Reducing Behavioural to Structural Properties of Programs with Procedures

Dilian Gurov[1]* and Marieke Huisman[2]**

[1] Royal Institute of Technology, Stockholm, Sweden
[2] INRIA Sophia Antipolis, France

Abstract There is an intimate link between program structure and behaviour. Exploiting this link to phrase program correctness problems in terms of the structural properties of a program graph rather than in terms of its unfoldings is a useful strategy for making analyses more tractable. The present paper presents a characterisation of behavioural program properties through sets of structural properties by means of a translation. The characterisation is given in the context of a program model based on control flow graphs of sequential programs with procedures, and properties expressed in a fragment of the modal μ-calculus with boxes and greatest fixed-points only. The property translation is based on a tableau construction that conceptually amounts to symbolic execution of the behavioural formula, collecting structural constraints along the way. By keeping track of the subformulae that have been examined, recursion in the structural constraints can be identified and captured by fixed-point formulae. The tableau construction terminates, and the characterisation is exact, *i.e.* the translation is sound and complete. A prototype implementation has been developed. We discuss several applications of the characterisation, in particular sound and complete compositional verification for behavioural properties based on maximal models.

1 Introduction

The relationship between a program's syntactical structure and its behaviour is fundamental in program analysis. For example, type systems analyse the structure of a program to deduce properties about its behaviour, while program synthesis studies how to realise a program structure for a desired program behaviour. The relationship is often exploited to phrase program correctness problems in terms of the structure of a program rather than in terms of its behaviour, in order to make analyses more tractable. If program data is abstracted away, and only the *control flow* of programs with (possibly recursive) procedures is considered, the relation between structure and behaviour is well-understood in one direction: program structure, essentially a finite "program graph", can be represented by a

* Partially funded by the IST FP6 programme of the EC, under the IST-FP6-STREP-27004 S3MS project.

** Partially funded by the IST FET programme of the EC, under the IST-2005-015905 MOBIUS project.

pushdown system that induces program behaviour as an "unfolding" of the structure in a context-free manner. This representation has been exploited widely, for example for interprocedural dataflow analysis (*e.g.,* in [18]) and for model checking of behavioural properties formalised in temporal logic (*e.g.,* in [8]). However, in the other direction, this relationship is much less understood: given a program behaviour, how can one capture the program structures that admit this behaviour?

Both program structure and behaviour can be specified by *temporal logic* formulae: structural properties concern the textual sequencing of instructions in a program, while behavioural properties consider their executional sequencing. The relationship between structure and behaviour is naturally expressed at the logic level through the following two questions:

1. when does a structural property entail a behavioural one and,
2. can a behavioural property be characterised by a finite set of structural ones?

This paper addresses this *characterisation problem* in the context of a program model based on control flow graphs of sequential programs with procedures, for properties expressed in a fragment of the modal μ-calculus with boxes and greatest fixed-points only. This temporal logic is suitable for expressing safety properties (*cf.* [4]) in terms of sequences of method invocations, such as security policies restricting access to given resources by means of API method calls (*cf.* [19]). In previous work (see [11] for an overview), we showed how this logic can be used for the specification and compositional verification of safety properties, both on the structural and on the behavioural level, and provided tool support and case studies. In particular, we derived an algorithmic solution to problem (1) stated above (see [11, p. 855]). Here, we give a precise solution to the (considerably more complex) problem (2), showing that every disjunction-free behavioural formula can be characterised by a finite set of structural formulae: a program satisfies the behavioural formula if and only if it satisfies *some* structural formula from the set. For example, the results of this paper allow to derive that the behavioural property "method a never calls method b" is characterised by the (set of) structural properties "there is no call–to–b instruction in (the text of) method a" and "in (the text of) method a, every call–to–b instruction and every return instruction is preceded by some call–to–a instruction" (and hence, due to recursion, control can never reach a call–to–b instruction).

Our solution is constructive, by means of a translation Π from behavioural properties into sets of structural properties. The translation has been implemented in Ocaml and can be tested online [10]. It conceptually amounts to a symbolic execution of the behavioural formula, collecting induced structural constraints along the way. A considerable difficulty is presented by (greatest fixed-point) recursion in the behavioural formula, which has to be captured by recursion in the structural ones (in the absence of recursion it is considerably easier to define such a translation, as we show in [13]). We handle recursion by means of a *tableau construction* that maintains (during the symbolic execution) a symbolic "call stack" indicating which subformulae have been explored for which method. We use this stack to (1) identify when a (sub)formula has been sufficiently explored, so that a branch of the tableau can be finished, and (2) to identify recursion in the collected structural constraints and capture this by fixed-point formulae. We prove that the construction terminates. Moreover, we show that the construction is *sound*, and in case the behavioural formula is disjunction-free, also *complete,* by viewing the tableau system as a proof system.

Applications In addition to its foundational value, the characterisation is useful in various ways. In earlier work, we defined a *maximal model* construction for the logic considered here, and adapted it to the construction of maximal program structures from structural properties [11]. The combination of this construction with the property translation Π presented here provides a solution to the problem of computing maximal program structures from behavioural properties. This allows several problems to be addressed. First, as Section 4 shows, it can be exploited to extend the *compositional verification* technique of [11], where local assumptions are required to be structural, to local behavioural properties. Second, we obtain a solution to the *realisation problem* of synthesising (possibly recursive) program structures from temporal specifications of the permitted method call sequences. Further, it can also be used to reduce infinite-state verification of behavioural control flow properties to finite-state verification of structural properties. Thus, tools supporting structural properties only can in effect be used for verifying behavioural properties. Moreover, in a mobile code deployment scheme, where the security policies of the platform are given as behavioural control flow properties, translating these into structural properties of the loaded applications enables efficient on-device conformance checking via static analysis.

Related Work Our property translation has been motivated by our previous work on adapting Grumberg and Long's approach of using maximal models for compositional verification [9] in the context of control flow properties of sequential programs with procedures [11]. A related problem, known to be problematic for infinite-state systems, is the realisation problem of synthesising a program (skeleton) from a temporal logic specification (see *e.g.*, [16,25]).

Our research is also related to previous work on tableau systems for the verification of infinite-state systems [6,20], model checking based on pushdown systems [5,7,8] or recursive state machines [2], temporal logics for nested calls and returns [3,1], interprocedural dataflow analysis [18], and abstract interpretation (*cf. e.g.,* the completeness result of [17]). However, these analyses infer from the structure of a given program facts about its behaviour; in contrast, our analysis infers, for *all* programs, facts about the structure from facts about the behaviour.

Organisation Section 2 formally defines the program model and logic. Next, Section 3 defines the translation, by means of the tableau construction. Section 4 uses the characterisation to develop a sound and complete compositional verification principle for local behavioural properties, while Section 5 concludes with a discussion of possible extensions and optimisations.

2 Preliminaries: Program Model and Logic

This section summarises the program model and logic for which we develop our property translation. For a more detailed account, the reader is referred to [21,11].

2.1 Specification and Logic

First, we define the general notions of model and specification.

Definition 1. (Model, Specification) *A* model *is a structure* $\mathcal{M} = (S, L, \rightarrow, A, \lambda)$*, where* S *is a set of states,* L *a set of labels,* $\rightarrow \subseteq S \times L \times S$ *a labelled transition relation,* A *a set of atomic propostitions, and* $\lambda\colon S \rightarrow \mathcal{P}(A)$ *a valuation, assigning to each state* s *the set of atomic propositions that hold in* s*. A* specification $\mathcal{S}$ *is a pair* $(\mathcal{M}, E)$*, with* $\mathcal{M}$ *a model and* $E \subseteq S$ *a set of entry states.*

As a property specification language, we use the fragment of the modal μ-calculus [14] with boxes and greatest fixed-points only. This temporal logic is capable of characterising simulation (*cf.* [21]) and is thus suitable for expressing safety properties. Throughout, we fix a set of labels L, a set of atomic propositions A, and a set of propositional variables V.

Definition 2. (Logic) *The formulae of our logic are inductively defined by:* $\phi ::= p \mid \neg p \mid X \mid \phi_1 \wedge \phi_2 \mid \phi_1 \vee \phi_2 \mid [a]\,\phi \mid \nu X.\phi$*, where* $p \in A$*,* $a \in L$ *and* $X \in V$*.*

Satisfaction on states $(\mathcal{M}, s) \models \phi$ (also denoted $s \models {}^{\mathcal{M}}\phi$) is defined in the standard fashion [14]. For instance, formula $[a]\,\phi$ holds of state s in model $\mathcal{M}$ if ϕ holds in all states accessible from s via an edge labelled a. A specification $(\mathcal{M}, E)$ satisfies a formula if all its entry states E satisfy the formula. The constant formulae *true* (denoted tt) and *false* (ff) are definable. For convenience, we use $p \Rightarrow \phi$ to abbreviate $\neg p \vee \phi$. We assume that formulae have pair-wise distinct fixed-point binders, and unless stated otherwise, are closed and guarded (*cf.* [24]).

2.2 Control Flow Structure and Behaviour

Our program model is control–flow based and thus over–approximates actual program behaviour. It defines two different views on programs: a structural and a behavioural view. Both views are instantiations of the general notions of model and specification. Notice in particular that these instantiations yield a structural and a behavioural version of the logic. Again, we refer to [21,11] for more detail.

Control Flow Structure As we abstract away from all data, program structure is defined as a collection of control flow graphs (or flow graphs), one for each of the program's methods. Let *Meth* be a countably infinite set of method names. A method specification is an instance of the general notion of specification.

```
class Number {
   public static boolean even(int n){
      if (n == 0)
         return true;
      else
         return odd(n-1);
   }

   public static boolean odd(int n){
      if (n == 0)
         return false;
      else
         return even(n-1);

   }}
```

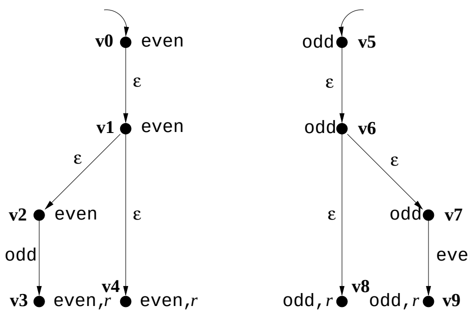


Figure 1. A simple Java class and its flow graph

Definition 3. (Method specification) *A* flow graph *for $m \in Meth$ over a set $M \subseteq Meth$ of method names is a finite model $\mathcal{M}_m = (V_m, L_m, \rightarrow_m, A_m, \lambda_m)$, with V_m the set of* control nodes *of m, $L_m = M \cup \{\varepsilon\}$, $A_m = \{m, r\}$, and $\lambda_m \colon V_m \rightarrow \mathcal{P}(A_m)$ so that $m \in \lambda_m(v)$ for all $v \in V_m$ (*i.e. *each node is tagged with its method name). The nodes $v \in V_m$ with $r \in \lambda_m(v)$ are* return points*. A* method specification *for $m \in Meth$ over M is a pair $(\mathcal{M}_m, E_m)$, s.t. $\mathcal{M}_m$ is a flow graph for m over M and $E_m \subseteq V_m$ a non–empty set of* entry points *of m.*

Next, we define flow graph interfaces. These ensure that control flow graphs can only be composed if their interfaces match.

Definition 4. (Flow graph interface) *A* flow graph interface *is a pair $I = (I^+, I^-)$, where $I^+, I^- \subseteq Meth$ are finite sets of names of* provided *and* required *methods, respectively. The* composition *of two interfaces $I_1 = (I_1^+, I_1^-)$ and $I_2 = (I_2^+, I_2^-)$ is defined by $I_1 \cup I_2 = (I_1^+ \cup I_2^+, I_1^- \cup I_2^-)$.*

The flow graph of a program is essentially the (disjoint) union of its method graphs. To formally define the notion *flow graph with interface*, we use the notion of disjoint union of specifications $\mathcal{S}_1 \uplus \mathcal{S}_2$, where each state is tagged with 1 or 2, respectively, and $(s, i) \xrightarrow{a}_{\mathcal{S}_1 \uplus \mathcal{S}_2} (t, i)$, for $i \in \{1, 2\}$, if and only if $s \xrightarrow{a}_{\mathcal{S}_i} t$.

Definition 5. (Flow graph) *A* flow graph *$\mathcal{G}$ with interface I, written $\mathcal{G} : I$, is defined inductively by*

- *$(\mathcal{M}_m, E_m) : (\{m\}, M)$ if $(\mathcal{M}_m, E_m)$ is a method specification for $m \in Meth$ over M, and*
- *$\mathcal{G}_1 \uplus \mathcal{G}_2 : I_1 \cup I_2$ if $\mathcal{G}_1 : I_1$ and $\mathcal{G}_2 : I_2$.*

Example 1. Figure 1 shows a simple Java class and the (simplified) flow graph it induces. The flow graph consists of two method specifications - one for method `even` and one for method `odd`. Entry nodes are depicted as usual through edges without source.

A flow graph is *closed* if $I^- \subseteq I^+$, *i.e.* it does not require any external methods. Satisfaction, instantiated to flow graphs, is called structural satisfaction $\models_s$.

Example 2. For the flow graph from Example 1, structural formula $\nu X.\, [\texttt{even}]\, r \wedge [\texttt{odd}]\, r \wedge [\varepsilon]\, X$ expresses the property "on every path from a program entry node, the first encountered call edge goes to a return node", in effect specifying that the program is tail-recursive.

Control Flow Behaviour Next, we instantiate specifications on the behavioural level. We use transition label τ to designate internal transfer of control, label m_1 call m_2 for the invocation of method m_2 by method m_1, and label m_2 ret m_1 for the corresponding return from the call.

Definition 6. (Behaviour) *Let $\mathcal{G} = (\mathcal{M}, E) : I$ be a closed flow graph where $\mathcal{M} = (V, L, \rightarrow, A, \lambda)$. The* behaviour *of $\mathcal{G}$ is defined as model $b(\mathcal{G}) = (\mathcal{M}_b, E_b)$, where $\mathcal{M}_b = (S_b, L_b, \rightarrow_b, A_b, \lambda_b)$, such that $S_b = V \times V^*$,* i.e., *states are pairs of control points v and stacks σ, $L_b = \{m_1\ k\ m_2 \mid k \in \{\mathsf{call}, \mathsf{ret}\},\ m_1, m_2 \in I^+\} \cup \{\tau\}$, $A_b = A$, $\lambda_b((v, \sigma)) = \lambda(v)$, and $\rightarrow_b \subseteq S_b \times L_b \times S_b$ is defined by the rules:*

$$\begin{array}{rll}
[\text{transfer}] & (v, \sigma) \xrightarrow{\tau}_b (v', \sigma) & \text{if } m \in I^+,\ v \xrightarrow{\varepsilon}_m v',\ v \models \neg r \\
[\text{call}] & (v_1, \sigma) \xrightarrow{m_1\ \mathsf{call}\ m_2}_b (v_2, v_1' \cdot \sigma) & \text{if } m_1, m_2 \in I^+,\ v_1 \xrightarrow{m_2}_{m_1} v_1',\ v_1 \models \neg r, \\
 & & \quad v_2 \models m_2,\ v_2 \in E \\
[\text{return}] & (v_2, v_1 \cdot \sigma) \xrightarrow{m_2\ \mathsf{ret}\ m_1}_b (v_1, \sigma) & \text{if } m_1, m_2 \in I^+,\ v_2 \models m_2 \wedge r,\ v_1 \models m_1
\end{array}$$

The set of initial states is defined by $E_b = E \times \{\epsilon\}$, where ϵ denotes the empty sequence over V.

Flow graph behaviour can alternatively be defined via *pushdown automata* (PDA) [11, Def. 34]. This is exploited by using PDA model checking for verifying behavioural properties (see for instance [5,7]).

Example 3. Consider the flow graph from Example 1. Because of possible unbounded recursion, it induces an infinite-state behaviour. One example execution of the program is represented by the following path from an initial to a final configuration:

$$\begin{array}{l}
(v_0, \epsilon) \xrightarrow{\tau}_b (v_1, \epsilon) \xrightarrow{\tau}_b (v_2, \epsilon) \xrightarrow{\mathsf{even\ call\ odd}}_b (v_5, v_3) \xrightarrow{\tau}_b (v_6, v_3) \xrightarrow{\tau}_b \\
(v_7, v_3) \xrightarrow{\mathsf{odd\ call\ even}}_b (v_0, v_9 \cdot v_3) \xrightarrow{\tau}_b (v_1, v_9 \cdot v_3) \xrightarrow{\tau}_b \\
(v_4, v_9 \cdot v_3) \xrightarrow{\mathsf{even\ ret\ odd}}_b (v_9, v_3) \xrightarrow{\mathsf{odd\ ret\ even}}_b (v_3, \epsilon)
\end{array}$$

Also on the behavioural level, we instantiate the definition of satisfaction: we define $\mathcal{G} \models_b \phi$ as $b(\mathcal{G}) \models \phi$. The resulting behavioural logic is sufficiently powerful to express the class of *security policies* defined by finite state security automata (*cf. e.g.* [19]).

Example 4. For the flow graph from Example 1, the behavioural formula $\neg \text{even} \vee \nu X.\, [\text{even } \mathsf{call} \text{ even}]\, \mathsf{ff} \wedge [\tau]\, X$ expresses the property "in every program execution starting in method `even`, the first call is not to method `even` itself".

Clean Flow Graphs Method specifications allow return points to have outgoing edges. However, the characterisation of behavioural properties by a set of structural formulae defined below is only correct if the flow graph has no such edges; such flow graphs are called *clean.* We define *cleaning* as a behaviour-preserving unary operation on method specifications, and lift it to flow graphs.

Definition 7. (Cleaning) *Given a method specification* $\mathcal{M}_m = (V_m, L_m, \rightarrow_m, A_m, \lambda_m)$*, the unary operation of* cleaning *is defined by:* $(\mathcal{M}_m)^\bullet = (V_m, L_m, \{s \xrightarrow{l}_m t \mid s \xrightarrow{l}_m t \wedge r \notin \lambda_m(s)\}, A_m, \lambda_m)$*.*

It is easy to see that cleaned flow graphs are clean. Cleaning is idempotent ($(\mathcal{G}^\bullet)^\bullet = \mathcal{G}^\bullet$) and preserves behavioural properties ($\mathcal{G} \models_b \phi \Leftrightarrow \mathcal{G}^\bullet \models_b \phi$). Moreover, any node that is a return point trivially satisfies any structural box formula ($(\mathcal{G}^\bullet, s) \models_s r \Rightarrow \forall l, \sigma.\, (\mathcal{G}^\bullet, s) \models_s [l]\,\sigma$). Below, we will use that the set of nodes that can be reached by a behaviour coincides with the (structural) notion of path in a clean flow graph.

