subseteq}\mathsf{P}^{\sim}$ that in turn can be shown with the same argument used in Lemma 20, applied to programs $\mathsf{P}_1, \mathsf{P}_2, \ldots, \mathsf{P}_k$, $k \in \mathbb{N} \cup \{\omega\}$. □

Lemma 23. *Under Assumption 3 for* Source *,* $\mathsf{RSC}^{\sim} \Rightarrow \mathsf{RHSC}^{\sim}$.

Proof. Recall that

$$\begin{aligned}\mathsf{RSC}^{\sim} \iff\ & \forall \mathsf{P} \forall \mathbf{C_T} \forall A_T \in \mathcal{O}_T.\ \mathtt{beh}(\mathbf{C_T}[\mathsf{P}\downarrow]) \cap A_T \neq \emptyset \Rightarrow \\ & (\exists \mathsf{C_S}.\ \tau(\mathtt{beh}(\mathsf{C_S}[\mathsf{P}]) \cap A_T \neq \emptyset) \\ \mathsf{RHSC}^{\sim} \iff\ & \forall \mathsf{P} \forall \mathbf{C_T} \forall H_T \in \mathcal{V}_{low}(\mathcal{O}_T).\ \mathtt{beh}(\mathbf{C_T}[\mathsf{P}\downarrow]) \cap H_T \neq \emptyset \Rightarrow \\ & (\exists \mathsf{C_S}.\ \{\tau(\mathtt{beh}(\mathsf{C_S}[\mathsf{P}])\} \cap H_T \neq \emptyset)\end{aligned}$$

and that the topology on hyperproperties have as sub-basic elements (see Definition 1)

$$\langle A_T \rangle = \{\pi_T \in \wp(\mathbf{Trace_T}) \mid \pi_T \cap A_T \neq \emptyset\}$$

for $A_T \in \mathcal{O}_T$ open sets in the topology on trace properties.
It follows that $\{\mathtt{beh}(\mathbf{C_T}[\mathsf{P}\downarrow])\} \cap \langle A_T \rangle \neq \emptyset$ is equivalent to $\mathtt{beh}(\mathbf{C_T}[\mathsf{P}\downarrow]) \cap A_T \neq \emptyset$.
For P_1, $\mathbf{C_T}$, and $H_T \in \mathcal{V}_{low}(\mathcal{O}_T)$,

$$H_T = \bigcup_{n \in \mathbb{N}, i_j \in I} \langle \mathbf{A_{i_1}} \rangle \cap \ldots \cap \langle \mathbf{A_{i_n}} \rangle$$

and the context

$$\mathsf{C_S} = \otimes \left\{ \mathsf{C_S}^{(i)} \;\middle|\; i \in I \wedge \mathbf{C_T}[\mathsf{P}\downarrow] \cap \mathbf{A_i} \neq \emptyset \Rightarrow \tau(\mathtt{beh}(\mathsf{C_S}^{(\mathsf{i})}[\mathsf{P}])) \cap \mathbf{A_i} \neq \emptyset \right\}$$

satisfies $\mathsf{RHSC}^{\sim}$. □

Lemma 24. *Under Assumption 3 for* Source *, for any* $k \in \mathbb{N} \cup \{\omega\}$, $\mathsf{RSC}^{\sim} \Rightarrow k\mathsf{RSC}^{\sim}$.

Proof. Recall that

$$\begin{aligned}\mathsf{RSC}^{\sim} \iff\ & \forall \mathsf{P} \forall \mathbf{C_T} \forall A_T \in \mathcal{O}_T.\ \mathtt{beh}(\mathbf{C_T}[\mathsf{P}\downarrow]) \cap A_T \neq \emptyset \Rightarrow \\ & (\exists \mathsf{C_S}.\ \tau(\mathtt{beh}(\mathsf{C_S}[\mathsf{P}]) \cap A_T \neq \emptyset) \\ k\mathsf{RSC}^{\sim} \equiv\ & \forall \mathsf{P}_1, \mathsf{P}_2, \ldots \mathsf{P}_k\ \forall \mathbf{C_T}\ \forall A_T \in {\mathcal{O}_T}^k. \\ & \mathtt{beh}(\mathbf{C_T}[\mathsf{P}_1\downarrow]) \times \mathtt{beh}(\mathbf{C_T}[\mathsf{P}_2\downarrow]) \times \ldots \times \mathtt{beh}(\mathbf{C_T}[\mathsf{P}_k\downarrow]) \cap A_T \neq \emptyset \Rightarrow \\ & \tau(\mathtt{beh}(\mathsf{C_S}[\mathsf{P}_1]) \times \mathtt{beh}(\mathsf{C_S}[\mathsf{P}_2]) \times \ldots \times \mathtt{beh}(\mathsf{C_S}[\mathsf{P}_k])) \cap A_T \neq \emptyset\end{aligned}$$

and that open sets in the product topology are product of open sets in the topology on trace properties (Definition 5), so that for $\mathsf{P}_1, \mathsf{P}_2, \ldots \mathsf{P}_k$, $\mathbf{C_T}$, and $\mathbf{A} = \mathbf{A_1} \times \mathbf{A_2} \times \ldots \mathbf{A_k}$, the context

$$\mathsf{C_S} = \otimes \left\{ \mathsf{C_S}^{(i)} \;\middle|\; i = 1 \ldots k \wedge \mathbf{C_T}[\mathsf{P}_i\downarrow] \cap \mathbf{A_i} \neq \emptyset \Rightarrow \tau(\mathtt{beh}(\mathsf{C_S}^{(\mathsf{i})}[\mathsf{P}_i])) \cap \mathbf{A_i} \neq \emptyset \right\}$$

satisfies $k\mathsf{RSC}^{\sim}$. □

Lemma 25. *Under Assumption 3 for* Source *, for any* $k \in \mathbb{N} \cup \{\omega\}$, $\mathsf{RSC}^{\sim} \Rightarrow k\mathsf{RHSC}^{\sim}$.

