EPiC Series in Computing

Volume 94, 2023, Pages 405–456

Proceedings of 24th International Conference on Logic for Programming, Artificial Intelligence and Reasoning

Cartesian Reachability Logic: A Language-Parametric Logic for Verifying k-Safety Properties

Jan Tušil[1], Traian Florin Șerbănuță[2], and Jan Obdržálek[1]

[1] Masaryk University, Brno, Czech Republic
jan.tusil@mail.muni.cz,obdrzalek@fi.muni.cz
[2] University of Bucharest, Bucharest, Romania
traian.serbanuta@unibuc.ro

Abstract

We introduce a language-parametric calculus for k-safety verification - Cartesian Reachability logic (CRL).

In recent years, formal verification of hyperproperties has become an important topic in the formal methods community. An interesting class of hyperproperties is known as k-safety properties, which express the absence of a *bad* k-tuple of execution traces. Many security policies, such as *noninterference*, and functional properties, such as commutativity, monotonicity, and transitivity, are k-safety properties. A prominent example of a logic that can reason about k-safety properties of software systems is Cartesian Hoare logic (CHL). However, CHL targets a specific, small imperative language. In order to use it for sound verification of programs in a different language, one needs to extend it with the desired features or hand-craft a translation. Both these approaches require a lot of tedious, error-prone work.

Unlike CHL, CRL is language-parametric: it can be instantiated with an operational semantics (of a certain kind) of any deterministic language. Its soundness theorem is proved once and for all, with no need to adapt or re-prove it for different languages or their variants. This approach can significantly reduce the development costs of tools and techniques for sound k-safety verification of programs in deterministic languages: for example, of smart contracts written for EVM (the language powering the Ethereum blockchain), which already has an operational semantics serving as a reference.

1 Introduction

Recent years have witnessed an increased interest in formal verification of *hyperproperties* [12]. Unlike properties, whose validity depends on a single execution trace, hyperproperties can relate multiple program executions. A particularly interesting class of hyperproperties are *k-safety (hyper-)properties* [17, 28, 1, 12] (first introduced in [12]). A k-safety property is a hyperproperty whose violation can be witnessed by a k-tuple of execution traces. Many *security policies* - for example, *noninterference* (requiring that sensitive or privileged data do not influence insensitive or unprivileged computations) or *observational determinism* - are k-safety hyperproperties [10, 11, 12]. Similarly, many functional correctness properties are

actually k-safety hyperproperties; for example, *transitivity* (which needs to be satisfied, e.g., by comparators when managing data in collections), associativity (important in the map/reduce paradigm), or monotonicity [28].

Techniques and tools for verifying hyperproperties of (finite-state) hardware [13, 18], as well as (infinite-state) software systems have been developed. For verification of software systems in particular, *Cartesian Hoare logic* (CHL), introduced in [28], extends Hoare logic to allow reasoning about *k-safety hyperproperties*. In [28], the authors not only managed to give CHL a sound and relatively complete proof system but also successfully used their logic to analyze several natural k-safety properties of Java programs. (The verification algorithm was even implemented in a fully automated tool.) To formalize Cartesian Hoare Logic, [28] uses a simple imperative language, whose looping constructs are while-loops with breaks. However, extending the approach of [28] to other constructs affecting control flow, or indeed other programming languages, can be both highly non-trivial and time-consuming.

On the other hand, there have been recent developments in the area of *language-parametric* software verification. *Reachability logic* (RL) [26, 27, 32] is a formalism for reasoning about partial correctness of software in the spirit of Hoare logics. Being implemented in the $\mathbb{K}$ framework [7], its biggest advantage is that reachability logic is *language parametric*: its proof system can be used unchanged to reason about programs in any language, as long as the language has a formal semantics in RL. Therefore, researchers no longer need to think about a particular language construct three times (once for the operational semantics, once for axiomatic, and once for the correspondence); additionally, a single researcher or an architect of a tool does not need to understand both the precise (and often intricate) semantics of a programming language *and* formal verification techniques, which makes *division of labor* possible. Through $\mathbb{K}$, reachability logic has been used to build verifiers for real-world languages, such as C ([19]), Java ([30]), JavaScript ([30]), and EVM ([24]).

In this paper, we argue that we can indeed have the best of both worlds. We propose a new logic called *Cartesian Reachability logic (CRL)*, which properly extends reachability logic to allow reasoning about k-safety hyperproperties. Similarly to CHL, CRL has a sound and relatively complete proof system. A major advantage of CRL against CHL is that it works with any deterministic language for which RL works; that is, with any deterministic language which has a RL-based formal semantics. This makes CRL applicable for programs (aka "smart contracts") running on a blockchain since the languages used there are typically deterministic and because many of them (e.g., EVM [20] powering the Ethereum blockchain, IELE [21] integrated into the Cardano blockchain) already have a RL-based formal semantics.

CRL does *not* extend CHL, because the two logics give different semantics to properties of nondeterministic programs; despite this distinction, CRL extends CHL on the deterministic fragment of the CHL-supported language. We elaborate on this relation in Section 5. We draw our inspiration from the literature on language-independent verification of partial correctness ([26, 27, 32]) and program equivalence ([9, 8]).

Contributions The approach of our paper can be summarized as follows:

- We propose Cartesian Reachability Logic, an extension of reachability logic for reasoning about k-safety properties along the lines of Cartesian Hoare Logic.
- We define a *language-parametric alternative to self-composition* ([2, 14]) (Section 3.3) and establish a relation between CRL validity of the original and RL validity of the composed system.

- We give CRL a sound and relatively complete *proof system* (Section 3.4). The proofs in this proof system can be translated to ordinary RL proofs of the composed system (for soundness), but also such that it allows relatively high-level reasoning about *circular behavior* and *lockstep execution* (for ease of verification and simplicity of invariants).

2 Preliminaries

2.1 Cartesian Hoare logic

Introduced in [28], Cartesian Hoare Logic is a formalism for specifying and reasoning about k-safety properties, in a similar way as Hoare logic is used to reason about (safety) properties. In Hoare logic, properties are specified by means of so-called *Hoare triples*. These have the shape $\{\varphi\}S\{\psi\}$, with the meaning that the formula ψ holds in any state after the termination (if any) of the program S, executed from a state satisfying φ. In Cartesian Hoare logic, the situation is similar: one can specify a triple $\langle\Phi\rangle\ S_1 \circledast \cdots \circledast S_k\ \langle\Psi\rangle$ with the following meaning: for any k-tuple $(\sigma_1, \ldots, \sigma_k)$ of states satisfying the formula Φ, if we execute each program S_i in the respective state σ_i and they all terminate, then the k-tuple $(\sigma'_1, \ldots, \sigma'_k)$ of the resulting program states satisfies Ψ [1].

As an example, consider the program $P(x, y) \equiv$ `while(x > 0) { x--; y++; }` and the 2-safety property of *monotonicity* stating, intuitively: with growing inputs x and y, the resulting y also grows. In CHL, this can be formalized as

$$\langle x_1 \leq x_2 \wedge y_1 \leq y_2\rangle\ P(x_1, y_1) \circledast P(x_2, y_2)\ \langle y_1 \leq y_2\rangle\,. \tag{1}$$

The main idea here is that the formulas in the precondition and postcondition can *relate* variables from different executions. Cartesian Hoare logic is equipped with a proof system that allows one to prove the validity of CHL triples. This proof system contains rules[2] like

$$\frac{\langle\Phi \wedge c\rangle\ (\texttt{B1; S}) \circledast R\ \langle\Psi\rangle \qquad \langle\Phi \wedge \neg c\rangle\ (\texttt{B2; S}) \circledast R\ \langle\Psi\rangle}{\langle\Phi\rangle\ (\texttt{(if (c) B1 else B2); S}) \circledast R\ \langle\Psi\rangle} \tag{2}$$

that replicate standard Hoare-logic reasoning, and is sound and complete. However, what is more interesting, the proof system allows one to perform *lockstep reasoning*, even for loops. This is achieved by means of the following rule (version for two executions):

$$\frac{\Phi \Rightarrow I \quad \langle I \wedge c_1 \wedge c_2\rangle\ \texttt{B1} \circledast \texttt{B2}\ \langle I\rangle \quad \langle I \wedge \neg c_2\rangle\ \texttt{while}(c_1)\texttt{B1}\ \langle\Psi\rangle \quad \langle I \wedge \neg c_1\rangle\ \texttt{while}(c_2)\texttt{B2}\ \langle\Psi\rangle}{\langle\Phi\rangle\ (\texttt{while }(c_1)\texttt{ B1}) \circledast (\texttt{while}(c_2)\texttt{ B2})\ \langle\Psi\rangle}$$

Note that the invariant I, assumed by this rule, can relate variables from *both* executions. The rule breaks reasoning about a pair of loops into three cases: the case where both loop conditions hold and two cases where one of the conditions does not hold. In the first case, both loops are executed "in lockstep", performing one iteration each, and their execution must preserve the invariant. In the remaining two cases, only one of the loops executes (in a state satisfying the invariant and negation of the other loop condition), resulting in a state satisfying the postcondition.

[1] An important technical assumption here is that every program S_i operates on its own set of program variables, distinct from variables of other programs S_j (for $i \neq j$) - otherwise, the formulas Φ and Ψ would not be able to distinguish between program variables of different programs.

[2] We have changed the notation slightly compared to the original paper.

This lockstep reasoning is a powerful tool because the required invariants (relating different executions) are often very simple. For example, consider the program $P(x, y)$ and the formula (1) above. In this case it is enough to choose the invariant to be the same as the precondition (i.e. $I \equiv x_1 \leq x_2 \wedge y_1 \leq y_2$). To prove the above example without lockstep reasoning, one must find (non-relational) loop invariants strong enough to summarize the whole loop.

Unfortunately, lockstep reasoning rules become more complicated as one adds other features into the language – for example, the `break` statement (as done in [28]). It is not immediately obvious how to extend this approach to handle, e.g., recursion, `continue`, or `goto`. We also observe that a single language feature (`while` loops) needed to be considered five times in order for CHL to support it soundly: the semantics of `while` is present in the operational semantics of the target language, in the Hoare logic for that language, in the Cartesian Hoare logic for that language, and in the proofs of soundness for both of the logics.

This paper aims to make the above ideas available for any deterministic language. Therefore, we review some tools from recent literature on language-parametric program verification in the following subsections.

2.2 Matching Logic

Before introducing reachability logic, we must first talk about matching logic, on top of which reachability logic is built. We work with the variant of matching logic described in [32, 27]. (There are other variants of matching logic, e.g. [6, 5], which are of no particular interest for this paper.)

A matching logic *formula* (commonly known as a *pattern*) is a first-order logic (FOL) formula which additionally allows terms (with variables) over some signature Σ as nullary predicates (we refer to these as "terms-as-predicates"). To enable reasoning about programming language syntax and semantics, the signature often contains the syntactical constructs of a programming language of interest. A typical example of a matching logic formula is $\varphi_{example}$, defined as[3]

$$\varphi_{example} \equiv \quad \ll \texttt{x--;} \mid \texttt{x} \mapsto X \gg \wedge\ (X >_{\text{Int}} 1 = \mathit{true}) \tag{3}$$

which, when interpreted in a model of a particular programming language, denotes the set of program configurations in which "`x--;`" is the code to be executed next, and the program variable `x` has a value X that is greater than 1. In this example, the subformula $\ll$ `x--;` | `x` $\mapsto X \gg$ is a nullary term used as a predicate (term-as-predicate), with X being the only free FOL variable. (`x` is not a FOL variable but a constant symbol from the signature of the programming language.) The subterm `x` $\mapsto X$ states that the program variable `x` has value X, and the $X >_{\text{Int}} 1 = \mathit{true}$ part then says that the realization of the function symbol "$>_{\text{Int}}$" returns the boolean value *true* (another constant symbol from the signature) when given X and 1 as arguments.

A matching logic Σ-*model* $\mathcal{T}$ is a Σ-algebra with non-empty carrier sets. The satisfaction relation $(M, \gamma, \rho) \models \varphi$ for a model M, a model element $\gamma \in M$, an M-valuation $\rho : \mathit{Var} \to M$, and a pattern φ, is defined inductively on the structure of φ. The definition is as in FOL; the main difference is the semantics of terms-as-predicates, which is given as

$$(M, \gamma, \rho) \models t \iff \gamma = \rho(t) \text{ if t is a term}$$

(where $\rho(t)$ is the homomorphic extension of ρ applied to the term t). For example, we might have a matching logic model M containing (concrete) program configurations of a particular

[3] In the syntax of the $\mathbb{K}$ framework this formula would look more like `<k>x--<k><st>x |-> X</st>` $\wedge$ `(X >Int 1 = true)`.

programming language. One such configuration might be:

$$\gamma_{example} \equiv \quad \ll \texttt{x--;} \mid \texttt{x} \mapsto 3 \gg$$

Then, we have that $(M, \gamma_{example}, \rho) \models \varphi_{example}$ for any valuation ρ satisfying $\rho(X) = 3$, and we say that $\varphi_{example}$ *matches* $\gamma_{example}$ in ρ.

A pattern φ is *valid in* M, written $M \models \varphi$, iff $(M, \gamma, \rho) \models \varphi$ for every γ and ρ. We observe that the validity of a structureless pattern (a pattern without terms-as-predicates) does not depend on the selected model element. Also, the validity of any pattern does not depend on those variables the pattern does not mention. A more formal treatment of matching logic is to be found in Appendix A.

2.3 One-path Reachability Logic

Reachability logic [26, 32] (RL) is a formalism for both a) defining formal semantics of programming languages, and b) specifying and reasoning about partial correctness properties of programs in those languages. On the formal semantics side, a programming language is modeled as a *reachability system* $\mathcal{S} = (\mathcal{T}, S)$, where $\mathcal{T}$ is a matching logic model (that is, a Σ-algebra) and S is a set of *reachability rules* of the shape $\varphi \Rightarrow^{\exists} \varphi'$, where φ and φ' are *matching logic* patterns over Σ describing sets of *source* and *target* program configurations.

Each reachability system naturally induces a *transition system* $(\mathcal{T}_{Cfg}, \Rightarrow_{\mathcal{S}})$, whose states are program configurations and transitions $\Rightarrow_{\mathcal{S}}$ are defined as follows: for $\gamma, \gamma' \in \mathcal{T}_{Cfg}$ we have $\gamma \Rightarrow_{\mathcal{S}} \gamma'$ iff there is some rule $\varphi \Rightarrow^{\exists} \varphi' \in S$ and some valuation $\rho : Var \to \mathcal{T}$ such that $(\mathcal{T}, \gamma, \rho) \models \varphi$ and $(\mathcal{T}, \gamma', \rho) \models \varphi'$.

As an example, consider the following reachability rule

$$\ll \texttt{if (}\mathit{true}\texttt{) then } P_1 \texttt{ else } P_2 \mid S \gg \Rightarrow^{\exists} \ll P_1 \mid S \gg \tag{4}$$

saying that the `if` construct of the particular language takes the first branch (P_1) whenever the condition is *true*. (Typically, there would be additional rules governing the evaluation of the condition.) This rule induces (among others) the transition

$$\ll \texttt{if (}\mathit{true}\texttt{) then x++ else x--} \mid \texttt{x} \mapsto 3 \gg \Rightarrow_{\mathcal{S}} \ll \texttt{x++} \mid \texttt{x} \mapsto 3 \gg . \tag{5}$$

On the *partial correctness* side, RL reuses the concept of reachability rules. For example, one can specify that the program `while(x > 0) do x--;` may, if it terminates at all, reach a configuration where nothing remains to be executed (represented by "$\cdot$") and where the program variable `x` has a non-positive value, by means of the following reachability rule

$$\ll \texttt{while (x>0) do x--;} \mid \texttt{x} \mapsto V \gg \Rightarrow^{\exists} \exists V'.\ \ll \cdot \mid \texttt{x} \mapsto V' \gg \wedge (V' \leq_{\text{Int}} 0 = \mathit{true})$$

Assuming the language is deterministic, this is equivalent to saying that if the program terminates, the resulting configuration will have a non-positive value of `x`. Formally, we say that a configuration $\gamma \in \mathcal{T}_{Cfg}$ terminates in $(\mathcal{T}_{Cfg}, \Rightarrow_{\mathcal{S}})$ iff there is no infinite chain $\gamma \Rightarrow_{\mathcal{S}} \gamma_1 \Rightarrow_{\mathcal{S}} \gamma_2 \Rightarrow_{\mathcal{S}} \ldots$. A rule of the shape $\varphi \Rightarrow^{\exists} \varphi'$ is *satisfied* in a reachability system $\mathcal{S} = (\mathcal{T}, S)$, written $\mathcal{S} \vDash_{\mathsf{RL}} \varphi \Rightarrow^{\exists} \varphi'$, iff for every $\gamma \in \mathcal{T}_{Cfg}$ such that γ terminates in $(\mathcal{T}_{Cfg}, \Rightarrow_{\mathcal{S}})$ and for any valuation $\rho : Var \to \mathcal{T}$ such that $(\mathcal{T}, \gamma, \rho) \models \varphi$, there exists some $\gamma' \in \mathcal{T}_{Cfg}$ such that $\gamma \Rightarrow^{*}_{\mathcal{S}} \gamma'$ and $(\mathcal{T}, \gamma', \rho) \models \varphi'$.

Reachability logic is equipped with a proof system that derives sequents of the shape $\mathcal{A}, C \vdash_{\mathsf{RL}} \varphi \Rightarrow^{\exists} \varphi'$ (where $\mathcal{A}$ is a reachability system and C is introduced below). The proof

system is sound and complete: an RL claim is satisfied in $\mathcal{S}$ iff $\mathcal{S}, \emptyset \vdash_{\mathsf{RL}} \varphi \Rightarrow^{\exists} \varphi'$. The set C, initially empty, contains so-called *circularities*, which are claims postulated to hold but not justified yet. Circularities, which correspond to the notion of *loop invariants* of Hoare logic, enable one to reason about repetitive behavior of programs. The proof system contains a rule

$$\text{Circularity}\ \frac{\mathcal{A}, C \cup \{\varphi \Rightarrow^{\exists} \varphi'\} \vdash_{\mathsf{RL}} \varphi \Rightarrow^{\exists} \varphi'}{\mathcal{A}, C \vdash_{\mathsf{RL}} \varphi \Rightarrow^{\exists} \varphi'}$$

which adds the current claim to the set of circularities. When progress is made (by means of other rules, essentially performing symbolic execution), the claim is moved from the set of circularities to $\mathcal{A}$ (using the *Transitivity* rule – see Appendix B.2) and can be reused, similarly to the way one assumes a loop invariant in order to prove it again. We refer the interested reader to [26] for more details.

Remark 1. *Following the original reachability logic literature ([26, 32]), we restrict the class of reachability systems we work with to those whose reachability rules have the shape*

$$\phi \wedge P \Rightarrow^{\exists} \phi' \wedge P'$$

where ϕ, ϕ' are terms-as-predicates, and P, P' contain no terms-as-predicate. As argued in these papers, such rules can support various styles of operational semantics, including evaluation contexts [16], the chemical abstract machine [3], and $\mathbb{K}$ [7]. We thus support all reachability systems supported by reachability logic.

3 Cartesian Reachability Logic

This section introduces *Cartesian Reachability logic (CRL)* - a language-parametric logic for reasoning about k-safety hyperproperties. Our aim with CRL is to make reasoning in the style of *Cartesian Hoare logic (CHL)* [28] available for any deterministic language for which a reachability-logic semantics S is available. For that purpose, we define the language of CRL and its semantics, and demonstrate the logic's expressiveness on a couple of examples. Finally, we give CRL a sound proof system, which is the main contribution of this paper.

3.1 Syntax and Semantics

Cartesian reachability logic is an extension of (one-path) reachability logic. To express k-safety properties we extend reachability rules $\varphi \Rightarrow^{\exists} \varphi'$ to *cartesian reachability claims* of the form

$$[\varphi_1, \ldots, \varphi_k] \wedge P \Rightarrow^{c\exists} \exists \vec{Y}. [\varphi'_1, \ldots, \varphi'_k] \wedge P'$$

The intuitive meaning of such a claim is as follows: there are k programs embedded into k source configurations, with i-th source configuration matching φ_i, and k target configurations matching φ'_is. Additionally, the FOL formula P can relate the source configurations, and the FOL formula P' the target configurations. We call formulas of the form $\exists \vec{Y}. [\varphi_1, \ldots, \varphi_k] \wedge P$ (where $\vec{Y}$ may be an empty vector) *existentially-quantified constrained list patterns* (ECLP).

For example, let us again consider the program

$$P \equiv \texttt{while(x > 0)\{ x--; y++; \}}.$$

Additionally, let $\xi(Q, X, Y)$ be a pattern matching those configurations of program Q where the program variable $\mathtt{x}$ has value X and the program variable $\mathtt{y}$ has value Y:

$$\xi(Q, X, Y) \equiv \ll Q \mid \mathtt{x} \mapsto X, \mathtt{y} \mapsto Y \gg \ . \tag{6}$$

Then, the claim Ω_{mono}, defined as

$$\begin{aligned}&[\xi(P, X_1, Y_1), \xi(P, X_2, Y_2)] \wedge X_1 \leq X_2 \wedge Y_1 \leq Y_2 \\ \Rightarrow^{c\exists}\ &\exists X_1', Y_1', X_2', Y_2'. [\xi(\epsilon, X_1', Y_1'), \xi(\epsilon, X_2', Y_2')] \wedge Y_1' \leq Y_2'\end{aligned}$$

(where ϵ denotes the empty program) expresses the property that the program P is *monotone*. That is, when we start an execution (using the semantics of the particular language) from some configuration γ_1 matching $\xi(P, X_1, Y_1)$ and a second execution from some configuration γ_2 matching $\xi(P, X_2, Y_2)$, if $X_1 \leq X_2$ and $Y_1 \leq Y_2$, we end up in configurations γ_1', γ_2' matching $\xi(\epsilon, X_1', Y_1')$ and $\xi(\epsilon, X_2', Y_2')$ for some X_1', Y_1', X_2', Y_2' satisfying $Y_1' \leq Y_2'$. Note that we need the existential quantification on the right-hand side of claims to be able to talk about the new values of program variables (whereas in CHL, we have original values in the precondition and new values in the postcondition).

We formally define the semantics of a CRL claim as follows:

Definition 1 (CRL semantics)**.** *A claim* $[\varphi_1, \ldots, \varphi_k] \wedge P \Rightarrow^{c\exists} \exists \vec{Y}. [\varphi_1', \ldots, \varphi_k'] \wedge P'$ *is* valid *in a reachability system* $\mathcal{S} = (\mathcal{T}, S)$*, written*

$$(\mathcal{T}, S) \vDash_{\mathsf{CRL}} [\varphi_1, \ldots, \varphi_k] \wedge P \Rightarrow^{c\exists} \exists \vec{Y}. [\varphi_1', \ldots, \varphi_k'] \wedge P' \,,$$

iff for all configurations $\gamma_1, \ldots, \gamma_k \in \mathcal{T}_{Cfg}$ *which terminate in* $(\mathcal{T}_{Cfg}, \Rightarrow_{\mathcal{S}})$ *and any* $\mathcal{T}$*-valuation* ρ*, whenever* $(\mathcal{T}, \gamma_i, \rho) \models \varphi_i \wedge P$ *for all* $i \in \{1, \ldots, k\}$*, then there exist configurations* $\gamma_1', \ldots, \gamma_k' \in \mathcal{T}_{Cfg}$ *such that* $\gamma_i \Rightarrow_{\mathcal{S}}^* \gamma_i'$ *for all* $i \in \{1, \ldots, k\}$*, and there also exists an* $\mathcal{T}$*-valuation* ρ' *satisfying* $\rho(v) = \rho'(v)$ *for any* $v \in Var \setminus \vec{Y}$*, and* $(\mathcal{T}, \gamma_i', \rho') \models \varphi_i' \wedge P'$ *for all* $i \in \{1, \ldots, k\}$*.*

As can be seen from the example above, CRL is more verbose than CHL. This is partly because of the need to specify patterns matching the whole program configurations and partly because of the need to existentially quantify those variables on the right side whose value is not determined by the left side. To alleviate this problem, we introduce the following notation, which we inherit from RL:

Notation Variables whose names start with a question mark are implicitly considered existentially quantified on the right side. Also, an underscore is used to denote anonymous variables, whose values we are not interested in. For example, we can write the claim Ω_{mono} as

$$[\xi(P, X_1, Y_1), \xi(P, X_2, Y_2)] \wedge X_1 \leq X_2 \wedge Y_1 \leq Y_2 \Rightarrow^{c\exists} [\xi(\epsilon, ?\_, ?Y_1), \xi(\epsilon, ?\_, ?Y_2)] \wedge ?Y_1 \leq ?Y_2 \,.$$

3.2 CRL as an extension of Reachability Logic

We want to point out that one cannot handle k-safety properties directly in RL, by simply replacing a CRL claim with k components by k RL claims. The reason is that in CRL formulas, one can relate variables from different components.

