Formal Proof of Soundness for an RL Prover

Andrei Arusoaie, David Nowak, Vlad Rusu, Dorel Lucanu

To cite this version:

Andrei Arusoaie, David Nowak, Vlad Rusu, Dorel Lucanu. Formal Proof of Soundness for an RL Prover. [Technical Report] RR-0471, INRIA Lille - Nord Europe; Alexandru Ioan Cuza, University of Iasi. 2015, pp.27. hal-01244578

HAL Id: hal-01244578

https://hal.inria.fr/hal-01244578

Submitted on 15 Dec 2015

Formal Proof of Soundness for an RL Prover

Andrei Arusoaie, David Nowak, Vlad Rusu, Dorel Lucanu

RESEARCH
REPORT
N° 471
december 2015
Project-Team Dreampal

ISSN 0249-6399 ISRN INRIA/RR--471--FR+ENG

Formal Proof of Soundness for an RL Prover

Andrei Arusoaie[*], David Nowak [†], Vlad Rusu[‡], Dorel Lucanu [§]

Project-Team Dreampal

Research Report n° 471 — december 2015 — 24 pages

Abstract: Proving programs correct is one of the major challenges that computer scientists have been struggling with during the last decades. For this purpose, Reachability Logic (RL) was proposed as a language-parametric generalisation of Hoare Logic. Recently, based on RL, an automatic verification procedure was given and proved sound. In this paper we generalise this procedure and prove its soundness formally in the Coq proof assistant. For the formalisation we had to deal with all the minutiae that were neglected in the paper proof. The trickiest one was appropriate renaming of free variables which, we discovered, was handled in the paper proof using an insufficient assumption. We also discovered a missing case in the paper proof, and we clarified some implicit and hidden hypotheses. Last but not least, the Coq formalisation provides us with a certified program-verification procedure.

Key-words: Coq, Matching Logic, Reachability Logic, program verification procedure.

[*] Inria Lille Nord Europe
[†] University of Lille1, France
[‡] Inria Lille Nord Europe
[§] University of Iasi, Romania

La preuve formelle de la correction pour un démonstrateur RL

Résumé : Prouver la correction des programmes est un des défis majeurs que les informaticiens ont été aux prises avec au cours des derniéres décennies. Dans ce but, Reachability Logic (RL) a été proposée comme une langue-parametric généralisation de Hoare Logic. Dernièrement, en s'appuyant sur RL, une procédure automatique de vérification a été propose et fait la preuve de sa correction. Dans ce papier, nous généraliser cette procédure et prouver formellement sa correction dans l'assistant de preuve Coq. Pour la formalisation nous devons nous occuper de tous les détails qui ont été négligés dans l'épreuve papier. La plus difficile était renommer correctement les variables libres qui, nous avons découvert, ont été traitée dans l'épreuve papier à l'aide d'une hypothése insuffisante. Nous avons également découvert un cas oublié dans l'épreuve papier et on a clarifié quelques hypothèses implicites ou cachées. Enfin et surtout, le formalisation dans Coq nous fournit une procédure certifiée de vérification des programmes.

Mots-clés : Coq, Matching Logic, Reachability Logic, procédure de vérification

1 Introduction

Proving programs correct is one of the major challenges that computer scientists have been struggling with during the last decades. Many techniques have been developed in order to automate this non-trivial and tedious process. Nowadays there are many software tools used for proving program correctness, but proving that the verification tools are themselves correct is often avoided. In this paper we address this issue for our own work. More precisely, we present the formalisation in Coq of the soundness of a procedure for program verification that was proposed in [15].

Floyd/Hoare Logic [11, 14], Separation Logic [18, 20], and Dynamic Logic [12] are probably the most well known logics dedicated to program verification. Recently, Matching Logic (ML) [28, 23] and Reachability Logic (RL) [26, 24, 27, 9] have been proposed as alternative approaches for dealing with this problem. ML and RL were inspired from the attempt to use rewrite-based operational semantics for program verification. Although operational semantics are considered too low level for verification, they are much easier to define and have the big advantage of being *executable* (and thus testable). For example, several large and complete operational semantics for real languages have been formalised with RL using the $\mathbb{K}$ framework [29, 22, 8]: C [10, 13], Java [5], JavaScript [19], etc.

ML is a first-order logic for specifying and reasoning about program *configurations*, e.g., code and infrastructure for executing it (heap, stack, registers, etc.). Intuitively, an ML formula φ is a configuration template accompanied by a constraint, and it denotes the set of configurations that *match* the template and satisfy the constraint. RL is a formalism which can be used for both defining operational semantics of programming language and expressing properties about program executions. An RL formula is a pair of ML formulas, denoted $\varphi \Rightarrow \varphi'$, which says that all terminating executions that start from a configuration in the set denoted by φ, eventually reach a configuration in the set denoted by φ'.

The state-of-the-art techniques for program verification involve defining a semantics that is thought of as being more appropriate for program reasoning than operational semantics. A well-known example is axiomatic semantics, which is typically given in the form of a Hoare Logic proof system for deriving Hoare triples $\{\textit{precondition}\}\texttt{code}\{\textit{postcondition}\}$. In [25] it has been shown that such a triple can be easily encoded as an RL formula $\varphi \Rightarrow \varphi'$, where φ denotes the configurations that contain the `code` to be executed and satisfy a constraint describing the *precondition* and φ' denotes the final (in the sense that there is no more code to be executed) configurations that satisfy a constraint describing the *postcondition*. There are several advantages of RL over Hoare Logic: First, RL does not depend on a specific programming language. Second, the proof system of RL [9] is proved *sound* and relatively complete for all languages, and thus, it eliminates the need for non-trivial soundness proofs for each language. Third, given an operational semantics, there is no need for an additional one which is amenable for program verification (i.e., no axiomatic semantics in the form of Hoare logic proof system is needed). Finally, RL is based on the operational semantics of a programming language, which is *executable* and hence enables the testing of programs prior to their verification. In contrast, axiomatic semantics are not executable, and thus, errors that could have been discovered by testing are discovered only later, after failed verification attempts.

One practical disadvantage of the RL proof system [9] is that the proofs tend to be very low level. Moreover, some creative user input is required when applying the rules of the proof system. Those difficulties were addressed in [15], where a procedure for program verification based on RL was proposed and the soundness of the procedure was proved. The intent behind this procedure is to overcome the lack of strategies for applying the rules of the RL proof system, and implicitly, move forward to automation. More precisely, the procedure applies in a fixed order three inference steps that are meant to capture the rules of the RL proof system. The procedure takes as inputs sets of RL formulas representing the operational semantics of a language and the program together with its specification to be proved; if it terminates, it returns either `success` or `failure`.

The soundness theorem says that if the procedure terminates and returns `success` then the program satisfies its specification. Its proof depends on several assumptions which are not precisely formalised in [15]. In order to provide additional confidence in this soundness result, we present in this paper a formalisation of the soundness proof in Coq. The paper proof from [15] raised a series of technical difficulties. One difficulty is hidden in an intermediary lemma whose proof requires intricate inductive reasoning. Then, several hypotheses and assumptions are added in an ad-hoc manner just for the proofs to hold. Also, some helper lemmas that are given inside other proofs are not precisely formulated, i.e., they need additional hypotheses. For a careful reader all these issues could raise some doubts about the validity of the paper proof. A proof in a research paper should keep a good balance between correctness and clarity. The soundness proof in [15] is monolithic, i.e. the soundness does not follow from the soundness of each derivation rule, as it is commonly done in soundness proofs. Therefore, even if the procedure is recursive, the nature of the proof is not inductive; the entire execution of the procedure is needed to prove that the goals are valid. This makes the proof complex and hence the task of keeping a good balance between correctness and clarity is difficult. In order to keep the proof concise, some details are omitted but this could have hidden side effects that are not discussed by the proof.

Clearly, a proof assistant is the solution here: it has no problem dealing with every detail because it keeps track rigorously of all cases, and thus, it ensures that no corner case is overlooked; and keys lemmas and propositions still show the main steps of the reasoning process.

Contributions We first generalise the procedure by introducing non-determinism and then encode it as an inductive relation in Coq. Then, we formally prove that our generalised procedure is sound. The soundness of the concrete procedure in [15] follows.

The formalisation led us to discovering a flaw in the paper proof: it is supposed that a claim holds for a set of RL formulas, but the claim was proved only for a subset of formulas. Fortunately, this flaw does not invalidate the final result, but it makes the proof in [15] incomplete. Moreover, in order to fix the proof we realised that we need an extra hypothesis in the soundness theorem. In Coq, we add both the required hypothesis and an additional case which handles the RL formulas in question and completes the proof.

Another issue that we discovered and fixed is related to renaming the free variables in RL formulas. In [15] this is imprecisely handled using an assumption saying that certain sets of variables are disjoint, and if they are not, then the free variables can always be renamed. However, why the variable renaming is sound is not established in the paper proof.

During the formalisation in Coq we were able to find and fix some other imprecisions in lemmas from [15]. For instance, Lemmas 7 and 8 (in [15]) only make sense in the context of the proof of Theorem 1 (in [15]). Here, these lemmas were reformulated to be self-contained, including all implicit or sometimes missing hypotheses. Last but not least, the Coq formalisation provides us with a certified program-verification procedure. Its use in practice depends on the availability in Coq of RL-based semantics for languages; such an effort is already underway in the $\mathbb{K}$ team [17].

Related work The closest related work is the formalisation in Coq of RL reported in [9]. Our initial attempt to prove the soundness of the procedure from [15] consisted in reusing the mechanised proof reported in [9]. However, the soundness of the procedure is not a direct consequence of the soundness of the RL proof system, because there is no direct correspondence between the inference steps and the rules of the proof system. This is the reason why we choose to give a direct proof of the soundness of the procedure. An additional benefit of a direct proof is that, compared to the RL proof system, our procedure is closer to an implementation. Thus, by formalising directly the soundness of the procedure in Coq (vs. formalising the soundness of a proof system) we bridge the gap between theory and implementation.

Another approach which is strongly connected to RL is [17]. It proposes a language-independent proof method for verifying reachability properties on programs. The approach is also implemented in Coq. Although it provides tactics to increase the level of automation, the tool is not fully automatic, unlike the procedure in [15]. From the same authors, there are also ongoing works for translating $\mathbb{K}$ language definitions to Coq.

There are many soundness results about proof systems for program verification (e.g., [16, 1, 32, 31, 30] to cite only a few). Most of them are based either on Hoare Logic or Separation Logic, and thus, they are language dependent. For instance, in [32] the authors present three Hoare logics that correspond to different notions of correctness for the simple While language; for all three they prove the soundness and the relative completeness. However, compared to [15], they do not provide a language-parametric program-verification automatic procedure.

There is moreover a wide range of certified sound procedures for deciding various logical fragments, some of them formalised in Coq (e.g. [6, 4]), others in Isabelle/HOL (e.g., [7]).

Outline. Section 2 provides a short overview of ML and RL, and presents the verification procedure proposed in [15] together with the corresponding soundness result. Section 3 describes our formalisation in Coq of the soundness of the procedure. It mainly contains the definitions and lemmas needed by the soundness result, and it also highlights the differences between the Coq proof and the one in [15]. We conclude in Section 4.

2 Background

2.1 Matching Logic & Reachability Logic

This section provides an overview of ML and RL. For both logics, we explain their syntax and semantics by means of simple examples. In addition, several notions and definitions specific to this paper are also presented. We assume the reader is familiar with First-Order Logic (hereafter, FOL).

Matching Logic ML is a first-order logic variant for specifying and reasoning about structure. We here consider the version of ML from [21]. Formally, ML formulas are defined as follows. If Σ is a many-sorted algebraic signature, Π is a set of predicate symbols, *Var* a set of (sorted) variables, and *Cfg* a sort for program configurations, then the syntax of ML formulas is:

$$\varphi ::= \pi \mid \top \mid p(t_1, \ldots, t_n) \mid \neg\varphi \mid \varphi \wedge \varphi \mid (\exists V)\varphi,$$

where π is a Σ-term of sort *Cfg* with variables (also called *basic pattern*), p is a predicate symbol in Π, t_i are Σ-terms with variables of appropriate sorts, and V a subset of *Var*. As usual, the other known FOL connectives ($\vee$, $\rightarrow$, ...) and quantifier $\forall$ can be expressed using the ones above.

Let us consider a signature Σ which contains a sort *AExp* that includes the integer numbers (as symbols) together with the usual binary operations (e.g., $\_+\_$, $\_-\_$, ...), and a sort *Cfg*, which has a single constructor $\langle \_, \_ \rangle : \mathit{AExp} \times \mathit{AExp} \rightarrow \mathit{Cfg}$ (we use $\_$ to denote the positions of the arguments). Also, let Π be the set which contains the predicate symbols $\_>\_$ and $\_\geq\_$ with arguments of sort *AExp*, and *Var* a set of variables of sort *AExp* (e.g. x, y, z, ...). Then, an example of ML formula is:

$$\varphi \triangleq \langle x-y, z\rangle \wedge x > y \wedge z \geq 0$$

Indeed, x, y and z are terms of sort *AExp*, $\langle x-y, z\rangle$ is a term of sort *Cfg* with variables (i.e., a basic pattern), $x > y \wedge z \geq 0$ is an ML formula, and $\langle x-y, z\rangle \wedge x > y \wedge z \geq 0$ is also an ML formula.

The set of free variables that occur in an ML formula φ is denoted by $\mathrm{FV}(\varphi)$. For instance, $\mathrm{FV}((\exists x, z)\langle x, y\rangle \wedge z > 0) = \{y\}$.

