Decomposing, Transforming and Composing Diagrams: The Joys of Modular Verification *

Luca de Alfaro Zohar Manna Henny Sipma

Email: luca@ic.eecs.berkeley.edu, {manna,sipma}@cs.stanford.edu

Abstract

The paper proposes a modular framework for the verification of temporal logic properties of systems based on the deductive transformation and composition of diagrams. The diagrams represent abstractions of the modules composing the system, together with information about the environment of the modules. The proof of a temporal specification is constructed with the help of diagram transformation and composition rules, which enable the gradual decomposition of the system into manageable modules, the study of the modules, and the final combination of the diagrams into a proof of the specification. We illustrate our methodology with the modular verification of a database demarcation protocol.

1 Introduction

One of the challenges of formal verification is to propose verification methodologies that are able to handle not only simple examples, but also realistic systems. Modular verification frameworks propose to address this issue by providing the means of decomposing the original system into modules of manageable size, studying each module separately, and composing the results into a proof of the correctness of the whole system. In this paper, we introduce a formal verification formalism based on the deductive transformation and composition of diagrams. The aim is to obtain a methodology that combines the modular approach to verification with the visual representation, the gradual proof construction and the provision of proof guidance made possible by diagrams.

The modular verification approach that we follow is based on the *assume-guarantee* paradigm of Abadi and Lamport [AL90]. In this paradigm, the system is partitioned into modules, which are studied with the help of assumptions about their environment. These assumptions, which must have the form of safety properties, usually specify restrictions to the possible state transitions of the module's environment. Once these assumptions are validated by an analysis of the other modules, the properties of the modules are combined into a correctness proof for the whole system.

The modular diagrams used in this paper belong to the family originated by the proposal of [MP94], later generalized by [BMS95]; another proposal using diagrams for the illustration of proofs is [Lam94]. In particular, this work represents a synthesis and an extension to modular verification of the proposals [dAM96, SUM96]. The diagrams provide a visual representation of the behavior of system modules and their environment: they consist in graphs whose vertices are labeled with assertions, and whose edges are labeled with transition formulas; additional components specify the progress properties that have been proved about them.

The proof that a system satisfies a temporal specification is constructed by applying a set of transformation rules to two initial diagrams, one representing the system and the other representing the negation of the specification. There are several classes of rules: *modular rules* split the system or one of its modules into submodules; *safety, progress* and *simplification* rules are used to study

*This research was supported in part by the National Science Foundation under grant CCR-95-27927, the Defense Advanced Research Projects Agency under NASA grant NAG2-892, ARO under grant DAAH04-95-1-0317, ARO under MURI grant DAAH04-96-1-0341, and by Army contract DABT63-96-C-0096 (DARPA).

the diagrams; a *composition operator* is used to compose diagrams for different modules into a single diagram. The aim of this process is to obtain a diagram that can be algorithmically shown to have empty language: by construction, this implies that all the behaviors of the original system satisfy the specification. The structure of the proof process, and of the composition operator, are reminiscent of the proposal of [GL94] for the modular verification of finite-state systems.

Integrating the modular framework with the use of diagrams makes it possible to generate the environment assumptions in a simple and often automatic way during the study of each module, and to discard them automatically during the composition of modules. The same operator also provides guidance for proof refinement in the case in which the environment assumptions cannot be validated; further guidance can be obtained from an automatic analysis of the diagrams, combining the insights of [dAM96, SUM96, dAKM97]. Our approach also leads to an increased flexibility of the process of modular analysis: the modules can be dynamically decomposed into submodules and recomposed during the construction of the proof, enabling a need-driven decomposition of modules. We illustrate our methodology with the modular verification of a protocol to enforce data constraints on distributed databases.

2 Preliminaries

Transition systems. Given a set $\mathcal{V}$ of variables, we denote by $\mathit{form}(\mathcal{V})$ the set of well-formed first-order formulas whose free variables are among $\mathcal{V}$. Our computational model is that of a transition system (TS) $S = (\mathcal{V}, \mathcal{T}, \Theta, J, C)$, where $\mathcal{V}$ is a set of typed state variables, $\mathcal{T}$ is a set of transitions, $\Theta \in \mathit{form}(\mathcal{V})$ is a satisfiable initial condition, $J \subseteq \mathcal{T}$ contains the just (weakly fair) transitions, and $C \subseteq \mathcal{T}$ contains the compassionate (strongly fair) transitions. A state s is a type-consistent interpretation of $\mathcal{V}$, and Σ denotes the set of all states. A transition $\tau \in \mathcal{T}$ is a function $\tau : \Sigma \mapsto 2^{\Sigma}$, and is represented by a transition formula $\rho_\tau \in \mathit{form}(\mathcal{V}, \mathcal{V}')$ that expresses the relation between the values of $\mathcal{V}$ in the current state and those in the next state, referred to by $\mathcal{V}' = \{x' \mid x \in \mathcal{V}\}$. Given a formula $\phi \in \mathit{form}(\mathcal{V})$, we denote with ϕ' the formula obtained by replacing each $x \in \mathcal{V}$ with x'. For $\tau \in \mathcal{T}$, the *enabling condition* $\mathit{En}(\tau)$ of τ is defined by $\exists \mathcal{V}' \,.\, \rho_\tau$. The set $\mathcal{T}$ must include the *idle transition* τ_{idle}, with transition relation $\rho_{\tau_{idle}} : \bigwedge_{x \in \mathcal{V}} (x = x')$.

The language $\mathcal{L}(S)$ of a transition system $S = (\mathcal{V}, \mathcal{T}, \Theta, J, C)$ consists of all the infinite sequences of states $s_0, s_1, s_2, \ldots \in \Sigma^\omega$ such that s_0 satisfies Θ, for every s_i, s_{i+1} there exists $\tau \in \mathcal{T}$ such that $(s_i, s_{i+1}) \models \rho_\tau$, and the fairness conditions are respected [MP91]. Note that $\mathcal{L}(S) \neq \emptyset$, since Θ is satisfiable and $\tau_{idle} \in \mathcal{T}$.

Specification language: linear-time temporal logic. The system specifications are written in the language TL_s consisting of first-order linear-time temporal logic formulas in which no temporal operator appears in the scope of a quantifier. The formulas of TL_s are thus obtained by combining first-order logic formulas by means of the future temporal operators $\bigcirc$ (next), $\Box$ (always), $\Diamond$ (eventually), $\mathcal{U}$ (until), and the corresponding past ones $\ominus$, $\boxminus$, $\diamondminus$ and $\mathcal{S}$ [MP91].