In CRL, one can localize the "global" constraints; for example, the claim Ω_{mono} is equivalent to

$$[\xi(P, X_1, Y_1), \xi(P, X_2, Y_2) \wedge X_1 \leq X_2 \wedge Y_1 \leq Y_2] \Rightarrow^{c\exists} [\xi(\epsilon, ?\_, ?Y_1), \xi(\epsilon, ?\_, ?Y_2) \wedge ?Y_1 \leq ?Y_2] \,.$$

However, if one were to "split" the CRL claim into two, the resulting claims might express a different property than monotonicity. For example, the two claims

$$\xi(Q, X_1, Y_1) \wedge X_1 \leq X_2 \wedge Y_1 \leq Y_2 \quad \Rightarrow^{\exists} \quad \xi(\epsilon, ?X_1, ?Y_1) \tag{7}$$

$$\xi(Q, X_2, Y_2) \wedge X_1 \leq X_2 \wedge Y_1 \leq Y_2 \quad \Rightarrow^{\exists} \quad \xi(\epsilon, ?X_2, ?Y_2) \wedge ?Y_1 \leq ?Y_2 \tag{8}$$

hold for any reasonable program Q (meaning that Q either executes fully or diverges), because $?Y_1$ in the second claim is unrelated to $?Y_1$ in the first claim and thus one can set it to the value of $?Y_2$. On the other hand, if in the second claim we renamed $?Y_1$ to Y_1' (without a question mark), the claim would require that $?Y_2$ is greater than or equal to *any* integer (because Y_1' is not present on the left side), which clearly cannot hold.

On the other hand, CRL is an extension of RL in the following natural sense: If we restrict configuration lists in CRL claims to contain a single configuration each, validity of CRL and RL claims neatly coincide (the proof of the following proposition can be found in Appendix B):

Proposition 1. $(\mathcal{T}, S) \vDash_{\mathsf{CRL}} [\varphi] \wedge \top \Rightarrow^{c\exists} [\varphi'] \wedge \top \iff (\mathcal{T}, S) \vDash_{\mathsf{RL}} \varphi \Rightarrow^{\exists} \varphi'$.

3.3 A language-independent alternative to self-composition

Before presenting a proof system for CRL, we must first discuss a concept crucial to proving its soundness. Self-composition [2, 14] is a technique where a program P together with a k-safety hyperproperty is reduced to a sequential composition of P with itself (with renamed variables) together with a safety property. This technique allows one to use tools and techniques for verification of safety properties to perform verification of k-safety hyperproperties. The challenge here is to generalize self-composition to work with any deterministic language, even if we do not know in advance how the language implements sequential composition, if at all.

To address this issue, we present a novel, general technique, which we call *star extension*. The main idea is to transform a CRL claim into a RL claim over an extended reachability system, whose configurations are lists of configurations of the original system. The function *flatten* then converts an ECLP (which contains a meta-level list of matching logic patterns) into a matching logic formula (containing an object-level list) that matches lists inside the model. The transformation is quite straightforward but technical, and we refer an interested reader to the Appendix B.1; here we state the main theorem:

Theorem 1. *There exist a function $\_^*$ on matching logic signatures, a (equally-named) function $\_^*$ from reachability systems over Σ to reachability systems over Σ^*, and a function flatten from ECLPs over Σ to matching logic Σ^*-formulas, such that*

$$\mathcal{S} \vDash_{\mathsf{CRL}} \Psi \Rightarrow^{c\exists} \Psi' \iff \mathcal{S}^* \vDash_{\mathsf{RL}} \mathit{flatten}(\Psi) \Rightarrow^{c\exists} \mathit{flatten}(\Psi')$$

The price paid for self-composition is that the property of the self-composed program is often hard to reason about. Therefore, in [28], the authors do not apply self-composition directly, but only use its soundness to justify their technique - namely, the soundness of their proof system, which avoids the explicit construction of self-composed programs. We use the star extension for the same purpose.

3.4 Proof System for CRL

We are now ready to give Cartesian Reachability Logic a proof system to facilitate mechanical reasoning. While the intuition behind the semantics of CRL is similar to that of CHL, with only

a few minor differences, coming up with a proof system for CRL is not straightforward. One could attempt to reuse the proof system of CHL and modify it *somehow* to be independent of the particular programming language. Since many CHL rules (e.g., its `If` rule shown in Equation (2)) are simply Hoare logic rules acting on a particular component of a tuple of formulas (that is, symbolic states), one could, for example, lift rules of reachability logic (RL) to the tuple-context and be done. That would indeed work, and the resulting proof system might even be complete.

However, the distinguishing feature of Cartesian Hoare Logic is *not* its completeness but its ability to simplify reasoning by performing lockstep execution of loops, because that can "greatly simplify the verification task (e.g., by requiring simpler invariants)" ([28]). To achieve that, it is not enough to just lift RL rules to tuple-context. And it is not entirely obvious how to support lockstep reasoning involving construct with repetitive behavior: in order to support `while`-loops with `break`, Cartesian Hoare logic itself uses five additional fairly complex rules (besides the lifted Hoare logic rules) that need to reflect the precise semantics of this construct.

We provide a single proof system consisting of only eight rules (Figure 1); the proof system is agnostic about the kinds of constructs with repetetive behavior that the supplied language supports, but enables lockstep reasoning about such constructs. The proof system derives claims of the shape

$$(\mathcal{T}, S) \vdash_{\mathsf{CRL}} \Phi \Downarrow_{C,E} \Psi$$

where Φ, Ψ are of the shape $\exists \vec{X}. (\varphi_1, \ldots, \varphi_k) \wedge P$. One can think about Φ as representing a *premise*, while Ψ, which propagates through the proof rules unchanged, as representing a *conclusion*. For each i, φ_i is a matching logic pattern representing a particular component, and P is a FOL formula (*global constraint*) relating variables from different components. The sets C and E contain *synchronization points* and *enabled synchronization points*, respectively. Together, they implement the concept of an *invariant* relating different components. In particular, set C represents the invariants that were postulated *right now*, while set E represents those postulated in the past and ready to be used. Initially, the proof search starts with $E = C = \emptyset$.

Similarly to *circularities* of reachability logic, *synchronization points* allow us to reason coinductively[4] about (cartesian) reachability claims: one can introduce them at any point of the proof, but they can be used only after progress has been made by means of symbolic execution. The progress requirements resemble the concept of *productivity* of co-recursive definitions.

The rules of our system are fairly simple, and general in the sense that none of them has the semantics of any particular language construct hard-wired into it. We now explain the proof rules of CRL (shown in Figure 1) one by one.

- The Circularity rule is a key rule which allows lockstep reasoning about arbitrary program constructs. It allows the user to postulate the validity of the current claim by means of adding the *current premise* into the set of *synchronization points*, from which a k-tuple of program configurations satisfying the postcondition is claimed to be reachable. Once progress is made (by means of the Step rule), the added synchronization points are *enabled* and can be used to finish the proof using the Axiom rule.

- The Step rule performs symbolic execution on a selected component i (represented by φ_i) using the semantic reachability rule $\varphi \Rightarrow^{\exists} \varphi' \in S$. For this rule to apply, its left side (φ) has to match all the program configurations matching φ_i. Therefore, the rule decomposes φ_i into φ and an additional constraint P', which can be thought of as a part of *path condition* that is *local* to the component i. This local path condition P' is then

[4] A formal treatement of the relationship between coinduction and reachability logic is to be found in [23].

$$\text{Circularity}\,\frac{(\mathcal{T}, S) \vdash_{\mathsf{CRL}} \Psi \Downarrow_{C \cup \{\Psi\}, E} \Psi'}{(\mathcal{T}, S) \vdash_{\mathsf{CRL}} \Psi \Downarrow_{C,E} \Psi'}$$

$$\text{Step}\,\frac{\begin{array}{c}\varphi \Rightarrow^{\exists} \varphi' \in S \\ \mathcal{T} \models \varphi_i \leftrightarrow \varphi \wedge P' \\ P' \text{ is a FOL formula} \\ (\mathcal{T}, S) \vdash_{\mathsf{CRL}} [\varphi_1, \ldots, \varphi_{i-1}, \varphi' \wedge P', \varphi_{i+1}, \ldots, \varphi_k] \wedge P \Downarrow_{\emptyset, (C \cup E)} \Psi'\end{array}}{(\mathcal{T}, S) \vdash_{\mathsf{CRL}} [\varphi_1, \ldots, \varphi_{i-1}, \varphi_i, \varphi_{i+1}, \ldots, \varphi_k] \wedge P \Downarrow_{C,E} \Psi'}$$

$$\text{Axiom}\,\frac{\Psi \in E}{(\mathcal{T}, S) \vdash_{\mathsf{CRL}} \Psi \Downarrow_{C,E} \Psi'}$$

$$\text{Reflexivity}\,\frac{}{(\mathcal{T}, S) \vdash_{\mathsf{CRL}} \Psi \Downarrow_{\emptyset,E} \Psi}$$

$$\text{Case}\,\frac{\begin{array}{c}(\mathcal{T}, S) \vdash_{\mathsf{CRL}} [\varphi_1, \ldots, \varphi_{i-1}, \varphi_i, \varphi_{i+1}, \ldots, \varphi_k] \wedge P' \Downarrow_{C,E} \Psi' \\ (\mathcal{T}, S) \vdash_{\mathsf{CRL}} [\varphi_1, \ldots, \varphi_{i-1}, \psi_i, \varphi_{i+1}, \ldots, \varphi_k] \wedge P' \Downarrow_{C,E} \Psi'\end{array}}{(\mathcal{T}, S) \vdash_{\mathsf{CRL}} [\varphi_1, \ldots, \varphi_{i-1}, (\varphi_i \vee \psi_i), \varphi_{i+1}, \ldots, \varphi_k] \wedge P' \Downarrow_{C,E} \Psi'}$$

$$\text{Conseq}\,\frac{(\mathcal{T}, S) \vdash_{\mathsf{CRL}} \Phi' \Downarrow_{C,E} \Psi' \quad \mathcal{T}^* \models \mathit{flatten}(\Phi) \rightarrow \mathit{flatten}(\Phi')}{(\mathcal{T}, S) \vdash_{\mathsf{CRL}} \Phi \Downarrow_{C,E} \Psi'}$$

$$\text{Abstract}\,\frac{X \notin FV(\Psi') \quad (\mathcal{T}, S) \vdash_{\mathsf{CRL}} \exists \vec{Y}. [\varphi_1, \ldots, \varphi_k] \wedge P \Downarrow_{C,E} \Psi'}{(\mathcal{T}, S) \vdash_{\mathsf{CRL}} \exists X, \vec{Y}. [\varphi_1, \ldots, \varphi_k] \wedge P \Downarrow_{C,E} \Psi'}$$

$$\text{Reduce}\,\frac{(\mathcal{T}^*, S^* \cup \mathit{flatten}^{\exists}(E, \Psi')), \emptyset \vdash_{\mathsf{RL}} \mathit{flatten}^{\exists}(\Psi, \Psi')}{(\mathcal{T}, S) \vdash_{\mathsf{CRL}} \Psi \Downarrow_{C,E} \Psi'}$$

Figure 1: A proof system for CRL

used to constrain the right-side φ' of the selected rule. This proof rule also enables the synchronization points from C by adding them to E.

- The Axiom rule uses an enabled synchronization point to finish the proof.
- The Reflexivity rule can be used to finish a proof when the premise corresponds to the conclusion.
- The Case rule implements case analysis on a selected component i.

- The Conseq rule is used to weaken (or generalize) the premise. It can also be used to propagate information between components and the global constraint.
- The Abstract rule can be used to remove existential quantifiers from the premise. Intuitively, this corresponds to a proof step in first-order logic that replaces an existential quantifier on the left side of an implication with a universal quantifier over the implication, assuming that the variable bound by the existential quantifier does not occur free in the right side. In our setting, the typical way of *obtaining* a top-level existential quantifier in the premise is by means of the Conseq rule.
- The Reduce rule is a way to get completeness into our proof system: it reduces the goal to reachability logic reasoning. This rule also provides a way to prove properties that do not benefit from lockstep reasoning.

The soundness of the proof system, as stated by the following theorem, is the main technical result of this paper.

Theorem 2 (Proof system soundness)**.**

$$(\mathcal{T}, S) \vdash_{\mathsf{CRL}} \Psi \Downarrow_{\emptyset,\emptyset} \Psi' \implies (\mathcal{T}, S) \vDash_{\mathsf{CRL}} \Psi \Rightarrow^{c\exists} \Psi'$$

In order to prove this theorem, we will need (beside Theorem 1, which we discuss in the previous section) a technical lemma stating that one can generate an RL proof on a star-extended system from a CRL proof. This lemma is the second major component of the soundness proof of CRL and its proof can be found in Appendix B.2.

Lemma 1.

$$(\mathcal{T}, S) \vdash_{\mathsf{CRL}} \Psi \Downarrow_{C,E} \Psi' \implies$$
$$(\mathcal{T}^*, S^* \cup \mathit{flatten}^{\exists}(E, \Psi')), \mathit{flatten}^{\exists}(C, \Psi') \vdash_{\mathsf{RL}} \mathit{flatten}^{\exists}(\Psi, \Psi')$$

With all the technical tools is place, the proof itself is a straightforward affair:

Proof of Theorem 2. Assume $(\mathcal{T}, S) \vdash_{\mathsf{CRL}} \Psi \Downarrow_{\emptyset,\emptyset} \Psi'$. By Lemma 1, we have $(\mathcal{T}, S) \vdash_{\mathsf{RL}} \mathit{flatten}^{\exists}(\Psi, \Psi')$. By soundness of reachability logic, we have $(\mathcal{T}, S) \vDash_{\mathsf{RL}} \mathit{flatten}^{\exists}(\Psi, \Psi')$. By Theorem 1, we have $(\mathcal{T}, S) \vDash_{\mathsf{CRL}} \Psi \Rightarrow^{c\exists} \Psi'$ and we are done. □

The proof system is also *relatively complete* (with respect to an oracle deciding validity in the underlying matching logic model $\mathcal{T}$).

Theorem 3 (Relative completeness)**.**

$$(\mathcal{T}, S) \vDash_{\mathsf{CRL}} \Psi \Rightarrow^{c\exists} \Psi' \implies (\mathcal{T}, S) \vdash_{\mathsf{CRL}} \Psi \Downarrow_{\emptyset,\emptyset} \Psi'$$

Proof of Theorem 3. Assume $(\mathcal{T}, S) \vDash_{\mathsf{CRL}} \Psi \Rightarrow^{c\exists} \Psi'$. By Theorem 1, we obtain

$$(\mathcal{T}^*, S^*) \vDash_{\mathsf{RL}} \mathit{flatten}^{\exists}(\Psi, \Psi')\,.$$

By relative completeness of reachability logic, we obtain

$$(\mathcal{T}^*, S^*) \vdash_{\mathsf{RL}} \mathit{flatten}^{\exists}(\Psi, \Psi')\,,$$

and we conclude the proof using the Inherit rule. Note that to apply relative completeness of RL, we need to have an oracle for deciding validity in the extended model. A construction of such oracle from the oracle for deciding validity in $\mathcal{T}$ is in Appendix B.3. The main idea is to reduce reasoning about list of configurations to reasoning about single natural numbers, using Gödel's β function for representation if sequences of natural numbers. □

Our completeness result is similar to the completeness result of CHL in the sense that the completeness does not involve features used for lockstep reasoning. It would be interesting to investigate whether we can have some complete proof system without the Reduce rule; we leave this for future work.

One may ask the question, "why give CRL a new proof system if one can perform the same reasoning by means of Theorem 1 and the existing proof system of reachability logic?". The answer is the following. When one reduces a CRL goal into RL using Theorem 1, the function *flatten* appears in the goal. To perform reasoning using the RL proof system, one then has to either simplify the goal by unfolding the definition of *flatten*, or use (and prove) some helper lemmas about the effect of applying RL proof rules on RL goals containing *flatten*. In the first case one lowers the abstraction level and have to reason about matching logic formulas containing translations of other matching logic formulas into FOL, which then makes RL reasoning difficult. In the second case, however, one may end up proving lemmas which, when combined, result in a proof system for CRL. Indeed, the soundness of our proof system is established by a (meta-)proof which constructs a RL proof from a CRL one (Lemma 1).

4 An example proof involving lockstep reasoning

We now present an example proof sketch using our proof system; the proof involves lockstep reasoning. To ease the notation, we write simply t_b instead of $t_b = \mathit{true}$ for boolean-sorted side conditions t_b. Consider again the claim Ω_{mono} from Section 3.1:

$$[\xi(P, X_1, Y_1), \xi(P, X_2, Y_2)] \wedge X_1 \leq X_2 \wedge Y_1 \leq Y_2 \Rightarrow^{c\exists} [\xi(\epsilon, ?\_, ?Y_1), \xi(\epsilon, ?\_, ?Y_2)] \wedge ?Y_1 \leq ?Y_2 \,.$$

Let Ψ'_{mono} denote the right side of the $\Rightarrow^{c\exists}$ above. We want to prove

$$\mathcal{S}_{imp} \vdash_{\mathsf{CRL}} [\xi(P, X_1, Y_1), \xi(P, X_2, Y_2)] \wedge X_1 \leq_{Int} X_2 \wedge Y_1 \leq_{Int} Y_2 \Downarrow_{\emptyset,\emptyset} \Psi'_{mono} \,.$$

To do so, we first want to add the current premise as a synchronization point. However, we need to make the synchronization point more general than the current claim, as we will see later in the proof. Therefore, we first apply the Conseq rule to change the goal to

$$\mathcal{S}_{imp} \vdash_{\mathsf{CRL}} \exists X_1, Y_1, X_2, Y_2. [\xi(P, X_1, Y_1), \xi(P, X_2, Y_2)] \wedge X_1 \leq_{Int} X_2 \wedge Y_1 \leq_{Int} Y_2 \Downarrow_{\emptyset,\emptyset} \Psi'_{mono} \,,$$

then apply the Circularity rule, followed by application of the Abstract rule, which basically changes the goal back, except that now we have a general synchronization point. The goal is now

$$\mathcal{S}_{imp} \vdash_{\mathsf{CRL}} [\xi(P, X_1, Y_1), \xi(P, X_2, Y_2)] \wedge X_1 \leq X_2 \wedge Y_1 \leq Y_2 \Downarrow_{\Phi_{sync},\emptyset} \Psi'_{mono} \,,$$

where

$$\Phi_{sync} \equiv \mathcal{S}_{imp} \vdash_{\mathsf{CRL}} \exists X_1, Y_1, X_2, Y_2. [\xi(P, X_1, Y_1), \xi(P, X_2, Y_2)] \wedge X_1 \leq_{Int} X_2 \wedge Y_1 \leq_{Int} Y_2 \,.$$

Now we perform symbolic execution on both components, using repeated applications of the Step rule, enabling the synchronization points. The exact details depend on the exact rules of $\mathcal{S}_{imp}$, but assuming that the semantics of the `while` statement is defined by unrolling into the `if` statement, we end up with a goal like

$$\mathcal{S}_{imp} \vdash_{\mathsf{CRL}} [\xi(\texttt{if}\ (X_1 >_{Int} 0)\{\texttt{y++;x--;P}\}, X_1, Y_1), \xi(\texttt{if}\ (X_2 >_{Int} 0)\{\texttt{y++;x--;P}\}, X_2, Y_2)] \\ \wedge X_1 \leq_{Int} X_2 \wedge Y_1 \leq_{Int} Y_2 \Downarrow_{\emptyset,\Phi_{sync}} \Psi'_{mono} \,,$$

Now we want to perform case analysis. To do so, we first use the Conseq rule to obtain disjunctions of patterns at the respective components; then we repeatedly apply the Case rule, leading to four goals; finally, we use the Conseq rule to propagate the constraints from the components to the global constraint. Then, the four goals will be as follows (the differences are shown in blue):

$$\mathcal{S}_{imp} \vdash_{\mathsf{CRL}} [\xi(\texttt{if (}\mathit{true}\texttt{)\{y++;x--;P\}}, X_1, Y_1), \xi(\texttt{if (}\mathit{true}\texttt{)\{y++;x--;P\}}, X_2, Y_2)] \\ \wedge X_1 \leq_{Int} X_2 \wedge Y_1 \leq_{Int} Y_2 \wedge X_1 >_{Int} 0 \wedge X_2 >_{Int} 0 \Downarrow_{\emptyset,\Phi_{sync}} \Psi'_{mono}\,, \tag{9}$$

$$\mathcal{S}_{imp} \vdash_{\mathsf{CRL}} [\xi(\texttt{if (}\mathit{true}\texttt{)\{y++;x--;P\}}, X_1, Y_1), \xi(\texttt{if (}\mathit{false}\texttt{)\{y++;x--;P\}}, X_2, Y_2)] \\ \wedge X_1 \leq_{Int} X_2 \wedge Y_1 \leq_{Int} Y_2 \wedge X_1 >_{Int} 0 \wedge X_2 \leq_{Int} 0 \Downarrow_{\emptyset,\Phi_{sync}} \Psi'_{mono}\,, \tag{10}$$

$$\mathcal{S}_{imp} \vdash_{\mathsf{CRL}} [\xi(\texttt{if (}\mathit{false}\texttt{)\{y++;x--;P\}}, X_1, Y_1), \xi(\texttt{if (}\mathit{true}\texttt{)\{y++;x--;P\}}, X_2, Y_2)] \\ \wedge X_1 \leq_{Int} X_2 \wedge Y_1 \leq_{Int} Y_2 \wedge X_1 \leq_{Int} 0 \wedge X_2 >_{Int} 0 \Downarrow_{\emptyset,\Phi_{sync}} \Psi'_{mono}\,, \tag{11}$$

$$\mathcal{S}_{imp} \vdash_{\mathsf{CRL}} [\xi(\texttt{if (}\mathit{false}\texttt{)\{y++;x--;P\}}, X_1, Y_1), \xi(\texttt{if (}\mathit{false}\texttt{)\{y++;x--;P\}}, X_2, Y_2)] \\ \wedge X_1 \leq_{Int} X_2 \wedge Y_1 \leq_{Int} Y_2 \wedge X_1 \leq_{Int} 0 \wedge X_2 \leq_{Int} 0 \Downarrow_{\emptyset,\Phi_{sync}} \Psi'_{mono}\,. \tag{12}$$

- The case in Equation (12) represents the situation when both loops have finished their execution. We can solve this case by symbolically executing both programs (using the Step rule) to the end. It is easy to see that then the premise implies the conclusion; therefore, we finish this case by generalizing the premise (using the Conseq rule) to be exactly the conclusion, and then applying the Reflexivity rule.

- The case in Equation (10) represents the situation when the first loop continues execution and the second is finished. This can never happen - we see that the side condition is contradictory. We finish this case using the Reduce rule.

- The case in Equation (11) represents the complementary situation - the first loop has finished its execution, while the second loop continues. This requires inventing an invariant capturing the idea that the execution of the second loop can only increase the difference between the values of `y`. We can Reduce this subgoal to simple RL reasoning. We could also prove this case without Reduce, using the other rules, but lockstep reasoning does not help there.

- The case in Equation (9) is the one when we utilize the ability to perform lockstep reasoning. We symbolically execute both components until their program parts become the `while` loops again; that is, P. The goal is now

  $$\mathcal{S}_{imp} \vdash_{\mathsf{CRL}} [\xi(P, X_1 -_{Int} 1, Y_1 +_{Int} 1), \xi(P, X_2 -_{Int} 1, Y_2 +_{Int} 1)] \\ \wedge X_1 \leq_{Int} X_2 \wedge Y_1 \leq_{Int} Y_2 \wedge X_1 \leq_{Int} 0 \wedge X_2 \leq_{Int} 0 \Downarrow_{\emptyset,\Phi_{sync}} \Psi'_{mono}\,.$$

  which implies the synchronization point Φ_{sync} which is already enabled. We can therefore conclude the proof using Conseq and Axiom.

We observe that the proof is rather low-level. On the other hand, the proof system itself is very simple, and one can prove derived rules that raise the abstraction level, as we show in Appendix B.4.

5 Related Work and Discussion

5.1 Language-parametric verification

The idea of a Circularity rule is already present in the existing literature on reachability logics [26, 27, 32], where it was used to generalize the notion of *loop invariant*. However, RL can not be used to reason about k-safety hyperproperties and does not perform lockstep execution. The literature on language-independent equivalence checking also contains the idea of a Circularity rule. In [9], circularities are used to synchronize executions of different programs (possibly in different languages); however, the proof system of [9] can be used only to reason about full program equivalence and not about k-safety hyperproperties. What makes the proof system of [9] really unusable in our situation is the fact that its Circularity rule

$$\frac{\mathcal{S} \vDash \varphi_1 \Rightarrow^{+} \varphi_1' \quad \mathcal{S} \vDash \varphi_2 \Rightarrow^{+} \varphi_2' \quad \mathcal{S} \vdash_{equiv} \langle \varphi_1', \varphi_2' \rangle \Downarrow^{\infty} E}{\mathcal{S} \vdash_{equiv} \langle \varphi_1, \varphi_2 \rangle \Downarrow^{\infty} E}$$

requires all the components to make progress (denoted by the "plus" sign on top of the arrows) before the circularity can be used, which would prohibit RL-style reasoning when one component has already finished execution.