Next, we assume a (Σ, Π)-model M. The satisfaction relation $\models_{\mathrm{ML}}$ of ML is akin to the satisfaction relation of FOL (denoted $\models_{\mathrm{FOL}}$). For instance, $(\gamma, \rho) \models_{\mathrm{ML}} (\exists V)\varphi$ iff there is $\rho' : \mathit{Var} \to M$ with $\rho'(x) = \rho(x)$ for all $x \notin V$ such that $(\gamma, \rho') \models_{\mathrm{ML}} \varphi$. This is similar to FOL, except that $\models_{\mathrm{ML}}$ is defined over pairs (γ, ρ), where γ is an element in M of sort *Cfg* called a *configuration*, $\rho : \mathit{Var} \to M$ is a valuation, and ML formulas φ. In addition to the other FOL constructs, $\models_{\mathrm{ML}}$ includes the ML particular case:

$$(\gamma, \rho) \models_{\mathrm{ML}} \pi \text{ iff } \rho(\pi) = \gamma.$$

Intuitively, a basic pattern π is satisfied by a pair (γ, ρ) if γ *matches* π with ρ; here, the valuation ρ can be thought of as the witness of the matching. For example, $\langle 7-3, 3\rangle$ matches the basic pattern $\langle x-y, z\rangle$ with a valuation ρ that maps x, y, and z to 7, 3, and 3, respectively: $\rho(\langle x-y, z\rangle) = \langle 7-3, 3\rangle$.

An ML formula φ is *valid* in M, written $M \models \varphi$, iff for all γ and $\rho : \mathit{Var} \to M$ we have $(\gamma, \rho) \models_{\mathrm{ML}} \varphi$.

Let M be a model which interprets the sort *AExp* as the set of integers, and the binary symbols $+$, $-$, ... as the usual operations over integers. Similarly, M interprets the predicate symbols $>$ and $\geq$ as the usual corresponding relations over integers. Also, consider the valuation $\rho : \mathit{Var} \to M$ with $\rho(x) = 7$, $\rho(y) = 3$, and $\rho(z) = 3$. Then, $(\langle 7-3, 3\rangle, \rho) \models_{\mathrm{ML}} \varphi$ because $\rho(\langle x-y, z\rangle) = \langle 7-3, 3\rangle$ and $(\gamma, \rho) \models_{\mathrm{ML}} x > y \wedge z \geq 0$ (since $7 > 3$ and $3 \geq 0$ in M).

It has been shown in [24] that ML can be encoded in FOL. If φ is an ML-formula then its encoding $\varphi^{=?}$ in FOL is the formula $(\exists z)\varphi'$, where φ' is obtained from φ by replacing each basic pattern occurrence π with $z = \pi$, and z is a variable which does not appear in the free variables of φ. Here are a few examples of formulas and their encodings:

φ	$\varphi^{=?}$
$(\pi_1 \wedge \phi_1) \vee (\pi_2 \wedge \phi_2)$	$(\exists z)((z = \pi_1 \wedge \phi_1) \vee (z = \pi_2 \wedge \phi_2))$
$\neg\pi$	$(\exists z)\neg(z = \pi)$
$\pi \vee \neg\pi$	$(\exists z)(z = \pi \vee \neg(z = \pi))$

The encoding of an ML formula φ has the following property [15]: *for all* ρ *and* φ, $\rho \models_{\mathrm{FOL}} \varphi^{=?}$ *iff there is* γ *such that* $(\gamma, \rho) \models_{\mathrm{ML}} \varphi$.

Reachability Logic An RL formula (also called *rule*) is a pair of ML formulas, written $\varphi \Rightarrow \varphi'$, which expresses reachability relationships between the two sets of configurations denoted by φ and φ'. For example, the RL formula

$$\alpha \triangleq \langle x, y\rangle \wedge x \geq 0 \Rightarrow \langle x-y, y\rangle$$

expresses the fact that configurations which satisfy the $\langle x-y, y\rangle$ (e.g. $\langle 7-3, 3\rangle$ with ρ aforesaid), can be reached from configurations which satisfy $\langle x, y\rangle \wedge x \geq 0$ (e.g., $\langle 7, 3\rangle$ with the same ρ).

By $\mathrm{FV}(\varphi \Rightarrow \varphi')$ we denote the union $\mathrm{FV}(\varphi) \cup \mathrm{FV}(\varphi')$. Also, if $\mathcal{S}$ is a set of RL formulas then $\mathrm{FV}(\mathcal{S}) \triangleq \bigcup_{\varphi\Rightarrow\varphi'\in\mathcal{S}} \mathrm{FV}(\varphi \Rightarrow \varphi')$.

A set $\mathcal{S}$ of RL formulas defines a transition system $(\mathit{Cfg}, \Rightarrow_{\mathcal{S}})$ over configurations: we say that $\gamma_0 \Rightarrow_{\mathcal{S}} \gamma_1$ if there is $\varphi_0 \Rightarrow \varphi_1 \in \mathcal{S}$ and there is a valuation $\rho' : \mathit{Var} \to M$ such that $(\gamma_0, \rho') \models_{\mathrm{ML}} \varphi_0$ and $(\gamma_1, \rho') \models_{\mathrm{ML}} \varphi_1$. We often use only $\Rightarrow_{\mathcal{S}}$ to denote the transition system $(\mathit{Cfg}, \Rightarrow_{\mathcal{S}})$. Back to our example, the set $\mathcal{S} = \{\alpha\}$ generates the transition $\langle 7, 3\rangle \Rightarrow_{\{\alpha\}} \langle 7 - 3, 3\rangle$.

A pattern is $\mathcal{S}$-*derivable* if there is a transition from one of its instances with $\mathcal{S}$. For example, $\varphi_0 \triangleq \langle x, y\rangle \wedge x \geq 7$ is $\{\alpha\}$-derivable because there is a transition from $\langle 7, 3\rangle$ (which satisfies φ_0 with ρ) to $\langle 7 - 3, 3\rangle$.

A set $\mathcal{S}$ of RL formulas is *total* if for all φ, γ, and ρ, if φ is $\mathcal{S}$-derivable and $(\gamma, \rho) \models_{\mathrm{ML}} \varphi$ then there is γ' such that $\gamma \Rightarrow_{\mathcal{S}} \gamma'$. Note that $\{\alpha\}$ is not total because of the condition $x \geq 0$ in its left hand side. To make it total, one should add to $\{\alpha\}$ another rule which handles the other possible case, namely $x < 0$.

The satisfaction relation $\models_{\mathrm{RL}}$ of RL is defined using *paths*, which are (possibly infinite) sequences of transitions $\gamma_0 \Rightarrow_{\mathcal{S}} \gamma_1 \Rightarrow_{\mathcal{S}} \gamma_2 \Rightarrow_{\mathcal{S}} \cdots$. For example, $\langle 7, 3\rangle \Rightarrow_{\{\alpha\}} \langle 7 - 3, 3\rangle \Rightarrow_{\{\alpha\}} \langle 7 - 3 - 3, 3\rangle \Rightarrow_{\{\alpha\}} \langle 7 - 3 - 3 - 3, 3\rangle$ is a finite path. Note that α cannot be applied anymore to $\langle 7 - 3 - 3 - 3, 3\rangle$ because of the constraint $x \geq 0$. If none of the rules in $\mathcal{S}$ can be applied then we say that the configuration (e.g. $\langle 7 - 3 - 3 - 3, 3\rangle$) is *terminating.*

Paths can be infinite too, e.g., $\langle 7, 0\rangle \Rightarrow_{\{\alpha\}} \langle 7 - 0, 0\rangle \Rightarrow_{\{\alpha\}} \cdots$. A path τ is *complete* if it is either finite and the last configuration in τ is terminating, or it is infinite. Both paths shown above are complete. Typically, we use τ to denote paths, i.e., $\tau \triangleq \gamma_0 \Rightarrow_{\mathcal{S}} \gamma_1 \Rightarrow_{\mathcal{S}} \gamma_2 \Rightarrow_{\mathcal{S}} \cdots$, and $\tau|_i$ to denote the sub-path of τ starting at position i, i.e., $\tau|_i \triangleq \gamma_i \Rightarrow_{\mathcal{S}} \gamma_{i+1} \Rightarrow_{\mathcal{S}} \cdots$. We also say that τ *starts from* from an ML formula φ if there is a valuation $\rho : \mathit{Var} \to M$ such that $(\gamma_0, \rho) \models_{\mathrm{ML}} \varphi$. For example, $\langle 7, 0\rangle \Rightarrow_{\{\alpha\}} \langle 7 - 0, 0\rangle \Rightarrow_{\{\alpha\}} \cdots$ and $\langle 7, 3\rangle \Rightarrow_{\{\alpha\}} \langle 7 - 3, 3\rangle \Rightarrow_{\{\alpha\}} \langle 7 - 3 - 3, 3\rangle \Rightarrow_{\{\alpha\}} \langle 7 - 3 - 3 - 3, 3\rangle$ start from $\langle x, y\rangle \wedge x \geq 0$.

In this paper we are only interested in the *all-path* interpretation [9] of RL formulas: a pair (τ, ρ) *satisfies* an RL formula $\varphi \Rightarrow \varphi'$, written $(\tau, \rho) \models \varphi \Rightarrow \varphi'$, iff (τ, ρ) starts from φ (recall that $\tau \triangleq \gamma_0 \Rightarrow_{\mathcal{S}} \gamma_1 \Rightarrow_{\mathcal{S}} \gamma_2 \Rightarrow_{\mathcal{S}} \cdots$), and either there exists $i \geq 0$ such that $(\gamma_i, \rho) \models_{\mathrm{ML}} \varphi'$, or τ is infinite; $\Rightarrow_{\mathcal{S}}$ *satisfies* $\varphi \Rightarrow \varphi'$, written $\Rightarrow_{\mathcal{S}} \models_{\mathrm{RL}} \varphi \Rightarrow \varphi'$, iff $(\tau, \rho) \models_{\mathrm{RL}} \varphi \Rightarrow \varphi'$ for all (τ, ρ) starting from φ with τ complete. We often write $\mathcal{S} \models_{\mathrm{RL}} \varphi \Rightarrow \varphi'$ instead of $\Rightarrow_{\mathcal{S}} \models_{\mathrm{RL}} \varphi \Rightarrow \varphi'$.

Let $\varphi \Rightarrow \varphi'$ be the RL formula:

$$\langle x, y\rangle \wedge x \geq y \wedge y > 0 \Rightarrow (\exists x')\langle x', y\rangle \wedge \mathit{mod}(x', x, y),$$

where mod is a predicate symbol with interpretation: $M_{\mathit{mod}}(r, x, y)$ holds iff r is the remainder after division of x by y. The path $\tau \triangleq \langle 7, 3\rangle \Rightarrow_{\{\alpha\}} \langle 7-3, 3\rangle \Rightarrow_{\{\alpha\}} \langle 7-3-3, 3\rangle \Rightarrow_{\{\alpha\}} \langle 7-3-3-3, 3\rangle$ is complete and starts from $\langle x, y\rangle \wedge x \geq y \wedge y > 0$ with a valuation $\rho : \mathit{Var} \to M$, which maps x and y to 7 and 3, respectively. However, $(\tau, \rho) \not\models \varphi \Rightarrow \varphi'$ because $(\langle 7 - 3 - 3 - 3, 3\rangle, \rho) \not\models_{\mathrm{ML}} \varphi'$ (since, in M, -2 is not the remainder after division of 7 by 3).

However, if $x \geq y$, the remainder after division of x by y can be obtained by repeated subtraction of y from the initial x if $y > 0$. Thus, let us consider a version of α:

$$\alpha' \triangleq \langle x, y\rangle \wedge x \geq y \wedge y > 0 \Rightarrow \langle x - y, y\rangle$$

The finite path $\tau' \triangleq \langle 7, 3\rangle \Rightarrow_{\{\alpha'\}} \langle 7 - 3, 3\rangle \Rightarrow_{\{\alpha'\}} \langle 7 - 3 - 3, 3\rangle$ is complete and starts from $\langle x, y\rangle \wedge x \geq y \wedge y > 0$ with the same valuation ρ as above. Also, there is a configuration $\langle 7 - 3 - 3, 3\rangle$ in τ' such that $(\langle 7 - 3 - 3, 3\rangle, \rho) \models_{\mathrm{ML}} \varphi'$. More precisely, $(\langle 7 - 3 - 3, 3\rangle, \rho) \models_{\mathrm{ML}} (\exists x')\langle x', y\rangle \wedge \mathit{mod}(x', x, y)$ iff there is a valuation $\rho' : \mathit{Var} \to M$ such that $\rho'(y) = \rho(y)$ and $(\langle 7 - 3 - 3, 3\rangle, \rho') \models_{\mathrm{ML}} \langle x', y\rangle \wedge \mathit{mod}(x', x, y)$; if we take ρ' such that $\rho'(v) = \rho(v)$ for all $v \in \mathit{Var} \setminus \{x'\}$ and $\rho'(x') = 1$, then $\rho'(\langle x', y\rangle) = \langle 1, 3\rangle$ and $M_{\mathit{mod}}(1, 7, 3)$ holds, which implies $(\langle 7 - 3 - 3, 3\rangle, \rho') \models_{\mathrm{ML}} \langle x', y\rangle \wedge \mathit{mod}(x', x, y)$. Therefore, we obtain $(\tau', \rho) \models_{\mathrm{RL}} \varphi \Rightarrow \varphi'$.

Language definitions In the example above, we have shown that the path τ', whose transitions are given by $\{\alpha'\}$, satisfies the RL formula $\varphi \Rightarrow \varphi'$ with a valuation ρ. The system $\mathcal{S} = \{\alpha'\}$ can be seen as the operational semantics of a language whose programs are pairs of (arithmetic) expressions. Formally, we consider language definitions $((\Sigma, \Pi, \mathit{Cfg}), M, \mathcal{S})$, where, for our simple language, Σ is the signature containing the sorts *Cfg* and *AExp* (together with the corresponding operation symbols), Π is the set of predicates including $\_{>}\_$ and $\_{\geq}\_$, M is a (Σ, Π) model, and $\mathcal{S} = \{\alpha'\}$.

This notion of language definition is general, in the sense that it captures real language definitions, such as those defined in the $\mathbb{K}$ framework, e.g., C [10, 13], Java [5], JavaScript [19], etc.