Proof. By Lemma 24 it suffices to show that $k\mathsf{RSC}^{\sim} \Rightarrow k\mathsf{RHSC}^{\sim}$ that in turn can be shown with the same argument used in Lemma 23, applied to programs $\mathsf{P}_1, \mathsf{P}_2, \ldots, \mathsf{P}_k$, $k \in \mathbb{N} \cup \{\omega\}$. □

Lemma 26. *Under Assumption 3 for* Source *, for any* $k \in \mathbb{N} \cup \{\omega\}$, $\mathsf{RHC}^{\sim} \Rightarrow k\mathsf{RHC}^{\sim}$.

Proof. The proof follows the same argument as Lemma 21. □

B.2 Internal Nondeterministic Choice

We show hereafter how the hierachy from Figure 5.1 looks like for a Source language in which contexts can be composed in parallel. We assume an operator $\bigoplus : \wp_{fin}(\mathcal{C}tx) \setminus \{\emptyset\} \to \mathcal{C}tx$ that composes a finite and non empty family of contexts into a single context that nondeterministically behaves as one of the contexts of the family. We may write $\bigoplus_{i=1,2,\ldots,n} C_i$ rather than $\bigoplus \{C_i \mid i = 1, 2, \ldots, n\}$ for the composition of a n contexts.

Assumption 4. We assume that $\bigoplus : \wp_{fin}(\mathcal{C}tx) \setminus \{\emptyset\} \to \mathcal{C}tx$ respects the following condition

$$\forall n \in \mathbb{N}.\ \forall C_1, C_2, \ldots, C_n\ \forall P.\ C = \bigoplus_{i=1,2,\ldots,n} C_i \in \mathcal{C}tx \wedge$$
$$\forall i = 1, 2, \ldots, n.\ \mathtt{beh}(C[P]) \supseteq \mathtt{beh}(C_i[P])$$

Under Assumption 4 for the Source language, the hierachy from Figure 5.1 gets simpler as criteria for classes of relational properties (and subset-closed hyperproperties) are a consequence of the corresponding non relational criteria, as long as the arity of the relation is finite.

Lemma 27. *Under Assumption 4 and for any $k \in \mathbb{N}$,*

(i) $\mathsf{RSC}^{\sim} \Rightarrow k\mathsf{RSC}^{\sim}$

(ii) $\mathsf{RTC}^{\sim} \Rightarrow k\mathsf{RTC}^{\sim}$

(iii) $\mathsf{RHSC}^{\sim} \Rightarrow k\mathsf{RHSC}^{\sim}$

(iv) $\mathsf{RH}_{\subseteq}\mathsf{C}^{\sim} \Rightarrow k\mathsf{RH}_{\subseteq}\mathsf{P}^{\sim}$

Proof. The proof follows the same arguments adopted in Section 4.6. □

Appendix C

Correct Compilation in a Probabilistic Setting

In this appendix we investigate correct compilation between probabilistic languages,i.e., languages equipped with primitives that allow programs to access to a (pseudo)random source during their execution. We believe the theory presented hereafter can also be extended to the secure compilation criteria from Section 5.4.

Following Cousot and Monerau [38] (see in particular their Section 3.1), we imagine that *all* the random choices that are performed during the execution of the program itself, have been made by an oracle *before* the execution started. This in turns allows us to compute the probability that a program satisfies a (hyper)property (Definition 26) and to show that compilers that are correct w.r.t. a non probabilistic semantics, preserve the probability of satisfying (hyper)properties (Theorem 27).

C.1 Probability of Satisfaction

We recall hereafter the notion of probabilistic behavior (probabilistic semantics in [38]), that assigns to every program a measurable function from the probability space of the possible random choices to the sets of trace properties. Intuitively, once the random choices ω have been fixed by an oracle, the probabilistic behavior reduces to (non-deterministic) $\mathtt{beh}(\cdot)$ from previous chapters.

Definition 25 (Probabilistic Behavior (Definition 1 in [38]))**.** The probabilistic behavior of W is a measurable function

$$\mathtt{beh}^{\mathbb{P}}(W) : (\Omega, \mathcal{F}, \mu) \rightarrowtail (\wp(\mathit{Trace}), \mathcal{S})$$

where $(\Omega, \mathcal{F}, \mu)$ is a probability space and $(\wp(\mathit{Trace}), \mathcal{S})$ is a measurable space, with $\mathcal{S} \subseteq \wp(\wp(\mathit{Trace}))$ collecting the hyperproperties that can be measured. For simplicity we assume that $\mathcal{S} = \wp(\wp(\mathit{Trace}))$. □

Intuition 1 (Randomness as superposition)**.** As anticipated above, Definition 25 captures the idea that if the random choices ω are known a priori, then a program $\mathtt{beh}^{\mathbb{P}}(W)(\omega)$ can be intended as the non-probabilistic (still non-deterministic) semantics, and W can be regarded as *superposition* of non-probabilistic programs W_ω, one for each random choice ω.

Example 20 (Dependent coin tossing (see also Example 3 by [38]))**.** We imagine a program that tosses a first fair coin x,i.e., head or tail are equi-probable. Then the program tosses a second coin y, that is fair only if $x = 0$, while if $x = 1$ the probability

that $y = 0$ is $\frac{3}{4}$ and the one that $y = 1$ is $1 = \frac{3}{4} = \frac{1}{4}$.

$$
\begin{aligned}
W = \quad & \\
& x = 0 \ \oplus_{1/2} \ x = 1; \\
& \texttt{if } x = 0 \texttt{ then} \\
& \quad y = 0 \ \oplus_{1/2} \ y = 1 \\
& \texttt{else} \\
& \quad y = 0 \ \oplus_{3/4} \ y = 1
\end{aligned}
$$

