(Co)inductive Proof Systems for Compositional Proofs in Reachability Logic

Vlad Rusu, David Nowak

(Co)inductive Proof Systems for Compositional Proofs in Reachability Logic*

Vlad Rusu

Inria, Lille, France

David Nowak

*CRIStAL**, Lille, France*

Abstract

Reachability Logic is a formalism that can be used, among others, for expressing partial-correctness properties of transition systems. In this paper we present three proof systems for this formalism, all of which are sound and complete and inherit the coinductive nature of the logic. The proof systems differ, however, in several aspects. First, they use induction and coinduction in different proportions. The second aspect regards compositionality, broadly meaning their ability to prove simpler formulas on smaller systems and to reuse those formulas as lemmas for proving more complex formulas on larger systems. The third aspect is the difficulty of their soundness proofs.

We show that the more induction a proof system uses, and the more specialised is its use of coinduction (with respect to our problem domain), the more compositional the proof system is, but the more difficult is its soundness proof.

We present formalisations of these results in the Coq proof assistant. In particular we have developed support for coinductive proofs that is comparable to that provided by Coq for inductive proofs. This may be of interest to a broader class of Coq users.

1. Introduction

Reachability Logic (RL) [1] has been introduced as a language-parametric program logic: a formalism for specifying the functional correctness of programs, which may belong to any programming language whose operational semantics is also specified in RL. The functional correctness of a program is stated as the validity of a set of RL formulas (specifying the program's properties) with respect to another set of RL formulas (specifying the operational semantics of the language containing the program).

Such statements are proved by means of a proof system, which has adequate meta-properties with respect to validity: soundness (only valid RL formulas can be proved)

*This work was partially funded by the CNRS-JSPS Joint Research Project "FoRmal tools for IoT sEcurity" (PRC2199).

**Univ. Lille, CNRS, Centrale Lille, UMR 9189 - CRIStAL - Centre de Recherche en Informatique Signal et Automatique de Lille, F-59000 Lille, France

and relative completeness (all valid RL formulas can, in principle, be proved, modulo the existence of "oracles" for auxiliary tasks). The proofs of these meta-properties are typically highly nontrivial, but for a given proof system they only need to be done once.

Program logics already have a half-century history between them, from the first occurrence of Hoare logic [2] to contemporary separation logics [3]. However, all those logics depend on a language's syntax and therefore have to be defined over and over again, for each new language (or even, for each new language version). In particular, the meta-properties of the corresponding proof systems should be reproved over and over again, a tedious task that is often postponed to an indeterminate future.

Despite being language-parametric, Reachability Logic does not come in only one version. Several versions have been proposed over the years [1, 4, 5]. The formalism has been generalised from programming languages to more abstract models: rewriting logic [6, 7] and transition systems [8], which can be used for specifying designs, and verifying them before they are implemented in program code. This does not replace code verification, just as code verification does not replace the testing of the final running software; but it enables the early catching of errors and the early discovery of key functional-correctness properties, all of which are known to have practical benefits.

Contributions. We further study RL on transition systems (TS). We propose three proof systems for RL, formalise them in Coq [9], and illustrate them on examples.

- the proof systems we propose have some common features: the soundness and completeness meta-properties, and the coinductive nature inherited from RL. However, they differ in other aspects: (i) the "amount" of induction they contain; (ii) their degree of compositionality (i.e., their ability to prove local formulas on "components" of a TS, and then to use those formulas as lemmas in proofs of global formulas on the TS); and (iii) the difficulty of their soundness proofs.

- we show that the more induction a proof system uses, and the closest its "brand" of coinduction to our problem domain (proving RL formulas), the more compositional the proof system is, but the more difficult its soundness proof. There is a winner: the most compositional proof system of the three, but we believe that the other ones exhibit interesting, worth-presenting features as well.

- the implementation of the three proof systems in Coq use different facets of coinduction available in the proof assistant. The basic, builtin ones are enough for the first (and simplest) of the three proof systems. For the second proof system, which uses a mixture of induction and coinduction, more advanced techniques are needed. The third and most complex proof system has a customised coinduction mechanism for our problem domain, implemented in an inductive setting.

- moreover, efforts have been made to develop the Coq formal proofs as close as possible to the paper proofs. In particular, coinductive proofs in Coq are well-formed corecursive programs, while the paper proofs are based on the Knaster-Tarski fixpoint theorems. To reconcile the two we encode instances of that theorem in Coq, whose statements are derived from the corresponding Coq coinductive definition, and whose proofs use the builtin, lower-level Coq mechanisms for

corecursive programming. Then, in subsequent proofs by coinduction we use the higher-level, Knaster-Tarski theorem instances instead of the lower-level, builtin mechanisms. We obtain a framework for proofs by coinduction that offers a level of support comparable (and dual) to that provided by Coq for proofs by induction. We also show that by using the proposed approach one can write proofs that are accepted by Coq, while direct proof attempts using Coq's builtin mechanisms are rejected. The approach is systematic, amenable to automation, and may be of interest to a broad class of Coq users.

Comparison with the conference version. The proof systems and their soundness/-completeness results (minus most of the longer, more technical proofs) have been published in the conference version [10]. The proofs are here developed in full. The Coq formalisation of the proof systems was only sketched in [10], and the Knaster-Tarski framework for proofs by coinduction in Coq is new. We present them here in detail, together with design choices and examples of application. Finally, in the conference version we also briefly presented the formalisation of the first two proof systems in the Isabelle/HOL proof assistant [11]. For space reasons we do not elaborate that part, but we do acknowledge the influence of Knaster-Tarski-based coinduction in Isabelle/HOL on our corresponding developments in Coq.

Comparison with related work. Most papers about Reachability Logic, including the ones cited above, mention the coinductive nature of the logic, but do not actually use it (nor in the paper proof, neither in Coq formalisations). In [12, 13] coinduction is used for formalising RL and for proving RL properties for programs and for term-rewriting systems, but that approach is not mechanised in a proof assistant. More closely related work to ours is reported in [14]; they attack, however, the problem in the opposite way: they develop a general theory of coinduction in Coq and use it to verify programs directly based on the semantics of programming languages, i.e., without using a proof system. They show that a proof system for RL is an instance of their approach for theoretical reasons - in order to show that their approach is complete in a formal sense.

Coinduction in Coq is based on the Curry-Howard isomorphism that views proofs as programs, hence, coinductive proofs are well-formed corecursive programs [15]. Coq offers a limited amount of support for performing coinductive proofs, much less so than for inductive ones. The basic coinduction mechanism (the `cofix` tactic) introduces a hypothesis that copies the current goal's conclusion, and that can only be used when *progress* has been achieved in the proof; technically speaking, the proof term being built has to be *syntactically guarded by constructors.* The exact theoretical definition of syntactical guardedness is complex [15], which does not make its use particularly easy in paper proofs. In recent work [16] we were faced with this problem when proving on paper the soundness and completeness of the first proof system also shown in this paper. To understand those proofs, readers had to "believe" that the syntactical guardedness holds, a complex hypothesis that we did not check in the paper proofs, but only in Coq. In the present paper we bring the Coq proofs closer to the paper ones (via the Knaster-Tarski theorem) rather than the other way around. The *implementation* of syntactical guardedness is also not very user-friendly; our framework based on Knaster-Tarski

alleviates this problem by "hiding" syntactical guardedness and enables some proofs that would not be possible by directly using the builtin `cofix`.

Related works directed at circumventing the syntactical guardedness coinduction, in Coq and other dependent type-theoretical assistants, include [17, 18, 19, 20, 21, 22]. None are based on the Knaster-Tarski fixpoint theorems, in contrast to the implementation of coinduction in the Isabelle/HOL proof assistant [23] that inspired us here.

Regarding coinduction in general, the book [24] serves as an introduction to the topic and explores the relationships between coinduction and bisimulation.

Regarding compositional verification, most existing techniques decompose proofs among parallel composition. Various compositional methods for various parallel composition operators (rely-guarantee, assumption-commitment, ...) are presented in the book [25]. We employ compositionality in a different sense - structural, for transition systems, and logical, for formulas. Many of the techniques presented in [25] have an implicit coinductive nature, which could perhaps be made explicit in future works.

Organisation. The next section recaps preliminary notions: Knaster-Tarski style induction and coinduction, transition systems, and RL on transition systems. A first compositionality result, of RL-validity with respect to certain sub-transition systems, is given. The three following sections present our three proof systems in increasing order of complexity. Soundness and completeness results are given and a notion of compositionality with respect to formulas, in two versions: asymmetrical and symmetrical, is introduced and combined with the compositionality regarding sub-transition systems. The three proof systems are shown to have increasingly demanding compositionality features. We then present the Coq mechanisations of the proof systems, and illustrate on examples how they can be used for compositional verification. The Coq formalisations are currently available at `http://project.inria.fr/jlamp2019`.

2. Preliminaries

2.1. Induction and Coinduction

Consider a complete lattice $(L, \sqsubseteq, \sqcup, \sqcap, \bot, \top)$ and a monotone function $F : L \to L$. According to the Knaster-Tarski fixpoint theorem, F has a least fixpoint μF (respectively, greatest fixpoint νF), which is the least (respectively, greatest) element x of L such that $F(x) \sqsubseteq x$ (respectively, $x \sqsubseteq F(x)$), hence, $F(x) \sqsubseteq x$ implies $\mu F \sqsubseteq x$, and $x \sqsubseteq F(x)$ implies $x \sqsubseteq \nu F$. We shall sometimes use a stronger version of the statement involving the greatest fixpoint: $X \sqsubseteq F(X \sqcup \nu F)$ iff $X \sqsubseteq \nu F$.

Those theorems can be used to define inductive and coinductive datatypes and recursive and corecursive functions. For example, the type of natural numbers is defined as the least fixpoint of the function $F(X) = \{0\} \cup \{Suc(x) \mid x \in X\}$. The greatest fixpoint of F is the type of natural numbers with infinity.

As another example, let $\mathcal{S} = (S, \to)$ be a transition system where S is the set of states and $\to \subseteq S \times S$ is the transition relation. A state s is *final*, and we write $\bullet s$, if there exists no s' such that $s \to s'$. A path is a nonempty, possibly infinite sequence of states. More formally, the set *Paths* of paths is the greatest fixpoint νF, where $F(X) = \{s \mid \bullet s\} \cup \{s\,\tau \mid s \in S \wedge \tau \in X \wedge s \to (hd\,\tau)\}$, with $hd : Paths \to S$

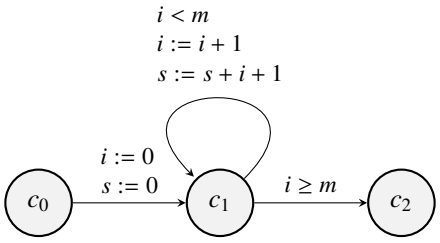

Figure 1: Sum up to m

being simultaneously defined as $hd(s) = s$ and $hd(s\,\tau) = s$ for all $s \in S$ and $\tau \in X$. One can then corecursively define the length of a path as a value in the natural numbers with infinity: $len\ s = 0$ and $len(s\,\tau) = Suc(len\,\tau)$.

Hereafter, whenever necessary, we emphasise the fact that certain notions are relative to a transition system $\mathcal{S}$ by postfixing them with $_{\mathcal{S}}$. We omit this subscript when it can be inferred from the context.

A complete lattice associated to a transition system $\mathcal{S} = (S, \rightarrow)$, is the set of state predicates Π defined as the set of functions from S to the set of Booleans $\mathbb{B} = \{\text{F}, \text{T}\}$. Its operations are defined by $p \sqsubseteq q \triangleq \forall s, p\ s \Rightarrow q\ s$, $(p \sqcup q)\ s \triangleq\ p\ s \vee q\ s$, $(p \sqcap q)\ s \triangleq p\ s \wedge q\ s$, $\bot\ s \triangleq \text{F}$, $\top\ s \triangleq \text{T}$. We also extend the transition relation $\rightarrow$ of $\mathcal{S}$ into a *symbolic transition function* $\partial : \Pi \rightarrow \Pi$, defined by $\partial p \triangleq \lambda s\ .\ \exists s'\ .\ p\ s' \wedge s' \rightarrow s$.

2.2. Reachability Formulas

We adapt Reachability Logic to transition systems. Assume a transition system $\mathcal{S} = (S, \rightarrow)$. Syntactically, a reachability formula (or, simply, a formula) over $\mathcal{S}$ is a pair $p \Rightarrow\!\!\diamond\ q$ with $p, q \in \Pi$. We let $lhs(p \Rightarrow\!\!\diamond\ q) \triangleq p$ and $rhs(p \Rightarrow\!\!\diamond\ q) \triangleq q$. We denote by $\Phi_{\mathcal{S}}$ the set of all reachability formulas over the transition system $\mathcal{S}$.

Example 1. *Figure 1 depicts an extended finite-state machine having three natural-number variables: i,s, and m, and three control nodes: c_0, c_1, and c_2. Arrows connect the nodes and are possibly decorated with a Boolean guard and a set of parallel assignments of the variables. The variable m is never assigned, thus, it stays constant. The purpose of the machine is to compute in s the sum of the first m natural numbers.*

The machine is also a finite representation of an infinite-state transition system whose set of states is the Cartesian product $\{c_0, c_1, c_2\} \times \mathbb{N} \times \mathbb{N} \times \mathbb{N}$. and whose transition relation is $\bigcup_{i,s,m \in \mathbb{N}} \{((c_0, i, s, m), (c_1, 0, 0, m))\} \cup \bigcup_{i,s,m \in \mathbb{N}, i<m} \{((c_1, i, s, m), (c_1, i + 1, s + i + 1, m))\} \cup \bigcup_{i,s,m \in \mathbb{N}, i \geq m} \{((c_1, i, s, m), (c_2, i, s, m))\}$. A formula expressing the transition system's functional correctness is $(c = c_0) \Rightarrow\!\!\diamond\ (c = c_2 \wedge s = m \times (m + 1)/2)$.

Note that the predicate expressions appearing in examples are abbreviations. For instance, the expression $c = c_2 \wedge s = m \times (m+1)/2$ in the above example is an abbreviation for the obvious function that maps a state (c, i, s, m) to a boolean.

To define the semantics of reachability formulas we introduce the following relation.

Definition 1. *$\rightsquigarrow$ is the largest set of pairs $(\tau, r) \in Paths \times \Pi$ such that: (i) $\tau = s$ for some $s \in S$, and $r\,s$; or (ii) $\tau = s\,\tau'$, for some $s \in S$, $\tau' \in Paths$, and $r\,s$; or (iii) $\tau = s\,\tau'$ for some $s \in S$, $\tau' \in Paths$, and $(\tau', r) \in \rightsquigarrow$.*

We write $\tau \rightsquigarrow r$ for $(\tau, r) \in \rightsquigarrow$. The instance of the Knaster-Tarski theorem for $\rightsquigarrow$ is:

Lemma 1. *For $R \subseteq Paths \times \Pi$, if for all $(\tau, r) \in R$, it holds that either $(\exists s.\tau = s \wedge r\,s)$, or $(\exists s.\exists \tau'.\tau = s\,\tau' \wedge r\,s)$ or $(\exists s.\exists \tau'.\tau = s\,\tau' \wedge (\tau', r) \in R)$, then $R \subseteq \rightsquigarrow$.*

Proof. Consider the function $F : \mathcal{P}(Paths \times \Pi) \to \mathcal{P}(Paths \times \Pi)$ defined by:

$F(X) = \{(\tau, r) \mid \exists (\tau', r') \in X.\, (\tau = \tau' = s \wedge r\,s) \vee (\tau = s\,\tau' \wedge r\,s) \vee (\tau = s\,\tau' \wedge \tau' \rightsquigarrow r)\}$

Then, F is monotone and by Knaster-Tarski's theorem, it has a greatest fixpoint νF, which coincides with the relation $\rightsquigarrow$. The theorem also says that for any $R \subseteq Paths \times \Pi$, if $R \subseteq F(R)$ then $R \subseteq \rightsquigarrow$. Now, let R be the relation in our lemma's hypotheses, and note that the hypothesis in question "*for all $(\tau, r) \in R$, it holds that either $(\exists s.\tau = s \wedge r\,s)$ or $(\exists s.\exists \tau'.\tau = s\,\tau' \wedge r\,s)$ or $(\exists s.\exists \tau'.\tau = s\,\tau' \wedge (\tau', r) \in R)$*" is just the expansion of the inclusion $R \subseteq F(R)$ for F defined as above. Hence the conclusion $R \subseteq \rightsquigarrow$. □

Definition 2 (Validity). *A formula $\varphi \in \Phi_{\mathcal{S}}$ is* valid *over $\mathcal{S}$, denoted by $\mathcal{S} \models \varphi$, whenever for all $\tau \in Paths_{\mathcal{S}}$ such that $(lhs\,\varphi)\,(hd\,\tau)$ holds, it also holds that $\tau \rightsquigarrow_{\mathcal{S}} (rhs\,\varphi)$.*

Example 2. *The formula $(c = c_0) \Rightarrow\!\Diamond (c = c_2 \wedge s = m \times (m+1)/2)$ is valid over the transition system denoted by the state-machine depicted in Figure 1. Intuitively, this means that all finite paths "starting" in the control node c_0 "eventually reach" c_2 with $s = m \times (m+1)/2$ holding. The "eventually reach" expression justifies the $\Rightarrow\!\Diamond$ notation borrowed from Linear Temporal Logic (LTL). Indeed, reachability formulas are essentially LTL formulas for a version of LTL* interpreted over finite paths *[26].*

The following alternative characterisation of validity will sometimes be useful. We hereafter denote by $fPaths_{\mathcal{S}} \subseteq Paths_{\mathcal{S}}$ the subset of finite paths of transition system $\mathcal{S}$.

Lemma 2. *$\mathcal{S} \models \varphi$ iff for all $\tau \in fPaths_{\mathcal{S}}$, $(lhs\,\varphi)(hd\,\tau)$ implies $(rhs\,\varphi)(\tau\,n)$ for some $n \leq len\,\tau$.*

Proof. We first make the following observations. If τ is finite, $\tau \rightsquigarrow_{\mathcal{S}} r$ is equivalent to the existence of $n \leq len\,\tau$ such that $r\,(\tau\ n)$. This is proved, in one direction, by induction on the finiteness property of the sequence τ, and in the other direction, on the natural number n. By contrast, if τ is infinite, $\tau \rightsquigarrow_{\mathcal{S}} r$ holds for any r, because, by item (iii) of Definition 1, $\tau \rightsquigarrow r$ can be reduced to $\tau' \rightsquigarrow_{\mathcal{S}} r$ where τ' is the "tail" of τ, and then $\tau' \rightsquigarrow_{\mathcal{S}} r$ can be reduced to $\tau'' \rightsquigarrow_{\mathcal{S}} r$ where τ'' is the "tail" of τ', and so on, *ad infinitum*. The lemma follows from the above observations and from Definition 2. □

Lemma 3 (Additional properties of validity). *For all predicates $l, l', l_1, l_2, m, r \in \Pi_{\mathcal{S}}$:*

- *(trivial) : $\mathcal{S} \models r \Rightarrow\!\Diamond r$;*
- *(strengthening) : $l \sqsubseteq l'$ and $\mathcal{S} \models l' \Rightarrow\!\Diamond r$ imply $\models l \Rightarrow\!\Diamond r$;*
- *(splitting) : $\mathcal{S} \models l_1 \Rightarrow\!\Diamond r$ and $\mathcal{S} \models l_2 \Rightarrow\!\Diamond r$ imply $\mathcal{S} \models (l_1 \sqcup l_2) \Rightarrow\!\Diamond r$;*

- *(transitivity) :* $\mathcal{S} \models l \Rightarrow\!\!\diamond\, m$ *and* $\mathcal{S} \models m \Rightarrow\!\!\diamond\, r$ *imply* $\mathcal{S} \models l \Rightarrow\!\!\diamond\, r$*;*
- *(step) :* $\mathcal{S} \models \partial l \Rightarrow\!\!\diamond\, r$ *and* $l \sqcap \bullet \sqsubseteq \bot$ *imply* $\mathcal{S} \models l \Rightarrow\!\!\diamond\, r$*.*

Proof. For all items except (transitivity) we use Definition 2 of validity; for (transitivity) it is more convenient to use the alternative characterisation (Lemma 2).

- Consider any path τ such that $(lhs(r \Rightarrow\!\!\diamond\, r))\,(hd\,\tau)$. Then $(rhs(r \Rightarrow\!\!\diamond\, r))\,(hd\,\tau)$, thus, by Definition 1, $\tau \rightsquigarrow r$. The conclusion $\mathcal{S} \models r \Rightarrow\!\!\diamond\, r$ follows by Definition 2.
- Consider any path τ such that $(lhs(l \Rightarrow\!\!\diamond\, r))\,(hd\,\tau)$, i.e., $l\,(hd\,\tau)$. From $l \sqsubseteq l'$ we obtain $l'\,(hd\,\tau)$, i.e., $(lhs(l' \Rightarrow\!\!\diamond\, r))\,(hd\,\tau)$. From $\mathcal{S} \models l' \Rightarrow\!\!\diamond\, r$ we obtain by Definition 2 that $\tau \rightsquigarrow r$, and $\mathcal{S} \models l \Rightarrow\!\!\diamond\, r$ follows from the same definition.
- Consider any path τ such that $(lhs((l_1 \sqcup l_2) \Rightarrow\!\!\diamond\, r))\,(hd\,\tau)$, i.e., $(l_1 \sqcup l_2)\,(hd\,\tau)$. Hence, $l_1\,(hd\,\tau)$ or $l_2\,(hd\,\tau)$. We consider the first case, the other one is symmetrical. From $l_1\,(hd\,\tau)$ and $\mathcal{S} \models l_1 \Rightarrow\!\!\diamond\, r$ we obtain $r \rightsquigarrow r$, and the conclusion $\mathcal{S} \models ((l_1 \sqcup l_2) \Rightarrow\!\!\diamond\, r)$ follows by from Definition 2.
- Consider any *finite* path τ such that $(lhs(l \Rightarrow\!\!\diamond\, r))\,(hd\,\tau)$, i.e., $l\,(hd\,\tau)$. Hence, $(lhs(l \Rightarrow\!\!\diamond\, m))\,(hd\,\tau)$, and from $\mathcal{S} \models l \Rightarrow\!\!\diamond\, m$, using Lemma 2 we obtain $k \leq len\,\tau$ such that $m\,(\tau\,k)$. Let τ' be the suffix of τ starting at $\tau\,k$. Then, τ' is a finite path, and $m\,(\tau\,k)$ means $m\,(hd\,\tau')$, i.e., $(lhs(m \Rightarrow\!\!\diamond\, r))\,(hd\,\tau')$, which, together with $\mathcal{S} \models m \Rightarrow\!\!\diamond\, r$ and Lemma 2 gives us $k' \leq len\,\tau'$ such that $r\,(\tau'\,k')$. Let $k'' = k + k'$, hence, $k'' \leq len\,\tau$, and the conclusion $\mathcal{S} \models l \Rightarrow\!\!\diamond\, r$ follows by Lemma 2.
- Consider any path τ such that $(lhs(l \Rightarrow\!\!\diamond\, r))\,(hd\,\tau)$, i.e., $l\,(hd\,\tau)$. Assume first that $\tau = s$ for some $s \in S$. Since τ is a path, s is final, i.e., $\bullet\, s$, which, together with $l\,s$ gives $(l \sqcap \bullet)\,s$, in contradiction with the hypothesis $l \sqcap \bullet \sqsubseteq \bot$. Hence, $\tau = s\,\tau'$ for some $s \in S$ and $\tau' \in Paths$, with $s \rightarrow (hd\,\tau')$. We now show $(\partial l)\,(hd\,\tau')$. Indeed, by the definition of the ∂ function, the statement $(\partial l)\,(hd\,\tau')$ amounts to the existence of some state $s' \in S$ such that $l\,s'$ and $s' \rightarrow (hd\,\tau')$; from what we obtained above, taking $s' = s$ satisfies this. Hence, $lhs(\partial l \Rightarrow\!\!\diamond\, r)\,(hd\,\tau')$ and from $\mathcal{S} \models \partial l \Rightarrow\!\!\diamond\, r$ we obtain by Definition 2 that $\tau' \rightsquigarrow r$, which, by Definition 1, implies $\tau \rightsquigarrow r$. The conclusion $\mathcal{S} \models l \Rightarrow\!\!\diamond\, r$ follows by Definition 2.