Derivatives of an ML (and RL) formula A notion introduced in [15] is that of *derivative* of an ML (and RL) formula. If φ is an ML formula and $\mathcal{S}$ is a set of RL formulas, then the derivative of φ with $\mathcal{S}$ is defined as follows:

$$\Delta_{\mathcal{S}}(\varphi) \triangleq \{(\exists \mathrm{FV}(\varphi_l, \varphi_r))(\varphi_l \wedge \varphi)^{=?} \wedge \varphi_r \mid \varphi_l \Rightarrow \varphi_r \in \mathcal{S}\}.$$

If $\varphi \Rightarrow \varphi'$ is an RL formula then

$$\Delta_{\mathcal{S}}(\varphi \Rightarrow \varphi') \triangleq \{\varphi_1 \Rightarrow \varphi' \mid \varphi_1 \in \Delta_{\mathcal{S}}(\varphi)\}.$$

Intuitively, the derivative of an ML formula φ encodes the concrete successors by $\Rightarrow_{\mathcal{S}}$ of configurations matching φ. Derivatives can be computed for sets of RL formulas G too:

$$\Delta_{\mathcal{S}}(G) \triangleq \bigcup_{\varphi \Rightarrow \varphi' \in G} \Delta_{\mathcal{S}}(\varphi \Rightarrow \varphi').$$

Let again $\alpha' \triangleq \langle x, y\rangle \wedge x \geq y \wedge y > 0 \Rightarrow \langle x - y, y\rangle$ and $\varphi \triangleq \langle u, v\rangle \wedge u \geq v \wedge v > 0$. Then $\Delta_{\{\alpha'\}}(\varphi)$ has only one element:

$$(\exists x, y)((\langle x, y\rangle \wedge x \geq y \wedge y > 0) \wedge (\langle u, v\rangle \wedge u \geq v \wedge v > 0))^{=?} \\ \wedge \langle x - y, y\rangle,$$

which, by the definition of the encoding, becomes:

$$(\exists x, y)(\langle x, y\rangle = \langle u, v\rangle) \wedge x \geq y \wedge y > 0 \wedge u \geq v \wedge v > 0 \\ \wedge \langle x - y, y\rangle.$$

This can be further simplified to $\varphi' \triangleq \langle u - v, v\rangle \wedge u \geq v \wedge v > 0$ using the fact that $\langle x, y\rangle = \langle u, v\rangle$ implies $x = u$ and $y = v$. The ML formula φ' specifies the configurations obtained from those satisfying φ after applying α'; for example, the configuration $\langle 7 - 3, 3\rangle$ (which satisfies φ' with a valuation ρ s.t. $\rho(u) = 7$, $\rho(v) = 3$) is a successor of $\langle 7, 3\rangle$ (that satisfies φ with the same ρ), i.e., $\langle 7, 3\rangle \Rightarrow_{\{\alpha'\}} \langle 7 - 3, 3\rangle$.

Remark 1 *Note that in the example above the free variables in φ are disjoint from those in α', i.e, $\{u, v\} \cap \{x, y\} = \emptyset$. If these sets are not disjoint, then there is no guarantee that the derivative encodes the set of all concrete successors of configurations that match φ. For instance, let $\varphi'' \triangleq \langle u, x\rangle \wedge u \geq x \wedge x > 0$ (i.e., φ'' is φ where variable v was replaced by x). When computing the derivative of φ'' with $\{\alpha'\}$, the variable $x \in \mathrm{FV}(\varphi'')$ is bound:*

$$(\exists x, y)(\langle x, y\rangle = \langle u, x\rangle) \wedge x \geq y \wedge y > 0 \wedge u \geq x \wedge x > 0 \\ \wedge \langle x - y, y\rangle.$$

Again, this formula can be simplified to $\varphi'' \triangleq \langle u - u, u\rangle \wedge u > 0$, using the fact that $\langle x, y\rangle = \langle u, x\rangle$ implies $x = u$ and $y = x$. Here, there is no valuation ρ' such that $(\langle 7 - 3, 3\rangle, \rho') \models \varphi''$, and thus, not all successors by $\Rightarrow_{\{\alpha'\}}$ of $\langle 7, 3\rangle$ are matched by the derivative.

Therefore, when computing $\Delta_{\mathcal{S}}(\varphi)$, it is important that the free variables of φ and those of $\mathcal{S}$ are in disjoint sets.

The issue described in REMARK 1 is also an issue in the verification procedure proposed in [15]: the procedure computes derivatives $\Delta_{\mathcal{S}}(\varphi)$ without checking (as it should) that $\mathrm{FV}(\mathcal{S})$ and $\mathrm{FV}(\varphi)$ are disjoint. This is discussed in detail in the next section.

2.2 A procedure for verifying Reachability Logic properties

A procedure for verifying RL formulas (Figure 1), called `prove`, was introduced in [15]. The intent behind `prove` is to overcome the lack of strategies for applying the rules of the RL proof system from [9], and implicitly, to move forward towards automatisation.

In a nutshell, the procedure uses symbolic execution (i.e., run programs with symbolic values instead of concrete ones, and collect the constraints over symbolic values) combined with circular reasoning. Intuitively, given an RL formula $\varphi \Rightarrow \varphi'$, one can symbolically execute φ and check whether every leaf node of the obtained symbolic execution tree implies φ'. The procedure uses derivatives to compute symbolic execution paths. An obvious problem of this approach is that the symbolic execution tree can be infinite due to loops or recursion. To overcome this problem RL allows the use of helper formulas called *circularities*, which generalise the notion of invariant.

Instead of proving a single formula, `prove` attempts to prove a set of formulas (goals). The procedure checks first whether a helper formula can be used (e.g., an RL formula specifying a loop) to discharge the current goal, rather than performing symbolic execution blindly. A goal can be used in the proof of a different goal or in its own proof, provided that at least one symbolic step has been performed from it.

Note that a proof by simple induction on the inferences steps of the procedure does not work because of its circular nature, that is, it uses goals to prove other goals (line 4 in Figure 1). This is sound only if a symbolic step from the current goal has been performed. If this condition is not met then every goal can be proved using itself! For this reason, the soundness theorem assumes a successful execution starting with $\Delta_{\mathcal{S}}(G_0)$ instead of G_0.

The procedure takes as input three sets of RL formulas: the operational semantics $\mathcal{S}$, a set of goals G_0, and a recursive argument G whose initial value is $\Delta_{\mathcal{S}}(G_0)$; and it returns either `success` or `failure`. At each recursive call, a goal from G is either eliminated and/or replaced by other goals. If $\varphi \Rightarrow \varphi' \in G$ is the current goal then it is processed as follows: if $M \models \varphi \rightarrow \varphi'$ then the goal is eliminated from G; else, the procedure checks whether there is a circularity available in G_0 which is used to generate a new goal that replaces the current one; if no circularity is found, but φ is $\mathcal{S}$-derivable, then $\varphi \Rightarrow \varphi'$ is replaced by a set of goals consisting in its symbolic successors; otherwise, `prove` returns `failure`. The procedure succeeds when all the goals in $\Delta_{\mathcal{S}}(G_0)$ together with those generated during the execution are eliminated. If the procedure does not terminate or it returns `failure` then it means that G_0 does not contain enough information to prove the goals. This is similar to proving invariants for loops in imperative programs, which requires users to provide strong-enough invariants.

Below, we show the soundness result and a sketch of the proof that was given in [15]. We illustrate by this the fact it is not a simple proof by induction on the recursive calls: even if they terminate, the calls do not induce a well-founded order over formulas ensuring a formula holds whenever the "smaller" formulas hold as well. Indeed, a formula may coinductively use itself in its own proof.

Theorem 1 (Theorem 1 in [15]) *Let* `prove` *be the procedure given in Figure 1. Assume that $\mathcal{S}$ is total. Let G_0 be such that for each $\varphi_c \Rightarrow \varphi'_c \in G_0$, φ_c is $\mathcal{S}$-derivable and satisfies* $\mathrm{FV}(\varphi'_c) \subseteq \mathrm{FV}(\varphi_c)$. *If* `prove(`$\mathcal{S}, G_0, \Delta_{\mathcal{S}}(G_0)$`)` *returns* `success` *then* $\mathcal{S} \models_{\mathrm{RL}} G_0$.

```
procedure prove($\mathcal{S}, G_0, G$)
1: if $G = \emptyset$ then return success
2:    else choose $\varphi \Rightarrow \varphi' \in G$
3:       if $M \models_{\text{ML}} \varphi \rightarrow \varphi'$ then return prove($\mathcal{S}, G_0, G \setminus \{\varphi \Rightarrow \varphi'\}$)
4:       else if there is $\varphi_c \Rightarrow \varphi'_c \in G_0$
                 s. t. $M \models_{\text{ML}} \varphi \rightarrow \overline{\varphi}_c$ then
5:          return prove($\mathcal{S}, G_0, G \setminus \{\varphi \Rightarrow \varphi'\} \cup \Delta_{\varphi_c \Rightarrow \varphi'_c}(\varphi \Rightarrow \varphi')$)
6:       else if $\varphi$ is $\mathcal{S}$-derivable then
7:          return prove($\mathcal{S}, G_0, G \setminus \{\varphi \Rightarrow \varphi'\} \cup \Delta_{\mathcal{S}}(\varphi \Rightarrow \varphi')$)
8:       else return failure.
```

Figure 1: RL verification procedure. $\overline{\varphi}_c$ denotes $(\exists \text{FV}(\varphi_c))\varphi_c$.

Proof sketch At each step of the procedure, the set G of goals evolves from $\Delta_{\mathcal{S}}(G_0)$ to $\emptyset$ (if `prove` terminates successfully). Let $\mathcal{F} \triangleq G_0 \bigcup_i G_i$, where i represents the i^{th} call of `prove`. The first call is made for $G_1 \triangleq \Delta_{\mathcal{S}}(G_0)$.

In order to prove that $\mathcal{S} \models G_0$ one has to prove that $\mathcal{S} \models_{\text{RL}} \varphi \Rightarrow \varphi'$ for all goals $\varphi \Rightarrow \varphi' \in G_0$, that is, $(\tau, \rho) \models \varphi \Rightarrow \varphi'$ for all (τ, ρ) starting from φ with τ complete.

We consider an arbitrary $\tau \triangleq \gamma_0 \Rightarrow_{\mathcal{S}} \gamma_1 \Rightarrow_{\mathcal{S}} \cdots$ such that τ starts from φ with a valuation $\rho : \mathit{Var} \rightarrow M$. If τ is infinite then, by definition, $(\tau, \rho) \models \varphi \Rightarrow \varphi'$. There remains to show that $(\tau, \rho) \models \varphi \Rightarrow \varphi'$ when the path is finite, i.e., $\tau \triangleq \gamma_0 \Rightarrow_{\mathcal{S}} \gamma_1 \Rightarrow_{\mathcal{S}} \cdots \Rightarrow_{\mathcal{S}} \gamma_n$. This case is proved using an intermediate lemma, which states the property for *all* goals in $\mathcal{F}$, including the ones in G_0:

Lemma 1 (Lemma 8 in [15]) *For all τ, for all ρ, for all $\varphi \Rightarrow \varphi' \in \mathcal{F}$, if τ is finite and complete, and (τ, ρ) starts from φ then $(\tau, \rho) \models \varphi \Rightarrow \varphi'$.*

The lemma works on the following assumption (from [15]):

(a_1) *"In what follows we consider only* ML *formulas φ with the following property: If φ does not occur as a member of a rule in $\mathcal{S}$ and $\varphi_l \Rightarrow \varphi_r \in \mathcal{S}$ then* $\text{FV}(\varphi) \cap \text{FV}(\varphi_l, \varphi_r) = \emptyset$. *This is not a real restriction since the free variables in rules can always be renamed."*

It also uses two additional results (from [15]):

(1) *For all $\varphi \Rightarrow \varphi' \in \mathcal{F}$ either $M \models \varphi \rightarrow \varphi'$ or φ is $\mathcal{S}$-derivable.*

(2) *For all γ_0, γ_1, φ, and ρ such that $\gamma \Rightarrow_{\{\alpha\}} \gamma'$ and $(\gamma, \rho) \models_{\text{ML}} \varphi$ there is $\varphi' \triangleq \Delta_{\{\alpha\}}(\varphi)$ such that $(\gamma', \rho) \models_{\text{ML}} \varphi'$.*

The proof of the lemma is by induction on the number of transitions (n) in $\tau \triangleq \gamma_0 \Rightarrow_{\mathcal{S}} \gamma_1 \Rightarrow_{\mathcal{S}} \cdots \Rightarrow_{\mathcal{S}} \gamma_n$. Since τ is finite, proving $(\tau, \rho) \models \varphi \Rightarrow \varphi'$ reduces to finding γ_i such that $(\gamma_i, \rho) \models_{\text{ML}} \varphi'$.

In the base case $n = 0$ and $\tau \triangleq \gamma_0$. Since $\varphi \Rightarrow \varphi' \in \mathcal{F}$ the auxiliary result (1) ensures that $M \models \varphi \rightarrow \varphi'$ or φ is $\mathcal{S}$-derivable. From (τ, ρ) starts from φ we obtain that $(\gamma_0, \rho) \models \varphi$. If $M \models \varphi \rightarrow \varphi'$, then we have γ_0 such that $(\gamma_0, \rho) \models \varphi'$. If φ is $\mathcal{S}$-derivable, then we obtain a contradiction in our hypotheses: on the one hand, γ_0 is terminating because τ is complete, and on the other hand γ_0 is not terminating because there is γ_1 such that $\gamma_0 \Rightarrow_{\mathcal{S}} \gamma_1$ which is implied by the totality of $\mathcal{S}$ and $(\gamma_0, \rho) \models \varphi$.

In the inductive case, we have $n > 0$ and $\tau \triangleq \gamma_0 \Rightarrow_{\mathcal{S}} \gamma_1 \Rightarrow_{\mathcal{S}} \cdots \Rightarrow_{\mathcal{S}} \gamma_n$. Note that $n > 0$ implies the fact that τ contains at least the first transition $\gamma_0 \Rightarrow_{\mathcal{S}} \gamma_1$, i.e., there is $\alpha \in \mathcal{S}$ such that $\gamma_0 \Rightarrow_{\{\alpha\}} \gamma_1$.