The possible traces are pairs (x, y), the events ω on which μ is defined formally are pairs of booleans,i.e., $\Omega = \mathbb{B} \times \mathbb{B}$ but we write them $\omega = (x = b_1, y = b_2)$,

$$\mu(x = 0, y = 0) = \frac{1}{2} \cdot \frac{1}{2} = \frac{1}{4}$$

$$\mu(x = 0, y = 1) = \frac{1}{2} \cdot \frac{1}{2} = \frac{1}{4}$$

$$\mu(x = 1, y = 0) = \frac{1}{2} \cdot \frac{3}{4} = \frac{3}{8}$$

$$\mu(x = 1, y = 1) = \frac{1}{2} \cdot \frac{1}{4} = \frac{1}{8}$$

So that $\texttt{beh}^{\mathbb{P}}(W)(x = 0, y = 0) = (0, 0)$ meaning that the random assignments $\omega = (x = 0, y = 0)$ leads to the trace $(0, 0)$. □

Since the probabilistic behaviors of a program are measurable functions, we can compute the probability a program satisfies a trace property or a hyperproperty.

Definition 26 (Probability of Satisfaction (Definition 2 in [38]))**.** The probability that a program W satisfies a hyperproperty H, is the μ-measure of the set of probabilistic choices under which $W \models H$,

$$\mathbb{P}(W \models H) = \mu(\texttt{beh}^{\mathbb{P}}(W)^{-1}(H)) = \mu(\left\{\omega \in \Omega \;\middle|\; \texttt{beh}^{\mathbb{P}}(W)(\omega) \in H\right\}).$$

The probability that W satisfies a trace property π is the probability that W satisfies the hyperproperty $\wp(\pi)$,

$$\mathbb{P}(W \models \pi) = \mu(\texttt{beh}^{\mathbb{P}}(W)^{-1}(\wp(\pi))) = \mu(\left\{\omega \in \Omega \;\middle|\; \texttt{beh}^{\mathbb{P}}(W)(\omega) \subseteq \pi\right\}).$$

We may also write

$$\mathbb{P}(W \models H) = \int_{\Omega} \mathbb{1}_H \circ \texttt{beh}^{\mathbb{P}}(W)(\omega) \; d\mu(\omega)$$

$$\mathbb{P}(W \models \pi) = \int_{\Omega} \mathbb{1}_{\wp(\pi)} \circ \texttt{beh}^{\mathbb{P}}(W)(\omega) \; d\mu(\omega)$$

□

Example 21. Consider the program W and the measure μ from Example 20.

The property below specifies that the two coin tosses give both head or both tail,

$$\pi_E = \{(b_1, b_2) \mid b_1 = b_2\}$$

and therefore

$$\begin{aligned}\mathbb{P}(W \models \pi_E) =& \mu(\mathtt{beh}^{\mathbb{P}}(W)^{-1}\wp(\pi_E)) = \\ & \mu(\{(x=0, y=0), (x=1, y=1)\}) = \\ & \mu(x=0, y=0) + \mu(x=1, y=1) = \\ & \frac{1}{4} + \frac{1}{8} = \frac{3}{8}\end{aligned}$$

The hyperproperty H can be satisfied only by non deterministic systems,

$$H = \{\pi \mid \forall (b_1, b_2)(b_1', b_2') \in \pi.\ b_1 = b_1' \wedge b_2 \neq b_2'\}$$

while for any fixed ω, the program $W(\omega)$ is deterministic, hence

$$\mathbb{P}(W \models H) = \mu(\mathtt{beh}^{\mathbb{P}}(W)^{-1}H) = \mu(\emptyset) = 0$$

□

C.2 Preserving the Probability of Satisfaction

Intuition 1 allows us to immediately lift the notions of correct compilation from Section 4.4 in probabilistic settings. For simplicity we assume the probability space $(\Omega, \mathcal{F}, \mu)$ to be common to Soruce and **Target** languages, that means random ω picked in Soruce can be picked in the **Target** with the same probability and vice versa. Every criterion from Section 4.4 can be expressed in the probabilistic setting requiring it holds for any common random choice,e.g., a $\cdot\downarrow : \mathsf{Source}^{\$} \to \mathbf{Target}^{\$}$ between two probabilistic languages is $\mathsf{CC}^{\sim}$ (see also Remark 11) *iff* for any $\omega \in \Omega$,

$$\forall \mathsf{W}.\ \mathtt{beh}^{\mathbb{P}}(\mathsf{W}\downarrow)(\omega) \subseteq \tau(\mathtt{beh}^{\mathbb{P}}(\mathsf{W})(\omega)).$$

Similarly $\cdot\downarrow$ is $\mathsf{HC}^{\sim}$ *iff* for any $\omega \in \Omega$,

$$\forall \mathsf{W}.\ \mathtt{beh}^{\mathbb{P}}(\mathsf{W}\downarrow)(\omega) = \tau(\mathtt{beh}^{\mathbb{P}}(\mathsf{W})(\omega)).$$

Theorem 3 (Theorem 6) allows one to easily relate the probability that $\mathsf{W}\downarrow$ satisfies target (hyper)properties and the probability that W satisfies source (hyper)properties.

Theorem 27 (Correct compilers preserve $\mathbb{P}(\cdot \models \pi)$)**.** *Let* $\sim \subseteq \mathit{Trace}_S \times \mathit{Trace}_T$ *and* $\tau : \wp(\mathsf{Trace_S}) \leftrightarrows \wp(\mathbf{Trace_T}) : \sigma$ *the corresponding Galois connection (see Definition 13) and* $\tau : \wp(\wp(\mathsf{Trace_S})) \leftrightarrows \wp(\mathbf{Trace_T}) : \breve{\sigma}$ *its lifting to hyperproperties (see Lemma 11). For any* W, $\pi_S \in \wp(\mathsf{Trace_S})$, $\pi_T \in \wp(\mathbf{Trace_T})$, $H_S \in \wp(\wp(\mathsf{Trace_S}))$ *and* $H_T \in \wp(\wp(\mathbf{Trace_T}))$