□

3. Structural Compositionality

We define a notion of *component* of a transition system, and show that, if a formula is valid on a component, then it is valid on the whole transition system. This simplifies verification since components are typically smaller than whole transition systems.

Definition 3 (Component). *A transition system* $(S', \rightarrow')$ *is a* component *of* $(S, \rightarrow)$ *if*

- $S' \subseteq S$ *and* $\rightarrow' \subseteq \rightarrow$*;*
- *for all* $s', s \in S'$*,* $s' \rightarrow s$ *implies* $s' \rightarrow' s$*;*

- *for all* $s' \in S'$, $s \in S \setminus S'$, $s' \to s$ *implies* $s' \in \bullet_{\mathcal{S}'}$.

We write $\mathcal{S}' \lhd \mathcal{S}$ *when* $\mathcal{S}'$ *is a component of* $\mathcal{S}$.

We often interchangeably use sets of states and their characteristic predicates, like we did for $\bullet_{\mathcal{S}'}$ above. Informally, $\mathcal{S}' \lhd \mathcal{S}$ means that $\mathcal{S}'$ is a full sub-transition system of $\mathcal{S}$, and one may only "exit" from $\mathcal{S}'$ via its final states. In particular, the third condition is implied by the first one when $(S, \to)$ is deterministic.

Example 3. *In Figure 1, the self-loop on the control node* c_1 *denotes a transition system* $\mathcal{S}'$*, which is a component of the transition system* $\mathcal{S}$ *denoted by the whole state machine. The state-space of* $\mathcal{S}'$ *is a subset of that of* $\mathcal{S}$ *since the control nodes occurring in the former are fewer than those occurring in the latter, and the state variables are the same.* $\mathcal{S}'$ *has fewer transitions than* $\mathcal{S}$*, and any transition of* $\mathcal{S}$ *between states of* $\mathcal{S}'$ *(i.e., those denoted by the self-loop on* c_1*) is also a transition of* $\mathcal{S}'$*, thus,* $\mathcal{S}'$ *is a full sub-transition system of* $\mathcal{S}$*.* $\mathcal{S}$ *is deterministic, which implies the third condition.*

One could also consider a class of hierarchical state machines, where the "component-of" relation holds by construction, but that is more a matter of component-based design than of verification; we do not explore that direction further in this paper.

Theorem 1 (Structural Compositionality of $\models$). $\mathcal{S}' \lhd \mathcal{S}$ *and* $\mathcal{S}' \models \varphi$ *imply* $\mathcal{S} \models \varphi$.

Proof. Let $\mathcal{S} = (S, \to)$, $\mathcal{S}' = (S', \to')$, $\varphi = l \Rightarrow\!\Diamond\, r$. Note that the hypothesis $\mathcal{S}' \models l \Rightarrow\!\Diamond\, r$ implies $l, r \in \Pi_{\mathcal{S}'}$, which by $\mathcal{S}' \lhd \mathcal{S}$ also implies $l, r \in \Pi_{\mathcal{S}}$. Let $\tau \in \mathit{fPaths}_{\mathcal{S}}$ be arbitrarily chosen such that $l\,(\mathit{hd}\,\tau)$. Since $l \in \Pi_{\mathcal{S}'}$, he have $(\mathit{hd}\,\tau) \in S'$. Hence, the set T'_τ of prefixes of τ is nonempty.

Since all sequences in T'_τ are finite, there is one τ'_m with maximal length $k = \mathit{len}\,\tau'_m$. Let $s_m = \tau'_m\, k$. Since $\tau'_m \in T'_\tau$, in order to show $\tau'_m \in \mathit{Paths}_{\mathcal{S}'}$ we only need (†): $s_m \in \bullet_{\mathcal{S}'}$. There are two cases:

- if $k = \mathit{len}\,\tau$ then s_m is the last state on $\tau \in \mathit{fPaths}_{\mathcal{S}}$, hence, $s_m \in \bullet_{\mathcal{S}} \subseteq \bullet_{\mathcal{S}'}$;
- if $k < \mathit{len}\,\tau$ then $\tau\,(k+1) \in S \setminus S'$, otherwise, from $s_m \in S'$ and $s_m \to \tau\,(k+1)$ and hypotheses one has $s_m \to' \tau\,(k+1)$, hence, $\tau'_m \to' \tau\,(k+1) \in T'_\tau$, in contradiction with the maximal length of τ'_m in T'_τ. From $\tau\,(k+1) \in S \setminus S'$ and $s_m \to \tau\,(k+1)$ we obtain from the lemma's hypotheses that $s' \in \bullet_{\mathcal{S}'}$.

Hence, (†) is proved, and per the above reasoning, so is $\tau'_m \in \mathit{Paths}_{\mathcal{S}'}$. Observe also that, since $\tau'_m \in T'_\tau$, we have that τ'_m is a prefix of τ, thus, for all $j \le k = \mathit{len}\,\tau'_m$, $\tau'_m\ j = \tau\ j$. In particular, $\mathit{hd}\,\tau'_m = \mathit{hd}\,\tau'$ and since we assumed $l\,(\mathit{hd}\,\tau)$ at the beginning, we also have $l\,(\mathit{hd}\,\tau'_m)$. From the latter and $\tau'_m \in \mathit{Paths}_{\mathcal{S}'}$ and $\mathcal{S}' \models l \Rightarrow\!\Diamond\, r$ and Lemma 2 we obtain $j \le \mathit{len}\,\tau'_m = k \le \mathit{len}\,\tau$ such that $r\,(\tau'_m\ j)$, hence, $r\,(\tau\ j)$.

Recapitulating, we started with $\tau \in \mathit{fPaths}_{\mathcal{S}}$ arbitrarily chosen such that $l\,(\mathit{hd}\,\tau)$, and obtained $j \le \mathit{len}\,\tau$ such that $r\,(\tau\ j)$. By Lemma 2, this means $\mathcal{S}' \models l \Rightarrow\!\Diamond\, r$, which proves the theorem. □

Example 4. *Consider the transition system* $\mathcal{S}$ *and its component* $\mathcal{S}'$ *introduced in Example 3. Let* $\varphi \triangleq (c = c_1 \wedge i = 0 \wedge s = 0) \Rightarrow\!\Diamond\, (c = c_1 \wedge i = m \wedge s = i \times (i+1)/2)$*. One can show that* $\mathcal{S}' \models \varphi$*, thus,* φ *is also valid over* $\mathcal{S}$*.*

$$\text{[Stp]}\ \frac{\mathcal{S} \vdash \partial l' \Rrightarrow_\Diamond r}{\mathcal{S} \vdash l \Rrightarrow_\Diamond r}\nu \ \textit{ if } l \sqsubseteq l' \sqcup r, l' \sqcap \bullet \sqsubseteq \bot$$

Figure 2: One-rule proof system

One could, in principle, prove the validity of reachability formulas directly from the semantical definitions. However, this has several disadvantages: lack of a methodology - each formula is proved in its own ad-hoc way, and lack of a notion of completeness - is there a uniform way for proving every valid formula? These issues are addressed by the proof systems presented by increasing order of complexity in the next sections.

4. A One-Rule Proof System

Our first proof system is depicted as the one-rule inference system in Figure 2. It is parameterised by a transition system $\mathcal{S}$, and everything therein depends on it; we omit $_\mathcal{S}$ subscripts for simplicity. Intuitively, an application of the [Stp] rule can be seen as a symbolic execution step, taking a formula $l \Rrightarrow_\Diamond r$ and "moving" l "one step closer" to r - specifically, taking an over-approximation l' of the "difference" between l and r (encoded in the side-condition $l \sqsubseteq l' \sqcup r$) that contains no final states ($l' \sqcap \bullet \sqsubseteq \bot$) and performing a symbolic execution step from l' (encoded in the ∂ symbolic transition function). The rule is applicable infinitely many times, hence the ν symbol next to it. Note that there are no hypotheses in the proof system: those would be reachability formulas in the left-hand side of the $\vdash$ symbol, not allowed here.

For a more formal definition, consider the function $F : \mathcal{P}(\Phi) \to \mathcal{P}(\Phi)$ defined by

$$F(X) = \bigcup_{l,l' r \in \Pi,\, l \sqsubseteq l' \sqcup r, l' \sqcap \bullet \sqsubseteq \bot,\, \partial l' \Rrightarrow_\Diamond r \in X} \{l \Rrightarrow_\Diamond r\}$$

F is monotone, and, by Knaster-Tarski's theorem, F has a greatest fixpoint νF. We now define $\mathcal{S} \vdash \varphi$ by $\varphi \in \nu F$. The Knaster-Tarski theorem induces the following property:

Lemma 4. *For all set $X \subseteq \Phi$ of hypotheses and $\varphi \in X$, if* for all $l \Rrightarrow_\Diamond r \in X$, there is $l' \in \Pi$ such that $l \sqsubseteq l' \sqcup r$, $l' \sqcap \bullet \sqsubseteq \bot$ and $\partial l' \Rrightarrow_\Diamond r \in X$, *then $\mathcal{S} \vdash \varphi$.*

Proof. Choose an arbitrary $X \subseteq \varphi$. The hypothesis "*for all $l \Rrightarrow_\Diamond r \in X$, there is $l' \in \Pi$ such that $l \sqsubseteq l' \sqcup r$, $l' \sqcap \bullet \sqsubseteq \bot$ and $\partial l' \Rrightarrow_\Diamond r \in X$* " is the expansion of $X \subseteq F(X)$ with F defined as above. By Knaster-Tarski's theorem, $X \subseteq \nu F$. By the above definition of $\vdash$ we obtain that for all $\varphi \in X$, it holds that $\mathcal{S} \vdash \varphi$, which proves the lemma. □

Soundness. Soundness (Theorem 2 below) means that only valid formulas are proved. Its proof uses the instance of Knaster-Tarski's theorem for the $\rightsquigarrow$ relation (Lemma 1), which occurs in the definition of validity, instantiated with the relation $R \subseteq \mathit{Paths} \times \Pi$ defined by $R \triangleq \lambda(\tau, r).\exists l.(\mathcal{S} \vdash l \Rrightarrow_\Diamond r \wedge l\,(hd\tau))$. As a general observation, all proofs by coinduction use a specific instance of Knaster-Tarski's theorem, instantiated with a specific predicate/relation. The instantiation is where the user's creativity is involved. In Section 7 we show users can figure out from the shape of Coq subgoals certain

instantiations that are (in some sense) adequate for proving a given inductive statement.

Theorem 2 (Soundness of $\vdash$). *$\mathcal{S} \vdash \varphi$ implies $\mathcal{S} \models \varphi$.*

Proof. We first prove the following fact (†): the relation $R \subseteq \mathit{Paths} \times \Pi$ defined as follows: $R \triangleq \lambda(\tau, r).\exists l.(\mathcal{S} \vdash l \Rightarrow\!\!\diamond\, r \wedge l\,(\mathit{hd}\tau))$ satisfies $R \subseteq \rightsquigarrow$. We use Lemma 1 for this purpose. The lemma requires us to prove that, assuming $(\tau, r) \in R$, it holds that (i) $(\exists s.\tau = s \wedge r\,s)$ or (ii) $(\exists s.\exists\tau'.\tau = s\,\tau' \wedge r\,s)$ or (iii) $(\exists s.\exists\tau'.\tau = s\,\tau' \wedge (\tau', r) \in R)$. We proceed by case analysis:

- first, assume $\tau = s$ for some $s \in S$. Since $(\tau, r) \in R$, there is $l \in \Pi$ such that $\mathcal{S} \vdash l \Rightarrow\!\!\diamond\, r$ and $l\,s$. Now, $\mathcal{S} \vdash l \Rightarrow\!\!\diamond\, r$ implies that there is $l' \in \Pi$ with $l \sqsubseteq l' \sqcup r$, $l' \sqcap \bullet \sqsubseteq \bot$, and $\mathcal{S} \vdash \partial l' \Rightarrow\!\!\diamond\, r$. Next, $l\,s$ and $l \sqsubseteq l' \sqcup r$ imply $l'\,s$ or $r\,s$. Assume first $l'\,s$. Since τ is the (singleton) path s, the state s is final, hence, $\bullet\,s$, which together with $l'\,s$ contradict $l' \sqcap \bullet \sqsubseteq \bot$. Hence, $l'\,s$ is impossible, and therefore $r\,s$; then, (i) is proved in this case. Note that here we did not use the fact $\mathcal{S} \vdash \partial l' \Rightarrow\!\!\diamond\, r$ - it will be used below.

- then, assume $\tau = s\,\tau'$, with $s \in S$ and $\tau' \in \mathit{Paths}$. We show $(\tau', r) \in R$. From $(\tau, r) \in R$ we obtain $l \in \Pi$ such that $\mathcal{S} \vdash l \Rightarrow\!\!\diamond\, r$ and $l\,s$. Now, $\mathcal{S} \vdash l \Rightarrow\!\!\diamond\, r$ implies, by definition of our proof system, that there is $l' \in \Pi$ with $l \sqsubseteq l' \sqcup r$, $l' \sqcap \bullet \sqsubseteq \bot$, and $\mathcal{S} \vdash \partial l' \Rightarrow\!\!\diamond\, r$. From $l\,s$ and $l \sqsubseteq l' \sqcup r$ we obtain $l'\,s$ or $r\,s$.

  - if $r\,s$ then (ii) holds;
  - if $l'\,s$, we prove $(\partial l')\,(\mathit{hd}\,\tau')$. Since τ is a path, then so is τ', and we have the transition $s \to (\mathit{hd}\,\tau')$. Since $\partial l' = \lambda s'.\exists s.l'\,s \wedge s \to s'$ we obtain that $(\partial l')(\mathit{hd}\,\tau')$. The existence of $\partial l'$ such that $\mathcal{S} \vdash \partial l' \Rightarrow\!\!\diamond\, r$ and $(\partial l')(\mathit{hd}\,\tau')$ ensures $(\tau', r) \in R$, hence, (iii) holds. Note also that we did not use $l' \sqcap \bullet \sqsubseteq \bot$ here, but this inclusion was used in an earlier case.

Summarising, in all possible cases, $(\tau, r) \in R$ implies either statements (i), (ii), or (iii) from Lemma 1. Hence, the lemma ensures $R \subseteq \rightsquigarrow$, and (†) is proved. Coming back to our theorem: consider an arbitrary $\varphi \triangleq l' \Rightarrow\!\!\diamond\, r \in \Phi$ such that $\mathcal{S} \vdash l' \Rightarrow\!\!\diamond\, r$. In order to show $\mathcal{S} \models l' \Rightarrow\!\!\diamond\, r$, i.e., the conclusion of the theorem, we only need to show that for all paths τ such that $l'\,(\mathit{hd}\,\tau)$, it holds that $\tau \rightsquigarrow r$. We do this by showing $(\tau, r) \in R$ and using $R \subseteq \rightsquigarrow$ from above. We have defined $R = \lambda(\tau, r).\exists l.(\mathcal{S} \vdash l \Rightarrow\!\!\diamond\, r \wedge l\,(\mathit{hd}\tau))$ and for our (τ, r) there does indeed exist $l \triangleq l'$ such that $\mathcal{S} \vdash l \Rightarrow\!\!\diamond\, r$ (hypothesis of the theorem) and $l\,(\mathit{hd}\,\tau)$ (from above). Hence $(\tau, r) \in R$, which concludes the proof. □

Completeness. Completeness is the reciprocal to soundness: any valid formula is provable. It is based on the following lemma, which essentially reduces reachability to a form of inductive invariance. Its proof uses the instance of Knaster-Tarski's theorem for $\vdash$, i.e., Lemma 4, with an appropriate instantiation of the set X therein.

Lemma 5. *If $l \sqsubseteq q \sqcup r$, $q \sqcap \bullet \sqsubseteq \bot$, and $\partial q \sqsubseteq q \sqcup r$ then $\mathcal{S} \vdash l \Rightarrow\!\!\diamond\, r$.*

Proof. Consider the set $X = \{l' \Rightarrow\!\!\diamond\, r \mid l' \sqsubseteq q \sqcup r \ \wedge\ q \sqcap \bullet \sqsubseteq \bot \ \wedge\ \partial q \sqsubseteq q \sqcup r\}$. We show that the premise of Lemma 4 holds with the chosen X: (†) *for all* $l' \Rightarrow\!\!\diamond\, r \in X$, *there exists* $l'' \in \Pi$ *such that* $l' \sqsubseteq l'' \sqcup r'$ *and* $l'' \sqcap \bullet \sqsubseteq \bot$ *and* $\partial l'' \Rightarrow\!\!\diamond\, r \in X$.

Let then $l' \Rightarrow\!\!\diamond\, r$ be an arbitrary element in X. Hence, (i) $l' \sqsubseteq q \sqcup r'$ and (ii) $q \sqcap \bullet \sqsubseteq \bot$ and (iii) $\partial q \sqsubseteq q \sqcup r$. Moreover, $\partial q \Rightarrow\!\!\diamond\, r \in X$, because of the hypothesis $\partial q \sqsubseteq q \sqcup r$ of our lemma and (ii) and (iii). By choosing in (†) to instantiate the existentially quantified l'' to q, for any formula in X, the (†) statement is proved. Hence, we can apply Lemma 4, and obtain that for all $\varphi \in X$, it holds that $\mathcal{S} \vdash \varphi$. Finally, the formula $l \Rightarrow\!\!\diamond\, r$ in our lemma's conclusion does belong to X since, by the lemma's hypotheses it satisfies all the conditions of membership in X. Hence, $\mathcal{S} \vdash l \Rightarrow\!\!\diamond\, r$, which concludes the proof. □

Example 5. *In order to establish* $\mathcal{S}' \models (c = c_1 \wedge i = 0 \wedge s = 0) \Rightarrow\!\!\diamond\, (c = c_1 \wedge i = m \wedge s = i \times (i+1)/2)$ *- which has been claimed in Example 4 - one can use Lemma 5 with* $q \triangleq (c = c_1 \wedge i < m \wedge s = i \times (i+1)/2)$.

The proof of the completeness of $\vdash$ (below) is constructive: it builds, for valid formulas $l \Rightarrow\!\!\diamond\, r$, a certain predicate q that is shown to satisfy the three inclusions of Lemma 5.

Theorem 3 (Completeness of $\vdash$). $\mathcal{S} \models \varphi$ *implies* $\mathcal{S} \vdash \varphi$.

Proof. Let $\varphi \triangleq l \Rightarrow\!\!\diamond\, r$. We define the state predicate $q \triangleq \lambda s.\, \neg r\, s \wedge \forall \tau \in \mathit{Paths}.(\, s = \mathit{hd}\, \tau \Rightarrow \tau \rightsquigarrow r)$ and show that it satisfies the three inclusions in the hypothesis of Lemma 5, which can then be applied to deduce our theorem's conclusion $\mathcal{S} \vdash \varphi$.

1. $l \sqsubseteq q \sqcup r$: let s be any state such that $l\, s$; we have to prove $(q \sqcup r)\, s$. If $r\, s$ the proof is done. Thus, assume $\neg r\, s$, and consider any path τ such that $s = \mathit{hd}\, \tau$. From $l\, s$ and $s = \mathit{hd}\, \tau$ and $\mathcal{S} \models l \Rightarrow\!\!\diamond\, r$ we obtain by Definition 2, that $\tau \rightsquigarrow r$, and by definition of q we have $q\, s$: the first inclusion is proved.
2. $q \sqcap \bullet \sqsubseteq \bot$: let s be any state such that $q\, s$; we prove that $\bullet\, s$ is impossible. By the above definition of q, $\neg r s$. Consider an arbitrary path τ such that $s = \mathit{hd}\, \tau$; again, by definition of q, $\tau \rightsquigarrow r$. Now, the only way $\tau \rightsquigarrow r$ can hold when $\neg r s$ holds is (cf. Definition 1) when $\tau = s\, \tau'$ for some path τ'. Hence, the arbitrarily chosen s is not final, thus no state satisfies $q \sqcap \bullet$, and our second inclusion is proved.
3. $\partial q \sqsubseteq q \sqcup r$: let s' be a state such that $(\partial q)\, s'$; we have to prove $(q \sqcup r)\, s'$. By the definition of the symbolic transition function ∂, there exists s such that $s \rightarrow s'$ and $q\, s$. By the definition of q, $\neg r\, s$ and for each $\tau \in \mathit{Paths}$ such that $s = \mathit{hd}\, \tau$, it holds that $\tau \rightsquigarrow r$. There are two subcases:
   - if $r\, s'$ then $(q \sqcup r)\, s'$, and our third inclusion is proved;
   - if $\neg r\, s'$: consider any path τ' such that $s' = \mathit{hd}\, \tau'$. Then, the sequence $\tau \triangleq s\, \tau'$ is a path and is such that $s = \mathit{hd}\, \tau$, and, per the above, $\tau \rightsquigarrow r$. We also have $\neg r\, s$, and then the only way $\tau \rightsquigarrow r$ may hold is via $\tau' \rightsquigarrow r$ (cf. Definition 1). Summarising, in the case $\neg r\, s'$, we get that any path τ' such that $s' = \mathit{hd}\, \tau'$ satisfies $\tau' \rightsquigarrow r$. Hence, $q\, s'$ by the definition of q, and therefore also $(q \sqcup r)\, s'$, which completes the proof of the third inclusion and of the theorem.