Since $\varphi \Rightarrow \varphi' \in \mathcal{F}(\triangleq \bigcup_{i\geq 0} G_i)$ then $\varphi \Rightarrow \varphi'$ was eliminated at some point, i.e, there is i such that $\varphi \Rightarrow \varphi' \in G_i \setminus G_{i+1}$; We distinguish the three situations (corresponding to the inference steps of the procedure):

- $M \models \varphi \rightarrow \varphi'$. Trivial: $(\gamma_0, \rho) \models \varphi$ implies $(\gamma_0, \rho) \models \varphi'$.
- there is $\varphi_c \Rightarrow \varphi'_c \in G_0$ s. t. $M \models_{\text{ML}} \varphi \rightarrow (\exists \text{FV}(\varphi_c))\varphi_c$.

  From $(\gamma_0, \rho) \models \varphi$ and $M \models_{\text{ML}} \varphi \rightarrow (\exists \text{FV}(\varphi_c))\varphi_c$ we get $(\gamma_0, \rho) \models (\exists \text{FV}(\varphi_c))\varphi_c$, i.e, there is ρ' such that $\rho'(x) = \rho(x)$ for all $x \notin \text{FV}(\varphi_c)$ and $(\gamma_0, \rho') \models_{\text{ML}} \varphi_c$. Then, by applying (2) to $(\gamma_0, \rho') \models_{\text{ML}} \varphi_c$ and $\gamma_0 \Rightarrow_{\mathcal{S}} \gamma_1$ there is $\varphi_1 \triangleq \Delta_{\{\alpha\}}(\varphi_c)$ and $(\gamma_1, \rho') \models_{\text{ML}} \varphi_1$.

  Next, the inductive hypothesis is applied twice.

  First, the inductive hypothesis is applied for $\tau|_1$, which starts from φ_1, and $\varphi_1 \Rightarrow \varphi'_c \in \Delta_{\mathcal{S}}(\varphi_c \Rightarrow \varphi_{c'}) \subseteq \mathcal{F}$, and thus, there is γ_j which satisfies φ'_c with ρ'; Then, using the assumption (a_1) from above, one can prove $(\gamma_j, \rho) \models_{\text{ML}} \Delta_{\varphi_c \Rightarrow \varphi'_c}(\varphi)$.

  Second, we apply the inductive hypothesis for $\tau|_j$, which starts from $\Delta_{\varphi_c \Rightarrow \varphi'_c}(\varphi)$, and $\Delta_{\varphi_c \Rightarrow \varphi'_c}(\varphi \Rightarrow \varphi') \in \mathcal{F}$, and thus, there is γ_i such that $(\gamma_i, \rho) \models_{\text{ML}} \varphi'$.
- φ is $\mathcal{S}$-derivable. By applying (2) for $(\gamma_0, \rho) \models_{\text{ML}} \varphi_c$ and $\gamma_0 \Rightarrow_{\mathcal{S}} \gamma_1$, there is $\varphi_1 \triangleq \Delta_{\{\alpha\}}(\varphi)$ and $(\gamma_1, \rho) \models_{\text{ML}} \varphi_1$. In this case we apply the inductive hypothesis for $\tau|_1$, which starts from $\varphi_1 \triangleq \Delta_{\{\alpha\}}(\varphi)$, and for $\varphi_1 \Rightarrow \varphi' \in \Delta_{\mathcal{S}}(G_0) \subseteq \mathcal{F}$, to obtain γ_i such that $(\gamma_i, \rho) \models_{\text{ML}} \varphi'$.

Remark 2 *Computing $\Delta_{\varphi_c \Rightarrow \varphi'_c}(\varphi)$ (in the second case of the inductive step) requires that sets* $\text{FV}(\varphi)$ *and* $\text{FV}(\varphi_c, \varphi'_c)$ *are disjoint (cf.* Remark *1). However, in the proof, there is no guarantee that this condition is met. Note that assumption* (a_1) *is insufficient in this case because it takes into account only rules in $\mathcal{S}$ (i.e., if $\varphi_l \Rightarrow \varphi_r \in \mathcal{S}$ then* $\text{FV}(\varphi) \cap \text{FV}(\varphi_l, \varphi_r) = \emptyset$*), but here $\varphi_c \Rightarrow \varphi'_c$ is from G_0. In fact,* (a_1) *is relevant only when applying the additional result* (2)*, since the rule α is from $\mathcal{S}$. Although the assumption* (a_1) *states that the free variables in rules can always be renamed, in the proof from [15] this is not handled explicitly.*

3 A formal proof of soundness in Coq

In this section we generalise the procedure by introducing non-determinism and then encode it as an inductive relation in Coq. In spite of this generalisation, the formal proof of soundness follows the same pattern as the paper proof in [15].

Using Coq, we were able to find a flaw in the paper proof: the claim that all the goals in $\mathcal{F}$ are eliminated by one of the inference steps of the procedure (made in the proof of Lemma 1) does not hold. More precisely, the claim holds only for goals from $\mathcal{F} \setminus G_0$ because the procedure is called with $G_1 \triangleq \Delta_{\mathcal{S}}(G_0)$, that is, it operates only with goals from $G_1, G_2, \ldots$. This flaw does not invalidate the final result, but it makes the proof incomplete: an additional case is needed to prove the conclusion of Lemma 1 for the goals in G_0. Note that, in the proof of Theorem 1, the goals (of interest) for which we apply Lemma 1 are exactly those from $G_0 \subseteq \mathcal{F}$. During the formalisation, we also realised that we need an extra hypothesis in the soundness theorem to fix the proof.

Unlike in [15], in the Coq proof we explicitly handle the renaming of variables in RL formulas. Also, we reformulate several lemmas from [15] (Lemmas 5, 7, and 8) such that they are self contained, and we update their proofs accordingly.

In Section 3.1 we introduce a set of axioms that capture only those constructs of ML needed in our proof, and in Section 3.2 we define RL, derivatives, and related notions. In Section 3.3 we present the procedure as a set of inference rules, and then, its encoding in Coq. We precisely show how the assumptions from [15] regarding variable renamings and well-formedness of RL formulas are dealt with. In Sections 3.4 and 3.5 we show the formalisation of the main lemmas and we emphasise the differences with respect to [15]. Finally, in Section 3.6 we present a complete formulation of the soundness theorem and its proof.

3.1 ML in Coq

A definition of ML such as in [15] would require us to formalise many things (algebraic specifications, FOL,...), which are irrelevant in our proof. Instead, we introduce a set of axioms that capture only what is needed by the proof: variables, ML implication ($\rightarrow$), the existential quantifier ($\exists$), a model M, a way to identify concrete program states in the model, and the RL satisfaction relation.

Variables (*Var*), program configurations (*State*), models (*Model*), and valuations (*Valuation*) are all parameters. From the ML syntax we only keep $\neg$, $\wedge$ (both used to express $\rightarrow$), and $\exists$, together with the corresponding axioms about the satisfaction relation $\models_{\text{ML}}$. To avoid ambiguities, we distinguish the logical connectives and quantifiers of ML ($\neg$, $\wedge$, $\exists$, ...) from those of Coq by using a bold font for the latter ($\boldsymbol{\neg}$, $\boldsymbol{\wedge}$, $\boldsymbol{\exists}$, ...):

$$\begin{array}{lll@{\qquad}rll} \mathit{Model} & : & \mathit{Type} & \mathit{State} & : & \mathit{Type} \\ \mathit{Var} & : & \mathit{Type} & \mathit{MLFormula} & : & \mathit{Type} \end{array}$$

$$\begin{array}{lll} \mathit{Valuation} & : & \mathit{Type} := \mathit{Var} \rightarrow \mathit{Model} \\ \models_{\text{ML}} & : & \mathit{State} \rightarrow \mathit{Valuation} \rightarrow \mathit{MLFormula} \rightarrow \mathit{Prop} \end{array}$$

$$\begin{array}{lll} \neg & : & \mathit{MLFormula} \rightarrow \mathit{MLFormula} \\ \wedge & : & \mathit{MLFormula} \rightarrow \mathit{MLFormula} \rightarrow \mathit{MLFormula} \\ \exists & : & \texttt{list}\ \mathit{Var} \rightarrow \mathit{MLFormula} \rightarrow \mathit{MLFormula} \end{array}$$

$$\begin{array}{lll} \models^{\neg} & : & \boldsymbol{\forall} \gamma\ \rho\ \varphi\ .\ (\gamma,\rho) \models_{\text{ML}} \neg\varphi \boldsymbol{\leftrightarrow} \boldsymbol{\neg}(\gamma,\rho) \models_{\text{ML}} \varphi \\ \models^{\wedge} & : & \boldsymbol{\forall} \gamma\ \rho\ \varphi\ \varphi'\ .\ (\gamma,\rho) \models_{\text{ML}} \varphi \wedge \varphi' \boldsymbol{\leftrightarrow} (\gamma,\rho) \models_{\text{ML}} \varphi \boldsymbol{\wedge} (\gamma,\rho) \models_{\text{ML}} \varphi' \\ \models^{\exists} & : & \boldsymbol{\forall} \varphi\ \gamma\ \rho\ V\ .\ (\gamma,\rho) \models_{\text{ML}} (\exists V)\varphi \boldsymbol{\leftrightarrow} \boldsymbol{\exists} \rho'\ .\ \boldsymbol{\forall} v\ .\ v \notin V \boldsymbol{\rightarrow} \rho'(v) = \rho(v) \boldsymbol{\wedge} (\gamma,\rho') \models \varphi \end{array}$$

The ML implication ($\rightarrow$) is defined using $\neg$ and $\wedge$:

$$\varphi \rightarrow \varphi' \quad \triangleq \quad \neg(\varphi \wedge \neg\varphi')$$

Also, the validity of an ML formula is defined as follows:

$$M \models_{\text{ML}} \varphi \quad \triangleq \quad \boldsymbol{\forall} \gamma\ \rho\ .\ (\gamma,\rho) \models \varphi$$

To collect the free variables of an ML formula we use a function FV, which takes a formula φ as an argument and returns the list of free variables occurring in φ. Because we do not instantiate the above axioms FV is a parameter:

$$\text{FV} \quad : \quad \mathit{MLFormula} \rightarrow \texttt{list}\ \mathit{Var}$$

In the proofs of our lemmas we often have to build new valuations from existing ones. For that we use the following function:

$$\varrho \quad : \quad \textit{Valuation} \to \textit{Valuation} \to \texttt{list}\ \textit{Var} \to \textit{Valuation}$$

Intuitively, $\varrho(\rho, \rho', V)$ returns a new valuation which is equal to ρ in all variables except for those in V, where $\varrho(\rho, \rho', V)$ is equal to ρ'. We capture this intuition in the following simple lemmas:

$$\begin{array}{lcl} \varrho^{\notin} & : & \forall x\ \rho\ \rho'\ V\ .\ x \notin V \to \varrho(\rho, \rho', V)(x) = \rho(x) \\ \varrho^{\in} & : & \forall x\ \rho\ \rho'\ V\ .\ x \in V \to \varrho(\rho, \rho', V)(x) = \rho'(x) \end{array}$$

If an ML formula φ is satisfied by a pair (γ, ρ), and none of its free variables are in V, then for any ρ', the pair $(\gamma, \varrho(\rho, \rho', V))$ also satisfies φ, because on those variables $\varrho(\rho, \rho', V)$ has the same effect as ρ. Conversely, if φ is satisfied by a pair (γ, ρ'), and all its free variables are included in V, then for any ρ, the pair $(\gamma, \varrho(\rho, \rho', V))$ also satisfies φ. Since the full definition of $\models_{\text{ML}}$ is abstract here, these properties are given as axioms:

$$\begin{array}{lcl} \models_{\varrho}^{\not\subseteq} & : & \forall \varphi\ \gamma\ \rho\ \rho'\ V\ .\ (\forall x\ .\ x \in \mathrm{FV}(\varphi) \to x \notin V) \land (\gamma, \rho) \models_{\text{ML}} \varphi \to (\gamma, \varrho(\rho, \rho', V)) \models_{\text{ML}} \varphi \\ \models_{\varrho}^{\subseteq} & : & \forall \varphi\ \gamma\ \rho\ \rho'\ V\ .\ \mathrm{FV}(\varphi) \subseteq V \land (\gamma, \rho') \models_{\text{ML}} \varphi \to (\gamma, \varrho(\rho, \rho', V)) \models_{\text{ML}} \varphi \end{array}$$

Another property which depends on $\models_{\text{ML}}$ is the one about the encoding of ML formulas into FOL: for all ρ and φ, $\rho \models_{\text{FOL}} \varphi^{=?}$ iff there is γ such that $(\gamma, \rho) \models_{\text{ML}} \varphi$. Here we do not define FOL, but we use the fact that any FOL formula is also an ML formula. Hence, the encoding will transform an ordinary ML formula into an ML formula without patterns, and the property above is axiomatised as:

$$\models^{=?} \quad : \quad \forall \gamma'\ \rho\ \varphi\ .\ (\gamma', \rho) \models_{\text{ML}} \varphi^{=?} \leftrightarrow (\exists \gamma)(\gamma, \rho) \models_{\text{ML}} \varphi$$

3.2 RL and derivatives

By definition, an RL formula is simply a pair of ML formulas:

$$\textit{RLFormula} \quad \triangleq \quad \textit{MLFormula} * \textit{MLFormula}$$

For clarity, in Coq we use the notation $\varphi \Rightarrow \varphi'$ instead of the pair notation (φ, φ'). In [15] the following assumption is made: for any RL formula $\varphi \Rightarrow \varphi'$ the condition $\mathrm{FV}(\varphi') \subseteq \mathrm{FV}(\varphi)$ holds. Here, we introduce the notion of *well-formed* RL formula:

$$\textit{wf}(\varphi \Rightarrow \varphi') \quad \triangleq \quad \mathrm{FV}(\varphi') \subseteq \mathrm{FV}(\varphi)$$

A set of RL formulas $\mathcal{S}$ defines a transition system over states:

$$\gamma \Rightarrow_{\mathcal{S}} \gamma' \quad \triangleq \quad \exists \varphi\ \varphi'\ \rho\ .\ \varphi \Rightarrow \varphi' \in \mathcal{S}\ \land (\gamma, \rho) \models_{\text{ML}} \varphi \land (\gamma', \rho) \models_{\text{ML}} \varphi'$$

In the rest of the paper we assume a given set of RL formulas $\mathcal{S}$. The satisfaction relation $\models_{\text{RL}}$ of RL is defined over execution paths τ, which could be either finite or infinite. Because of that, we formalise them as functions from *nat* to `option` *State* (where `option` *State* is the extension of *State* with an extra element `None`). The i^{th} element of a path τ is $\tau(i)$ and can be either a state γ_i or `None`. The subpath of τ which starts at position i is denoted $\tau|_i$. Also, a path is *well-formed*, written *wfPath*(τ), if every two consecutive states (say γ_i and γ_{i+1}) are in the transition relation given by $\mathcal{S}$ ($\gamma_i \Rightarrow_{\mathcal{S}} \gamma_{i+1}$), and for all i and j such that $i < j$, if $\tau(i) = $ `None` then $\tau(j) = $ `None`.