(i) if $\cdot\downarrow$ *is* $\mathsf{CC}^{\sim}$ *then,*

$$\mathbb{P}(\mathsf{W}\downarrow \models \tau(\pi_S)) \geq \mathbb{P}(\mathsf{W} \models \pi_S) \quad \textit{and} \quad \mathbb{P}(\mathsf{W}\downarrow \models \pi_T) \geq \mathbb{P}(\mathsf{W} \models \sigma(\pi_T))$$

(ii) if $\cdot\downarrow$ *is* $\mathsf{HC}^{\sim}$ *then,*

$$\mathbb{P}(\mathsf{W}\downarrow \models \tau(H_S)) = \mathbb{P}(\mathsf{W} \models H_S) \quad \textit{and} \quad \mathbb{P}(\mathsf{W}\downarrow \models H_T) = \mathbb{P}(\mathsf{W}\downarrow \models \breve{\sigma}(\pi_T))$$

Proof.

(i) We first show that $\mathbb{P}(\mathsf{W}{\downarrow} \models \tau(\pi_S)) \geq \mathbb{P}(\mathsf{W} \models \pi_S)$

$$
\begin{aligned}
&\mathbb{P}(\mathsf{W}{\downarrow} \models \tau(\pi_S)) = && \text{[by definition]}\\
&\mu(\{\omega \in \Omega \mid \mathtt{beh}^{\mathbb{P}}(\mathsf{W}{\downarrow})(\omega) \in \wp(\tau(\pi_S))\}) =\\
&\mu(\{\omega \in \Omega \mid \mathtt{beh}^{\mathbb{P}}(\mathsf{W}{\downarrow})(\omega) \subseteq \tau(\pi_S)\}) \geq && \text{[monotonicity of } \mu\text{]}\\
&\mu(\{\omega \in \Omega \mid \tau(\mathtt{beh}^{\mathbb{P}}(\mathsf{W})(\omega)) \subseteq \tau(\pi_S)\}) \geq && \text{[monotonicity of } \tau, \mu\text{]}\\
&\mu(\{\omega \in \Omega \mid \mathtt{beh}^{\mathbb{P}}(\mathsf{W})(\omega) \subseteq \pi_S\}) = && \text{[by definition]}\\
&\mathbb{P}(\mathsf{W} \models \pi_S)
\end{aligned}
$$

We now show that $\mathbb{P}(\mathsf{W}{\downarrow} \models \pi_T) \geq \mathbb{P}(\mathsf{W} \models \sigma(\pi_T))$

$$
\begin{aligned}
&\mathbb{P}(\mathsf{W}{\downarrow} \models \pi_T) = && \text{[by definition]}\\
&\mu(\{\omega \in \Omega \mid \mathtt{beh}^{\mathbb{P}}(\mathsf{W}{\downarrow})(\omega) \in \wp(\pi_T)\}) =\\
&\mu(\{\omega \in \Omega \mid \mathtt{beh}^{\mathbb{P}}(\mathsf{W}{\downarrow})(\omega) \subseteq \pi_T\}) \geq && \text{[monotonicity of } \mu\text{]}\\
&\mu(\{\omega \in \Omega \mid \tau(\mathtt{beh}^{\mathbb{P}}(\mathsf{W})(\omega)) \subseteq \pi_T\}) = && \text{[Definition 11 for } \tau \leftrightarrows \sigma, \text{monotoniticy of } \mu\text{]}\\
&\mu(\{\omega \in \Omega \mid \mathtt{beh}^{\mathbb{P}}(\mathsf{W})(\omega) \subseteq \sigma(\pi_T)\}) = && \text{[by definition]}\\
&\mathbb{P}(\mathsf{W} \models \sigma(\pi_T))
\end{aligned}
$$

(ii) We first show that $\mathbb{P}(\mathsf{W}{\downarrow} \models \tau(H_S)) = \mathbb{P}(\mathsf{W} \models H_S)$

$$
\begin{aligned}
&\mathbb{P}(\mathsf{W}{\downarrow} \models \tau(H_S)) = && \text{[by definition]}\\
&\mu(\{\omega \in \Omega \mid \mathtt{beh}^{\mathbb{P}}(\mathsf{W}{\downarrow})(\omega) \in \tau(H_S)\}) =\\
&\mu(\{\omega \in \Omega \mid \tau(\mathtt{beh}^{\mathbb{P}}(\mathsf{W})(\omega)) \in \tau(H_S)\})) = && \text{[by definition]}\\
&\mathbb{P}(\mathsf{W} \models H_S)
\end{aligned}
$$

We now show that $\mathbb{P}(\mathsf{W}{\downarrow} \models H_T) = \mathbb{P}(\mathsf{W} \models \breve{\sigma}(H_T))$

$$
\begin{aligned}
&\mathbb{P}(\mathsf{W}{\downarrow} \models \pi_T) = && \text{[by definition]}\\
&\mu(\{\omega \in \Omega \mid \mathtt{beh}^{\mathbb{P}}(\mathsf{W}{\downarrow})(\omega) \in H_T\}) =\\
&\mu(\{\omega \in \Omega \mid \tau(\mathtt{beh}^{\mathbb{P}}(\mathsf{W})(\omega)) \in H_T\}) =\\
&\mu(\{\omega \in \Omega \mid \{\tau(\mathtt{beh}^{\mathbb{P}}(\mathsf{W})(\omega))\} \subseteq H_T\}) = && \text{[Definition 11 for } \tau \leftrightarrows \breve{\sigma}\text{]}\\
&\mu(\{\omega \in \Omega \mid \{\mathtt{beh}^{\mathbb{P}}(\mathsf{W})(\omega)\} \subseteq \breve{\sigma}(H_T)\}) =\\
&\mu(\{\omega \in \Omega \mid \mathtt{beh}^{\mathbb{P}}(\mathsf{W})(\omega) \in \breve{\sigma}(H_T)\}) = && \text{[by definition]}\\
&\mathbb{P}(\mathsf{W} \models \sigma(\pi_T))
\end{aligned}
$$

□

Bibliography

[1] Martín Abadi. "Protection in programming-language translations". In: *Secure Internet programming*. Springer, 1999, pp. 19–34.