Example: demarcation protocol. We illustrate the proposed methodology by verifying a safety property of the protocol shown in Figure 1. The protocol is a more parallel version of the *demarcation protocol* presented in [BGM92], and is used to maintain linear arithmetic consistency constraints in a distributed database while minimizing communication costs between sites. In the example shown we have two sites, with data variables x and y, and we need to maintain the constraint $x \leq y$. The demarcation protocol shown maintains two safe limits x_l and y_l: Site 1 (Site 2) can modify x (y) independently as long as it stays below x_l (above y_l). When a site, e.g. Site 1, wishes to go beyond the safe limit, it asks permission from Site 2 to increase the limit x_l. If the new

local α, β : **channel of integer**

$Site_1$

local x, x_l : **integer where** $x = 0, x_l = 0$

$Requestor_1$

local x_g, x_w : **integer where** $x_g = 0$
local a_x : **boolean**

ℓ_0: **loop forever do**
  ℓ_1: **produce** x_w
  ℓ_2: $\left\langle\!\left\langle \textbf{if } x_w \le x_l \textbf{ then } (x, a_x) := (x_w, \mathrm{F}) \textbf{ else } a_x := \mathrm{T} \right\rangle\!\right\rangle$
  ℓ_3: **if** a_x **then**
    ℓ_4: $\alpha \Leftarrow (x_w - x_l)$
    ℓ_5: $\alpha \Rightarrow x_g$
    ℓ_6: $(x_l, x_g) := (x_l + x_g, 0)$
    ℓ_7: $\langle\!\langle \textbf{if } x_w \le x_l \textbf{ then } x := x_w \rangle\!\rangle$

||

$Grantor_1$

local x_r, x_p : **integer where** $x_r > 0, x_p = 0$

m_0: **loop forever do**
  m_1: $\beta \Rightarrow x_r$
  m_2: $\left\langle\!\left\langle \textbf{if } x_l - x_r \ge x \textbf{ then } (x_l, x_p) := (x_l - x_r, x_r) \textbf{ else } x_p := 0 \right\rangle\!\right\rangle$
  m_3: $\beta \Leftarrow x_p$

||

$Site_2$

local y, y_l : **integer where** $y = 0, y_l = 0$

$Requestor_2$

local y_g, y_w : **integer where** $y_g = 0$
local a_y : **boolean**

k_0: **loop forever do**
  k_1: **produce** y_w
  k_2: $\left\langle\!\left\langle \textbf{if } y_l \le y_w \textbf{ then } (y, a_y) := (y_w, \mathrm{F}) \textbf{ else } a_y := \mathrm{T} \right\rangle\!\right\rangle$
  k_3: **if** a_y **then**
    k_4: $\beta \Leftarrow (y_l - y_w)$
    k_5: $\beta \Rightarrow y_g$
    k_6: $(y_l, y_g) := (y_l - y_g, 0)$
    k_7: $\langle\!\langle \textbf{if } y_l \le y_w \textbf{ then } y := y_w \rangle\!\rangle$

||

$Grantor_2$

local y_r, y_p : **integer where** $y_r > 0, y_p = 0$

n_0: **loop forever do**
  n_1: $\alpha \Rightarrow y_r$
  n_2: $\left\langle\!\left\langle \textbf{if } y_l + y_r \le y \textbf{ then } (y_l, y_p) := (y_l + y_r, y_r) \textbf{ else } y_p := 0 \right\rangle\!\right\rangle$
  n_3: $\alpha \Leftarrow y_p$

Figure 1: Demarcation protocol

limit is still below y, Site 2 will grant the request (and update its own limit y_l); otherwise it will deny it. The conversion of this program to a fair transition system S_d is straightforward (see also [MP91]). Each statement gives rise to a transition; the statements enclosed by angle brackets are interpreted as atomic statements. For example, the transition relation for the statement labeled by ℓ_2 is $(x_w \le x_l \wedge x' = x_w \wedge a'_x = \mathit{false}) \ \vee \ (x_w > x_l \wedge a'_x = \mathit{true})$. We will verify that the protocol satisfies the temporal specification $\Box(x \le x_l \le y_l \le y)$.

3 Modular Diagrams

The diagrams used in this paper are derived from the *fairness diagrams* of [dAM96]. A diagram $A = (\mathcal{U}, \mathcal{V}, V, F, E, \mu, \theta, \nu, tr, \mathcal{F})$ for a TS S consists of the following components:

1. A subset $\mathcal{U} \subseteq \mathcal{T}$ indicating the transitions that are studied by diagram A.
2. A set $\mathcal{V}$ of typed variables.
3. A set V of vertices, and two disjoint sets F, E of edges, with associated functions $tl, hd : F \cup E \mapsto V$ that give the source (tail) $tl(e)$ and the target (head) $hd(e)$ of each edge $e \in F \cup E$. The edges in F, called *system edges*, represent subsets of the set $\mathcal{U}$ of transitions; the edges in E, called *environment edges*, represent the transitions in $\mathcal{T} - \mathcal{U}$.

4. Two mappings $\mu, \theta : V \mapsto \mathit{form}(\mathcal{V})$ that associate with each vertex $v \in V$ a formula $\mu(v)$ (resp. $\theta(v)$) denoting the states (resp. initial states) associated with v.
5. A mapping $\nu : F \cup E \mapsto \mathit{form}(\mathcal{V}, \mathcal{V}')$, which associates with each edge $e \in F \cup E$ a transition formula $\nu(e)$.
6. A mapping $\mathit{tr} : F \cup E \mapsto 2^{\mathcal{T}}$, labeling each edge $e \in F \cup E$ with the subset of transitions it represents. We require that $\mathit{tr}(e) \subseteq \mathcal{U}$ for $e \in F$, and $\mathit{tr}(e) = \mathcal{T} - \mathcal{U}$ for $e \in E$.
7. A *fairness set* $\mathcal{F}$, consisting of triples of the form (J, C, G), where $J, C : V \mapsto \mathit{form}(\mathcal{V})$ and $G : F \mapsto 2^{\mathcal{U}}$. For $e \in F$ we require that $G(e) \subseteq \mathit{tr}(e)$. Each triple, called a *fairness constraint*, is used to represent a fairness property of the diagram, as will be explained below.

Given $u, v \in V$ and a set H of edges, we denote by $H(u) = \{e \in H \mid \mathit{tl}(e) = u\}$ and $H(u, v) = \{e \in H \mid \mathit{tl}(e) = u \wedge \mathit{hd}(e) = v\}$ the set of edges from u, and from u to v, respectively.