A path τ is infinite, written *infinite*(τ), if for all i, $\tau(i) \neq$ `None`. A configuration γ is terminating, denoted as *terminating*(γ), if there is no γ' such that $\gamma \Rightarrow_{\mathcal{S}} \gamma'$. A well-formed path τ is complete and has n transitions, written *complete*(τ, n) if τ is finite and *terminating*$(\tau(n))$. Also, a well-formed path τ is complete, denoted by *complete*(τ), if *infinite*(τ) or *complete*(τ, n) for some natural number n. Note that, if the path τ is complete and well-formed then for all i (if τ is finite then $i \leq n$) the subpath $\tau|_i$ is also complete and well-formed. This statement is assumed in [15] but in Coq we prove it explicitly. Finally, we say that (τ, ρ) *startsFrom* φ if $(\tau(0), \rho) \models_{\text{ML}} \varphi$.

The satisfaction relation of RL is defined for complete paths τ: $(\tau, \rho) \models_{\text{RL}} \varphi \Rightarrow \varphi'$ iff (τ, ρ) *startsFrom* φ and, either there is $i \geq 0$ such that $(\tau(i), \rho) \models_{\text{ML}} \varphi'$ or *infinite*(τ). Also, $\Rightarrow_{\mathcal{S}} \models_{\text{RL}} \varphi \Rightarrow \varphi'$ iff $(\tau, \rho) \models_{\text{RL}} \varphi \Rightarrow \varphi'$ for all pairs (τ, ρ) starting from φ with τ complete. Here are the exact definitions:

$$\begin{aligned}
(\tau, \rho) \models_{\text{RL}} \varphi \Rightarrow \varphi' \triangleq\ & (\tau, \rho)\ \textit{startsFrom}\ \varphi\ \wedge ((\exists\ i\ n\ \gamma'\ .\ i \leq n \wedge \textit{complete}(\tau, n) \wedge \\
& \qquad\qquad\qquad\qquad \tau(i) = \gamma' \wedge (\gamma', \rho) \models_{\text{ML}} \varphi') \\
& \vee\ \textit{infinite}(\tau)) \\
\Rightarrow_{\mathcal{S}} \models_{\text{RL}} \varphi \Rightarrow \varphi' \triangleq\ & \forall \tau\ \rho\ .\ \textit{wfPath}(\tau)\ \wedge \textit{complete}(\tau)\ \wedge (\tau, \rho)\ \textit{startsFrom}\ \varphi \wedge (\tau, \rho) \models_{\text{RL}} \varphi \Rightarrow \varphi' \\
\Rightarrow_{\mathcal{S}} \models_{\text{RL}} G \triangleq\ & \forall\ \varphi \Rightarrow \varphi'\ .\ \varphi \Rightarrow \varphi' \in G \rightarrow \Rightarrow_{\mathcal{S}} \models_{\text{RL}} \varphi \Rightarrow \varphi'
\end{aligned}$$

An ML formula φ is *$\mathcal{S}$-derivable* if there are γ, ρ, and γ' such that $(\gamma, \rho) \models_{\text{ML}} \varphi$ and $\gamma \Rightarrow_{\mathcal{S}} \gamma'$. The $\mathcal{S}$-derivability of an ML formula must not be confused with the notion of *$\mathcal{S}$-derivative*, i.e, the derivative with $\mathcal{S}$ of an ML formula (Section 2.1). To compute the $\mathcal{S}$-derivative of an ML formula in Coq, we use the following function:

$$\Delta_{\varphi_l \Rightarrow \varphi_r}(\varphi) \triangleq (\exists \text{FV}(\varphi_l))(\varphi_l \wedge \varphi)^{=?} \wedge \varphi_r.$$

Note that, unlike in the definition of derivatives from Section 2.1, here we quantify only the variables of φ_l because *wf*$(\varphi_l \Rightarrow \varphi_r)$, i.e., $\text{FV}(\varphi_r) \subseteq \text{FV}(\varphi_l)$. We use a similar notation for RL formulas:

$$\Delta_{\varphi_l \Rightarrow \varphi_r}(\varphi \Rightarrow \varphi') \triangleq (\exists \text{FV}(\varphi_l))(\varphi_l \wedge \varphi)^{=?} \wedge \varphi_r \Rightarrow \varphi'.$$

$\Delta_{\mathcal{S}}(\varphi \Rightarrow \varphi')$ returns the list of $\mathcal{S}$-derivatives of $\varphi \Rightarrow \varphi'$, and it is defined using $\Delta_{\varphi_l \Rightarrow \varphi_r}(\varphi \Rightarrow \varphi')$ for every rule in $\varphi_l \Rightarrow \varphi_r \in \mathcal{S}$.

Next, we introduce two relations $\approx$ and $\sim$, where the former is over RL formulas, and the latter is over pairs of ML formulas and valuations.

The relation $\approx$ is used to handle the renaming of free variables in RL formulas that is required when computing derivatives. More precisely, instead of computing $\Delta_{\varphi_l \Rightarrow \varphi_r}(\varphi \Rightarrow \varphi')$ we compute $\Delta_{\varphi_{l'} \Rightarrow \varphi_{r'}}(\varphi \Rightarrow \varphi')$, where $\varphi_{l'} \Rightarrow \varphi_{r'}$ is $\varphi_l \Rightarrow \varphi_r$ with renamed free variables such that $\text{FV}(\varphi_{l'}) \cap \text{FV}(\varphi) = \emptyset$. This disjointness condition is essential for computing derivatives: if this condition is not met, then there is no guarantee that the derivative above encodes the successors of configurations matching φ (cf. REMARK 1). The RL formula $\varphi_{l'} \Rightarrow \varphi_{r'}$ is a renamed version of $\varphi_l \Rightarrow \varphi_r$ and we express that using $\approx$: $\varphi_{l'} \Rightarrow \varphi_{r'} \approx \varphi_l \Rightarrow \varphi_r$.

If two RL formulas are related by $\approx$ then they denote the same set of transitions between configurations, but with different valuations. For instance, the transition from $\langle 7, 3\rangle$ to $\langle 7-3, 3\rangle$ is given by both $\varphi_0 \Rightarrow \varphi_0' \triangleq \langle x, y\rangle \Rightarrow \langle x - y, y\rangle$ and $\varphi_1 \Rightarrow \varphi_1' \triangleq \langle u, v\rangle \Rightarrow \langle u - v, v\rangle$ with different valuations: a valuation ρ such that $\rho(x) = 7$ and $\rho(y) = 3$ for the former, and a valuation ρ' such that $\rho'(u) = 7$ and $\rho'(v) = 3$ for the latter. Thus, $\langle 7, 3\rangle$ satisfies φ_0 with ρ and φ_1 with ρ',while $\langle 7-3, 3\rangle$ satisfies φ_0' with ρ and φ_1' with ρ'.

To express the fact that two RL formulas in relation $\approx$ denote the same transitions between configuration, but with different valuations, we use the relation $\sim$:

$$(\varphi, \rho) \sim (\varphi', \rho') \quad \triangleq \quad \forall\, \gamma \,.\, (\gamma, \rho) \models_{\text{ML}} \varphi \rightarrow (\gamma, \rho') \models_{\text{ML}} \varphi',$$

and the following property:

$$\begin{array}{lll} \approx^{\sim} & : & \forall\, \varphi_0\ \varphi'_0\ \varphi_1\ \varphi'_1 \,.\, \varphi_0 \Rightarrow \varphi'_0 \approx \varphi_1 \Rightarrow \varphi'_1 \rightarrow \\ & & \forall\, \rho \,.\, \exists\, \rho' \,.\, (\varphi_0, \rho) \sim (\varphi_1, \rho') \wedge (\varphi'_0, \rho) \sim (\varphi'_1, \rho') \end{array}$$

3.3 The procedure as a relation

The inference steps that the procedure in Figure 1 performs are summarised below; for the moment, let us ignore the fact that `prove` operates with sets of goals:

$$\frac{M \models_{\text{ML}} \varphi \rightarrow \varphi'}{\mathcal{S} \models_{\text{RL}} \varphi \Rightarrow \varphi'}[\text{IMPL}]$$

$$\frac{\varphi_c \Rightarrow \varphi'_c \in G_0 \quad M \models_{\text{ML}} \varphi \rightarrow \overline{\varphi}_c \quad \mathcal{S} \models_{\text{RL}} \Delta_{\{\varphi_c \Rightarrow \varphi'_c\}}(\varphi \Rightarrow \varphi')}{\mathcal{S} \models_{\text{RL}} \varphi \Rightarrow \varphi'}[\text{CIRC}]$$

$$\frac{\varphi \textit{ is } \mathcal{S}\textit{-derivable} \quad \mathcal{S} \models_{\text{RL}} \Delta_{\mathcal{S}}(\varphi \Rightarrow \varphi')}{\mathcal{S} \models_{\text{RL}} \varphi \Rightarrow \varphi'}[\text{SYMB}]$$

The set G_0 is the initial set of goals and $\overline{\varphi}_c \triangleq (\exists \text{FV}(\varphi_c))\varphi_c$. The connection with the procedure is straightforward: [IMPL] says that if the current goal $\varphi \Rightarrow \varphi'$ can be proved by checking the implication $M \models_{\text{ML}} \varphi \rightarrow \varphi'$ then $\mathcal{S} \models_{\text{RL}} \varphi \Rightarrow \varphi'$ (see line 3 of the procedure); [CIRC] captures the case when a circularity can be applied (lines 4 and 5); [SYMB] corresponds to the case when the current goal is discharged to its symbolic successors (lines 6 and 7). Thus, the procedure can be thought of as a strategy which applies the above rules in a fixed order: [IMPL], [CIRC], [SYMB].

The procedure shown in Figure 1 might not terminate. Because all Coq functions are terminating, the procedure cannot be encoded as a Coq function. Also, the procedure operates with a set of goals instead of a single goal. In our formalisation, we adapt the inference steps above to work with lists of goals, since in Coq it is more convenient to work with lists rather than sets. One could claim that working with lists might restrict the generality of the procedure, but our encoding does not rely on list-specific features. Another aspect that we have to deal with within our Coq formalisation concerns the renaming of free variables in RL formulas when computing derivatives. Thus, the inference steps [CIRC] and [SYMB] have to be changed such that they compute derivatives with conveniently renamed RL formulas.

In Coq, we introduce non-determinism by encoding the procedure as an inductive relation:

$$\frac{\varphi \Rightarrow \varphi' \in G \quad M \models_{\text{ML}} \varphi \to \varphi'}{step(G, G \setminus \{\varphi \Rightarrow \varphi'\})}[\text{IMPL}]$$

$$\frac{\varphi \Rightarrow \varphi' \in G \quad \begin{array}{c} \varphi_c \Rightarrow \varphi'_c \in G_0 \\ M \models_{\text{ML}} \varphi \to \overline{\varphi}_c \end{array} \quad \begin{array}{c} \varphi_c \Rightarrow \varphi'_c \approx \varphi_{c'} \Rightarrow \varphi'_{c'} \\ \text{FV}(\varphi_{c'}) \cap \text{FV}(\varphi) = \emptyset \end{array}}{step(G, G \setminus \{\varphi \Rightarrow \varphi'\} \cup \{\Delta_{\varphi_{c'} \Rightarrow \varphi'_{c'}}(\varphi \Rightarrow \varphi')\})}[\text{CIRC}]$$

$$\frac{\varphi \Rightarrow \varphi' \in G \quad \varphi \textit{ is } \mathcal{S}-\textit{derivable} \quad \begin{array}{c} \mathcal{S} \approx \mathcal{S}' \\ \text{FV}(\varphi) \cap \text{FV}(\mathcal{S}') = \emptyset \end{array}}{step(G, G \setminus \{\varphi \Rightarrow \varphi'\} \cup \Delta_{\mathcal{S}'}(\varphi \Rightarrow \varphi'))}[\text{SYMB}]$$

Here, $\text{FV}(\mathcal{S}')$ denotes the list of free variables that occur in all the RL formulas in $\mathcal{S}'$. By abuse of notation we write $\varphi \Rightarrow \varphi' \in G$ to denote the fact that $\varphi \Rightarrow \varphi'$ is in list G. Intuitively, *step* relates two lists of goals G and G', where G contains the current goal $\varphi \Rightarrow \varphi'$, and G' contains the remaining goals from $G \setminus \{\varphi \Rightarrow \varphi'\}$ and possibly the new goals generated by [CIRC] and [STEP]. At each step, only a single goal is removed from G, but zero, one, or more goals can be added. Note that, like in the original procedure (Figure 1) which uses sets, it does not matter which goal is chosen from the list G, we only require that $\varphi \Rightarrow \varphi' \in G$. Also, note that derivatives are always computed using equivalent (w.r.t. $\approx$) RL formulas whose variables are conveniently renamed.

A successful execution of the non-deterministic version of the procedure is defined using *steps*:

$$\frac{}{steps(\emptyset)}[\text{BASE}] \qquad \frac{step(G, G') \quad steps(G')}{steps(G)}[\text{STEPS}]$$

Intuitively, given a set of initial goals G, *step* is applied multiple times until the goals in G, together with the intermediary ones generated by [CIRC] and [SYMB], are all eliminated by [IMPL].