We believe that all the mentioned proof systems that use a variant of the Circularity rule are instances of a more general framework, known as *Circular coinduction* in the literature [25].

5.2 Relation to Cartesian Hoare logic

5.2.1 (Non)determinism

We base our work on the *one-path* variant of reachability logic. Consequently, CRL inherits a known limitation of one-path reachability logic: that the tight correspondence between one-path RL and Hoare logics is limited to deterministic languages. This is because in the CRL semantics we existentially quantify over reachable configurations, while in CHL, target states are quantified universally. Despite that, we can prove the following theorem (the proof is technical and can be found in Appendix B.5).

Theorem 4. *CRL extends CHL on the deterministic fragment of the CHL-supported language. That is, assume a sound reachability-logic formalization (that is, a reachability system) $\mathcal{S}_{IMP}$ of the CHL's imperative language. Then there exist translation functions tr and end such that given any statement P in the deterministic fragment of the CHL's imperative language and any first order formulas φ, ψ,*

$$\vDash_{CHL} ||\varphi||\ P\ ||\psi|| \iff \mathcal{S}_{IMP} \vDash_{\mathsf{CRL}} tr(P, \varphi) \Rightarrow^{c\exists} end(P, \psi)\,.$$

5.2.2 Similarities

Our understanding of the inner workings of CHL is based on the extended, unpublished version ([29]) of [28]. There, the authors define a *linearization operation* on lists of programs, which roughly corresponds to our *star extension* of the language's semantics. Then, the authors prove lemmas saying that a CHL triple with a list of programs inside is derivable in the CHL proof system if and only if a Hoare triple having the same list of programs but linearized inside, is derivable; the "only if" implication corresponds to our Lemma 1, where we construct an RL proof from a CRL proof, while the "if" implication corresponds to our Reduce rule.

Furthermore, in the proof of soundness of CHL, the authors assume soundness of the self-composition technique; self-composition corresponds to our *star extension* and its soundness to our Theorem 1. Finally, [29] assumes soundness and relative completeness of the underling Hoare logic; similarly, we assume soundness and relative completeness of reachability logic. However, for completeness, we had to prove that the *star extension* preserves decidability of validity.

5.2.3 Differences

There are also differences between the CHL and CRL techniques. First, our proof system has only 8 rules, and they do not mention any programming language construct, while CHL has 17 rules (not counting the Expand rule), half of which are specific to the underlying language.

Second, there is a redundancy between the language-specific CHL rules and the Hoare logic rules of the programming language: for example, the conditional statement ("if") has (1) a rule in the formal semantics of the language, (2) a rule in the Hoare logic (not shown in the paper), and (3) a rule in CHL. When considering that the three rules have to play nice together (that is, CHL and Hoare logic rules have to be sound with respect to the semantics), someone had to think about the conditional statement at least five times. We consider this to be highly uneconomical - and the situation is even worse for the looping construct ("while-with-breaks"), which is supported using additional CHL rules. In contrast, in the CRL/RL framework, it is enough to design each language construct once - when giving its operational semantics.

Third, in CHL the support for lockstep reasoning is hard-wired into the rules for loops, while in our framework, lockstep reasoning is not limited to loops, but can support arbitrary sources of circular behavior - including loops, recursion, gotos.

5.3 Other Related Work

In [15], the authors develop a *logic for hyper-triple composition (LHC)* that allows reasoning about $k-safety$ properties compositionally. Similarly to CHL, LHC targets a particular small imperative language. We believe that compositionality is orthogonal to language-parametricity, and thus we would like to generalize their work to language-parametric settings in future work.

A game-based technique for verifying software hyperproperties beyond k-safety has been developed in [4]. This technique works with *symbolic transition systems*, so it already is language-independent *in some sense*. However, it is not clear how to use the technique with an arbitrary language L, without writing a *compiler* from L to symbolic transition systems first.

6 Future Work and Conclusion

We have presented Cartesian Reachability logic - a logic for reasoning about k-safety hyperproperties in any deterministic language equipped with a RL-based operational semantics. The logic has a simple, sound, and complete proof system and allows lockstep reasoning similar to Cartesian Hoare logic. Instantiating CRL with a new language does not require any changes to the soundness proof; therefore, CRL has the potential to significantly reduce the costs of the development of tools and techniques for k-safety verification.

In the future, we want to develop a variant of CRL that would not require the language to be deterministic. We believe this to be viable because (1) CHL has some support for nondeterminism and because (2) reachability logic, on which we base our work, has a newer variant that supports nondeterminism, too. On the theoretical side, we would like to know whether

our proof system would be complete even in the absence of the Reduce rule. An orthogonal line of future research is compositionality: we would like to enable compositional reasoning using the technique developed in [15]. Finally, we plan to develop a practical, language-parametric tool implementing CRL, using the $\mathbb{K}$ semantic framework, and use it for verification of smart contracts (typically written in deterministic languages); we already have an early prototype[5] capable of using lockstep reasoning for verification of the example from Section 4.

Acknowledgment We would like to thank the anonymous reviewers for their suggestions and comments.

References

[1] Shreya Agrawal and Borzoo Bonakdarpour. Runtime verification of k-safety hyperproperties in hyperltl. In *IEEE 29th Computer Security Foundations Symposium, CSF 2016, Lisbon, Portugal, June 27 - July 1, 2016*, pages 239–252. IEEE Computer Society, 2016.

[2] Gilles Barthe, Pedro R. D'Argenio, and Tamara Rezk. Secure information flow by self-composition. In *17th IEEE Computer Security Foundations Workshop, (CSFW-17 2004), 28-30 June 2004, Pacific Grove, CA, USA*, pages 100–114. IEEE Computer Society, 2004.

[3] Gérard Berry and Gérard Boudol. The chemical abstract machine. *Theor. Comput. Sci.*, 96(1):217–248, 1992.

[4] Raven Beutner and Bernd Finkbeiner. Software verification of hyperproperties beyond k-safety. In Sharon Shoham and Yakir Vizel, editors, *Computer Aided Verification - 34th International Conference, CAV 2022, Haifa, Israel, August 7-10, 2022, Proceedings, Part I*, volume 13371 of *Lecture Notes in Computer Science*, pages 341–362. Springer, 2022.

[5] Xiaohong Chen, Dorel Lucanu, and Grigore Rosu. Matching logic explained. *J. Log. Algebraic Methods Program.*, 120:100638, 2021.

[6] Xiaohong Chen and Grigore Rosu. Matching μ-logic. In *34th Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2019, Vancouver, BC, Canada, June 24-27, 2019*, pages 1–13. IEEE, 2019.

[7] Xiaohong Chen and Grigore Rosu. The K vision for the future of programming language design and analysis. In Ezio Bartocci, Yliès Falcone, and Martin Leucker, editors, *Formal Methods in Outer Space - Essays Dedicated to Klaus Havelund on the Occasion of His 65th Birthday*, volume 13065 of *Lecture Notes in Computer Science*, pages 3–9. Springer, 2021.

[8] Ştefan Ciobâcă, Dorel Lucanu, Vlad Rusu, and Grigore Rosu. A language-independent proof system for mutual program equivalence. In Stephan Merz and Jun Pang, editors, *Formal Methods and Software Engineering - 16th International Conference on Formal Engineering Methods, ICFEM 2014, Luxembourg, Luxembourg, November 3-5, 2014. Proceedings*, volume 8829 of *Lecture Notes in Computer Science*, pages 75–90. Springer, 2014.

[9] Ştefan Ciobâcă, Dorel Lucanu, Vlad Rusu, and Grigore Rosu. A language-independent proof system for full program equivalence. *Formal Aspects Comput.*, 28(3):469–497, 2016.

[10] Michael R. Clarkson, Bernd Finkbeiner, Masoud Koleini, Kristopher K. Micinski, Markus N. Rabe, and César Sánchez. Temporal logics for hyperproperties. In Martín Abadi and Steve Kremer, editors, *Principles of Security and Trust - Third International Conference, POST 2014, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2014, Grenoble, France, April 5-13, 2014, Proceedings*, volume 8414 of *Lecture Notes in Computer Science*, pages 265–284. Springer, 2014.

[11] Michael R. Clarkson, Bernd Finkbeiner, Masoud Koleini, Kristopher K. Micinski, Markus N. Rabe, and César Sánchez. Temporal logics for hyperproperties. *CoRR*, abs/1401.4492, 2014.

[5] available at https://github.com/h0nzZik/crl-tool/

[12] Michael R. Clarkson and Fred B. Schneider. Hyperproperties. In *Proceedings of the 21st IEEE Computer Security Foundations Symposium, CSF 2008, Pittsburgh, Pennsylvania, USA, 23-25 June 2008*, pages 51–65. IEEE Computer Society, 2008.

[13] Norine Coenen, Bernd Finkbeiner, César Sánchez, and Leander Tentrup. Verifying hyperliveness. In Isil Dillig and Serdar Tasiran, editors, *Computer Aided Verification - 31st International Conference, CAV 2019, New York City, NY, USA, July 15-18, 2019, Proceedings, Part I*, volume 11561 of *Lecture Notes in Computer Science*, pages 121–139. Springer, 2019.

[14] Ádám Darvas, Reiner Hähnle, and David Sands. A theorem proving approach to analysis of secure information flow. In Dieter Hutter and Markus Ullmann, editors, *Security in Pervasive Computing, Second International Conference, SPC 2005, Boppard, Germany, April 6-8, 2005, Proceedings*, volume 3450 of *Lecture Notes in Computer Science*, pages 193–209. Springer, 2005.

[15] Emanuele D'Osualdo, Azadeh Farzan, and Derek Dreyer. Proving hypersafety compositionally. *Proc. ACM Program. Lang.*, 6(OOPSLA2):289–314, 2022.

[16] Matthias Felleisen, Robert Bruce Findler, and Matthew Flatt. *Semantics Engineering with PLT Redex*. MIT Press, 2009.

[17] Bernd Finkbeiner, Lennart Haas, and Hazem Torfah. Canonical representations of k-safety hyperproperties. In *32nd IEEE Computer Security Foundations Symposium, CSF 2019, Hoboken, NJ, USA, June 25-28, 2019*, pages 17–31. IEEE, 2019.

[18] Bernd Finkbeiner, Markus N. Rabe, and César Sánchez. Algorithms for model checking hyperltl and hyperctl ^*. In Daniel Kroening and Corina S. Pasareanu, editors, *Computer Aided Verification - 27th International Conference, CAV 2015, San Francisco, CA, USA, July 18-24, 2015, Proceedings, Part I*, volume 9206 of *Lecture Notes in Computer Science*, pages 30–48. Springer, 2015.

[19] Dwight Guth, Chris Hathhorn, Manasvi Saxena, and Grigore Rosu. Rv-match: Practical semantics-based program analysis. In Swarat Chaudhuri and Azadeh Farzan, editors, *Computer Aided Verification - 28th International Conference, CAV 2016, Toronto, ON, Canada, July 17-23, 2016, Proceedings, Part I*, volume 9779 of *Lecture Notes in Computer Science*, pages 447–453. Springer, 2016.

[20] Everett Hildenbrandt, Manasvi Saxena, Nishant Rodrigues, Xiaoran Zhu, Philip Daian, Dwight Guth, Brandon M. Moore, Daejun Park, Yi Zhang, Andrei Stefanescu, and Grigore Rosu. KEVM: A complete formal semantics of the ethereum virtual machine. In *31st IEEE Computer Security Foundations Symposium, CSF 2018, Oxford, United Kingdom, July 9-12, 2018*, pages 204–217. IEEE Computer Society, 2018.

[21] Theodoros Kasampalis, Dwight Guth, Brandon M. Moore, Traian-Florin Serbanuta, Yi Zhang, Daniele Filaretti, Virgil Nicolae Serbanuta, Ralph Johnson, and Grigore Rosu. IELE: A rigorously designed language and tool ecosystem for the blockchain. In Maurice H. ter Beek, Annabelle McIver, and José N. Oliveira, editors, *Formal Methods - The Next 30 Years - Third World Congress, FM 2019, Porto, Portugal, October 7-11, 2019, Proceedings*, volume 11800 of *Lecture Notes in Computer Science*, pages 593–610. Springer, 2019.

[22] E. Mendelson. *Introduction to Mathematical Logic, Fourth Edition.* Discrete Mathematics and Its Applications. Taylor & Francis, 1997.

[23] Brandon M. Moore, Lucas Peña, and Grigore Rosu. Program verification by coinduction. In Amal Ahmed, editor, *Programming Languages and Systems - 27th European Symposium on Programming, ESOP 2018, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2018, Thessaloniki, Greece, April 14-20, 2018, Proceedings*, volume 10801 of *Lecture Notes in Computer Science*, pages 589–618. Springer, 2018.

[24] Daejun Park, Yi Zhang, Manasvi Saxena, Philip Daian, and Grigore Rosu. A formal verification tool for ethereum VM bytecode. In Gary T. Leavens, Alessandro Garcia, and Corina S. Pasareanu, editors, *Proceedings of the 2018 ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering, ESEC/SIGSOFT FSE 2018, Lake*

Buena Vista, FL, USA, November 04-09, 2018, pages 912–915. ACM, 2018.

[25] Grigore Rosu and Dorel Lucanu. Circular coinduction: A proof theoretical foundation. In Alexander Kurz, Marina Lenisa, and Andrzej Tarlecki, editors, *Algebra and Coalgebra in Computer Science, Third International Conference, CALCO 2009, Udine, Italy, September 7-10, 2009. Proceedings*, volume 5728 of *Lecture Notes in Computer Science*, pages 127–144. Springer, 2009.

[26] Grigore Roșu and Andrei Ștefănescu. Checking reachability using matching logic. In Gary T. Leavens and Matthew B. Dwyer, editors, *Proceedings of the 27th Annual ACM SIGPLAN Conference on Object-Oriented Programming, Systems, Languages, and Applications, OOPSLA 2012, part of SPLASH 2012, Tucson, AZ, USA, October 21-25, 2012*, pages 555–574. ACM, 2012.

[27] Grigore Roșu, Andrei Ștefănescu, Ștefan Ciobâcă, and Brandon M. Moore. One-path reachability logic. In *28th Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2013, New Orleans, LA, USA, June 25-28, 2013*, pages 358–367. IEEE Computer Society, 2013.

[28] Marcelo Sousa and Isil Dillig. Cartesian hoare logic for verifying k-safety properties. In Chandra Krintz and Emery D. Berger, editors, *Proceedings of the 37th ACM SIGPLAN Conference on Programming Language Design and Implementation, PLDI 2016, Santa Barbara, CA, USA, June 13-17, 2016*, pages 57–69. ACM, 2016.

[29] Marcelo Sousa and Isil Dillig. Cartesian hoare logic for verifying k-safety properties. 2016.

[30] Andrei Stefanescu, Daejun Park, Shijiao Yuwen, Yilong Li, and Grigore Rosu. Semantics-based program verifiers for all languages. In Eelco Visser and Yannis Smaragdakis, editors, *Proceedings of the 2016 ACM SIGPLAN International Conference on Object-Oriented Programming, Systems, Languages, and Applications, OOPSLA 2016, part of SPLASH 2016, Amsterdam, The Netherlands, October 30 - November 4, 2016*, pages 74–91. ACM, 2016.

[31] Vlad Zamfir, Mihai Calancea, Denisa Diaconescu, Brandon M. Moore, Karl Palmskog, Traian-Florin Serbanuta, and Michael Stay. VLSM: validating labelled state transition and message production systems. *CoRR*, abs/2202.12662, 2022.

[32] Andrei Ștefănescu, Ștefan Ciobâcă, Radu Mereuță, Brandon M. Moore, Traian Florin Șerbănuță, and Grigore Roșu. All-path reachability logic. *Log. Methods Comput. Sci.*, 15(2), 2019.

A Matching and Reachability logic

Definition 2 (Matching logic syntax and semantics)**.** *We define the matching logic syntax and semantics as follows.*

1. *A matching logic* signature $\mathbf{\Sigma} = (\Sigma, Var)$ *is a many-sorted algebraic signature* Σ *together with a sort-wise infinite set of variables* Var*; we let* Var_s *denote the set of variables of sort* s*.*

2. *Let* $T_\Sigma(Var)$ *denote the free* Σ*-algebra of terms with variables in* Var*. Let* $T_{\Sigma,s}(Var)$ *denote the set of* Σ*-terms (with variables in* Var*) of sort* s*. More precisely, the syntax of terms of sort* s *is defined by the following grammar:*

$$t_s ::= x \mid f(t_{s_1}, \ldots, t_{s_n}) \tag{13}$$

   where $x \in Var_s$ *ranges over variables of of sort* s *and* f *over* n*-ary function symbols of arity* $s_1 \times \ldots \times s_n \to s$*.*

3. *Let* $\mathcal{T}$ *be a* Σ*-algebra, and* f *be a symbol from* Σ*. We use the notation* $\mathcal{T}_f$ *to denote the interpretation of* f *in* $\mathcal{T}$*.*

4. *A function* $\rho : Var \to \mathcal{T}$*, where* $\mathcal{T}$ *is a* Σ*-algebra, extends uniquely (in the usual way) to a* Σ*-algebra morphism* $\rho : T_\Sigma(Var) \to \mathcal{T}$*.*

5. *The set of* nullary predicate symbols $P_\epsilon((\Sigma, Var))$ *(or just* P_ϵ *if* (Σ, Var) *is known from the context) is defined to contain exactly terms* $\phi \in T_{\Sigma,s}(Var)$.

6. *A matching logic* (Σ, Var)*-formula (aka* (Σ, Var)-pattern*) of sort* s *is a* $(\Sigma, P_\epsilon) - \mathsf{FOL}_=$ *formula (that is, a* $\mathsf{FOL}_=$ *formula where function symbols are exactly symbols from* Σ*, and where nullary predicate symbols are exactly terms* $\phi \in T_{\Sigma,s}(Var)$*, without any* k*-ary predicate symbols for* $k > 0$*). We let* $\mathsf{Pattern}(\mathbf{\Sigma})$ *(or just* $\mathsf{Pattern}$ *when* $\mathbf{\Sigma}$ *is known from the context) denote the set of all* $\mathbf{\Sigma}$*-patterns. More precisely, the syntax of patterns is defined by the following grammar:*

$$\varphi ::= t \mid t_s = t_s \mid \varphi_1 \wedge \varphi_2 \mid \neg\varphi \mid \exists x : s.\, \varphi \tag{14}$$

7. *Let* $FV(\varphi)$ *denote the set of all free variables of* φ.

8. *A matching logic pattern is* structureless *if it contains no terms* $\phi \in T_{\Sigma,s}(Var)$ *used as predicates.*

9. *A matching logic* Σ-model $\mathcal{T}$ *is a* Σ*-algebra with non-empty carrier sets.*

10. *Given a matching logic* (Σ, Var) *signature and a* Σ-model $\mathcal{T}$*, a* $\mathcal{T}$*-valuation* ρ *is a function from* Var *to* $\mathcal{T}$ *respecting sorts.*

11. *A matching logic* semantics *is given by means of the satisfaction relation* $\models$ *between a matching logic* Σ*-model, a model element, and a valuation, defined inductively as*

    - $(\mathcal{T}, \gamma, \rho) \models t_1 = t_2$ *iff* $\rho(t_1) = \rho(t_2)$*, where* $t_1, t_2 \in T_{\Sigma,s}(Var)$ *(for some sort* s*);*
    - $(\mathcal{T}, \gamma, \rho) \models \phi$ *iff* $\gamma = \rho(\phi)$*, where* $\phi \in T_{\Sigma,s}(Var)$*;*
    - $(\mathcal{T}, \gamma, \rho) \models \varphi_1 \wedge \varphi_2$ *iff* $(\mathcal{T}, \gamma, \rho) \models \varphi_1$ *and* $(\mathcal{T}, \gamma, \rho) \models \varphi_2$*;*
    - $(\mathcal{T}, \gamma, \rho) \models \neg\varphi$ *iff* $(\mathcal{T}, \gamma, \rho) \not\models \varphi$*;*
    - $(\mathcal{T}, \gamma, \rho) \models \exists x : s.\, \varphi$ *iff* $(\mathcal{T}, \gamma, \rho') \models \varphi$ *for some* ρ' *such that* $\rho'(y) = \rho(y)$ *for all* $y \in Var \setminus \{x\}$ *and* $\rho'(x) \in \mathcal{T}_s$*;*

Lemma 2. *Let* $\mathcal{T}$ *be a matching logic* Σ*-model,* $\gamma, \gamma' \in \mathcal{T}$ *model elements, and* ρ *a* $\mathcal{T}$*-valuation. Then for any structureless pattern* P*,*

$$(\mathcal{T}, \gamma, \rho) \models P \iff (\mathcal{T}, \gamma', \rho) \models P\,.$$

Therefore, when P *is structureless, we may sometimes write* $(\mathcal{T}, \rho) \models P$ *to mean that* $(\mathcal{T}, \gamma, \rho) \models P$ *for* some $\gamma \in \mathcal{T}$.

Proof of Lemma 2. Let P be a structureless pattern. We perform induction on P. If $P \equiv t_1 = t_2$, the equivalence holds trivially, since the semantics does not mention γ. If $P \equiv \phi$, we get contradiction with the assumption that P is structureless; therefore, the conclusion holds by *ex falso quodlibet*. Other cases follow from the induction hypotheses. □

Lemma 3. *Let* $\mathcal{T}$ *be a matching logic model and* t *a term. Then for any two* $\mathcal{T}$*-valuations* ρ, ρ' *satisfying* $\rho'(y) = \rho(y)$ *for any* $y \in FV(t)$*, we have* $\rho(t) = \rho'(t)$.

Proof. By induction on t. □

Lemma 4. *Let $\mathcal{T}$ be a matching logic model, γ an element of this model, and φ a pattern. Then for any two $\mathcal{T}$-valuations ρ, ρ' satisfying $\rho'(y) = \rho(y)$ for any $y \in FV(\varphi)$,*

$$(\mathcal{T}, \gamma, \rho) \models \varphi \iff (\mathcal{T}, \gamma, \rho') \models \varphi\,.$$

Proof. By induction on the structure of φ. The base case where φ is a term is proved by Lemma 3. The case of $t_1 = t_2$ also follows from Lemma 3. The conjunction and negation cases follow from the induction hypothesis. For the case of existential quantification, we assume that $\rho'(y) = \rho(y)$ for any $y \in FV(\exists x : s.\, \varphi)$; the induction hypothesis says that for any two valuations ρ_1, ρ_2 satisfying $\rho_1(y) = \rho_2(y)$ for all $y \in FV(\varphi)$, we have

$$(\mathcal{T}, \gamma, \rho_1) \models \varphi \iff (\mathcal{T}, \gamma, \rho_2) \models \varphi$$

for any γ; and have to prove that there exists some ρ_3 such that (1) $\rho_3(y) = \rho(y)$ for all $y \in Var \setminus \{x\}$, (2) $\rho_3(x) \in \mathcal{T}_s$, and (3) $(\mathcal{T}, \gamma, \rho_3) \models \varphi$, if and only if there exists some ρ_4 such that (4) $\rho_4(y) = \rho'(y)$ for all $y \in Var \setminus \{x\}$, (5) $\rho_4(x) \in \mathcal{T}_s$, and (6) $(\mathcal{T}, \gamma, \rho_4) \models \varphi$. We show here only the proof of the "only if" implication, since the "if" one is similar. Assume there exists such ρ_3. Choose

$$\rho_4(y) = \begin{cases} \rho'(y) & \text{if } y \neq x \\ \rho_3(y) & \text{if } y = x \end{cases}$$

from which (4),(5) trivially follows. The point (6) follows from (3) and the induction hypothesis after proving $\rho_3(y) = \rho_4(y)$ for all $y \in FV(\varphi)$: either $y = x$, in which case $\rho_3(y) = \rho_4(y)$ holds by the definition of ρ_4, or $y \neq x$, in which case we reduce the goal using the definition of ρ_4 to $\rho_3(y) = \rho'(y)$. By (1), it is enough to show that $\rho(y) = \rho'(y)$, which follows from the initial assumption. □

Lemma 5 (A semantic property of variable renaming)**.** *For any two variables X, Y of the same sort, and for any two $\mathcal{T}$-valuations ρ_1, ρ_2 which agree on all variables other than X, Y, if $\rho_1(X) = \rho_2(Y)$, then for any matching logic formula φ in which Y does not occur free,*

$$(\mathcal{T}, \gamma, \rho_1) \models \varphi \iff (\mathcal{T}, \gamma, \rho_2) \models \varphi[Y/X]\,.$$

Proof of Lemma 5. We first prove that for any term $t \in T_{\Sigma,s}(Var)$, $\rho_1(t) = \rho_2(t[Y/X])$, by induction in t: if t is a variable, then we perform case analysis on whether the variable is X or not, and we are done; if t is a function application, then we use the induction hypothesis and the fact that function application preserves equality. Now we prove the lemma by induction on the size of φ, generalizing over ρ_1 and ρ_2.