□

We note that completeness is a very strong statement, but it is based on strong assumptions: that expressive predicates such as q above are available. If a decision procedure for the $\sqsubseteq$ relation between such predicates were one could prove validity directly from the semantics of formulas, without going through a proof system. Hence, completeness is a theoretical property; the practically useful property is Lemma 5, which users have to instantiate in a creative manner. In [16] we use this approach in the verification of a nontrivial example (a model of a security hypervisor).

Looking again at the proof system $\vdash$, we note that it is purely coinductive - no induction is present at all. This is unlike the proof systems in forthcoming sections. Regarding structural compositionality (with respect to transition systems) our proof system has it, since, by soundness and completeness and Theorem 1, one has that $\mathcal{S}' \lhd \mathcal{S}$ and $\mathcal{S}' \vdash \varphi$ implies $\mathcal{S} \vdash \varphi$. However, we show below that $\vdash$ does not have another, equally desirable feature: *logical compositionality*, with respect to formulas.

Logical compositionality (asymmetrical version). A proof system with this feature decomposes a proof of a formula φ into a proof of a formula φ' and one of φ assuming φ'. The asymmetry between the formulas involved suggested the property's name. There is also a symmetrical version, discussed ahead in the paper. In Definition 4 below, $\Vdash$ is a binary relation - a subset of $\mathcal{P}(\Phi) \times \Phi$ (equivalently, a predicate of type $\mathcal{P}(\Phi) \to \Phi \to \mathbb{B}$), parameterised by a transition system $\mathcal{S}$. For *hypotheses* $\mathcal{H} \subseteq \Phi$ and $\varphi \in \Phi$, we write $\mathcal{S}, \mathcal{H} \Vdash \phi$ for $(\mathcal{H}, \phi) \in \Vdash$ and $\mathcal{S} \Vdash \phi$ for $\mathcal{S}, \emptyset \Vdash \phi$.

Definition 4 (Asymmetrical compositionality). *A proof system $\Vdash$ is asymmetrically compositional if $\mathcal{S} \Vdash \varphi'$ and $\mathcal{S}, \{\varphi'\} \Vdash \varphi$ imply $\mathcal{S} \Vdash \varphi$.*

The proof system $\vdash$ does not have this property because that requires hypotheses, which $\vdash$ does not have. One could add hypotheses to it, and a new rule to prove a formula if it is found among the hypotheses. This (and more) is done in the second proof system.

5. An Asymmetrically-Compositional Proof System

In this section we propose another proof system and show that it is both structurally compositional (with respect to transition systems) and asymmetrically compositional (with respect to formulas). This is achieved thanks to the introduction of inductive rules in the proof system, with a better distribution of roles between these rules and the remaining coinductive rule, at the cost of a more involved soundness proof.

Our second proof system is depicted in Figure 3. It is a binary relation - a subset of $\mathcal{P}(\Phi) \times \Phi$ (or equivalently, a binary predicate of type $\mathcal{P}(\Phi) \to \Phi \to \mathbb{B}$), parameterised by a transition system $\mathcal{S}$. Its rules are inspired from the statement of Lemma 3. Intuitively, the rule [Stp], labelled with ν, is coinductive, i.e., it can be applied infinitely many times, and the rules [Hyp], [Trv], [Str], [Spl], and [Tra], labelled by μ are inductive, i.e., they can only be applied finitely many times between two consecutive applications of [Stp]. Stated differently, a proof in $\Vdash$ is a possibly infinite series of *phases*, and in each phase there are finitely many applications of [Hyp], [Trv], [Str], [Spl], and [Tra] and, except in the last phase (if such a last phase exists), one application of [Stp].

$$\text{[Hyp]}\ \frac{}{\mathcal{S},\mathcal{H} \Vdash \varphi}\mu \ \textit{ if } \varphi \in \mathcal{H}$$

$$\text{[Trv]}\ \frac{}{\mathcal{S},\mathcal{H} \Vdash r \Rightarrow\!\!\diamond\, r}\mu$$

$$\text{[Str]}\ \frac{\mathcal{S},\mathcal{H} \Vdash l' \Rightarrow\!\!\diamond\, r}{\mathcal{S},\mathcal{H} \Vdash l \Rightarrow\!\!\diamond\, r}\mu \ \textit{ if } l \sqsubseteq l'$$

$$\text{[Spl]}\ \frac{\mathcal{S},\mathcal{H} \Vdash l_1 \Rightarrow\!\!\diamond\, r \qquad \mathcal{S},\mathcal{H} \Vdash l_2 \Rightarrow\!\!\diamond\, r}{\mathcal{S},\mathcal{H} \Vdash (l_1 \sqcup l_2) \Rightarrow\!\!\diamond\, r}\mu$$

$$\text{[Tra]}\ \frac{\mathcal{S} \models l \Rightarrow\!\!\diamond\, m \qquad \mathcal{S},\mathcal{H} \Vdash m \Rightarrow\!\!\diamond\, r}{\mathcal{S},\mathcal{H} \Vdash l \Rightarrow\!\!\diamond\, r}\mu$$

$$\text{[Stp]}\ \frac{\mathcal{S},\mathcal{H} \Vdash \partial l \Rightarrow\!\!\diamond\, r}{\mathcal{S},\mathcal{H} \Vdash l \Rightarrow\!\!\diamond\, r}\nu \ \textit{ if } l \sqcap \bullet \sqsubseteq \bot$$

Figure 3: Mixed inductive-coinductive proof system

Note that making the inductive rules [Str], [Spl], and [Tra], coinductive would compromise soundness, because any of them could forever reduce a proof of any formula to itself, thus proving any formula, valid or not. The rules [Hyp] and [Trv] are not recursive so it does not matter if they are inductive or coinductive; we choose the former.

The roles of the rules are the following ones. [Hyp] allows one to prove a formula if it is among the hypotheses. [Trv] is in charge of proving trivially valid formulas. [Str] is a general principle that amounts to strengthening a formula before proving it. [Spl] is used for getting rid of disjunctions in left-hand sides of formulas, which occur when several, alternative symbolic behaviours are explored in a proof search. [Tra] is a transitivity rule, used for proving facts about sequential symbolic behaviour. Note also the asymmetry in hypotheses of the rule [Tra]: for one formula validity is required, while for the other one, it is provability. This asymmetry is used to avoid technical difficulties that arise when proving the soundness of $\Vdash$, but, as we shall see, it generates difficulties of its own. Finally, [Stp] makes the connection between the concrete paths and the symbolic ones, which the proof system explores during proof search.

For a formal definition: the following functions from $\mathcal{P}(\Phi)$ to $\mathcal{P}(\Phi)$ give a formal meaning to (top-down) applications of the rules of the proof system to sets of formulas. They are parameterised by a transition system $\mathcal{S}$ and a set of hypotheses $\mathcal{H}$, which are those occurring at the left-hand side of the $\Vdash$ symbol, and an additional set of formulas Y.

- $\vdash^{\text{[Hyp]}}_{\mathcal{S},\mathcal{H},Y}(X) = \mathcal{H}$
- $\vdash^{\text{[Trv]}}_{\mathcal{S},\mathcal{H},Y}(X) = \bigcup_{r\in\Pi}\{r \Rightarrow\!\!\diamond\, r\}$
- $\vdash^{\text{[Str]}}_{\mathcal{S},\mathcal{H},Y}(X) = \bigcup_{l,l',r\in\Pi,\, l' \Rightarrow\!\!\diamond\, r\in X,\, l\sqsubseteq l'}\{l \Rightarrow\!\!\diamond\, r\}$
- $\vdash^{\text{[Spl]}}_{\mathcal{S},\mathcal{H},Y}(X) = \bigcup_{l_1,l_2,r\,\in\Pi,\,\{l_1 \Rightarrow\!\!\diamond\, r,\, l_2 \Rightarrow\!\!\diamond\, r\}\subseteq X}\{(l_1 \sqcup l_2) \Rightarrow\!\!\diamond\, r\}$
- $\vdash^{\text{[Tra]}}_{\mathcal{S},\mathcal{H},Y}(X) = \bigcup_{l,r,m\in\Pi,\,\mathcal{S}\models l \Rightarrow\!\!\diamond\, m,\, m \Rightarrow\!\!\diamond\, r\in X}\{l \Rightarrow\!\!\diamond\, r\}$

- $\vdash^{[\mathsf{Stp}]}_{\mathcal{S},\mathcal{H},Y}(X) = \bigcup_{l,r\in\Pi,\, l\sqcap\bullet\sqsubseteq\bot,\, \partial l \Rightarrow\!\!\diamond\, r\in Y}\{l \Rightarrow\!\!\diamond\, r\}$

Some explanations: the function $\vdash^{[\mathsf{Hyp}]}_{\mathcal{S},\mathcal{H},Y}$ returns, whatever its argument X, the set of formulas $\mathcal{H}$. This expresses the fact that the rule [Hyp] can prove exactly the formulas in the set $\mathcal{H}$ of hypotheses. Similar explanations apply to the remaining functions: given a set X of formulas, they can prove any formula obtained by applying the corresponding rule of the proof system to some formula in X.

Note that the additional set Y of formulas is only used in the function $\vdash^{[\mathsf{Stp}]}_{\mathcal{S},\mathcal{H},Y}$ that corresponds to the rule [Stp]. It enables proofs of formulas $l \Rightarrow\!\!\diamond\, r$, whenever $\partial l \Rightarrow\!\!\diamond\, r \in Y$ (and $l \sqcap \bullet \sqsubseteq \bot$). It is used below for constructing a greatest fixpoint.
Let $\vdash_{\mathcal{S},\mathcal{H},Y}(X) = \vdash^{[\mathsf{Hyp}]}_{\mathcal{S},\mathcal{H},Y}(X) \cup \vdash^{[\mathsf{Trv}]}_{\mathcal{S},\mathcal{H},Y}(X) \cup \vdash^{[\mathsf{Str}]}_{\mathcal{S},\mathcal{H},Y}(X) \cup \vdash^{[\mathsf{Spl}]}_{\mathcal{S},\mathcal{H},Y}(X) \cup \vdash^{[\mathsf{Tra}]}_{\mathcal{S},\mathcal{H},Y}(X) \cup \vdash^{[\mathsf{Stp}]}_{\mathcal{S},\mathcal{H},Y}(X)$.
It is not hard to show that $\vdash_{\mathcal{S},\mathcal{H},Y}: \mathcal{P}(\Phi) \to \mathcal{P}(\Phi)$ is monotonous, thus, by the Knaster-Tarski smallest fixpoint theorem it has a smallest fixpoint, denoted by $\mu\vdash_{\mathcal{S},\mathcal{H},Y}$. Intuitively, $\mu\vdash_{\mathcal{S},\mathcal{H},Y}$ is the set of formulas that can be proved, starting from the empty set $\emptyset$ and by finitely many applications of the rules of the proof system, with a "special permission" given to the rule [Stp]: prove all formulas $l \Rightarrow\!\!\diamond\, r$ if $\partial l \Rightarrow\!\!\diamond\, r \in Y$ and $l \sqcap \bullet \sqsubseteq \bot$. (The "finitely many" above is a consequence of having a smallest fixpoint.) Of course, the parameter Y has to disappear at some point from our definition of the $\Vdash$ proof system. For this, we define the function $F_{\mathcal{S},\mathcal{H}} : \mathcal{P}(\Phi) \to \mathcal{P}(\Phi)$ by $F_{\mathcal{S},\mathcal{H}}(Y) = \mu\vdash_{\mathcal{S},\mathcal{H},Y}$. $F_{\mathcal{S},\mathcal{H}}$ is monotone, thus, it has a greatest fixpoint $\nu F_{\mathcal{S},\mathcal{H}} = \nu(\lambda Y.\mu\vdash_{\mathcal{S},\mathcal{H},Y}) = \nu\mu\vdash_{\mathcal{S},\mathcal{H}}$.

Finally, we define the relation $\Vdash$ as follows : for all $\mathcal{H} \subseteq \Phi$ and $\varphi \in \Phi$, $\mathcal{S}, \mathcal{H} \Vdash \phi$ iff $\varphi \in \nu\mu\vdash_{\mathcal{S},\mathcal{H}}$. The inductive-coinductive nature of $\Vdash$ is visible from its definition.

Lemma 6. *If $Y \subseteq \mu\vdash_{\mathcal{S},\mathcal{H},Y}$ then for all $\varphi \in Y$ it holds that $\mathcal{S}, \mathcal{H} \Vdash \varphi$.*

Proof. The hypothesis $Y \subseteq \mu\vdash_{\mathcal{S},\mathcal{H},Y}$ is $Y \subseteq F_{\mathcal{S},\mathcal{H}}(Y)$, the conclusion is $Y \subseteq \nu F_{\mathcal{S},\mathcal{H}}$, and the lemma follows from the Knaster-Tarski greatest fixpoint theorem applied to $F_{\mathcal{S},\mathcal{H}}$. □

We illustrate the above lemma by proving a key lemma for the completeness of $\Vdash$. Hereafter we use the abbreviation $\mathcal{S} \Vdash \varphi$ for $\mathcal{S}, \emptyset \Vdash \varphi$.

Lemma 7. *If $l \sqsubseteq q \sqcup r$, $q \sqcap \bullet \sqsubseteq \bot$, and $\partial q \sqsubseteq q \sqcup r$ then $\mathcal{S} \Vdash l \Rightarrow\!\!\diamond\, r$.*

Proof. We use Lemma 6 with the given transition system $\mathcal{S}$, $\mathcal{H} = \emptyset$, and the set $Y = \{l \Rightarrow\!\!\diamond\, r, q \sqcup r \Rightarrow\!\!\diamond\, r, r \Rightarrow\!\!\diamond\, r, q \Rightarrow\!\!\diamond\, r, \partial q \Rightarrow\!\!\diamond\, r\}$ that includes the formula $l \Rightarrow\!\!\diamond\, r$ that we want to prove. For this, Lemma 6 requires us to show $Y \subseteq \mu\vdash_{\mathcal{S},\mathcal{H},Y}$. Now, as noted above, $\mu\vdash_{\mathcal{S},\mathcal{H},Y}$ is the set of formulas that can be proved starting from the empty set $\emptyset$ by finitely many applications of the rules of the proof system $\Vdash$, with a "special permission" for the rule [Stp] to prove all formulas $l \Rightarrow\!\!\diamond\, r$ if $\partial l \Rightarrow\!\!\diamond\, r \in Y$ and $l \sqcap \bullet \sqsubseteq \bot$.

- $q \Rightarrow\!\!\diamond\, r$ is proved as just stated above using [Stp], because $\partial q \Rightarrow\!\!\diamond\, r \in Y$ and $q \sqcap \bullet \sqsubseteq \bot$ is a hypothesis of our lemma;
- $r \Rightarrow\!\!\diamond\, r$ is proved directly using [Trv];
- $q \sqcup r \Rightarrow\!\!\diamond\, r$ is proved using [Spl], using $q \Rightarrow\!\!\diamond\, r$ and $r \Rightarrow\!\!\diamond\, r$ proved above;
- $l \Rightarrow\!\!\diamond\, r$ is proved using [Str], using $q \sqcup r \Rightarrow\!\!\diamond\, r$ above and the hypothesis $l \sqsubseteq q \sqcup r$;

- and finally, $\partial q \Rightarrow\!\!\diamond r$ is also proved using [Str], also by using $q \sqcup r \Rightarrow\!\!\diamond r$ proved above and the hypothesis $\partial q \sqsubseteq q \sqcup r$.

The inclusion $Y \subseteq \mu\vdash_{\mathcal{S},\mathcal{H},Y}$ is now proved. By Lemma 6, each formula φ in Y, including $l \Rightarrow\!\!\diamond r$, is such that $\mathcal{S} \Vdash \varphi$, which proves our lemma. □

Completeness. By analogy with Theorem 3, but using Lemma 7 instead of Lemma 5:

Theorem 4 (Completeness of $\Vdash$). *$\mathcal{S} \models \varphi$ implies $\mathcal{S} \Vdash \varphi$.*

Soundness. We define the recursive function $suf : \{\tau \in Paths\} \to \{i : \mathbb{N} \mid i \leq (len\,\tau)\} \to Paths$ by $suf\,\tau\,0 = \tau$ and $suf(s\,\tau)(i+1) = suf\,\tau\,i$. Intuitively, $suf\,\tau\,i$ is the sequence obtained by removing $i \leq (len\,\tau)$ elements from the "beginning" of τ. This is required in the definition of the following relation and is used hereafter.

Definition 5. *$\hookrightarrow \subseteq Paths \times \Pi$ is the largest set of pairs (τ, r) such that: (i) $\tau = s$ for some $s \in S$ such that $r\,s$; or (ii) $\tau = s\,\tau'$, for some $s \in S$, $\tau' \in Paths$ such that $r\,s$; or (iii) $\tau = s\,\tau'$ for some $s \in S$, $\tau' \in Paths$ and $n \leq (len\,\tau')$ such that $((suf\,\tau'\,n), r) \in \hookrightarrow$.*

We write $\tau \hookrightarrow r$ instead of $(\tau, r) \in \hookrightarrow$. By analogy with Lemma 1 (Knaster-Tarski's theorem for the $\rightsquigarrow$ relation), but by using the strong version of the theorem, we obtain:

Lemma 8. *Let $R \subseteq Paths \times \Pi$ be s.t. $(\tau, r) \in R \Rightarrow (\exists s.\tau = s \wedge r\,s) \vee (\exists s.\exists\tau'.\tau = s\,\tau' \wedge r\,s) \vee (\exists s.\exists\tau'.\exists n.\exists\tau''.\tau = s\,\tau' \wedge \tau'' = (suf\,\tau'\,n) \wedge ((\tau'', r) \in R \vee \tau'' \hookrightarrow r))$. Then $R \subseteq \hookrightarrow$.*

The following lemma is easily proved, by instantiating the parameter R, which occurs in both Lemmas 1 and 8 for the relations $\rightsquigarrow, \hookrightarrow$, with the other lemma's parameter R:

Lemma 9 ($\rightsquigarrow$ equals $\hookrightarrow$). *For all $\tau \in Paths$ and $r \in \Pi$, $\tau \rightsquigarrow r$ if and only if $\tau \hookrightarrow r$.*

Theorem 5 (Soundness of $\Vdash$). *If for all $\varphi' \in \mathcal{H}$, $\mathcal{S} \models \varphi'$, then $\mathcal{S}, \mathcal{H} \Vdash \varphi$ implies $\mathcal{S} \models \varphi$.*

Proof. Like $\Vdash$ itself this soundness proof uses both coinduction and induction.

For the coinductive part: we show that one can apply Lemma 8 (Knaster-Tarski's theorem for the $\hookrightarrow$ relation) with the parameter $R \triangleq \lambda(\tau, r).\exists l.(\mathcal{S}, \mathcal{H} \Vdash l \Rightarrow\!\!\diamond r \wedge l\,(hd\,\tau) \wedge \tau \in fPaths)$, where $fPaths \subseteq Paths$ is the subset of finite paths of $\mathcal{S}$.

This amounts to showing (†): *for all $l \Rightarrow\!\!\diamond r \in \Phi$ and $\tau \in fPaths$, if $\mathcal{S}, \mathcal{H} \Vdash l \Rightarrow\!\!\diamond r$ and $l\,(hd\,\tau)$ then either (i) or (ii) or (iii) holds, where*

- *(i)* $(\exists s.\tau = s \wedge r\,s)$*, or*
- *(ii)* $(\exists s.\exists\tau'.\tau = s\,\tau' \wedge r\,s)$*, or*
- *(iii)* $(\exists s.\exists\tau'.\exists n.\exists\tau''.\tau = s\,\tau' \wedge \tau'' = (suf\,\tau'\,n) \wedge ((\tau'', r) \in R \vee \tau'' \hookrightarrow r))$.

In order to prove (†), remember that $\mathcal{S}, \mathcal{H} \Vdash \varphi$ is just $\varphi \in \nu\mu\vdash_{\mathcal{S},\mathcal{H}}$. Since the latter is a fixpoint of the monotonous function $Y \mapsto \mu\vdash_{\mathcal{S},\mathcal{H},Y}$ we have that $\nu\mu\vdash_{\mathcal{S},\mathcal{H}} = \mu\vdash_{\mathcal{S},\mathcal{H},\nu\mu\vdash_{\mathcal{S},\mathcal{H}}}$. Hence, (†) amounts to proving that for all $l \Rightarrow\!\!\diamond r \in \mu\vdash_{\mathcal{S},\mathcal{H},\nu\mu\vdash_{\mathcal{S},\mathcal{H}}}$ and $\tau \in fPaths$, if and $l\,(hd\,\tau)$ then either (i) or (ii) or (iii) above holds.

In order to get rid of the set $\mu\vdash_{\mathcal{S},\mathcal{H},\nu\mu\vdash_{\mathcal{S},\mathcal{H}}}$ from above we use the inductive part of the proof, i.e., Knaster-Tarski's theorem for *smallest* fixpoints: for any $P \subseteq \Phi$, if

1. $\mathcal{H} \subseteq P$;
2. for all $r \in \Pi$, $r \Rightarrow\!\!\diamond\, r \in P$;
3. for all $l, l', r \in \Pi$, $l \sqsubseteq l'$ and $l' \Rightarrow\!\!\diamond\, r \in P$ implies $l \Rightarrow\!\!\diamond\, r \in P$;
4. for all $l_1, l_2, r \in \Pi$, $l_1 \Rightarrow\!\!\diamond\, r \in P$ and $l_2 \Rightarrow\!\!\diamond\, r \in P$ imply $(l_1 \sqcup l_2) \Rightarrow\!\!\diamond\, r \in P$;
5. for all $l, m, r \in \Pi$, $\mathcal{S} \models l \Rightarrow\!\!\diamond\, m$ and $m \Rightarrow\!\!\diamond\, r \in P$ imply $l \Rightarrow\!\!\diamond\, r \in P$;
6. for all $l, r \in \Pi$, $l \sqcap \bullet \sqsubseteq \bot$ and $\partial l \Rightarrow\!\!\diamond\, r \in \nu\mu \vdash_{\mathcal{S},\mathcal{H}}$ imply $l \Rightarrow\!\!\diamond\, r \in P$;

then P is a fixpoint of the monotonous function $X \mapsto \vdash_{\mathcal{S},\mathcal{H},\nu\mu\vdash_{\mathcal{S},\mathcal{H}}}(X)$ and therefore $\mu \vdash_{\mathcal{S},\mathcal{H},\nu\mu\vdash_{\mathcal{S},\mathcal{H}}} \subseteq P$. Thus, in order to prove (†) it is enough to find a set P of formulas satisfying the above constraints, and such that for all $l \Rightarrow\!\!\diamond\, r \in P$ and $\tau \in \mathit{fPaths}$, if and $l\,(\mathit{hd}\,\tau)$ then either (i) or (ii) or (iii) above holds.