Notational Conventions We use label ε for transfer edges in flow graph structures, and τ for silent behavioural transitions, while ϵ denotes the empty sequence.

In our translation of simulation logic formulae we allow sequences α of labels to appear in box modalities, with the obvious translation $\widehat{\cdot}$ to standard formulae: $\widehat{[\epsilon]\,\psi} = \psi$ and $\widehat{[l \cdot \alpha]\,\psi} = [l]\,\widehat{[\alpha]\,\psi}$, where ϵ denotes the empty sequence, and ψ is already a standard formula.

3 Mapping Behavioural into Structural Properties

This section defines a mapping Π from behavioural properties to sets of structural properties. As mentioned above, the implementation of the mapping can be tested online. Throughout the section we assume that flow graphs are clean, and that behavioural properties are disjunction-free; in Section 5 we discuss how Π can be extended to behavioural formulae with disjunction, though at the expense of completeness. We show that Π computes, from a behavioural property ϕ and closed interface I, a set of structural formulae that characterises ϕ and I. That is, for any closed flow graph $\mathcal{G}$ with interface I and any behavioural formula ϕ that only mentions labels that are in the behaviour of $\mathcal{G}$ (*i.e.,* L_b in Definition 6):

$$\mathcal{G} \models_b \phi \;\Leftrightarrow\; \exists \chi \in \Pi_I(\phi).\, \mathcal{G} \models_s \chi \tag{1}$$

To deal with the fixed-point formulae of the logic, mapping Π is defined with the help of a *tableau construction.* A behavioural formula ϕ gives rise to a (maximal) tableau that induces a set of structural formulae through its leaves. The constructed tableau is finite, *i.e.,* tableau construction terminates.

3.1 Tableau Construction

Our translation is based on a *symbolic execution* of the behavioural property by means of a *tableau construction.* When tracing a symbolic execution path, we tag all subformulae of the formula with unique propositional constants from a set $\mathcal{C}$. We use a global map $\mathcal{S} : \phi \to \mathcal{C}$ to map formulae to their tags. We consider $\mathcal{S}$ as an implicit parameter of the tableau construction (and where necessary, we also use its inverse $\mathcal{S}^{-1}$). The tableau construction operates on sequents of the shape $\vdash_{H,U,C} \phi$ parametrised on:

- a non-empty *history stack* $H \in (I^+ \times (I^- \cup \{\varepsilon\} \cup \mathcal{C})^*)^+$, where each element is a pair consisting of the current method name and a sequence (called frame) of edge labels and propositional constants abbreviating subformulae of ϕ. For any frame $F \in (I^- \cup \{\varepsilon\} \cup \mathcal{C})^*$, we use $\widetilde{F}$ to denote this frame cleaned from propositional constants $X \in \mathcal{C}$:

$$\widetilde{\epsilon} = \epsilon \qquad \widetilde{m \cdot \sigma} = m \cdot \widetilde{\sigma} \qquad \widetilde{\varepsilon \cdot \sigma} = \varepsilon \cdot \widetilde{\sigma} \qquad \widetilde{X \cdot \sigma} = \widetilde{\sigma}$$

- a *fixed-point stack* U, defining an environment for propositional variables by means of a sequence of definitions of the shape $X = \nu X.\psi$; an *open* formula ϕ in a sequent parametrised by U can then be understood via a suitable notion of substitution, based on the standard notion of substitution $\psi\{\theta/X\}$ of a formula θ for a propositional variable X in a formula ψ: the *substitution* of ϕ under U is inductively defined as follows:

$$\phi[\epsilon] = \phi \qquad \phi[(X = \nu X.\psi) \cdot U] = (\phi\{\nu X.\psi/X\})[U]$$

- a *store* C, used for accumulating structural constraints during symbolic execution[1].

We use $\varnothing_{H,m}$, $\varnothing_U$ and $\varnothing_C$ to denote the single-element history stack (m, ϵ) and the empty fixed-point stack and store, respectively.

For a given closed behavioural formula ϕ and method m, we construct a maximal tableau with root $\vdash_{\varnothing_{H,m},\varnothing_U,\varnothing_C} \phi$ that induces a set of structural formulae through its leaves, as described below. We denote the set of induced structural formulae for ϕ and m with $\pi_m(\phi)$. We then define the translation of ϕ *w.r.t.* a given interface I:

$$\Pi_I(\phi) = \{\textstyle\bigwedge_{m \in I^+} \chi_m \mid \chi_m \in \pi_m(\phi)\}$$

During tableau construction, the history stack, fixed-point stack and store are updated as follows, provided the current sequent is not a repeat of an earlier sequent (see below):

1. First, if ϕ is not a fixed-point formula, the propositional constant $\mathcal{S}(\phi)$ tagging the behavioural property ϕ of the current sequent is appended to the end of the frame of the top element of H;

[1] Using stores is not strictly necessary, but simplifies the presentation of the extraction of structural formulae and the correctness proofs.

2. Next,
   - if the behavioural property ϕ prescribes an internal transfer, then ε is appended to the end of the frame of the top element of H;
   - if ϕ prescribes a call from a to b, and the top element of H is in method a, then b is added at the end of the frame of the top element of H, and a new element (b, ϵ) is pushed onto H;
   - if ϕ prescribes a return from a to b, the top element of H is in method a and the next element is in method b, then a new structural constraint is added to the store, reflecting the possibility of currently not being at a return point, and the top element is popped from H; and
   - if ϕ is a fixed-point formula $\nu X.\phi$, then a new equation $X = \nu X.\phi$ is pushed onto the fixed-point stack U, if not already there; this conditional addition is denoted by $(X = \nu X.\phi) \circ U$.

Below, page 13 and further, contains several example tableaux that illustrate this symbolic execution. Notice that non-emptiness of the history stack and closedness of $\phi[U]$ are invariants of the tableau construction.

Tableau System The tableau system is given in Figure 2 as a set of goal-directed rules. Axioms are presented as rules with an empty set of premises denoted by '$-$'. The condition $\mathsf{Ret}(i, a, b, H)$ used in the return rules is defined as $i = a \wedge H \neq \epsilon \wedge \exists F, H'.H = (b, F) \cdot H'$, *i.e.,* there is a pending call from method b on the top of the history stack. Formally, a *tableau* $\mathcal{T} = (T, \lambda)$ is a tree T equipped with a labelling function λ mapping each tree node to a triple consisting of a sequent, a rule name (of the rule applied to this sequent), and a set of triples of shape (i, F, q) where q are literals (that is, atomic propositions in positive or negated form or propositional variables). The triple sets are non-empty only at applications of axiom rules; such leaves are termed *contributing*, and the set of triples is depicted (by convention) as a premise to the rule. A tableau for formula ϕ and method m is a tree with root $\vdash_{\varnothing_{H,m},\varnothing_U,\varnothing_C} \phi$ obtained by applying the rules. A tableau is termed *maximal* if all its leaves are axioms.

If in a tableau there is a leaf node $\vdash_{(i,F)\cdot H,U,C} \phi$ for which there is an internal node $\vdash_{(i,F')\cdot H',U',C'} \phi$ such that F' is a prefix of F, U' is a suffix of U, and C' is a subset of C, we term the former node a *pseudo-repeat*; any node of the latter kind we term a *companion.* An internal tableau node is said to be *stable* if all its descendant leaves are axioms or pseudo-repeats. A tableau is called stable if its root node is stable.

Tableau construction proceeds as follows. First, a minimal stable tableau is computed, *i.e.* if a node is a pseudo-repeat, it is not further explored. If all pseudo-repeats in this tableau satisfy some repeat condition for any of their companions (see below), the tableau is maximal and construction is complete. Otherwise, all pseudo-repeats that are not satisfying any of the repeat conditions are simultaneously unfolded, using a breadth-first exploration strategy, and tableau construction continues until the tableau is stable again, upon which the checking for the repeat conditions is repeated. As discussed below, this process is guaranteed to terminate, resulting in a finite maximal tableau.

$$p\ \frac{\vdash_{(i,F)\cdot H,U,C}\, p}{\{(i,F,p)\}\cup\{(i',F',\mathsf{ff})\mid(i',F')\in H\}\cup C} \qquad \neg p\ \frac{\vdash_{(i,F)\cdot H,U}\, \neg p}{\{(i,F,\neg p)\}\cup\{(i',F',\mathsf{ff})\mid(i',F')\in H\}\cup C}$$

$$\nu X\ \frac{\vdash_{(i,F)\cdot H,U,C}\, \nu X.\phi}{\vdash_{(i,F)\cdot H,(X=\nu X.\phi)\circ U,C}\, X} \qquad X\ \mathrm{unf}\ \frac{\vdash_{(i,F)\cdot H,U,C}\, X}{\vdash_{(i,F\cdot\mathcal{S}(X))\cdot H,U,C}\, \phi}\ (X=\nu X.\phi)\in U$$

$$\wedge\ \frac{\vdash_{(i,F)\cdot H,U,C}\, \phi_1\wedge\phi_2}{\vdash_{(i,F\cdot\mathcal{S}(\phi_1\wedge\phi_2))\cdot H,U,C}\, \phi_1 \qquad \vdash_{(i,F\cdot\mathcal{S}(\phi_1\wedge\phi_2))\cdot H,U,C}\, \phi_2} \qquad \tau\ \frac{\vdash_{(i,F)\cdot H,U,C}\, [\tau]\phi}{\vdash_{(i,F\cdot\mathcal{S}([\tau]\phi)\cdot\varepsilon)\cdot H,U,C}\, \phi}$$

$$\mathrm{call}_0\ \frac{\vdash_{(i,F)\cdot H,U,C}\, [a\ \mathsf{call}\ b]\phi}{-}\ i\neq a \qquad \mathrm{call}_1\ \frac{\vdash_{(i,F)\cdot H,U,C}\, [a\ \mathsf{call}\ b]\phi}{\vdash_{(b,\epsilon)\cdot(i,F\cdot\mathcal{S}([a\ \mathsf{call}\ b]\phi)\cdot b)\cdot H,U,C}\, \phi}\ i=a$$

$$\mathrm{ret}_0\ \frac{\vdash_{(i,F)\cdot H,U,C}\, [a\ \mathsf{ret}\ b]\phi}{-}\ \neg\mathsf{Ret}(i,a,b,H) \qquad \mathrm{ret}_1\ \frac{\vdash_{(i,F)\cdot H,U,C}\, [a\ \mathsf{ret}\ b]\phi}{\vdash_{H,U,C\cup\{(i,F,\neg r)\}}\, \phi}\ \mathsf{Ret}(i,a,b,H)$$

$$\mathrm{IRep}\ \frac{\vdash_{(i,F)\cdot H,U,C}\, \phi}{\{(i,F,\mathcal{S}(\phi))\}\cup C}\ \mathsf{IntRep}(\mathcal{S}(\phi),(i,F)\cdot H) \quad \mathrm{CRep}\ \frac{\vdash_{(i,F)\cdot H,U,C}\, \phi}{-}\ \mathsf{CallRep}(\mathcal{S}(\phi),(i,F)\cdot H,c)$$

$$\mathrm{RRep}\ \frac{\vdash_{(i,F)\cdot H,U,C}\, \phi}{-}\ \mathsf{RetRep}(\mathcal{S}(\phi),(i,F)\cdot H,c)$$

Figure 2. Tableau system

Repeat Conditions We now formulate the three repeat conditions used in the tableau system, giving rise to three types of repeat nodes. Only repeats of the first type, *i.e.,* internal repeats, contribute to triples, giving rise to recursion in structural formulae. In contrast, the other two repeat conditions only recognise that a similar situation has been reached before, and thus no new information will be obtained by further exploration. The first repeat condition requires merely the examination of the top frame of the history stack of the current sequent; the second one requires the examination of the whole path from the root to the pseudo-repeat; while the third one requires the examination of all remaining paths.

Internal repeat Tableau construction guarantees that every tableau node of shape $\vdash_{H'\cdot(i,F'\cdot\mathcal{S}(\phi)\cdot F'')\cdot H'',U,C}\ \phi$ possesses an ancestor node $\vdash_{(i,F')\cdot H'',U',C'}\ \phi$ such that U' is a suffix of U and C' is a subset of C. As a consequence, every node of shape $\vdash_{(i,F'\cdot\mathcal{S}(\phi)\cdot F'')\cdot H,U,C}\ \phi$ is a pseudo-repeat (with companion some ancestor node of shape $\vdash_{(i,F')\cdot H,U',C'}\ \phi$); such pseudo-repeats are termed *internal repeat*s. Intuitively, an internal repeat indicates that a regularity in the structure of method i has been discovered, and thus this regularity should be reflected in the structural formulae. Therefore, in this case $(i,F'\cdot\mathcal{S}(\phi)\cdot F'',\mathcal{S}(\phi))$ is added to the triple set of the IRep axiom. (Notice that in fact the propositional constant $\mathcal{S}(\phi)$ is mapped to a fresh propositional variable, here, and in the construction of the structural formulae. However, for clarity of presentation, we overload the symbols themselves, as their intended meaning should be clear from the context.)

Call repeat A pseudo-repeat $\vdash_{(i,F)\cdot H,U,C}\ \phi$, which has an ancestor node as companion but is not an internal repeat, is a *call repeat* if H matches the call stack of the companion upto the latter's *return depth* (where matching means that the same methods are on the stack, with identical frames); in the special case where both stacks are shorter than the return depth, they have to be identical.

The return depth of a node is only defined if the subtableau of the companion is complete (*i.e.* the pseudo-repeat is the only open branch). When we construct a tableau for a formula with multiple fixed-points, it can happen that two pseudo-repeats occur in the subtableaux of their respective companions. In this case, if both nodes are call repeats exploration terminates (for the current return depth), otherwise, by virtue of the tableau construction, we know that the pseudo-repeat that is not a call repeat, will never become one when continuing the tableau construction. Therefore, we can explore this node further, and break the mutual dependency.

The return depth of a tableau node n, denoted as $\rho(n)$, is defined as the maximal difference between the number of applied return rules and the number of applied call rules on any path from n to a descendant node. Formally, where r and δ range over rule names and sequences of rule names, respectively, while $\mathit{rules}(\pi)$ denotes the sequence of rule names along a tableau path π:

$$\rho'(\epsilon) = 0 \qquad \rho'(r \cdot \delta) = \begin{cases} \rho'(\delta) + 1 & \text{if } r \in \{\mathsf{ret}_0, \mathsf{ret}_1\} \\ \rho'(\delta) - 1 & \text{if } r \in \{\mathsf{call}_0, \mathsf{call}_1\} \\ \rho'(\delta) & \text{otherwise} \end{cases}$$
$$\rho(n) = \mathit{max}\ \{\rho'(\mathit{rules}(\pi)) \mid \pi \text{ a path from } n \text{ to a descendant node}\} \cup \{0\}$$

Return repeat A pseudo-repeat is called a *return repeat* if it has a companion on a different path from the root, such that its history stack is identical to the one of the companion.

Formally, the repeat conditions are defined as follows, where X is $\mathcal{S}(\phi)$, and c is the companion node of the pseudo-repeat with history stack H_c.

$$\begin{aligned} \mathsf{IntRep}(X, (i, F) \cdot H) &\Leftrightarrow X \in F \\ \mathsf{CallRep}(X, (i, F) \cdot H, c) &\Leftrightarrow X \notin F \wedge \mathsf{take}(\rho(c) + 1, (i, F) \cdot H) = \mathsf{take}(\rho(c) + 1, H_c) \\ \mathsf{RetRep}(X, (i, F) \cdot H, c) &\Leftrightarrow (i, F) \cdot H = H_c \end{aligned}$$

Termination The repeat conditions ensure termination of tableau construction.

Theorem 1. *Maximal tableaux are finite.*

Proof. (Sketch) Assume given a behavioural formula ϕ and a method name m. First, observe that repeat condition IntRep puts a bound on the length of frames in stacks, since: (i) ϕ has a finite set of sub-formulae and thus a finite number of propositional constants can occur in frames, (ii) each propositional constant can occur at most once in a frame, and (iii) every method name (or ϵ) in a frame is preceded by some propositional constant. As a consequence, since ϕ can only mention finitely many method names, the set of possible stores is also finite, and so is the set of fixed-point stacks.

Further, we shall show that repeat condition RetRep puts a bound on the return depth of nodes in any tableau for ϕ and m. Hence, since there is only a finite number of method names and frames that can occur in history stacks, function take takes values from a finite set. As a consequence (since formula ϕ has a finite set of sub-formulae and mentions finitely many method names), there is a bound on the length of paths from the tableau root without reaching a node satisfying one of the repeat conditions: a path reaches either (i) an axiom, (ii) an internal repeat, (iii) a return repeat, or else (iv) eventually a pseudo-repeat satisfying repeat condition $\mathsf{CallRep}$.

Boundedness of the return depth is established as follows. Assume on the contrary that no such bound exists; that is, that for each natural number d there is, in some tableau for ϕ and m, a tableau node and a path from this node having return depth d. Notice that every path of the tableau construction can be viewed as an execution of a *pushdown automaton* with control state defined by the formula, fixed point stack and store of the current sequent (thus from a finite domain), and with stack defined by the history stack. It can be shown, by referring to the *Pumping lemma* for context-free languages, but viewed from a pushdown automata perspective (see Appendix A), that for a sufficiently large d, any tableau path (from any tableau node) of return depth d contains nodes $n_1 :\vdash_{(i,F)\cdot H,U,C} \phi$, $n_2 :\vdash_{(i,F)\cdot H'\cdot H,U',C'} \phi$, $n_3 :\vdash_{H'\cdot H,U'',C''} \phi'$, and $n_4 :\vdash_{H,U''',C'''} \phi'$ along the path (in the given order), such that sequent $\vdash_{H,U''',C'''} \phi'$ can be derived from sequent $\vdash_{(i,F)\cdot H,U,C} \phi$ by the same sequence of rules by which node n_3 has been derived from node n_2. Therefore, by virtue of the tableau construction, there must be a node $n_4' :\vdash_{H,U''',C'''} \phi'$ on another, shorter path from node n_1, and hence node n_4 is a pseudo-repeat satisfying repeat condition RetRep. But this contradicts the statement that n_4 is an internal node, and thus the initial assumption that no bound exists on the return depth must be wrong. □

Structural Formulae Induced by a Tableau A maximal tableau for ϕ and m induces, through the sets of triples accumulated in the leaves, a set of structural formulae $\pi_m(\phi)$ in the following manner:

1. Let $\mathcal{L}$ be the set of non-empty triple sets collected from the leaves of the tableau. Build a collection of *choice sets* $\Lambda(\mathcal{L})$, by choosing one triple from each element in $\mathcal{L}$.
2. For each choice set $\lambda \in \Lambda(\mathcal{L})$,
   (a) Group the triples of λ according to method names: for each $i \in I$, define

$$\Xi_i = \{(F, q) \mid (i, F, q) \in \lambda\}$$

   (b) For each $i \in I$ such that $\Xi_i \neq \varnothing$, build a formula $i \Rightarrow \Omega(\Xi_i)$, where

$$\begin{aligned}\Omega(\Xi) &= \textstyle\bigwedge_{\phi\in\Omega'(\Xi)} \phi\\ \Omega'(\Xi) &= \{[a]\,\Omega(\Xi') \mid a \in I^- \wedge \Xi' = \{(F,q) \mid (a\cdot F, q) \in \Xi\} \wedge \Xi' \neq \varnothing\}\cup\\ &\quad\ \{\nu X.\Omega(\Xi') \mid X \in \mathcal{C} \wedge \Xi' = \{(F,q) \mid (X\cdot F, q) \in \Xi\} \wedge \Xi' \neq \varnothing\}\cup\\ &\quad\ \{q \mid (\epsilon, q) \in \Xi\}\end{aligned}$$

(c) The *induced formula* χ for λ is the conjunction of the formulae obtained in the previous step.