[2] Martín Abadi, Bruno Blanchet, and Cédric Fournet. "The applied pi calculus: Mobile values, new names, and secure communication". In: *Journal of the ACM (JACM)* 65.1 (2017), pp. 1–41.

[3] Martın Abadi and Cédric Fournet. "Private authentication". In: *Theoretical Computer Science* 322.3 (2004), pp. 427–476.

[4] Martín Abadi, Cédric Fournet, and Georges Gonthier. "Secure implementation of channel abstractions". In: *Proceedings. Thirteenth Annual IEEE Symposium on Logic in Computer Science (Cat. No. 98CB36226)*. IEEE. 1998, pp. 105–116.

[5] Martın Abadi and Andrew D Gordon. "A calculus for cryptographic protocols: The spi calculus". In: *Information and computation* 148.1 (1999), pp. 1–70.

[6] Carmine Abate, Matteo Busi, and Stelios Tsampas. "Fully Abstract and Robust Compilation". In: *Programming Languages and Systems*. Ed. by Hakjoo Oh. Cham: Springer International Publishing, 2021, pp. 83–101. ISBN: 978-3-030-89051-3.

[7] Carmine Abate et al. "An Extended Account of Trace-relating Compiler Correctness and Secure Compilation". In: *ACM Transactions on Programming Languages and Systems (TOPLAS)* 43.4 (2021), pp. 1–48.

[8] Carmine Abate et al. "Journey beyond full abstraction: Exploring robust property preservation for secure compilation". In: *2019 IEEE 32nd Computer Security Foundations Symposium (CSF)*. IEEE. 2019, pp. 256–25615.

[9] Carmine Abate et al. "Ssprove: A foundational framework for modular cryptographic proofs in coq". In: *2021 IEEE 34th Computer Security Foundations Symposium (CSF)*. IEEE. 2021, pp. 1–15.

[10] Carmine Abate et al. "Trace-relating compiler correctness and secure compilation". In: *European Symposium on Programming*. Springer. 2020, pp. 1–28.

[11] Carmine Abate et al. "When good components go bad: Formally secure compilation despite dynamic compromise". In: *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*. 2018, pp. 1351–1368.

[12] José Bacelar Almeida et al. "Jasmin: High-assurance and high-speed cryptography". In: *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*. 2017, pp. 1807–1823.

[13] José Bacelar Almeida et al. "The last mile: High-assurance and high-speed cryptographic implementations". In: *2020 IEEE Symposium on Security and Privacy (SP)*. IEEE. 2020, pp. 965–982.

[14] Bowen Alpern and Fred B Schneider. "Defining liveness". In: *Information processing letters* 21.4 (1985), pp. 181–185.

[15] Bowen Alpern and Fred B Schneider. "Recognizing safety and liveness". In: *Distributed computing* 2.3 (1987), pp. 117–126.

[16] Danil Annenkov, Jakob Botsch Nielsen, and Bas Spitters. "ConCert: a smart contract certification framework in Coq". In: *Proceedings of the 9th ACM SIGPLAN International Conference on Certified Programs and Proofs*. 2020, pp. 215–228.

[17] Danil Annenkov et al. "Extracting smart contracts tested and verified in Coq". In: *Proceedings of the 10th ACM SIGPLAN International Conference on Certified Programs and Proofs*. 2021, pp. 105–121.

[18] Timos Antonopoulos et al. "Decomposition instead of self-composition for proving the absence of timing channels". In: *ACM SIGPLAN Notices* 52.6 (2017), pp. 362–375.

[19] Gilles Barthe, Juan Manuel Crespo, and César Kunz. "Product programs and relational program logics". In: *Journal of Logical and Algebraic Methods in Programming* 85.5 (2016), pp. 847–859.

[20] Gilles Barthe, Pedro R D'argenio, and Tamara Rezk. "Secure information flow by self-composition". In: *Mathematical Structures in Computer Science* 21.6 (2011), pp. 1207–1252.

[21] Gilles Barthe, Benjamin Grégoire, and Vincent Laporte. "Secure compilation of side-channel countermeasures: the case of cryptographic "constant-time"". In: *2018 IEEE 31st Computer Security Foundations Symposium (CSF)*. IEEE. 2018, pp. 328–343.

[22] Gilles Barthe et al. "Formal verification of a constant-time preserving C compiler". In: *Proceedings of the ACM on Programming Languages* 4.POPL (2019), pp. 1–30.

[23] Massimo Bartoletti and Roberto Zunino. "BitML: a calculus for Bitcoin smart contracts". In: *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*. 2018, pp. 83–100.

[24] Bruno Blanchet. "Automatic proof of strong secrecy for security protocols". In: *IEEE Symposium on Security and Privacy, 2004. Proceedings. 2004*. IEEE. 2004, pp. 86–100.

[25] Marcel Boehme, Cristian Cadar, and Abhik Roychoudhury. "Fuzzing: Challenges and Reflections." In: *IEEE Softw.* 38.3 (2021), pp. 79–86.

[26] Borzoo Bonakdarpour and Bernd Finkbeiner. "Controller synthesis for hyperproperties". In: *2020 IEEE 33rd Computer Security Foundations Symposium (CSF)*. IEEE. 2020, pp. 366–379.

[27] William J Bowman and Amal Ahmed. "Noninterference for free". In: *ACM SIGPLAN Notices* 50.9 (2015), pp. 101–113.

[28] Sergey Bratus et al. "Exploit programming: From buffer overflows to weird machines and theory of computation". In: *USENIX; login* 36.6 (2011), pp. 13–21.

[29] Matteo Busi. "Secure Compilation All the Way Down". In: (2021).

[30] Matteo Busi, Pierpaolo Degano, and Letterio Galletta. "Translation validation for security properties". In: *arXiv preprint arXiv:1901.05082* (2019).