We choose $P = \{l \Rightarrow\!\!\diamond\, r \in \Phi \mid \forall \tau \in \mathit{fPaths}, l\,(\mathit{hd}\tau) \Rightarrow O(\tau, r)\}$, where $O(\tau, r)$ the disjunction of (i), (ii), and (iii), and show that this P satisfies the constraints (1)-(6).

- Constraints 1 and 2 refer to valid formulas. For such formulas, say, $l \Rightarrow\!\!\diamond\, r$, it holds by definition of validity and Lemma 9 that for all $\tau \in \mathit{fPaths}$ such that $l\,(\mathit{hd}\,\tau)$, either $(\exists s.\tau = s \wedge r\,s)$ or $(\exists s.\exists\tau'.\tau = s\,\tau' \wedge r\,s)$ or $(\exists s.\exists\tau'.\exists n.\exists\tau''.\tau = s\,\tau' \wedge \tau'' = (\mathit{suf}\,\tau'\,n) \wedge \tau'' \hookrightarrow r)$. This implies $l \Rightarrow\!\!\diamond\, r \in P$;

- For constraint 3, assume $l \sqsubseteq l'$ and $l' \Rightarrow\!\!\diamond\, r \in P$, and consider any $\tau \in \mathit{fPaths}$ such that $l\,(\mathit{hd}\,\tau)$. Then, we also have $l'\,(\mathit{hd}\,\tau)$, and from $l' \Rightarrow\!\!\diamond\, r \in P$ we obtain (i) or (ii) or (iii), which do not depend on l' and also work to show $l \Rightarrow\!\!\diamond\, r \in P$;

- For constraint 4, consider any $\tau \in \mathit{fPaths}$ such that $(l_1 \sqcup l_2)\,(\mathit{hd}\,\tau)$, i.e., $l_1\,(\mathit{hd}\,\tau)$ or $l_2\,(\mathit{hd}\,\tau)$. If $l_1\,(\mathit{hd}\,\tau)$, then $l_1 \Rightarrow\!\!\diamond\, r \in P$ implies (i) or (ii) or (iii), which also work for proving, $(l_1 \sqcup l_2) \Rightarrow\!\!\diamond\, r \in P$. The case $l_2\,(\mathit{hd}\,\tau)$ is similar;

- For constraint 5: consider any $\tau \in \mathit{fPaths}$ such that $l\,(\mathit{hd}\,\tau)$. From $\mathcal{S} \models l \Rightarrow\!\!\diamond\, m$ we obtain thanks to Lemma 3 (*transitivity* item) some $k \leq (\mathit{len}\,\tau)$ such that $m\,(\tau\,k)$. Let $\tau' = (\mathit{suf}\,\tau\,k)$, then $m\,(\tau\,k)$ means $m\,(\mathit{hd}\,\tau')$, and from $m \Rightarrow\!\!\diamond\, r \in P$ we obtain that either (a) $(\exists s.\tau' = s \wedge r\,s)$ or (b) $(\exists s.\exists\tau''.\tau' = s\,\tau'' \wedge r\,s)$ or (c) $(\exists s.\exists\tau''.\exists n.\exists\tau'''.\tau' = s\,\tau'' \wedge \tau''' = (\mathit{suf}\,\tau''\,n) \wedge ((\tau''', r) \in R \vee \tau''' \hookrightarrow r))$. Cases (a) and (b) imply either conditions (i) or (ii) for $l \Rightarrow\!\!\diamond\, r \in P$, hence, we focus on case (c), in which there exist s, τ'', n, τ''' such that $\tau' = s\,\tau'' \wedge \tau''' = (\mathit{suf}\,\tau''\,n) \wedge ((\tau''', r) \in R \vee \tau''' \hookrightarrow r))$. We obtain that there do exist $s_0 = (\mathit{hd}\,\tau)$, $\tau''_0 = \mathit{suf}\,\tau\,1$, $n_0 = n + k, \tau'''_0 = \mathit{suf}\,\tau''\,n_0$ such that $\tau = s_0\,\tau''_0 \wedge \tau'''_0 = (\mathit{suf}\,\tau''\,n_0) \wedge ((\tau'''_0, r) \in R \vee \tau'''_0 \hookrightarrow r))$. [Specifically, $(\tau''', r) \in R$ implies $(\tau'''_0, r) \in R$, and $\tau''' \hookrightarrow r$ implies $\tau'''_0 \hookrightarrow r$ due to the definitions of R and $\hookrightarrow$.] The existence of $s_0, \tau''_0, n_0, \tau'''_0$ with the above properties is condition (iii) for $l \Rightarrow\!\!\diamond\, r \in P$. Note that the asymmetry in the [Tra] rule of our proof system gave us the hypothesis $\mathcal{S} \models l \Rightarrow\!\!\diamond\, m$, which is essential in this case: without it, $l \Rightarrow\!\!\diamond\, r \in P$ cannot be proved;

- For constraint 6: consider any $\tau \in \mathit{fPaths}$ such that $l\,(\mathit{hd}\,\tau)$. Assume $\tau = s$ for some $s \in S$. Thus, $l\,s$ and s is final, contradicting the hypothesis $l \sqcap \bullet \sqsubseteq \bot$. Hence, $\tau = s\,\tau'$ for some $s \in S$ and $\tau' \in \mathit{fPaths}$. From $\partial l \Rightarrow\!\!\diamond\, r \in \nu\mu \vdash_{\mathcal{S},\mathcal{H}}$ we obtain $\mathcal{S}, \mathcal{H} \Vdash \partial l \Rightarrow\!\!\diamond\, r$. Moreover, from the definition of the symbolic transition function ∂ and $s \rightarrow (\mathit{hd}\,\tau')$ and $l\,s$ we obtain $(\partial l)\,(\mathit{hd}\,\tau')$. From the definition of R,

with the existentially quantified variable therein set to ∂l, we obtain $(\tau', r) \in R$. Hence, there do exist $s_0 = s$, $\tau''_0 = \tau'$, $n_0 = 0$, $\tau'''_0 = \mathit{suf}\,\tau''\,n_0$ such that $\tau = s_0\,\tau''_0 \wedge \tau'''_0 = (\mathit{suf}\,\tau''\,n_0) \wedge (\tau'''_0, r) \in R$, implying condition (iii) for $l \Rightarrow\!\!\diamond\, r \in P$, which concludes the proof of this last case and of the statement (†).

Hence, one can apply Lemma 8 with the parameter $R \triangleq \lambda(\tau, r).\exists l.(\mathcal{S}, \mathcal{H} \Vdash l \Rightarrow\!\!\diamond\, r \wedge l\,(\mathit{hd}\,\tau) \wedge \tau \in \mathit{fPaths})$. As a consequence, $R \subseteq \hookrightarrow$, thus, any $\varphi \in \Phi$ that satisfies the hypotheses of the theorem, in particular, such that $\mathcal{S}, \mathcal{H} \Vdash \varphi$, and any $\tau \in \mathit{fPaths}$ such that $(\mathit{lhs}\,\varphi)\,(\mathit{hd}\,\tau)$, have the property $\tau \hookrightarrow (\mathit{rhs}\,\varphi)$, thus, by the definition of validity and Lemma 9, $\mathcal{S} \models \varphi$, i.e., the conclusion of our theorem. □

Example 6. *We sketch a proof of the fact that the transition system $\mathcal{S}$ denoted by the state machine in Figure 1 meets its functional correctness property: (i) $\mathcal{S} \models (c = c_0) \Rightarrow\!\!\diamond\, (c = c_2 \wedge s = m \times (m+1)/2)$. We first show (ii) $\mathcal{S} \models (c = c_0) \Rightarrow\!\!\diamond\, (c = c_1 \wedge i = 0 \wedge s = 0)$, which can be done using in sequence the rules [Stp], [Str], and [Trv] of the $\Vdash$ proof system together with its soundness. Using (ii) and the [Tra] rule, (i) reduces to proving (iii) $\mathcal{S} \Vdash (c = c_1 \wedge i = 0 \wedge s = 0) \Rightarrow\!\!\diamond\, (c = c_2 \wedge s = m \times (m+1)/2)$. Next, in Examples 4 and 5 we established*[1] *$\mathcal{S} \models (c = c_1 \wedge i = 0 \wedge s = 0) \Rightarrow\!\!\diamond\, (c = c_1 \wedge i = m \wedge s = i \times (i+1)/2)$, hence, using this and the [Tra] rule, (iii) reduces to proving $\mathcal{S} \Vdash (c = c_1 \wedge i = m \wedge s = i \times (i+1)/2) \Rightarrow\!\!\diamond\, (c = c_2 \wedge s = m \times (m+1)/2)$. This is performed by applying the sequence [Stp], [Str], and [Trv], which concludes the proof.*

Compositionality. Remembering Definition 4 of asymmetrical compositionality:

Theorem 6. *$\Vdash$ is asymmetrically compositional.*

Proof. We have to show that if (i) $\mathcal{S} \Vdash \varphi'$ and (ii) $\mathcal{S}, \{\varphi'\} \Vdash \varphi$ then $\mathcal{S} \Vdash \varphi$. Now, (i) and (ii) and the soundness of $\Vdash$ imply $\mathcal{S} \models \varphi'$ and then $\mathcal{S} \models \varphi$, and then the conclusion $\mathcal{S} \Vdash \varphi$ holds by the completeness of $\Vdash$. □

Note that the statement (i) can be replaced by a weaker $\mathcal{S}' \Vdash \varphi'$ for components $\mathcal{S}' \lhd \mathcal{S}$, thanks to the soundness and completeness of $\Vdash$ and of Theorem 1. This allows us to mix structural compositionality with the logical (asymmetrical) one in proofs of $\Vdash$.

The $\Vdash$ proof system is thus better at compositionality than $\vdash$, thanks to the inclusion of inductive rules, in particular, of the rule [Hyp], but at the cost of a substantially more involved soundness proof. It still has a problem: the asymmetry of the [Tra] rule, required by the soundness proof, is not elegant since the rule mixes semantics $\models$ and syntax $\Vdash$. This is not only an issue of elegance, but a practical issue as well.

Example 7. *We attempt to prove the property (i) from Example 6 using the asymmetrical compositionality of $\Vdash$. The first step, similar to that of Example 6, is proving (ii') $\mathcal{S} \Vdash (c = c_0) \Rightarrow\!\!\diamond\, (c = c_1 \wedge i = 0 \wedge s = 0)$ by using in sequence the rules [Stp], [Str], and [Trv] of $\Vdash$. Then, Theorem 6 reduces (i) to (iii') $\mathcal{S}, \{(c = c_0) \Rightarrow\!\!\diamond\, (c = c_1 \wedge i = 0 \wedge s = 0)\} \Vdash (c = c_0) \Rightarrow\!\!\diamond\, (c = c_2 \wedge s = m \times (m+1)/2)$. The natural next step would be to use the [Tra] rule of $\Vdash$, splitting (iii') in two parts: $\mathcal{S}, \{(c = c_0) \Rightarrow\!\!\diamond\, (c = c_1 \wedge i =$*

[1] Example 5 used the proof system $\vdash$ and its Lemma 5, but $\Vdash$ and its Lemma 7 can be used just as well.

$$\text{[Hyp]}\ \frac{}{\mathcal{S},\mathcal{H} \Vdash (\mathrm{T},\varphi)}\mu \ \textit{ if } (\mathrm{F},\varphi) \in \mathcal{H}$$

$$\text{[Trv]}\ \frac{}{\mathcal{S},\mathcal{H} \Vdash (b, r \Rightarrow\!\Diamond\, r)}\mu$$

$$\text{[Str]}\ \frac{\mathcal{S},\mathcal{H} \Vdash (b, l' \Rightarrow\!\Diamond\, r)}{\mathcal{S},\mathcal{H} \Vdash (b, l \Rightarrow\!\Diamond\, r)}\mu \ \textit{ if } l \sqsubseteq l'$$

$$\text{[Spl]}\ \frac{\mathcal{S},\mathcal{H} \Vdash (b, l_1 \Rightarrow\!\Diamond\, r) \qquad \mathcal{S},\mathcal{H} \Vdash (b, l_2 \Rightarrow\!\Diamond\, r)}{\mathcal{S},\mathcal{H} \Vdash (b, (l_1 \sqcup l_2)) \Rightarrow\!\Diamond\, r}\mu$$

$$\text{[Tra]}\ \frac{\mathcal{S},\mathcal{H} \Vdash (b, l \Rightarrow\!\Diamond\, m) \qquad \mathcal{S},\mathcal{H} \Vdash (b, m \Rightarrow\!\Diamond\, r)}{\mathcal{S},\mathcal{H} \Vdash (b, l \Rightarrow\!\Diamond\, r)}\mu$$

$$\text{[Stp]}\ \frac{\mathcal{S},\mathcal{H} \Vdash (\mathrm{T}, \partial l \Rightarrow\!\Diamond\, r)}{\mathcal{S},\mathcal{H} \Vdash (b, l \Rightarrow\!\Diamond\, r)}\mu \textit{ if } l \sqcap \bullet \sqsubseteq \bot$$

$$\text{[Cut]}\ \frac{\mathcal{S},\mathcal{H} \Vdash (\mathrm{F},\varphi') \qquad \mathcal{S},\mathcal{H} \cup \{(\mathrm{F},\varphi')\} \Vdash (b,\varphi)}{\mathcal{S},\mathcal{H} \Vdash (b,\varphi)}\mu$$

$$\text{[Cof]}\ \frac{\mathcal{S},\mathcal{H} \cup \{(\mathrm{F},\varphi)\} \Vdash (\mathrm{F},\varphi)}{\mathcal{S},\mathcal{H} \Vdash (b,\varphi)}\mu$$

$$\text{[Clr]}\ \frac{\mathcal{S},\mathcal{H} \Vdash (b,\varphi)}{\mathcal{S},\mathcal{H} \cup \{(b',\varphi')\} \Vdash (b,\varphi)}\mu$$

Figure 4: Inductive proof system, with an encoding of domain-specific coinduction

$0 \wedge s = 0)\} \Vdash (c = c_0) \Rightarrow\!\Diamond\, (c = c_1 \wedge i = 0 \wedge s = 0)$*, to be discharged by [Hyp], and then* $\mathcal{S}, \{(c = c_0) \Rightarrow\!\Diamond\, (c = c_1 \wedge i = 0 \wedge s = 0)\} \Vdash (c = c_1 \wedge i = 0 \wedge s = 0) \Rightarrow\!\Diamond\, (c = c_2 \wedge s = m \times (m+1)/2)$*. But the [Tra] rule of* $\Vdash$*, as it is, does not allow this, and [Hyp] cannot be used to discharge the goal above because the hypothesis in the left-hand side of* $\Vdash$ *is lost. Hence, when one uses compositionality in* $\Vdash$*-proofs one may get stuck.*

These issues are solved in the third proof system, which incorporates even more induction that the second one, and specialises its coinduction even closer to our problem domain. The third proof system also has better compositionality features. These gains come, however, at the cost of an even more involved soundness proof.

6. A Symmetrically-Compositional Proof System

Our third proof system is depicted in Figure 4. A first difference with the previous one is that hypotheses and conclusions are pairs of a Boolean tag and a formula. We call them tagged formulas, or simply formulas when there is no risk of confusion. The role of the tags is to avoid unsoundness. To see this, note that without the Boolean tags, one could assume any formula φ under proof as a new hypothesis by (bottom-up) applying the rule [Cof], and finish the proof with [Hyp]. However, with the Boolean flag this unsound behaviour is impossible: after the [Cof] rule is applied, the Boolean tags of both hypothesis and conclusion are F, which prevents the application of `[Hyp]`. We called the rule [Cof] because it "emulates" in our proof system the Coq `cofix` tactic, which does something similar at the level of Coq's proof system.

The second difference is that the proof system is inductive, i.e., there are no more infinite proofs, and no instances of the Knaster-Tarski theorem any more. It encodes a domain-specific coinduction, tailored to our problem of RL formula verification.

Another difference, especially with the second proof system $\Vdash$, is that the hypotheses set is not constant. The following rules change the hypotheses set. First, the [Cof] rule, already explained above. Then, the [Cut] rule, which says that in order to prove (b, φ) under hypotheses $\mathcal{H}$, it is enough to prove $(\textsc{f}, \varphi')$ - for some formula φ' - under hypotheses $\mathcal{H}$, and to prove (b, φ) under $\mathcal{H} \cup \{(\textsc{f}, \varphi')\}$. This resembles a standard cut rule, except for the way in which the Boolean is handled, which is, again, tailored to our specific setting, in order to avoid unsoundness in RL formula verification. Third, the [Clr] rule removes a formula from the hypotheses. Note that the [Stp] rule, when applied bottom to top, switches the Boolean from whatever value b it has to $\textsc{t}$. Hence, it is [Stp] that makes "progress" in our setting, enabling the use of [Hyp] in a sound way. The other rules have the same respective roles as their homonyms in the $\Vdash$ proof system. In particular, the asymmetry in the rule [Tra] has been fixed.

Soundness. We present the soundness proof of $\Vdash\!\!\!\vdash$ at a higher level of abstraction than for the other proof systems. For example, we define $\Vdash\!\!\!\vdash$-proofs as finite trees, and (reasonably) assume that finite trees are known to the readers. For the other proof systems we adopted a more formal approach because the proofs in those systems were certain kinds of possibly infinite trees, whose a priori knowledge cannot be assumed.

Definition 6 (Proof). *A* proof *of a tagged formula (b, φ) for a transition system $\mathcal{S}$ and under hypotheses $\mathcal{H}$ - for short, a proof of $\mathcal{S}, \mathcal{H} \Vdash\!\!\!\vdash (b, \varphi)$ - is a finite tree, whose root is labelled by the sequent $\mathcal{S}, \mathcal{H} \Vdash\!\!\!\vdash (b, \varphi)$, and whose other nodes are also labelled by sequents, obtained by applying bottom-up the rules depicted in Figure 4.*

We sometimes just write $\mathcal{S}, \mathcal{H} \Vdash\!\!\!\vdash (b, \varphi)$ for "there is a proof of $\mathcal{S}, \mathcal{H} \Vdash\!\!\!\vdash (b, \varphi)$" as defined above. The following definition introduces the sets of all hypotheses and of all conclusions occurring in a proof.

Definition 7 (All hypotheses and conclusions occurring in proof). *Assume a proof Θ of $\mathcal{S}, \mathcal{H} \Vdash\!\!\!\vdash (b, \varphi)$. The set* Hyp *is the union of all sets $\mathcal{H}'$ of formulas, for all the node-labels $\mathcal{S}, \mathcal{H}' \Vdash\!\!\!\vdash (b', \varphi')$ of nodes occurring in the tree Θ. The set* Con *is the set of all formulas (b', φ'), for all the node-labels $\mathcal{S}, \mathcal{H}' \Vdash\!\!\!\vdash (b', \varphi')$ occurring in Θ.*

Hereafter in the current subsection about soundness we assume a proof (tree) Θ of $\mathcal{S}, \mathcal{H} \Vdash\!\!\!\vdash (b, \varphi)$ with corresponding sets Hyp and Con. The following technical lemma is proved by structural induction on such trees. It says that tagged formulas in Hyp are among the hypotheses $\mathcal{H}$ present at the root of Θ, plus the conclusions Con.

Lemma 10. $\mathsf{Hyp} \subseteq \mathcal{H} \cup \mathsf{Con}$.

Proof. By induction on the proof of $\mathcal{S}, \mathcal{H} \Vdash\!\!\!\vdash (b, \varphi)$ that generated the sets Hyp and Con. The base cases consist of proofs that are single applications of the rules [Hyp] or [Trv], before and after which $\mathsf{Hyp} = \mathcal{H}$, which trivially satisfies the lemma. For the inductive step: assume the lemma holds for a given proof. If the proof is augmented

with applications of the rules [Str], [Spl], [Tra], or [Stp], the set Hyp does not change, and the set Con either grows or stays constant, which ensures that the lemma still holds after applying such rules. If the applied rule are [Cut] or [Cof], both Hyp and Con are enriched with a tagged formula of the form $(\textsc{f}, \varphi)$, ensuring that the lemma remains true after applying such rules. Finally, if the rule is [Clr], Hyp loses one element, which, again, has the effect that the lemma still holds after applying the rule. □

Some more notions need to be defined before we can prove soundness. First, a *pad* in a tree is a sequence of consecutive edges, and the length of a pad is the number of nodes on the pad. Hence, the length of a pad is strictly positive.

Definition 8. *The* last occurrence of a tagged formula $(b', \varphi') \in \mathsf{Con}$ in Θ *is the maximal length of a pad from the root* $\mathcal{S}, \mathcal{H} \Vdash (b, \varphi)$ *of* Θ *to some node labelled by* $\mathcal{S}, \mathcal{H}' \Vdash (b', \varphi')$. *For formulas* $(b', \varphi') \notin \mathsf{Con}$ *we define by convention their last occurrence in* Θ *to be* 0. *This defines a total function last* : $\mathbb{B} \times \Phi \to \mathbb{N}$.

Remember that *fPaths* denote the set of finite paths of the transition system under consideration. We now define the set $\mathcal{D} \triangleq \{(\tau', b', \varphi') \in \mathit{fPaths} \times \mathbb{B} \times \Phi \mid (\mathit{lhs}\,\varphi')(\mathit{hd}\,\tau') \wedge (b', \varphi') \in \mathsf{Con}\}$ on which we shall reason by well-founded induction. We equip $\mathcal{D}$ with a well-founded order, namely, with the restriction to $\mathcal{D}$ of the lexicographic-product order on $\mathit{fPaths} \times \mathbb{B} \times \Phi$ defined by $(\tau_1, b_1, \varphi_1) \prec (\tau_2, b_2, \varphi_2)$ iff

1. $\mathit{len}\,\tau_1 < \mathit{len}, \tau_2$, or
2. $\mathit{len}\,\tau_1 = \mathit{len}, \tau_2$ and $b_1 < b_2$, with $<$ on Booleans is defined by $\textsc{f} < \textsc{t}$, or
3. $\mathit{len}\,\tau_1 = \mathit{len}, \tau_2$ and $b_1 = b_2$, and $\mathit{last}(b_1, \varphi_1) > \mathit{last}(b_2, \varphi_2)$.