A *location* of a diagram is a pair $(v, s) : v \in V, s \models \mu(v)$ composed of a vertex and of a corresponding state. A *run* of a diagram is an infinite sequence of locations (v_0, s_0), (v_1, s_1), (v_2, s_2), …, such that $s_0 \models \theta(v_0)$, and for all $i \geq 0$ there is $e \in F(v_i, v_{i+1}) \cup E(v_i, v_{i+1})$ such that $(s_i, s_{i+1}) \models \nu(e)$. Given an edge $e \in E \cup F$, we denote by $\vec{\nu}(e)$ the formula $\nu(e) \wedge \mu(\mathit{tl}(e)) \wedge \mu'(\mathit{hd}(e))$, which denotes the state transitions that can occur when edge e is traversed. The computations of a diagram are defined in terms of its accepting runs.

Definition 1 (accepting runs) A run $\sigma : (v_0, s_0), (v_1, s_1), (v_2, s_2), \ldots$ of a diagram A is an *accepting run* if, for each constraint $(J, C, G) \in \mathcal{F}$, if there is $n \geq 0$ such that $s_i \models J(v_i)$ for all $i \geq n$ and $s_i \models C(v_i)$ for infinitely many $i \geq 0$, then there are infinitely many $j \geq 0$ such that $\exists \tau \in G(v_j, v_{j+1}) . (s_j, s_{j+1}) \models \rho_\tau$. If $\sigma : (v_0, s_0), (v_1, s_1), (v_2, s_2), \ldots$ is an accepting run of A, the sequence of states $s_0, s_1, s_2, \ldots$ is a *computation* of A. We denote by $\mathit{Runs}(A)$, $\mathcal{L}(A)$ the sets of accepting runs and computations of A, respectively. ∎

4 The Structure of Proofs

Given a TS S and a specification $\phi \in TL_s$, a proof of $S \models \phi$ consists of a directed acyclic graph (dag) whose nodes are labeled with diagrams. The diagrams labeling the roots of the dag are obtained from S and ϕ; the diagrams labeling the non-root nodes of the dag are obtained using diagram transformation rules that will be discussed in detail in the next section.

Definition 2 (proof dag) Given a transition system $S = (\mathcal{V}, \mathcal{T}, \Theta, J, C)$ and a formula $\phi \in TL_s$, a proof dag for S and ϕ is a directed acyclic graph (dag) D, in which every node $d \in D$ is labeled with a diagram A_d. Dag D has two root nodes, labeled with diagrams $A\langle\neg\phi, S\rangle$ and $A\langle S\rangle$; every non-root node $d \in D$ has either one or two parents. If d has a single parent d', then A_d has been obtained from $A_{d'}$ by one application of a transformation rule; if d has two parents d_0, d_1, then $A_d = A_{d_0} \otimes A_{d_1}$, where $\otimes$ is the diagram composition operator. ∎

The diagrams labeling the roots of the proof dag are constructed as follows.

Construction 1 ($A\langle S\rangle$) The diagram $A\langle S\rangle = (\mathcal{U}, \mathcal{V}, \{v_0\}, \{f_0\}, \emptyset, \mu, \theta, \nu, \mathit{tr}, \emptyset)$ consists of a single vertex v_0 with one self-loop system edge f_0. The vertex and edge labels are defined by $\mu(v_0) = \mathit{true}$, $\theta(v_0) = \Theta$, $\nu(f_0) = \mathit{true}$, $\mathit{tr}(e_0) = \mathcal{T}$. ∎

Construction 2 ($A\langle\neg\phi, S\rangle$) Let $N_{\neg\phi}$ be the (first-order) Streett automaton that accepts all the state sequences that do not satisfy ϕ [Saf88]. The automaton $N_{\neg\phi}$ consists of the components

$(\mathcal{V}, (V, F), \mu, Q, \mathcal{A})$, where $\mathcal{V}$, μ are as in a diagram; (V, F) is a directed graph; $Q \subseteq V$ is the set of *initial vertices*, and $\mathcal{A}$, called the *acceptance list*, is a set of pairs $(P, R) : P, R \subseteq V$.

From $N_{\neg\phi}$ we construct $A\langle\neg\phi, S\rangle = (\mathcal{V}, V, F, \emptyset, \mu, \theta, \nu, tr, \mathcal{F})$, where $\nu(e) = true$, $tr(e) = \mathcal{T}$ for $e \in F$, $\theta(v) = \Theta$ for $v \in Q$, and $\theta(v) = false$ for $v \in V - Q$. For each $(P, R) \in \mathcal{A}$, there is a constraint $(J, C, G) \in \mathcal{F}$ defined by $J(v) = true$, if $v \in V - P$ then $C(v) = true$ else $C(v) = false$, and if $hd(e) \in R$ then $G(e) = \mathcal{T}$ else $G(e) = \emptyset$, for all $v \in V$ and $e \in F$. ∎

Due to the granularity of the acceptance condition of $A\langle\neg\phi, S\rangle$, we do not necessarily have $\mathcal{L}(N_{\neg\phi}) = \mathcal{L}(A\langle\neg\phi, S\rangle)$. However, the following lemma suffices for our purposes.

Lemma 1 $\mathcal{L}(S) \cap \mathcal{L}(A\langle\neg\phi, S\rangle) = \{\omega \in \mathcal{L}(S) \mid \omega \not\models \phi\}$.

The aim of the construction of the proof dag is to obtain a leaf labeled with a diagram that can be algorithmically shown to have empty language, indicating that all computations of S satisfy ϕ. The algorithm for language emptiness relies on a terminating proof procedure $\vdash$ for the first-order language used in the specification and in the labels of the diagram. Given a first-order formula ψ, we write $\vdash \psi$, $\not\vdash \psi$ depending on whether $\vdash$ terminates with or without a proof of ψ, respectively. We assume that the procedure $\vdash$ is at least able to prove the validity of all substitution instances of propositional tautologies. The check for language emptiness is based on an analysis of the strongly connected components (SCCs) of the graph underlying the diagram.

Definition 3 (persistent and non-persistent SCCs) Given a diagram A, we say that a strongly connected component U of the graph $(V_A, F_A \cup E_A)$ is *non-persistent* if there is a constraint $(J, C, G) \in \mathcal{F}_A$ such that the following conditions hold:

$$\forall v \in U \,.\, \vdash \mu(v) \to J(v) \qquad \exists v \in U \,.\, \vdash \mu(v) \to C(v) \qquad \forall e \in F \,.\, G(e) = \emptyset \,.$$

Otherwise, we say that U is persistent. ∎

The set of vertices that appear infinitely often along any accepting run of a diagram must be a persistent SCC. Thus, we have the following criterion for language emptiness [dAM96, SUM96].