3. The set $\pi_m(\phi)$ of induced formulae is the set of induced formulae for $\lambda \in \Lambda(\mathcal{L})$.

For example, the choice set $\lambda = \{(a, X \cdot b, \neg r), (a, X \cdot b, X)\}$ induces (by step 2) the structural formula $a \Rightarrow \nu X. [b] (\neg r \wedge X)$. Notice that all induced formulae are closed and guarded whenever the original behavioural one is.

Examples To illustrate the tableau construction, we discuss several examples. The first example illustrates how an internal repeat gives rise to a structural formula with fixed point. Consider the behavioural formula $\phi = \nu X. [a\ \mathsf{call}\ b]\, X \wedge [b\ \mathsf{ret}\ a]\, (\neg r \wedge X)$. Figure 3 shows the mapping $\mathcal{S}$ from the subformulae of ϕ to propositional constants, and the tableau that is constructed for this formula. The first node where a triple is produced is the one labelled ret_1; the triple is then propagated to the two leaves that result from application of the rule for atomic propositions, and simple repeat, respectively. The tableau has two leaves with non-empty triple sets; $\mathcal{L}$ thus consists of two sets of two triples each.

Thus, to construct the set of structural formulae, we compute structural formulae for the four choice sets resulting from $\mathcal{L}$:

$$\begin{array}{l}
\{(a, X_4 \cdot X_1 \cdot X_2 \cdot b \cdot X_5, \neg r), (a, X_4 \cdot X_1 \cdot X_2 \cdot b \cdot X_5, X_4)\} \\
\{(a, X_4 \cdot X_1 \cdot X_2 \cdot b \cdot X_5, \neg r), (b, X_4 \cdot X_1, \neg r)\} \\
\{(b, X_4 \cdot X_1, \neg r), (a, X_4 \cdot X_1 \cdot X_2 \cdot b \cdot X_5, X_4)\} \\
\{(b, X_4 \cdot X_1, \neg r)\}
\end{array}$$

The first set gives rise to the structural formula $a \Rightarrow \nu X_4.\nu X_1.\nu X_2. [b]\, \nu X_5.(\neg r \wedge X_4)$, which simplifies to $\chi_1 = a \Rightarrow \nu X.[b](\neg r \wedge X)$. The last set gives rise to the formula (after simplification) $\chi_2 = b \Rightarrow \neg r$. The formulae constructed from the second and third set are subsumed by χ_2, and hence $\pi_a(\phi) = \{\chi_1, \chi_2\}$. For ϕ and method b there is a single tableau, which has no leaf triples, and hence $\pi_b(\phi) = \{\mathsf{tt}\}$. Thus, $\Pi(\phi) = \{\chi_1, \chi_2\}$.

The next example illustrates why the call repeat needs to consider the return depth. Consider the behavioural formula $\phi = \nu X. [a\ \mathsf{call}\ b]\, \mathsf{ff} \wedge [a\ \mathsf{call}\ a]\, X \wedge [a\ \mathsf{ret}\ a]\, X$. Figure 4 shows the tableau that is constructed for this formula. During tableau construction, the node $\vdash_{(a,\epsilon)\cdot(a,X_1\cdot X_2\cdot X_5\cdot X_6\cdot a),X=\phi,\varnothing_C} X$ is a pseudo-repeat (with $\vdash_{(a,\epsilon),X=\phi,\varnothing_C} X$ as companion candidate). However, because of the application of ret_0 in the subtree of the companion candidate, the return depth is 1, and thus this pseudo repeat is not a repeat. After unfolding the fixed point once more, tableau construction terminates. Notice that if the earlier pseudo repeat had been a repeat, the triple $(a, X_1 \cdot X_2 \cdot X_5, \neg r)$ (and therewith the structural formula $a \Rightarrow [b]\, \mathsf{ff} \wedge \neg r)$ would not have been found.

The next example shows why the repeat condition has to be tested on all subformulae, and not only on fixed points . Consider the behavioural formula $\phi = [a\ \mathsf{call}\ b]\, (\nu X.r \wedge [b\ \mathsf{ret}\ a]\, (\neg r \wedge [a\ \mathsf{call}\ b]\, X))$. Figure 5 shows the tableau that is constructed for this formula. When symbolically executing the formula, the

X_0	$\nu X.\, [a\ \mathsf{call}\ b]\, X \wedge [b\ \mathsf{ret}\ a]\, (\neg r \wedge X)$	X_4	$[b\ \mathsf{ret}\ a]\, (\neg r \wedge X)$
X_1	X	X_5	$\neg r \wedge X$
X_2	$[a\ \mathsf{call}\ b]\, X \wedge [b\ \mathsf{ret}\ a]\, (\neg r \wedge X)$	X_6	$\neg r$
X_3	$[a\ \mathsf{call}\ b]\, X$		

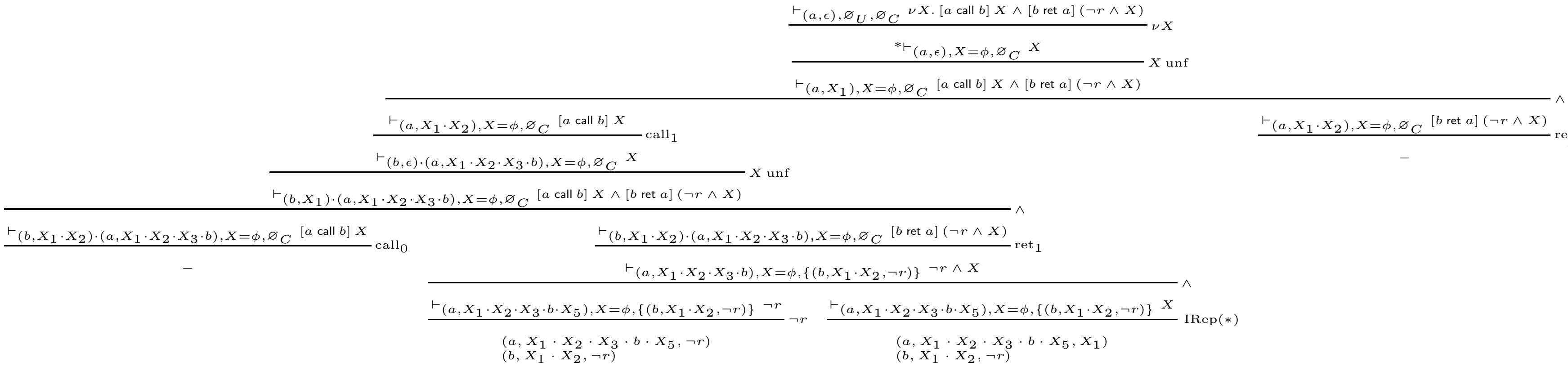
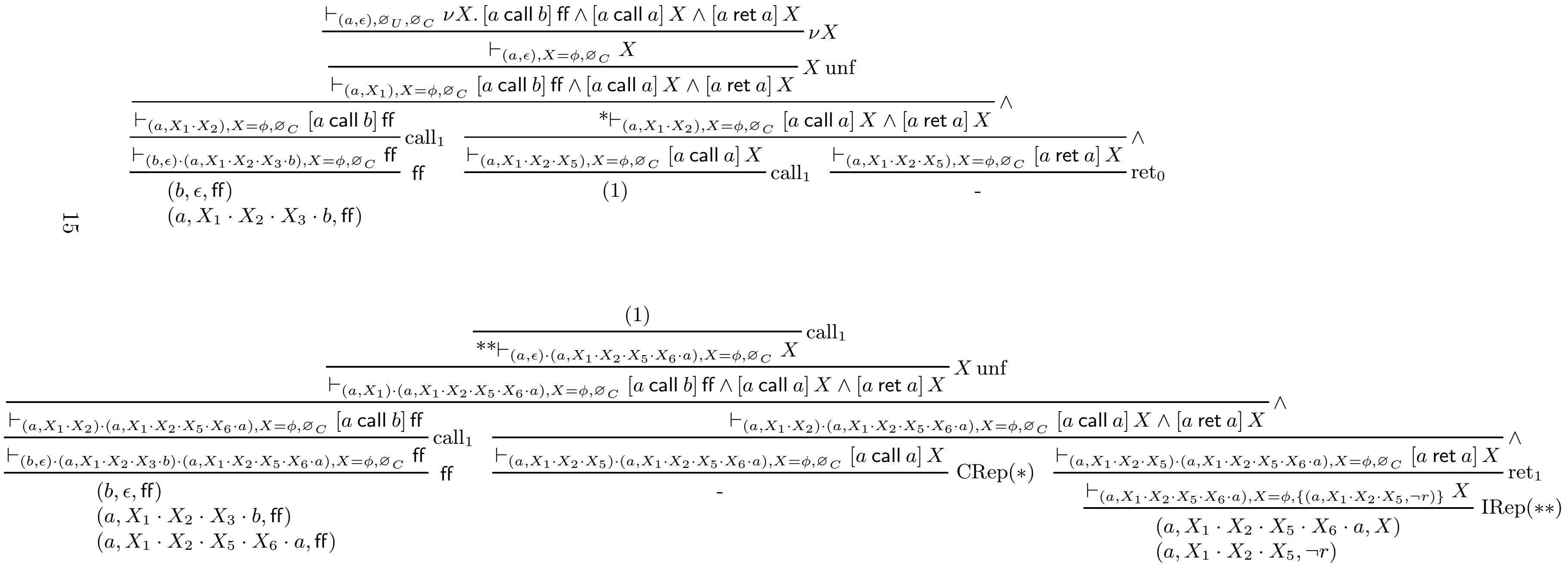

$\vdash_{(a,\epsilon),\varnothing_U,\varnothing_C} \nu X.\, [a\ \mathsf{call}\ b]\, X \wedge [b\ \mathsf{ret}\ a]\, (\neg r \wedge X)$
νX
$^{*}\vdash_{(a,\epsilon),X=\phi,\varnothing_C} X$
X unf
$\vdash_{(a,X_1),X=\phi,\varnothing_C} [a\ \mathsf{call}\ b]\, X \wedge [b\ \mathsf{ret}\ a]\, (\neg r \wedge X)$
$\wedge$
$\vdash_{(a,X_1\cdot X_2),X=\phi,\varnothing_C} [a\ \mathsf{call}\ b]\, X$
call_1
$\vdash_{(a,X_1\cdot X_2),X=\phi,\varnothing_C} [b\ \mathsf{ret}\ a]\, (\neg r \wedge X)$
ret_0
$-$
$\vdash_{(b,\epsilon)\cdot(a,X_1\cdot X_2\cdot X_3\cdot b),X=\phi,\varnothing_C} X$
X unf
$\vdash_{(b,X_1)\cdot(a,X_1\cdot X_2\cdot X_3\cdot b),X=\phi,\varnothing_C} [a\ \mathsf{call}\ b]\, X \wedge [b\ \mathsf{ret}\ a]\, (\neg r \wedge X)$
$\wedge$
$\vdash_{(b,X_1\cdot X_2)\cdot(a,X_1\cdot X_2\cdot X_3\cdot b),X=\phi,\varnothing_C} [a\ \mathsf{call}\ b]\, X$
call_0
$-$
$\vdash_{(b,X_1\cdot X_2)\cdot(a,X_1\cdot X_2\cdot X_3\cdot b),X=\phi,\varnothing_C} [b\ \mathsf{ret}\ a]\, (\neg r \wedge X)$
ret_1
$\vdash_{(a,X_1\cdot X_2\cdot X_3\cdot b),X=\phi,\{(b,X_1\cdot X_2,\neg r)\}} \neg r \wedge X$
$\wedge$
$\vdash_{(a,X_1\cdot X_2\cdot X_3\cdot b\cdot X_5),X=\phi,\{(b,X_1\cdot X_2,\neg r)\}} \neg r$
$\neg r$
$(a, X_1 \cdot X_2 \cdot X_3 \cdot b \cdot X_5, \neg r)$
$(b, X_1 \cdot X_2, \neg r)$
$\vdash_{(a,X_1\cdot X_2\cdot X_3\cdot b\cdot X_5),X=\phi,\{(b,X_1\cdot X_2,\neg r)\}} X$
IRep$(*)$
$(a, X_1 \cdot X_2 \cdot X_3 \cdot b \cdot X_5, X_1)$
$(b, X_1 \cdot X_2, \neg r)$

Figure 3. Tableau for $\nu X.\, [a\ \mathsf{call}\ b]\, X \wedge [b\ \mathsf{ret}\ a]\, (\neg r \wedge X)$ and a, giving rise to $\{a \Rightarrow \nu X.\, [b]\, (\neg r \wedge X), b \Rightarrow \neg r\}$

X_0	$\nu X.\, [a\ \mathsf{call}\ b]\, \mathsf{ff} \wedge [a\ \mathsf{call}\ a]\, X \wedge [a\ \mathsf{ret}\ a]\, X$		X_4	ff
X_1	X		X_5	$[a\ \mathsf{call}\ a]\, X \wedge [a\ \mathsf{ret}\ a]\, X$
X_2	$[a\ \mathsf{call}\ b]\, \mathsf{ff} \wedge [a\ \mathsf{call}\ a]\, X \wedge [a\ \mathsf{ret}\ a]\, X$		X_6	$[a\ \mathsf{call}\ a]\, X$
X_3	$[a\ \mathsf{call}\ b]\, \mathsf{ff}$		X_7	$[a\ \mathsf{ret}\ a]\, X$



$\vdash_{(a,\epsilon),\varnothing_U,\varnothing_C} \nu X.\, [a\ \mathsf{call}\ b]\, \mathsf{ff} \wedge [a\ \mathsf{call}\ a]\, X \wedge [a\ \mathsf{ret}\ a]\, X$
νX
$\vdash_{(a,\epsilon),X=\phi,\varnothing_C} X$
X unf
$\vdash_{(a,X_1),X=\phi,\varnothing_C} [a\ \mathsf{call}\ b]\, \mathsf{ff} \wedge [a\ \mathsf{call}\ a]\, X \wedge [a\ \mathsf{ret}\ a]\, X$
$\wedge$
$\vdash_{(a,X_1\cdot X_2),X=\phi,\varnothing_C} [a\ \mathsf{call}\ b]\, \mathsf{ff}$
call_1
$\vdash_{(b,\epsilon)\cdot(a,X_1\cdot X_2\cdot X_3\cdot b),X=\phi,\varnothing_C} \mathsf{ff}$
ff
(b,ϵ,ff)
$(a, X_1\cdot X_2\cdot X_3\cdot b, \mathsf{ff})$
$^{*}\vdash_{(a,X_1\cdot X_2),X=\phi,\varnothing_C} [a\ \mathsf{call}\ a]\, X \wedge [a\ \mathsf{ret}\ a]\, X$
$\wedge$
$\vdash_{(a,X_1\cdot X_2\cdot X_5),X=\phi,\varnothing_C} [a\ \mathsf{call}\ a]\, X$
call_1
(1)
$\vdash_{(a,X_1\cdot X_2\cdot X_5),X=\phi,\varnothing_C} [a\ \mathsf{ret}\ a]\, X$
ret_0
-
(1)
call_1
$^{**}\vdash_{(a,\epsilon)\cdot(a,X_1\cdot X_2\cdot X_5\cdot X_6\cdot a),X=\phi,\varnothing_C} X$
X unf
$\vdash_{(a,X_1)\cdot(a,X_1\cdot X_2\cdot X_5\cdot X_6\cdot a),X=\phi,\varnothing_C} [a\ \mathsf{call}\ b]\, \mathsf{ff} \wedge [a\ \mathsf{call}\ a]\, X \wedge [a\ \mathsf{ret}\ a]\, X$
$\wedge$
$\vdash_{(a,X_1\cdot X_2)\cdot(a,X_1\cdot X_2\cdot X_5\cdot X_6\cdot a),X=\phi,\varnothing_C} [a\ \mathsf{call}\ b]\, \mathsf{ff}$
call_1
$\vdash_{(b,\epsilon)\cdot(a,X_1\cdot X_2\cdot X_3\cdot b)\cdot(a,X_1\cdot X_2\cdot X_5\cdot X_6\cdot a),X=\phi,\varnothing_C} \mathsf{ff}$
ff
(b,ϵ,ff)
$(a, X_1\cdot X_2\cdot X_3\cdot b, \mathsf{ff})$
$(a, X_1\cdot X_2\cdot X_5\cdot X_6\cdot a, \mathsf{ff})$
$\vdash_{(a,X_1\cdot X_2)\cdot(a,X_1\cdot X_2\cdot X_5\cdot X_6\cdot a),X=\phi,\varnothing_C} [a\ \mathsf{call}\ a]\, X \wedge [a\ \mathsf{ret}\ a]\, X$
$\wedge$
$\vdash_{(a,X_1\cdot X_2\cdot X_5)\cdot(a,X_1\cdot X_2\cdot X_5\cdot X_6\cdot a),X=\phi,\varnothing_C} [a\ \mathsf{call}\ a]\, X$
CRep($*$)
-
$\vdash_{(a,X_1\cdot X_2\cdot X_5)\cdot(a,X_1\cdot X_2\cdot X_5\cdot X_6\cdot a),X=\phi,\varnothing_C} [a\ \mathsf{ret}\ a]\, X$
ret_1
$\vdash_{(a,X_1\cdot X_2\cdot X_5\cdot X_6\cdot a),X=\phi,\{(a,X_1\cdot X_2\cdot X_5,\neg r)\}} X$
IRep($**$)
$(a, X_1\cdot X_2\cdot X_5\cdot X_6\cdot a, X)$
$(a, X_1\cdot X_2\cdot X_5, \neg r)$