In [15], for a successful run of the procedure the set $\mathcal{F} \triangleq G_0 \cup \bigcup_i G_i$ is defined, where G_0 is the initial set of goals, $G_1 = \Delta_{\mathcal{S}}(G_0)$, and G_i ($i > 1$) are the sets of goals generated by each recursive call. In Coq, we define the following relation (which says when a goal g is in $\mathcal{F}$):

$$\frac{g \in G_0}{g \in \mathcal{F}}[\text{IN-}G_0] \qquad \frac{step(G, G') \quad g \in G \backslash G' \quad G' \subseteq \mathcal{F}}{g \in \mathcal{F}}[\text{IN-STEP}]$$

A goal $g \in \mathcal{F}$ if it is either in G_0, or there is step where g was eliminated (i.e., $step(G, G')$ with $g \in G \backslash G'$) and the remaining goals, including the ones introduced by this step, are also in $\mathcal{F}$. To ensure that this definition is consistent with the one in [15], we prove the following technical lemma:

Lemma 2 *For all lists of goals G, if $steps(G)$ then $G \subseteq \mathcal{F}$.*

Proof We give the proof for the sake of completeness. It goes by induction on $steps(G)$.

The [BASE] case is trivial because $G = \emptyset \subseteq \mathcal{F}$.

If [STEPS] was applied, then we have $step(G, G')$, $steps(G')$, and, the inductive hypothesis $G' \subseteq \mathcal{F}$. We choose an arbitrary $g \in G$. If g was eliminated by $step(G, G')$ then $g \in G \setminus G'$, and we get $g \in \mathcal{F}$ by applying [IN-STEP]. Otherwise, if g was not eliminated by $\mathit{step}(G, G')$, we trivially show that $g \in G'$ using the definition of *step*. Then, we use the inductive hypothesis $G' \subseteq \mathcal{F}$, and thus, $g \in \mathcal{F}$. Since g was arbitrarily chosen then $G \subseteq \mathcal{F}$. □

In particular, if $G \triangleq \Delta_{\mathcal{S}}(G_0)$ in LEMMA 2 then $\Delta_{\mathcal{S}}(G_0) \subseteq \mathcal{F}$. We note that *steps* and $\mathcal{F}$ encapsulate the entire execution of the procedure. They also address the monolithic nature of the proof.

3.4 Helper lemmas

In this section we present and prove two lemmas that correspond to the additional results (1) and (2) shown in Section 2.2. These lemmas have been also proved in [15], but here we reformulate them more precisely.

The first lemma states that every goal which is generated by a successful execution is either eliminated by [IMPL], or its left hand side is $\mathcal{S}$-derivable. In [15] this lemma is given inside the proof of another lemma, and inherently, it depends on several hypotheses of that lemma. Here, we disentangle the required hypotheses and we formulate it as a stand-alone lemma:

Lemma 3 *For all* RL *formulas $\varphi \Rightarrow \varphi' \in \mathcal{F}$, if $\mathcal{S}$ is total, all goals in G_0 are $\mathcal{S}$-derivable, and there are γ and ρ such that $(\gamma, \rho) \models_{\mathrm{ML}} \varphi$ then $M \models_{\mathrm{ML}} \varphi \to \varphi'$ or φ is $\mathcal{S}$-derivable.*

Proof. If $\varphi \Rightarrow \varphi' \in \mathcal{F}$ then either $\varphi \Rightarrow \varphi' \in G_0$ (cf. [IN-G_0]), or there is a step where $\varphi \Rightarrow \varphi'$ is eliminated, i.e, $\mathit{step}(G, G')$ with $\varphi \Rightarrow \varphi' \in G \setminus G'$, and $G' \subseteq \mathcal{F}$ (cf. [IN-STEP]):

- $\varphi \Rightarrow \varphi' \in G_0$: trivial, because all goals in G_0 are $\mathcal{S}$-derivable.
- In this case, $\varphi \Rightarrow \varphi'$ was eliminated when applying one of the rules [IMPL], [CIRC], or [SYMB]:

[IMPL] : Trivial: $M \models_{\mathrm{ML}} \varphi \to \varphi'$.

[CIRC]: There is $\varphi_c \Rightarrow \varphi'_c \in G_0$ such that $M \models_{\mathrm{ML}} \varphi \to \overline{\varphi}_c$. We have to show that there are γ_0, γ_1, and ρ' such that $(\gamma_0, \rho') \models_{\mathrm{ML}} \varphi$ and $\gamma_0 \Rightarrow_{\mathcal{S}} \gamma_1$. Let us choose $\gamma_0 \triangleq \gamma$ and $\rho' \triangleq \rho$. We easily obtain $(\gamma, \rho) \models_{\mathrm{ML}} \varphi$ from the hypothesis. In order to get a transition from γ we use the fact that $\mathcal{S}$ is total: first, $M \models_{\mathrm{ML}} \varphi \to \overline{\varphi}_c$ implies $(\gamma, \rho) \models \overline{\varphi}_c$, that is, there is a valuation ρ'' such that $\rho''(x) = \rho(x)$ for all $x \notin \mathrm{FV}(\varphi_c)$ and $(\gamma, \rho'') \models \varphi_c$ (cf. $\models^{\exists}$); second, from $(\gamma, \rho'') \models \varphi_c$, φ_c is $\mathcal{S}$-derivable (because $\varphi_c \Rightarrow \varphi'_c \in G_0$), and $\mathcal{S}$ total we obtain γ' such that $\gamma \Rightarrow_{\mathcal{S}} \gamma'$.

[SYMB]: Trivial: φ is $\mathcal{S}$-derivable. □

The second lemma states that for every transition $\gamma \Rightarrow_{\mathcal{S}} \gamma'$ which starts from φ there is a symbolic successor φ' of φ such that γ' is an instance of φ'. The lemma can be generalised to paths, that is, one can prove that for a concrete execution path there is a symbolic one which *covers* it. There are two important conditions required by this lemma: all rules from $\mathcal{S}$ have distinct free variables from variables in φ and all formulas from $\mathcal{S}$ and G_0 are well-formed. In [15] these conditions hold thanks to assumption (a_1). Here, we add these conditions as hypotheses:

Lemma 4 *For all transitions $\gamma \Rightarrow_{\mathcal{S}} \gamma'$, for all valuations ρ, and for all formulas φ, if $(\gamma, \rho) \models_{\mathrm{ML}} \varphi$, $f \in \mathcal{S} \cup G_0$ implies $\mathit{wf}(f)$, and $\mathrm{FV}(\mathcal{S}) \cap \mathrm{FV}(\varphi) = \emptyset$, then there are $\alpha \in \mathcal{S}$ and $\varphi' \triangleq \Delta_\alpha(\varphi)$ such that $(\gamma', \rho) \models_{\mathrm{ML}} \varphi'$.*

Proof The proof is constructive in the sense that we explicitly provide α and φ'. The transition $\gamma \Rightarrow_{\mathcal{S}} \gamma'$ implies that there is a rule $\varphi_l \Rightarrow \varphi_r \in \mathcal{S}$ and a valuation ρ' such that $(\gamma, \rho') \models \varphi_l$ and $(\gamma', \rho') \models \varphi_r$.

Let us choose $\alpha \triangleq \varphi_l \Rightarrow \varphi_r \in \mathcal{S}$ and $\varphi' \triangleq \Delta_{\varphi_l \Rightarrow \varphi_r}(\varphi)$. For the chosen φ' we have to prove $(\gamma', \rho') \models_{\text{ML}} \varphi'$, that is, $(\gamma', \rho') \models_{\text{ML}} (\exists \text{FV}(\varphi_l))(\varphi_l \wedge \varphi_r)^{=?} \wedge \varphi_r$. This is equivalent (cf. $\models^{\exists}$) to showing that there is a valuation $\rho'' : \mathit{Var} \to M$ such that $\rho''(x) = \rho'(x)$ for all $x \notin \text{FV}(\varphi_l)$ and $(\gamma', \rho'') \models_{\text{ML}} (\varphi_l \wedge \varphi_r)^{=?} \wedge \varphi_r$. Let $\rho'' \triangleq \varrho(\rho, \rho', \text{FV}(\varphi_l))$. By $\varrho^{\notin}$ we have $\rho''(x) = \varrho(\rho, \rho', \text{FV}(\varphi_l))(x) = \rho'(x)$ for all $x \notin \text{FV}(\varphi_l)$. There remains to show that $(\gamma', \rho'') \models_{\text{ML}} (\varphi_l \wedge \varphi_r)^{=?} \wedge \varphi_r$, that is (cf. $\models^{\wedge}$), $(\gamma', \varrho(\rho, \rho', \text{FV}(\varphi_l))) \models_{\text{ML}} (\varphi_l \wedge \varphi_r)^{=?}$ and $(\gamma', \varrho(\rho, \rho', \text{FV}(\varphi_l))) \models_{\text{ML}} \varphi_r$.

The first part is equivalent to showing that there is γ_0 such that $(\gamma_0, \varrho(\rho, \rho', \text{FV}(\varphi_l))) \models_{\text{ML}} \varphi_l \wedge \varphi_r$ (cf. $\models^{=?}$). Let $\gamma_0 \triangleq \gamma$. On one hand, from $(\gamma, \rho') \models_{\text{ML}} \varphi_l$ we obtain $(\gamma_0, \varrho(\rho, \rho', \text{FV}(\varphi_l))) \models_{\text{ML}} \varphi_l$ (cf. $\models^{\subseteq}_{\varrho}$), and on the other hand, from $(\gamma, \rho) \models_{\text{ML}} \varphi$ we obtain $(\gamma_0, \varrho(\rho, \rho', \text{FV}(\varphi_l))) \models_{\text{ML}} \varphi$ (cf. $\models^{\not\subseteq}_{\varrho}$). Then, using $\models^{\wedge}$ we obtain $(\gamma_0, \varrho(\rho, \rho', \text{FV}(\varphi_l))) \models_{\text{ML}} \varphi_l \wedge \varphi_r$.

Next, since $\varphi_l \Rightarrow \varphi_r \in \mathcal{S}$ then $\mathit{wf}(\varphi_l \Rightarrow \varphi_r)$, i.e, $\text{FV}(\varphi_r) \subseteq \text{FV}(\varphi_l)$. Therefore, using $\models^{\subseteq}_{\varrho}$ and the fact that $(\gamma', \rho') \models_{\text{ML}} \varphi_r$ we obtain $(\gamma', \varrho(\rho, \rho', \text{FV}(\varphi_l))) \models_{\text{ML}} \varphi_r$. □

3.5 The soundness for finite paths

The proof of the soundness theorem (sketched in Section 2.2) depends on an intermediate technical lemma (Lemma 1). In this section we reformulate the lemma by adding all the hypotheses needed by its proof so that it does not rely on hidden assumptions. Briefly, assuming a successful execution $\mathit{steps}(\Delta_{\mathcal{S}}(G_0))$, the lemma states that for all complete finite paths τ, and for all formulas $\varphi \Rightarrow \varphi' \in \mathcal{F}$ such that τ starts from φ with a valuation ρ, (τ, ρ) satisfies $\varphi \Rightarrow \varphi'$. The lemma can be thought of as a version of soundness for finite paths, which involves all the goals in $\mathcal{F}$, not only those from G_0. Its proof is by induction on the number of the transitions from the path τ.

With respect to [15], there are several significant differences. First, the proof takes into account the fact that derivatives are computed with renamed RL formulas. Second, we find and fix the flaw from the proof in [15], where a false assumption about the goals in $\mathcal{F}$ is made: given a successful execution of the procedure starting with $\Delta_{\mathcal{S}}(G_0)$, all the goals in $\mathcal{F}$ are eliminated by a step of the procedure. This cannot be true, since $\mathcal{F}$ includes G_0, but the procedure only processes goals starting with $\Delta_{\mathcal{S}}(G_0)$. The assumption does not invalidate the result from [15], but it makes the proof incomplete. In fact, the goals in G_0, which are of interest in the soundness theorem, are not handled. This flaw was detected only when formalising the proof in Coq. To fix it, we add a new hypothesis: the free variables that occur in rules in $\mathcal{S}$ and those that occur in rules in G_0 constitute disjoint sets (i.e., $\text{FV}(\mathcal{S}) \cap \text{FV}(G_0) = \emptyset$).

In the following, we formulate the lemma, and we emphasise the main changes that occur in this proof (w.r.t. to the one in [15]) using gray font.

Lemma 5 *For all finite and complete paths τ, for all valuations ρ, and for all formulas $\varphi \Rightarrow \varphi' \in \mathcal{F}$, if (τ, ρ) startsFrom φ, $G_0 \neq \emptyset$, wfPath(τ), steps$(\Delta_{\mathcal{S}}(G_0))$, $\mathcal{S}$ is total, the left hand sides of goals in G_0 are $\mathcal{S}$-derivable, all formulas in $\mathcal{S} \cup G_0$ are well formed, and* $\text{FV}(\mathcal{S}) \cap \text{FV}(G_0) = \emptyset$, *then $(\tau, \rho) \models_{\text{RL}} \varphi \Rightarrow \varphi'$.*

Proof Let n be the the number of transitions in τ. The proof is by well-founded induction on n:

Case $n = 0$. Here, $\tau \triangleq \gamma_0$ and $\mathit{complete}(\tau, 0)$. We use LEMMA 3 for $\varphi \Rightarrow \varphi' \in \mathcal{F}$ and $(\gamma_0, \rho) \models_{\text{ML}} \varphi$ (implied by (τ, ρ) *startsFrom* φ) to split the proof in two cases: $M \models_{\text{ML}} \varphi \to \varphi'$ or φ is $\mathcal{S}$-derivable:

- If $M \models_{\text{ML}} \varphi \to \varphi'$ and $(\gamma_0, \rho) \models_{\text{ML}} \varphi$ then $(\gamma_0, \rho) \models_{\text{ML}} \varphi'$. Finally, $(\tau, \rho) \models_{\text{RL}} \varphi \Rightarrow \varphi'$ is obtained directly using the definition of $\models_{\text{RL}}$ with $i = 0$, $n = 0$, and $\gamma' = \gamma_0$.

- If φ is $\mathcal{S}$-derivable then, by the definition of totality of $\mathcal{S}$ and $(\gamma_0, \rho) \models_{\text{ML}} \varphi$ there is γ' such that $\gamma \Rightarrow_{\mathcal{S}} \gamma'$. On the other hand, $\mathit{complete}(\tau, 0)$ implies $\mathit{terminating}(\gamma_0)$, which, by definition, means that there is no γ' such that $\gamma \Rightarrow_{\mathcal{S}} \gamma'$. Thus, we obtain a contradiction.