Theorem 1 *If all the SCCs of a diagram A are non-persistent, then $\mathcal{L}(A) = \emptyset$.*

Definition 4 (proof of $S \models \phi$) A dag D for S and ϕ is a proof of $S \models \phi$ if there is a leaf $l \in D$ such that $\mathcal{U}_{A_l} = \mathcal{T}$ and Theorem 1 can show that $\mathcal{L}(A(l)) = \emptyset$. ∎

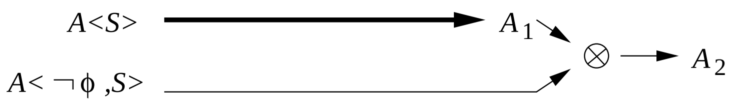
Since diagram $A\langle S\rangle$ can be obtained from $A\langle\neg\phi, S\rangle$ by means of transformation rules, it would be sufficient to consider dags with only the root $A\langle\neg\phi, S\rangle$. However, it is convenient to have $A\langle S\rangle$ as alternate root, since it enables the study of the system starting from a simpler diagram that is independent of the specification. The soundness of the methodology is expressed by the theorem below, discussed in the appendix.

Theorem 2 *If there is a dag D which is a proof of $S \models \phi$, then $S \models \phi$ holds.*

Examples of Proof Dags

In the following, we present some examples of proof dags for a transition system S and a specification ϕ. Each dag exemplifies a different proof style, underlining the flexibility that the modular decomposition and composition lend to the methodology. In the dags, thin lines indicate the application of at most one transformation rule, thick lines indicate the application of zero or more rules; we assume that the rightmost diagram has $\mathcal{U}$ =]calt and can be shown to have empty language using Theorem 1.

4.1 The system-analysis style.

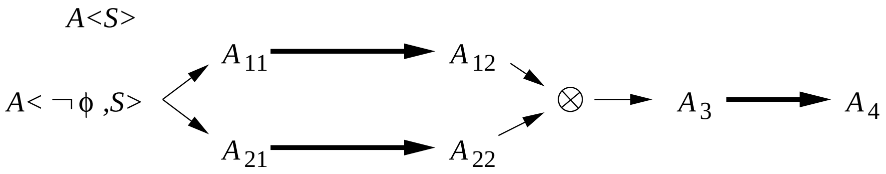


This first dag corresponds to the proof style proposed in [dAM96, dAKM97]. In this style, the root diagram $A\langle S\rangle$ is studied by means of successive transformations, until the product of the resulting diagram with $A\langle\neg\phi, S\rangle$ has empty language.

4.2 The deductive model-checking style.

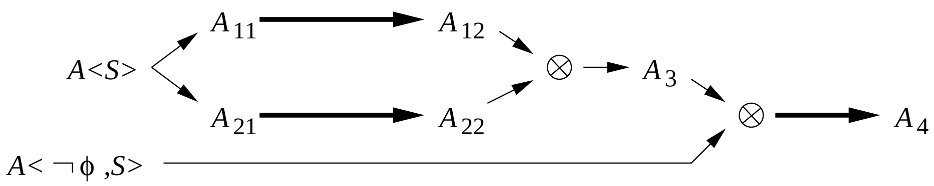
This dag corresponds to the proof style proposed in [SUM96]. In this style, the diagram $A\langle\neg\phi, S\rangle$ is studied by means of successive transformations until it can be shown to have empty language.

4.3 The modular deductive model-checking style.



This dag corresponds to a modular version of the proof style proposed in [SUM96]. In this style, the diagram $A\langle\neg\phi, S\rangle$ is decomposed into the two diagrams A_{11}, A_{21}, each corresponding to a module of S. These diagrams are then first studied in isolation (leading to A_{12}, A_{22}), and then composed into a joint diagram A_3. This diagram is then subject to transformations, until the resulting diagram A_4 can be shown to have empty language.

4.4 A system-analysis and model-checking modular style.



This dag illustrates a proof in which the diagram $A\langle S\rangle$ for the TS S is first studied by means of modular decomposition, leading to the two diagrams A_{11} and A_{21}. These diagrams are studied, leading to diagrams A_{21} and A_{22}, which are then combined into diagram A_3. Then, diagram A_3 is combined with $A\langle\neg\phi, S\rangle$, and from this point on the proof proceeds as in the deductive model-checking approach. Diagrams for more complex proof structures can be drawn in similar ways.

5 Diagram Transformations

The transformation rules enable the analysis of the temporal properties of diagrams and the modular decomposition of the system. There are four classes of rules: *modular rules* split modules into submodules, *safety* and *progress rules* study the safety and progress properties of diagrams, and *simplification* rules simplify the structure of diagrams. While some rules can be applied without preconditions, others require the proof of first-order verification conditions. The diagram composition operator, based on a special type of synchronous composition, is used to combine diagrams for different submodules of a system. It can also be used for proof reuse and backtracking, but such uses are beyond the scope of this paper. Due to space constraints, we present in detail only one rule for modular decomposition and the composition operator $\otimes$; discussing only the general features of the other classes of rules. The definitions of additional rules can be found in the appendix.

Modular rules. The rule below is used to perform modular decomposition: given a diagram A that studies $\mathcal{U} \subseteq \mathcal{T}$ and given $\mathcal{R} \subseteq \mathcal{U}$, the rule produces a diagram for the subset $\mathcal{R}$ of transitions. An additional rule, not discussed in this paper, enables the introduction of auxiliary variables.

Module split rule. Given a proper non-empty subset $\mathcal{R} \subset \mathcal{U}$ of the transitions of diagram A, the diagram *module-split*$(A, \mathcal{R})$ is obtained by restricting $\mathcal{U}$ to $\mathcal{R}$, and by splitting each edge $e \in F$ into two new edges $e_1 \in F$, $e_2 \in E$ with labels $tr(e_1) = tr(e) \cap \mathcal{R}$, $tr(e_2) = \mathcal{T} - \mathcal{R}$, $\nu(e_1) = \nu(e_2) = \nu(e)$. Each time an edge e is split into e_1 and e_2, all the constraints (J, C, G) such that $G(e) \not\subseteq \mathcal{R}$ are dropped from the fairness set of the diagram.

Safety rules. There are two types of safety rules: the rules that strengthen the vertex or edge labels, and the rules that split vertices and edges into new vertices and edges. The application of rules of the first type corresponds to the proof of inductive invariants, obtained for example using the methods of [BBM97]. Strengthening the vertex labels automatically strengthens the environment by restricting the admissible environment transitions. An additional rule enables the arbitrary strengthening (and pruning) of environment edges, in preparation to the application of progress rules.