Figure 4. Tableau for $\nu X.\, [a\ \mathsf{call}\ b]\, \mathsf{ff} \wedge [a\ \mathsf{call}\ a]\, X \wedge [a\ \mathsf{ret}\ a]\, X$ and a, giving rise to $\{a \Rightarrow \nu X_1.\, [b]\, \mathsf{ff} \wedge [a]\, X_1, a \Rightarrow [b]\, \mathsf{ff} \wedge \neg r\}$

X_0	$[a\ \mathsf{call}\ b]\,(\nu X.r \wedge [b\ \mathsf{ret}\ a]\,(\neg r \wedge [a\ \mathsf{call}\ b]\,X))$		X_5	$[b\ \mathsf{ret}\ a]\,(\neg r \wedge [a\ \mathsf{call}\ b]\,X)$
X_1	$\nu X.r \wedge [b\ \mathsf{ret}\ a]\,(\neg r \wedge [a\ \mathsf{call}\ b]\,X)$		X_6	$\neg r \wedge [a\ \mathsf{call}\ b]\,X$
X_2	X		X_7	$\neg r$
X_3	$r \wedge [b\ \mathsf{ret}\ a]\,(\neg r \wedge [a\ \mathsf{call}\ b]\,X)$		X_8	$[a\ \mathsf{call}\ b]\,X$
X_4	r			

$$
\dfrac{\vdash_{(a,\epsilon),\varnothing_U,\varnothing_C}\ [a\ \mathsf{call}\ b]\,(\nu X.r \wedge [b\ \mathsf{ret}\ a]\,(\neg r \wedge [a\ \mathsf{call}\ b]\,X))}
{\dfrac{\vdash_{(b,\epsilon)\cdot(a,X_0\cdot b),\varnothing_U,\varnothing_C}\ \nu X.r \wedge [b\ \mathsf{ret}\ a]\,(\neg r \wedge [a\ \mathsf{call}\ b]\,X)}
{\dfrac{\vdash_{(b,\epsilon)\cdot(a,X_0\cdot b),X=X_1,\varnothing_C}\ X}
{\dfrac{\vdash_{(b,X_2)\cdot(a,X_0\cdot b),X=X_1,\varnothing_C}\ r \wedge [b\ \mathsf{ret}\ a]\,(\neg r \wedge [a\ \mathsf{call}\ b]\,X)}
{\dfrac{\vdash_{(b,X_2\cdot X_3)\cdot(a,X_0\cdot b),X=X_1,\varnothing_C}\ r}{\begin{array}{c}(b,X_2\cdot X_3,r)\\(a,X_0\cdot b,\mathsf{ff})\end{array}}\ r
\qquad
\dfrac{\vdash_{(b,X_2\cdot X_3)\cdot(a,X_0\cdot b),X=X_1,\varnothing_C}\ [b\ \mathsf{ret}\ a]\,(\neg r \wedge [a\ \mathsf{call}\ b]\,X)}
{\dfrac{{}^{*}\vdash_{(a,X_0\cdot b),X=X_1,\{(b,X_2\cdot X_3,\neg r)\}}\ \neg r \wedge [a\ \mathsf{call}\ b]\,X}
{\dfrac{\vdash_{(a,X_0\cdot b\cdot X_6),X=X_1,\{(b,X_2\cdot X_3,\neg r)\}}\ \neg r}{\begin{array}{c}(a,X_0\cdot b\cdot X_6,\neg r)\\(b,X_2\cdot X_3,\neg r)\end{array}}\ \neg r
\qquad
\dfrac{\vdash_{(a,X_0\cdot b\cdot X_6),X=X_1,\{(b,X_2\cdot X_3,\neg r)\}}\ [a\ \mathsf{call}\ b]\,X}{(1)}\ \mathrm{call}_1}\ \wedge}\ \mathrm{ret}_1}\ \wedge}\ X\ \mathrm{unf}}\ \nu X}\ \mathrm{call}_1
$$

$$
\dfrac{(1)}
{\dfrac{\vdash_{(b,\epsilon)\cdot(a,X_0\cdot b\cdot X_6\cdot X_8\cdot b),X=X_1,\{(b,X_2\cdot X_3,\neg r)\}}\ X}
{\dfrac{\vdash_{(b,X_2)\cdot(a,X_0\cdot b\cdot X_6\cdot X_8\cdot b),X=X_1,\{(b,X_2\cdot X_3,\neg r)\}}\ r \wedge [b\ \mathsf{ret}\ a]\,(\neg r \wedge [a\ \mathsf{call}\ b]\,X)}
{\dfrac{\vdash_{(b,X_2\cdot X_3)\cdot(a,X_0\cdot b\cdot X_6\cdot X_8\cdot b),X=X_1,\{(b,X_2\cdot X_3,\neg r)\}}\ r}{\begin{array}{c}(b,X_2\cdot X_3,r)\\(a,X_0\cdot b\cdot X_6\cdot X_8\cdot b,\mathsf{ff})\\(b,X_2\cdot X_3,\neg r)\end{array}}\ r
\quad
\dfrac{\vdash_{(b,X_2\cdot X_3)\cdot(a,X_0\cdot b\cdot X_6\cdot X_8\cdot b),X=X_1,\{(b,X_2\cdot X_3,\neg r)\}}\ [b\ \mathsf{ret}\ a]\,(\neg r \wedge [a\ \mathsf{call}\ b]\,X)}
{\dfrac{\vdash_{(a,X_0\cdot b\cdot X_6\cdot X_8\cdot b),X=X_1,\{(b,X_2\cdot X_3,\neg r)\}}\ \neg r \wedge [a\ \mathsf{call}\ b]\,X}{\begin{array}{c}(a,X_0\cdot b\cdot X_6\cdot X_8\cdot b,X_6)\\(b,X_2\cdot X_3,\neg r)\end{array}}\ \mathrm{IRep}(*)}\ \mathrm{ret}_1}\ \wedge}\ X\ \mathrm{unf}}
$$

Figure 5. Tableau for $[a\ \mathsf{call}\ b]\,(\nu X.r \wedge [b\ \mathsf{ret}\ a]\,(\neg r \wedge [a\ \mathsf{call}\ b]\,X))$ and a, giving rise to $\{(a \Rightarrow [b]\,(\nu X_6.\neg r \wedge [b]\,X_6)) \wedge (b \Rightarrow r)\}$

fixed point is unfolded in method b. However, the subsequent return from b to a removes the frame related to this call to b, and thus destroys the tag that the fixed point was unfolded. However, because we tag occurrences of all subformulae, the tableau construction recognises the repeat *after* returning from b to a.

Finally, the last example concerns a behavioural formula with a return loop within a call loop: $\phi = \nu X.\nu Y.\, [a\ \mathsf{ret}\ a]\, (\neg r \wedge Y) \wedge [\tau]\, X \wedge [a\ \mathsf{call}\ a]\, X$. For termination of the tableau, all three different repeat conditions are necessary. Figure 6 shows the tableau that is constructed for this formula (over 2 pages).

3.2 Tableau Unfolding

A maximal tableau gives rise to a (potentially infinite) unfolded tableau defined as follows.

Definition 8. (Tableau unfolding) *Let $\mathcal{T} = (T, \lambda)$ be a maximal tableau. The* unfolding *$(\mathcal{T}^*, \tau)$ of $\mathcal{T}$ consists of a maximal tableau $\mathcal{T}^* = (T^*, \lambda^*)$ for the same root sequent but in a modified tableau system without repeat rules, and of a tree morphism $\tau : T^* \rightarrow T$ satisfying the following constraints:*

(i) *τ preserves the root and respects the order of children;*
(ii) *τ preserves rule names in labels, except for nodes mapped to repeat nodes;*
(iii) *if a node n^* is mapped to a repeat node n with companion n', then the rule name of n^* is the one of n', and the child of n^* is mapped to the child of n'.*

The repeat conditions guarantee well-formedness of the unfolding, since applicability of the tableau rules (in the modified system) depends only on the history stack up to the return depth of the corresponding sequent in the original tableau. Figure 7 shows an initial fragment of the unfolding of the tableau in Figure 3.

To prove correctness of the tableau construction in the next section, we show how triples in the leaves of the tableau unfolding correspond to valid formulae. For this, we first define the notion of projection of a structural formula. Given a structural formula χ and label α, the *projection* of χ on α, denoted χ_α, is inductively defined as follows.

$$\begin{array}{rrr}
m_\alpha = m & r_\alpha = \mathsf{ff} & (\chi_1 \wedge \chi_2)_\alpha = (\chi_1)_\alpha \wedge (\chi_2)_\alpha \\
(\neg m)_\alpha = \neg m & (\neg r)_\alpha = \mathsf{tt} & (\chi_1 \vee \chi_2)_\alpha = (\chi_1)_\alpha \vee (\chi_2)_\alpha \\
([\beta]\, \phi)_\alpha = \begin{cases} \phi & \text{if } \beta = \alpha \\ \mathsf{tt} & \text{otherwise} \end{cases} & & (\nu X.\phi)_\alpha = (\phi[\nu X.\phi/X])_\alpha
\end{array}$$

Intuitively, χ_α denotes the strongest property induced by χ on the nodes reachable from the current one by following an edge labelled α. Note that projection is well-defined for guarded formulae. As an example, for $\chi = a \Rightarrow \nu X.\, [b]\, (\neg r \wedge X)$ we have $\chi_b = a \Rightarrow (\neg r \wedge \nu X.\, [b]\, (\neg r \wedge X))$.

We lift projection to clean frames F by inductively defining $\chi_\epsilon = \chi$ and $\chi_{\alpha \cdot F} = (\chi_\alpha)_F$. The following result establishes an essential property of χ_F, where (here) $\models_s \chi$ denotes validity of χ in all clean control flow graphs.

X_0	$\nu X.\nu Y.\,[a\;\mathsf{ret}\;a]\,(\neg r\wedge Y)\wedge[\tau]\,X\wedge[a\;\mathsf{call}\;a]\,X$	X_6	$\neg r\wedge Y$
X_1	X	X_7	$\neg r$
X_2	$\nu Y.\,[a\;\mathsf{ret}\;a]\,(\neg r\wedge Y)\wedge[\tau]\,X\wedge[a\;\mathsf{call}\;a]\,X$	X_8	$[\tau]\,X\wedge[a\;\mathsf{call}\;a]\,X$
X_3	$\nu Y.\,[a\;\mathsf{ret}\;a]\,(\neg r\wedge Y)$	X_9	$[\tau]\,X$
X_4	Y	X_{10}	$[a\;\mathsf{call}\;a]\,X$
X_5	$[a\;\mathsf{ret}\;a]\,(\neg r\wedge Y)$		

$$\dfrac{\vdash_{(a,\epsilon),\varnothing_U,\varnothing_C} \nu X.\nu Y.\,[a\;\mathsf{ret}\;a]\,(\neg r\wedge Y)\wedge[\tau]\,X\wedge[a\;\mathsf{call}\;a]\,X}{\dfrac{{}^{*}\vdash_{(a,\epsilon),X=\phi,\varnothing_C}\; X}{\dfrac{\vdash_{(a,X_1),X=\phi,\varnothing_C} \nu Y.\,[a\;\mathsf{ret}\;a]\,(\neg r\wedge Y)\wedge[\tau]\,X\wedge[a\;\mathsf{call}\;a]\,X}{\dfrac{\vdash_{(a,X_1\cdot X_2),X=\phi,\varnothing_C} \nu Y.\,[a\;\mathsf{ret}\;a]\,(\neg r\wedge Y)}{\dfrac{\vdash_{(a,X_1\cdot X_2),(Y=X_3)\cdot(X=\phi),\varnothing_C}\; Y}{\dfrac{\vdash_{(a,X_1\cdot X_2\cdot X_4),(Y=X_3)\cdot(X=\phi),\varnothing_C}\;[a\;\mathsf{ret}\;a]\,(\neg r\wedge Y)}{-}\,\mathrm{ret}_0}\,Y\text{ unf}}\,\nu Y \qquad \dfrac{\vdash_{(a,X_1\cdot X_2),X=\phi,\varnothing_C}\;[\tau]\,X\wedge[a\;\mathsf{call}\;a]\,X}{\dfrac{\vdash_{(a,X_1\cdot X_2\cdot X_8),X=\phi,\varnothing_C}\;[\tau]\,X}{\dfrac{\vdash_{(a,X_1\cdot X_2\cdot X_8\cdot X_9\cdot\epsilon),X=\phi,\varnothing_C}\; X}{(a,X_1\cdot X_2\cdot X_8\cdot X_9\cdot\epsilon,X)}\,\mathrm{IRep}(*)}\,\tau \qquad \dfrac{\vdash_{(a,X_1\cdot X_2\cdot X_8),X=\phi,\varnothing_C}\;[a\;\mathsf{call}\;a]\,X}{\dfrac{{}^{**}\vdash_{(a,\epsilon)\cdot(a,X_1\cdot X_2\cdot X_8\cdot X_{10}\cdot a),X=\phi,\varnothing_C}\; X}{(1)}\,X\text{ unf}}\,\mathrm{call}_1}\,\wedge}\,\wedge}\,X\text{ unf}}\,\nu X$$

$$\dfrac{(1)}{\dfrac{\vdash_{(a,X_1)\cdot(a,X_1\cdot X_2\cdot X_8\cdot X_{10}\cdot a),X=\phi,\varnothing_C} \nu Y.\,[a\;\mathsf{ret}\;a]\,(\neg r\wedge Y)\wedge[\tau]\,X\wedge[a\;\mathsf{call}\;a]\,X}{\dfrac{\vdash_{(a,X_1\cdot X_2)\cdot(a,X_1\cdot X_2\cdot X_8\cdot X_{10}\cdot a),X=\phi,\varnothing_C} \nu Y.\,[a\;\mathsf{ret}\;a]\,(\neg r\wedge Y)}{(2)}\,\nu Y \qquad \dfrac{{}^{****}\vdash_{(a,X_1\cdot X_2)\cdot(a,X_1\cdot X_2\cdot X_8\cdot X_{10}\cdot a),X=\phi,\varnothing_C}\;[\tau]\,X\wedge[a\;\mathsf{call}\;a]\,X}{\dfrac{\vdash_{(a,X_1\cdot X_2\cdot X_8)\cdot(a,X_1\cdot X_2\cdot X_8\cdot X_{10}\cdot a),X=\phi,\varnothing_C}\;[\tau]\,X}{\dfrac{\vdash_{(a,X_1\cdot X_2\cdot X_8\cdot X_9\cdot\epsilon)\cdot(a,X_1\cdot X_2\cdot X_8\cdot X_{10}\cdot a),X=\phi,\varnothing_C}\; X}{(a,X_1\cdot X_2\cdot X_8\cdot X_9\cdot\epsilon,X_1)}\,\mathrm{IRep}(**)}\,\tau \qquad \dfrac{\vdash_{(a,X_1\cdot X_2\cdot X_8)\cdot(a,X_1\cdot X_2\cdot X_8\cdot X_{10}\cdot a),X=\phi,\varnothing_C}\;[a\;\mathsf{call}\;a]\,X}{\dfrac{\vdash_{(a,\epsilon)\cdot(a,X_1\cdot X_2\cdot X_8\cdot X_{10}\cdot a)\cdot(a,X_1\cdot X_2\cdot X_8\cdot X_{10}\cdot a),X=\phi,\varnothing_C}\; X}{(3)}\,X\text{ unf}}\,\mathrm{call}_1}\,\wedge}\,\wedge}\,X\text{ unf}$$

$$\dfrac{(2)}{\dfrac{\vdash_{(a,X_1\cdot X_2)\cdot(a,X_1\cdot X_2\cdot X_8\cdot X_{10}\cdot a),(Y=X_3)\cdot(X=\phi),\varnothing_C}\ Y}{\dfrac{\vdash_{(a,X_1\cdot X_2\cdot X_4)\cdot(a,X_1\cdot X_2\cdot X_8\cdot X_{10}\cdot a),(Y=X_3)\cdot(X=\phi),\varnothing_C}\ [a\ \mathsf{ret}\ a]\,(\neg r\wedge Y)}{\dfrac{{*}{*}{*}\vdash_{(a,X_1\cdot X_2\cdot X_8\cdot X_{10}\cdot a),(Y=X_3)\cdot(X=\phi),\{(a,X_1\cdot X_2\cdot X_4,\neg r)\}}\ \neg r\wedge Y}{\dfrac{\vdash_{(a,X_1\cdot X_2\cdot X_8\cdot X_{10}\cdot a\cdot X_6),(Y=X_3)\cdot(X=\phi),\{(a,X_1\cdot X_2\cdot X_4,\neg r)\}}\ \neg r}{\begin{array}{c}(a,X_1\cdot X_2\cdot X_8\cdot X_{10}\cdot a\cdot X_6,\neg r)\\(a,X_1\cdot X_2\cdot X_4,\neg r)\end{array}}\,\neg r\qquad\dfrac{\vdash_{(a,X_1\cdot X_2\cdot X_8\cdot X_{10}\cdot a\cdot X_6),(Y=X_3)\cdot(X=\phi),\{(a,X_1\cdot X_2\cdot X_4,\neg r)\}}\ Y}{\dfrac{\vdash_{(a,X_1\cdot X_2\cdot X_8\cdot X_{10}\cdot a\cdot X_6\cdot X_4),(Y=X_3)\cdot(X=\phi),\{(a,X_1\cdot X_2\cdot X_4,\neg r)\}}\ [a\ \mathsf{ret}\ a]\,(\neg r\wedge Y)}{\text{-}}\,\mathrm{ret}_0}\,Y\ \mathrm{unf}}\,\wedge}\,\mathrm{ret}_1}\,Y\ \mathrm{unf}}\,\nu Y$$