Case $n > 0$. Let $\tau \triangleq \gamma_0 \Rightarrow_{\mathcal{S}} \cdots \Rightarrow_{\mathcal{S}} \gamma_n$ and $\mathit{complete}(\tau, n)$. The inductive hypothesis holds for all complete paths of length strictly less than n, that are well-formed and start from the left hand side of a goal in $\mathcal{F}$. We have to show that $(\tau, \rho) \models_{\text{RL}} \varphi \Rightarrow \varphi'$, that is, (τ, ρ) *startsFrom* φ and there is γ_i such that $(\gamma_i, \rho) \models_{\text{ML}} \varphi'$. The first part is given already in the hypothesis of the lemma, so it remains to find γ_i such that $(\gamma_i, \rho) \models_{\text{ML}} \varphi'$.
If $\varphi \Rightarrow \varphi' \in \mathcal{F}$ then either $\varphi \Rightarrow \varphi' \in G_0$ (cf. [IN-G_0]), or there is a step where $\varphi \Rightarrow \varphi'$ is eliminated, i.e, $\mathit{step}(G, G')$ with $\varphi \Rightarrow \varphi' \in G \backslash G'$, and $G' \subseteq \mathcal{F}$ (cf. [IN-STEP]):

- This case was overlooked in the paper proof [15]. Assume $\varphi \Rightarrow \varphi' \in G_0$. Since τ has length $n > 0$ then there is at least one transition $\gamma_0 \Rightarrow_{\mathcal{S}} \gamma_1$ in τ. Using LEMMA 4 and $(\gamma_0, \rho) \models_{\text{ML}} \varphi$, there are $\alpha \in \mathcal{S}$ and $\varphi_1 \triangleq \Delta_\alpha(\varphi)$ such that $(\gamma_1, \rho) \models_{\text{ML}} \varphi_1$. Note that LEMMA 4 requires that $\varphi \Rightarrow \varphi'$ is well-formed and $\text{FV}(\varphi) \cap \text{FV}(\mathcal{S}) = \emptyset$, all formulas in $\mathcal{S}$ are well-formed, and all the free variables occurring in them are different from those in φ. All these can be trivially proved using the hypotheses of the lemma. First, from $\varphi \Rightarrow \varphi' \in G_0$, and $\text{FV}(\mathcal{S}) \cap \text{FV}(G_0) = \emptyset$ we get $\text{FV}(\varphi) \cap \text{FV}(\mathcal{S}) = \emptyset$. Second, if τ is complete and well-formed then $\tau|_1$ is also complete and well-formed. Finally, since τ has length $n (> 0)$ then $\tau|_1$ has length $n - 1$.

  The next step in the proof is to apply the inductive hypothesis for $\tau|_1$ and $\varphi_1 \Rightarrow \varphi'$. For this, we show that $\tau|_1$ *startsFrom* φ_1 and $\varphi_1 \Rightarrow \varphi' \in \mathcal{F}$. First, the fact that $\tau|_1$ *startsFrom* φ_1 is implied by $(\gamma_1, \rho) \models_{\text{ML}} \varphi_1$. Second, since $\varphi_1 \triangleq \Delta_\alpha(\varphi) \in \Delta_{\mathcal{S}}(\varphi)$ and $\varphi \Rightarrow \varphi' \in G_0$ then the RL formula $\varphi_1 \Rightarrow \varphi' \in \Delta_{\mathcal{S}}(G_0) \subseteq \mathcal{F}$ (cf. LEMMA 2 with $G \triangleq \Delta_{\mathcal{S}}(G_0)$). Now, by the inductive hypothesis, $(\tau|_1, \rho) \models_{\text{ML}} \varphi_1 \Rightarrow \varphi'$, i.e., there is γ_i such that $(\gamma_i, \rho) \models_{\text{ML}} \varphi'$.

- If $\mathit{step}(G, G')$, $\varphi \Rightarrow \varphi' \in G \setminus G'$, and $G' \subseteq \mathcal{F}$, then $\varphi \Rightarrow \varphi'$ was eliminated when applying one of the rules [IMPL], [CIRC], or [STEP]:

  [IMPL] : $\varphi \Rightarrow \varphi' \in G$ and $M \models \varphi \to \varphi'$. Trivial: from $(\gamma_0, \rho) \models_{\text{ML}} \varphi$ and $M \models \varphi \to \varphi'$ we have $(\gamma_0, \rho) \models_{\text{ML}} \varphi'$.

  [CIRC] : there are $\varphi_c \Rightarrow \varphi'_c \in G_0$ and $\varphi_{c'} \Rightarrow \varphi'_{c'}$ such that $M \models \varphi \to \overline{\varphi}_c$, $\varphi_c \Rightarrow \varphi'_c \approx \varphi_{c'} \Rightarrow \varphi'_{c'}$, and $\text{FV}(\varphi_{c'}) \cap \text{FV}(\varphi) = \emptyset$. Since $(\gamma_0, \rho) \models_{\text{ML}} \varphi$ and $M \models \varphi \to \overline{\varphi}_c$ then $(\gamma_0, \rho) \models_{\text{ML}} \overline{\varphi}_c$. Using $\models^\exists$ we obtain a valuation $\rho' : \mathit{Var} \to M$ such that $\rho'(x) = \rho(x)$ for all $x \notin \text{FV}(\varphi_{c'})$ and $(\gamma_0, \rho') \models_{\text{ML}} \varphi_c$. Because $n > 0$ the path τ contains at least the first transition $\gamma_0 \Rightarrow_{\mathcal{S}} \gamma_1$. Thus, by LEMMA 4, there are $\alpha \in \mathcal{S}$ and $\varphi_1 \triangleq \Delta_\alpha(\varphi_c)$ such that $(\gamma_1, \rho') \models_{\text{ML}} \varphi_1$. Since $(\gamma_1, \rho') \models_{\text{ML}} \varphi_1$ then $(\tau|_1, \rho')$ *startsFrom* φ_c. Also, from $\varphi_c \Rightarrow \varphi'_c \in G_0$ and $\varphi_1 \triangleq \Delta_\alpha(\varphi_c) \in \Delta_{\mathcal{S}}(\varphi_c)$ we obtain $\varphi_1 \Rightarrow \varphi'_c \in \Delta_{\mathcal{S}}(G_0) \subseteq \mathcal{F}$ (cf. LEMMA 2 with $G \triangleq \Delta_{\mathcal{S}}(G_0)$). The inductive hypothesis applied for $(\tau|_1, \rho')$ (the length of $\tau|_1$ is $n - 1 < n$) and $\varphi_1 \Rightarrow \varphi'_c$, implies $(\tau|_1, \rho') \models_{\text{RL}} \varphi_1 \Rightarrow \varphi'_c$, that is, there is γ_j such that $(\gamma_j, \rho') \models_{\text{ML}} \varphi'_c$.

  The next step in our proof is to show that $(\gamma_j, \rho) \models_{\text{ML}} \Delta_{\varphi_{c'} \Rightarrow \varphi'_{c'}}(\varphi)$. With respect to [15], this part of the proof follows the same pattern, but it takes into account that the derivative is computed using a renaming of the circularity. Also, here we do not rely on assumption (a_1) (Section 2.2).

By the definition of derivatives, we have to show $(\gamma_j, \rho) \models_{\text{ML}} (\exists(\text{FV}(\varphi_{c'}))(\varphi_{c'} \wedge \varphi)^{=?} \wedge \varphi'_{c'}$ (note that $\varphi_{c'} \Rightarrow \varphi'_{c'} \approx \varphi_c \Rightarrow \varphi'_c$ and $\text{FV}(\varphi_{c'}) \cap \text{FV}(\varphi_c) = \emptyset$). This reduces to showing that there is a valuation ρ'' such that $\rho''(x) = \rho(x)$ for all $x \notin \text{FV}(\varphi_{c'})$ and $(\gamma_j, \rho'') \models_{\text{ML}} (\varphi_{c'} \wedge \varphi)^{=?} \wedge \varphi'_{c'}$ (cf. $\models^{\exists}$).

Before providing ρ'', we apply $\approx^{\sim}$ for $\varphi_c \Rightarrow \varphi'_c \approx \varphi_{c'} \Rightarrow \varphi'_{c'}$ and ρ' to obtain a valuation $\rho^* : \mathit{Var} \to M$ such that $(\varphi_c, \rho') \sim (\varphi_{c'}, \rho^*)$ and $(\varphi'_c, \rho') \sim (\varphi'_{c'}, \rho^*)$.

Next, we consider $\rho'' = \varrho(\rho, \rho^*, \text{FV}(\varphi_{c'}))$. The condition $\varrho(\rho, \rho^*, \text{FV}(\varphi_{c'}))(x) = \rho(x)$ for all $x \notin \text{FV}(\varphi_{c'})$ is directly obtained using $\varrho^{\notin}$. Therefore, there remains to prove $(\gamma_j, \varrho(\rho, \rho^*, \text{FV}(\varphi_{c'}))) \models_{\text{ML}} (\varphi_{c'} \wedge \varphi)^{=?} \wedge \varphi'_{c'}$, that is, $(\gamma_j, \varrho(\rho, \rho^*, \text{FV}(\varphi_{c'}))) \models_{\text{ML}} (\varphi_{c'} \wedge \varphi)^{=?}$ and $(\gamma_j, \varrho(\rho, \rho^*, \text{FV}(\varphi_{c'}))) \models_{\text{ML}} \varphi'_{c'}$ (cf. $\models^{\wedge}$). The latter can be proved using $\models^{\subseteq}_{\varrho}$ in combination with $\text{FV}(\varphi'_{c'}) \subseteq \text{FV}(\varphi_{c'})$ (because $\varphi_{c'} \Rightarrow \varphi'_{c'}$) is well-formed), and then using $(\varphi'_c, \rho') \sim (\varphi'_{c'}, \rho^*)$ and the fact that $(\gamma_j, \rho') \models_{\text{ML}} \varphi'_c$. For the former, $(\gamma_j, \varrho(\rho, \rho^*, \text{FV}(\varphi_{c'}))) \models_{\text{ML}} (\varphi_{c'} \wedge \varphi)^{=?}$, we use $\models^{=?}$ and we equivalently prove that there is $\gamma = \gamma_0$ such that $(\gamma_0, \varrho(\rho, \rho^*, \text{FV}(\varphi_{c'}))) \models_{\text{ML}} \varphi_{c'} \wedge \varphi$: first, $(\gamma_0, \varrho(\rho, \rho^*, \text{FV}(\varphi_{c'}))) \models_{\text{ML}} \varphi_{c'}$ is obtained using $\models^{\subseteq}_{\varrho}$, $(\varphi_c, \rho') \sim (\varphi_{c'}, \rho^*)$, and $(\gamma_0, \rho') \models_{\text{ML}} \varphi_c$; second, $(\gamma, \varrho(\rho, \rho^*, \text{FV}(\varphi_{c'}))) \models_{\text{ML}} \varphi$ is given directly by $\models^{\not\subseteq}_{\varrho}$ and the fact that $(\gamma_0, \rho) \models_{\text{ML}} \varphi$.

Now, $(\gamma_j, \rho) \models_{\text{ML}} \Delta_{\varphi_{c'} \Rightarrow \varphi'_{c'}}(\varphi)$ implies $(\tau|_j, \rho)$ *startsFrom* $\Delta_{\varphi_{c'} \Rightarrow \varphi'_{c'}}(\varphi)$. Moreover, $\Delta_{\varphi_{c'} \Rightarrow \varphi'_{c'}}(\varphi \Rightarrow \varphi') \in G' \subseteq \mathcal{F}$ because it has been just added to G' by [CIRC]. Thus, we apply the inductive hypothesis for $(\tau|_j, \rho)$ (the length of $\tau|_j$ is $n - j < n$) and $\Delta_{\varphi_{c'} \Rightarrow \varphi'_{c'}}(\varphi \Rightarrow \varphi')$ to obtain $(\tau|_j, \rho) \models \Delta_{\varphi_{c'} \Rightarrow \varphi'_{c'}}(\varphi \Rightarrow \varphi')$, i.e, there is γ_i such that $(\gamma_i, \rho) \models_{\text{ML}} \varphi'$.

[SYMB] The path τ contains at least the first transition $\gamma_0 \Rightarrow_{\mathcal{S}} \gamma_1$ because $n > 0$.

In [15], the next step in the proof is to apply LEMMA 4 for $\gamma_0 \Rightarrow_{\mathcal{S}} \gamma_1$ and $(\gamma_0, \rho) \models_{\text{ML}} \varphi$. Here, we show first that $\gamma_0 \Rightarrow_{\mathcal{S}'} \gamma_1$. By definition, the transition $\gamma_0 \Rightarrow_{\mathcal{S}} \gamma_1$ implies that there is a rule $\varphi_l \Rightarrow \varphi_r \in \mathcal{S}$ and a valuation ρ' such that $(\gamma_0, \rho') \models_{\text{ML}} \varphi_l$ and $(\gamma_1, \rho') \models_{\text{ML}} \varphi_r$. Since $\mathcal{S} \approx \mathcal{S}'$ then for any $\varphi_l \Rightarrow \varphi_r \in \mathcal{S}$ there is $\varphi_{l'} \Rightarrow \varphi_{r'} \in \mathcal{S}'$ such that $\varphi_l \Rightarrow \varphi_r \approx \varphi_{l'} \Rightarrow \varphi_{r'}$. This implies that there is ρ'' such that $(\varphi_l, \rho') \sim (\varphi_{l'}, \rho'')$ and $(\varphi_r, \rho') \sim (\varphi_{r'}, \rho'')$ (cf. $\approx^{\sim}$). By the definition of $\sim$, we obtain $(\gamma_0, \rho'') \models_{\text{ML}} \varphi_{l'}$ and $(\gamma_1, \rho'') \models_{\text{ML}} \varphi_{r'}$; thus, $\gamma_0 \Rightarrow_{\varphi_{l'} \Rightarrow \varphi_{r'}} \gamma_1$, i.e, $\gamma_0 \Rightarrow_{\mathcal{S}'} \gamma_1$.