Progress rules. The progress rules derive new progress properties about the diagrams, and represent them as fairness constraints that are then added to the diagrams; they are obtained by adapting the rules of [dAM96] to the notation used in this paper. Since the rules do not distinguish between system and environment edges, a fairness constraint can be proved only if it is compatible with the environment: hence the need for the previously mentioned rule to strengthen the environment.

Simplification rules. The simplification rules enable the weakening of the labels of vertices and system edges, and the merging of sets of vertices and edges. The purpose of these rules is to summarize and simplify portions of diagrams that have already been analyzed.

In order to preserve soundness, the simplification rules never weaken the labels of environment edges. To understand the reason, consider as an example an application of a safety rule that over-strengthens the vertex labels, causing the pruning of a portion of the diagram that in fact was reachable by a computation of the TS S. In this case, the application of the rule generates an environment that is too restrictive to account for all the possible transitions of the other modules. Since the environment assumptions will not be weakened, when a descendant of the diagram is composed with the diagrams for the other modules, some transitions of these diagrams will not satisfy the environment assumptions, and the composition operator $\otimes$ will create edges to a "sink" vertex. The computations of S that are excluded by the diagram strengthening will then be represented by transitions to the sink vertex, thus preserving computations of S. A similar argument can be made for rules that add progress constraints by relying on an improperly strengthened environment.

Composition operator. Given two diagrams A, B, the composition operator $\otimes$ combines them into a diagram $C = A \otimes B$ that corresponds to the synchronous composition of A and B, with one additional "sink" node. All the state changes of A (resp. B) that are due to transitions in $\mathcal{U}_A - \mathcal{U}_B$ (resp. $\mathcal{U}_B - \mathcal{U}_A$) but are not accounted for by the environment of B (resp. A) give rise to transitions leading to the sink node, preserving the computations that would otherwise be excluded. Diagram $C = A \otimes B$ is defined as follows:

1. $\mathcal{U}_C = \mathcal{U}_A \cup \mathcal{U}_B$.
2. $V_C = \{v^*\} \cup \{(u,v) \mid u \in V_A \wedge v \in V_B \wedge \not\vdash \neg[\mu(u) \wedge \mu(v)]\}$, where v^* is a new vertex used as "sink" for the computations of one diagram that do not match with the environment of the other diagram. For all $(u,v) \in V_C$, we let $\mu_C(u,v) = \mu_A(u) \wedge \mu_B(v)$ and $\theta_C(u,v) = \theta_A(u) \wedge \theta_B(v)$; for the sink node, $\mu_C(v^*) = \mathit{true}$, $\theta_C(v^*) = \mathit{false}$.
3. Initially, the sets F_C and E_C contain only two edges f^*, e^*, respectively, that are self-loops for the sink vertex v^*. These edges are labeled by $\nu_C(e^*) = \nu_C(f^*) = \mathit{true}$, $\mathit{tr}(f^*) = \mathcal{U}_C$, $\mathit{tr}(e^*) = \mathcal{T} - \mathcal{U}_C$. Then, additional edges are added in two steps.
   (a) First we add the "good" edges, representing synchronous steps of the two diagrams. Consider all pairs of vertices $(u,v), (u',v') \in V_C$. For each $e \in F_A(u,u') \cup E_A(u,u')$ and $f \in F_B(v,v') \cup E_B(v,v')$, if $\mathit{tr}_A(e) \cap \mathit{tr}_B(f) \neq \emptyset$ and $\not\vdash \neg[\nu_A(e) \wedge \nu_B(f)]$, we construct an edge g from (u,v) to (u',v'), labeled by $\nu_C(g) = \nu_A(e) \wedge \nu_B(f)$ and $\mathit{tr}_C(g) = \mathit{tr}_A(e) \cap \mathit{tr}_B(f)$. If $e \in E_A$ and $f \in E_B$, we insert g in E_C; otherwise we insert it in F_C.
   (b) Next, if an edge allows transitions violating the environment of the other diagram, we construct an edge to the sink node.
   Consider all vertices $(u,v) \in V_C$. For each edge $e \in F_A(u)$ such that $\mathit{tr}_A(e) \not\subseteq \mathcal{U}_B$, we check whether $\vdash [\vec{\nu}_A(e) \wedge \mu_B(v)] \rightarrow \bigvee_{f \in E_B(v)} \vec{\nu}_B(f)$. If the implication cannot be proved, we add to F_C an edge g from (u,v) to v^*, labeled by $\nu_C(g) = \vec{\nu}_A(e) \wedge \neg \bigvee_{f \in E_B(v)} \vec{\nu}_B(f)$, $\mathit{tr}_C(g) = \mathit{tr}_A(e) - \mathcal{U}_B$. We then perform the symmetrical check for each $f \in F_B(v)$, adding an edge from v to v^* if the corresponding implication cannot be proved.
4. For each constraint $(J, C, G) \in \mathcal{F}_A$, we insert in $\mathcal{F}_C$ the constraint $(\hat{J}, \hat{C}, \hat{G})$ defined by:
   (a) $\hat{J}(v^*) = \hat{C}(v^*) = \mathit{false}$, and for all $u \in V_A$, $v \in V_B$, $\hat{J}(u,v) = J(u)$, $\hat{C}(u,v) = C(u)$
   (b) For each edge $g \in F_C$ generated from $e \in F_A$ as in Step 1, let $\hat{G}(g) = G(e) \cap \mathit{tr}(g)$.

   We then perform the symmetrical step for each constraint $(J, C, G) \in \mathcal{F}_B$.

6 Demarcation Protocol: Diagram Proof

To prove that the demarcation protocol shown in Figure 1 satisfies $\phi : \Box(x \leq x_l \leq y_l \leq y)$ we construct the two roots $A\langle S_d \rangle$ and $A\langle \neg\phi, S_d \rangle$ of the proof dag. The communication structure of the program suggests to decompose the system into two modules consisting of $\mathit{Requestor}_1\mathit{Grantor}_2$ (R_1G_2) and $\mathit{Requestor}_2\mathit{Grantor}_1$ (R_2G_1); the proof will follow the style depicted in Section 4.4.

First, we apply the decomposition rule twice to $A\langle S_d \rangle$, once with $\mathcal{R} = \{\ell_{0\ldots7}, n_{0\ldots3}, \mathit{idle}\}$, and another time with $\mathcal{R}$ equal to the remaining transitions, obtaining diagrams A_{11} and A_{21}.