[31] Matteo Busi et al. "Provably secure isolation for interruptible enclaved execution on small microprocessors". In: *2020 IEEE 33rd Computer Security Foundations Symposium (CSF)*. IEEE. 2020, pp. 262–276.

[32] Vincent Cheval, Véronique Cortier, and Stéphanie Delaune. "Deciding equivalence-based properties using constraint solving". In: *Theoretical Computer Science* 492 (2013), pp. 1–39.

[33] Michael R Clarkson and Fred B Schneider. "Hyperproperties". In: *Journal of Computer Security* 18.6 (2010), pp. 1157–1210.

[34] Norine Coenen et al. "Verifying hyperliveness". In: *International Conference on Computer Aided Verification*. Springer. 2019, pp. 121–139.

[35] Patrick Cousot. "Constructive design of a hierarchy of semantics of a transition system by abstract interpretation". In: *Theoretical Computer Science* 277.1-2 (2002), pp. 47–103.

[36] Patrick Cousot and Radhia Cousot. "Abstract interpretation: a unified lattice model for static analysis of programs by construction or approximation of fixpoints". In: *Proceedings of the 4th ACM SIGACT-SIGPLAN symposium on Principles of programming languages*. 1977, pp. 238–252.

[37] Patrick Cousot and Radhia Cousot. "Abstract interpretation frameworks". In: *Journal of logic and computation* 2.4 (1992), pp. 511–547.

[38] Patrick Cousot and Michael Monerau. "Probabilistic abstract interpretation". In: *European Symposium on Programming*. Springer. 2012, pp. 169–193.

[39] Ole-Johan Dahl, Edsger Wybe Dijkstra, and Charles Antony Richard Hoare. *Structured programming*. Academic Press Ltd., 1972.

[40] Arthur Azevedo De Amorim et al. "Micro-policies: Formally verified, tag-based security monitors". In: *2015 IEEE Symposium on Security and Privacy*. IEEE. 2015, pp. 813–830.

[41] Antoine Delignat-Lavaud et al. "Implementing and proving the TLS 1.3 record layer". In: *2017 IEEE Symposium on Security and Privacy (SP)*. IEEE. 2017, pp. 463–482.

[42] Dominique Devriese, Marco Patrignani, and Frank Piessens. "Fully-abstract compilation by approximate back-translation". In: *Proceedings of the 43rd Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*. 2016, pp. 164–177.

[43] Volker Diekert and Martin Leucker. "Topology, monitorable properties and runtime verification". In: *Theoretical Computer Science* 537 (2014), pp. 29–41.

[44] Danny Dolev and Andrew Yao. "On the security of public key protocols". In: *IEEE Transactions on information theory* 29.2 (1983), pp. 198–208.

[45] Vijay D'Silva, Mathias Payer, and Dawn Song. "The correctness-security gap in compiler optimization". In: *2015 IEEE Security and Privacy Workshops*. IEEE. 2015, pp. 73–87.

[46] Zakir Durumeric et al. "The matter of heartbleed". In: *Proceedings of the 2014 conference on internet measurement conference*. 2014, pp. 475–488.

[47] Pedro R D'Argenio et al. "Is your software on dope?" In: *European Symposium on Programming*. Springer. 2017, pp. 83–110.

[48] Akram El-Korashy et al. "SecurePtrs: Proving Secure Compilation with Data-Flow Back-Translation and Turn-Taking Simulation". In: *arXiv preprint arXiv:2110.01439* (2021).

[49] Joost Engelfriet. "Determinancy→(observation equivalence= trace equivalence)". In: *Theoretical Computer Science* 36 (1985), pp. 21–25.

[50] Matthias Felleisen. “On the expressive power of programming languages”. In: *Science of computer programming* 17.1-3 (1991), pp. 35–75.

[51] Bernd Finkbeiner and Sven Schewe. “Bounded synthesis”. In: *International Journal on Software Tools for Technology Transfer* 15.5 (2013), pp. 519–539.

[52] Bernd Finkbeiner et al. “Monitoring hyperproperties”. In: *International Conference on Runtime Verification*. Springer. 2017, pp. 190–207.

[53] Bernd Finkbeiner et al. “Temporal hyperproperties”. In: *Bulletin of EATCS* 3.123 (2017).

[54] Hubert Garavel, Maurice H Ter Beek, and Jaco Van De Pol. “The 2020 expert survey on formal methods”. In: *International Conference on Formal Methods for Industrial Critical Systems*. Springer. 2020, pp. 3–69.

[55] Paul HB Gardiner, Clare E Martin, and Oege De Moor. “An algebraic construction of predicate transformers”. In: *Science of computer programming* 22.1-2 (1994), pp. 21–44.

[56] Giangiacomo Gerla. “Pointless geometries”. In: *Handbook of incidence geometry*. Elsevier, 1995, pp. 1015–1031.

[57] Roberto Giacobazzi, Neil D Jones, and Isabella Mastroeni. “Obfuscation by partial evaluation of distorted interpreters”. In: *Proceedings of the ACM SIGPLAN 2012 workshop on Partial evaluation and program manipulation*. 2012, pp. 63–72.

[58] Roberto Giacobazzi and Isabella Mastroeni. “Abstract non-interference: a unifying framework for weakening information-flow”. In: *ACM Transactions on Privacy and Security (TOPS)* 21.2 (2018), pp. 1–31.

[59] Roberto Giacobazzi, Isabella Mastroeni, and Mila Dalla Preda. “Maximal incompleteness as obfuscation potency”. In: *Formal Aspects of Computing* 29.1 (2017), pp. 3–31.

[60] Joseph A Goguen and José Meseguer. “Security policies and security models”. In: *1982 IEEE Symposium on Security and Privacy*. IEEE. 1982, pp. 11–11.

[61] Daniele Gorla and Uwe Nestmann. “Full abstraction for expressiveness: history, myths and facts”. In: *Mathematical Structures in Computer Science* 26.4 (2016), pp. 639–654.