$$\dfrac{(3)}{\dfrac{\vdash_{(a,X_1)\cdot(a,X_1\cdot X_2\cdot X_8\cdot X_{10}\cdot a)\cdot(a,X_1\cdot X_2\cdot X_8\cdot X_{10}\cdot a),X=\phi,\varnothing_C}\ \nu Y.\,[a\ \mathsf{ret}\ a]\,(\neg r\wedge Y)\wedge[\tau]\,X\wedge[a\ \mathsf{call}\ a]\,X}{\dfrac{\vdash_{(a,X_1\cdot X_2)\cdot(a,X_1\cdot X_2\cdot X_8\cdot X_{10}\cdot a)\cdot(a,X_1\cdot X_2\cdot X_8\cdot X_{10}\cdot a),X=\phi,\varnothing_C}\ \nu Y.\,[a\ \mathsf{ret}\ a]\,(\neg r\wedge Y)}{\dfrac{\vdash_{(a,X_1\cdot X_2)\cdot(a,X_1\cdot X_2\cdot X_8\cdot X_{10}\cdot a)\cdot(a,X_1\cdot X_2\cdot X_8\cdot X_{10}\cdot a),(Y=X_3)\cdot(X=\phi),\varnothing_C}\ Y}{(4)}\,Y\ \mathrm{unf}}\,\nu Y\qquad\dfrac{\vdash_{(a,X_1\cdot X_2)\cdot(a,X_1\cdot X_2\cdot X_8\cdot X_{10}\cdot a)\cdot(a,X_1\cdot X_2\cdot X_8\cdot X_{10}\cdot a),X=\phi,\varnothing_C}\ [\tau]\,X\wedge[a\ \mathsf{call}\ a]\,X}{\text{-}}\,\mathrm{CRep}({*}{*}{*}{*})}\,\wedge}\,X\ \mathrm{unf}$$

$$\dfrac{(4)}{\dfrac{\vdash_{(a,X_1\cdot X_2\cdot X_4)\cdot(a,X_1\cdot X_2\cdot X_8\cdot X_{10}\cdot a)\cdot(a,X_1\cdot X_2\cdot X_8\cdot X_{10}\cdot a),(Y=X_3)\cdot(X=\phi),\varnothing_C}\ [a\ \mathsf{ret}\ a]\,(\neg r\wedge Y)}{\dfrac{\vdash_{(a,X_1\cdot X_2\cdot X_8\cdot X_{10}\cdot a)\cdot(a,X_1\cdot X_2\cdot X_8\cdot X_{10}\cdot a),(Y=X_3)\cdot(X=\phi),\{(a,X_1\cdot X_2\cdot X_4,\neg r)\}}\ \neg r\wedge Y}{\dfrac{\vdash_{(a,X_1\cdot X_2\cdot X_8\cdot X_{10}\cdot a\cdot X_6)\cdot(a,X_1\cdot X_2\cdot X_8\cdot X_{10}\cdot a),(Y=X_3)\cdot(X=\phi),\{(a,X_1\cdot X_2\cdot X_4,\neg r)\}}\ \neg r}{\begin{array}{c}(a,X_1\cdot X_2\cdot X_8\cdot X_{10}\cdot a\cdot X_6,\neg r)\\(a,X_1\cdot X_2\cdot X_8\cdot X_{10}\cdot a,\mathsf{ff})\\(a,X_1\cdot X_2\cdot X_4,\neg r)\end{array}}\,\neg r\qquad\dfrac{\vdash_{(a,X_1\cdot X_2\cdot X_8\cdot X_{10}\cdot a\cdot X_6)\cdot(a,X_1\cdot X_2\cdot X_8\cdot X_{10}\cdot a),(Y=X_3)\cdot(X=\phi),\{(a,X_1\cdot X_2\cdot X_4,\neg r)\}}\ Y}{\dfrac{\vdash_{(a,X_1\cdot X_2\cdot X_8\cdot X_{10}\cdot a\cdot X_6\cdot X_4)\cdot(a,X_1\cdot X_2\cdot X_8\cdot X_{10}\cdot a),(Y=X_3)\cdot(X=\phi),\{(a,X_1\cdot X_2\cdot X_4,\neg r)\}}\ [a\ \mathsf{ret}\ a]\,(\neg r\wedge Y)}{\dfrac{\vdash_{(a,X_1\cdot X_2\cdot X_8\cdot X_{10}\cdot a),(Y=X_3)\cdot(X=\phi),\{(a,X_1\cdot X_2\cdot X_4,\neg r),(a,X_1\cdot X_2\cdot X_8\cdot X_{10}\cdot a\cdot X_6\cdot X_4,\neg r)\}}\ \neg r\wedge Y}{\text{-}}\,\mathrm{RRep}({*}{*}{*})}\,\mathrm{ret}_1}\,Y\ \mathrm{unf}}\,\wedge}\,\mathrm{ret}_1}\,Y\ \mathrm{unf}$$

Figure 6. Tableau for $\nu X.\nu Y.\,[a\ \mathsf{ret}\ a]\,(\neg r\wedge Y)\wedge[\tau]\,X\wedge[a\ \mathsf{call}\ a]\,X$ and a, giving rise to $\{a\Rightarrow\nu X_1.\neg r\wedge[\varepsilon]\,X_1, a\Rightarrow\nu X_1.\,[\varepsilon]\,X_1\wedge[a]\,\neg r\}$

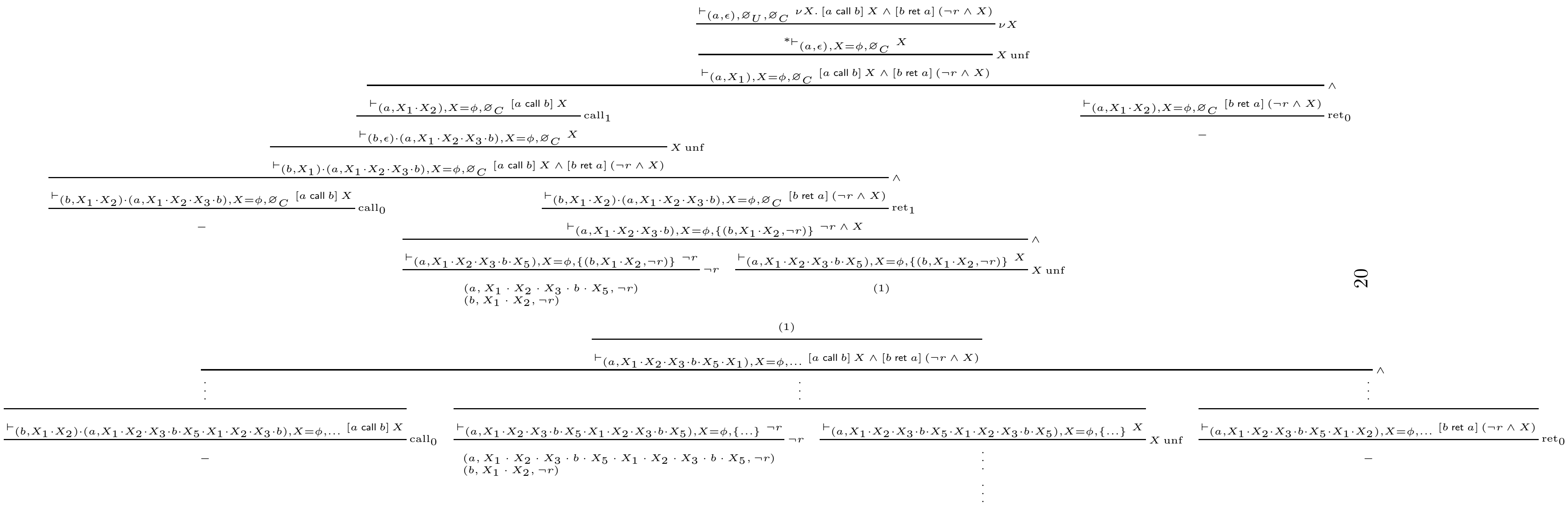

$\vdash_{(a,\epsilon),\varnothing_U,\varnothing_C} \nu X.\, [a\ \mathsf{call}\ b]\, X \wedge [b\ \mathsf{ret}\ a]\, (\neg r \wedge X)$
νX
$^{*}\vdash_{(a,\epsilon),X=\phi,\varnothing_C} X$
X unf
$\vdash_{(a,X_1),X=\phi,\varnothing_C} [a\ \mathsf{call}\ b]\, X \wedge [b\ \mathsf{ret}\ a]\, (\neg r \wedge X)$
$\wedge$
$\vdash_{(a,X_1\cdot X_2),X=\phi,\varnothing_C} [a\ \mathsf{call}\ b]\, X$
call_1
$\vdash_{(a,X_1\cdot X_2),X=\phi,\varnothing_C} [b\ \mathsf{ret}\ a]\, (\neg r \wedge X)$
ret_0
$-$
$\vdash_{(b,\epsilon)\cdot(a,X_1\cdot X_2\cdot X_3\cdot b),X=\phi,\varnothing_C} X$
X unf
$\vdash_{(b,X_1)\cdot(a,X_1\cdot X_2\cdot X_3\cdot b),X=\phi,\varnothing_C} [a\ \mathsf{call}\ b]\, X \wedge [b\ \mathsf{ret}\ a]\, (\neg r \wedge X)$
$\wedge$
$\vdash_{(b,X_1\cdot X_2)\cdot(a,X_1\cdot X_2\cdot X_3\cdot b),X=\phi,\varnothing_C} [a\ \mathsf{call}\ b]\, X$
call_0
$-$
$\vdash_{(b,X_1\cdot X_2)\cdot(a,X_1\cdot X_2\cdot X_3\cdot b),X=\phi,\varnothing_C} [b\ \mathsf{ret}\ a]\, (\neg r \wedge X)$
ret_1
$\vdash_{(a,X_1\cdot X_2\cdot X_3\cdot b),X=\phi,\{(b,X_1\cdot X_2,\neg r)\}} \neg r \wedge X$
$\wedge$
$\vdash_{(a,X_1\cdot X_2\cdot X_3\cdot b\cdot X_5),X=\phi,\{(b,X_1\cdot X_2,\neg r)\}} \neg r$
$\neg r$
$(a,\, X_1 \cdot X_2 \cdot X_3 \cdot b \cdot X_5,\, \neg r)$
$(b,\, X_1 \cdot X_2,\, \neg r)$
$\vdash_{(a,X_1\cdot X_2\cdot X_3\cdot b\cdot X_5),X=\phi,\{(b,X_1\cdot X_2,\neg r)\}} X$
X unf
(1)
(1)
$\vdash_{(a,X_1\cdot X_2\cdot X_3\cdot b\cdot X_5\cdot X_1),X=\phi,\ldots} [a\ \mathsf{call}\ b]\, X \wedge [b\ \mathsf{ret}\ a]\, (\neg r \wedge X)$
$\wedge$
$\vdash_{(b,X_1\cdot X_2)\cdot(a,X_1\cdot X_2\cdot X_3\cdot b\cdot X_5\cdot X_1\cdot X_2\cdot X_3\cdot b),X=\phi,\ldots} [a\ \mathsf{call}\ b]\, X$
call_0
$-$
$\vdash_{(a,X_1\cdot X_2\cdot X_3\cdot b\cdot X_5\cdot X_1\cdot X_2\cdot X_3\cdot b\cdot X_5),X=\phi,\{\ldots\}} \neg r$
$\neg r$
$(a,\, X_1 \cdot X_2 \cdot X_3 \cdot b \cdot X_5 \cdot X_1 \cdot X_2 \cdot X_3 \cdot b \cdot X_5,\, \neg r)$
$(b,\, X_1 \cdot X_2,\, \neg r)$
$\vdash_{(a,X_1\cdot X_2\cdot X_3\cdot b\cdot X_5\cdot X_1\cdot X_2\cdot X_3\cdot b\cdot X_5),X=\phi,\{\ldots\}} X$
X unf
$\vdash_{(a,X_1\cdot X_2\cdot X_3\cdot b\cdot X_5\cdot X_1\cdot X_2),X=\phi,\ldots} [b\ \mathsf{ret}\ a]\, (\neg r \wedge X)$
ret_0
$-$

Figure 7. Unfolding of tableau for $\nu X.\, [a\ \mathsf{call}\ b]\, X \wedge [b\ \mathsf{ret}\ a]\, (\neg r \wedge X)$ and a

Proposition 1. *For all structural formulae χ, χ' and clean frames F*

$$\models_s \chi \Rightarrow [F]\,\chi' \;\Leftrightarrow \models_s \chi_F \Rightarrow \chi'$$

Proof. We prove

$$\models_s \chi \Rightarrow [\alpha]\,\chi' \;\Leftrightarrow \models_s \chi_\alpha \Rightarrow \chi'$$

from which the result follows by a straightforward induction on the length of F. The proof is by well-founded induction on χ, on the usual ordering $\prec$ on the Fisher-Ladner closure of fixed-point formulae [14], where $\phi[\nu X.\phi/X] \prec \nu X.\phi$ and where minimal elements are literals and box formulae. We consider the possible shapes of χ, assuming the equivalence holds for all $\chi'' \prec \chi$.

Case m (base case).

$$\begin{array}{ll}
\models_s (m)_\alpha \Rightarrow \chi' & \\
\Leftrightarrow \models_s m \Rightarrow \chi' & \{\text{Def. } \chi_\alpha\} \\
\Leftrightarrow \models_s m \Rightarrow [\alpha]\,\chi' & \{\text{Prop. } \models_s\}
\end{array}$$

Case $\neg m$ (base case). Similar.

Case r (base case).

$$\begin{array}{ll}
\models_s r_\alpha \Rightarrow \chi' & \\
\Leftrightarrow \models_s \mathsf{ff} \Rightarrow \chi' & \{\text{Def. } \chi_\alpha\} \\
\Leftrightarrow \mathsf{true} & \{\text{Prop. } \models_s\} \\
\Leftrightarrow \models_s r \Rightarrow [\alpha]\,\chi' & \{\text{Clean applet}\}
\end{array}$$

Case $\neg r$ (base case).

$$\begin{array}{ll}
\models_s (\neg r)_\alpha \Rightarrow \chi' & \\
\Leftrightarrow \models_s \mathsf{tt} \Rightarrow \chi' & \{\text{Def. } \chi_\alpha\} \\
\Leftrightarrow \models_s \chi' & \{\text{Prop. } \models_s\} \\
\Leftrightarrow \models_s \neg r \Rightarrow [\alpha]\,\chi' & \{\text{Prop. } \models_s\}
\end{array}$$

Case $\chi_1 \wedge \chi_2$.

$$\begin{array}{ll}
\models_s (\chi_1 \wedge \chi_2)_\alpha \Rightarrow \chi' & \\
\Leftrightarrow \models_s (\chi_1)_\alpha \wedge (\chi_2)_\alpha \Rightarrow \chi' & \{\text{Def. } \chi_\alpha\} \\
\Leftrightarrow \exists \chi_1', \chi_2' : \models_s \chi' \Leftrightarrow \chi_1' \wedge \chi_2'.\ \models_s ((\chi_1)_\alpha \Rightarrow \chi_1') \wedge ((\chi_2)_\alpha \Rightarrow \chi_2') & \{\text{Prop. } \models_s\} \\
\Leftrightarrow \exists \chi_1', \chi_2' : \models_s \chi' \Leftrightarrow \chi_1' \wedge \chi_2'.\ \models_s (\chi_1)_\alpha \Rightarrow \chi_1' \wedge\ \models_s (\chi_2)_\alpha \Rightarrow \chi_2' & \{\text{Prop. } \models_s\} \\
\Leftrightarrow \exists \chi_1', \chi_2' : \models_s \chi' \Leftrightarrow \chi_1' \wedge \chi_2'.\ \models_s \chi_1 \Rightarrow [\alpha]\,\chi_1' \wedge\ \models_s \chi_2 \Rightarrow [\alpha]\,\chi_2' & \{\text{Ind. hyp.}\} \\
\Leftrightarrow \exists \chi_1', \chi_2' : \models_s \chi' \Leftrightarrow \chi_1' \wedge \chi_2'.\ \models_s (\chi_1 \Rightarrow [\alpha]\,\chi_1') \wedge (\chi_2 \Rightarrow [\alpha]\,\chi_2') & \{\text{Prop. } \models_s\} \\
\Leftrightarrow \exists \chi_1', \chi_2' : \models_s [\alpha]\,\chi' \Leftrightarrow [\alpha]\,\chi_1' \wedge [\alpha]\,\chi_2'.\ \models_s (\chi_1 \Rightarrow [\alpha]\,\chi_1') \wedge (\chi_2 \Rightarrow [\alpha]\,\chi_2') & \{\text{Prop. } \models_s\} \\
\Leftrightarrow \exists \chi_1'', \chi_2'' : \models_s [\alpha]\,\chi' \Leftrightarrow \chi_1'' \wedge \chi_2''.\ \models_s (\chi_1 \Rightarrow \chi_1'') \wedge (\chi_2 \Rightarrow \chi_2'') & \{\text{Prop. } \models_s\} \\
\Leftrightarrow \models_s \chi_1 \wedge \chi_2 \Rightarrow [\alpha]\,\chi' & \{\text{Prop. } \models_s\}
\end{array}$$

Case $\chi_1 \vee \chi_2$.

$$\begin{array}{ll}
\models_s (\chi_1 \vee \chi_2)_\alpha \Rightarrow \chi' & \\
\Leftrightarrow \models_s (\chi_1)_\alpha \vee (\chi_2)_\alpha \Rightarrow \chi' & \{\text{Def. } \chi_\alpha\} \\
\Leftrightarrow \models_s ((\chi_1)_\alpha \Rightarrow \chi') \wedge ((\chi_2)_\alpha \Rightarrow \chi') & \{\text{Prop. } \models_s\} \\
\Leftrightarrow \models_s (\chi_1)_\alpha \Rightarrow \chi' \wedge\ \models_s (\chi_2)_\alpha \Rightarrow \chi' & \{\text{Prop. } \models_s\} \\
\Leftrightarrow \models_s \chi_1 \Rightarrow [\alpha]\,\chi' \wedge\ \models_s \chi_2 \Rightarrow [\alpha]\,\chi' & \{\text{Ind. hyp.}\} \\
\Leftrightarrow \models_s (\chi_1 \Rightarrow [\alpha]\,\chi') \wedge (\chi_2 \Rightarrow [\alpha]\,\chi') & \{\text{Prop. } \models_s\} \\
\Leftrightarrow \models_s \chi_{1\chi_b = a \Rightarrow (\neg r \wedge [b]\nu X.[b](\neg r \wedge X))} \vee \chi_2 \Rightarrow [\alpha]\,\chi' & \{\text{Prop. } \models_s\}
\end{array}$$

Case $[\alpha]\,\phi$ (base case).