We want to show that the module corresponding to diagram A_{11} maintains $x \leq x_l$. The only statement that may potentially violate this is ℓ_6: thus, we require $\mathit{at\_}\ell_6 \rightarrow x_g \geq 0$. Note that n_2 cannot violate $x \leq x_l$ due to its guard and the atomicity of the grouped statement. Performing backpropagation based on $at\_\ell_6 \to x_g \geq 0$, following the methods described in [BBM97], and splitting and strengthening the vertices accordingly, we obtain the diagram shown in Figure 2.

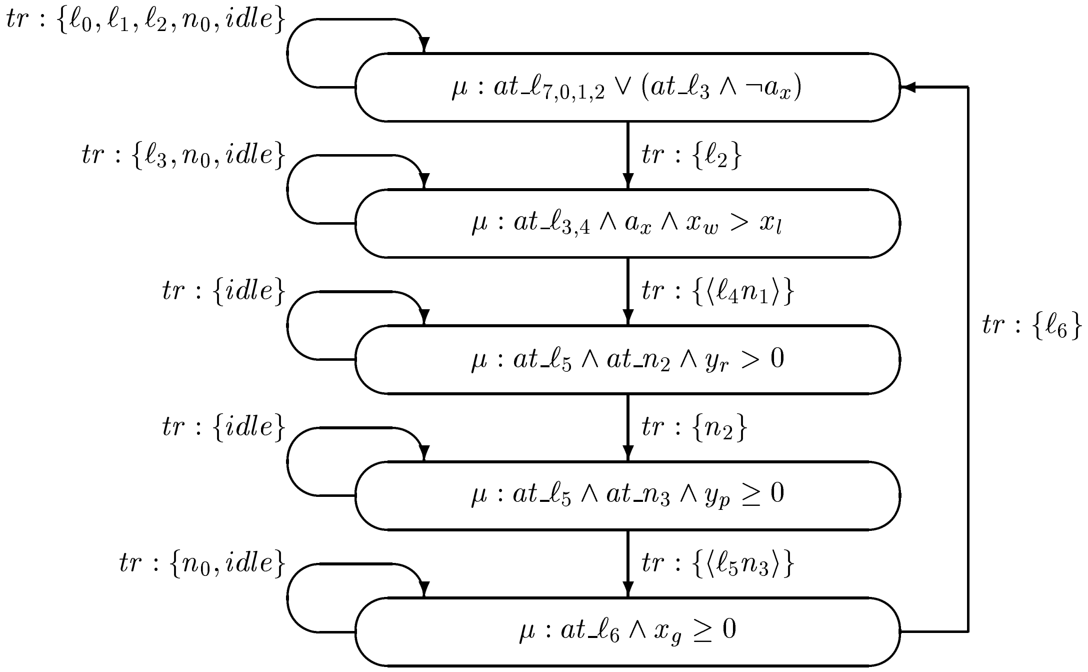


Figure 2: Diagram for module $R_1 G_2$

To reduce cluttering, environment edges have been omitted. However, each vertex has an environment edge connecting it to itself, labeled by $tr : \{k_{0\ldots7}, m_{0\ldots3}\}$. The ν-labeling of the system edges consists of the disjunction of the transition relations associated with the transitions in tr; the ν labeling of the environment edges is identically equal to *true*. The labeling θ is equal to Θ on the first vertex of the diagram, and is *false* on the other ones. The fairness set $\mathcal{F}$ is empty.

The environment assumptions are encoded by the $\vec{\nu}$-formulas associated with the environment edges: for example, the assumption corresponding to the second vertex is given by

$$(at\_\ell_{3,4} \wedge a_x \to x_w > x_l) \quad \to \quad (at'\_\ell_{3,4} \wedge a'_x \to x'_w > x'_l) \, .$$

The second property we want to analyze for this module is $\Box(x_l \leq y_l)$. This may be falsified by statements n_2 and again ℓ_6. Thus we require $at\_n_2 \to y_r \geq 0$ and $at\_\ell_6 \to x_l + x_g \leq y_l$. These assertions can be added to the vertices of the diagram shown in Figure 2 by application of the safety rules without the need for further splitting of vertices; the diagram obtained corresponds to diagram A_{12} of Section 4.4.

After performing a similar analysis for the module $R_2 G_1$, obtaining diagram A_{22}, we compose the two resulting diagrams obtaining diagram $A_3 = A_{12} \otimes A_{22}$. The edges leading to the "sink" vertex of A_3 are labeled by $false$, so that the sink vertex is unreachable and can be eliminated. Note that only the transition relation of the environment edge of the second vertex of A_{12} (resp. A_{22}) has to be validated against its environment, since the assertions labeling the other vertices only refer to variables local to $R_1 G_2$ (resp. $R_2 G_1$), and thus cannot be falsified by other modules; on the other hand, x_l and y_l are not local to the modules.

A final composition of this diagram with $A\langle\neg\phi\rangle$ leads then to diagram A_4, which can be shown to have empty language by Theorem 1.

By comparison, the proof of this property using non-modular verification diagrams requires the input of a diagram with many more vertices, labeled with complex assertions.

7 Completeness Results and Guidance

The completeness of the methodology presented in this paper follows easily from an analysis of the completeness proof for the diagram transformation methodology of [dAM96]. In fact, the completeness proof for the methodology of [dAM96] is based on the construction of a chain of diagram transformations that proves the property; the construction of this chain can be easily recast as the construction of a non-modular proof dag. We can thus state the following theorem.

Theorem 3 (completeness) *For a TS S and $\phi \in TL_s$, if $S \models \phi$ then there is a dag D that is a proof of $S \models \phi$.*

Guidance in constructing the proof can be obtained in several ways, depending on the position of the diagram A under study in the proof dag. If the only root ancestor of A is $A\langle S\rangle$, it is possible to obtain guidance by computing the product $A \otimes A\langle\neg\phi, S\rangle$: the projection of the persistent SCCs of the product back onto A gives an indication of the components of A that have to be shown either unreachable or non-persistent [dAM96, dAKM97].

If the root ancestors of A include $A\langle\neg\phi, S\rangle$, then attention can be focused on the persistent SCSs of A; all of these have to be shown to be unreachable, or have to be broken or shown non-persistent by the addition of progress constraints [SUM96].

When analyzing a diagram for a subsystem, a $\otimes$ product with a diagram for the rest of the system will tell whether the environment assumptions are satisfied by the rest of the system. In case sink edges are created, the assumption were too restrictive: the source vertex and the transition relation of the sink edge give information about the improper restriction of the environment assumptions.

References

[AL90] M. Abadi and L. Lamport. Composing specifications. In *Stepwise Refinement of Distributed Systems: Models, Formalism, Correctness*, volume 430 of *LNCS*, pages 1–41. Springer-Verlag, 1990.