- If $\varphi \equiv t$ for some $t \in T_{\Sigma,s}(Var)$, we apply the above property.
- If $\varphi \equiv t_1 = t_2$, we also apply the above property.
- If φ is a conjunction or negation, the desired property follows directly from the induction hypothesis.
- If $\varphi \equiv \exists Z : s.\, \varphi'$.
  - If $X = Z$, then we have to prove that $(\mathcal{T}, \gamma, \rho') \models \varphi'$ for some ρ' satisfying $\rho'(y) = \rho_1(y)$ for all $y \in Var \setminus \{Z\}$, iff $(\mathcal{T}, \gamma, \rho'') \models \varphi'$ for some ρ'' satisfying $\rho''(y) = \rho_2(y)$ for all $y \in Var \setminus \{Z\}$ - which is easy.

– If $X \neq Z$, then we have to prove that $(\mathcal{T}, \gamma, \rho') \models \varphi'$ for some ρ' satisfying $\rho'(y) = \rho_1(y)$ for all $y \in Var \setminus \{Z\}$, iff $(\mathcal{T}, \gamma, \rho'') \models \varphi'[Y/X]$ for some ρ'' satisfying $\rho''(y) = \rho_2(y)$ for all $y \in Var \setminus \{Z\}$. Since the size of φ' is smaller then the size of φ, and since $\rho'(X) = \rho_1(X) = \rho_2(Y) = \rho''(Y)$, we can use the induction hypothesis and finish the proof using firstorder reasoning.

□

Definition 3 ([32]). *Given a matching logic (Σ, Var)-formula φ of sort Cfg, and a fresh (with respect to φ) variable $\square$ of sort Cfg, we let $\varphi^{\square}$ denote the $\mathsf{FOL}_{=}$ formula formed from φ by replacing nullary predicate symbols $\phi \in T_{\Sigma, Cfg}(Var)$ with equalities $\square = \phi$. Given a matching logic (Σ, Var)-model $\mathcal{T}$, a $\mathcal{T}$-valuation ρ, and an element $\gamma \in \mathcal{T}_{Cfg}$, we let the $\mathcal{T}$-valuation ρ^{γ} be such that $\rho^{\gamma}(\square) = \gamma$, and $\rho^{\gamma}(x) = \rho(x)$ for $x \neq \square$.*

Lemma 6 ([32]). *Whenever $\square$ is fresh in φ, we have*

$$(\mathcal{T}, \gamma, \rho) \models \varphi \iff (\mathcal{T}, \rho^{\gamma}) \models \varphi^{\square}$$

Lemma 7. *$P^{\square} \equiv P$ for any structureless pattern P*

Proof. By induction. □

Lemma 8 (On implication and FOL translation)**.** *Let φ_1, φ_2 be two matching logic formulas such that $\models \varphi_1 \to \varphi_2$. Then $\models (\varphi_1^{\square})[X/\square] \to (\varphi_2^{\square})[X/\square]$.*

Proof. Let M be any matching logic model, γ an element of M, and ρ an M-valuation. We have to prove that $(M, \gamma, \rho) \models (\varphi_1^{\square})[X/\square] \to (\varphi_2^{\square})[X/\square]$, which is (by definition of the squaring function and substitution) equivalent to $(M, \gamma, \rho) \models ((\varphi_1 \to \varphi_2)^{\square})[X/\square]$, which is (by Lemma 5, because $\rho(X) = \rho[\square := \rho(X)](\square)$) equivalent to $(M, \gamma, \rho[\square := \rho(X)]) \models (\varphi_1 \to \varphi_2)^{\square}$, which is (by Lemma 6, because $\rho[\square := \rho(X)] = \rho^{\rho(X)}$) equivalent to $(M, \rho(X), \rho) \models \varphi_1 \to \varphi_2$, which holds by the assumption. □

Lemma 9 (On equivalence and FOL translation)**.** *Let φ_1, φ_2 be two matching logic formulas such that $\models \varphi_1 \leftrightarrow \varphi_2$. Then $\models (\varphi_1^{\square})[X/\square] \leftrightarrow (\varphi_2^{\square})[X/\square]$.*

Proof. Apply Lemma 8 twice. □

Definition 4 ([32, 26]). *We define reachability-logic signatures, rules, and systems as follows.*

1. *A reachability-logic* signature *is a pair $(\mathbf{\Sigma}, Cfg)$, where $\mathbf{\Sigma}$ is a matching logic signature and Cfg is a sort from $\mathbf{\Sigma}$.*
2. *A* one-path reachability rule *over reachability logic signature $(\mathbf{\Sigma}, Cfg)$ is a pair $\varphi \Rightarrow^{\exists} \varphi'$, where φ and φ' are $\mathbf{\Sigma}$-patterns (which can have free variables) of sort Cfg.*
3. *A* reachability system *over a reachability-logic signature $((\Sigma, Var), Cfg)$ is a pair $\mathcal{S} = (\mathcal{T}, S)$, where $\mathcal{T}$ is a Σ-algebra and S is a set of reachability rules over $((\Sigma, Var), Cfg)$.*
4. *A rule $\varphi \Rightarrow^{\exists} \varphi'$ over $((\Sigma, Var), Cfg)$ is* weakly well-defined *with respect to Σ-algebra $\mathcal{T}$ iff for any $\gamma \in \mathcal{T}_{Cfg}$ and $\rho : Var \to \mathcal{T}$ with $(\mathcal{T}, \gamma, \rho) \models \varphi$, there exists $\gamma' \in \mathcal{T}_{Cfg}$ with $(\mathcal{T}, \gamma', \rho) \models \varphi'$.*
5. *A reachability system $\mathcal{S}$ is* weakly well-defined *iff each its rule is weakly well-defined.*

6. *A reachability system $\mathcal{S} = (\mathcal{T}, S)$ over $((\Sigma, Var), Cfg)$ induces a* transition system *$(\mathcal{T}_{Cfg}, \Rightarrow_{\mathcal{S}})$, where $\gamma \Rightarrow_{\mathcal{S}} \gamma'$ for $\gamma, \gamma' \in \mathcal{T}_{Cfg}$ iff there is some rule $\varphi \Rightarrow^{\exists} \varphi' \in S$ and some valuation $\rho : Var \to \mathcal{T}$ with $(\mathcal{T}, \gamma, \rho) \models \varphi$ and $(\mathcal{T}, \gamma', \rho) \models \varphi'$.*
7. *A reachability system $(\mathcal{T}, S)$ is* deterministic *iff the induced transition system is deterministic.*
8. *A reachability system $(\mathcal{T}, S)$ is* ϵ-free *iff for any two configurations $\sigma, \sigma' \in \mathcal{T}_{Cfg}$, if $\sigma \Rightarrow_{\mathcal{S}} \sigma'$, then $\sigma \neq \sigma'$.*
9. *A configuration $\gamma \in \mathcal{T}_{Cfg}$* terminates *in $(\mathcal{T}_{Cfg}, \Rightarrow_{\mathcal{S}})$ iff there is no infinite $\Rightarrow_{\mathcal{S}}$)-sequence starting with γ.*
10. *A $\Rightarrow_{\mathcal{S}}$-*path *is a finite sequence $\gamma_0 \Rightarrow_{\mathcal{S}} \gamma_1 \Rightarrow_{\mathcal{S}} \ldots \Rightarrow_{\mathcal{S}} \gamma_n$ with $\gamma_0, \ldots, \gamma_n \in \mathcal{T}_{Cfg}$.*
11. *A $\Rightarrow_{\mathcal{S}}$-path is* complete *iff it is not a strict prefix of any other $\Rightarrow_{\mathcal{S}}$-path.*
12. *A one-path reachability rule $\varphi \Rightarrow^{\exists} \varphi'$ is* satisfied *in a reachability system $\mathcal{S} = (\mathcal{T}, S)$, written $\mathcal{S} \vDash_{\mathsf{RL}} \varphi \Rightarrow^{\exists} \varphi'$, iff for every $\gamma \in \mathcal{T}_{Cfg}$ such that γ terminates in $(\mathcal{T}_{Cfg}, \Rightarrow_{\mathcal{S}})$ and for any valuation $\rho : Var \to \mathcal{T}$ such that $(\mathcal{T}, \gamma, \rho) \models \varphi$, there exists some $\gamma' \in \mathcal{T}_{Cfg}$ such that $\gamma \Rightarrow^{*}_{\mathcal{S}} \gamma'$ and $(\mathcal{T}, \gamma', \rho) \models \varphi'$.*

Remark 2. *We work only with ϵ-free reachability systems. A reachability system $(\mathcal{T}, S)$ is* ϵ-free *iff for any two configurations $\sigma, \sigma' \in \mathcal{T}_{Cfg}$, if $\sigma \Rightarrow_{\mathcal{S}} \sigma'$, then $\sigma \neq \sigma'$. (We are not aware of any practical reachability system that would use these ϵ steps.)*

B Cartesian Reachability logic

Definition 5 (CRL Syntax)**.** *We define the syntax of Cartesian Reachability logic as follows:*

- *A* list-pattern *has the shape $[\varphi_1, \ldots, \varphi_k]$, where each φ_j (for $j \in \{1, \ldots, k\}$) is a matching logic pattern.*
- *A* constrained list-pattern (CLP) *is a conjunction $\Psi_0 \wedge P$ of a list-pattern Ψ_0 and a structureless pattern P.*
- *An* existentially-quantified constrained list-pattern (ECLP) *has the form $\exists \vec{Y}. \Psi$, where Ψ is a CLP and $\vec{Y}$ is a (possibly empty) list of variables.*
- *A* One-Path Cartesian reachability claim of arity k *has the shape $\Phi \Rightarrow^{c\exists} \Psi$, where Φ is a CLP and Ψ is an ECLP.*

Proof of Proposition 1. Follows by firstorder reasoning from the definitions of semantics of CRL (Definition 1) and RL (Definition 4). □

B.1 Proof of Theorem 1

Definition 6. *We translate a language semantics into a semantics for lists of configurations as follows.*

1. *Let $((\Sigma, Var), Cfg)$ be a reachability-logic signature. Then*

$$((\Sigma, Var^*), Cfg)^* = ((\Sigma^*, Var^*), Cfg^*)$$

where

   (a) $\Sigma^* = \Sigma \cup \{cfgitem, cfgconcat, cfgheat, cfgnil\}$
   (b) Cfg^* *is a fresh sort (representing the sort of lists of configurations);*
   (c) $Var^* = Var \cup Var_{Cfg^*}$, *where* Var_{Cfg^*} *is an infinite set of variables of sort* Cfg^*, *distinct from varibles in* Var;
   (d) $cfgitem$ *a fresh symbol of sort* $Cfg \to Cfg^*$;
   (e) $cfgnil$ *a fresh symbol of sort* Cfg^*;
   (f) $cfgconcat$ *a fresh symbol of sort* $Cfg^* \times Cfg^* \to Cfg^*$; *and*
   (g) $cfgheat$ *is a fresh symbol of sort* $Cfg^* \times Cfg \times Cfg^* \to Cfg^*$.

2. *Let* S *be a set of reachability rules over* $(\mathbf{\Sigma}, Cfg)$. *We generate a set of reachability rules* S^* *over* $(\mathbf{\Sigma}, Cfg)^*$ *by*
   (a) *defining a function* $heat : Var \times \mathsf{Pattern} \times Var \to \mathsf{Pattern}$ *by*
   $$heat(L, \phi \wedge P, R) = cfgheat(L, \phi, R) \wedge P$$
   (b) *setting*
   $$S^* = \{heat(L, \varphi, R) \Rightarrow^{\exists} heat(L, \varphi', R) \mid (\varphi \Rightarrow^{\exists} \varphi') \in S\},$$
   where L, R *are distinct fresh variables (not occurring in any rule in* S*).*

3. *Let* (Σ, Var) *be a matching logic signature, and let* $\mathcal{T}$ *be a configuration model; that is, a* Σ*-algebra. We generate a* Σ^**-algebra* $\mathcal{T}*$, *which interprets all sorts and symbols from* Σ *as in* $\mathcal{T}$, *and in addition interprets*
   (a) *the sort* $Cfg*$ *as the set of all finite lists* $[c_1; \ldots; c_n]$ *for* $n \in \mathbb{N}$, *where* c_i *is an element of sort* Cfg *for any* $0 \leq i \leq n$;
   (b) *the symbol* $cfgitem$ *as the function* $\lambda c.\, [c]$;
   (c) *the symbol* $cfgnil$ *as the empty list* ($[]$);
   (d) *the symbol* $cfgconcat$ *as the function* $\lambda l_1, l_2.\, l_1 \texttt{++} l_2$, *where* `++` *is list concatenation; and*
   (e) *the symbol* $cfgheat$ *as the function* $\lambda l_1, c, l_2.\, l_1 \texttt{++}[c] \texttt{++} l_2$.

4. *Let* (Σ, Var) *be a matching logic signature, let* $\mathcal{T}$ *be a configuration model, and let* ρ *be a* $\mathcal{T}$*-valuation. We define a* $\mathcal{T}^*$*-valuation* ρ^* *by letting* $\rho^*(v) = \rho(v)$ *for any* $v \in Var$, *and letting* $\rho^*(v) = a$, *where* a *is some arbitrary element of sort* Cfg^*, *for any variable* v *of sort* Cfg^*.

5. *Let* $\mathcal{S} = (\mathcal{T}, S)$ *be a reachability system over* (Σ, Cfg). *We generate a reachability system* $\mathcal{S}^*$ *over* $(\Sigma, Cfg)^*$ *by setting* $\mathcal{S}^* = (\mathcal{T}^*, S^*)$.

Lemma 10. *Let* (Σ, Var) *be a matching logic signature,* $\mathcal{T}$ *be a configuration model, and* ρ *be a* $\mathcal{T}$*-valuation. Then for any* (Σ, Var)*-term* t,

$$\rho^*(t) = \rho(t)\,. \tag{15}$$

Proof of Lemma 10. By induction on the term t.

- $t \equiv v$ for $v \in \mathit{Var}$ - follows from the definition of ρ^*.
- $t \equiv f(t_1, \ldots, t_k)$ - we have $\rho(t_i) = \rho^*(t_i)$ for any $i \in \{1, \ldots, k\}$ by the induction hypothesis. Then

$$\begin{aligned}\rho^*(f(t_1, \ldots, t_k)) =& \mathcal{T}^*{}_f(\rho^*(t_1), \ldots, \rho^*(t_k)) \\ =& \mathcal{T}^*{}_f(\rho(t_1), \ldots, \rho(t_k)) \\ =& \mathcal{T}_f(\rho(t_1), \ldots, \rho(t_k)) \\ =& \rho(f(t_1, \ldots, t_k))\end{aligned}$$

  where the second-to-last equality holds by definition of $\mathcal{T}^*$.

□

Lemma 11. *The star extension on matching logic models is conservative, in the following sense. For any Σ-model $\mathcal{T}$, any $\mathcal{T}$-valuation ρ, any Σ-sort s, any $\gamma \in \mathcal{T}_s$, and any matching logic s-pattern φ,*

$$(\mathcal{T}, \gamma, \rho) \models \varphi \iff (\mathcal{T}^*, \gamma, \rho^*) \models \varphi$$

Proof of Lemma 11. By induction on φ.

- $\varphi \equiv t_1 = t_2$ - follows from Lemma 10
- $\varphi \equiv t$, where π is a term (of sort s) - follows from Lemma 10.
- $\varphi \equiv \varphi_1 \wedge \varphi_2$ - follows from the induction hypothesis.
- $\varphi \equiv \neg\varphi'$ - follows from the induction hypothesis.
- $\varphi \equiv \exists x : s'.\, \varphi'$.

  The induction hypothesis is: for any $\mathcal{T}$, γ, ρ,

$$(\mathcal{T}, \gamma, \rho) \models \varphi' \iff (\mathcal{T}^*, \gamma, \rho^*) \models \varphi'$$

  We want to prove that for any $\mathcal{T}$, γ, ρ,

$$(\mathcal{T}, \gamma, \rho) \models \exists x : s'.\varphi' \iff (\mathcal{T}^*, \gamma, \rho^*) \models \exists x : s'.\varphi'$$

  First, let us prove the left-to-right implication. The left-hand-side of the claim is equivalent with

$$\exists \rho'.(\forall y.y \neq x \to \rho'(y) = \rho(y)) \wedge (T, \gamma, \rho') \models \varphi'$$

  From the induction hypothesis, this is further equivalent with

$$\exists \rho'.(\forall y.y \neq x \to \rho'(y) = \rho(y)) \wedge (T^*, \gamma, \rho'^*) \models \varphi'$$

  Since $\forall y.y \neq x \to \rho'^*(y) = \rho'(y) = \rho(y) = \rho^*(y)$, we deduce $(\mathcal{T}^*, \gamma, \rho^*) \models \exists x : s'.\varphi'$.

  Conversely, the right-hand-side of the claim is equivalent with

$$\exists \rho''.(\forall y.y \neq x \to \rho'(y) = \rho * (y)) \wedge (T^*, \gamma, \rho'') \models \varphi'$$

  Let ρ' be defined by $\rho'(y) = \rho(y)$ if $y \neq x$ and $\rho'(x) = \rho''(x)$. Then it is easy to see that $\rho'^* = \rho''$, whence by the induction hypothesis we obtain that $(T, \gamma, \rho') \models \varphi'$, and by the definition of ρ', $(T, \gamma, \rho) \models \exists x : s'.\varphi'$.

□

Lemma 12. *We have*

$$C \Rightarrow_{\mathcal{S}^*} C'$$

if and only if

there exists a rule $\phi \wedge P \Rightarrow^{\exists} \phi' \wedge P' \in S$ *and valuation* $\rho : Var^* \to \mathcal{T}^*$ *such that*

- $(\mathcal{T}^*, \rho) \models P$; *and*
- $(\mathcal{T}^*, \rho) \models P'$; *and*
- $C = \rho(L) +\!\!+ [\rho(\phi)] +\!\!+ \rho(R)$; *and*
- $C' = \rho(L) +\!\!+ [\rho(\phi')] +\!\!+ \rho(R)$,

Proof. We have

$$C \Rightarrow_{\mathcal{S}^*} C'$$

iff (by Definition 4)

there exists a rule $\varphi \Rightarrow^{\exists} \varphi' \in S^*$ and valuation $\rho : Var^* \to \mathcal{T}^*$ such that $(\mathcal{T}^*, C, \rho) \models \varphi$ and $(\mathcal{T}^*, C', \rho) \models \varphi'$,

iff (by Remark 1 and Definition 6)

there exists a rule $\phi \wedge P \Rightarrow^{\exists} \phi' \wedge P' \in S$ and valuation $\rho : Var^* \to \mathcal{T}^*$ such that

$$(\mathcal{T}^*, C, \rho) \models cfgheat(L, \phi, R) \wedge P$$

and

$$(\mathcal{T}^*, C', \rho) \models cfgheat(L, \phi', R) \wedge P' \,,$$

iff (by Definition 2 and Lemma 2)

there exists a rule $\phi \wedge P \Rightarrow^{\exists} \phi' \wedge P' \in S$ and valuation $\rho : Var^* \to \mathcal{T}^*$ such that $(\mathcal{T}^*, \rho) \models P$ and $(\mathcal{T}^*, \rho) \models P'$ and

$$(\mathcal{T}^*, C, \rho) \models cfgheat(L, \phi_l, R)$$

and

$$(\mathcal{T}^*, C', \rho) \models cfgheat(L, \phi'_j, R) \,,$$

iff (by Definition 2 and Definition 6)

there exists a rule $\phi \wedge P \Rightarrow^{\exists} \phi' \wedge P' \in S$ and valuation $\rho : \mathit{Var}^* \to \mathcal{T}^*$ such that

- $(\mathcal{T}^*, \rho) \models P$; and
- $(\mathcal{T}^*, \rho) \models P'$; and
- $C = \rho(L)\texttt{++}[\rho(\phi)]\texttt{++}\rho(R)$; and
- $C' = \rho(L)\texttt{++}[\rho(\phi')]\texttt{++}\rho(R)$.

That proves the desired equivalence. □

Lemma 13. *Let $\mathcal{S} = (\mathcal{T}, S)$ be a reachability system over (Σ, Cfg). For any $k \geq 1$, any configurations $c_1, \ldots, c_k, c' \in \mathcal{T}_{\mathit{Cfg}}$, and any $1 \leq i \leq k$, we have*

$$c_i \Rightarrow_{\mathcal{S}} c' \iff [c_1, \ldots, c_k] \Rightarrow_{\mathcal{S}^*} [c_1, \ldots, c_{i-1}, c', c_{i+1}, \ldots, c_k]$$

Proof of Lemma 13. We have

$$[c_1, \ldots, c_k] \Rightarrow_{\mathcal{S}^*} [c_1, \ldots, c_{i-1}, c', c_{i+1}, \ldots, c_k]$$

iff (by Lemma 12)

there exists a rule $\phi \wedge P \Rightarrow^{\exists} \phi' \wedge P' \in S$ and valuation $\rho : \mathit{Var}^* \to \mathcal{T}^*$ such that

- $(\mathcal{T}^*, \rho) \models P$; and
- $(\mathcal{T}^*, \rho) \models P'$; and
- $[c_1, \ldots, c_k] = \rho(L)\texttt{++}[\rho(\phi_l)]\texttt{++}\rho(R)$; and
- $[c_1, \ldots, c_{i-1}, c', c_{i+1}, \ldots, c_k] = \rho(L)\texttt{++}[\rho(\phi_j)]\texttt{++}\rho(R)$.

Suppose we have such valuation ρ. We can surely construct valuation $\rho_0 : \mathit{Var} \to \mathcal{T}$ by letting

$$\rho_0(v) = \begin{cases} \rho(v) & \text{if } \rho(v) \in \mathcal{T} \\ a & \text{if } \rho(v) \notin \mathcal{T} \end{cases}$$

(where $a \in \mathcal{T}$ is some arbitrary element). Now, for any $v \in FV(\phi) \cup FV(\phi') \cup FV(P) \cup FV(P')$ it holds that $((\rho_0)^*)(v) = \rho(v)$. Why? Because v has some sort s from Σ (that is, $s \neq \mathit{Cfg}^*$). Therefore, we can use Lemma 4 to change the goal to one saying that

there exists a rule $\phi \wedge P \Rightarrow^{\exists} \phi' \wedge P' \in S$ and valuation $\rho_0 : \mathit{Var} \to \mathcal{T}$ such that

- $(\mathcal{T}^*, (\rho_0)^*) \models P$; and
- $(\mathcal{T}^*, (\rho_0)^*) \models P'$; and
- $[c_1, \ldots, c_k] = ((\rho_0)^*)(L)\texttt{++}[((\rho_0)^*)(\phi_l)]\texttt{++}((\rho_0)^*)(R)$; and
- $[c_1, \ldots, c_{i-1}, c', c_{i+1}, \ldots, c_k] = ((\rho_0)^*)(L)\texttt{++}[((\rho_0)^*)(\phi_j)]\texttt{++}((\rho_0)^*)(R)$

(where the opposite implication follows by choice $\rho := (\rho_0)^*$). Now, we use Lemma 11 and definition of starred valuation to change the goal to one saying that

there exists a rule $\phi \wedge P \Rightarrow^{\exists} \phi' \wedge P' \in S$ and valuation $\rho_0 : \mathit{Var} \to \mathcal{T}$ such that

- $(\mathcal{T}, \rho_0) \models P$; and
- $(\mathcal{T}, \rho_0) \models P'$; and
- $[c_1, \ldots, c_k] = \rho_0(L)\texttt{++}[\rho_0(\phi_l)]\texttt{++}\rho_0(R)$; and
- $[c_1, \ldots, c_{i-1}, c', c_{i+1}, \ldots, c_k] = \rho_0(L)\texttt{++}[\rho_0(\phi_j)]\texttt{++}\rho_0(R)$.

Now, by list reasoning, this is equivalent to saying that

there exists a rule $\phi \wedge P \Rightarrow^{\exists} \phi' \wedge P' \in S$ and valuation $\rho_0 : \mathit{Var} \to \mathcal{T}$ such that there exists some i' satisfying $1 \leq i' \leq k$ such that

- $(\mathcal{T}, \rho_0) \models P_l$; and
- $(\mathcal{T}, \rho_0) \models P_j$; and
- $[c_1, \ldots, c_{i'-1}] = \rho_0(L)$; and
- $c_{i'} = \rho_0(\phi_l)$; and
- $[c_{i'+1}, \ldots, c_k] = \rho_0(R)$; and
- $[c_1, \ldots, c_{i-1}, c', c_{i+1}, \ldots, c_k] = \rho_0(L)\texttt{++}[\rho_0(\phi_j)]\texttt{++}\rho_0(R)$.