[62] Cătălin Hriţcu et al. “Testing noninterference, quickly”. In: *Journal of Functional Programming* 26 (2016).

[63] Sebastian Hunt and Isabella Mastroeni. “The PER model of abstract noninterference”. In: *International Static Analysis Symposium*. Springer. 2005, pp. 171–185.

[64] Chung-Kil Hur and Derek Dreyer. “A Kripke logical relation between ML and assembly”. In: *Proceedings of the 38th annual ACM SIGPLAN-SIGACT symposium on Principles of programming languages*. 2011, pp. 133–146.

[65] Johannes Kinder. “Hypertesting: The case for automated testing of hyperproperties”. In: *3rd Workshop on Hot Issues in Security Principles and Trust (HotSpot)*. Citeseer. 2015.

[66] Bartek Klin. “Bialgebras for structural operational semantics: An introduction”. In: *Theoretical Computer Science* 412.38 (2011), pp. 5043–5069.

[67] Paul Kocher et al. “Spectre attacks: Exploiting speculative execution”. In: *2019 IEEE Symposium on Security and Privacy (SP)*. IEEE. 2019, pp. 1–19.

[68] Máté Kovács, Helmut Seidl, and Bernd Finkbeiner. "Relational abstract interpretation for the verification of 2-hypersafety properties". In: *Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security*. 2013, pp. 211–222.

[69] Orna Kupferman and Moshe Y Vardi. "Robust satisfaction". In: *International Conference on Concurrency Theory*. Springer. 1999, pp. 383–398.

[70] Kazimierz Kuratowski. *Topology: Volume I*. Vol. 1. Elsevier, 2014.

[71] Leslie Lamport. "Logical foundation". In: *Distributed systems-methods and tools for specification* 190 (1985), pp. 119–130.

[72] Leslie Lamport. *Specifying systems*. Vol. 388. Addison-Wesley Boston, 2002.

[73] Leslie Lamport and Fred B Schneider. "The"Hoare Logic"of CSP, and All That". In: *ACM Transactions on Programming Languages and Systems (TOPLAS)* 6.2 (1984), pp. 281–296.

[74] Xavier Leroy. "Formal verification of a realistic compiler". In: *Communications of the ACM* 52.7 (2009), pp. 107–115.

[75] Moritz Lipp et al. "Meltdown: Reading kernel memory from user space". In: *27th {USENIX} Security Symposium ({USENIX} Security 18)*. 2018, pp. 973–990.

[76] Isabella Mastroeni and Michele Pasqua. "Verifying bounded subset-closed hyperproperties". In: *International Static Analysis Symposium*. Springer. 2018, pp. 263–283.

[77] John McLean. "A general theory of composition for trace sets closed under selective interleaving functions". In: *Proceedings of 1994 IEEE Computer Society Symposium on Research in Security and Privacy*. IEEE. 1994, pp. 79–93.

[78] Austin Melton, David A Schmidt, and George E Strecker. "Galois connections and computer science applications". In: *Category Theory and Computer Programming*. Springer, 1986, pp. 299–312.

[79] Ernest Michael. "Topologies on spaces of subsets". In: *Transactions of the American Mathematical Society* 71.1 (1951), pp. 152–182.

[80] John C Mitchell. "On abstraction and the expressive power of programming languages". In: *Science of Computer Programming* 21.2 (1993), pp. 141–163.

[81] F Lockwood Morris. "Advice on structuring compilers and proving them correct". In: *Proceedings of the 1st annual ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages*. 1973, pp. 144–152.

[82] Toby Murray et al. "Compositional verification and refinement of concurrent value-dependent noninterference". In: *2016 IEEE 29th Computer Security Foundations Symposium (CSF)*. IEEE. 2016, pp. 417–431.

[83] Kedar S Namjoshi and Lucas M Tabajara. "Witnessing secure compilation". In: *International Conference on Verification, Model Checking, and Abstract Interpretation*. Springer. 2020, pp. 1–22.

[84] Georg Neis et al. "Pilsner: A compositionally verified compiler for a higher-order imperative language". In: *Proceedings of the 20th ACM SIGPLAN International Conference on Functional Programming*. 2015, pp. 166–178.

[85] Max S New, William J Bowman, and Amal Ahmed. "Fully abstract compilation via universal embedding". In: *Proceedings of the 21st ACM SIGPLAN International Conference on Functional Programming*. 2016, pp. 103–116.

[86] Peter W O'Hearn. "Incorrectness logic". In: *Proceedings of the ACM on Programming Languages* 4.POPL (2019), pp. 1–32.

[87] Aleph One. "Smashing the stack for fun and profit". In: *Phrack magazine* 7.49 (1996), pp. 14–16.

[88] James Painter. "Correctness of a compiler for arithmetic expressions". In: *Proceedings of a Symposium in Applied Mathematics*. Vol. 19. 1967, pp. 33–41.

[89] Joachim Parrow. "General conditions for full abstraction". In: *Mathematical Structures in Computer Science* 26.4 (2016), pp. 655–657.

[90] Michele Pasqua and Isabella Mastroeni. "On Topologies for (Hyper) Properties." In: *ICTCS/CILC*. 2017, pp. 150–161.

[91] Marco Patrignani. "The Tome of Secure Compilation: Fully Abstract Compilation to Protected Modules Architectures". PhD thesis. Leuven, Belgium: KU Leuven, May 2015.

[92] Marco Patrignani. "Why should anyone use colours? or, syntax highlighting beyond code snippets". In: *arXiv preprint arXiv:2001.11334* (2020).

[93] Marco Patrignani, Amal Ahmed, and Dave Clarke. "Formal approaches to secure compilation: A survey of fully abstract compilation and related work". In: *ACM Computing Surveys (CSUR)* 51.6 (2019), pp. 1–36.