$\models_s ([\alpha]\,\phi)_\alpha \Rightarrow \chi'$
$\Leftrightarrow \models_s \phi \Rightarrow \chi'$ {Def. χ_α}
$\Leftrightarrow \models_s [\alpha]\,\phi \Rightarrow [\alpha]\,\chi'$ {Prop. $\models_s$}

Case $[\beta]\,\phi$ **where** $\beta \neq \alpha$ (base case).

$\models_s ([\beta]\,\phi)_\alpha \Rightarrow \chi'$
$\Leftrightarrow \models_s \mathsf{tt} \Rightarrow \chi'$ {Def. χ_α}
$\Leftrightarrow \models_s \chi'$ {Prop. $\models_s$}
$\Leftrightarrow \models_s [\beta]\,\phi \Rightarrow [\alpha]\,\chi'$ {Prop. $\models_s$}

Case $\nu X.\phi$.

$\models_s (\nu X.\phi)_\alpha \Rightarrow \chi'$
$\Leftrightarrow \models_s (\phi[\nu X.\phi/X])_\alpha \Rightarrow \chi'$ {Def. χ_α}
$\Leftrightarrow \models_s \phi[\nu X.\phi/X] \Rightarrow [\alpha]\,\chi'$ {Ind. hyp.}
$\Leftrightarrow \models_s \nu X.\phi \Rightarrow [\alpha]\,\chi'$ {Def. $\models_s$}

□

Further, we state two more auxiliary results that are used to relate triples to valid formulae.

Proposition 2. *Let λ be a choice set of a maximal tableau, let χ be the induced structural formula, and let (i, F, q) be a triple in λ, where q is p or $\neg p$ for some $p \in A$. Then $\models_s i \Rightarrow (\chi_{\widetilde{F}} \Rightarrow q)$.*

Proof. (Sketch) If (i, F, q) is a triple in λ, where q is p or $\neg p$ for some $p \in A$, then, by construction of χ, the latter must contain a conjunct $i \Rightarrow \varOmega(\varXi_i)$ where $(F, q) \in \varXi_i$, and therefore, by definition of $\varOmega(\varXi)$ and projection, $\chi_{\widetilde{F}}$ must be of the shape $(i \Rightarrow \nu\overrightarrow{X}.(q \wedge \chi') \wedge \chi'') \wedge \chi'''$ for some (optional) χ', χ'', χ''' and (zero or more) propositional variables $\overrightarrow{X}$. Hence $i \Rightarrow (\chi_{\widetilde{F}} \Rightarrow q)$ is valid in all flow graph structures. □

Example 5. For the tableau in Figure 3, the choice set $\lambda = \{(a, X_4 \cdot X_1 \cdot X_2 \cdot b \cdot X_5, \neg r), (a, X_4 \cdot X_1 \cdot X_2 \cdot b \cdot X_5, X)\}$gives rise to the formula $\chi = a \Rightarrow \nu X.\,[b]\,(\neg r \wedge X)$. Only the first triple in this set is of the form (i, F, q), where q is p or $\neg p$. For this triple, $\tilde{F} = b$, and as mentioned above, $\chi_b = \neg r \wedge \nu X.\,[b]\,(\neg r \wedge X)$. Clearly, $\models_s a \Rightarrow (\chi_b \Rightarrow \neg r)$.

Proposition 3. *Let $\mathcal{T}$ be a maximal tableau and let $(\mathcal{T}^*, \tau)$ be its unfolding. Let n be a leaf node in $\mathcal{T}^*$ mapped by τ to a leaf node n' in $\mathcal{T}$, and let (i, F', q) be a triple in the label of n'. Then, there is a triple (i, F, q) in the label of n, a (possibly empty) index set J so that $X_j \in F'$ for all $j \in J$, a mapping β from indices $j \in J$ to nonempty index sets $\beta(j)$, and a set of rewrite rules $\varDelta = \{X_j \rightarrow X_j \cdot F_{j,k} \cdot X_j \mid j \in J, k \in \beta(j), F_{j,k} \in (I^- \ \cup \{\varepsilon\} \cup \mathcal{C})^*\}$, such that F' can be rewritten to F under $\varDelta$ by applying each rewrite rule at least once.*

Proof. (Sketch) Let $\mathcal{T}$, $(\mathcal{T}^*, \tau)$, n, n' and (i, F', q) be as described above. There are three possible shapes of the sequent by which n' is labelled, depending on which part of the sequent has contributed the triple (i, F', q), namely (i) of shape $\vdash_{(i,F')\cdot H',U',C'} q$, (ii) of shape $\vdash_{H'_1\cdot(i,F')\cdot H'_2,U',C'} q_1$ (but only if $q = \mathsf{ff}$), or finally

(iii) of shape $\vdash_{H',U',C'\cup\{(i,F',q)\}} q_1$ (but only if $q = \neg r$). We shall consider the first case here only; the remaining two cases are handled similarly.

Let n' be labelled by a sequent $\vdash_{(i,F')\cdot H',U',C'} q$. Consider the path from the root node of $\mathcal{T}^*$ to the leaf node n. The mapping τ maps this path to a path in $\mathcal{T}$ (if we identify in $\mathcal{T}$ repeat nodes with their companions). Then, let X_j be the constants in F' for which n' has ancestor nodes $n'_j :\vdash_{(i,F'_j)\cdot H',U'_j,C'_j} \mathcal{S}^{-1}(X_j)$ that are companions to internal repeats $n'_{j,k} :\vdash_{(i,F'_j\cdot X_j\cdot F_{j,k})\cdot H',U'_{j,k},C'_{j,k}} \mathcal{S}^{-1}(X_j)$ visited along the latter path, where F'_j is the prefix of F' up to the occurrence of X_j, and let every such repeat $n'_{j,k}$ contribute a production $X_j \to X_j \cdot F_{j,k} \cdot X_j$ in Δ (and thus a frame fragment $F_{j,k}$ in $\beta(j)$). Every repeat $n'_{j,k}$ contains a triple $(i, F'_j \cdot X_j \cdot F_{j,k}, X_j)$ in its label; the result then follows by virtue of tableau construction and unfolding. □

Example 6. Consider the tableau unfolding displayed in Figure 7. The leaf node $n \vdash_{(a,X_1\cdot X_2\cdot X_3\cdot b\cdot X_5\cdot X_1\cdot X_2\cdot X_3\cdot b\cdot X_5),X=\phi,\{(b,X_1\cdot X_2,\neg r)\}} \neg r$ maps to the leaf node n' $\vdash_{(a,X_1\cdot X_2\cdot X_3\cdot b\cdot X_5),X=\phi,\{(b,X_1\cdot X_2,\neg r)} \neg r$ in the tableau in Figure 3. The triple $(b, X_1 \cdot X_2, \neg r)$ of n' is trivially written (by the identity rewrite rule) into a triple of n. The triple $(a, X_1 \cdot X_2 \cdot X_3 \cdot b \cdot X_5, \neg r)$ of n' is rewritten into the triple $(a, X_1 \cdot X_2 \cdot X_3 \cdot b \cdot X_5 \cdot X_1 \cdot X_2 \cdot X_3 \cdot b \cdot X_5, \neg r)$ of n by the rewrite rule $X_1 \to X_1 \cdot X_2 \cdot X_3 \cdot b \cdot X_5 \cdot X_1$.

Lemma 1. *Let $\mathcal{T}$ be a maximal tableau, let $(\mathcal{T}^*, \tau)$ be its unfolding, and let χ be one of the structural formulae induced by $\mathcal{T}$. Then, in the label of every contributing leaf of $\mathcal{T}^*$ there is a triple (i, F, q) such that for all flow graphs $\mathcal{G}$, $\mathcal{G} \models_s \chi$ implies $\mathcal{G} \models_s i \Rightarrow \left[\widetilde{F}\right] q$.*

Proof. (Sketch) Let n be a leaf node of the unfolding $\mathcal{T}^*$, let τ map this leaf to a leaf n' in $\mathcal{T}$, and let (i, F', q) be the triple in the label of n' that is in the choice set λ inducing χ. By Proposition 3, there is a triple (i, F, q) in the label of n, a (possibly empty) index set J so that $X_j \in F'$ for all $j \in J$, a mapping β from indices $j \in J$ to nonempty index sets $\beta(j)$, and a set of rewrite rules $\Delta = \{X_j \to X_j \cdot F_{j,k} \cdot X_j \mid j \in J, k \in \beta(j), F_{j,k} \in (I^- \cup \{\varepsilon\} \cup \mathcal{C})^*\}$, such that F' can be rewritten to F under Δ by applying each rewrite rule at least once. In the case of $\Delta = \varnothing$, and hence $F = F'$, the result holds by Proposition 2. Consider the remaining case when $\Delta \neq \varnothing$. Again, we consider here only the case of n' being labelled by a sequent of shape $\vdash_{(i,F')\cdot H',U',C'} q$.

Let nodes n'_j and $n'_{j,k}$ be as described in the proof of Proposition 3. If for any such node $n'_{j,k}$ the triple $(i, F'_j \cdot X_j \cdot F_{j,k}, X_j)$ is not in the choice set λ, then λ must have selected at $n'_{j,k}$ some triple from the store $C'_{j,k}$, which must then also be in the store of the sequent at node n, and hence in the set of triples in the label of n; the result then follows immediately from Proposition 2.

So, consider the remaining case when λ has chosen from the label of every node $n'_{j,k}$ exactly the triple $(i, F'_j \cdot X_j \cdot F_{j,k}, X_j)$. Then, by the definitions of induced formula and projection, for all $j \in J$ and $k \in \beta(j)$, the structural formula $(\Omega(\Xi_i))_{\widetilde{F'_j}}$ must be identical to $(\Omega(\Xi_i))_{\widetilde{F'_j}\cdot\widetilde{F_{j,k}}}$ up to repeating or trivial conjuncts. By induction on the length of the rewriting sequence deriving F from

F', $(\mathit{\Omega}(\mathit{\Xi}_i))_{\widetilde{F}}$ must be identical to $(\mathit{\Omega}(\mathit{\Xi}_i))_{\widetilde{F'}}$ up to repeating or trivial conjuncts. Since by assumption (i, F', q) is in λ, $(\mathit{\Omega}(\mathit{\Xi}_i))_{\widetilde{F'}}$ must be of shape $\nu\overrightarrow{X}.(q \wedge \chi') \wedge \chi''$ for some (optional) χ', χ'' and (zero or more) propositional variables $\overrightarrow{X}$, and hence also $(\mathit{\Omega}(\mathit{\Xi}_i))_{\widetilde{F}}$ must be of this shape. Therefore $i \Rightarrow (\chi_{\widetilde{F}} \Rightarrow q)$ is valid in all flow graph structures, and thus, because of Proposition 1, we can conclude that $\mathcal{G} \models_s \chi$ implies $\mathcal{G} \models_s i \Rightarrow \left[\widetilde{F}\right] q$ for all flow graphs $\mathcal{G}$. □

Example 7. Consider again the tableau unfolding in 7. All contributing leaves are of the shape$\vdash_{(a, X_1 \cdot (X_2 \cdot X_3 \cdot b \cdot X_5 \cdot X_1)^* \cdot X_2 \cdot X_3 \cdot b \cdot X_5, X = \phi, \{(b, X_1 \cdot X_2, \neg r)\}} \neg r$, *i.e.*, they contain triples $(a, X_1 \cdot (X_2 \cdot X_3 \cdot b \cdot X_5 \cdot X_1)^* \cdot X_2 \cdot X_3 \cdot b \cdot X_5, \neg r$ and $(b, X_1 \cdot X_2, \neg r)$. Thus, for the first triple we have that $F = b^+$, *i.e.*, it contains one or more b's. Consider the induced formula $\chi = a \Rightarrow \nu X. [b] (\neg r \wedge X)$. Clearly, for this formula we have that for any flow graph $\mathcal{G}$: $\mathcal{G} \models_s \chi$ implies $\mathcal{G} \models_s a \Rightarrow [b^+] \neg r$.

3.3 Correctness of the Translation

We use a proof system and an isomorphic translation of tableau unfoldings into proof trees to show correctness of the tableau system. We show that the proof tree induced by the unfolding of the tableau for behavioural formula ϕ and method m generating the set of structural formulae $\mathcal{X}$ constitutes a proof that every flow graph satisfying some $\chi \in \mathcal{X}$ also satisfies ϕ.

Proof System The sequents of the proof system are of shape $\mathcal{X} \vdash_{(i,F) \cdot H, U, C} \phi$. To define validity of a sequent, we first define a function reach that computes for a given flow graph $\mathcal{G}$, set of nodes V and sequence of labels L, the set of nodes reachable in $\mathcal{G}$ from the nodes in V by following edges with the labels in L (via non-return nodes).

$$\mathsf{reach}_{\mathcal{G}}(V, \epsilon) = V \qquad \mathsf{reach}_{\mathcal{G}}(V, l \cdot L) = \mathsf{reach}_{\mathcal{G}}(\{v' \mid v \in V \wedge v \xrightarrow{l}_{\mathcal{G}} v'\}, L)$$

For clean flow graphs, reachability coincides with a structural box formula.

Proposition 4. *For all flow graphs $\mathcal{G}$, method names m, clean frames F and structural formulae χ:*

$$\mathcal{G} \models_s m \Rightarrow [F] \chi \;\Leftrightarrow\; \forall v \in \mathsf{reach}_{\mathcal{G}}(E_m, F).\, v \models_s^{\mathcal{G}} \chi$$

One can easily show $\forall v \in V.v \models_s^{\mathcal{G}} [L] \chi \Leftrightarrow \forall v' \in \mathsf{reach}_{\mathcal{G}}(V, L).v' \models_s^{\mathcal{G}} \chi$ by induction on the length of L. The result then follows, since by definition and simple logic $\mathcal{G} \models_s m \Rightarrow \phi \Leftrightarrow \forall v \in E_m.v \models_s^{\mathcal{G}} \phi$, by instantiating L with F. □

Next, we define a *matching* predicate $\gamma_{\mathcal{G}}(\sigma, H)$ relating history stacks H and call stacks σ. Note that $\gamma_{\mathcal{G}}(v \cdot \sigma, \varnothing_{H,m})$ holds exactly when $v \in E_m$ and $\sigma = \epsilon$.

$$\gamma_{\mathcal{G}}(\epsilon, \epsilon) \Leftrightarrow \mathsf{tt} \quad \gamma_{\mathcal{G}}(v \cdot \sigma, \epsilon) \Leftrightarrow \mathsf{ff} \quad \gamma_{\mathcal{G}}(\epsilon, (m, F) \cdot H) \Leftrightarrow \mathsf{ff}$$
$$\gamma_{\mathcal{G}}(v \cdot \sigma, (m, F) \cdot H) \Leftrightarrow v \in \mathsf{reach}_{\mathcal{G}}(E_m, \widetilde{F}) \wedge \gamma_{\mathcal{G}}(\sigma, H)$$

$$\textsf{Prop}\ \frac{\mathcal{X} \vdash_{(i,F)\cdot H,U,C} p}{\mathcal{V}(\{(i,F,p)\} \cup \{(i',F',\mathsf{ff}) \mid (i',F') \in H\} \cup C, \mathcal{X})} \qquad \textsf{NegProp}\ \frac{\mathcal{X} \vdash_{(i,F)\cdot H,U,C} \neg p}{\mathcal{V}(\{(i,F,\neg p)\} \cup \{(i',F',\mathsf{ff}) \mid (i',F') \in H\} \cup C, \mathcal{X})}$$

$$\textsf{NuX}\ \frac{\mathcal{X} \vdash_{(i,F)\cdot H,U,C} \nu X.\phi}{\mathcal{X} \vdash_{(i,F)\cdot H,(X=\nu X.\phi)\circ U,C} X} \qquad \textsf{XUnf}\ \frac{\mathcal{X} \vdash_{(i,F)\cdot H,U,C} X}{\mathcal{X} \vdash_{(i,F\cdot \mathcal{S}(X))\cdot H,U,C} \phi}\ (X = \nu X.\phi) \in U$$

$$\textsf{And}\ \frac{\mathcal{X} \vdash_{(i,F)\cdot H,U,C} \phi_1 \wedge \phi_2}{\mathcal{X} \vdash_{(i,F\cdot \mathcal{S}(\phi_1 \wedge \phi_2))\cdot H,U,C} \phi_1 \qquad \mathcal{X} \vdash^{\mathcal{A}}_{(i,F\cdot \mathcal{S}(\phi_1 \wedge \phi_2))\cdot H,U,C} \phi_2} \qquad \textsf{Tau}\ \frac{\mathcal{X} \vdash_{(i,F)\cdot H,U,C} [\tau]\phi}{\mathcal{X} \vdash_{(i,F\cdot \mathcal{S}([\tau]\phi)\cdot \varepsilon)\cdot H,U,C} \phi}$$

$$\textsf{Call}_0\ \frac{\mathcal{X} \vdash_{(i,F)\cdot H,U,C} [a\ \textsf{call}\ b]\phi}{-}\ i \neq a \qquad \textsf{Call}_1\ \frac{\mathcal{X} \vdash_{(i,F)\cdot H,U,C} [a\ \textsf{call}\ b]\phi}{\mathcal{X} \vdash_{(b,\epsilon)\cdot (i,F\cdot \mathcal{S}([a\ \textsf{call}\ b]\phi)\cdot b)\cdot H,U,C} \phi}\ i = a$$

$$\textsf{Ret}_0\ \frac{\mathcal{X} \vdash_{(i,F)\cdot H,U,C} [a\ \textsf{ret}\ b]\phi}{-}\ \neg\mathrm{Ret}(i,a,b,H) \qquad \textsf{Ret}_1\ \frac{\mathcal{X} \vdash_{(i,F)\cdot H,U,C} [a\ \textsf{ret}\ b]\phi}{\mathcal{X} \vdash_{H,U,C \cup \{(i,F,\neg r)\}} \phi}\ \mathrm{Ret}(i,a,b,H)$$

Figure 8. Proof system

Finally, for a set C of structural constraints of shape (i, F, q), where q is p or $\neg p$, let $\mathcal{V}(C, \mathcal{G})$ abbreviate $\exists (i, F, q) \in C.\, \mathcal{G} \models_s i \Rightarrow \left[\widetilde{F}\right] q$, $\mathcal{V}(C, \chi)$ abbreviate $\forall \mathcal{G}.\, (\mathcal{G} \models_s \chi \Rightarrow \mathcal{V}(C, \mathcal{G}))$, and $\mathcal{V}(C, \mathcal{X})$ abbreviate $\forall \chi \in \mathcal{X}.\, \mathcal{V}(C, \chi)$.

We are now ready to define validity of sequents.