Now, let us define c'_z by

$$c'_z = \begin{cases} c' & \text{if } z = i \\ c_z & \text{if } z \neq i \end{cases}$$

after which the goal is equivalent to saying that

there exists a rule $\phi \wedge P \Rightarrow^{\exists} \phi' \wedge P' \in S$ and valuation $\rho_0 : \mathit{Var} \to \mathcal{T}$ such that there exists some i' satisfying $1 \leq i' \leq k$ such that

- $(\mathcal{T}, \rho_0) \models P$; and
- $(\mathcal{T}, \rho_0) \models P'$; and
- $[c_1, \ldots, c_{i'-1}] = \rho_0(L)$; and
- $c_{i'} = \rho_0(\phi)$; and
- $[c_{i'+1}, \ldots, c_k] = \rho_0(R)$; and
- $[c'_1, \ldots, c'_{i'-1}] = \rho_0(L)$; and
- $c'_{i'} = \rho_0(\phi')$; and
- $[c'_{i'+1}, \ldots, c'_k] = \rho_0(R)$.

Since L, R were fresh, they do not occur in ϕ nor in ϕ'. Therefore, using Lemma 4, we can equivalently say that

there exists a rule $\phi \wedge P \Rightarrow^{\exists} \phi' \wedge P' \in S$ and valuation $\rho_0 : \mathit{Var} \to \mathcal{T}$ such that there exists some i' satisfying $1 \leq i' \leq k$ such that

- $(\mathcal{T}, \rho_0) \models P$; and
- $(\mathcal{T}, \rho_0) \models P'$; and
- $c_{i'} = \rho_0(\phi_l)$; and
- $c'_{i'} = \rho_0(\phi_j)$.

(The downwards implication is trivial, as it is only removing constraints; the upwards implication is from the fact that we can always choose a valuation ρ_0 satisfying the constraints.) But that is equivalent (Definition 2) to saying that

there exists a rule $\phi \wedge P \Rightarrow^{\exists} \phi' \wedge P' \in S$ and there exists some i' satisfying $1 \leq i' \leq k$ and valuation $\rho_0 : \mathit{Var} \to \mathcal{T}$ such that

- $(\mathcal{T}, c_{i'}, \rho_0) \models \phi \wedge P$; and
- $(\mathcal{T}, c'_{i'}, \rho_0) \models \phi' \wedge P'$.

But that is equivalent to saying that

there exists some i' satisfying $1 \leq i' \leq k$ such that $c_{i'} \Rightarrow_{\mathcal{S}} c'_{i'}$,

which is almost equivalent to the left side of the equivalence we want to prove: that

$c_i \Rightarrow_{\mathcal{S}} c_i'$.

The upwards implication is trivial; the downwards is as follows. If $i = i'$, we are done. But otherwise, it would follow (by definition of $c_{i'}'$) that $c_{i'} \Rightarrow_{\mathcal{S}} c_{i'}$, which contradicts Remark 2. □

Lemma 14. *$(\mathcal{T}^*, C, \rho) \models mkList(\phi_1, \ldots, \phi_k)$ iff there exists $c_1, \ldots, c_k \in \mathcal{T}_{Cfg}$ such that $C = [c_1, \ldots, c_k]$ and for every $\rho' : Var \to \mathcal{T}$ satisfying $\rho'(v) = \rho(v)$ for any $v \in FV(mkList(\phi_1, \ldots, \phi_k))$, it holds that $(\mathcal{T}, c_1, \rho') \models \phi_1$ and ... and $(\mathcal{T}, c_k, \rho') \models \phi_k$.*

Proof. By induction on k.

- If $k = 1$, then we have to prove that

  $(\mathcal{T}^*, C, \rho) \models cfgitem(\phi_1)$ iff there exists $c_1 \in \mathcal{T}_{Cfg}$ such that $C = [c_1]$ and for every $\rho' : Var \to \mathcal{T}$ satisfying $\rho'(v) = \rho(v)$ for any $v \in FV(cfgitem(\phi_1))$, it holds that $(\mathcal{T}, c, \rho') \models \phi_1$.

  By definition 2, this is equivalent to

  $C = \rho(cfgitem(\phi_1))$ iff there exists $c_1 \in \mathcal{T}_{Cfg}$ such that $C = [c_1]$ and for every $\rho' : Var \to \mathcal{T}$ satisfying $\rho'(v) = \rho(v)$ for any $v \in FV(cfgitem(\phi_1))$, it holds that $c_1 = \rho'(\phi_1)$.

  By Definition 6, this is equivalent to

  $C = [\rho(\phi_1)]$ iff there exists $c_1 \in \mathcal{T}_{Cfg}$ such that $C = [c_1]$ and for every $\rho' : Var \to \mathcal{T}$ satisfying $\rho'(v) = \rho(v)$ for any $v \in FV(cfgitem(\phi_1))$, it holds that $c_1 = \rho'(\phi_1)$.

  We prove each implication separately. For the left-to-right implication, we let $c_1 := \rho(\phi_1)$ and have to prove that $\rho(\phi_1) = \rho'(\phi_1)$, which follows from Lemma 4. The right-to-left implication also follows from Lemma 4.

- If $k = k' + 1$, we assume the induction hypothesis saying that

  for every $C, \phi_1, \ldots, \phi_{k'}$, $(\mathcal{T}^*, C, \rho) \models mkList(\phi_1, \ldots, \phi_{k'})$ iff there exists $c_1, \ldots, c_{k'} \in \mathcal{T}_{Cfg}$ such that $C = [c_1, \ldots, c_{k'}]$ and for every $\rho' : Var \to \mathcal{T}$ satisfying $\rho'(v) = \rho(v)$ for any $v \in FV(mkList(\phi_1, \ldots, \phi_{k'}))$, it holds that $(\mathcal{T}, c_1, \rho') \models \phi_1$ and ... and $(\mathcal{T}, c_{k'}, \rho') \models \phi_k$,

  and have to prove that

  $(\mathcal{T}^*, C, \rho) \models mkList(\phi_1, \ldots, \phi_{k'+1})$ iff there exists $c_1, \ldots, c_{k'+1} \in \mathcal{T}_{Cfg}$ such that $C = [c_1, \ldots, c_{k'+1}]$ and for every $\rho' : Var \to \mathcal{T}$ satisfying $\rho'(v) = \rho(v)$ for any $v \in FV(mkList(\phi_1, \ldots, \phi_{k'+1}))$, it holds that $(\mathcal{T}, c_1, \rho') \models \phi_1$ and ... and $(\mathcal{T}, c_{k'+1}, \rho') \models \phi_{k'+1}$,

  which is (by Definition 2) equivalent to

$C = \rho(mkList(\phi_1, \ldots, \phi_{k'+1}))$ iff there exists $c_1, \ldots, c_{k'+1} \in \mathcal{T}_{Cfg}$ such that $C = [c_1, \ldots, c_{k'+1}]$ and for every $\rho' : Var \to \mathcal{T}$ satisfying $\rho'(v) = \rho(v)$ for any $v \in FV(mkList(\phi_1, \ldots, \phi_{k'+1}))$, it holds that $c_1 = \rho'(\phi_1)$ and $\ldots$ and $c_{k'+1} = \rho'(\phi_{k'+1})$,

which is (by Definition 6) equivalent to

$C = [\rho(\phi_1)]$++C' and $C' = \rho(mkList(\phi_2, \ldots, \phi_{k'+1}))$ iff there exists $c_1, \ldots, c_{k'+1} \in \mathcal{T}_{Cfg}$ such that $C = [c_1, \ldots, c_{k'+1}]$ and for every $\rho' : Var \to \mathcal{T}$ satisfying $\rho'(v) = \rho(v)$ for any $v \in FV(mkList(\phi_1, \ldots, \phi_{k'+1}))$, it holds that $c_1 = \rho'(\phi_1)$ and $\ldots$ and $c_{k'+1} = \rho'(\phi_{k'+1})$,

which is by the induction hypothesis with $\phi_1 := \phi_2, \ldots, \phi_k := \phi_{k'+1}$ and α-renaming equivalent to

$C = [\rho(\phi_1)]$++C' and there exists $c_2, \ldots, c_{k'+1} \in \mathcal{T}_{Cfg}$ such that $C' = [c_2, \ldots, c_{k'+1}]$ and for every $\rho' : Var \to \mathcal{T}$ satisfying $\rho'(v) = \rho(v)$ for any $v \in FV(mkList(\phi_2, \ldots, \phi_{k'+1}))$, it holds that $(\mathcal{T}, c_2, \rho') \models \phi_2$ and $\ldots$ and $(\mathcal{T}, c_{k'+1}, \rho') \models \phi_{k'+1}$, iff there exists $c_1, \ldots, c_{k'+1} \in \mathcal{T}_{Cfg}$ such that $C = [c_1, \ldots, c_{k'+1}]$ and for every $\rho' : Var \to \mathcal{T}$ satisfying $\rho'(v) = \rho(v)$ for any $v \in FV(mkList(\phi_1, \ldots, \phi_{k'+1}))$, it holds that $c_1 = \rho'(\phi_1)$ and $\ldots$ and $c_{k'+1} = \rho'(\phi_{k'+1})$,

which is (by firstorder reasoning and simplification of list append) equivalent to

there exists $c_2, \ldots, c_{k'+1} \in \mathcal{T}_{Cfg}$ such that $C = [\rho(\phi_1), c_2, \ldots, c_{k'+1}]$ and for every $\rho' : Var \to \mathcal{T}$ satisfying $\rho'(v) = \rho(v)$ for any $v \in FV(mkList(\phi_2, \ldots, \phi_{k'+1}))$, it holds that $(\mathcal{T}, c_2, \rho') \models \phi_2$ and $\ldots$ and $(\mathcal{T}, c_{k'+1}, \rho') \models \phi_{k'+1}$, iff there exists $c_1, \ldots, c_{k'+1} \in \mathcal{T}_{Cfg}$ such that $C = [c_1, \ldots, c_{k'+1}]$ and for every $\rho' : Var \to \mathcal{T}$ satisfying $\rho'(v) = \rho(v)$ for any $v \in FV(mkList(\phi_1, \ldots, \phi_{k'+1}))$, it holds that $c_1 = \rho'(\phi_1)$ and $\ldots$ and $c_{k'+1} = \rho'(\phi_{k'+1})$.

We simplify the goal using Definition 2 to

there exists $c_2, \ldots, c_{k'+1} \in \mathcal{T}_{Cfg}$ such that $C = [\rho(\phi_1), c_2, \ldots, c_{k'+1}]$ and for every $\rho' : Var \to \mathcal{T}$ satisfying $\rho'(v) = \rho(v)$ for any $v \in FV(mkList(\phi_2, \ldots, \phi_{k'+1}))$, it holds that $c_2 = \rho'(\phi_2)$ and $\ldots$ and $c_{k'+1} = \rho'(\phi_{k'+1})$, iff there exists $c_1, \ldots, c_{k'+1} \in \mathcal{T}_{Cfg}$ such that $C = [c_1, \ldots, c_{k'+1}]$ and for every $\rho' : Var \to \mathcal{T}$ satisfying $\rho'(v) = \rho(v)$ for any $v \in FV(mkList(\phi_1, \ldots, \phi_{k'+1}))$, it holds that $c_1 = \rho'(\phi_1)$ and $\ldots$ and $c_{k'+1} = \rho'(\phi_{k'+1})$.

We prove each implication separately.

– Assuming

> there exists $c_2, \ldots, c_{k'+1} \in \mathcal{T}_{Cfg}$ such that $C = [\rho(\phi_1), c_2, \ldots, c_{k'+1}]$ and for every $\rho' : Var \to \mathcal{T}$ satisfying $\rho'(v) = \rho(v)$ for any $v \in FV(mkList(\phi_2, \ldots, \phi_{k'+1}))$, it holds that $c_2 = \rho'(\phi_1)$ and ... and $c_{k'+1} = \rho'(\phi_k)$,

we prove that

> there exists $c_1, \ldots, c_{k'+1} \in \mathcal{T}_{Cfg}$ such that $C = [c_1, \ldots, c_{k'+1}]$ and for every $\rho' : Var \to \mathcal{T}$ satisfying $\rho'(v) = \rho(v)$ for any $v \in FV(mkList(\phi_1, \ldots, \phi_{k'+1}))$, it holds that $c_1 = \rho'(\phi_1)$ and ... and $c_{k'+1} = \rho'(\phi_{k'+1})$.

by choosing $c_1 := \rho'(\phi_1)$ and using Lemma 4
(note that $FV(mkList(\phi_2, \ldots, \phi_{k'+1})) \subseteq FV(mkList(\phi_1, \ldots, \phi_{k'+1}))$).

– Assuming

> there exists $c_1, \ldots, c_{k'+1} \in \mathcal{T}_{Cfg}$ such that $C = [c_1, \ldots, c_{k'+1}]$ and for every $\rho' : Var \to \mathcal{T}$ satisfying $\rho'(v) = \rho(v)$ for any $v \in FV(mkList(\phi_1, \ldots, \phi_{k'+1}))$, it holds that $c_1 = \rho'(\phi_1)$ and ... and $c_{k'+1} = \rho'(\phi_{k'+1})$,

we prove that

> there exists $c_2, \ldots, c_{k'+1} \in \mathcal{T}_{Cfg}$ such that $C = [\rho(\phi_1), c_2, \ldots, c_{k'+1}]$ and for every $\rho' : Var \to \mathcal{T}$ satisfying $\rho'(v) = \rho(v)$ for any $v \in FV(mkList(\phi_2, \ldots, \phi_{k'+1}))$, it holds that $c_2 = \rho'(\phi_1)$ and ... and $c_{k'+1} = \rho'(\phi_k)$

by setting $c_i := c_i$ and again noting that

$$FV(mkList(\phi_2, \ldots, \phi_{k'+1})) \subseteq FV(mkList(\phi_1, \ldots, \phi_{k'+1}))\,.$$

□

Lemma 15. *Let $\mathcal{S} = (\mathcal{T}, S)$ be a reachability system over (Σ, Cfg). Then for any $C, C' \in \mathcal{T}^*_{Cfg^*}$, if $C \Rightarrow_{\mathcal{S}^*} C'$, then the length of C (it is a list) is the same as the length of C'.*

Proof. Assume $C \Rightarrow_{\mathcal{S}^*} C'$. Then by Lemma 12,

> there exists a rule $\phi \wedge P_l \Rightarrow^{\exists} \phi' \wedge P' \in S$ and valuation $\rho : Var^* \to \mathcal{T}^*$ such that
>
> - $(\mathcal{T}^*, \rho) \models P$; and
> - $(\mathcal{T}^*, \rho) \models P'$; and
> - $C = \rho(L)$++$[\rho(\phi)]$++$\rho(R)$; and
> - $C' = \rho(L)$++$[\rho(\phi')]$++$\rho(R)$.

But then C and C' have the same length. □

Lemma 16 (At most one component changes)**.** *Let $\mathcal{S} = (\mathcal{T}, S)$ be a reachability system over (Σ, Cfg). Then for any $C, C' \in \mathcal{T}^* Cfg^*$ satisfying $C \Rightarrow_{\mathcal{S}^*} C'$ there exists some $i \in \mathbb{N}$ such that for every $i' \in \mathbb{N}$ such that $i' \neq i$, we have $C[i'] = C'[i']$ if both are defined.*

Proof. Assume $C \Rightarrow_{\mathcal{S}^*} C'$. Then by Lemma 12,

there exists a rule $\phi \wedge P \Rightarrow^{\exists} \phi' \wedge P' \in S$ and valuation $\rho : Var^* \to \mathcal{T}^*$ such that

- $(\mathcal{T}^*, \rho) \models P_l$; and
- $(\mathcal{T}^*, \rho) \models P_j$; and
- $C = \rho(L)$++$[\rho(\phi)]$++$\rho(R)$; and
- $C' = \rho(L)$++$[\rho(\phi')]$++$\rho(R)$.

But then we can let $i := |\rho(L)|$, and the rest follows. □

The following definition and theorem on filtering infinite sequences are based on the Coq development of [31] (specifically, on https://github.com/runtimeverification/vlsm/blob/d6c8cee56708c7be2431b9743fe80ca6a7a29a58/theories/VLSM/Lib/StreamFilters.v).

Definition 7 (Filtering subsequence)**.** *Given a set A, a subset $P \subseteq A$ and a function $s : \mathbb{N} \to A$, a function $ns : \mathbb{N} \to \mathbb{N}$ is called a filtering subsequence for P on s, iff*

1. *ns is monotone;*
2. *$s(x) \notin P$ for any $x < ns(0)$;*
3. *$s(ns(j)) \in P$ for any $j \in \mathbb{N}$; and*
4. *for every $j \in \mathbb{N}$ and every x such that $ns(j) < x < ns(j+1)$, $s(x) \notin P$.*

Intuitively, the last condition says that ns does not skip any P-element in s.

Lemma 17 (Existence of filtering sequence for infinite occurrences)**.** *Let A be a set, let $P \subseteq A$, and let $s : \mathbb{N} \to A$ be a function whose output falls to P infinitely often (that is, $s(i) \in P$ for infinitely many i). Then there exists a filtering subsequence for P on s.*

Lemma 18. *For any reachability system $\mathcal{S} = (\mathcal{T}, S)$, any $C \in \mathcal{T}^*_{Cfg^*}$, and any $c_1, \ldots, c_k \in \mathcal{T}_{Cfg}$ such that $C = [c_1, \ldots, c_k]$, C is terminating in $(\mathcal{T}^*_{Cfg^*}, \Rightarrow_{S^*})$ iff for every $j \in \{1, \ldots, k\}$, c_j is terminating in $(\mathcal{T}_{Cfg}, \Rightarrow_S)$.*

Proof of Lemma 18. We prove both implications separately, by contraposition.

- Suppose some c_j is not terminating in $(\mathcal{T}_{Cfg}, \Rightarrow_S)$. In other words, there exists some infinite $\Rightarrow_{\mathcal{S}}$-sequence $c_j = d(0) \Rightarrow_{\mathcal{S}} d(1) \Rightarrow_{\mathcal{S}} d(2) \Rightarrow_{\mathcal{S}} \ldots$. Then

  $$C = [c_1, \ldots, c_{j-1}, d(0), c_{j+1}, \ldots, c_k] \Rightarrow_{\mathcal{S}^*} [c_1, \ldots, c_{j-1}, d(1), c_{j+1}, \ldots, c_k] \Rightarrow_{\mathcal{S}^*} \ldots$$

  is (by Lemma 13) an infinite $\Rightarrow_{\mathcal{S}^*}$-sequence. Therefore, C is not terminating in $(\mathcal{T}^*_{Cfg^*}, \Rightarrow_{S^*})$.

- Suppose C is not terminating in $(\mathcal{T}^*_{Cfg^*}, \Rightarrow_{S^*})$. In other words, there exists an infinite sequence $C = D(0) \Rightarrow_{\mathcal{S}^*} D(1) \Rightarrow_{\mathcal{S}^*} \ldots$. Then there exists a component j of the sequence which changes infinitely often in the sequence, because we have only k components. Now, consider the function $s : \mathbb{N} \to \mathcal{T}^*_{Cfg^*} \times \mathcal{T}^*_{Cfg^*}$ defined by $s(i) = (D(i), D(i+1))$, and let $P \subseteq \mathcal{T}^*_{Cfg^*} \times \mathcal{T}^*_{Cfg^*}$ be defined by $(X, X') \in P$ iff $X[j] \neq X'[j]$. By Lemma 16, we know that whenever $(X, X') \in P$, then for any j' satisfying $1 \leq j' \leq k$ and $j' \neq j$, we have $X[j'] = X'[j']$. Then, $s(i) \in P$ iff in the sequence C, on position i, it is exactly the jth component (and no other) which makes step. Now, by Lemma 17, there exists a filtering subsequence ns for P on s. But then

$$D(ns(0))[j] \Rightarrow_S D(ns(1))[j] \Rightarrow_S D(ns(2))[j] \Rightarrow_S \ldots$$

is a $(\mathcal{T}_{Cfg}, \Rightarrow_S)$ sequence witnessing the non-termination of $D(0)[j] = c_j$. Indeed, we have

  - $D(ns(0))[j] = D(0)[j]$, by (2) of Definition 7, the definition of P, and transitivity of equality;
  - for any $i \in \mathbb{N}$, $D(ns(i))[j] \Rightarrow_{\mathcal{S}} D(ns(i+1))[j]$. We prove this as follows. By (4) of Definition 7 and definition of P we have $D(ns(i+1))[j] = D(ns(i)+1)[j]$. Therefore, it is enough to show that

$$D(ns(i))[j] \Rightarrow_{\mathcal{S}} D(ns(i)+1)[j]\,.$$

    By (3) of Definition 7 and definition of P we have $D(ns(i))[j] \neq D(ns(i)+1)[j]$. By Lemma 13, it is enough to show that there exists $k \geq 1$, $c_1, \ldots, c_k, c' \in \mathcal{T}_{Cfg}$, and some i satisfying $1 \leq i \leq k$, such that $[c_1, \ldots, c_k] = D(ns(i))$ and $[c_1, \ldots, c_{i-1}, c', c_{i+1}, c_k] = D(ns(i)+1)$. But that follows from the fact that $D(ns(i)) \Rightarrow_{\mathcal{S}^*} D(ns(i)+1)$ and that $D(ns(i))[j] \neq D(ns(i)+1)[j]$ by Lemma 15 and Lemma 16.

□

Lemma 19. *For any reachability system $\mathcal{S} = (\mathcal{T}, S)$, any $C, C' \in \mathcal{T}^*_{Cfg^*}$, and any*

$$c_1, \ldots, c_k, c'_1, \ldots, c'_k \in \mathcal{T}_{Cfg}$$

such that $C = [c_1, \ldots, c_k]$ and $C' = [c'_1, \ldots, c'_k]$, $C \Rightarrow^*_{\mathcal{S}^*} C'$ iff for every $i \in \{1, \ldots, k\}$, $c_i \Rightarrow^*_{\mathcal{S}} c'_i$.

Proof of Lemma 19. We prove each implication separately.

- For the "if" implication, we assume that $c_i \Rightarrow^*_{\mathcal{S}} c'_i$ for any $i \in \{1, \ldots, k\}$, and have to prove that $[c_1, \ldots, c_k] \Rightarrow^*_{\mathcal{S}^*} [c'_1, \ldots, c'_k]$. We will prove that for any $j \in \{1, \ldots, k\}$, we it holds that

$$[c'_1, \ldots, c'_{j-1}, c_j, c_{j+1}, \ldots, c_k] \Rightarrow^*_{\mathcal{S}^*} [c'_1, \ldots, c'_{j-1}, c'_j, c_{j+1}, \ldots, c_k]\,,$$

  from which the goal follows by transitivity. Ok then, let $j \in \{1, \ldots, k\}$. By Lemma 13, it is enough to prove that $c_j \Rightarrow^*_{\mathcal{S}} c'_j$. But that holds by the assumption.

- For the "only if" implication, we assume $C \Rightarrow^*_{\mathcal{S}^*} C'$, $i \in \{1, \ldots, k\}$, and have to prove that $c_i \Rightarrow^*_{\mathcal{S}} c'_i$. Let $C_1, \ldots, C_l \in \mathcal{T}^*_{Cfg^*}$ be a sequence witnessing $C \Rightarrow^*_{\mathcal{S}^*} C'$; that is, we have $C = C_1$, $C_l = C'$, and $C_j \Rightarrow_{\mathcal{S}^*} C_{j+1}$ for any $j \in \{1, \ldots, l-1\}$. Let $i_1, \ldots, i_m \in \{1, \ldots, l-1\}$ be a strictly increasing sequence of maximal length such that $C_{i_j}[i] \neq C_{i_j+1}[i]$ for any $j \in \{1, \ldots, m\}$; that is, the sequence of positions in the witnessing sequence when the

component i changes. Then clearly, $C_1[i] = C_{i_1}[i]$ (otherwise we could create a longer sequence). Similarly, $C_l[i] = C_{i_m}[i]$. Now we claim that $C_{i_1}[i] \Rightarrow_{\mathcal{S}} \ldots \Rightarrow_{\mathcal{S}} C_{i_m}[i]$, from which the conclusion easily follows. We have to prove that for any $o \in \{1, \ldots, m\}$, it holds that $C_{i_o}[i] \Rightarrow_{\mathcal{S}} C_{i_{o+1}}[i]$. Let $d := i_{o+1} - i_o$; clearly, we have $d > 0$. By the definition of d, we have $C_{i_{o+1}}[i] = C_{i_o+d}[i]$. By definition of the sequence, in particular by maximality, we have $C_{i_o+d}[i] = C_{i_o+1}[i]$ (because there can be no change of the component i between the change at the position i_o and the change at the position i_{o+1}). Therefore, it is enough to show that $C_{i_o}[i] \Rightarrow_{\mathcal{S}} C_{i_o+1}[i]$. By Lemma 13 (using also Lemma 16 and Lemma 15), it is enough to show that $C_{i_o} \Rightarrow_{\mathcal{S}^*} C_{i_o+1}$, but that is trivial and we are done.