[94] Marco Patrignani and Sam Blackshear. "Robust Safety for Move". In: *CoRR* abs/2110.05043 (2021). arXiv: 2110.05043. URL: https://arxiv.org/abs/2110.05043.

[95] Marco Patrignani and Deepak Garg. "Robustly Safe Compilation, an Efficient Form of Secure Compilation". In: *ACM Transactions on Programming Languages and Systems (TOPLAS)* 43.1 (2021), pp. 1–41.

[96] Marco Patrignani, Eric Mark Martin, and Dominique Devriese. "On the semantic expressiveness of recursive types". In: *Proceedings of the ACM on Programming Languages* 5.Popl (2021), pp. 1–29.

[97] Daniel Patterson and Amal Ahmed. "The next 700 compiler correctness theorems". In: *Proceedings of the 24th ACM SIGPLAN International Conference on Functional Programming, ICFP*. Vol. 19.

[98] Jennifer Paykin et al. "Weird Machines as Insecure Compilation". In: *arXiv preprint arXiv:1911.00157* (2019).

[99] Frank Piessens. "Security across abstraction layers: old and new examples". In: *2020 IEEE European Symposium on Security and Privacy Workshops (EuroS&PW)*. IEEE. 2020, pp. 271–279.

[100] Amir Pnueli, Michael Siegel, and Eli Singerman. "Translation validation". In: *International Conference on Tools and Algorithms for the Construction and Analysis of Systems*. Springer. 1998, pp. 151–166.

[101] John Regehr. "A guide to undefined behavior in C and C++, part 3". In: *Embedded in Academia blog* 47 (2010), pp. 57–65.

[102] John C Reynolds. "Separation logic: A logic for shared mutable data structures". In: *Proceedings 17th Annual IEEE Symposium on Logic in Computer Science*. IEEE. 2002, pp. 55–74.

[103] Grigore Rosu. "On safety properties and their monitoring". In: *Scientific Annals of Computer Science* 22.2 (2012), p. 327.

[104] Andrei Sabelfeld and Andrew C Myers. "Language-based information-flow security". In: *IEEE Journal on selected areas in communications* 21.1 (2003), pp. 5–19.

[105] Amr Sabry and Philip Wadler. "A reflection on call-by-value". In: *ACM transactions on programming languages and systems (TOPLAS)* 19.6 (1997), pp. 916–941.

[106] Mooly Sagiv and Francesco Ranzatto. "Xavier Rival". In: (2005).

[107] Fred B Schneider. *On concurrent programming*. Springer Science & Business Media, 2012.

[108] Robert Sison and Toby Murray. "Verifying that a compiler preserves concurrent value-dependent information-flow security". In: *arXiv preprint arXiv:1907.00713* (2019).

[109] Lau Skorstengaard. "An Introduction to Logical Relations". In: *CoRR* abs/1907.11133 (2019). arXiv: 1907.11133. URL: http://arxiv.org/abs/1907.11133.

[110] Lau Skorstengaard, Dominique Devriese, and Lars Birkedal. "StkTokens: Enforcing well-bracketed control flow and stack encapsulation using linear capabilities". In: *Journal of Functional Programming* 31 (2021).

[111] Brian Cantwell Smith. "Reflection and semantics in Lisp". In: *Proceedings of the 11th ACM SIGACT-SIGPLAN symposium on Principles of programming languages*. 1984, pp. 23–35.

[112] Michael B Smyth. "Power domains and predicate transformers: A topological view". In: *International Colloquium on Automata, Languages, and Programming*. Springer. 1983, pp. 662–675.

[113] Youngju Song et al. "CompCertM: CompCert with C-assembly linking and lightweight modular verification". In: *Proceedings of the ACM on Programming Languages* 4.POPL (2019), pp. 1–31.

[114] Gordon Stewart et al. "Compositional compcert". In: *Proceedings of the 42Nd Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*. 2015, pp. 275–287.

[115] Yong Kiam Tan et al. "The verified CakeML compiler backend". In: *Journal of Functional Programming* 29 (2019).

[116] Jérémy Thibault and Cătălin Hriţcu. "Nanopass back-translation of multiple traces for secure compilation proofs". In: *Ret* 3 (2021), p. 5.

[117] Stelios Tsampas et al. "A categorical approach to secure compilation". In: *International Workshop on Coalgebraic Methods in Computer Science*. Springer. 2020, pp. 155–179.

[118] Daniele Turi and Gordon Plotkin. "Towards a mathematical operational semantics". In: *Proceedings of Twelfth Annual IEEE Symposium on Logic in Computer Science*. IEEE. 1997, pp. 280–291.

[119] Steven Vickers. *Topology via logic*. Cambridge University Press, 1996.

[120] Mitchell Wand and Daniel P Friedman. "The mystery of the tower revealed: A nonreflective description of the reflective tower". In: *Lisp and Symbolic Computation* 1.1 (1988), pp. 11–38.

[121] Hiroshi Watanabe. "Well-behaved translations between structural operational semantics". In: *Electronic Notes in Theoretical Computer Science* 65.1 (2002), pp. 337–357.

[122] Glynn Winskel. *The formal semantics of programming languages: an introduction*. MIT press, 1993.

[123] Li-yao Xia et al. "Interaction trees: representing recursive and impure programs in Coq". In: *Proceedings of the ACM on Programming Languages* 4.POPL (2019), pp. 1–32.

[124] Jianzhou Zhao et al. "Formalizing the LLVM intermediate representation for verified program transformations". In: *Proceedings of the 39th annual ACM SIGPLAN-SIGACT symposium on Principles of programming languages*. 2012, pp. 427–440.

[125] Michael Zhivich and Robert K Cunningham. "The real cost of software errors". In: *IEEE Security & Privacy* 7.2 (2009), pp. 87–90.

[126] Roberto Zunino and Pierpaolo Degano. "Handling exp,×(and timestamps) in protocol analysis". In: *International Conference on Foundations of Software Science and Computation Structures*. Springer. 2006, pp. 413–427.