Then, since $(\gamma_0, \rho) \models_{\text{ML}} \varphi$ and $\gamma_0 \Rightarrow_{\mathcal{S}'} \gamma_1$, by LEMMA 4 there are $\alpha \in \mathcal{S}'$ and $\varphi_1 \triangleq \Delta_\alpha(\varphi)$ such that $(\gamma_1, \rho) \models_{\text{ML}} \varphi_1$. Note that LEMMA 4 requires that $\varphi \Rightarrow \varphi'$ is well-formed (from the inductive hypothesis), $\text{FV}(\varphi) \cap \text{FV}(\mathcal{S}') = \emptyset$ (again, from the inductive hypothesis), and all formulas in $\mathcal{S}'$ are well-formed (because those in $\mathcal{S}$ are well-formed).

The next step in the proof is to apply the inductive hypothesis for $\tau|_1$ (which is well-formed and complete of length $n - 1$) and $\varphi_1 \Rightarrow \varphi'$. On the one hand, we have $(\gamma_1, \rho) \models_{\text{ML}} \varphi_1$ and thus, $\tau|_1$ *startsFrom* φ_1. On the other hand, since $\varphi_1 \triangleq \Delta_\alpha(\varphi) \in \Delta_{\mathcal{S}'}(\varphi)$ then the RL formula $\varphi_1 \Rightarrow \varphi' \in \Delta_{\mathcal{S}'}(G_0) \subseteq G' \subseteq \mathcal{F}$ because $\Delta_{\mathcal{S}'}(G_0)$ has been just added to G' by [SYMB]. By the inductive hypothesis we obtain $(\tau|_1, \rho) \models_{\text{ML}} \varphi_1 \Rightarrow \varphi'$, i.e, there is γ_i such that $(\gamma_i, \rho) \models_{\text{ML}} \varphi$. □

3.6 The soundness theorem

In this section we show the formalisation of the soundness theorem and its proof. Several assumptions from [15] are here transformed in hypotheses: all formulas in $\mathcal{S}$ and G_0 are well formed and for all the goals $\varphi \Rightarrow \varphi'$ in G_0 φ is $\mathcal{S}$-derivable. Moreover, we add the additional hypothesis required by LEMMA 5: $\mathrm{FV}(\mathcal{S}) \cap \mathrm{FV}(G_0) = \emptyset$.

Theorem 2 *If $\mathcal{S}$ is total, all formulas in $\mathcal{S} \cup G_0$ are well-formed, the left hand sides of goals in G_0 are $\mathcal{S}$-derivable, $\mathrm{FV}(\mathcal{S}) \cap \mathrm{FV}(G_0) = \emptyset$, and steps$(\Delta_{\mathcal{S}}(G_0))$ then $\Rightarrow_{\mathcal{S}} \models_{\mathrm{RL}} G_0$.*

Proof. We arbitrarily choose $\varphi \Rightarrow \varphi' \in G_0$. $\mathcal{S} \models_{\mathrm{RL}} \varphi \Rightarrow \varphi'$ iff for all complete paths τ and all valuations $\rho : Var \to M$ with (τ, ρ) *startsFrom* φ we have $(\tau, \rho) \models_{\mathrm{RL}} \varphi \Rightarrow \varphi'$. If τ is infinite, then, by definition, $(\tau, \rho) \models_{\mathrm{RL}} \varphi \Rightarrow \varphi'$. If τ is finite of length n, then, by LEMMA 5 (note that $\varphi \Rightarrow \varphi' \in G_0 \subseteq \mathcal{F}$ and $G_0 \neq \emptyset$), $(\tau, \rho) \models_{\mathrm{RL}} \varphi \Rightarrow \varphi'$. Since $\varphi \Rightarrow \varphi'$ was arbitrarily chosen from G_0, we have $\mathcal{S} \models G_0$. $\square$

Overview of the formal proof The formal proof of the soundness of the procedure was developed using version `8.4pl5` of the Coq proof assistant. The code can be found at: `https://fmse.info.uaic.ro/imgs/soundness-proof.tar.gz`. The archive also contains additional information about the code organisation and instructions for compilation. The Coq code is spread over several files. The main file is called `sound.v` and it contains the definitions for *step* and *steps*; it also includes several helper lemmas and, at the end of the file, the main lemmas and the soundness theorem shown in Section 3. The axioms that we use for ML (shown in Section 3.1) can be found in `ml.v`. The definition of RL, derivatives, and the other related notions from Section 3.2 are located in `rl.v` and `derivatives.v`.

4 Conclusions

In this paper we present the Coq formalisation of the soundness of a procedure for program verification that was proposed in [15]. The procedure is formalised as an inductive relation, and, unlike in [15], it explicitly handles the renaming of the free variables in RL formulas when computing derivatives. We precisely formulate the intermediary lemmas and the soundness theorem, and then we prove them in Coq. During the development of the proof in Coq we discovered (and fixed) some flaws in the proof from [15].

Alternative proofs for the soundness of the procedure from [15] can be found in [2] and [3]. However, those are less suitable to be encoded in Coq: the proof in [2] is more intuitive (it uses a digraph for visualisation) but not that precise, while the one in [3] uses coinduction; however, Coq's support for coinduction is not as good as the one for induction (e.g., no coinduction principle is generated automatically).

Future work In terms of future work there are several directions to follow. The first one consists in extracting a certified OCaml program from our inductive relation, and then use it for verification within the $\mathbb{K}$ framework. This requires RL-based semantics of languages in OCaml, which could be extracted from corresponding semantics in Coq, an effort already underway in the $\mathbb{K}$ team, who are developing a Coq backend [17] for $\mathbb{K}$.

Another future work is related to a *computational* definition of derivatives, based on rewriting, which is also proposed in [15]. It is there shown equivalent to the current (logical) definition of derivatives. A step further would be to formalise that proof of equivalence as well, thereby obtaining a more efficient certified program-verification procedure.

References

[1] Andrew W. Appel, Robert Dockins, Aquinas Hobor, Lennart Beringer, Josiah Dodds, Gordon Stewart, Sandrine Blazy, and Xavier Leroy. *Program Logics for Certified Compilers.* Cambridge University Press, New York, NY, USA, 2014.

[2] Andrei Arusoaie, Dorel Lucanu, David Nowak, and Vlad Rusu. Verifying Reachability-Logic Properties on Rewriting-Logic Specifications. Technical Report TR 15-01, "Al.I.Cuza" University of Iaşi, Faculty of Computer Science, 2015. `http://www.infoiasi.ro/~tr/tr.pl.cgi`.

[3] Andrei Arusoaie, Dorel Lucanu, and Vlad Rusu. A Generic Framework for Symbolic Execution. Research Report RR-8189, Inria, September 2015.

[4] Frédéric Besson. Fast reflexive arithmetic tactics the linear case and beyond. In *Types for Proofs and Programs, International Workshop, TYPES 2006, Nottingham, UK, April 18-21, 2006, Revised Selected Papers*, pages 48–62, 2006.

[5] Denis Bogdănaş and Grigore Roşu. K-Java: A Complete Semantics of Java. In *Proceedings of the 42nd Symposium on Principles of Programming Languages (POPL'15)*, pages 445–456. ACM, January 2015.

[6] Thomas Braibant and Damien Pous. Deciding kleene algebras in coq. *Logical Methods in Computer Science*, 8(1), 2012.

[7] A. Chaieb and T. Nipkow. Verifying and reflecting quantifier elimination for Presburger arithmetic. In G. Stutcliffe and A. Voronkov, editors, *Logic for Programming, Artificial Intelligence, and Reasoning*, volume 3835 of *LNAI*. Springer, 2005.

[8] Traian Florin Şerbănuţă, Andrei Arusoaie, David Lazar, Chucky Ellison, Dorel Lucanu, and Grigore Roşu. The primer (version 3.3). *Electronic Notes in Theoretical Computer Science*, 304:57 – 80, 2014. Proceedings of the Second International Workshop on the K Framework and its Applications (K 2011).

[9] Andrei Ştefănescu, Ştefan Ciobâcă, Radu Mereuţă, Brandon M. Moore, Traian Florin Şerbănuţă, and Grigore Roşu. All-path reachability logic. In *Proceedings of the Joint 25th International Conference on Rewriting Techniques and Applications and 12th International Conference on Typed Lambda Calculi and Applications (RTA-TLCA'14)*, volume 8560 of *LNCS*, pages 425–440. Springer, July 2014.

[10] Chucky Ellison and Grigore Roşu. An executable formal semantics of C with applications. In *Proceedings of the 39th Symposium on Principles of Programming Languages (POPL'12)*, pages 533–544. ACM, 2012.

[11] Robert W. Floyd. Assigning meanings to programs. *Proceedings of Symposium on Applied Mathematics*, 19:19–32, 1967.

[12] David Harel, Dexter Kozen, and Jerzy Tiuryn. Dynamic logic. *SIGACT News*, 32(1):66–69, 2001.

[13] Chris Hathhorn, Chucky Ellison, and Grigore Roşu. Defining the undefinedness of c. In *Proceedings of the 36th ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI'15)*, pages 336–345. ACM, June 2015.

[14] Charles Antony Richard Hoare. An Axiomatic Basis for Computer Programming. *Comm. ACM*, 12(10):576–580, 583, October 1969.

[15] Dorel Lucanu, Vlad Rusu, Andrei Arusoaie, and David Nowak. Verifying reachability-logic properties on rewriting-logic specifications. In Narciso Martí-Oliet, Peter Csaba Ölveczky, and Carolyn L. Talcott, editors, *Logic, Rewriting, and Concurrency - Essays dedicated to José Meseguer on the Occasion of His 65th Birthday*, volume 9200 of *Lecture Notes in Computer Science*, pages 451–474. Springer, 2015.

[16] Nicolas Marti, Reynald Affeldt, and Akinori Yonezawa. Formal verification of the heap manager of an operating system using separation logic. In *Proceedings of the 8th International Conference on Formal Methods and Software Engineering*, ICFEM'06, pages 400–419, Berlin, Heidelberg, 2006. Springer-Verlag.

[17] Brandon Moore and Grigore Roşu. Program verification by coinduction. Technical Report http://hdl.handle.net/2142/73177, University of Illinois, February 2015.

[18] Peter W. O'Hearn and David J. Pym. The logic of bunched implications. *Bulletin of Symbolic Logic*, 5(2):215–244, 1999.

[19] Daejun Park, Andrei Ştefănescu, and Grigore Roşu. KJS: A complete formal semantics of JavaScript. In *Proceedings of the 36th ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI'15)*, pages 346–356. ACM, June 2015.

[20] John C. Reynolds. Separation Logic: A Logic for Shared Mutable Data Structures. In *LICS*, pages 55–74, 2002.

[21] Grigore Roşu. Matching logic — extended abstract. In *Proceedings of the 26th International Conference on Rewriting Techniques and Applications (RTA'15)*, volume 36 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 5–21, Dagstuhl, Germany, July 2015. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik.

[22] Grigore Roşu and Traian Florin Şerbănuţă. K overview and simple case study. In *Proceedings of International K Workshop (K'11)*, volume 304 of *ENTCS*, pages 3–56. Elsevier, June 2014.

[23] Grigore Roşu and Andrei Ştefănescu. Matching Logic: A New Program Verification Approach (NIER Track). In *ICSE'11: Proceedings of the 30th International Conference on Software Engineering*, pages 868–871. ACM, 2011.

[24] Grigore Roşu and Andrei Ştefănescu. Checking reachability using matching logic. In *Proceedings of the 27th Conference on Object-Oriented Programming, Systems, Languages, and Applications (OOPSLA'12)*, pages 555–574. ACM, 2012.

[25] Grigore Roşu and Andrei Ştefănescu. From hoare logic to matching logic reachability. In *Proceedings of the 18th International Symposium on Formal Methods (FM'12)*, volume 7436 of *Lecture Notes in Computer Science*, pages 387–402. Springer, 2012.

[26] Grigore Roşu and Andrei Ştefănescu. Towards a unified theory of operational and axiomatic semantics. In *Proceedings of the 39th International Colloquium on Automata, Languages and Programming (ICALP'12)*, volume 7392 of *Lecture Notes in Computer Science*, pages 351–363. Springer, 2012.

[27] Grigore Roşu, Andrei Ştefănescu, Ştefan Ciobâcă, and Brandon M. Moore. One-path reachability logic. In *Proceedings of the 28th Symposium on Logic in Computer Science (LICS'13)*, pages 358–367. IEEE, June 2013.

[28] Grigore Roşu, Chucky Ellison, and Wolfram Schulte. Matching logic: An alternative to Hoare/Floyd logic. In Michael Johnson and Dusko Pavlovic, editors, *Proceedings of the 13th International Conference on Algebraic Methodology And Software Technology (AMAST '10)*, volume 6486 of *Lecture Notes in Computer Science*, pages 142–162, 2010.

[29] Grigore Roşu and Traian Florin Şerbănuţă. An overview of the K semantic framework. *Journal of Logic and Algebraic Programming*, 79(6):397–434, 2010.

[30] Norbert Schirmer. A verification environment for sequential imperative programs in isabelle/hol. In Franz Baader and Andrei Voronkov, editors, *Logic for Programming, Artificial Intelligence, and Reasoning*, volume 3452 of *Lecture Notes in Computer Science*, pages 398–414. Springer Berlin Heidelberg, 2005.

[31] Harvey Tuch and Gerwin Klein. A unified memory model for pointers. In *Proceedings of the 12th International Conference on Logic for Programming, Artificial Intelligence and Reasoning*, pages 474–488, Montego Bay, Jamaica, dec 2005.

[32] Tjark Weber. Towards mechanized program verification with separation logic. In Jerzy Marcinkowski and Andrzej Tarlecki, editors, *Computer Science Logic – 18th International Workshop, CSL 2004, 13th Annual Conference of the EACSL, Karpacz, Poland, September 2004, Proceedings*, volume 3210 of *Lecture Notes in Computer Science*, pages 250–264. Springer, September 2004.

Publisher
Inria
Domaine de Voluceau - Rocquencourt
BP 105 - 78153 Le Chesnay Cedex
inria.fr

ISSN 0249-6399