□

Definition 8. *We define mkList by letting*

- $\mathit{mkList}(\phi) = \mathit{cfgitem}(\phi)$; *and*
- $\mathit{mkList}(\phi_1, \ldots, \phi_k) = \mathit{cfgconcat}(\mathit{cfgitem}(\phi), \mathit{mkList}(\phi_2, \ldots, \phi_k))$ *whenever* $k > 1$.

Definition 9. *We define a function flatten from (potentially existentially-quantified) constrained list patterns to matching logic patterns over a star-extended signature by*

$$\begin{aligned}&\mathit{flatten}(\exists \vec{X}.\, [\varphi_1, \ldots, \varphi_k) \wedge P] \equiv \\ &\exists \vec{X}.\, \mathit{mkList}(Y_1, \ldots, Y_k) \wedge (\varphi_1^{\Box})[Y_1/\Box] \wedge \ldots \wedge (\varphi_k^{\Box})[Y_k/\Box] \wedge P\,,\end{aligned}$$

where $Y_1, \ldots, Y_k, \Box$ *are fresh. Furthermore, we let*

$$\mathit{flatten}^{\exists}(\Psi, \Psi') \equiv \mathit{flatten}(\Psi) \Rightarrow^{\exists} \mathit{flatten}(\Psi')\,.$$

Lemma 20 (On Flattening)**.** *For any matching logic* Σ*-model* $\mathcal{T}$*, any* $C \in \mathcal{T}^*_{\mathit{Cfg}^*}$*, and any* $\mathcal{T}^*$*-valuation* ρ*, we have*

$$(\mathcal{T}^*, C, \rho) \models \mathit{flatten}(\exists \vec{X}.\, [\varphi_1, \ldots, \varphi_k] \wedge P)$$

if and only if there exist configurations $c_1, \ldots, c_k \in \mathcal{T}_{\mathit{Cfg}}$ *such that* $C = [c_1, \ldots, c_k]$ *and there exists a* $\mathcal{T}$*-valuation* ρ_0 *satisfying* $\rho_0(v) = \rho(v)$ *for any* $v \in \mathit{Var} \setminus \vec{X}$ *such that for any* $j \in \{1, \ldots, k\}$*,* $(\mathcal{T}, c_j, \rho_0) \models \varphi_j \wedge P$*.*

Proof of Lemma 20. We have

$$(\mathcal{T}^*, C, \rho) \models \mathit{flatten}(\exists \vec{X}.\, (\varphi_1, \ldots, \varphi_k) \wedge P)$$

if and only if (by unfolding the definition of *flatten* and Definition 2)

there exists a $\mathcal{T}^*$-valuation ρ' satisfying $\rho'(v) = \rho(v)$ for any $v \in \mathit{Var}^* \setminus \vec{X}$ such that $(\mathcal{T}^*, C, \rho') \models \mathit{mkList}(Y_1, \ldots, Y_k)$ and for any $j \in \{1, \ldots, k\}$, $(\mathcal{T}^*, C, \rho') \models (\varphi_j^{\Box})[Y_j/\Box] \wedge P$,

if and only if (by Lemma 14)

there exists a $\mathcal{T}^*$-valuation ρ' satisfying $\rho'(v) = \rho(v)$ for any $v \in Var^* \setminus \vec{X}$ such that

- there exist configurations $c_1, \ldots, c_k \in \mathcal{T}_{Cfg}$ such that $C = [c_1, \ldots, c_k]$ and for every $\mathcal{T}$-valuation ρ'' satisfying $\rho''(Y_j) = \rho'(Y_j)$ for any $j \in \{1, \ldots, k\}$, it holds that for any $j \in \{1, \ldots, k\}$, $(\mathcal{T}, c_j, \rho'') \models Y_j$; and
- for any $j \in \{1, \ldots, k\}$, $(\mathcal{T}^*, C, \rho') \models (\varphi_j^{\square})[Y_j/\square] \wedge P$,

if and only if (by Definition 2 and firstorder reasoning)

there exists a $\mathcal{T}^*$-valuation ρ' satisfying $\rho'(v) = \rho(v)$ for any $v \in Var^* \setminus \vec{X}$ such that

- there exist configurations $c_1, \ldots, c_k \in \mathcal{T}_{Cfg}$ such that $C = [c_1, \ldots, c_k]$ and for any $j \in \{1, \ldots, k\}$, $\rho'(Y_j) = c_j$; and
- for any $j \in \{1, \ldots, k\}$, $(\mathcal{T}^*, C, \rho') \models (\varphi_j^{\square})[Y_j/\square] \wedge P$,

if and only if (by firstorder reasoning, Definition 2, and Lemma 2)

there exist configurations $c_1, \ldots, c_k \in \mathcal{T}_{Cfg}$ such that $C = [c_1, \ldots, c_k]$, and there exists a $\mathcal{T}^*$-valuation ρ' satisfying $\rho'(v) = \rho(v)$ for any $v \in Var^* \setminus \vec{X}$ such that for any $j \in \{1, \ldots, k\}$,

- $\rho'(Y_j) = c_j$;
- $(\mathcal{T}^*, \rho') \models (\varphi_j^{\square})[Y_j/\square]$; and
- $(\mathcal{T}^*, \rho') \models P$,

if and only if (by Lemma 11 and firstorder reasoning)

there exist configurations $c_1, \ldots, c_k \in \mathcal{T}_{Cfg}$ such that $C = [c_1, \ldots, c_k]$, and there exists a $\mathcal{T}$-valuation ρ' satisfying $\rho'(v) = \rho(v)$ for any $v \in Var \setminus \vec{X}$ such that for any $j \in \{1, \ldots, k\}$,

- $\rho'(Y_j) = c_j$;
- $(\mathcal{T}, \rho') \models (\varphi_j^{\square})[Y_j/\square]$; and
- $(\mathcal{T}, \rho') \models P$,

if and only if (by Lemma 5, since we have $\rho'(Y_j) = c_j$ on one side and $\rho_0^{c_j}(\square) = c_j$ on the other; for the implication from bottom to top, we also need the assumption that Y_j was fresh and Lemma 4 - in order to choose the valuation ρ' satisfying $\rho'(Y_j) = c_j$)

there exist configurations $c_1, \ldots, c_k \in \mathcal{T}_{Cfg}$ such that $C = [c_1, \ldots, c_k]$ and there exists a $\mathcal{T}$-valuation ρ_0 satisfying $\rho_0(v) = \rho(v)$ for any $v \in Var \setminus \vec{X}$ such that for any $j \in \{1, \ldots, k\}$,

- $(\mathcal{T}, \rho_0^{c_j}) \models \varphi_j^{\square}$; and
- $(\mathcal{T}, \rho_0) \models P$,

if and only if (by Lemma 6, Definition 2, and Lemma 2)

there exist configurations $c_1, \ldots, c_k \in \mathcal{T}_{Cfg}$ such that $C = [c_1, \ldots, c_k]$ and there exists a $\mathcal{T}$-valuation ρ_0 satisfying $\rho_0(v) = \rho(v)$ for any $v \in Var \setminus \vec{X}$ such that for any $j \in \{1, \ldots, k\}$, $(\mathcal{T}, c_j, \rho_0) \models \varphi_j \wedge P$,

which is what we wanted to prove. □

Lemma 21.

$$\mathcal{S} \vDash_{\mathsf{CRL}} \Psi \Rightarrow^{c\exists} \Psi' \iff \mathcal{S}^* \vDash_{\mathsf{RL}} \mathit{flatten}(\Psi) \Rightarrow^{c\exists} \mathit{flatten}(\Psi')$$

Proof of Lemma 21. We prove each implication separately.

1. For the left-to-right implication, we let $\mathcal{S} = (\mathcal{T}, S)$ and $\Psi \equiv (\varphi_1, \ldots, \varphi_k) \wedge P$ and $\Psi' \equiv \exists \vec{Y}. (\varphi'_1, \ldots, \varphi'_k) \wedge P'$, and assume that

   $$\mathcal{S} \vDash_{\mathsf{CRL}} \Psi \Rightarrow^{c\exists} \Psi' ;$$

   that is, (i)

   for all configurations $c_1, \ldots, c_k \in \mathcal{T}_{Cfg}$ which terminate in $(\mathcal{T}_{Cfg}, \Rightarrow_{\mathcal{S}})$ and any $\mathcal{T}$-valuation ρ_1, whenever $(\mathcal{T}, c_1, \rho_1) \models \varphi_1 \wedge P$ and ... and $(\mathcal{T}, c_k, \rho_1) \models \varphi_k \wedge P$, then there exist configurations $c'_1, \ldots, c'_k \in \mathcal{T}_{Cfg}$ such that $c_1 \Rightarrow^*_{\mathcal{S}} c'_1$ and ... and $c_k \Rightarrow^*_{\mathcal{S}} c'_k$, and there also exists an $\mathcal{T}$-valuation ρ_2 satisfying $\rho_1(v) = \rho_2(v)$ for any $v \in Var \setminus \vec{Y}$, and $(\mathcal{T}, c'_1, \rho_2) \models \varphi'_1 \wedge P'$ and ... and $(\mathcal{T}, c'_k, \rho_2) \models \varphi'_k \wedge P'$.

   We have to prove that

   for every $C \in \mathcal{T}^*_{Cfg^*}$ such that C terminates in $(\mathcal{T}^*_{Cfg^*}, \Rightarrow_{\mathcal{S}^*})$ and for any valuation $\rho : Var^* \to \mathcal{T}^*$ such that $(\mathcal{T}^*, C, \rho) \models \mathit{flatten}(\Psi)$, there exists some $C' \in \mathcal{T}^*_{Cfg^*}$ such that $C \Rightarrow^*_{\mathcal{S}^*} C'$ and $(\mathcal{T}^*, C', \rho) \models \mathit{flatten}(\Psi')$.

   Let us then have some $C \in \mathcal{T}^*_{Cfg^*}$ such that C terminates in $(\mathcal{T}^*_{Cfg^*}, \Rightarrow_{\mathcal{S}^*})$, and a valuation $\rho : Var^* \to \mathcal{T}^*$ such that $(\mathcal{T}^*, C, \rho) \models \mathit{flatten}(\Psi)$. We have to prove that

   there exists some $C' \in \mathcal{T}^*_{Cfg^*}$ such that $C \Rightarrow^*_{\mathcal{S}^*} C'$ and $(\mathcal{T}^*, C', \rho) \models \mathit{flatten}(\Psi')$.

   We will proceed in five steps:

(a) We prove that there exists $c_1, \ldots, c_k$ such that $C = [c_1, \ldots, c_k]$.

(b) We prove that $c_1, \ldots, c_k$ are terminating.

(c) We find appropriate valuation $\rho_1 : \mathit{Var} \to \mathcal{T}$ and prove the premise of the assumption (i): that $(\mathcal{T}, c_1, \rho_1) \models \varphi_1 \wedge P$ and ... and $(\mathcal{T}, c_k, \rho_1) \models \varphi_k \wedge P$.

(d) From the assumption (i) we get the appropriate $c'_1, \ldots, c'_k$, as well as a valuation $\rho_2 : \mathit{Var} \to \mathcal{T}$ satisfying $\rho_1(v) = \rho_2(v)$ for any $v \in \mathit{Var} \setminus \vec{Y}$, and

(e) We let $C' := [c'_1, \ldots, c'_k]$ and prove that it is reachable from C, as well as that it satisfies the flattened Ψ' in ρ.

We have

$(\mathcal{T}^*, C, \rho) \models \mathit{flatten}(\Psi)$;

that is,

$(\mathcal{T}^*, C, \rho) \models \mathit{flatten}((\varphi_1, \ldots, \varphi_k) \wedge P)$;

which is by Lemma 20 equivalent to (ii)

there exist configurations $c_1, \ldots, c_k \in \mathcal{T}_{Cfg}$ such that $C = [c_1, \ldots, c_k]$, and there exists a $\mathcal{T}$-valuation ρ_0 satisfying $\rho_0(v) = \rho(v)$ for any $v \in \mathit{Var}$, such that for any $j \in \{1, \ldots, k\}$, $(\mathcal{T}, c_j, \rho_0) \models \varphi_j \wedge P$.

Now, let ρ_0 be such valuation, and let $c_1, \ldots, c_k$ be such configurations. We have just proved Item 1a. To prove Item 1b, saying that the configurations $c_1, \ldots, c_k$ are terminating, we simply use Lemma 18. To prove Item 1c, we let $\rho_1 := \rho_0$, and apply the assumption (ii). Now it follows that (iii)

there exist configurations $c'_1, \ldots, c'_k \in \mathcal{T}_{Cfg}$ such that $c_1 \Rightarrow^*_{\mathcal{S}} c'_1$ and ... and $c_k \Rightarrow^*_{\mathcal{S}} c'_k$, and there also exists an $\mathcal{T}$-valuation ρ_2 satisfying $\rho_1(v) = \rho_2(v)$ for any $v \in \mathit{Var} \setminus \vec{Y}$, and $(\mathcal{T}, c'_1, \rho_2) \models \varphi'_1 \wedge P'$ and ... and $(\mathcal{T}, c'_k, \rho_2) \models \varphi'_k \wedge P'$.

Let us have such configurations $c'_1, \ldots, c'_k$ and valuation ρ_2. We choose $C' := [c'_1, \ldots, c'_k]$, and it remains to be proven that

$C \Rightarrow^*_{\mathcal{S}^*} C'$ and $(\mathcal{T}^*, C', \rho) \models \mathit{flatten}(\Psi')$

(where ρ is the valuation that we started with). The part saying that $C \Rightarrow^*_{\mathcal{S}^*} C'$ holds follows by Lemma 19. The other part can be changed using Lemma 20 into

there exist configurations $c'_1, \ldots, c'_k \in \mathcal{T}_{Cfg}$ such that $C' = [c'_1, \ldots, c'_k]$ and there exists a $\mathcal{T}$-valuation ρ'_0 satisfying $\rho'_0(v) = \rho(v)$ for any $v \in \mathit{Var} \setminus \vec{Y}$ such that for any $j \in \{1, \ldots, k\}$, $(\mathcal{T}, c'_j, \rho'_0) \models \varphi'_j \wedge P$.

Since we have constructed C' as a list of smaller configurations, it remains to be proven that

> there exists a $\mathcal{T}$-valuation ρ'_0 satisfying $\rho'_0(v) = \rho(v)$ for any $v \in \mathit{Var} \setminus \vec{Y}$ such that for any $j \in \{1, \ldots, k\}$, $(\mathcal{T}, c'_j, \rho'_0) \models \varphi'_j \wedge P$.

Let us choose ρ'_0 defined by $\rho'_0(v) = \rho_2(v)$ for any $v \in \mathit{Var}$. We verify that $\rho'_0(v) = \rho_2(v) = \rho_1(v) = \rho_0(v) = \rho(v)$ for any $v \in \mathit{Var} \setminus \vec{Y}$, and the rest follows from the assumption (iii) by Lemma 4. This concludes the proof of the first implication.

2. For the opposite implication, we again assume that

$$\Psi \equiv (\varphi_1, \ldots, \varphi_k) \wedge P\,,$$

$\Psi' \equiv \exists \vec{Y}.\, (\varphi'_1, \ldots, \varphi'_k) \wedge P'$, $\varphi_j = \phi_j \wedge P_j$ and $\varphi'_j = \phi'_j \wedge P'_j$ for any $j \in \{1, \ldots, k\}$, and assume that

> $$\mathcal{S}^* \vDash_{\mathsf{RL}} \mathit{flatten}(\Psi) \Rightarrow^{\exists} \mathit{flatten}(\Psi')\,;$$

that is (i),

> for every $C \in \mathcal{T}^*_{\mathit{Cfg}^*}$ such that C terminates in $(\mathcal{T}_{\mathit{Cfg}^*}, \Rightarrow_{\mathcal{S}^*})$ and for any valuation $\rho : \mathit{Var}^* \to \mathcal{T}^*$ such that $(\mathcal{T}^*, C, \rho) \models \mathit{flatten}(\Psi)$, there exists some $C' \in \mathcal{T}^*_{\mathit{Cfg}^*}$ such that $C \Rightarrow^*_{\mathcal{S}^*} C'$ and $(\mathcal{T}^*, C', \rho) \models \mathit{flatten}(\Psi')$;

we have to prove that

> $$\mathcal{S} \vDash_{\mathsf{CRL}} \Psi \Rightarrow^{c\exists} \Psi'\,;$$

that is,

> for all configurations $c_1, \ldots, c_k \in \mathcal{T}_{\mathit{Cfg}}$ which terminate in $(\mathcal{T}_{\mathit{Cfg}}, \Rightarrow_{\mathcal{S}})$ and any $\mathcal{T}$-valuation ρ_1, whenever $(\mathcal{T}, c_1, \rho_1) \models \varphi_1 \wedge P$ and $\ldots$ and $(\mathcal{T}, c_k, \rho_1) \models \varphi_k \wedge P$, then there exist configurations $c'_1, \ldots, c'_k \in \mathcal{T}_{\mathit{Cfg}}$ such that $c_1 \Rightarrow^*_{\mathcal{S}} c'_1$ and $\ldots$ and $c_k \Rightarrow^*_{\mathcal{S}} c'_k$, and there also exists an $\mathcal{T}$-valuation ρ_2 satisfying $\rho_1(v) = \rho_2(v)$ for any $v \in \mathit{Var} \setminus \vec{Y}$, and $(\mathcal{T}, c'_1, \rho_2) \models \varphi'_1 \wedge P'$ and $\ldots$ and $(\mathcal{T}, c'_k, \rho_2) \models \varphi'_k \wedge P'$.

Let us then have such terminating configurations $c_1, \ldots, c_k \in \mathcal{T}_{\mathit{Cfg}}$ and such valuation $\rho_1 : \mathit{Var} \to \mathcal{T}$. We have to show that

> there exist configurations $c'_1, \ldots, c'_k \in \mathcal{T}_{\mathit{Cfg}}$ such that $c_1 \Rightarrow^*_{\mathcal{S}} c'_1$ and $\ldots$ and $c_k \Rightarrow^*_{\mathcal{S}} c'_k$, and there also exists an $\mathcal{T}$-valuation ρ_2 satisfying $\rho_1(v) = \rho_2(v)$ for any $v \in \mathit{Var} \setminus \vec{Y}$, and $(\mathcal{T}, c'_1, \rho_2) \models \psi_1 \wedge P'$ and $\ldots$ and $(\mathcal{T}, c'_k, \rho_2) \models \psi_k \wedge P'$.

We will proceed in the following steps.

(a) We prove the premise of (i) for $C := [c_1, \ldots, c_k]$, that is:
   i. $[c_1, \ldots, c_k]$ terminates in $(\mathcal{T}_{Cfg^*}, \Rightarrow_{\mathcal{S}^*})$; and
   ii. $(\mathcal{T}^*, [c_1, \ldots, c_k], \rho) \models \mathit{flatten}(\Psi)$ for some constructed valuation ρ.

(b) We "destruct" the obtained C' into $[c'_1, \ldots, c'_k]$;

(c) We prove the desired properties of c'_j from the properties of C'.

First, $[c_1, \ldots, c_k]$ is terminating by Lemma 18. Next, we have to show that

$$(\mathcal{T}^*, [c_1, \ldots, c_k], \rho) \models \mathit{flatten}((\varphi_1, \ldots, \varphi_k) \wedge P)\,;$$

where $\rho(v) = \rho_1(v)$ for any $v \in \mathit{Var}$ (and $\rho(v)$ has arbitrary value for v outside of Var). By Lemma 20, this is equivalent to showing that

there exist configurations $c_1, \ldots, c_k \in \mathcal{T}_{Cfg}$ such that $[c_1, \ldots, c_k] = [c_1, \ldots, c_k]$ and there exists a $\mathcal{T}$-valuation ρ_0 satisfying $\rho_0(v) = \rho(v)$ for any $v \in \mathit{Var}$ such that for any $j \in \{1, \ldots, k\}$, $(\mathcal{T}, c_j, \rho_0) \models \varphi_j \wedge P$.

We choose $c_j := c_j$ and $\rho_0 := \rho_1$; it remains to be proven that

$$(\mathcal{T}, c_j, \rho_1) \models \varphi_j \wedge P.$$

which holds by assumption. Now we have obtained the following:

there exists some $C' \in \mathcal{T}^*_{Cfg}$ such that $[c_1, \ldots, c_k] \Rightarrow^*_{\mathcal{S}^*} C'$ and $(\mathcal{T}^*, C', \rho) \models \mathit{flaten}(\Psi')$.

Now, by Lemma 15 (and using induction on the length of the sequence witnessing the reachability), we get some $c'_1, \ldots, c'_k \in \mathcal{T}_{Cfg}$ such that

$$[c_1, \ldots, c_k] \Rightarrow^*_{\mathcal{S}^*} [c'_1, \ldots, c'_k] \text{ and } (\mathcal{T}^*, [c'_1, \ldots, c'_k], \rho) \models \mathit{flaten}(\Psi').$$

Our goal is to prove that

there exist configurations $c'_1, \ldots, c'_k \in \mathcal{T}_{Cfg}$ such that $c_1 \Rightarrow^*_{\mathcal{S}} c'_1$ and $\ldots$ and $c_k \Rightarrow^*_{\mathcal{S}} c'_k$, and there also exists an $\mathcal{T}$-valuation ρ_2 satisfying $\rho_1(v) = \rho_2(v)$ for any $v \in \mathit{Var} \setminus \vec{Y}$, and $(\mathcal{T}, c'_1, \rho_2) \models \psi_1 \wedge P'$ and $\ldots$ and $(\mathcal{T}, c'_k, \rho_2) \models \psi_k \wedge P'$,

so we choose $c'_j := c'_j$ and have to prove that

$c_1 \Rightarrow^*_{\mathcal{S}} c'_1$ and $\ldots$ and $c_k \Rightarrow^*_{\mathcal{S}} c'_k$, and there also exists an $\mathcal{T}$-valuation ρ_2 satisfying $\rho_1(v) = \rho_2(v)$ for any $v \in \mathit{Var} \setminus \vec{Y}$, and $(\mathcal{T}, c'_1, \rho_2) \models \psi_1 \wedge P'$ and $\ldots$ and $(\mathcal{T}, c'_k, \rho_2) \models \psi_k \wedge P'$.

The first part follows from Lemma 19; it remains to be proven that

> there also exists an $\mathcal{T}$-valuation ρ_2 satisfying $\rho_1(v) = \rho_2(v)$ for any $v \in \mathit{Var} \setminus \vec{Y}$, and $(\mathcal{T}, c'_1, \rho_2) \models \psi_1 \wedge P'$ and $\ldots$ and $(\mathcal{T}, c'_k, \rho_2) \models \psi_k \wedge P'$.

and we already have

> $(\mathcal{T}^*, [c'_1, \ldots, c'_k], \rho) \models \mathit{flaten}(\Psi')$;

that is, by Lemma 20 we know that

> there exists a $\mathcal{T}$-valuation ρ_0 satisfying $\rho_0(v) = \rho(v)$ for any $v \in \mathit{Var} \setminus \vec{Y}$ such that for any $j \in \{1, \ldots, k\}$, $(\mathcal{T}, c'_j, \rho_0) \models \varphi'$.

Let ρ'_0 be such valuation. In the goal, we let $\rho_2(v) := \rho'_0(v)$ for any $v \in \mathit{Var}$; we then note that $\rho_2(v) = \rho'_0(v) = \rho(v) = \rho_1(v)$ for any $v \in \mathit{Var} \setminus \vec{Y}$ by definitions. The rest of the goal follows from the assumption by Lemma 4. This concludes the proof.

□

B.2 Proof of Theorem 2

Proof of Lemma 1. By induction on the structure of the CRL proof.