[BBM97] N.S. Bjørner, A. Browne, and Z. Manna. Automatic generation of invariants and intermediate assertions. *Theor. Comp. Sci.*, 1997. To appear.

[BGM92] D. Barbara and H. Garcia-Molina. The demarcation protocol: A technique for maintaining linear arithmetic constraints in distributed database systems. In *Advances in Database Technology - 3rd Int. Conf. on Extending Database Technology*, pages 373–388. Springer-Verlag, 1992.

[BMS95] A. Browne, Z. Manna, and H.B. Sipma. Generalized verification diagrams. In *15th Conference on the Foundations of Software Technology and Theoretical Computer Science*, volume 1026 of *LNCS*, pages 484–498, 1995.

[dAKM97] L. de Alfaro, A. Kapur, and Z. Manna. Hybrid diagrams: A deductive-algorithmic approach to hybrid system verification. In *14th Symposium on Theoretical Aspects of Computer Science*, February 1997.

[dAM96] L. de Alfaro and Z. Manna. Temporal verification by diagram transformations. In *Proc. 8^{th} Intl. Conference on Computer Aided Verification*, volume 1102 of *LNCS*, pages 287–299, July 1996.

[GL94] O. Grumberg and D.E. Long. Model checking and modular verification. *ACM Trans. Prog. Lang. Sys.*, 16(3):843–871, May 1994.

[Lam94] L. Lamport. TLA in pictures. Technical Report 127, Digital Equipment Corporation, Systems Research Center, September 1994.

[MP91] Z. Manna and A. Pnueli. *The Temporal Logic of Reactive and Concurrent Systems: Specification.* Springer-Verlag, New York, 1991.

[MP94] Z. Manna and A. Pnueli. Temporal verification diagrams. In *Proc. Int. Symp. on Theoretical Aspects of Computer Software*, volume 789 of *LNCS*, pages 726–765. Springer-Verlag, 1994.

[Saf88] S. Safra. On the complexity of ω-automata. In *Proc. 29th IEEE Symp. Found. of Comp. Sci.*, pages 319–327, 1988. An extended version to appear in *J. Comp. Sys. Sci.*

[SUM96] H.B. Sipma, T.E. Uribe, and Z. Manna. Deductive model checking. In *Proc.* 8^{th} *Intl. Conference on Computer Aided Verification*, volume 1102 of *LNCS*, pages 208–219. Springer-Verlag, 1996.

Appendix

A Diagram Transformation Rules

Below, we present several diagram transformation rules. We have included all the safety rules needed to complete the proof of the demarcation protocol. Moreover, we present the only (elementary) progress rule original from this paper, and we indicate how to adapt the rules of [dAM96] to the present notation.

In the presentation of the transformation rules, we let $A = (\mathcal{U},\ \mathcal{V},\ V,\ F,\ E,\ \mu,\ \theta,\ \nu,\ tr,\ \mathcal{F})$ be the diagram to be transformed. Moreover, we adopt the following convention to describe the modification of diagram components. Given a mapping $\eta : C \mapsto D$ and two elements a, b with $b \in D$, we define the result of *updating* η by $\eta(a) := b$ to be the mapping $\eta^* : C \cup \{a\} \mapsto D$ such that $\eta^*(a) = b$, and $\eta^*(x) = \eta(x)$ for $x \in C - \{a\}$.

Once a transformation rule has been applied, yielding a diagram B, we remove all the vertices $v \in V^*$ such that $\mu_B(v) \equiv \mathit{false}$, along with all the edges originating from and leading to these vertices. Next, we remove all edges $e \in E_B \cup F_B$ such that either $\nu_B(e) \equiv \mathit{false}$ or $tr_B(e) = \emptyset$. Last, we remove from the diagram all the vertices that are not reachable in the graph $(V_B, E_B \cup F_B)$ from some vertex u with $\theta_B(u) \wedge \mu_B(u)$ satisfiable.

A.1 Safety Rules

Rule 1 (vertex strengthen) Let $V = \{v_1, v_2, \ldots, v_m\}$, and consider a list of formulas $\phi_1, \phi_2, \ldots, \phi_m \in \mathit{form}(\mathcal{V})$. Assume that the implication $(\theta(v_i) \wedge \mu(v_i)) \rightarrow \phi_i$ and $(\phi_i \wedge \vec{\nu}(e)) \rightarrow \phi'_j$ holds for all $1 \leq i, j \leq m$ and all $e \in F(v_i, v_j)$. Then, the diagram $\mathit{v\text{-}streng}(A, \phi_1, \ldots, \phi_m)$ is obtained by updating $\mu(v_i) := \mu(v_i) \wedge \phi_i$, $\theta(v_i) := \theta(v_i) \wedge \phi_i$, for $1 \leq i \leq m$. ∎

Rule 2 (system-edge strengthen) Given $e \in F$, the diagram $\mathit{se\text{-}streng}(A, e)$ is obtained by updating $\nu(e) := \vec{\nu}(e) \wedge \bigvee_{\tau \in tr(e)} \rho_\tau$. ∎

The following rule enables the arbitrary strengthening (and pruning) of environment edges.

Rule 3 (environment-edge strengthen) Given $e \in F$ and $\phi \in \mathit{form}(\mathcal{V}, \mathcal{V}')$, the diagram $\mathit{ee\text{-}streng}(A, e, \phi)$ is obtained by updating $\nu(e) := \nu(e) \wedge \phi$. ∎

Rule 4 (drop edge label) Given $e \in F$ and $\tau \in tr(e)$, assume that $\rho_\tau \wedge \nu(e) \equiv \mathit{false}$. Then, the diagram $\mathit{drop\text{-}label}(A, e, \tau)$ is obtained by updating $tr(e) := tr(e) - \{\tau\}$, and by updating every constraint (J, C, G) by $G(e) := G(e) - \{\tau\}$. ∎

Rule 5 (vertex split) Given $v \in V$ and $\phi \in \mathit{form}(\mathcal{V})$, the diagram $\mathit{v\text{-}split}(A, v, \phi)$ is obtained by replacing the vertex v of A with two new vertices v_1, v_2, with labels $\mu(v_1) = \mu(v) \wedge \phi$, $\mu(v_2) = \mu(v) \wedge \neg\phi$. Then, we replace each edge $e \in E(v, v) \cup F(v, v)$ with four new edges $\{e_{ij}\}_{i,j \in \{1,2\}}$, where edge e_{ij} leads from v_i to v_j, and we add these edges to E if $e \in E$ and to F is $e \in F$. We replace each edge $e \in E \cup F$ leading to v with two new edges e_1, e_2 leading to v_1, v_2 respectively. We replace each edge $e \in E(v) \cup F(v)$ with two new edges e_1, e_2 departing from v_1, v_2 respectively. The new edges have the same labels as the edges they replace. Finally, we update each constraint (J, C, G) by $J(v_1) := J(v)$, $J(v_2) := J(v)$, $C(v_1) := C(v)$, $C(v_2) := C(v)$. If edge f has replaced edge e, we update $G(f) := G(e)$. ∎

A.2 Progress Rules

Our first progress rule is used to add fairness constraints that represent the fairness of the transitions of the original transition system.