The first two orders in the product, on natural numbers and on Booleans, are well-founded. For the third one, since the order $\prec$ on $\mathit{fPaths} \times \mathbb{B} \times \Phi$ is restricted to $\mathcal{D}$, all last occurrences are bounded by the height of Θ, ensuring that the inequality $\mathit{last}(b_1, \varphi_1) > \mathit{last}(b_2, \varphi_2)$ induces a well-founded order. Hence, the restriction of $\prec$ on $\mathcal{D}$ (also denoted by $\prec$) is a well-founded order as well. The following lemma uses this.

Lemma 11. *Assume* $\mathcal{S}, \mathcal{H} \Vdash (\textsc{f}, l \Rightarrow\!\!\diamond\, r)$ *and for all* $(b', \varphi') \in \mathcal{H}$, $b' = \textsc{f}$ *and* $\mathcal{S} \models \varphi'$. *Let* $\mathcal{D}$ *be the domain corresponding to* $\mathcal{S}, \mathcal{H} \Vdash (\textsc{f}, l \Rightarrow\!\!\diamond\, r)$. *Then, for all* $(\tau, b, \varphi) \in \mathcal{D}$, *there is* $k \leq \mathit{len}\,\tau$ *such that* $(\mathit{rhs}\,\varphi)\,(\tau\,k)$.

Proof. Let Θ be a proof of $\mathcal{S}, \mathcal{H} \Vdash (\textsc{f}, l \Rightarrow\!\!\diamond\, r)$ and consider any $(\tau, b, \varphi) \in \mathcal{D}$; let $\varphi = l_\varphi \Rightarrow\!\!\diamond\, r_\varphi$, hence, $(b, l_\varphi \Rightarrow\!\!\diamond\, r_\varphi) \in \mathsf{Con}$. Thus, the last occurrence $\mathit{last}(b, l_\varphi \Rightarrow\!\!\diamond\, r_\varphi)$ of $(b, l_\varphi \Rightarrow\!\!\diamond\, r_\varphi)$ in Θ is a strictly positive natural number. In particular, there is $\mathcal{H}'$ and a node N labelled $\mathcal{S}, \mathcal{H}' \Vdash (b, l_\varphi \Rightarrow\!\!\diamond\, r_\varphi)$ that is on a pad of length $\mathit{last}(b, l_\varphi \Rightarrow\!\!\diamond\, r_\varphi)$ from the root of Θ.

We shall be using the following observation several times hereafter: (†) for any direct successor, labelled, say, $\mathcal{S}, \mathcal{H}'' \Vdash (b'', l'' \Rightarrow\!\!\diamond\, r'')$ of N, $\mathit{last}(b'', l'' \Rightarrow\!\!\diamond\, r'') > \mathit{last}(b, l_\varphi \Rightarrow\!\!\diamond\, r_\varphi)$. Indeed, there are instances of $(b'', l'' \Rightarrow\!\!\diamond\, r'')$ occurring further from the root of Θ than the furthest instance of $(b, l_\varphi \Rightarrow\!\!\diamond\, r_\varphi)$, which is in the node N. In particular, no direct successor of the node N has the conclusion $(b, l_\varphi \Rightarrow\!\!\diamond\, r_\varphi)$.

- if the node N is a leaf, then, the leaf results from applying either [Hyp] or [Trv].

  - if the leaf results from applying [Hyp], then $b = \mathrm{T}$ and $(\mathrm{F}, \varphi) \in \mathcal{H}' \subseteq \mathsf{Hyp}$. Using Lemma 10:
    * either $(\mathrm{F}, \varphi) \in \mathcal{H}$, where $\mathcal{H}$ is the set of initial hypotheses. Hence, $\mathcal{S} \models \varphi$, and using Lemma 2 we obtain $k \leq \mathit{len}\,\tau$ such that $(\mathit{rhs}\,\varphi)\,(\tau\,k)$, which proves the lemma in this case.
    * or $(\mathrm{F}, \varphi) \in \mathsf{Con}$. It follows that $(\tau, \mathrm{F}, \varphi) \in \mathcal{D}$, and, since $b = \mathrm{T}$, $(\tau, \mathrm{F}, \varphi) \prec (\tau, b, \varphi)$. Using the well-founded induction hypothesis, we obtain $k \leq \mathit{len}\,\tau$ such that $(\mathit{rhs}\,\varphi)\,(\tau\,k)$, which proves the lemma in this case.
  - if the leaf results from applying [Trv], then $l_\varphi = r_\varphi$, and from $(\tau, b, l_\varphi \Rightarrow\!\!\diamond\, r_\varphi) \in \mathcal{D}$ we have $(l_\varphi)\,(\mathit{hd}\,\tau)$, hence, $(r_\varphi)\,(\mathit{hd}\,\tau)$, thus with $k = 0$ the lemma is proved in this case.

- if the node N is not a leaf, then it has one or two successors in Θ generated by applying some rule of our proof system except [Hyp] and [Trv]. Depending on the rule:
  - if the rule is [Str], then N has one successor labelled $\mathcal{S}, \mathcal{H}' \Vdash (b, l' \Rightarrow\!\!\diamond\, r_\varphi)$ with $l_\varphi \sqsubseteq l'$. It follows that $(b, l' \Rightarrow\!\!\diamond\, r_\varphi) \in \mathsf{Con}$ and from $(l_\varphi)\,(\mathit{hd}\,\tau)$ and $l_\varphi \sqsubseteq l'$ we get $l'\,(\mathit{hd}\,\tau)$, thus, $(\tau, b, l' \Rightarrow\!\!\diamond\, r_\varphi) \in \mathcal{D}$. Moreover, using (†), $\mathit{last}(b, l' \Rightarrow\!\!\diamond\, r_\varphi) > \mathit{last}(b, l_\varphi \Rightarrow\!\!\diamond\, r_\varphi)$. Hence, $(\tau, b, l' \Rightarrow\!\!\diamond\, r_\varphi) \prec (\tau, b, l_\varphi \Rightarrow\!\!\diamond\, r_\varphi)$ and then using the well-founded induction hypothesis we obtain $k \leq \mathit{len}\,\tau$ such that $(\mathit{rhs}\,\varphi)\,(\tau\,k)$, proving the lemma in this case.
  - if the rule is [Spl] then $l_\varphi \triangleq l_1 \sqcup l_2$ and the node N has two successors, labelled $\mathcal{S}, \mathcal{H}' \Vdash (b, l_1 \Rightarrow\!\!\diamond\, r_\varphi)$ and $\mathcal{S}, \mathcal{H}' \Vdash (b, l_1 \Rightarrow\!\!\diamond\, r_\varphi)$, respectively. Hence, $(b, l_1 \Rightarrow\!\!\diamond\, r_\varphi) \in \mathsf{Con}$ and $(b, l_2 \Rightarrow\!\!\diamond\, r_\varphi) \in \mathsf{Con}$. From $(l_\varphi)\,(\mathit{hd}\,\tau)$ we obtain $l_1\,(\mathit{hd}\,\tau)$ or $l_2\,(\mathit{hd}\,\tau)$. We first consider the subcase $l_1\,(\mathit{hd}\,\tau)$. Then, $(\tau, b, l_1 \Rightarrow\!\!\diamond\, r_\varphi) \in \mathcal{D}$. Using (†), $\mathit{last}(b, l_1 \Rightarrow\!\!\diamond\, r_\varphi) > \mathit{last}(b, l_\varphi \Rightarrow\!\!\diamond\, r_\varphi)$. Hence, $(\tau, b, l_1 \Rightarrow\!\!\diamond\, r_\varphi) \prec (\tau, b, l_\varphi \Rightarrow\!\!\diamond\, r_\varphi)$ and using the well-founded induction hypothesis we obtain $k \leq \mathit{len}\,\tau$ such that $(\mathit{rhs}\,\varphi)\,(\tau\,k)$, proving the lemma in this subcase. The subcase $l_2\,(\mathit{hd}\,\tau)$ is identical.
  - if the rule is [Tra] then the node N has two successors, labelled by $\mathcal{S}, \mathcal{H}' \Vdash (b, l_\varphi \Rightarrow\!\!\diamond\, m)$ and by $\mathcal{S}, \mathcal{H}' \Vdash (b, m \Rightarrow\!\!\diamond\, r_\varphi)$, hence, $(b, l_\varphi \Rightarrow\!\!\diamond\, m) \in \mathsf{Con}$ and $(b, m \Rightarrow\!\!\diamond\, r_\varphi) \in \mathsf{Con}$. It follows that $(\tau, b, l_\varphi \Rightarrow\!\!\diamond\, m) \in \mathcal{D}$. Using (†), $\mathit{last}(b, l_\varphi \Rightarrow\!\!\diamond\, m) > \mathit{last}(b, l_\varphi \Rightarrow\!\!\diamond\, r_\varphi)$. Hence, $(\tau, b, l_\varphi \Rightarrow\!\!\diamond\, m) \prec (\tau, b, l_\varphi \Rightarrow\!\!\diamond\, r_\varphi)$. The well-founded induction hypothesis gives a $k_1 \leq \mathit{len}\,\tau$ such that $m\,(\tau\,k_1)$.
    * if $k_1 = 0$ then $m\,(\mathit{hd}\,\tau)$. It follows that $(\tau, b, m \Rightarrow\!\!\diamond\, r_\varphi) \in \mathcal{D}$, and, using (†), $\mathit{last}(b, m \Rightarrow\!\!\diamond\, r_\varphi) > \mathit{last}(b, l_\varphi \Rightarrow\!\!\diamond\, r_\varphi)$. Hence, $(\tau, b, m \Rightarrow\!\!\diamond\, r_\varphi) \prec (\tau, b, l_\varphi \Rightarrow\!\!\diamond\, r_\varphi)$. Using the well-founded induction hypothesis we obtain $k \leq \mathit{len}\,\tau$ such that $r_\varphi\,(\tau\,k)$, proving the lemma in this subcase.
    * if $k_1 > 0$ then let τ' be the suffix of τ starting at k_1. Hence, $\mathit{len}\,\tau' < \mathit{len}\,\tau$ and $m\,(\mathit{hd}\,\tau')$. It follows that $(\tau', b, m \Rightarrow\!\!\diamond\, r_\varphi) \in \mathcal{D}$, and, by definition of $\prec$, $(\tau', b, m \Rightarrow\!\!\diamond\, r_\varphi) \prec (\tau, b, l_\varphi \Rightarrow\!\!\diamond\, r_\varphi)$. Using the well-founded induction hypothesis, we then obtain $k_2 \leq \mathit{len}\,\tau'$ such that $r_\varphi\,(\tau'\,k_2)$. But $k \triangleq k_1 + k_2 \leq \mathit{len}\,\tau$, and $\tau\,k = \tau'\,k_2$, hence, $r_\varphi\,(\tau\,k)$, proving the lemma in this subcase.

- if the rule is [Stp] then the node N has one successor labelled $\mathcal{S}, \mathcal{H}' \Vdash (\mathrm{T}, \partial l_\varphi \Rightarrow\!\!\diamond\, r_\varphi)$, thus, $(\mathrm{T}, \partial l_\varphi \Rightarrow\!\!\diamond\, r_\varphi) \in \mathsf{Con}$, and $l_\varphi \sqcap \bullet \sqsubseteq \bot$.
  - assume first $\tau \triangleq s$ for some $s \in S$. Hence, $\bullet\, s$ and since $l_\varphi\, (hd\, \tau)$ we get $l_\varphi\, s$ which together with $\bullet\, s$ contradict $l_\varphi \sqcap \bullet \sqsubseteq \bot$.
  - thus, $\tau \triangleq s\, \tau'$, for some $\tau' \in fPaths$ with $s \to (hd\, \tau')$. From this and $l_\varphi\, s$ and using the definition of the ∂ function, $(\partial l_\varphi)\, (hd\, \tau')$. Since $len\, \tau' = len\, \tau - 1$ it follows that $(\tau', \mathrm{T}, \partial l_\varphi \Rightarrow\!\!\diamond\, r_\varphi) \prec (\tau, b, l_\varphi \Rightarrow\!\!\diamond\, r_\varphi)$ and using the well-founded induction hypothesis, there is $k' \leq len\, \tau'$ such that $r_\varphi\, (\tau'\, k')$. Setting $k = k' + 1$ we get $k \leq len\, \tau$ and $\tau\, k = \tau'\, k'$, thus, $r_\varphi\, (\tau\, k)$, which proves the lemma in this case.
- if the rule is [Cut], then the node N has two successors labelled $\mathcal{S}, \mathcal{H}' \Vdash (\mathrm{F}, l'_\varphi \Rightarrow\!\!\diamond\, r'_\varphi)$ and $\mathcal{S}, \mathcal{H}' \cup \{(\mathrm{F}, l'_\varphi \Rightarrow\!\!\diamond\, r'_\varphi)\} \Vdash (b, l_\varphi \Rightarrow\!\!\diamond\, r_\varphi)$, respectively. However, by (†), N has no successor in Θ with the conclusion $(b, l_\varphi \Rightarrow\!\!\diamond\, r_\varphi)$, a contradiction. The rule is not applicable for the chosen node.
- if the rule is [Cof] then the node N has one successor, labelled $\mathcal{S}, \mathcal{H}' \cup \{\mathrm{F}, l_\varphi \Rightarrow\!\!\diamond\, r_\varphi\} \Vdash (\mathrm{F}, l_\varphi \Rightarrow\!\!\diamond\, r_\varphi)$. Assuming $b = \mathrm{F}$ we obtain as above that the rule is not applicable for the chosen node. Hence, $b = \mathrm{T}$. We thus have $(\mathrm{T}, l_\varphi \Rightarrow\!\!\diamond\, r_\varphi) \in \mathsf{Con}$ and $(\tau, \mathrm{T}, l_\varphi \Rightarrow\!\!\diamond\, r_\varphi) \in \mathcal{D}$ and $(\tau, \mathrm{F}, l_\varphi \Rightarrow\!\!\diamond\, r_\varphi) \prec (\tau, \mathrm{T}, l_\varphi \Rightarrow\!\!\diamond\, r_\varphi)$, and using the well-founded induction hypothesis we obtain $k \leq len\, \tau$ such that $r_\varphi\, (\tau\, k)$, proving the lemma in this subcase.
- if the rule is [Clr] then $\mathcal{H}' \triangleq \mathcal{H}'' \cup \{\varphi''\}$, and the node N has one successor, labelled $\mathcal{S}, \mathcal{H}'' \Vdash (b, l_\varphi \Rightarrow\!\!\diamond\, r_\varphi)$. We obtain as above that the rule is not applicable for the chosen node, which proves the lemma in this last case.

Hence, in all possible cases, for the arbitrarily chosen $(\tau, b, \varphi) \in \mathcal{D}$ we found $k \leq len\, \tau$ such that $(rhs\, \varphi)\, (\tau\, k)$: the lemma is proved. □

Theorem 7 (Soundness of $\Vdash$). *If for all $(b', \varphi') \in \mathcal{H}$, $b' = \mathrm{F}$ and $\mathcal{S} \models \varphi'$, then $\mathcal{S}, \mathcal{H} \Vdash (\mathrm{F}, \varphi)$ implies $\mathcal{S} \models \varphi$.*

Proof. Let $\varphi \triangleq l \Rightarrow\!\!\diamond\, r$. Let $\mathcal{D}$ be the domain corresponding to $\mathcal{S}, \mathcal{H} \Vdash (\mathrm{F}, l \Rightarrow\!\!\diamond\, r)$. Consider any $\tau \in fPaths$ such that $l\, (hd\, \tau)$. Then, $(\tau, \mathrm{F}, l \Rightarrow\!\!\diamond\, r) \in \mathcal{D}$, hence, using lemma 11, there is $k \leq len\, \tau$ such that $r\, (\tau\, k)$. Hence, for any $\tau \in fPaths$ such that $l\, (hd\, \tau)$, there is $k \leq len\, \tau$ such that $r\, (\tau\, k)$. Using Lemma 2 we obtain $\mathcal{S} \models l \Rightarrow\!\!\diamond\, r$, which concludes the proof. □

Completeness. Proving the completeness of $\Vdash$ is the same as for the other proof system: prove a lemma reducing reachability to an invariance property and then show that for valid formulas that property holds.

Lemma 12. *If $l \sqsubseteq q \sqcup r$, $q \sqcap \bullet \sqsubseteq \bot$, and $\partial q \sqsubseteq q \sqcup r$ then $\mathcal{S} \Vdash (\mathrm{F}, l \Rightarrow\!\!\diamond\, r)$.*

Proof. We build a proof (tree) for $\mathcal{S} \Vdash (\mathrm{F}, l \Rightarrow\!\!\diamond\, r)$. The root of the tree is a node N_0 labelled $\mathcal{S} \Vdash (\mathrm{F}, l \Rightarrow\!\!\diamond\, r)$. N_0 has one successor N_1, generated by the [Str] rule, thanks to the hypothesis $l \sqsubseteq q \sqcup r$, and labelled $\mathcal{S} \Vdash (\mathrm{F}, (q \sqcup r) \Rightarrow\!\!\diamond\, r)$. N_1 has two successors $N_{2,1}$ and $N_{2,2}$, generated by the [Spl] rule, and labelled $\mathcal{S} \Vdash (\mathrm{F}, q \Rightarrow\!\!\diamond\, r)$ and

$\mathcal{S} \Vdash (\textsc{f}, r \Rightarrow\!\Diamond\, r)$, respectively. Using the [Trv] rule, $N_{2,2}$ has no successors. $N_{2,1}$ has one successor N_3, generated by the [Cof] rule, labelled $\mathcal{S}, \{(\textsc{f}, q \Rightarrow\!\Diamond\, r)\} \Vdash (\textsc{f}, q \Rightarrow\!\Diamond\, r)$. N_3 has one successor N_4, generated by the [Stp] rule, thanks to the hypothesis $q \sqcap \bullet \sqsubseteq \bot$, and labelled $\mathcal{S}, \{(\textsc{f}, q \Rightarrow\!\Diamond\, r)\} \Vdash (\textsc{t}, \partial q \Rightarrow\!\Diamond\, r)$. Note that the Boolean has switched from F to T, which enables us to later use the [Hyp] rule. The node N_4 has one successor, generated by the [Str] rule thanks to the hypothesis $\partial q \sqsubseteq q \sqcup r$: $\mathcal{S}, \{(\textsc{f}, q \Rightarrow\!\Diamond\, r)\} \Vdash (\textsc{t}, (q \sqcup r) \Rightarrow\!\Diamond\, r)$. N_4 has two successors $N_{5,1}$ and $N_{5,2}$, labelled $\mathcal{S}, \{(\textsc{f}, q \Rightarrow\!\Diamond\, r)\} \Vdash (\textsc{t}, q \Rightarrow\!\Diamond\, r)$ and $\mathcal{S}, \{(\textsc{f}, q \Rightarrow\!\Diamond\, r)\} \Vdash (\textsc{t}, r \Rightarrow\!\Diamond\, r)$, respectively. Neither has any successor: $N_{5,1}$, by the [Hyp] rule, and $N_{5,2}$, by the [Trv] rule. □

By analogy with Theorems 3 and 4 but using Lemma 12 (instead of Lemmas 5 and 7, respectively):

Theorem 8 (Completeness of $\Vdash$). *$\mathcal{S} \models \varphi$ implies $\mathcal{S} \Vdash \varphi$.*

Compositionality. $\Vdash$ has a symmetrical version of logical compositionality:

Theorem 9. *$\mathcal{S}, \mathcal{H} \cup \{(\textsc{f}, \varphi_1)\} \Vdash (\textsc{f}, \varphi_2)$ and $\mathcal{S}, \mathcal{H} \cup \{(\textsc{f}, \varphi_2)\} \Vdash (\textsc{f}, \varphi_1)$ imply $\mathcal{S}, \mathcal{H} \Vdash (\textsc{f}, \varphi_1)$ and $\mathcal{S}, \mathcal{H} \Vdash (\textsc{f}, \varphi_2)$.*

Proof. The statement is symmetrical in φ_1, φ_2; we prove it for the first formula. The rule [Cof] generates one successor for the root N_0 labelled $\mathcal{S}, \mathcal{H} \Vdash (\textsc{f}, \varphi_1)$: N_1, labelled $\mathcal{S}, \mathcal{H} \cup \{(\textsc{f}, \varphi_1)\} \Vdash (\textsc{f}, \varphi_1)$. From N_1, the rule [Cut] generates two successors, $N_{2,1}$ labelled $\mathcal{S}, \mathcal{H} \cup \{(\textsc{f}, \varphi_1)\} \Vdash (\textsc{f}, \varphi_2)$, which we assumed as a hypothesis of our theorem, and $N_{2,2}$, labelled $\mathcal{S}, \mathcal{H} \cup \{(\textsc{f}, \varphi_1), (\textsc{f}, \varphi_2)\} \Vdash (\textsc{f}, \varphi_1)$. From $N_{2,2}$ the rule [Clr] removes the first hypothesis and generates a node labelled $\mathcal{S}, \mathcal{H} \cup \{(\textsc{f}, \varphi_2)\} \Vdash (\textsc{f}, \varphi_1)$, which we assumed as a hypothesis of our theorem as well. □

Example 8. *In Example 7 we tried to prove $\mathcal{S} \models (c = c_0) \Rightarrow\!\Diamond\, (c = c_2 \wedge s = m \times (m + 1)/2)$ using the asymmetrical compositionality of $\vdash$, and noted that a certain proof step was impossible because of the asymmetry of the [Tra] rule of $\vdash$. We show that $\Vdash$ does not suffer from the same issue. The problem, reformulated in terms of $\Vdash$, was to start the sequent (iii') $\mathcal{S}, \{(\textsc{f}, (c = c_0) \Rightarrow\!\Diamond\, (c = c_1 \wedge i = 0 \wedge s = 0))\} \Vdash (\textsc{f}, (c = c_0) \Rightarrow\!\Diamond\, (c = c_2 \wedge s = m \times (m + 1)/2))$ and to use the [Tra] rule in order to split this sequent in two: $\mathcal{S}, \{(\textsc{f}, (c = c_0) \Rightarrow\!\Diamond\, (c = c_1 \wedge i = 0 \wedge s = 0))\} \Vdash (\textsc{f}, (c = c_0) \Rightarrow\!\Diamond\, (c = c_1 \wedge i = 0 \wedge s = 0))$ and then $\mathcal{S}, \{(\textsc{f}, (c = c_0) \Rightarrow\!\Diamond\, (c = c_1 \wedge i = 0 \wedge s = 0))\} \Vdash (\textsc{f}, (c = c_1 \wedge i = 0 \wedge s = 0) \Rightarrow\!\Diamond\, (c = c_2 \wedge s = m \times (m + 1)/2))$. This inference step is not a problem in $\Vdash$.*

Finally, we show how to combine structural and (symmetrical) logical compositionality. The following lemma says that $\Vdash$ is structurally compositional, even in the presence of hypotheses.