1. If the proof ends with *Reduce*, then we are done, since $\mathit{flatten}^{\exists}(\emptyset, \psi') = \emptyset$.
2. If the proof ends with *Reflexivity*, then we need to prove
$$\mathcal{S}^* \cup \mathit{flatten}^{\exists}(E, \psi), \emptyset \vdash_{\mathsf{RL}} \mathit{flatten}^{\exists}(\psi, \psi)$$
which we do by applying the Reflexivity proof rule.
3. If the proof ends with *Axiom*, then $\psi \in E$, and we have to prove that
$$\mathcal{S}^* \cup \mathit{flatten}'(E, \psi'), \mathit{flatten}'(C, \psi') \vdash_{\mathsf{RL}} \mathit{flatten}'(\psi, \psi') \,.$$
By applying the Axiom proof rule of RL, it is enough to show that
$$\mathit{flatten}'(\psi, \psi') \in \mathit{flatten}'(E, \psi') \,,$$
which follows from $\psi \in E$.
4. If the proof ends with *Case*, then we have
$$\mathcal{S}^* \cup \bar{E}, \bar{C} \vdash_{\mathsf{RL}} \mathit{flatten}^{\exists}((\varphi_1, \ldots, \varphi_{i-1}, \varphi_i, \varphi_{i+1}, \ldots, \varphi_k) \wedge P', \Psi')$$
and
$$\mathcal{S}^* \cup \bar{E}, \bar{C} \vdash_{\mathsf{RL}} \mathit{flatten}^{\exists}((\varphi_1, \ldots, \varphi_{i-1}, \psi_i, \varphi_{i+1}, \ldots, \varphi_k) \wedge P', \Psi')$$
as hypotheses, and we have to prove
$$\mathcal{S}^* \cup \bar{E}, \bar{C} \vdash_{\mathsf{RL}} \mathit{flatten}^{\exists}((\varphi_1, \ldots, \varphi_{i-1}, (\varphi_i \vee \psi_i), \varphi_{i+1}, \ldots, \varphi_k) \wedge P', \Psi') \,.$$

$$\text{Axiom}\ \frac{\varphi \Rightarrow^{\exists} \varphi' \in \mathcal{A}}{\mathcal{A}, C \vdash_{\mathsf{RL}} \varphi \Rightarrow^{\exists} \varphi'}$$

$$\text{Reflexivity}\ \frac{}{\mathcal{A}, \emptyset \vdash_{\mathsf{RL}} \varphi \Rightarrow^{\exists} \varphi}$$

$$\text{Transitivity}\ \frac{(\mathcal{T}, A), C \vdash_{\mathsf{RL}} \varphi_1 \Rightarrow^{+\exists} \varphi_2 \quad (\mathcal{T}, A \cup C), \emptyset \vdash_{\mathsf{RL}} \varphi_2 \Rightarrow^{\exists} \varphi_3}{(\mathcal{T}, A), C \vdash_{\mathsf{RL}} \varphi_1 \Rightarrow^{\exists} \varphi_3}$$

$$\text{Logic Framing}\ \frac{\mathcal{A}, C \vdash_{\mathsf{RL}} \varphi \Rightarrow^{\exists} \varphi' \quad \psi \text{ is a FOL formula}}{\mathcal{A}, C \vdash_{\mathsf{RL}} \varphi \wedge \psi \Rightarrow^{\exists} \varphi' \wedge \psi}$$

$$\text{Consequence}\ \frac{\mathcal{T} \models \varphi_1 \to \varphi_1' \quad (\mathcal{T}, A), C \vdash_{\mathsf{RL}} \varphi_1' \Rightarrow^{\exists} \varphi_2' \quad \mathcal{T} \models \varphi_2' \to \varphi_2}{(\mathcal{T}, A), C \vdash_{\mathsf{RL}} \varphi_1 \Rightarrow^{\exists} \varphi_2}$$

$$\text{Case Analysis}\ \frac{\mathcal{A}, C \vdash_{\mathsf{RL}} \varphi_1 \Rightarrow^{\exists} \varphi \quad \mathcal{A}, C \vdash_{\mathsf{RL}} \varphi_2 \Rightarrow^{\exists} \varphi}{\mathcal{A}, C \vdash_{\mathsf{RL}} \varphi_1 \vee \varphi_2 \Rightarrow^{\exists} \varphi}$$

$$\text{Abstraction}\ \frac{\mathcal{A}, C \vdash_{\mathsf{RL}} \varphi \Rightarrow^{\exists} \varphi' \quad X \notin FV(\varphi')}{\mathcal{A}, C \vdash_{\mathsf{RL}} \exists X.\, \varphi \Rightarrow^{\exists} \varphi'}$$

$$\text{Circularity}\ \frac{A, C \cup \{\varphi \Rightarrow^{\exists} \varphi'\} \vdash_{\mathsf{RL}} \varphi \Rightarrow^{\exists} \varphi'}{A, C \vdash_{\mathsf{RL}} \varphi \Rightarrow^{\exists} \varphi'}$$

Figure 2: One-path reachability-logic proof system. The use of $\Rightarrow^{+\exists}$ in a sequent means that it was derived without Reflexivity.

(where $\bar{E} = \mathit{flatten}^{\exists}(E, \psi')$ and $\bar{C} = \mathit{flatten}^{\exists}(C, \psi')$). After simplifications, we get

$$\mathcal{S}^* \cup \bar{E}, \bar{C} \vdash_{\mathsf{RL}} \mathit{mkList}(X_1, \ldots, X_k) \wedge \left(\bigwedge_{j=1}^{k} (\varphi_j^{\Box})[X_j/\Box] \right) \wedge P' \Rightarrow^{\exists} \mathit{flatten}(\Psi')$$

and

$$\mathcal{S}^* \cup \bar{E}, \bar{C} \vdash_{\mathsf{RL}} \mathit{mkList}(Y_1, \ldots, Y_k) \wedge \left(\bigwedge_{j=1, j \neq i}^{k} (\varphi_j^{\square})[Y_j/\square] \right) \wedge (\psi_i^{\square})[Y_i/\square] \wedge P' \Rightarrow^{\exists} \mathit{flatten}(\Psi')$$

as hypotheses, and have to prove

$$\mathcal{S}^* \cup \bar{E}, \bar{C} \vdash_{\mathsf{RL}} \mathit{mkList}(Z_1, \ldots, Z_k) \wedge \left(\bigwedge_{j=1, j \neq i}^{k} (\varphi_j^{\square})[Z_j/\square] \right) \wedge ((\varphi_i \vee \psi_i)^{\square})[Z_i/\square] \Rightarrow^{\exists} \mathit{flatten}(\Psi')$$

(where $X_1, \ldots, X_k, Y_1, \ldots, Y_k, Z_1, \ldots, Z_k$ are fresh variables). We first apply the Consequence RL rule to the goal to distribute the $\varphi_i \vee \psi_i$ disjunction to the top, changing the goal to

$$\mathcal{S}^* \cup \bar{E}, \bar{C} \vdash_{\mathsf{RL}} \left(\mathit{mkList}(Z_1, \ldots, Z_k) \wedge \left(\bigwedge_{j=1, j \neq i}^{k} (\varphi_j^{\square})[Z_j/\square] \right) \wedge (\varphi_i^{\square})[Z_i/\square] \right) \vee \left(\mathit{mkList}(Z_1, \ldots, Z_k) \wedge \left(\bigwedge_{j=1, j \neq i}^{k} (\varphi_j^{\square})[Z_j/\square] \right) \wedge (\psi_i^{\square})[Z_i/\square] \right) \Rightarrow^{\exists} \mathit{flatten}(\Psi') \,.$$

Now we apply the Case Analysis rule. Then we transform the hypotheses to the respective goals by existentially quantifying the X_js (and Y_js, respectively) in the hypotheses using the Abstraction RL rule, alpha-renaming (using the Consequence rule) the X_js (and Y_js, respectively) into Z_js, and stripping the existential quantifiers (using the Consequence rule, again), and we are done.

5. If the proof ends with *Step*, we can assume a structureless FOL formula P', a rule $\varphi \Rightarrow^{\exists} \varphi' \in S$ such that $\mathcal{T} \models \varphi_i \leftrightarrow \varphi \wedge P'$, and an induction hypothesis

$$(\mathcal{T}^*, S^* \cup \mathit{flatten}^{\exists}(C \cup E, \Psi')), \emptyset \vdash_{\mathsf{RL}} \mathit{flatten}([\varphi_1, \ldots, \varphi_{i-1}, \varphi' \wedge P', \varphi_{i+1}, \ldots, \varphi_k] \wedge P) \Rightarrow^{\exists} \mathit{flatten}(\Psi')$$

and have to construct

$$(\mathcal{T}^*, S^* \cup \mathit{flatten}^{\exists}(E, \Psi')), \mathit{flatten}^{\exists}(C, \Psi') \vdash_{\mathsf{RL}} \mathit{flatten}([\varphi_1, \ldots, \varphi_{i-1}, \varphi_i, \varphi_{i+1}, \ldots, \varphi_k] \wedge P) \Rightarrow^{\exists} \mathit{flatten}(\Psi') \,.$$

By definition of S^*, we also have

$$(\mathit{heat}(L, \varphi, R) \Rightarrow^{\exists} \mathit{heat}(L, \varphi', R)) \in S^* \,.$$

We apply the Transitivity rule with the second premise being our first inductive hypothesis, and it remains to prove the second premise, which is

$$(\mathcal{T}, S)^*, \mathit{flatten}^{\exists}(E, \psi'), \mathit{flatten}^{\exists}(C, \psi') \vdash_{\mathsf{RL}} \mathit{flatten}([\varphi_1, \ldots, \varphi_{i-1}, \varphi_i, \varphi_{i+1}, \ldots, \varphi_k] \wedge P) \Rightarrow^{\exists} \mathit{flatten}([\varphi_1, \ldots, \varphi_{i-1}, \varphi' \wedge P', \varphi_{i+1}, \ldots, \varphi_k] \wedge P) \,.$$

that is (after simplification, assuming a reasonable choice of fresh variables)

$$\begin{aligned}&(\mathcal{T}, S)^*, \mathit{flatten}^{\exists}(E, \psi'), \mathit{flatten}^{\exists}(C, \psi')\\ &\vdash_{\mathsf{RL}} \mathit{mkList}(X_1, \ldots, X_k) \wedge \left(\bigwedge_{j=1, j\neq i}^{k} (\varphi_j^{\Box})[X_j/\Box]\right) \wedge (\varphi_i^{\Box})[X_i/\Box] \wedge P\\ &\quad \Rightarrow^{\exists} \mathit{mkList}(X_1, \ldots, X_k) \wedge \left(\bigwedge_{j=1, j\neq i}^{k} (\varphi_j^{\Box})[X_j/\Box]\right) \wedge ((\varphi' \wedge P')^{\Box})[X_i/\Box] \wedge P\,.\end{aligned}$$

By Lemma 9, our assumption that $\mathcal{T} \models \varphi_i \leftrightarrow (\varphi \wedge P')$, and conservativeness, we can apply the Consequence rule, and the goal changes to

$$\begin{aligned}&(\mathcal{T}, S)^*, \mathit{flatten}^{\exists}(E, \psi'), \mathit{flatten}^{\exists}(C, \psi')\\ &\vdash_{\mathsf{RL}} \mathit{mkList}(X_1, \ldots, X_k) \wedge \left(\bigwedge_{j=1, j\neq i}^{k} (\varphi_j^{\Box})[X_j/\Box]\right) \wedge ((\varphi \wedge P')^{\Box})[X_i/\Box] \wedge P\\ &\quad \Rightarrow^{\exists} \mathit{mkList}(X_1, \ldots, X_k) \wedge \left(\bigwedge_{j=1, j\neq i}^{k} (\varphi_j^{\Box})[X_j/\Box]\right) \wedge ((\varphi' \wedge P')^{\Box})[X_i/\Box] \wedge P\,.\end{aligned}$$

We apply the Consequence rule again, changing the goal to

$$\begin{aligned}&(\mathcal{T}, S)^*, \mathit{flatten}^{\exists}(E, \psi'), \mathit{flatten}^{\exists}(C, \psi')\\ &\vdash_{\mathsf{RL}} (\varphi^{\Box})[X_i/\Box] \wedge (\mathit{mkList}(X_1, \ldots, X_k) \wedge \left(\bigwedge_{j=1, j\neq i}^{k} (\varphi_j^{\Box})[X_j/\Box]\right) \wedge ((P')^{\Box})[X_i/\Box] \wedge P)\\ &\quad \Rightarrow^{\exists} ((\varphi')^{\Box})[X_i/\Box] \wedge (\mathit{mkList}(X_1, \ldots, X_k) \wedge \left(\bigwedge_{j=1, j\neq i}^{k} (\varphi_j^{\Box})[X_j/\Box]\right) \wedge ((P')^{\Box})[X_i/\Box] \wedge P)\,.\end{aligned}$$

Now we apply Logic Framing to remove the structureless parts that are the same in both the left and right sides, resulting in the goal

$$\begin{aligned}&(\mathcal{T}, S)^*, \mathit{flatten}^{\exists}(E, \psi'), \mathit{flatten}^{\exists}(C, \psi')\\ &\vdash_{\mathsf{RL}} (\varphi^{\Box})[X_i/\Box] \wedge \mathit{mkList}(X_1, \ldots, X_k)\\ &\quad \Rightarrow^{\exists} ((\varphi')^{\Box})[X_i/\Box] \wedge \mathit{mkList}(X_1, \ldots, X_k)\,.\end{aligned}$$

Now, from the assumption that $\varphi \Rightarrow^{\exists} \varphi' \in S$ and the construction of S^* it follows that $\mathit{heat}(L, \varphi, R) \Rightarrow^{\exists} \mathit{heat}(L, \varphi', R) \in S^*$; and therefore $\mathit{cfgheat}(L, \phi, R) \wedge Q \Rightarrow^{\exists} \mathit{cfgheat}(L, \phi', R) \wedge Q' \in S^*$ where $\varphi \equiv \phi \wedge Q$ and $\varphi' \equiv \phi' \wedge Q'$. By semantic reasoning we can prove that

$$\begin{aligned}\mathcal{T}^* \models &((\varphi^{\Box})[X_i/\Box] \wedge \mathit{mkList}(X_1, \ldots, X_k))\\ \leftrightarrow &\mathit{cfgheat}(L, X_i, R)\\ &\wedge L = \mathit{mkList}(X_1, \ldots, X_{i-1})\\ &\wedge R = \mathit{mkList}(X_{i+1}, \ldots, X_k)\\ &\wedge (\phi^{\Box})[X_i/\Box] \wedge Q\end{aligned}$$

and that

$$\begin{aligned}\mathcal{T}^* \models & (((\varphi')^{\Box})[X_i/\Box] \wedge \mathit{mkList}(X_1, \ldots, X_k)) \\ \leftrightarrow & \mathit{cfgheat}(L, X_i, R) \\ & \wedge L = \mathit{mkList}(X_1, \ldots, X_{i-1}) \\ & \wedge R = \mathit{mkList}(X_{i+1}, \ldots, X_k) \\ & \wedge ((\phi')^{\Box})[X_i/\Box] \wedge Q' \,.\end{aligned}$$

Now we apply Consequence and subsequently strip the L,R equalities using Logic Framing, thus getting

$$\begin{aligned}& (\mathcal{T}, S)^*, \mathit{flatten}^{\exists}(E, \psi'), \mathit{flatten}^{\exists}(C, \psi') \\ & \vdash_{\mathsf{RL}} \mathit{cfgheat}(L, X_i, R) \wedge (\phi^{\Box})[X_i/\Box] \wedge Q \\ & \quad \Rightarrow^{\exists} \mathit{cfgheat}(L, X_i, R) \wedge ((\phi')^{\Box})[X_i/\Box] \wedge Q' \,.\end{aligned}$$

Now we use Consequence to expand $\phi^{\Box}$ and $(\phi')^{\Box}$ into equalities, perform the substitution, and use the equalities to replace the X_i subterm of *cfgheat* with ϕ and ϕ', respectively; this way the goal becomes

$$\begin{aligned}& (\mathcal{T}, S)^*, \mathit{flatten}^{\exists}(E, \psi'), \mathit{flatten}^{\exists}(C, \psi') \\ & \vdash_{\mathsf{RL}} \mathit{cfgheat}(L, \phi, R) \wedge Q \\ & \quad \Rightarrow^{\exists} \mathit{cfgheat}(L, \phi', R) \wedge Q' \,.\end{aligned}$$

We finish the proof of this case using the Axiom rule.

6. If the proof ends with *Circularity*, we can assume

$$(\mathcal{T}^*, S^* \cup \mathit{flatten}^{\exists}(E, \Psi')), \mathit{flatten}^{\exists}(C \cup \{\Psi\}, \Psi') \vdash_{\mathsf{RL}} \mathit{flatten}^{\exists}(\Psi, \Psi')$$

which simplifies to

$$(\mathcal{T}^*, S^* \cup \mathit{flatten}^{\exists}(E, \Psi')), \mathit{flatten}^{\exists}(C, \Psi') \cup \mathit{flatten}^{\exists}(\{\Psi\}, \Psi') \vdash_{\mathsf{RL}} \mathit{flatten}^{\exists}(\Psi, \Psi')$$

and have to prove

$$(\mathcal{T}^*, S^* \cup \mathit{flatten}^{\exists}(E, \Psi')), \mathit{flatten}^{\exists}(C, \Psi') \vdash_{\mathsf{RL}} \mathit{flatten}^{\exists}(\Psi, \Psi')$$

which follows from the assumption by Circularity.

7. If the proof ends with *Conseq*, we can assume

$$\mathcal{T}^* \models \mathit{flatten}(\Phi) \rightarrow \mathit{flatten}(\Phi')$$

and

$$(\mathcal{T}^*, S^* \cup \mathit{flatten}^{\exists}(E, \Psi')), \mathit{flatten}^{\exists}(C, \Psi') \vdash_{\mathsf{RL}} \mathit{flatten}^{\exists}(\Phi', \Psi') \,,$$

and have to prove

$$(\mathcal{T}^*, S^* \cup \mathit{flatten}^{\exists}(E, \Psi')), \mathit{flatten}^{\exists}(C, \Psi') \vdash_{\mathsf{RL}} \mathit{flatten}^{\exists}(\Phi, \Psi') \,.$$

The second assumption simplifies to

$$(\mathcal{T}^*, S^* \cup \mathit{flatten}^{\exists}(E, \Psi')), \mathit{flatten}^{\exists}(C, \Psi') \vdash_{\mathsf{RL}} \mathit{flatten}(\Phi') \Rightarrow^{\exists} \mathit{flatten}(\Psi')\,,$$

while the goal to

$$(\mathcal{T}^*, S^* \cup \mathit{flatten}^{\exists}(E, \Psi')), \mathit{flatten}^{\exists}(C, \Psi') \vdash_{\mathsf{RL}} \mathit{flatten}(\Phi) \Rightarrow^{\exists} \mathit{flatten}(\Psi')\,;$$

therefore, we can apply the *Consequence* rule.

8. If the proof ends with *Abstract*, we assume

$$X \notin FV(\Psi')$$

and

$$(\mathcal{T}^*, S^* \cup \mathit{flatten}^{\exists}(E, \Psi')), \mathit{flatten}^{\exists}(C, \Psi') \vdash_{\mathsf{RL}} \mathit{flatten}^{\exists}(\exists \vec{Y}.\, (\varphi_1, \ldots, \varphi_k) \wedge P, \Psi')$$

and have to prove that

$$(\mathcal{T}^*, S^* \cup \mathit{flatten}^{\exists}(E, \Psi')), \mathit{flatten}^{\exists}(C, \Psi') \vdash_{\mathsf{RL}} \mathit{flatten}^{\exists}(\exists X, \vec{Y}.\, (\varphi_1, \ldots, \varphi_k) \wedge P, \Psi')\,.$$

After simplifications, the second premise becomes

$$\begin{aligned}&(\mathcal{T}^*, S^* \cup \mathit{flatten}^{\exists}(E, \Psi')), \mathit{flatten}^{\exists}(C, \Psi') \vdash_{\mathsf{RL}}\\ &\exists \vec{Y}.\, (\mathit{mkList}(Z_1, \ldots, Z_k) \wedge (\varphi_1^{\square})[Z_1/\square] \wedge \ldots (\varphi_1^{\square})[Z_k/\square]) \wedge P)\\ &\Rightarrow^{\exists} \mathit{flatten}(\Psi')\,,\end{aligned}$$

while the goal becomes

$$\begin{aligned}&(\mathcal{T}^*, S^* \cup \mathit{flatten}^{\exists}(E, \Psi')), \mathit{flatten}^{\exists}(C, \Psi') \vdash_{\mathsf{RL}}\\ &\exists X. \exists \vec{Y}.\, (\mathit{mkList}(Z_1, \ldots, Z_k) \wedge (\varphi_1^{\square})[Z_1/\square] \wedge \ldots (\varphi_1^{\square})[Z_k/\square]) \wedge P)\,.\end{aligned}$$

We prove the goal using the Abstraction rule (note that $X \notin FV(\Psi')$ implies $X \notin FV(\mathit{flatten}(\Psi'))$ because we are free to choose the fresh variables inside the *flatten* such).

This concludes the proof. □

B.3 Completeness

To finish the proof of completeness, we lift the given oracle for a model $\mathcal{T}$ into an oracle for the model $\mathcal{T}^*$ by means of a reduction function Θ. We assume a framework similar to that of [32]. Specifically, we assume that the model $\mathcal{T}$ interprets

- the symbol α as an injective function from configurations into natural numbers;
- the symbols $<$,$+$,$-$,$*$ as the usual ordering, addition, subtraction, multiplication on natural numbers.

We represent Gödel's β function, defined by

$$\beta(x_1, x_2, x_3) = x_1 \bmod (1 + (x_3 + 1) \cdot x_2)\,,$$

as e.g., in [22] - as a formula

$$\beta_t(x_1, x_2, x_3, y) \equiv \exists w : \mathit{Nat}.\, x_1 = (1 + (x_3 + 1) * x_2) * w + y \wedge y \wedge 1 + (x_3 + 1) * x_2\,.$$

Theorem 5 (Oracle lifting)**.** *For every Σ-model $\mathcal{T}$ satisfying the conditions above there exists a function Θ from matching logic patterns over Σ^* to matching logic patterns over Σ such that $\mathcal{T} \models \Theta(\varphi) \iff \mathcal{T}^* \models \varphi$.*

Proof. Intuitively, we perform Gödelization of the formula $\varphi^{\Box}$, using Gödel's β_t predicate. The idea is that the only construct appearing in FOL formulas over Σ^* and not in FOL formulas over Σ is comparison of lists of configurations for equality, and we reduce this construct to equality of the corresponding elements of the list. For this purpose, we introduce a function $\mathit{lookup}(M, l, n, y)$ representing a predicate saying that the term l representing a list of configurations that has at position n the configuration y, where M is a mapping from list variables into the two variables representing a list in the Godel encoding. The predicate $\mathit{length}(M, l, n)$ holds if the length of l is exactly n.

We define functions *lookup* and *length* by mutual structural recursion on their second parameter:

- $\mathit{lookup}(M, l, n, y) = \beta_t(a_l, b_l, n, \alpha(y))$ if l is a variable of sort Cfg^* and $(l, a_l, b_l) \in M$;
- $\mathit{lookup}(\_, \mathit{cfgnil}, \_, \_) = \bot$.
- $\mathit{lookup}(M, \mathit{cfgconcat}(t_1, t_2), n, y) =$
  $\forall l : \mathit{Nat}.\, \mathit{length}(M, t_1, l) \to \mathit{ite}(n < l, \mathit{lookup}(M, t_1, n, y), \mathit{lookup}(M, t_2, n - l, y))$
- $\mathit{lookup}(M, \mathit{cfgheat}(l_1, c, l_2), n, y) =$

  $$\begin{aligned}&\forall l : \mathit{Nat}.\, \mathit{length}(M, t_1, l) \to \\ &\mathit{ite}(n < l, \mathit{lookup}(M, t_1, n, y), \mathit{ite}(n = l, y = c, \mathit{lookup}(M, t_2, n - l - 1, y)))\end{aligned}$$

- $\mathit{length}(M, l, n) = \forall i : \mathit{Nat}.\, (0 \le i < n) \leftrightarrow (\exists v : \mathit{Cfg}.\mathit{lookup}(M, l, i, v))$

The two functions have the following property.