Definition 9. (Sequent validity) *Let $\mathcal{X}$ a set of structural formulae, and ϕ a behavioural formula. Sequent $\mathcal{X} \vdash_{(i,F)\cdot H,U,C} \phi$ is* valid*, denoted $\mathcal{X} \models_{(i,F)\cdot H,U,C} \phi$, iff for every $\chi \in \mathcal{X}$ and every flow graph $\mathcal{G}$ such that $\mathcal{G} \models_s \chi$,*

$$\mathcal{V}(C, \mathcal{G}) \vee \forall v, \sigma.\, (\gamma_{\mathcal{G}}(v \cdot \sigma, (i, F) \cdot H) \Rightarrow (v, \sigma) \models^{\mathcal{G}}_{b} \phi[U])$$

Figure 8 gives the proof system as a set of goal-directed rules. Notice that there is a one-to-one correspondence between proof rules and tableau rules used in tableau unfolding, where the tableau rules giving rise to sets of triples C correspond to axiom rules giving rise to a *proof obligation* of shape $\mathcal{V}(C, \mathcal{X})$.

A (potentially infinite) proof tree is a *proof* if (*cf.* [22]):

1. all leaves are axioms giving rise to valid proof obligations; and
2. the *global discharge condition* holds: along every infinite path, some propositional variable X is unfolded infinitely often.

Notice that in our proof system the second condition holds trivially.

The proof system is *locally sound*, in the sense that all local rules of the proof system are sound (in the standard sense), and *locally complete*, in the sense that if a sequent is valid, then a rule is applicable (backwards) to this sequent that yields valid sequents only.

Lemma 2. *The proof system is locally sound and complete.*

Proof. We consider each rule in turn, implicitly quantifying over all $\chi \in \mathcal{X}$. We present here the more interesting cases only.

Rule Prop.

$$
\begin{aligned}
&\quad \chi \models_{(i,F)\cdot H,U,C} p \\
&\Leftrightarrow \forall \mathcal{G}.\, (\mathcal{G} \models_s \chi \;\Rightarrow && \{\text{Def. } 9\} \\
&\qquad \forall v, \sigma.\, (\gamma_{\mathcal{G}}(v \cdot \sigma, (i,F) \cdot H) \Rightarrow (v,\sigma) \models^{\mathcal{G}}_{b} p) \,\vee\, \mathcal{V}(C,\mathcal{G})) \\
&\Leftrightarrow \forall \mathcal{G}.\, (\mathcal{G} \models_s \chi \;\Rightarrow && \{\text{Def. } \gamma_{\mathcal{G}}\} \\
&\qquad \forall v, \sigma.\, (v \in \mathsf{reach}_{\mathcal{G}}(E_i, \widetilde{F}) \wedge \gamma_{\mathcal{G}}(\sigma, H) \Rightarrow (v,\sigma) \models^{\mathcal{G}}_{b} p) \,\vee\, \mathcal{V}(C,\mathcal{G})) \\
&\Leftrightarrow \forall \mathcal{G}.\, (\mathcal{G} \models_s \chi \;\Rightarrow && \left\{\text{Def. } \models^{\mathcal{G}}_{b}\right\} \\
&\qquad \forall v, \sigma.\, (v \in \mathsf{reach}_{\mathcal{G}}(E_i, \widetilde{F}) \wedge \gamma_{\mathcal{G}}(\sigma, H) \Rightarrow v \models^{\mathcal{G}}_{s} p) \,\vee\, \mathcal{V}(C,\mathcal{G})\, Proof.Proof.) \\
&\Leftrightarrow \forall \mathcal{G}.\, (\mathcal{G} \models_s \chi \;\Rightarrow && \{\text{Logic}\} \\
&\qquad \forall v \in \mathsf{reach}_{\mathcal{G}}(E_i, \widetilde{F}).\, v \models^{\mathcal{G}}_{s} p \,\vee\, \forall \sigma.\, \neg \gamma_{\mathcal{G}}(\sigma, H) \,\vee\, \mathcal{V}(C,\mathcal{G})) \\
&\Leftrightarrow \forall \mathcal{G}.\, (\mathcal{G} \models_s \chi \;\Rightarrow && \{\text{Def. } \gamma_{\mathcal{G}}\} \\
&\qquad \forall v \in \mathsf{reach}_{\mathcal{G}}(E_i, \widetilde{F}).\, v \models^{\mathcal{G}}_{s} p \,\vee \\
&\qquad \exists (i', F') \in H.\, \forall v \in \mathsf{reach}_{\mathcal{G}}(E_{i'}, \widetilde{F'}).\, v \models^{\mathcal{G}}_{s} \mathsf{ff} \vee \mathcal{V}(C,\mathcal{G})) \\
&\Leftrightarrow \forall \mathcal{G}.\, (\mathcal{G} \models_s \chi \;\Rightarrow && \{\text{Prop. } 4\} \\
&\qquad \mathcal{G} \models_s i \Rightarrow \left[\widetilde{F}\right] p \,\vee \\
&\qquad \exists (i', F') \in H.\, \mathcal{G} \models_s i' \Rightarrow \left[\widetilde{F'}\right] \mathsf{ff} \vee \mathcal{V}(C,\mathcal{G})) \\
&\Leftrightarrow \mathcal{V}(\{(i,F,p)\} \cup \{(i',F',\mathsf{ff}) \mid (i',F') \in H\} \cup C, \chi) && \{\text{Def. } \mathcal{V}(C,\chi)\}
\end{aligned}
$$

Rule NuX.

$$
\begin{aligned}
&\quad \chi \models_{(i,F)\cdot H,U,C} \nu X.\phi \\
&\Leftrightarrow \forall \mathcal{G}.\, (\mathcal{G} \models_s \chi \;\Rightarrow && \{\text{Def.}\, 9\} \\
&\qquad \forall v, \sigma.(\gamma_{\mathcal{G}}(v \cdot \sigma, (i,F) \cdot H) \Rightarrow (v,\sigma) \models^{\mathcal{G}}_{b} (\nu X.\phi)[U]) \,\vee\, \mathcal{V}(C,\mathcal{G})) \\
&\Leftrightarrow \forall \mathcal{G}.\, (\mathcal{G} \models_s \chi \;\Rightarrow && \{\text{Def. } \phi[U]\} \\
&\qquad \forall v, \sigma.(\gamma_{\mathcal{G}}(v \cdot \sigma, (i,F) \cdot H) \Rightarrow (v,\sigma) \models^{\mathcal{G}}_{b} X[(X = \nu X.\phi) \cdot U]) \,\vee\, \mathcal{V}(C,\mathcal{G})) \\
&\Leftrightarrow \chi \models_{(i,F)\cdot H,(X=\nu X.\phi)\cdot U,C} X && \{\text{Def.}\, 9\}
\end{aligned}
$$

Rule Call$_1$.

Let $i = a$.

$$
\begin{aligned}
&\quad \chi \models_{(i,F)\cdot H,U,C} [a\ \mathsf{call}\ b]\, \phi \\
&\Leftrightarrow \forall \mathcal{G}.\, (\mathcal{G} \models_s \chi \;\Rightarrow && \{\text{Def.}\, 9, \gamma_{\mathcal{G}}\} \\
&\qquad \forall v, \sigma.\, (v \in \mathsf{reach}_{\mathcal{G}}(E_i, \widetilde{F}) \wedge \gamma_{\mathcal{G}}(\sigma, H) \;\Rightarrow \\
&\qquad\qquad (v,\sigma) \models^{\mathcal{G}}_{b} [a\ \mathsf{call}\ b]\, \phi[U]) \,\vee\, \mathcal{V}(C,\chi)) \\
&\Leftrightarrow \forall \mathcal{G}.\, (\mathcal{G} \models_s \chi \;\Rightarrow && \left\{\text{Def. } \models^{\mathcal{G}}_{b}\right\} \\
&\qquad \forall v, \sigma.\, (v \in \mathsf{reach}_{\mathcal{G}}(E_i, \widetilde{F}) \wedge \gamma_{\mathcal{G}}(\sigma, H) \;\Rightarrow \\
&\qquad\qquad \forall v_1, v_2.(v \xrightarrow{b}_{a} v_1 \wedge v_2 \in E_b \Rightarrow (v_2, v_1 \cdot \sigma) \models^{\mathcal{G}}_{b} \phi[U])) \,\vee\, \mathcal{V}(C,\chi)) \\
&\Leftrightarrow \forall \mathcal{G}.\, (\mathcal{G} \models_s \chi \;\Rightarrow && \{\text{Logic}\} \\
&\qquad \forall v, v_1, v_2, \sigma.\, (v \in \mathsf{reach}_{\mathcal{G}}(E_i, \widetilde{F}) \wedge \gamma_{\mathcal{G}}(\sigma, H) \wedge v \xrightarrow{b}_{a} v_1 \wedge v_2 \in E_b \;\Rightarrow \\
&\qquad\qquad (v_2, v_1 \cdot \sigma) \models^{\mathcal{G}}_{b} \phi[U])) \,\vee\, \mathcal{V}(C,\chi)) \\
&\Leftrightarrow \forall \mathcal{G}.\, (\mathcal{G} \models_s \chi \;\Rightarrow && \{\text{Def. } \mathsf{reach}_{\mathcal{G}}, \tilde{}\} \\
&\qquad \forall v_1, v_2, \sigma.\, (v_1 \in \mathsf{reach}_{\mathcal{G}}(E_i, F \cdot \mathcal{S}(\widetilde{[a\ \mathsf{call}\ b]\, \phi}) \cdot b) \wedge \\
&\qquad\qquad \gamma_{\mathcal{G}}(\sigma, H) \wedge v_2 \in \mathsf{reach}_{\mathcal{G}}(E_b, \epsilon) \;\Rightarrow \\
&\qquad\qquad (v_2, v_1 \cdot \sigma) \models^{\mathcal{G}}_{b} \phi[U])) \,\vee\, \mathcal{V}(C,\chi)) \\
&\Leftrightarrow \chi \models_{(b,\epsilon)\cdot(i,F\cdot\mathcal{S}([a\ \mathsf{call}\ b]\phi)\cdot b)\cdot H,U,C} \phi && \{\text{Def.}\, 9, \gamma_{\mathcal{G}}\}
\end{aligned}
$$

Rule Ret_1**.**

Let $\mathrm{Ret}(i, a, b, H)$. We use $\sigma^\sharp$ and $\sigma^\flat$ to denote the head and tail of σ, respectively.

$$\begin{array}{ll}
\quad \chi \models_{(i,F)\cdot H,U,C} [a\ \mathsf{ret}\ b]\, \phi & \\
\Leftrightarrow \forall \mathcal{G}.\, (\mathcal{G} \models_s \chi \ \Rightarrow & \{\mathrm{Def.}\, 9, \gamma_{\mathcal{G}}\} \\
\qquad \forall v, \sigma.\, (v \in \mathsf{reach}_{\mathcal{G}}(E_i, \widetilde{F}) \wedge \gamma_{\mathcal{G}}(\sigma, H) \ \Rightarrow & \\
\qquad\qquad (v, \sigma) \models_b^{\mathcal{G}} [a\ \mathsf{ret}\ b]\, \phi[U]) \ \vee\ \mathcal{V}\,(C, \chi)) & \\
\Leftrightarrow \forall \mathcal{G}.\, (\mathcal{G} \models_s \chi \ \Rightarrow & \left\{\mathrm{Def.}\ \models_b^{\mathcal{G}}\right\} \\
\qquad \forall v, \sigma.\, (v \in \mathsf{reach}_{\mathcal{G}}(E_i, \widetilde{F}) \wedge \gamma_{\mathcal{G}}(\sigma, H) \ \Rightarrow & \\
\qquad\qquad v \models_s^{\mathcal{G}} (r \wedge a) \wedge \sigma \neq \epsilon \wedge \sigma^\sharp \models_s^{\mathcal{G}} b \Rightarrow (\sigma^\sharp, \sigma^\flat) \models_b^{\mathcal{G}} \phi[U]) \ \vee\ \mathcal{V}\,(C, \chi)) & \\
\Leftrightarrow \forall \mathcal{G}.\, (\mathcal{G} \models_s \chi \ \Rightarrow & \{\mathrm{Ret}(i, a, b, H)\} \\
\qquad \forall v.\, (v \in \mathsf{reach}_{\mathcal{G}}(E_i, \widetilde{F}) \Rightarrow v \models_s^{\mathcal{G}} \neg r) \vee & \\
\qquad \forall \sigma.\, (\gamma_{\mathcal{G}}(\sigma, H) \Rightarrow (\sigma^\sharp, \sigma^\flat) \models_b^{\mathcal{G}} \phi[U]) \ \vee\ \mathcal{V}\,(C, \chi)) & \\
\Leftrightarrow \forall \mathcal{G}.\, (\mathcal{G} \models_s \chi \ \Rightarrow & \{\mathrm{Prop.}\ 4\} \\
\qquad \mathcal{G} \models_s i \Rightarrow \left[\widetilde{F}\right] \neg r \vee & \\
\qquad \forall \sigma.\, (\gamma_{\mathcal{G}}(\sigma, H) \Rightarrow (\sigma^\sharp, \sigma^\flat) \models_b^{\mathcal{G}} \phi[U]) \ \vee\ \mathcal{V}\,(C, \chi)) & \\
\Leftrightarrow \forall \mathcal{G}.\, (\mathcal{G} \models_s \chi \ \Rightarrow\ \forall \sigma.\, (\gamma_{\mathcal{G}}(\sigma, H) \Rightarrow (\sigma^\sharp, \sigma^\flat) \models_b^{\mathcal{G}} \phi[U]) \ \vee\ \mathcal{V}\,(C \cup \{(i, F, \neg r)\}, \chi)) & \{\mathrm{Def.}\ \mathcal{V}\,(C, \chi)\} \\
\Leftrightarrow \chi \models_{H,U,C\cup\{(i,F,\neg r)\}} \phi & \{\mathrm{Def.}\, 9, \gamma_{\mathcal{G}}\}
\end{array}$$

□

The proof system is sound in the standard sense.

Theorem 2. *$\mathcal{X} \models_{\varnothing_{H,m},\varnothing_U,\varnothing_C} \phi$ if there is a proof with root $\mathcal{X} \vdash_{\varnothing_{H,m},\varnothing_U,\varnothing_C} \phi$.*

Proof. Assume on the contrary that there is a proof with root $\mathcal{X} \vdash_{\varnothing_{H,m},\varnothing_U,\varnothing_C} \phi$ but $\mathcal{X} \models_{\varnothing_{H,m},\varnothing_U,\varnothing_C} \phi$ does not hold. From local soundness of the proof system it follows that there must be an infinite path in the proof starting from the root and consisting of invalid sequents only, and along this path, some propositional variable is unfolded infinitely often.

Define the *signature* of a sequent as the least mapping from propositional variables to ordinals for which the approximated fixed-point formula makes the sequent invalid (*cf.* [23]). Formally, pick a total ordering $\prec_\phi$ on the propositional variables V_ϕ occurring in ϕ so that $X \prec_\phi Y$ whenever $X \neq Y$ and νY is in the scope of νX in ϕ. This ordering induces a well-founded, lexicographic ordering on the set of mappings of type $\kappa : V_\phi \to \mathit{Ord}$, where Ord denotes the set of ordinals. Let $(\phi[U])^\kappa$ denote the fixed-point approximant of $\phi[U]$ induced by κ. Then, the signature of $\mathcal{X} \vdash_{(i,F)\cdot H,U,C} \phi$ is defined as the least $\kappa : V \to \mathit{Ord}$ for which there exists a structural formula $\chi \in \mathcal{X}$ and a control flow graph $\mathcal{G}$ so that $\mathcal{G} \models_s \chi$, with node v and stack σ so that $\gamma_{\mathcal{G}}(v \cdot \sigma, (i, F) \cdot H)$, for which $(v, \sigma) \not\models_b^{\mathcal{G}} (\phi[U])^\kappa$.

The signature decreases along the path every time unfolding is applied, since ϕ is guarded, and is preserved by all other proof rules. Hence, the signature must decrease infinitely often along the path. This, however, is impossible, and hence the initial assumption must be wrong. Therefore, the proof system is sound. □

Soundness and Completeness of Translation The tableau for behavioural formula ϕ and method m giving rise to the set of structural formulae $\mathcal{X}$ induces, through its unfolding, a proof tree with root $\mathcal{X} \vdash_{\varnothing_{H,m},\varnothing_U,\varnothing_C} \phi$ by applying the corresponding rules. This proof tree constitutes a proof.

Lemma 3. *Maximal tableaux induce proofs.*

Proof. Let $\mathcal{T}$ be a tableau for behavioural formula ϕ and method m inducing $\mathcal{X}$, let $(\mathcal{T}^*, \tau)$ be its unfolding, and let P be the induced proof tree with root $\mathcal{X} \vdash_{\varnothing_{H,m},\varnothing_U,\varnothing_C} \phi$. By Lemma 1, for every leaf of P labelled by a proof obligation $\mathcal{V}(C, \mathcal{X})$, and for every $\chi \in \mathcal{X}$, there is a triple (i, F, q) in C such that $\mathcal{G} \models_s \chi$ implies $\mathcal{G} \models_s i \Rightarrow \left[\widetilde{F}\right] q$ for all flow graphs $\mathcal{G}$. But:

$$\begin{array}{ll}
\quad \forall \mathcal{G}.\, (\mathcal{G} \models_s \chi \;\Rightarrow\; \mathcal{G} \models_s i \Rightarrow \left[\widetilde{F}\right] q) & \\
\Rightarrow \forall \mathcal{G}.\, (\mathcal{G} \models_s \chi \;\Rightarrow\; \mathcal{V}(C, \mathcal{G})) & \{\text{Definition } \mathcal{V}(C, \mathcal{G})\} \\
\Leftrightarrow \mathcal{V}(C, \chi) & \{\text{Definition } \mathcal{V}(C, \chi)\}
\end{array}$$

and therefore the proof obligation $\mathcal{V}(C, \mathcal{X})$ is valid. Furthermore, the global discharge condition holds by virtue of the tableau construction and tableau unfolding. Therefore P constitutes a proof. □

The translation from behavioural to structural formulae is *sound*, *i.e.,* every flow graph satisfying a structural formula χ induced by a behavioural formula ϕ by means of the translation Π defined above, also satisfies ϕ.