Lemma 13. *If $\mathcal{S}', \mathcal{H} \Vdash (b, \varphi)$ and $\mathcal{S}' \lhd \mathcal{S}$ then $\mathcal{S}, \mathcal{H} \Vdash (b, \varphi)$.*

Proof. We first make the following observation: $\mathcal{S}', \mathcal{H} \Vdash (b, \varphi)$ implicitly means $\varphi \in \Phi_{\mathcal{S}'}$, which thanks to the state-set inclusion induced by $\mathcal{S}' \lhd \mathcal{S}$ implies $\varphi \in \Phi_{\mathcal{S}}$ as well.

The proof goes by induction on the assumed proof Θ' of $\mathcal{S}', \mathcal{H} \Vdash (b, \varphi)$. We build a proof Θ of $\mathcal{S}, \mathcal{H} \Vdash (b, \varphi)$ and a partial function M from the nodes of Θ to those of Θ', such that at any point in the construction, each leaf, labelled $\mathcal{S}, \mathcal{H}' \Vdash (b', \varphi')$ of

a partially-constructed tree Θ is mapped by M to exactly one node labelled $\mathcal{S}', \mathcal{H}' \Vdash (b', \varphi')$ in Θ'.

The root of Θ is labelled $\mathcal{S}, \mathcal{H} \Vdash (b, \varphi)$ and is mapped by M to the root of Θ', labelled $\mathcal{S}', \mathcal{H} \Vdash (b, \varphi)$.

Assume Θ and M are partially built; we show how to continue this process. Let $\mathcal{S}, \mathcal{H}' \Vdash (b', \varphi')$ be the label of a current leaf L in current partially-build tree Θ. Using the induction hypothesis, L is mapped by M to exactly one node L' labelled $\mathcal{S}', \mathcal{H}' \Vdash (b', \varphi')$ of Θ'. The construction proceeds as follows:

- if L' is a leaf in Θ', then L remains a leaf in Θ.
- if L' is not a leaf in Θ and its successors $L'_1 \dots L'_k$ (for $k = 1$ or $k = 2$) are generated by any rule of the proof system except [Stp], then the same rule is applied to L and generates the same number of successor node $L_1 \dots L_k$, such that if L'_i is labelled by $\mathcal{S}', \mathcal{H}'_i \Vdash (b'_i, \varphi'_i)$ then the corresponding L_i is labelled by $\mathcal{S}, \mathcal{H}'_i \Vdash (b'_i, \varphi'_i)$; and M is extended to map each new L_i of Θ to the corresponding L'_i.
- L' is not a leaf in Θ and its successor L'_1 is generated by the rule [Stp]: let $\mathcal{S}', \mathcal{H}' \Vdash (b', \varphi')$ be the label of L', with $\varphi' \triangleq l \Rightarrow\!\!\diamond\, r$. Since [Stp] has been applied, $l \sqcap \bullet_{\mathcal{S}'} \sqsubseteq \bot$. Since $\rightarrow' \subseteq \rightarrow$, $\bullet_{\mathcal{S}} \subseteq \bullet_{\mathcal{S}'}$, hence, $l \sqcap \bullet_{\mathcal{S}} \subseteq l \sqcap \bullet_{\mathcal{S}'} \sqsubseteq \bot$, thus, [Stp] can also be applied to L, generating a new node $\widehat{L_1}$. Now, thanks to the rule [Stp], L'_1 is labelled $\mathcal{S}', \mathcal{H}' \Vdash (\mathrm{T}, \partial' l \Rightarrow\!\!\diamond\, r)$ where $\partial' l = \lambda s. \exists s'. l\, s' \wedge s' \rightarrow' s$. Similarly, $\widehat{L_1}$ is labelled $\mathcal{S}, \mathcal{H}' \Vdash (\mathrm{T}, \partial l \Rightarrow\!\!\diamond\, r)$ where $\partial l = \lambda s. \exists s'. l\, s' \wedge s' \rightarrow s$. Let now $\rightarrow'' \triangleq \rightarrow \setminus \rightarrow'$. Thus, $\rightarrow = \rightarrow' \cup \rightarrow''$, and we have the inclusion $\partial l = (\lambda s. \exists s'. l\, s' \wedge s' \rightarrow s) \sqsubseteq (\partial' l) \sqcup (\lambda s. \exists s'. l\, s' \wedge s' \rightarrow'' s)$.

  Let us assume there exist $s', s \in S$ such that $l\, s' \wedge s' \rightarrow'' s$. Since $l \in \Pi_{\mathcal{S}'}$, from $l\, s'$ we obtain $s' \in S'$. It follows that $s \in S \setminus S'$ because otherwise (by the second item in the definition of $\mathcal{S}' \lhd \mathcal{S}$) one would also have $s' \rightarrow' s$, in contradiction with $\rightarrow'' = \rightarrow \setminus \rightarrow'$. Now, $s' \rightarrow'' s$ implies $s' \rightarrow s$, which together with $s' \in S'$ and with $s \in S \setminus S'$ and the third item in the definition of $\mathcal{S}' \lhd \mathcal{S}$ implies $s' \in \bullet_{\mathcal{S}'}$. From the latter and $l\, s'$ we obtain a contradiction with $l \sqcap \bullet_{\mathcal{S}'} \sqsubseteq \bot$, which arose from assuming there exist $s', s \in S$ such that $l\, s' \wedge s' \rightarrow'' s$. It follows that there does not exist $s \in S$ such that $\exists s'. l\, s' \wedge s' \rightarrow'' s$; thus, $(\lambda s. \exists s'. l\, s' \wedge s' \rightarrow'' s) \sqsubseteq \bot$, and using the above inclusions, $\partial l \sqsubseteq \partial' l$.

  We apply to $\widehat{L_1}$ the rule [Str] using $\partial l \sqsubseteq \partial' l$ and obtain a leaf L_1 in Θ, labelled $\mathcal{S}, \mathcal{H}' \Vdash (\mathrm{T}, \partial' l \Rightarrow\!\!\diamond\, r)$, and we extend M to map L_1 to L'_1, which does have the corresponding label $\mathcal{S}', \mathcal{H}' \Vdash (\mathrm{T}, \partial' l \Rightarrow\!\!\diamond\, r)$.

The inductive construction of the proof Θ and of the map M is complete; we deduce $\mathcal{S}, \mathcal{H} \Vdash (b, \varphi)$. □

Combining Theorem 9 and Lemma 13 we obtain as a corollary the following theorem, which combines structural and logical compositionality:

Theorem 10. *If, for $i \in \{0, 1\}$, $\mathcal{S}_i \lhd \mathcal{S}$ and $\mathcal{S}_i, \mathcal{H} \cup \{(\mathrm{F}, \varphi_{1-i})\} \Vdash (\mathrm{F}, \varphi_i)$, then, for $i \in \{0, 1\}$, $\mathcal{S}, \mathcal{H} \Vdash (\mathrm{F}, \varphi_i)$.*

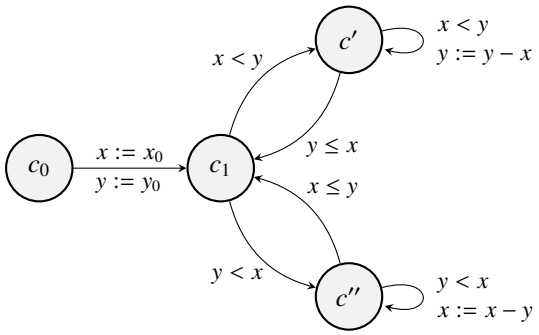

Figure 5: Computing a greatest common divisor

Example 9. *We sketch the verification of another infinite-state transition system, denoted by the state machine in Figure 5, which computes the greatest common divisor of two natural numbers. The obtained proof is probably not the simplest; for such simple systems a global (non-compositional) proof is typically shorter. Our goal here is to illustrate the compositionality features of* $\Vdash$ *embodied in Theorem 10.*

The state machine has four control nodes and operates with four natural-number variables: x, y, x_0 *and* y_0*. The last two variables are "symbolic constants", not modified by the transitions of the state machine, whose greatest-common divisor the machine is supposed to compute. On the leftmost transition x and y are initialised to* x_0 *and* y_0*. Then, depending on whether* $x < y$ *or* $y < x$*, the machine moves to either locations* c' *or* c''*; if* $x = y$ *the machine remains in* c_1 *and the computation terminates. On the upper self-loop on* c'*, x is subtracted from y while the guard* $x < y$ *holds; when the guard does not hold any more, the machine moves back to* c_1*. The lower self-loop on* c'' *reverses the roles of x and y. The state-machine denotes an infinite-state transition system* $\mathcal{S}$ *having the state-set* $\{c_0, c_1, c', c''\} \times \mathbb{N}^4$ *and transition relation*

$\bigcup_{x,y,x_0,y_0 \in \mathbb{N}} \{((c_0, x, y, x_0, y_0), (c_1, x_0, y_0, x_0, y_0))\} \cup$
$\bigcup_{x,y,x_0,y_0 \in \mathbb{N}, x<y} \{((c_1, x, y, x_0, y_0), (c', x, y, x_0, y_0))\} \cup$
$\bigcup_{x,y,x_0,y_0 \in \mathbb{N}, x<y} \{((c', x, y, x_0, y_0), (c', x, y - x, x_0, y_0))\} \cup$
$\bigcup_{x,y,x_0,y_0 \in \mathbb{N}, y \leq x} \{((c', x, y, x_0, y_0), (c_1, x, y, x_0, y_0))\} \cup$
$\bigcup_{x,y,x_0,y_0 \in \mathbb{N}, y<x} \{((c_1, x, y, x_0, y_0), (c'', x, y, x_0, y_0))\} \cup$
$\bigcup_{x,y,x_0,y_0 \in \mathbb{N}, y<x} \{((c'', x, y, x_0, y_0), (c'', x - y, y, x_0, y_0))\} \cup$
$\bigcup_{x,y,x_0,y_0 \in \mathbb{N}, x \leq y} \{((c'', x, y, x_0, y_0), (c_1, x, y, x_0, y_0))\}$

We identify two components of this transition system: $\mathcal{S}_1$*, encoded by the upper-right submachine, and* $\mathcal{S}_2$*, encoded by the lower-right submachine, having the respective state-spaces* $\{c_1, c'\} \times \mathbb{N}^4$ *and* $\{c_1, c''\} \times \mathbb{N}^4$ *and respective transition relations*

$\bigcup_{x,y,x_0,y_0 \in \mathbb{N}, x<y} \{((c_1, x, y, x_0, y_0), (c', x, y, x_0, y_0))\} \cup$
$\bigcup_{x,y,x_0,y_0 \in \mathbb{N}, x<y} \{((c', x, y, x_0, y_0), (c', x, y - x, x_0, y_0))\} \cup$
$\bigcup_{x,y,x_0,y_0 \in \mathbb{N}, y \leq x} \{((c', x, y, x_0, y_0), (c_1, x, y, x_0, y_0))\}$

and

$\bigcup_{x,y,x_0,y_0 \in \mathbb{N}, y<x} \{((c_1, x, y, x_0, y_0), (c'', x, y, x_0, y_0))\} \cup$
$\bigcup_{x,y,x_0,y_0 \in \mathbb{N}, y<x} \{((c'', x, y, x_0, y_0), (c'', x - y, y, x_0, y_0))\} \cup$

$\bigcup_{x,y,x_0,y_0\in\mathbb{N},x\leq y}\{((c'',x,y,x_0,y_0),(c_1,x,y,x_0,y_0))\}$
induced by their respective nodes and arrows subsets. We will show

(1) $\mathcal{S} \models (c = c_0 \wedge x_0 > 0 \wedge y_0 > 0) \Rightarrow\Diamond (c = c_1 \wedge x = y \wedge x = gcd(x_0, y_0))$.

which states the functional correctness of the system. Using [Stp] and several times [Str] and [Spl], and also $(x = x_0 \wedge y = y_0) \sqsubseteq (gcd(x, y) = gcd(x_0, y_0))$*, (1) reduces to*

(2) : $\mathcal{S} \Vdash (\text{F}, (c = c_1, gcd(x, y) = gcd(x_0, y_0) \wedge x_0 > 0 \wedge y_0 > 0 \wedge x < y) \Rightarrow\Diamond$
$(c = c_1 \wedge x = y \wedge x = gcd(x_0, y_0)))$*;*

(3) : $\mathcal{S} \Vdash (\text{F}, (c = c_1, gcd(x, y) = gcd(x_0, y_0) \wedge x_0 > 0 \wedge y_0 > 0 \wedge x = y) \Rightarrow\Diamond$
$(c = c_1 \wedge x = y \wedge x = gcd(x_0, y_0)))$*;*

(4) : $\mathcal{S} \Vdash (\text{F}, (c = c_1, gcd(x, y) = gcd(x_0, y_0) \wedge x_0 > 0 \wedge y_0 > 0 \wedge y < x) \Rightarrow\Diamond$
$(c = c_1 \wedge x = y \wedge x = gcd(x_0, y_0)))$.

The subgoal (3) is immediately discharged by applying the rules [Str] and [Trv]. The two other ones we prove by reducing them, thanks to Theorem 10 to the two following subgoals, with $\varphi_1 \triangleq (c = c_1 \wedge gcd(x, y) = gcd(x_0, y_0) \wedge x_0 > 0 \wedge y_0 > 0 \wedge x < y) \Rightarrow\Diamond (c = c_1 \wedge x = y \wedge x = gcd(x_0, y_0))$ *and* $\varphi_2 \triangleq (c = c_1 \wedge gcd(x, y) = gcd(x_0, y_0) \wedge x_0 > 0 \wedge y_0 > 0 \wedge y < x) \Rightarrow\Diamond (c = c_1 \wedge x = y \wedge x = gcd(x_0, y_0))$*:*

(5) : $\mathcal{S}_1, \{(\text{F}, \varphi_2)\} \Vdash (\text{F}, \varphi_1)$ *and (6) :* $\mathcal{S}_2, \{(\text{F}, \varphi_1)\} \Vdash (\text{F}, \varphi_2)$*. We prove (5), and note that the proof of (6) is symmetrical to that of (5). Using [Stp] then [Str], (5) becomes*

(7) : $\mathcal{S}_1, \{(\text{F}, \varphi_2)\} \Vdash (\text{T}, \varphi'_1)$

where $\varphi'_1 \triangleq (c = c' \wedge gcd(x, y) = gcd(x_0, y_0) \wedge x_0 > 0 \wedge y_0 > 0 \wedge x < y) \Rightarrow\Diamond (c = c_1 \wedge x = y \wedge x = gcd(x_0, y_0))$*. Using [Cof], (7) becomes*

(8) : $\mathcal{S}_1, \{(\text{F}, \varphi_2), (\text{F}, \varphi'_1)\} \Vdash (\text{F}, \varphi'_1)$.

Next, we apply [Stp] then several times [Str] and [Spl], and reduce (8) to

(9) : $\mathcal{S}_1, \{(\text{F}, \varphi_2), (\text{F}, \varphi'_1)\} \Vdash (\text{T}, (c = c_1 \wedge gcd(x, y) = gcd(x_0, y_0) \wedge x_0 > 0 \wedge y_0 > 0 \wedge y < x) \Rightarrow\Diamond (c = c_1 \wedge x = y \wedge x = gcd(x_0, y_0))$

(10) : $\mathcal{S}_1, \{(\text{F}, \varphi_2), (\text{F}, \varphi'_1)\} \Vdash (\text{T}, (c = c_1 \wedge gcd(x, y) = gcd(x_0, y_0) \wedge x_0 > 0 \wedge y_0 > 0 \wedge y = x) \Rightarrow\Diamond (c = c_1 \wedge x = y \wedge x = gcd(x_0, y_0))$

(11) : $\mathcal{S}_1, \{(\text{F}, \varphi_2), (\text{F}, \varphi'_1)\} \Vdash (\text{T}, (c = c' \wedge gcd(x, y) = gcd(x_0, y_0) \wedge x_0 > 0 \wedge y_0 > 0 \wedge x < y) \Rightarrow\Diamond (c = c_1 \wedge x = y \wedge x = gcd(x_0, y_0))$.

(12) : $\mathcal{S}_1, \{(\text{F}, \varphi_2), (\text{F}, \varphi'_1)\} \Vdash (\text{T}, (c = c' \wedge gcd(x, y) = gcd(x_0, y_0) \wedge x_0 > 0 \wedge y_0 > 0 \wedge y \leq x) \Rightarrow\Diamond (c = c_1 \wedge x = y \wedge x = gcd(x_0, y_0))$.

We note that the conclusion of (9) is (T, φ_2)*, and since* (F, φ_2) *is among its hypotheses, (9) is discharged by [Hyp]. Then, (11) is also discharged by [Hyp], as its conclusion is* (T, φ'_1) *and it has* (F, φ'_1) *among its hypotheses. Next, (10) is discharged by [Str] then [Trv]. Finally, for (12) one more application of [Stp] followed by [Str] and [Spl] reduces it to (9) and (10), which are discharged as explained above. All subgoals have been proved, hence, the proof of (1) is complete.*

7. Implementation

In this section we describe the implementation of the three proof systems, of their soundness and completeness proofs, and of their compositional features.

The main challenge has been to obtain Coq proofs that are close to paper proofs while keeping the paper proofs understandable. The difficulty arises from the fact that

certain mechanisms that one uses in the proof assistant produce formal proofs that do not translate well to paper (where the "natural" reasoning techniques are different).

This phenomenon especially affects our coinductive proofs: paper proofs are "naturally" based on the Knaster-Tarski theorem, while Coq proofs are "naturally" corecursive programs. The solution has been to develop a framework for deriving instances of the Knaster-Tarski theorem from Coq coinductive definitions and for proving those theorems in a systematic way, based on Coq's builtin coinduction mechanisms. Then, each time we employ Knaster-Tarski in a paper proof, we do the same in the corresponding Coq proof. For our first RL proof system this leads to essentially isomorphic Coq and paper proofs. This is in contrast with earlier work [16], where paper proofs tried to "mimic" Coq proofs and certain reasoning steps were essentially of the form "because Coq said so". For the second proof system, the additional benefit of using our framework is that we are able to obtain Coq proofs that are impossible with the direct use of the builtin mechanisms, due to their raw syntactical nature.

On the other hand, sometimes it becomes overly complicated to ignore some features available in the proof assistant when trying to obtain formal proofs that are isomorphic to paper proofs. We illustrate this with the third proof system and with the compositionality results, where we had to use certain features of Coq (dependent types and modules, respectively) just to be able to obtain Coq proofs. As a consequence the Coq proofs follow slightly different paths than the corresponding paper proofs. Reciprocally, adding all those details in the paper proofs would make them unreadable.

7.1. Basic Definitions

We start with some basic definitions, common to all three proof systems. At the end of this subsection we show a first instance of the Knaster-Tarski fixpoint theorem.

7.1.1. Transition Systems

Transitions systems consist of states and a transition relation between states; in Coq we declare `Parameter State:Type` and `Parameter trans:State->State->Prop`, where `Prop` is Coq's type for logical statements. For definitions the keyword `Parameter` for declarations is replaced with either `Definition` of `Inductive` or `CoInductive`, depending on the context. We define nonempty, possibly infinite sequences of states:

```
CoInductive Seq: Type:= one: State -> Seq |add: State -> Seq -> Seq
```

That is, we are defining the type `Seq` that consists either of a single state, wrapped into the *constructor* `one`, or, recursively, of the addition (via the constructor `add`) of a state to another sequence. The `CoInductive` keyword indicates that the `add` constructor can be applied infinitely many times, thus yielding infinite sequences. However, not all sequences are infinite: they may also end with an application of the constructor `one`.

Now that we have defined sequences, we can define *paths*, which are sequences of states connected by the transition relation, whose last state (if any) is *final*, i.e., it has no successor in the transition relation. We first define final states as follows:

```
Definition final:State -> Prop:= fun s=> forall s':State, ~trans s s'
```

Thus, `final` is a predicate on `State` (i.e., a function of type `State->Prop`) that is satisfied by states `s` such that for any other state `s'`, `trans s s'` does *not* hold. We can now define a predicate characterising the subset of sequences that are paths:

```
CoInductive isPath := single:  forall s, final s -> isPath (one s)
|more: forall s tau,  isPath tau -> trans s (head tau) -> isPath (add s tau)
```

This means that `isPath` is satisfied by paths that are singletons `one s` such that `s` is final, and by paths of the form `add s tau` such that the state `s` and the *head* of the sequence `tau` are in the relation `trans`, and, moreover, `tau` also satisfies `isPath`. Here, `head tau` is defined by case analysis: for sequences of the form `one s`, as well as for sequences of the form `add s tau'`, `head tau` is defined to be the state `s`. Note that we are using a coinductive definition for paths; had we used an inductive one, by replacing `CoInductive` with `Inductive`, we would have wrongly constrained paths to be finite.

Example 10. *The transition system denoted by the state machine in Fig. 1 is written in Coq as follows. One first defines the type* `Loc` *of control locations, which consists of values (or "inhabitants" in Coq terminology)* `c0`*,* `c1`*,* `c2`*. Then we define the type of states:* `Definition State := Loc*nat*nat*nat` *where, by convention, the first natural number will hold the counter* `i`*, the second one will hold the sum* `s`*, and the third one will hold the bound* `m`*. For the transition relation we use an inductive type:*

```
Inductive trans:  State -> State -> Prop :=
|init:  forall i s m, trans (c0,m,i,s) (c1,m,0,0)
|loop:  forall i s m i' s', i < m -> trans(c1,i,s,m)(c1,i+1,s+i+1,m)
|stop:  forall i s m, i >= m -> trans(c1,i,s,m)(c2,i,s,m)
```

The three constructors `init`*,* `loop`*, and* `stop` *correspond to the three arrows in Fig. 1.*

7.1.2. State Predicates and Symbolic Transition Relation

State predicates are defined to be the type `SymState:=State->Prop`. One particular state predicate is that of final states, which in the case of our example can be shown to be equivalent to `fun s => match s with (c,_,_,_) => c = c2 end`, that is, to states for which the control lies in `c2` and the natural-number variables have arbitrary values (hence, the "don't care" notation "`_`" in the above pattern-matching statement).