Lemma 22. *Let t be a term of sort Cfg^* and ρ a $\mathcal{T}$-valuation. Let M be a relation such that for every free variable l of sort Cfg^* of t, there exist unique variables a_l, b_l of sort Nat such that $(l, a_l, b_l) \in M$ and $\rho(a_l), \rho(b_l)$ β-encodes $\rho(l)$. Then for any natural numer n, a configuration γ, a variable c of sort Cfg, and a variable i of sort Nat, if $\rho(c) = \gamma$, then $\rho(t)[n] = \gamma \iff M, \rho \models \mathit{lookup}(M, t, i, c)$. Also, for every natural numer n and a variable i of sort Nat, if $\rho(i) = n$, then we have that the length of $\rho(t)$ is exactly n iff $\mathcal{T}, \rho \models \mathit{length}(M, t, i)$.*

Proof. By mutual structural induction on t. □

Next, we define a function tr which performs basic recursion on a given FOL Σ^* formula, except that quantification over lists and equality of lists is handled as follows:

- $tr(M, \forall(l : \mathit{Cfg}^*).\, \varphi) = \forall a_l : \mathit{Nat}, b_l : \mathit{Nat}.\, \mathit{valid}(a_l, b_l) \to tr(M \cup \{(l, a_l, b_l)\}, \varphi)$ for some fresh a_l, b_l;
- $tr(M, l_1 = l_2) = \forall c : \mathit{Nat}, d : \mathit{Cfg}.\, (\mathit{lookup}(M, l_1, c, d) \leftrightarrow \mathit{lookup}(M, l_2, c, d))$

where $\mathit{valid}(a, b) = \exists l : \mathit{Nat}.\forall i : \mathit{Nat}.\, i < l \leftrightarrow \exists y : \mathit{Nat}.\, y < a \wedge \beta_p(a, b, i, y + 1)$. Intuitively, the *valid* formula used as a guard in the universal quantification case ensures that we consider only those a_l, b_l pairs which β-encode some sequence of natural numbers. Also, the definition of β guarantees that for fixed a, b, for large enough indices i, $\beta(a, b, i) = a$; the implementation of *valid* uses this property to guess the length l of the sequence being represented by $a, b.$,

$$\text{ConseqLocal}\ \frac{\mathcal{T} \models \varphi_i \to \varphi_i' \qquad (\mathcal{T}, S) \vdash_{\mathsf{CRL}} \exists \vec{X}.\, [\varphi_1, \ldots, \varphi_{i-1}, \varphi_i', \varphi_{i+1}, \ldots, \varphi_k] \wedge P \Downarrow_{C,E} \Psi'}{(\mathcal{T}, S) \vdash_{\mathsf{CRL}} \exists \vec{X}.\, [\varphi_1, \ldots, \varphi_{i-1}, \varphi_i, \varphi_{i+1}, \ldots, \varphi_k] \wedge P \Downarrow_{C,E} \Psi'}$$

$$\text{ConseqGlobal}\ \frac{\mathcal{T} \models P \to P' \qquad (\mathcal{T}, S) \vdash_{\mathsf{CRL}} \exists \vec{X}.\, [\varphi_1, \ldots, \varphi_{i-1}, \varphi, \varphi_{i+1}, \ldots, \varphi_k] \wedge P' \Downarrow_{C,E} \Psi'}{(\mathcal{T}, S) \vdash_{\mathsf{CRL}} \exists \vec{X}.\, [\varphi_1, \ldots, \varphi_{i-1}, \varphi, \varphi_{i+1}, \ldots, \varphi_k] \wedge P \Downarrow_{C,E} \Psi'}$$

$$\text{PropOut}\ \frac{(\mathcal{T}, S) \vdash_{\mathsf{CRL}} [\varphi_1, \ldots, \varphi_{i-1}, \varphi, \varphi_{i+1}, \ldots, \varphi_k] \wedge (P \wedge P') \Downarrow_{C,E} \Psi'}{(\mathcal{T}, S) \vdash_{\mathsf{CRL}} [\varphi_1, \ldots, \varphi_{i-1}, \varphi \wedge P', \varphi_{i+1}, \ldots, \varphi_k] \wedge P \Downarrow_{C,E} \Psi'}$$

$$\text{PropIn}\ \frac{(\mathcal{T}, S) \vdash_{\mathsf{CRL}} [\varphi_1, \ldots, \varphi_{i-1}, \varphi \wedge P', \varphi_{i+1}, \ldots, \varphi_k] \wedge P \Downarrow_{C,E} \Psi'}{(\mathcal{T}, S) \vdash_{\mathsf{CRL}} [\varphi_1, \ldots, \varphi_{i-1}, \varphi, \varphi_{i+1}, \ldots, \varphi_k] \wedge (P \wedge P') \Downarrow_{C,E} \Psi'}$$

$$\text{ExFalso}\ \frac{}{(\mathcal{T}, S) \vdash_{\mathsf{CRL}} [\varphi_1, \ldots, \ldots, \varphi_k] \wedge \mathit{false} \Downarrow_{C,E} \Psi'}$$

$$\text{GenWithCirc}\ \frac{\vec{X} = FV(\Phi) \setminus FV(\Psi') \qquad (\mathcal{T}, S) \vdash_{\mathsf{CRL}} \Phi \Downarrow_{C \cup \{\exists \vec{X}.\, \Phi\}, E} \Psi'}{(\mathcal{T}, S) \vdash_{\mathsf{CRL}} \Phi \Downarrow_{C,E} \Psi'}$$

Figure 3: Selected derived rules of CRL.

Let use consider the universal closure φ_c of $\varphi^{\square}$. Finally, we define $\Theta(\varphi) = tr(\varphi_c)$; the desired equivalence holds by properties of the Godel β_t predicate and properties of lists in the extended model $\mathcal{T}^*$.

□

B.4 Derived Rules

Lemma 23. *The rules of Figure 3 can be derived from the rules of Figure 1.*

Proof. We prove them one by one.

- ConseqLocal - follows from Conseq. We need to prove that $\mathcal{T} \models \varphi_i \to \varphi_i'$ implies

$$\begin{aligned} \mathcal{T}^* \models & \mathit{flatten}(\exists \vec{X}.\, [\varphi_1, \ldots, \varphi_{i-1}, \varphi_i, \varphi_{i+1}, \ldots, \varphi_k] \wedge P) \\ \to & \mathit{flatten}(\exists \vec{X}.\, [\varphi_1, \ldots, \varphi_{i-1}, \varphi_i', \varphi_{i+1}, \ldots, \varphi_k] \wedge P) \end{aligned}$$

which, after unfolding *flatten*, follows from Lemma 8.

- ConseqGlobal - follows from Conseq. We need to prove that $\mathcal{T} \models P \to P'$ implies

$$\mathcal{T}^* \models \mathit{flatten}(\exists \vec{X}. [\varphi_1, \ldots, \varphi_k] \wedge P) \to \mathit{flatten}(\exists \vec{X}. [\varphi_1, \ldots, \varphi_k] \wedge P')$$

  which follows immediatelly after unfolding *flatten*.

- PropIn, PropOut - follows from Conseq after unfolding *flatten* and using Lemma 7.
- ExFalso - follows from Reduce.
- GenWithCirc - follows from Conseq, followed by Circularity, followed by Abstract.

□

B.5 Relation to CHL

In order to tie (one-path) CRL to CHL, we first define an all-path variant of CRL.

Definition 10 (All-Path CRL semantics)**.** *A claim*

$$[\varphi_1, \ldots, \varphi_k] \wedge P \Rightarrow^{c\forall} \exists \vec{Y}. [\varphi'_1, \ldots, \varphi'_k] \wedge P'$$

is valid *in a reachability system $\mathcal{S} = (\mathcal{T}, S)$, written*

$$(\mathcal{T}, S) \vDash_{\mathsf{CRL}} [\varphi_1, \ldots, \varphi_k] \wedge P \Rightarrow^{c\forall} \exists \vec{Y}. [\varphi'_1, \ldots, \varphi'_k] \wedge P' \,,$$

iff for all configurations $\gamma_1, \ldots, \gamma_k \in \mathcal{T}_{Cfg}$ and any $\mathcal{T}$-valuation ρ, whenever $(\mathcal{T}, \gamma_1, \rho) \models \varphi_1 \wedge P$ and ... and $(\mathcal{T}, \gamma_k, \rho) \models \varphi_k \wedge P$, then for all complete $\Rightarrow_{\mathcal{S}}$-paths (that is, finite paths which cannot be extended further) $\pi_1, \ldots, \pi_k$ satisfying $\pi_i[0] = \gamma_i$ for any $i \in \{1, \ldots, k\}$ there exist natural numbers $j_1, \ldots, j_k$ such that $\gamma_1 \Rightarrow^*_{\mathcal{S}} \pi_1[j_1]$ and ... and $\gamma_k \Rightarrow^*_{\mathcal{S}} \pi_k[j_k]$, and $(\mathcal{T}, \pi_1[j_1], \rho') \models \varphi'_1 \wedge P'$ and ... and $(\mathcal{T}, \pi_k[j_k], \rho') \models \varphi'_k \wedge P'$ for some ρ' that agrees with ρ on all variables except $\vec{Y}$.

All-path CRL and one-path CRL have the same semantics on deterministic languages, assuming the RHS of the claim is terminal.

Lemma 24 (One-path / All-path CRL correspondence)**.** *Let Φ be a CLP and Φ' an ECLP such that Φ' is terminal (that is, configurations matching Φ' cannot take a step). Then $\mathcal{S} \vDash_{\mathsf{CRL}} \Phi \Rightarrow^{c\exists} \Phi'$ if and only if $\mathcal{S} \vDash_{\mathsf{CRL}} \Phi \Rightarrow^{c\forall} \Phi'$.*

Proof. Let $\mathcal{S} = (\mathcal{T}, S)$. Assume $\Phi \equiv [\varphi_1, \ldots, \varphi_k] \wedge P$ and $\Phi' \equiv \exists \vec{Y}. [\varphi'_1, \ldots, \varphi'_k] \wedge P'$. For the "if" part, assume $\mathcal{S} \vDash_{\mathsf{CRL}} \Phi \Rightarrow^{c\forall} \Phi'$. Assume some configurations $\gamma_1, \ldots, \gamma_k$ which terminate, and some valuation ρ satisfying $(\mathcal{T}, \gamma_1, \rho) \models \varphi_1 \wedge P$ and ... and $(\mathcal{T}, \gamma_k, \rho) \models \varphi_k \wedge P$. Let π_1 be some complete path $\gamma_1 = \gamma_1^1, \gamma_1^2, \ldots, \gamma_1^{l_1}$ and ... and pi_k be some complete path $\gamma_k = \gamma_k^1, \gamma_k^2, \ldots, \gamma_k^{l_k}$. Such paths exist, because γ_i (for $i \in \{1, \ldots, k\}$) are terminating - we can start with a path consisting of γ_j only and repeatedly extend the path until the last element has no successor. Now, since every ϕ'_i is terminal, all the j_is that exist by Definition 10 refer to the last configurations in the paths, because only those are terminal. That is, we have $j_i = l_i$ for any $i \in \{1, \ldots, k\}$. Therefore, we have some ρ' and $\gamma_1^{l_1}, \ldots, \gamma_k^{l_k}$ satisfying $(\mathcal{T}, \gamma_1^{l_1}, \rho') \models \varphi'_1 \wedge P$ and ... and $(\mathcal{T}, \gamma_k^{l_k}, \rho') \models \varphi'_k \wedge P$, as required by Definition 1.

For the "only if" part, assume $\mathcal{S} \vDash_{\mathsf{CRL}} \Phi \Rightarrow^{c\exists} \Phi'$. Assume some configurations $\gamma_1, \ldots, \gamma_k$ and some valuation ρ satisfying $(\mathcal{T}, \gamma_1, \rho) \models \varphi_1 \wedge P$ and ... and $(\mathcal{T}, \gamma_k, \rho) \models \varphi_k \wedge P$. Let $\pi_1, \ldots, \pi_k$ be

complete paths $\gamma_1^1, \gamma_1^2, \ldots, \gamma_1^{l_1}$ and ... and $\gamma_k^1, \gamma_k^2, \ldots, \gamma_k^{l_k}$ starting with $\gamma_1, \ldots, \gamma_k$, respectively. By Definition 1, there exist configurations $\gamma_1', \ldots, \gamma_k'$ such that $\gamma_1 \Rightarrow_{\mathcal{S}}^* \gamma_1'$ and ... and $\gamma_k \Rightarrow_{\mathcal{S}}^* \gamma_k'$, and there also exists a valuation ρ' that agrees with ρ on all variables outside $\vec{Y}$ such that $(\mathcal{T}, \gamma_1', \rho') \models \varphi_1' \wedge P$ and ... and $(\mathcal{T}, \gamma_k', \rho') \models \varphi_k' \wedge P$. Now by determinism and the fact that γ_i' are terminating (because φ_i' are terminating), we have $\gamma_i^{l_i} = \gamma_i'$. Thus, $(\mathcal{T}, \pi_1[j_1], \rho') \models \varphi_1' \wedge P'$ and ... and $(\mathcal{T}, \pi_k[j_k], \rho') \models \varphi_k' \wedge P'$, which concludes the proof. □

Definition 11. *Let L_{CHL} denote the CHL's imperative language and $\Sigma_{L_{CHL}}$ denote the matching logic signature of a RL-based formalization of L_{CHL} that that has a distinct constant symbol $sym_{var}(x)$ and a distinct variable $var_{inj}(x)$ in the signature for ever program variable x of L_{CHL}, and that subsumues the syntax of the codomain of CHL state. We define a function tr_{con} by*

$$tr_{con}(P, \sigma) \quad \equiv \quad \ll P \mid sym_{var}(x_1) \mapsto \sigma(x_1), \ldots, sym_{var}(x_n) \mapsto \sigma(x_n) \gg$$

and a function end_{con} by

$$end_{con}(P, \sigma) \quad \equiv \quad \ll \texttt{skip} \mid sym_{var}(x_1) \mapsto \sigma(x_1), \ldots, sym_{var}(x_n) \mapsto \sigma(x_n) \gg$$

(where $x_1, \ldots, x_n$ are program variables occurring in P).

Assumption 1. *We assume a sound formalization of L_{CHL} in the form of a reachability system $S_{L_{CHL}}$ that for any P, σ and any terminal γ' satisfies*

$$tr_{con}(P, \sigma) \Rightarrow_{S_{L_{CHL}}}^* \gamma' \iff (\sigma, S \Downarrow \sigma') \wedge (\gamma' = end_{con}(P, \sigma'))$$

(where $\Downarrow$ denotes the relation defined by the original big-step semantics of L_{CHL}).

Definition 12. *Let f be an injective function on program variables of L_{CHL}. We define a function tr_{sym} by*

$$tr_{sym}(P, f_{inj}) \equiv \ll P \mid sym_{var}(x_1) \mapsto var_{inj}(f(x_1)), \ldots, sym_{var}(x_n) \mapsto var_{inj}(f(x_n)) \gg$$

and a function end_{sym} by

$$end_{sym}(P, f_{inj}) \equiv \ll \texttt{skip} \mid sym_{var}(x_1) \mapsto var_{inj}(f(x_1)), \ldots, sym_{var}(x_n) \mapsto var_{inj}(f(x_n)) \gg$$

where P is a statement of L_{CHL} over variables $x_1, \ldots, x_n$.

Lemma 25 (Symbolic and concrete match)**.** *For any statement P, any injective function f on program variables of L_{CHL}, any finite map σ and any configuration γ, we have*

$$(\gamma, \rho) \models tr_{sym}(P, f)$$

iff

$$(\gamma = tr_{con}(P, \sigma[f])) \wedge (\forall j \in \{1, \ldots, n\}.\, \rho(var_{inj}(f(x_j))) = (\sigma[f])(x_j))\,;$$

similarly, we have

$$(\gamma, \rho) \models end_{sym}(P, f)$$

iff

$$(\gamma = end_{con}(P, \sigma[f])) \wedge (\forall j \in \{1, \ldots, n\}.\, \rho(var_{inj}(f(x_j))) = (\sigma[f])(x_j))$$

(where $x_1, \ldots, x_n$ are program variables occurring in P).

Proof. Follows by simple pattern matching. □

Lemma 26 (Static reasoning in CHL vs CRL)**.** *Let $\sigma_1, \ldots, \sigma_k$ be program states over $\vec{x}$. Let $r_i(x) = x_i$ for any $x \in \vec{x}$ and any $i \in \{1, \ldots, k\}$. Let P be a statement over variables $r_1(\vec{x}), \ldots, r_k(\vec{x})$. Let $\sigma = \biguplus_{1 \leq i \leq k} \sigma_i[r_i]$. Let ρ_σ be a matching logic valuation sending matching logic variables $var_{inj}(x)$ to $\sigma(x)$. Then:*

$$\begin{aligned} \sigma \vDash_{\mathsf{FOL}} \varphi &\iff \forall P. \forall i \in \{1, \ldots, k\}. (tr_{con}(P, \sigma_i[r_i]), \rho_\sigma) \models tr_{sym}(P, r_i) \wedge \varphi \\ &\iff \forall P. \forall i \in \{1, \ldots, k\}. (end_{con}(P, \sigma_i[r_i]), \rho_\sigma) \models end_{sym}(P, r_i) \wedge \varphi \end{aligned}$$

Proof. We show proof of the first equivalence only, since the second is similar. By Lemma 25 and properties of matching logic conjunction and structureless formulas, the RHS is equivalent to

- for any $j \in \{1, \ldots, n\}$, $\rho_\sigma(var_{inj}(r_i(x_j))) = (\sigma_i[r_i])(x_j)$; and
- $\rho_\sigma \models \varphi$.

The first item is always true: by the defining property of ρ_σ, it is enough to show that $\sigma(r_i(x_j)) = (\sigma_i[r_i])(x_j)$, which is trivially true. The second item is then equivalent to the LHS. □

Lemma 27 (All-path CRL vs CHL)**.** *Let P be a statement of L_{CHL} over variables $\vec{x} = x_1, \ldots, x_n$, and let Φ, Ψ be FOL formulas over variables $\vec{x}_1, \ldots, \vec{x}_k$. Let $\vec{Y} = var_{inj}(\vec{x}_1, \ldots, \vec{x}_k)$. Then $||\varphi||\ P\ ||\psi||$ if and only if*

$$[tr_{sym}(P, r_1), \ldots, tr_{sym}(P, r_k)] \wedge \varphi \Rightarrow^{c\forall} \exists \vec{Y}. [end_{sym}(P, r_1), \ldots, end_{sym}(P, r_k)] \wedge \psi\,.$$

Proof. The left side is true iff (by definition)

for every set of valuation pairs $\{(\sigma_1, \sigma_1'), \ldots, (\sigma_k, \sigma_k')\}$ satisfying

$$\left(\biguplus_{1 \leq i \leq k} \sigma_i[r_i] \vDash_{\mathsf{FOL}} \varphi \right)$$

and

$$\forall i \in \{1, \ldots, k\}. \sigma_i, P \Downarrow \sigma_i'$$

we also have

$$\left(\biguplus_{1 \leq i \leq k} \sigma_i'[r_i] \vDash_{\mathsf{FOL}} \psi \right)$$

which is (by Lemma 26 and Assumption 1) equivalent to

for every set of valuation pairs $\{(\sigma_1, \sigma_1'), \ldots, (\sigma_k, \sigma_k')\}$ satisfying

$$\forall P.\, \forall i \in \{1, \ldots, k\}.\, (tr_{con}(P, \sigma_i[r_i]), \rho_\sigma) \models tr_{sym}(P, r_i) \wedge \varphi$$

and

$$\forall i \in \{1, \ldots, k\}.\, tr_{con}(P, \sigma_i) \Rightarrow^*_{S_{L_{CHL}}} end_{con}(P, \sigma_i')$$

we also have

$$\forall P.\, \forall i \in \{1, \ldots, k\}.\, (end_{con}(P, \sigma_i[r_i]), \rho_{\sigma'}) \models end_{sym}(P, r_i) \wedge \psi$$

(where $\sigma = \biguplus_{1 \leq i \leq k} \sigma_i[r_i]$ and $\sigma' = \biguplus_{1 \leq i \leq k} \sigma_i'[r_i]$)

We want to show this to be equivalent with

for all configurations $\gamma_1, \ldots, \gamma_k \in \mathcal{T}_{Cfg}$ and any $\mathcal{T}$-valuation ρ, whenever $(\mathcal{T}, \gamma_1, \rho) \models tr_{sym}(P, r_1) \wedge \varphi$ and $\ldots$ and $(\mathcal{T}, \gamma_k, \rho) \models tr_{sym}(P, r_k) \wedge \varphi$, then for all complete $\Rightarrow_{\mathcal{S}}$-paths (that is, finite paths which cannot be extended further) $\pi_1, \ldots, \pi_k$ satisfying $\pi_i[0] = \gamma_i$ for any $i \in \{1, \ldots, k\}$ there exist natural numbers $j_1, \ldots, j_k$ such that $\gamma_1 \Rightarrow^*_{S_{L_{CHL}}} \pi_1[j_1]$ and $\ldots$ and $\gamma_k \Rightarrow^*_{S_{L_{CHL}}} \pi_k[j_k]$, and $(\mathcal{T}, \pi_1[j_1], \rho') \models end_{sym}(P, r_1) \wedge \psi$ and $\ldots$ and $(\mathcal{T}, \pi_k[j_k], \rho') \models end_{sym}(P, r_k) \wedge \psi$ for some ρ' that agrees with ρ on all variables except $\vec{Y}$.

1. For the top-down implication, assume such configurations and such complete paths. Let $l_1, \ldots, l_k$ denote the lengths of the paths $\pi_1, \ldots, \pi_k$. By Lemma 25, we have for every $j \in \{1, \ldots, k\}$:

   $$\gamma_j = tr_{con}(P, \sigma[r_j])$$

   and therefore

   $$tr_{con}(P, \sigma[r_j]) \Rightarrow^*_{S_{L_{CHL}}} \pi_j[l_j]\,.$$

   By Assumption 1, we have

   $$\pi_j[l_j] = end_{con}(P, \sigma[r_j])\,.$$

   Because we have

   $$\forall P.\, \forall i \in \{1, \ldots, k\}.\, (end_{con}(P, \sigma_i[r_i]), \rho_{\sigma'}) \models end_{sym}(P, r_i) \wedge \psi\,,$$

   it follows that

   $$(\pi_j[l_j], \rho_{\sigma'}) \models end_{sym}(P, r_i) \wedge \psi\,,$$

   which is what we needed to prove.

2. For the bottom-up implication, assume a set of valuation pairs $\{(\sigma_1, \sigma_1'), \ldots, (\sigma_k, \sigma_k')\}$ satisfying

   $$\forall P.\, \forall i \in \{1, \ldots, k\}.\, (tr_{con}(P, \sigma_i[r_i]), \rho_\sigma) \models tr_{sym}(P, r_i) \wedge \varphi$$

   and

   $$\forall i \in \{1, \ldots, k\}.\, tr_{con}(P, \sigma_i) \Rightarrow^*_{S_{L_{CHL}}} end_{con}(P, \sigma_i')$$

The second means we have for every i some complete trace π_i of length l_i starting in $tr_{con}(P, \sigma_i)$ and ending in $end_{con}(P, \sigma_i')$. We need to prove:

$$\forall P.\, \forall i \in \{1, \ldots, k\}.\, (end_{con}(P, \sigma_i[r_i]), \rho_{\sigma'}) \models end_{sym}(P, r_i) \land \psi$$

(where $\sigma = \biguplus_{1 \le i \le k} \sigma_i[r_i]$ and $\sigma' = \biguplus_{1 \le i \le k} \sigma_i'[r_i]$). Let P' be some program and let i be in $\{1, \ldots, k\}$. We need to show that

$$(end_{con}(P', \sigma_i[r_i]), \rho_{\sigma'}) \models end_{sym}(P', r_i) \land \psi\,.$$

By Lemma 25, it is enough to show that

$$(\forall j \in \{1, \ldots, n\}.\, \rho_{\sigma'}(var_{inj}(f(x_j))) = (\sigma'[f])(x_j))$$

(which holds by definition of $\rho_{\sigma'}$) and that

$$\rho_{\sigma'} \models \psi\,.$$

It is enough to show that $(\mathcal{T}, \pi_i[l_i], \rho') \models end_{sym}(P, r_i) \land \psi$ which follows from the assumptions by firstorder reasoning.

□

Now we can combine the above into the following result:

Theorem 6 (One-path CRL vs CHL)**.** *Let P be a* deterministic *statement of L_{CHL} over variables $\vec{x} = x_1, \ldots, x_n$, and let φ, ψ be FOL formulas over variables $\vec{x}_1, \ldots, \vec{x}_k$. Let $\vec{Y} = var_{inj}(\vec{x}_1, \ldots, \vec{x}_k)$. Then $||\varphi||\ P\ ||\psi||$ if and only if*

$$[tr_{sym}(P, r_1), \ldots, tr_{sym}(P, r_k)] \land \varphi \Rightarrow^{c\exists} \exists \vec{Y}.\, [end_{sym}(P, r_1), \ldots, end_{sym}(P, r_k)] \land \psi\,.$$

Proof. Use Lemma 27 and Lemma 24. □

Proof of Theorem 4. We define

$$tr(P, \varphi) \equiv [tr_{sym}(P, r_1), \ldots, tr_{sym}(P, r_k)] \land \varphi$$
$$end(P, \psi) \equiv \exists \vec{Y}.\, [end_{sym}(P, r_1), \ldots, end_{sym}(P, r_k)] \land \psi$$

(where P is a deterministic statement over $\vec{x} = x_1, \ldots, x_n$, and φ, ψ are FOL formulas over variables $\vec{x}_1, \ldots, \vec{x}_k$, and $\vec{Y} = var_{inj}(\vec{x}_1, \ldots, \vec{x}_k)$) and apply Theorem 6. □