Rule 6 (add constraints from TS) Let $\tau \in \mathcal{U}$ be a just (resp. compassionate) transition of the TS S. The diagram *add-constraint*(A, τ) is obtained from A by adding the constraint (J, C, G) defined by $J(v) = En(\tau)$ (resp. $J(v) = true$) if $\tau \in J$ (resp. $\tau \in C$), $C(v) = En(\tau)$, and $G(e) = tr(e) \cap \{\tau\}$ for all $v \in V$ and $e \in F$. ∎

Once the fairness of the transitions is represented by fairness constraints, other progress rules are used to reason on these constraints and obtain new constraints, that are added to the diagram. These rules are obtained by adapting the progress rules of [dAM96] to the notation of this paper. As an example, we give below the adapted version of the rule that derives new constraints from the concatenation of already existing ones.

Rule 7 (concatenation of constraints) Given a diagram A and a constraint (J, C, G), assume that there is a constraint $(J_0, C_0, G_0) \in \mathcal{F}$ with $(J_0(u) \wedge \mu(u)) \rightarrow J(u)$ for all $u \in V$ and a ranking function δ such that the following implications are valid.

1. For all $u, v \in V$ and $e \in F(u, v) \cup E(u, v)$,

$$\begin{array}{rcl} J(u) \wedge \vec{\nu}(e) & \rightarrow & \neg J'(v) \vee \delta(u) \geq \delta'(v) \vee \bigvee_{\tau \in G(e)} \rho_\tau \\ J(u) \wedge \vec{\nu}(e) \wedge \bigvee_{\tau \in G_0(e)} \rho_\tau & \rightarrow & \neg J'(v) \vee \delta(u) > \delta'(v) \vee \bigvee_{\tau \in G(e)} \rho_\tau \ , \end{array}$$

   with the convention that $G(e) = \emptyset$ for $e \in E$.

2. Either $(C(u) \wedge \mu(u)) \rightarrow C_0(u)$ for all $u \in V$, or there is $(J_1, C_1, G_1) \in \mathcal{F}$ such that for all $u, v \in V$, the implications

$$\begin{array}{rcl} J(u) \wedge \mu(u) & \rightarrow & J_1(u) \vee J_0(u) \\ C(u) \wedge \mu(u) & \rightarrow & C_1(u) \vee C_0(u) \\ J(u) \wedge \vec{\nu}(e) \wedge \bigvee_{\tau \in G_1(e)} \rho_\tau & \rightarrow & \neg J'(v) \vee C_0'(v) \vee \bigvee_{\tau \in G(e)} \rho_\tau \end{array}$$

   are valid for all $e \in F(u, v) \cup E(u, v)$.

Then, diagram *conc-cons*$(A, (J, C, G))$ is obtained by adding the constraint (J, C, G) to the fairness set of diagram A. ∎

B Soundness of the Methodology

Differently from [dAM96], the diagram transformations we present do not preserve language containment, since it is possible to strengthen arbitrarily the transition formulas labeling the environment edges. The lemma below provides a characterization of the language of diagrams in the proof dag. In the lemma, we denote by $\hat{\sigma}$ the sequence of states corresponding to a run σ (which can be accepting or not accepting). The lemma can be proved by induction on the structure of the dag, using the definitions of the transformation rules and the composition operator; the proof has been omitted due to its length.

Lemma 2 *Given a proof dag D for a TS S, let D_0 be the set of nodes that have $A\langle S\rangle$ as unique root ancestor, and let D_1 be the set of nodes that have $A\langle\neg\phi, S\rangle$ among the root ancestors. Then, for $A \in \{A_d \mid d \in D_0\}$ (resp. $A \in \{A_d \mid d \in D_1\}$) there is a function Λ_0^A (resp. Λ_1^A) that maps the accepting runs $\{\sigma \in Runs(A\langle S\rangle) \mid \hat{\sigma} \in \mathcal{L}(S)\}$ (resp. $\{\sigma \in Runs(A\langle\neg\phi, S\rangle) \mid \hat{\sigma} \in \mathcal{L}(S)\}$) into runs (not necessarily accepting) or run prefixes of A. For $i = 0, 1$, these functions have the following properties:*

Faithfulness. *For $\sigma' = \Lambda_i^A(\sigma)$, if σ' is infinite then $\hat{\sigma}' = \hat{\sigma}$, otherwise $\hat{\sigma}'$ is a prefix of $\hat{\sigma}$.*

Termination. *If $\sigma' = \Lambda_i^A(\sigma)$ is finite, let $(v, s), (v', s')$ be the first step of σ that does not have a correspondent in σ', and let (u, s) be the last location of σ'. Then, for all $\tau \in \mathcal{T}$, if $(s, s') \models \tau$ then $\tau \notin \mathcal{U}_A$, so that the missing step in A is the responsibility of the environment. Moreover, $(s, s') \not\models \nu_A(e)$ for all $e \in E_A(u)$, indicating that σ' cannot be extended due to the (excessive) strengthening of the environment of A.*

Progress. *If $\sigma' = \Lambda_i^A(\sigma)$ is infinite, then σ' is also accepting.* ∎

As a consequence of this lemma, we have the following result.

Theorem 4 *In a proof dag D, consider a node d labeled by a diagram A with $\mathcal{U}_A = \mathcal{T}$. If d has $A\langle\mathcal{T}\rangle$ as only root ancestor, then $\mathcal{L}(S) \subseteq \mathcal{L}(A)$. If $A\langle\neg\phi, S\rangle$ is among the root ancestors of d, then $\{\omega \in \mathcal{L}(S) \mid \omega \not\models \phi\} \subseteq \mathcal{L}(A)$.*

By the above theorem, if dag D contains a node labeled with a diagram A such that $\mathcal{U}_A = \mathcal{T}$ and $\mathcal{L}(A) = \emptyset$, then $\{\omega \in \mathcal{L}(S) \mid \omega \not\models \phi\} = \emptyset$. Theorem 2, expressing the soundness of the methodology, follows as a consequence.