The top, bottom, conjunction, disjunction, and negation operations are defined next:

```
Definition top:SymState := fun _ => True.
Definition bot:SymState := fun _ => False.
Definition cnj(p q:SymState):SymState := fun s => p s ∧ q s.
Definition dsj(p q:SymState):SymState := fun s => p s ∨ q s.
Definition neg (p:SymState):SymState := fun s => ~p s.
```

The implication (sometimes called inclusion) between state predicates is defined by `imp(p q:SymState):=forall s,p s->q s.` The symbolic transition function is

```
symTrans(q:SymState):SymState:= fun s => exists s', q s' ∧ trans s' s.
```

7.1.3. Reachability Logic

Syntactically, RL formulas are inhabitants of a Coq type `Rlf` consisting of pairs `l =><> r` with `l` and `r` of type `SymState`. We define `lhs (l =><> r) = l`, `rhs (l =><> r) = r`. For the semantics, we first define the relation `reach` that corresponds to $\rightsquigarrow$ (cf. Def. 1). Its three constructors encode the conditions (i), (ii) and (iii) in that definition.

```
CoInductive reach:Seq -> SymState-> Prop :=
```

```
|reach_now_one:  forall s (r:SymState), r s-> reach (one s) r
|reach_now_add:  forall s tau (r:SymState), r s-> reach (add s tau) r
|reach_later:  forall s tau r,reach tau r -> reach (add s tau) r
```

Then, validity is essentially a literal translation of Definition 2. Based on this definitions we also state and prove the additional properties of validity (Lemma 3).

```
Definition valid(f:Rlf):= forall tau,isPath tau -> (lhs f)(head tau) ->
    reach tau (rhs f)
```

Example 11. *A formula for the functional correctness of the loop in the transition system whose Coq definition is shown in Example 10, is written as* `l =><> r` *with:*

```
Definition l:= fun s=>match s with (l,i,s,_) => l=l1 ∧ i=0 ∧ s=0 end.
Definition r:= fun s=>match s with (l,i,s,m) => l=l1 ∧ i=m ∧ 2*s=i*(i+1) end.
```

7.1.4. The Knaster-Tarski Theorem for `reach`

For proving implications whose right-hand side is `reach`, such as validity (above), one can use the instance of the Knaster-Tarski Theorem for `reach` that we describe below. Other instances of the theorem are used later in the paper and obtained in a similar way. The Knaster-Tarski theorem here refers to a function whose greatest post-fixpoint is `reach`. The first step is to define that function from the definition of `reach`. Since `reach` has type `seq -> SymState-> Prop`, the function whose greatest post-fixpoint it is, denoted by convention below by `_reach`, has type `(seq -> SymState-> Prop) -> (seq -> SymState-> Prop)`, and is inductively defined as follows:

```
Inductive _reach(X:Seq -> SymState-> Prop):Seq -> SymState-> Prop:=
  |_reach_now_one:  forall r s, r s-> _reach X (one s) r
  |_reach_now_add:  forall s tau, r s-> _reach X (add s tau) r
  |_reach_later:  forall s tau, X tau r -> _reach X (add s tau) r
```

The transformation from `reach` to `_reach` is syntactical:

- `CoInductive` in `reach` becomes `Inductive` in `_reach`;
- `_reach` gets an argument, here, `X`, of the same type `seq -> SymState-> Prop` as `reach`, and its "return" type is the same type `seq -> SymState-> Prop`;
- the constructors of `reach` are translated into homonymous constructors of `_reach`, except for an additional "`_`" prefix;
- in the return types of the constructors, `reach` is replaced by `_reach X`;
- recursive calls to `reach` in the constructors are replaced by calls to `X`.

Although this is not used in the sequel, as a sanity check we can prove that `reach` is a post-fixpoint of `_reach` (with the order relation between any two `X` and `Y` of type `seq -> SymState-> Prop` defined by `forall tau r, X tau r -> Y tau r`):

```
Lemma postfixpoint_reach: forall tau s,reach tau s -> _reach reach tau s
```

The above lemma is proved by case analysis on the constructor of `reach`, and in each case, by applying the corresponding constructor of `_reach`. We can now state the instance of the Knaster-Tarski theorem for `reach`:

```
Lemma tarski_reach:  forall (R: seq -> SymState -> Prop),
  (forall tau r, R tau r -> _reach R tau r) ->
  forall tau r, R tau r -> reach tau r.
```

That is, `reach` is the greatest postfixpoint of `_reach` with respect to the given order.

We now prove the `tarski_reach` lemma. Since we did not develop a general framework of monotone functions over a lattice in Coq and a general Knaster-Tarski theorem in such a setting (which would have been a *deep embedding* approach) we resort to *shallow embedding*: proving each such instance of the theorem using the available Coq builtin mechanisms for coinductive reasoning. The Coq proof is explained below.

```
Proof.
cofix tarski_reach.(*copy lemma's statement in ''coinductive hypothesis''*)
intros R Hcoind tau r HR.(*introduce the quantified entities in hypotheses*)
destruct (Hcoind _ _ HR).(*case analysis on constructors of _reach X tau r*)
(*each case, identified by a - below, is solved by a constructor of reach*)
-constructor 1; auto.
-constructor 2; auto.
-econstructor 3 ; eauto.(* here the coinductive hypothesis has been used.*)
Qed.(* Success!  Coq accepts the proof term generated by the above script *)
```

The main ideas of the proof are given in the above *proof script* as comments, enclosed within `(* *)`. Some more explanations follow. The `cofix tarski_reach` command makes the lemma's statement, which is the proof's initial (desired) *c*onclusion, also available as a *coinductive hypothesis*, and also called `tarski_reach`. That new hypothesis only becomes available later. Otherwise, one could immediately use it to prove the lemma, and this would work for any statement (valid of not) about coinductive entities, which is unsound. Note the similarity with the rule [Cof] of our third proof system.

The next proof steps are standard logical manipulations in the proof assistant, which have the effect that one obtains a new goal with a hypothesis `Hco: _reach R tau r` and the conclusion `reach tau r`. The `destruct` tactic applied to that hypothesis generates a case analysis according to the three constructors of `_reach`. Each of those cases is solved by applying the corresponding constructor of `reach` as a lemma, followed by Coq automatic tactic `auto`. In particular, in the last case, this leads to automatically applying the coinductive hypothesis `tarski_reach`. That the proof is accepted by Coq means that the proof script has built a well-formed corecursive program, in which the (unique) recursive call to `tarski_reach` is, syntactically, directly under a constructor of `reach`. We say that the proof program (more accurately, proof *term*) generated by the given script is *syntactically guarded by constructors*. When this is not the case, Coq rejects the proof: it cannot go past the `Qed` command at the end of the proof script.

The above technique generalises to other instances of the Knaster-Tarski theorem. We use them for proving coinductive statements - including the soundness and completeness of proof systems for RL, discussed in the next section. This technique enables one to write Coq proofs that are close to the corresponding paper proofs (that use the same instances of the Knaster-Tarski theorem). Moreover, it enables one to write Coq proofs that are accepted by the proof assistant, whereas attempts directly using `cofix` are rejected due to violation of the condition of the proof term being syntactically guarded by constructors. Examples of such situations are given hereafter.

7.2. First Proof System

The one-rule RL proof system, denoted by $\vdash$ and shown in Figure 2, is defined in Coq as the coinductive predicate `proof` having only one constructor `Stp`:

```
CoInductive proof:  Rlf -> Prop :=
|Stp:  forall (l l' r:  SymState),
  imp l (dsj l' r) ->
  imp (cnj l' final) bot ->
  proof (symTrans l' =><> r) -> proof (l =><> r).
```

The conditions `imp l (dsj l' r)` and `imp (cnj l' final) bot` translate the side-conditions $l \sqsubseteq (l' \sqcup r)$ and $l' \sqcap \bullet \sqsubseteq \bot$, while `proof (l =><> r)` and `proof (symTrans l' =><> r)` are, respectively, the conclusion and the hypothesis of the [Stp] rule.

Soundness. The soundness of $\vdash$ is stated as Theorem 2, which translates to Coq as:

```
Theorem soundness:  forall f, proof f -> valid f.
```

In the proof of Theorem 2 we have used the instance of the Knaster-Tarski theorem for $\rightsquigarrow$, i.e., Lemma 1. The key step in the Coq proof does, *mutatis mutandis*, the same:

```
apply tarski_reach with (R:=
fun tau r => exists l,proof (l =><> r) ∧ isPath tau ∧ l (head tau))
```

That is, the Coq instance of the Knaster-Tarski theorem for `reach` is applied, with the same instance for the parameter `R` as that chosen for R in Theorem 2. We do not detail here the rest of the Coq proof, but that too mimics the paper proof, i.e., it has the same decomposition into cases. Hence, thanks to the Coq Knaster-Tarski framework we obtain a Coq proof that is essentially the same as the paper proof.

This is in contrast with a previous soundness proof for the $\vdash$ proof system [16], where the Coq proof directly used the builtin `cofix` mechanism, and the paper proof tried to mimic that mechanism. Since `cofix` requires that proof terms being built meet the syntactical guardedness condition, in principle we had to check that condition in the paper proof as well, but we did not actually do it since it is too complex. Hence, the earlier paper version of the soundness proof was not completely satisfying.

Canonical instantiation. Each instance of the Knaster-Tarski theorem is parameterised by a predicate (e.g., `R` in `tarski_reach`), which still needs to be instantiated when the theorem is used in the proof of a coinductive statement. In general this instantiation requires creativity from the user, but there is a *canonical* one that can be generated from the statement (or *goal*, in Coq) whose proof uses the Knaster-Tarski theorem.

In general Coq goals (and goals in other proof assistants such as Isabelle/HOL) can be written as $\forall X.(H_1 \to \cdots \to H_n \to C)$, where $H_1, \ldots, H_n$ are the goal's *hypotheses*, C is the *conclusion*, and X is the set of *free variables* occurring in them. We denote by $V(F)$ the set of free variables occurring in a formula F, thus, $X = V(H_1) \cup \cdots \cup H_n \cup C$.

We are specifically interested in *coinductive* goals, where the conclusion C is the greatest fixpoint νF of a function $F : (T \to Prop) \to (T \to Prop)$, i.e., F takes a predicate on some type T and also returns a predicate on the same type T. Thus, we are interested in proving coinductive goals of the form $\forall X.(H_1 \to \cdots \to H_n \to \nu F)$.

Let us assume for now that (a) $V(\nu F) \subseteq V(H_1) \cup \cdots \cup V(H_n)$, (b) the formula νF in the goal is a predicate directly applied to variables, and (c) the formula νF in the goal

is *linear*, i.e., each variable in it occurs exactly once. These conditions are required because they enable the application of the Coq version of the Knaster-Tarski theorem for F (see below). We will also show below how goals that do not satisfy (a), (b), (c) can be equivalently transformed into goals that satisfy these properties.

The Coq version of the Knaster-Tarski theorem for $F : (T \to Prop) \to (T \to Prop)$ states that, for any predicate $R : T \to Prop$ with $V(R) = V(\nu F)$, in order to prove $\forall V(R).(R \to \nu F)$ it is enough to prove $\forall V(R).(R \to F(R))$. Here, νF has to satisfy the conditions $V(R) = V(\nu F)$ as well as (b) and (c) above, because under these conditions the implications $\forall V(R).(R \to \nu F)$ and $\forall V(R).(R \to F(R))$ encode the inclusions $R \sqsubseteq \nu F$ and $R \sqsubseteq F(R)$ from the general formulation of the Knaster-Tarski theorem.

When this theorem is applied in the proof a given coinductive goal $\forall X.(H_1 \to \cdots \to H_n \to \nu F)$ an instance for R has to be provided. In order to come up with the instance we equivalently transform the coinductive goal as follows. Let H denote the conjunction $H_1 \wedge \cdots \wedge H_n$. Then the coinductive goal can be equivalently written $\forall V(\nu F).((\exists (V(H) \setminus V(\nu F)).H) \to \nu F)$, that is, the initial chain of implications between hypotheses has been transformed into a conjunction and variables have been moved closest the conjunction with their quantifiers appropriately changed.

The canonical instance for R is $(\exists (V(H) \setminus V(\nu F)).H)$. From (a) above we have that $V(\nu F) \subseteq V(H))$, thus, $V(R) = V(\nu F)$. We have assumed (b) and (c) as well regarding the formula νF in our goal, thus, based on the Knaster-Tarski theorem, if we prove $\forall V(R).(R \to F(R))$ for the chosen R than we have also proved our coinductive goal.

Example 12. *The* `soundness` *theorem above :* `forall f,prove f -> valid f` *becomes, after unfolding the definition of* `valid` *and decomposing* `f` *as* `l =><> r`*:*

```
forall l r tau, prove(l =><> r) -> isPath tau -> l(head tau) -> reach tau r
```

The above formula satisfies conditions (a), (b), (c). We transform it as prescribed into

```
forall r tau,
  (exists l, prove(l =><> r) ∧ isPath tau ∧ l(head tau)) -> reach tau r
```

from which we extract the canonical instance of `R` *in the application of* `tarski_reach`*:*

```
fun tau r => exists l, prove(l =><> r) ∧ isPath tau ∧ l(head tau)
```

Of course, $\forall V(R).(R \to F(R))$ is not guaranteed to hold for the canonical R; one may have to come up with an instance R' that is *weaker* than R. Given that the formula H occurring in R is a conjunction $H_1 \wedge \cdots \wedge H_n$, one way to achieve this is to keep only a subset of the formulas in the conjunction. This worked for all but the last application of Knaster-Tarski theorem shown in this section, and it also worked for many other cases.

Thus, when proving a coinductive statement, the canonical instance of predicates is obtained from the statement's hypotheses, with existential quantification of some of their variables. This is dual to what happens with induction: in a proof by induction one applies an *induction principle*, and canonical instances are obtained from the statement's conclusion, generalised by universally quantifying some of its variables. For induction, Coq automatically generates and proves such principles and provides them with canonical instantiations when they are applied. Thus, our framework for coinduction provides Coq with support that is similar (and dual) to that provided by Coq for induction. The difference is that our approach is not yet automatic. It is, however,

systematic, and can be implemented using a Coq parser for generating instances of the Knaster-Tarski theorem and the Coq tactic language *Ltac* for the remaining transformations. Those include the following ones, which turn a general coinductive goal $\forall X.(H_1 \to \cdots \to H_n \to \nu F)$ into one satisfying the conditions (a), (b) and (c) above.

We first denote $H \triangleq H_1 \wedge \cdots \wedge H_n$ and rewrite the goal as $\forall X.(H \to \nu F)$.

- eliminating variables in $V(\nu F) \setminus V(H)$: for each $x \in V(\nu F) \setminus V(H)$ the goal $\forall X.(H \to \nu F)$ is transformed into $\forall(X \cup \{x'\}).(H \wedge (x' = x) \to \nu F[x'/x])$, where $x' \notin X$ and $\nu F[x'/x]$ denotes the replacement of every occurrence of x by x' in νF. After this transformation we are left with goals satisfying (a).

- eliminating non-variable terms in the conclusion: while the resulting goal can be written as $\forall X.(H \to C[t/x])$ for some non-variable term t, it is transformed into $\forall(X \cup \{x'\}).(H \wedge (x' = t) \to C[x'/x])$, where $x' \notin X$. After the first two transformations we are left with goals satisfying (a) and (b).

- linearising the conclusion: while the resulting goal $\forall X.(H \to C)$ contains two instances of the same variable x: let C' denote the formula C in which one copy of x is replaced by some $x' \notin X$. The goal becomes $\forall(X \cup \{x'\}).(H \wedge (x' = t) \to C')$. After the three transformations we are left with goals satisfying (a), (b) and (c). This completes the paragraph on the canonical instantiation.

Completeness. One key result for completeness is Lemma 5, which uses an instance of Knaster-Tarski's theorem for $\vdash$ (Lemma 4). We thus follow the approach outlined in Section 7.1.4 in order to state and proves the Coq version of Lemma 4, i.e., the Knaster-Tarski theorem for `proof`. The first step is to define the functional:

```
Inductive _proof(X: Rlf -> Prop) :  Rlf -> Prop :=
  |_Stp:  forall (l l' r:  SymState),
  imp l (dsj l' r) ->
  imp (cnj l' final) bot ->
  X (symTrans l' =><> r) ->
  _proof X (l =><> r).
```

Then, the Knaster-Tarski theorem for `proof` is

```
Lemma tarski_proof:  forall (X: Rlf -> Prop),
  (forall f, X f -> _proof X f) ->
  forall f, X f -> proof f.
```

It is proved using `cofix` as shown in Section 7.1.4. The Coq version of Lemma 5:

```
Lemma strategy:  forall f q, imp (lhs f)(dsj q (rhs f)) ->
  imp (cnj q final) bot -> imp (cnj q final) bot -> proof f
```

is proved using `tarski_proof` with the same instantiation for `X` as that used in the proof of Lemma 5. It is also the canonical instantiation as described in an earlier paragraph.

```
apply tarski_proof with (X := fun f => exists l', imp (lhs f) l' ∧
imp l'(dsj q (rhs f)) ∧ imp (cnj q final) bot ∧ imp (symTrans q) (dsj q (rhs f)))
```

After this step the proof of lemma `strategy` follows the same structure as that of Lemma 5. This is, again, in contrast with an earlier approach [16] where `strategy`

was proved directly using `cofix` and the paper proof tried to mimic the Coq proof, with incomplete justification as to why the resulting proof is acceptable.

Finally, the Coq version of the completeness result (Theorem 3) is based on finding `q` such that, for valid formulas, the preconditions of applying `strategy` are satisfied. The same `q` is used in both versions, and the structures of the proofs are the same.

7.3. Second Proof System

Our second proof system (Figure 3) has a mixed inductive-coinductive nature. Defining it in Coq and proving its soundness and completeness has been quite challenging. Carefully choosing its definition, and using the Knaster-Tarski theorem, are essential here, as direct proofs with `cofix` more often than not result in proof terms that are not syntactically guarded and are rejected by Coq. The approach that worked best (compared to our other attempts) is a coinductive predicate, "bounded" by a natural number that "forces" the inductive rules to be applied finitely many times only.

```
CoInductive proof (H: Rlf -> Prop):  nat -> Rlf -> Prop :=
  |Hyp:  forall n f, H f -> proof H n f
  |Trv:  forall n r, proof H n (r =><> r)
  |Str:  forall n l l' r, imp l l'-> proof H n (l' =><> r)->
    proof H (S n) (l =><> r)
  |Spl:  forall n l1 l2 r, proof H n (l1 =><> r) ->
    proof H n (l2 =><> r) -> proof H (S n) ((dsj l1 l2) =><> r)
  |Tra:  forall n l m r, valid (l =><> m)-> proof H n (m =><> r)->
    proof H (S n) (l =><> r)
  |Stp:  forall n k l r, proof H n (symTrans l =><> r)->
    imp (cnj l final) bot -> proof H k (l =><> r).
```

The first two rules are not recursive, so the natural number `n` plays a passive role. However, in the rules `Str`, `Spl`, and `Tra`, the number increases, from `n` to its successor `S n`. How far it can increase is determined by the last rule, `Stp`, which says that in order to have a proof of `l =><> r` bounded by some `k`, it is enough to produce a proof `symTrans l =><> r`, bounded by `n`. In this way, the proof system runs as a possibly infinite number of "phases", separated by `Stp`, which "launches" each new phase with arbitrarily large but (of course) finite natural-number bound `n`. This constrains the other rules to be applied finitely many times in each phase.

Hence, using an additional natural-number parameter, we obtain the desired mixture of induction and coinduction. The only disadvantage is that the new parameter will occur everywhere in the proofs. We thus slightly depart from the presentation of the proof system; denoted $\Vdash$ in Section 5, where the natural number does not occur. Our attempts at alternative approaches that do not include the natural number have failed due to the already mentioned expressiveness limitations of Coq's `cofix` tactic.

Soundness. The soundness for this proof system is Theorem 5, translated to Coq as

```
Theorem soundness:  forall n H, (forall f, H f -> valid f) ->
  forall f, proof H n f -> valid f
```

We first note that a direct proof of soundness using `cofix` builds a proof term that is not syntactically guarded by constructors and is rejected by Coq. The main reason

is that such a proof needs both a coinductive hypothesis, in order to prove a subgoal dealing with the (coinductive) `Stp` rule, and an inductive one, for dealing with the rules that have been "forced into induction". In the resulting proof term, the coinductive hypothesis is not directly under a constructor, but under the induction principle for the natural number. Coq rejects this proof term since it is not syntactically guarded by constructors.

Fortunately, using our Knaster-Tarski framework we succeeded in proving soundness. First, an instance of the Knaster-Tarski theorem is proved for a semantically equivalent yet more convenient variant of the relation `reach`, denoted by $\hookrightarrow$ in Section 5 and by `reach'` hereafter in the Coq code. We denote by `_reach'` the corresponding functional. The Knaster-Tarski theorem for `reach'` is, again, proved using `cofix`:

```
Lemma tarski_reach':  forall (R: seq -> SymState -> Prop),
  (forall tau r, R tau r -> _reach' R tau r) ->
  forall tau r, R tau r -> reach' tau r.
```

In order to prove our soundness theorem we apply the `tarski_reach'` lemma, with the predicate `R` instantiated to a value that is (up to the natural number `n`) the same as that for R in Theorem 5, and is also the canonical instantiation in this setting:

```
apply tarski_reach' with (X:= fun tau r => exists (l:SymState) (n:nat),
  proof H n (l =><> r) ∧ l(head tau) ∧ isPath tau ∧ isFinite tau).
```

After this application there is an induction on `n`, much like the inner induction in the proof of Theorem 5, and Coq eventually accepts the resulting proof term.

The main difference with the failed attempt that used `cofix` directly is that, now, `cofix` is safely "encapsulated" in the proof of the `tarski_reach'` lemma, which is a well-formed and accepted by Coq. Then, unlike what happened in the direct attempt, the induction principle "calls" the `tarski_reach'` lemma, *not* the inner call to a coinductive hypothesis. Hence the syntactical guardedness of the resulting proof term.

The technique of hiding `cofix` into other proofs in order to prevent it from generating unguarded proof terms is general and might be of interest to other Coq users.

Completeness. Like for the first proof system, completeness requires a proof of a lemma (Lemma 7) giving a strategy for reducing reachability to invariance:

```
Lemma strategy:  forall f q, imp (lhs f)(dsj q (rhs f)) ->
  imp (cnj q final) bot -> imp (cnj q final) bot ->
    exists n, proof (fun _ => False) n f
```

That is, under the usual assumptions about a formula `f` and a state predicate `q`, a proof of `f` bounded by some natural number `n` can be found, with an empty set of hypotheses. To prove this lemma we derive from the definition of `proof` its corresponding functional `_proof`. The instance of Knaster-Tarski for `proof` is, again, proved using `cofix`.

```
Lemma tarski_proof:  forall(F:Rlf -> Prop)(Y: nat -> Rlf -> Prop),
  (forall n f, Y n f -> _proof H Y n f) ->
  forall n f, Y n f -> proof H X n f.
```

What enables us to obtain a proof term accepted by Coq, is that, unlike what happened in our first (and failed) proof attempt of soundness, here there is no need to perform an induction on `n` in order to prove the subgoals related to the rules `Str`, `Spl`, and `Tra`.

The chosen definition for `proof` enables us to consider those rules as either inductive or coinductive, depending on what is more convenient in a given context. Here, we see them as coinductive, and use the coinductive hypothesis in all the cases.

The proof of `strategy` involves, like before, applying the Knaster-Tarski theorem for `proof`, with the parameter `Y` instantiated to essentially the set of formulas Y from the corresponding Lemma 7. However, each formula in the set is paired with a natural number, which can be chosen to be the number of rule applications until `Stp` is applied, or some arbitrary number for the formulas whose proof does not involve `Stp`:

```
apply tarski_proof with (F:= fun n f =>
    (n = 3 ∧ f = (l =><> r)) ∨
    (n = 2 ∧ f= ((dsj q r) =><> r)) ∨
    (n = 3 ∧ f = (symTrans q =><> r)) ∨
    (n = 1 ∧ f = (q =><> r)) ∨
    (n = 1 ∧ f = (r =><> r))).
```

The natural number comes in as somewhat unhandy in proofs, but, on the other hand, including it was essential for obtaining the soundness and completeness results without which such proofs do not really have any meaning. We have also tried other solutions, including one that does not require a natural-number parameter, but requires the proof system to be defined by *parametric coinduction* implemented in the Coq package *Paco* [19]. That solution works, but it is based on other (stronger) theorems than Knaster-Tarski, leading to differences with respect to our paper proof. It also makes our code dependent on an external package, which may or may not keep up with Coq.

Whatever the solution for proving `strategy`, the proof of the completeness theorem uses `strategy` and is the same as that shown earlier for the first proof system.

7.4. Third Proof System

The third proof system (cf. Fig. 4) is inductive and thus naturally encoded as an inductive relation `proof`, of type `list (bool*Rlf) -> bool*Rlf -> Type`. We do not give it here due to space constraints. With respect to the paper version we use lists (instead of arbitrary sets) of hypotheses, here, consisting of a Boolean and an RL formula. One may also note that the relation `proof` is not in `Prop` but in `Type`. This is because we need to perform computations on proofs, which, in Coq, cannot be done with terms of type `Prop`. For example, proofs are trees, and we need to compute their subtrees.

The main challenge for the third proof system has been to define the domain (denoted by $\mathcal{D}$ in Section 6) and to equip it with a well-founded order, on which the soundness proof is based. The domain $\mathcal{D}$ has been defined as a set $\mathcal{D} \triangleq \{(\tau', b', \varphi') \in \mathit{fPaths} \times \mathbb{B} \times \Phi \mid (\mathit{lhs}\,\varphi')(\mathit{hd}\,\tau') \wedge (b', \varphi') \in \mathsf{Con}\}$, where Con is the set of all formulas occurring as conclusions of goals in the proof of a given formula. In Coq, these dependencies (e.g., $\mathcal{D}$ depends on Con, which depends on the proof of a formula) are made explicit using Coq's dependent types. This leads to quite intricate types and terms. For example, the Coq implementation of the domain $\mathcal{D}$, denoted by `dom`, is written as

```
dom := fun (H:list (bool*Rlf)) (f:bool*Rlf) (p:proof H f) =>
{tau:seq &{g:goals H f p & isPath tau & lhs(getForm H f p g)(head tau)}}
```

That is, the term `dom` depends on hypotheses `H`, a formula `f`, a proof-tree `p` of `f` under `H`, and consists of a path `tau` and a term `g` of type `goals H f p`, i.e., a subtree of `p` such that the path "starts" in the left-hand side of the formula at the root of the subtree.

Working with such dependent types is unavoidable here, and it is feasible because the proof assistant keeps tracks of all the dependencies and rejects incorrectly-typed terms. On paper, however, this quickly becomes unreadable. There, the dependencies were left implicit, and as any implicit assumptions they may lead to errors.

The dependent-type formulation also has some practical advantages for equipping `dom` with a well-founded order. Two elements `(tau1,g1)` and `(tau2,g2)` of the domain are ordered if either `tau1` is shorter than `tau2`, or if the paths have the same length but the boolean extracted from the root of `g1` is less than the one extracted from the root of `g2`, or if both path lengths and booleans are equal but `g1` is a strict subtree of `g2`. This lexicographic order is well-founded, in particular, because the "strict subtree" relation is well-founded. By contrast, in the order $\prec$ on the domain $\mathcal{D}$, it is not trees but formulas (in the set Con) that need to be given a well-founded order, and this is impossible since a formula in Con may occur in its own proof. This is why we defined the last occurrence of a formula in a proof and worked with last occurrences only.

Modulo those differences, the soundness and completeness proofs in Coq follow the general structure given in Section 6. The differences have been carefully introduced to keep an already complex paper proof readable while benefiting from Coq's dependent types to clarify the implicit dependencies and the assumptions in the paper proofs.

7.5. Compositionality

The structural compositionality and (where relevant) its interaction with logical compositionality of proof systems have also been implemented. The definition of component (Definition 3) is about two transition systems, which implies two versions of reachability logic and two copies of the proof systems. To avoid code duplication in Coq we place the code for reachability logic and the proof systems into *modules*, which are *parameterised* by a given transition system. Thus, the code for reachability logic is in `Module ReachabilityLogic(TS: SigTSMod)`, that is, a module called `ReachabilityLogic`, parameterised by a transition system `TS` of *module type* `SigTSMod`. The module type `SigTSMod` declares the parameters related to transition systems:

```
Module Type SigTSMod.
  Parameter State:  Type.
  Parameter trans:  State -> State -> Prop.
  Parameter final :  State -> Prop.
  Axiom final_is_final :  forall s, final s <-> forall s', ~trans s s'.
End SigTSMod.
```

Thus, parameters for the type of states, for the transition relation, and for the final states are given, together with an axiom characterising the final states. Any given transition system implemented in Coq, such as that described in Example 10, will be written as a module of module type `SigTSMod`, and contain definitions and theorems implementing (and homonymous to) the respective parameters and axioms of the module type.

Going back to Definition 3 of a component, we notice that it requires two sets of states that are in the *subset* relation. In Coq there is no primitive notion of set; we

have used types for states instead, but Coq has no *subtypes* corresponding to subsets, inhabited by some but possibly not all inhabitants of a given type.

To deal with this problem we generalise the notion of set inclusion, by requiring that there is an *injective function* between two sets. This is convenient for Coq, but not in the paper version since it would complicate notations without any actual benefit.

Since we chose to implement transition systems as instances of the module type `SigTSMod`, we implement Definition 3 of components as a module type as well, *parameterised* by the two transition systems that are in the "component" relation:

```
Module Type SigCompMod(TS:SigTSMod)(TS':SigTSMod).
  Parameter state_map :  TS'.State -> TS.State.
  Axiom state_map_inj:  forall s'1 s'2,
    state_map s'1 = state_map s'2 -> s'1 = s'2.
  Axiom subtrans:  forall s'1 s'2,
    TS'.trans s'1 s'2 -> TS.trans (state_map s'1) (state_map s'2).
  Axiom full_subtrans:  forall s1 s2 s'1 s'2,
    TS.trans s1 s2 -> state_map s'1 = s1 -> state_map s'2 = s2 ->
      TS'.trans (s'1) (s'2).
  Axiom clean_border :  forall (s'1:TS'.State) (s2:TS.State),
  TS.trans(state_map s'1) s2 -> ~exists s'2, s2 = state_map s'2 -> TS'.final s'1.
End SigCompMod.
```

The module type `SigCompMod` is parameterised by transition systems `TS` and `TS'`. Their states and transitions are, respectively, `TS.State` , `TS.trans`,`TS'.State` , `TS'.trans`. The parameter `state_map` is the above-mentioned function between types `TS'.State` and `TS.State`, and is declared to be injective in the axiom `state_map_inj`. The three remaining axioms correspond to the three conditions in Definition 3 for `TS'` to be a component of `TS'`, using the injection `state_map` instead of subset relations.

Example 13. *We show how the component in Example 4 is encoded in Coq. We first define two instances of* `SigTSComp`*, one for the component and one for the full transition system. For the component, let* `State:= nat*nat*nat` *and* `trans` *be defined by*

```
Inductive trans := loop: forall i m s, i<m -> trans(i,s,m)(i+1,s+i+1,m).
```

The encoding of the full transition system in Coq has been given in Example 10. In particular, note that the state-space of the full transition system has an additional component, which is a control location. Then, we define a module `Comp` *of type* `SigCompMod` *with parameters* `TS, TS'` *set to modules containing the above-defined transition systems, say, modules* `T` *and* `C`*. The function* `state_map` *sends each state* `(i,s,m)` *from the component* `C` *to* `(c1,i,s,m)` *of the full transition system* `T`*. Finally,* `Comp` *contains proofs of the injectiveness of* `state_map` *and of the three conditions for* `C` *to be a component of* `T`*.*

The Coq equivalent of Theorem 1 about validity of a formula on a component implying the validity of the formula on the full transition system is written in a separate module, an excerpt of which is shown below:

```
Module StructCompMod(TS:SigTSMod)(TS':SigTSMod)(Comp:SigCompMod TS TS').
  Module RL := ReachabilityLogicMod TS.
```

```
Module RL' := ReachabilityLogicMod TS'.
Definition symState_map :  RL'.SymState -> RL.SymState :=
  fun (p:RL'.SymState)(s:TS.State) => exists s',p s' ∧ (s = state_map s').
Theorem ValidComp: forall (l' r':RL'.SymState), RL'.valid (l'=><>r') ->
  RL.valid ((symState_map l')=><>(symState_map r')).
```

The above module has two transition systems as parameters, and a third parameter providing evidence that the second transition system is a component of the first one. It starts by defining modules for reachability logic instantiated to the two transition systems. Then, the function mapping states of the second transition system to states of the first one is lifted to state predicates. Finally, the theorem states the expected result: validity of a formula in the component implies validity of the formula in the whole transition system, after suitably mapping the left and right-hand side state predicates.

Example 14. *One can use the theorem* `ValidComp` *to reduce the validity of the formula* `l =><> r` *from Example 11 to that of* `l' =><> r'` *on the "loop" component, with*

```
Definition l' := fun s => match s' with (_,i,s) => i = 0 ∧ s = 0 end.
Definition r' := fun s => match s with (m,i,s) => i = m ∧ 2*s = i*(i+1) end.
```

A similar development has been made around the compositionality of the third proof system (Theorem 10), only somewhat more involved because there are now three transition systems: two smaller ones, which are components of one larger transition system. Example 9 for computing greatest common divisors has also been implemented.

For such small examples the benefits of compositional verification are, of course, not visible. We expect, however, that it pays off on larger examples, such as the hypervisor that we verified non-compositionally in [16], which naturally decomposes into components corresponding to machine code-analysis and machine-code execution.

8. Conclusions and Future Work

For each of the main contributions we draw some conclusions and suggest future work.

Proof systems. We have noted a correlation between the amount of induction, the degree of compositionality, and the difficulty of soundness proofs in our three proof systems. This correlation can be explained by the fact that the proof systems are also increasingly closer to our specific problem domain - verifying RL formulas on transition systems. The first, one-rule proof system is a rather generic search procedure in the infinite tree of symbolic executions generated by a transition system. In the second proof system, specific properties of RL are encoded into inductive rules. The third proof system adds to those new rules a coinduction principle for our problem domain. A proof system more specific to our domain requires more rules and therefore has more flexibility to accommodate compositional reasoning, but also more cases to consider in soundness proofs. Now, on the practical side, while the first proof system is convenient to prove meta-results, it is too generic to be used. The most practical is the third one, while the second one is in-between. That is why in future work we are planning to further use the third, most compositional proof system to demonstrate that compositional verification is effective. Another possible direction for future work could be to

investigate to what extent our proof systems are instances of the approach proposed in [14]. This would provide with an alternative (indirect) proof of their soundness and completeness.

Compositionality. We have proposed both structural compositionality (for transition systems) and logical compositionality (for RL formulas). Of those, structural compositionality currently requires the user to identify components and to prove certain conditions ensuring the "is-component-of" relation. We have presented in the Coq implementation a way to represent transition systems and their tentative components, with proof obligations ensuring that the said relation holds. The current component-based *verification* is a top-down approach: steps in the proofs of a global RL formula are "localised" and proved on components. We are planning to explore a bottom-up approach: component-based *design*, where a global transition system would be a set of components, combined using certain operations ensuring the "is-component-of" relation by construction, and each component would be characterised by a set of local RL formulas. The verification of a global RL formula would then be built on top of the local formulas, ignoring details about why the local formulas hold on the components.

Knaster-Tarski framework in Coq. Part of our Coq implementation is a framework based on the Knaster-Tarski theorem for proving coinductive statements. The framework serves both for writing Coq proofs that are similar to the paper proofs (for soundness and completeness of the proof systems), but also as a means to obtain Coq proofs when the builtin mechanisms for Coq coinductive reasoning do not work directly. In our approach those mechanisms are still used, but indirectly, wrapped into proofs of Knaster-Tarski theorems, where the syntactical conditions that they require are met. We have shown that the approach is systematic, amenable to automation, and that it offers to proofs by coinduction a level of support close to that offered by Coq for proofs by induction. For future work we are considering automating the approach in Coq.

Acknowledgment. We acknowledge the support of the CNRS-JSPS Joint Research Project "FoRmal tools for IoT sEcurity" (PRC2199), and thank all the participants of this project for fruitful discussions.

References

[1] A. Stefanescu, Ş. Ciobâcă, R. Mereuta, B. M. Moore, T. Serbanuta, G. Rosu, All-path reachability logic, Log. Methods Comput. Sci. 15 (2) (2019). doi:10.23638/LMCS-15(2:5)2019.

[2] C. A. R. Hoare, An axiomatic basis for computer programming, Commun. ACM 12 (10) (1969) 576–580. doi:10.1145/363235.363259.

[3] P. W. O'Hearn, Separation logic, Commun. ACM 62 (2) (2019) 86–95. doi:10.1145/3211968.

[4] G. Rosu, A. Stefanescu, Ş. Ciobâcă, B. M. Moore, One-path reachability logic, in: 28th Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2013, New Orleans, LA, USA, June 25-28, 2013, IEEE Computer Society, 2013, pp. 358–367. doi:10.1109/LICS.2013.42.

[5] A. Stefanescu, D. Park, S. Yuwen, Y. Li, G. Rosu, Semantics-based program verifiers for all languages, in: E. Visser, Y. Smaragdakis (Eds.), Proceedings of the 2016 ACM SIGPLAN International Conference on Object-Oriented Programming, Systems, Languages, and Applications, OOPSLA 2016, part of SPLASH 2016, Amsterdam, The Netherlands, October 30 - November 4, 2016, ACM, 2016, pp. 74–91. doi:10.1145/2983990.2984027.

[6] D. Lucanu, V. Rusu, A. Arusoaie, D. Nowak, Verifying reachability-logic properties on rewriting-logic specifications, in: N. Martí-Oliet, P. C. Ölveczky, C. L. Talcott (Eds.), Logic, Rewriting, and Concurrency - Essays dedicated to José Meseguer on the Occasion of His 65th Birthday, Vol. 9200 of Lecture Notes in Computer Science, Springer, 2015, pp. 451–474. doi:10.1007/978-3-319-23165-5_21.

[7] S. Skeirik, A. Stefanescu, J. Meseguer, A constructor-based reachability logic for rewrite theories, in: F. Fioravanti, J. P. Gallagher (Eds.), Logic-Based Program Synthesis and Transformation - 27th International Symposium, LOPSTR 2017, Namur, Belgium, October 10-12, 2017, Revised Selected Papers, Vol. 10855 of Lecture Notes in Computer Science, Springer, 2017, pp. 201–217. doi:10.1007/978-3-319-94460-9_12.

[8] V. Rusu, G. Grimaud, M. Hauspie, Proving partial-correctness and invariance properties of transition-system models, in: J. Pang, C. Zhang, J. He, J. Weng (Eds.), 2018 International Symposium on Theoretical Aspects of Software Engineering, TASE 2018, Guangzhou, China, August 29-31, 2018, IEEE Computer Society, 2018, pp. 60–67. doi:10.1109/TASE.2018.00016.

[9] Y. Bertot, P. Castéran, Interactive Theorem Proving and Program Development - Coq'Art: The Calculus of Inductive Constructions, Texts in Theoretical Computer Science. An EATCS Series, Springer, 2004. doi:10.1007/978-3-662-07964-5.

[10] V. Rusu, D. Nowak, (Co)inductive proof systems for compositional proofs in reachability logic, in: M. Marin, A. Craciun (Eds.), Proceedings Third Symposium on Working Formal Methods, FROM 2019, Timişoara, Romania, 3-5 September 2019, Vol. 303 of EPTCS, 2019, pp. 32–47. doi:10.4204/EPTCS.303.3.

[11] T. Nipkow, L. C. Paulson, M. Wenzel, Isabelle/HOL - A Proof Assistant for Higher-Order Logic, Vol. 2283 of Lecture Notes in Computer Science, Springer, 2002. doi:10.1007/3-540-45949-9.

[12] Ş. Ciobâcă, D. Lucanu, A coinductive approach to proving reachability properties in logically constrained term rewriting systems, in: D. Galmiche, S. Schulz,

R. Sebastiani (Eds.), Automated Reasoning - 9th International Joint Conference, IJCAR 2018, Held as Part of the Federated Logic Conference, FloC 2018, Oxford, UK, July 14-17, 2018, Proceedings, Vol. 10900 of Lecture Notes in Computer Science, Springer, 2018, pp. 295–311. doi:10.1007/978-3-319-94205-6_20.

[13] D. Lucanu, V. Rusu, A. Arusoaie, A generic framework for symbolic execution: A coinductive approach, J. Symb. Comput. 80 (2017) 125–163. doi:10.1016/j.jsc.2016.07.012.

[14] B. M. Moore, L. Peña, G. Rosu, Program verification by coinduction, in: A. Ahmed (Ed.), Programming Languages and Systems - 27th European Symposium on Programming, ESOP 2018, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2018, Thessaloniki, Greece, April 14-20, 2018, Proceedings, Vol. 10801 of Lecture Notes in Computer Science, Springer, 2018, pp. 589–618. doi:10.1007/978-3-319-89884-1_21.

[15] E. Giménez, Codifying guarded definitions with recursive schemes, in: P. Dybjer, B. Nordström, J. M. Smith (Eds.), Types for Proofs and Programs, International Workshop TYPES'94, Båstad, Sweden, June 6-10, 1994, Selected Papers, Vol. 996 of Lecture Notes in Computer Science, Springer, 1994, pp. 39–59. doi:10.1007/3-540-60579-7_3.

[16] V. Rusu, G. Grimaud, M. Hauspie, Proving partial-correctness and invariance properties of transition-system models, Sci. Comput. Program. 186 (2020). doi:10.1016/j.scico.2019.102342.

[17] Y. Bertot, E. Komendantskaya, Inductive and coinductive components of corecursive functions in Coq, in: J. Adámek, C. Kupke (Eds.), Proceedings of the Ninth Workshop on Coalgebraic Methods in Computer Science, CMCS 2008, Budapest, Hungary, April 4-6, 2008, Vol. 203 of Electronic Notes in Theoretical Computer Science, Elsevier, 2008, pp. 25–47. doi:10.1016/j.entcs.2008.05.018.

[18] Y. Bertot, E. Komendantskaya, Using structural recursion for corecursion, in: S. Berardi, F. Damiani, U. de'Liguoro (Eds.), Types for Proofs and Programs, International Conference, TYPES 2008, Torino, Italy, March 26-29, 2008, Revised Selected Papers, Vol. 5497 of Lecture Notes in Computer Science, Springer, 2008, pp. 220–236. doi:10.1007/978-3-642-02444-3_14.

[19] C. Hur, G. Neis, D. Dreyer, V. Vafeiadis, The power of parameterization in coinductive proof, in: R. Giacobazzi, R. Cousot (Eds.), The 40th Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL '13, Rome, Italy - January 23 - 25, 2013, ACM, 2013, pp. 193–206. doi:10.1145/2429069.2429093.

[20] D. Pous, Coinduction all the way up, in: M. Grohe, E. Koskinen, N. Shankar (Eds.), Proceedings of the 31st Annual ACM/IEEE Symposium on Logic in Computer Science, LICS '16, New York, NY, USA, July 5-8, 2016, ACM, 2016, pp. 307–316. doi:10.1145/2933575.2934564.

[21] N. A. Danielsson, Beating the productivity checker using embedded languages, in: E. Komendantskaya, A. Bove, M. Niqui (Eds.), Partiality and Recursion in Interactive Theorem Provers, PAR@ITP 2010, Edinburgh, UK, July 15, 2010, Vol. 5 of EPiC Series, EasyChair, 2010, pp. 34–54.

[22] A. Abel, B. Pientka, Well-founded recursion with copatterns and sized types, J. Funct. Program. 26 (2016) e2. doi:10.1017/S0956796816000022.

[23] A. Lochbihler, Coinductive, Archive of Formal Proofs http://isa-afp.org/entries/Coinductive.html, Formal proof development (Feb. 2010).

[24] D. Sangiorgi, Introduction to Bisimulation and Coinduction, Cambridge University Press, New York, NY, USA, 2011.

[25] W. P. de Roever, F. S. de Boer, U. Hannemann, J. Hooman, Y. Lakhnech, M. Poel, J. Zwiers, Concurrency Verification: Introduction to Compositional and Noncompositional Methods, Vol. 54 of Cambridge Tracts in Theoretical Computer Science, Cambridge University Press, 2001.

[26] G. D. Giacomo, M. Y. Vardi, Linear temporal logic and linear dynamic logic on finite traces, in: F. Rossi (Ed.), IJCAI 2013, Proceedings of the 23rd International Joint Conference on Artificial Intelligence, Beijing, China, August 3-9, 2013, IJCAI/AAAI, 2013, pp. 854–860.