Theorem 3. *Translation Π from behavioural to structural formulae is sound.*

Proof. Let ϕ be a behavioural formula, and let $\chi \in \Pi_{I_{\mathcal{G}}}(\phi)$. Then, by the translation, for every method $m \in I^+$ there is a conjunct χ_m of χ induced by a maximal tableau for ϕ and m. We therefore have:

$$\begin{array}{ll}
\quad \mathcal{G} \models_s \chi & \\
\Rightarrow \forall m \in I^+.\, \exists \chi_m \in \pi_m(\phi).\, (\mathcal{G} \models_s \chi_m \;\wedge\; \chi_m \vdash_{\varnothing_{H,m},\varnothing_U,\varnothing_C} \phi) & \{\text{Lemma 3}\} \\
\Rightarrow \forall m \in I^+.\, \exists \chi_m \in \pi_m(\phi).\, (\mathcal{G} \models_s \chi_m \;\wedge\; \chi_m \models_{\varnothing_{H,m},\varnothing_U,\varnothing_C} \phi) & \{\text{Theorem 2}\} \\
\Rightarrow \forall m \in I^+.\, \forall v \in E_m.\, (v, \epsilon) \models_b^{\mathcal{G}} \phi & \{\text{Definition 9}\} \\
\Leftrightarrow \mathcal{G} \models_b \phi & \{\text{Definition } \models_b\}
\end{array}$$

and hence the translation is sound. □

The translation is *complete*, *i.e.,* every flow graph satisfying a behavioural formula ϕ also satisfies a structural formula χ induced by ϕ by means of Π.

Theorem 4. *Translation Π from behavioural to structural formulae is complete.*

Proof. (Sketch) First, we show that for every structural formula χ and method m,

$$\chi \models_{\varnothing_{H,m},\varnothing_U,\varnothing_C} \phi \;\Rightarrow\; \forall \mathcal{G}.\, (\mathcal{G} \models_s \chi \Rightarrow \exists \chi' \in \pi_m(\phi).\, \mathcal{G} \models_s \chi') \tag{2}$$

Then, we use the notion of *characteristic formula* of a specification $\mathcal{S}$, denoted $\chi_{\mathcal{S}}$, and its properties (*viz.* Theorem 2.7, Theorem 3.9, and Corollary 2.16 in [21]), to obtain:

$$\begin{array}{ll}
\mathcal{G} \models_b \phi & \\
\Leftrightarrow \forall \mathcal{G}'.\, (\mathcal{G}' \models_s \chi_{\mathcal{G}} \Rightarrow \mathcal{G}' \models_b \phi) & \{\text{Properties } \chi_{\mathcal{G}}\} \\
\Leftrightarrow \forall \mathcal{G}'.\, (\mathcal{G}' \models_s \chi_{\mathcal{G}} \Rightarrow \forall m \in I^+.\, \forall v \in E_m.\, (v, \epsilon) \models_b^{\mathcal{G}'} \phi) & \{\text{Definition } \models_b\} \\
\Leftrightarrow \forall m \in I^+.\, \forall \mathcal{G}'.\, (\mathcal{G}' \models_s \chi_{\mathcal{G}} \Rightarrow \forall v \in E_m.\, (v, \epsilon) \models_b^{\mathcal{G}'} \phi) & \{\text{Logic}\} \\
\Leftrightarrow \forall m \in I^+.\, \chi_{\mathcal{G}} \models_{\varnothing_{H,m},\varnothing_U,\varnothing_C} \phi & \{\text{Definition 9}\} \\
\Rightarrow \forall m \in I^+.\, \forall \mathcal{G}'.\, (\mathcal{G}' \models_s \chi_{\mathcal{G}} \Rightarrow \exists \chi' \in \pi_m(\phi).\, \mathcal{G}' \models_s \chi') & \{\text{Implication (2)}\} \\
\Rightarrow \forall m \in I^+.\, \exists \chi' \in \pi_m(\phi).\, \mathcal{G} \models_s \chi' & \{\text{Properties } \chi_{\mathcal{G}}\} \\
\Leftrightarrow \exists \chi'' \in \Pi(\phi).\mathcal{G} \models_s \chi'' & \{\text{Definition } \Pi(\phi)\}
\end{array}$$

which proves completeness.

Implication (2) is established as follows. Assume $\chi \models_{\varnothing_{H,m},\varnothing_U,\varnothing_C} \phi$. Behavioural formula ϕ and method m induce a tableau $\mathcal{T}$ giving rise to a set of structural formulae $\pi_m(\phi)$, an unfolding $(\mathcal{T}^*, \tau)$, and a corresponding proof with root sequent $\pi_m(\phi) \vdash_{\varnothing_{H,m},\varnothing_U,\varnothing_C} \phi$. Consider the proof tree obtained from this proof by replacing, in all sequents, $\pi_m(\phi)$ by χ. By local completeness of the proof system, it follows that for every leaf of the latter proof tree corresponding to a contributing leaf of $\mathcal{T}^*$ labelled with triple set C, the proof obligation $\mathcal{V}(C, \chi)$ must be valid, that is $\forall \mathcal{G}.\, (\mathcal{G} \models_s \chi \Rightarrow \exists (i, F, q) \in C.\, \mathcal{G} \models_s i \Rightarrow \left[\widetilde{F}\right] q)$. Let $\mathcal{G}$ be a flow graph such that $\mathcal{G} \models_s \chi$. Then, from every leaf we can choose a triple (i, F, q) for which $\mathcal{G} \models_s i \Rightarrow \left[\widetilde{F}\right] q$ holds. Furthermore, one can show that these triples can be chosen *consistently*, that is, in accordance with the relationship between the leaves of the tableau (called primary leaves) and its unfolding established in Proposition 3. Then, the choices made on the primary leaves define a choice set λ, which induces a structural formula $\chi' \in \pi_m(\phi)$. In addition, one can show that the conjunction over all formulae $i \Rightarrow \left[\widetilde{F}\right] q$ defined by the consistently chosen triples (i, F, q) is logically equivalent to the conjunction over the fixed-point approximants of χ', and thus to χ' itself. Therefore, $\mathcal{G} \models_s \chi'$ holds. □

Theorem 3 and Theorem 4 together establish equivalence (1).

4 Application: Compositional Verification

The original motivation for the present work has been the wish to extend an earlier developed compositional verification method [11] to behavioural properties. The compositional verification method is based on the computation of maximal models: a model is said to be *maximal* for a given property ϕ, if it satisfies ϕ and simulates (*w.r.t.* a property-preserving simulation relation) all other models satisfying ϕ. Due to the close connection between simulation and satisfaction in our logic, we obtain the following compositional verification principle:

to show $\mathcal{G}_1 \uplus \mathcal{G}_2 \models \psi$, it suffices to show $\mathcal{G}_1 \models \phi$ (component $\mathcal{G}_1$ satisfies a suitably chosen *local assumption* ϕ) and $\mathcal{G}_\phi \uplus \mathcal{G}_2 \models \psi$ (component $\mathcal{G}_2$, when composed with the maximal flow graph $\mathcal{G}_\phi$ for ϕ, satisfies the *global guarantee* ψ).

Thus, the compositional verification problem is reduced to finding maximal flow graphs. However, given a property ϕ over a flow graph (behaviour), there is no guarantee that the maximal model of ϕ is a valid flow graph (behaviour). At the structural level this problem can be solved, because we can precisely characterise legal flow graphs *w.r.t.* an interface I as a structural formula θ_I in our logic. Then, if ϕ is an arbitrary structural formula, the maximal model of the formula $\phi \wedge \theta_I$ is a flow graph $\mathcal{G}_{\phi,I}$ which precisely characterises all flow graphs with interface I that satisfy ϕ.

However, there is no such way to precisely characterise flow graph behaviour (*cf.* [11]), and thus one cannot directly apply the compositional verification principle to behavioural properties. In [11], we proposed a "mixed" rule where global guarantees are behavioural, but local assumptions are structural. With the results of the present paper, however, this rule can be combined with the characterisation (1) to yield the following sound and complete compositional verification principle, where both the global guarantee (required to be disjunction-free) and the local assumption are behavioural.

$$\frac{\mathcal{G}_1 \models_b \phi \qquad\qquad \left\{\mathcal{G}_{\chi,I_{\mathcal{G}_1}} \uplus \mathcal{G}_2 \models_b \psi\right\}_{\chi \in \Pi_{I_{\mathcal{G}_1}}(\phi)}}{\mathcal{G}_1 \uplus \mathcal{G}_2 \models_b \psi} \quad \mathcal{G}_1 \text{ closed}$$

Notice that when applying the rule, instead of showing $\mathcal{G}_1 \models_b \phi$ it suffices to show $\mathcal{G}_1^\bullet \models_s \chi$ for some $\chi \in \Pi_{I_{\mathcal{G}_1}}(\phi)$. Completeness of the principle guarantees that no *false negatives* are possible: if the second premise fails, then there is indeed a legal flow graph $\mathcal{G}$ with interface $I_{\mathcal{G}_1}$ such that $\mathcal{G} \models_b \phi$ but $\mathcal{G} \uplus \mathcal{G}_2 \not\models_b \psi$.

Another way of using the characterisation equivalence result (1) for compositional verification is to apply it on the global guarantee, and then to apply a variation of the compositional verification principle, involving structural formulae only. This leads to the following sound and complete compositional verification principle.

$$\frac{\mathcal{G}_1 \models_s \chi' \qquad \exists \chi \in \Pi_{I_{\mathcal{G}_1 \uplus \mathcal{G}_2}}(\psi).\, (\mathcal{G}_{\chi',I_{\mathcal{G}_1}} \uplus \mathcal{G}_2 \models_s \chi)}{\mathcal{G}_1 \uplus \mathcal{G}_2 \models_b \psi}$$

Notice that this rule does not restrict component $\mathcal{G}_1$ to be closed; however, it restricts property χ' to be structural.

5 Conclusion

This report presents a precise characterisation of (disjunction-free) behavioural formulae as sets of structural formulae, in a context where programs are abstracted as flow graphs, and properties are expressed in a fragment of the modal μ-calculus with boxes and greatest fixed points only. To the best of our knowledge, no similar characterisations exist that are based on temporal logic. As one significant application, we state a sound and complete *compositional verification* principle for behavioural properties based on maximal models. Other possible applications of the translation are the reduction of infinite-state verification of behavioural control flow properties to finite-state verification of structural properties, and in the area of program synthesis.

Extensions Unlike the other connectives of the logic, validity of sequents is not compositional *w.r.t. disjunction* in our tableau system. Disjunction can still be handled, though at the expense of completeness, by adding two symmetric tableau rules that simply "drop" the right respectively the left disjunct. A behavioural formula and a method will thus give rise to a set of tableaux, for which we take the union of their induced sets of structural formulae. Alternatively, to obtain a complete translation, we plan to generalise the sequent format, *e.g.,* in the style of Gentzen sequents, and then also tableau construction and formula extraction. We also plan to study whether the characterisation can be extended for the logic with diamonds and least fixed points, and for richer program models (*e.g.,* with exceptions, or multithreading, as in [12]), and whether the compositional verification principle can be generalised to *open* components. For the last extension, two different approaches will be considered: (i) the translation is generalised to formulae over open interfaces, requiring the generalisation of Definition 6 for open flow graphs, and (ii) every open component is "closed" by composing it with a *most general environment* before the characterisation is applied.

Implementation An implementation of the translation has been developed in Ocaml, and is available via a web-based interface [10]. It returns a tableau per method, plus a set of structural formulae (after applying some basic logical simplifications, *e.g.*, removing unused fixed-points, to make the output more readable). It has been applied on all examples in this report. In all cases, the output is produced within seconds. Various *optimisations* of the translation are possible. For instance, since logically subsumed formulae are redundant in the characterisation, the construction of choice sets can be optimised as follows: if a triple is picked from a contributing leaf, then the same triple must be selected from all other contributing leaves containing it.

In future work, the *complexity* of the tableau construction will be studied by finding upper bounds for the size of generated tableaux, and for the number and size of generated formulae.

References

1. R. Alur, M. Arenas, P. Barcelo, K. Etessami, N. Immerman, and L. Libkin. First-order and temporal logics for nested words. In *Logic in Computer Science (LICS '07)*, pages 151–160, Washington, DC, USA, 2007. IEEE Computer Society.
2. R. Alur, M. Benedikt, K. Etessami, P. Godefroid, T. Reps, and M. Yannakakis. Analysis of recursive state machines. *ACM TOPLAS*, 27:786–818, 2005.
3. R. Alur, K. Etessami, and P. Madhusudan. A temporal logic for nested calls and returns. In *Tools and Algorithms for the Analysis and Construction of Software (TACAS '04)*, volume 2998 of *LNCS*, pages 467–481. Springer, 2004.

4. A. Bouajjani, J.C. Fernandez, S. Graf, C. Rodriguez, and J. Sifakis. Safety for branching time semantics. In *Automata, Languages and Programming (ICALP '91)*, volume 501 of *LNCS*, pages 76–92. Springer Verlag, 1991.
5. O. Burkart, D. Caucal, F. Moller, and B. Steffen. Verification on infinite structures. In J.A. Bergstra, A. Ponse, and S.A. Smolka, editors, *Handbook of Process Algebra*, pages 545–623. North Holland, 2000.
6. M. Dam and D. Gurov. μ-calculus with explicit points and approximations. *Journal of Logic and Computation*, 12(2):43–57, 2002.
7. J. Esparza, D. Hansel, P. Rossmanith, and S. Schwoon. Efficient algorithms for model checking pushdown systems. In *Computer Aided Verification (CAV '00)*, volume 1855 of *LNCS*, pages 232–247. Springer, 2000.
8. J. Esparza and S. Schwoon. A BDD-based model checker for recursive programs. In *Computer Aided Verification (CAV '01)*, volume 2102 of *LNCS*, pages 324–336. Springer, 2001.
9. O. Grumberg and D. Long. Model checking and modular verification. *ACM TOPLAS*, 16(3):843–871, 1994.
10. D. Gurov and M. Huisman. From behavioural to structural properties: A tool web interface. `http://www.csc.kth.se/~dilian/Projects/CVPP/beh2struct.php`.
11. D. Gurov, M. Huisman, and C. Sprenger. Compositional verification of sequential programs with procedures. *Information and Computation*, 206(7):840–868, 2008.
12. M. Huisman, I. Aktug, and D. Gurov. Program models for compositional verification. In *International Conference on Formal Engineering Methods (ICFEM '08)*, volume 5256 of *LNCS*, pages 147–166. Springer, 2008.
13. M. Huisman and D. Gurov. Composing modal properties of programs with procedures. In *Formal Foundations of Embedded Software and Component-Based Software Architectures (FESCA '07)*, Electronic Notes in Theoretical Computer Science, 2008. To appear.
14. D. Kozen. Results on the propositional μ-calculus. *Theoretical Computer Science*, 27:333–354, 1983.
15. D. Kozen. *Automata and Computability.* Springer-Verlag New York, Inc., Secaucus, NJ, USA, 1997.
16. A. Pnueli and R. Rosner. On the synthesis of a reactive module. In *Principles of Programming Languages, (POPL '89)*, pages 179–190. ACM, 1989.
17. U. Reddy and S. Kamin. On the power of abstract interpretation. *Computer Languages*, 19(2):79–89, 1993.
18. T. Reps, S. Schwoon, S. Jha, and D. Melski. Weighted pushdown systems and their application to interprocedural dataflow analysis. *Science of Computer Programming*, 58(1-2):206–263, 2005.
19. F. B. Schneider. Enforceable security policies. *ACM Trans. Infinite Systems Security*, 3(1):30–50, 2000.
20. U. Schöpp and A. K. Simpson. Verifying temporal properties using explicit approximants: Completeness for context-free processes. In *Foundations of Software Science and Computation Structures (FoSSaCS '02)*, volume 2303 of *LNCS*, pages 372–386. Springer, 2002.
21. C. Sprenger, D. Gurov, and M. Huisman. Compositional verification for secure loading of smart card applets. In *Formal Methods and Models for Co-Design (MEMOCODE '04)*, pages 211–222. IEEE Computer Society, 2004.
22. C. Stirling. *Modal and Temporal Logics of Processes.* Springer, 2001.
23. R. S. Streett and E. A. Emerson. An automata theoretic decision procedure for the propositional μ-calculus. *Information and Computation*, 81(3):249–264, 1989.

24. I. Walukiewicz. Completeness of Kozen's axiomatisation of the propositional mu-calculus. In *Logic in Computer Science (LICS '95)*, pages 14–24, 1995.
25. P. Wolper. On the relation of programs and computations to models of temporal logic. In *Time and logic: a computational approach*, pages 131–178, London, UK, UK, 1995. UCL Press Ltd.

A Pumping Lemma for CFLs: A PDA view

The well-known Pumping lemma for context-free languages (see *e.g.* [15], pp.148-156) is usually proved by referring to parse trees of words of the given context-free language, generated by some context-free grammar (in Chomsky normal form) for that language. The main property of (binary) trees which the proof is based on is that the depth of a tree is bound by the number of its leaves. As a consequence, for any sufficiently long word of the given context-free language, every parse tree must repeat some non-terminal along some path from the root to some leaf. The sub-trees induced by the repeating occurrences of the non-terminal can be substituted for each other in a context-free fashion, thus giving rise to the "pumping" nature of CFLs.

The Pumping lemma could - less conveniently in general, but relevant for the present work - be phrased by referring to (non-deterministic) pushdown automata instead of context-free grammars.

Lemma 4 (Pumping Lemma for PDAs). *Let $M = (Q, \Sigma, \Gamma, \delta, s, \bot)$ be a non-deterministic PDA accepting on empty stack. Then, there exists $k \geq 0$, such that for every $z \in \Sigma^*$ accepted by M such that $|z| \geq k$, there exist strings $u, v, w, x, y \in \Sigma^*$ where $z = uvwxy$, $vx \neq \epsilon$ and $|vwx| \leq k$, control states $q, q', q'' \in Q$, non-terminal $T \in \Gamma$ and non-terminal strings $\mathcal{X}, \mathcal{Y} \in \Gamma^*$, such that for all $i \geq 0$ there is a run of the PDA of the shape*

$$\langle s, \bot\rangle \xRightarrow{u} \langle q, T\mathcal{Y}\rangle \xRightarrow{v^i} \langle q, T\mathcal{X}^i\mathcal{Y}\rangle \xRightarrow{w} \langle q', \mathcal{X}^i\mathcal{Y}\rangle \xRightarrow{x^i} \langle q', \mathcal{Y}\rangle \xRightarrow{y} \langle q'', \epsilon\rangle$$

Proof. (Sketch) Each production $\langle q_1, A\rangle \overset{a}{\hookrightarrow} \langle q_2, \gamma\rangle$ of a PDA can be viewed as a CFG production $A \rightarrow a\gamma$ (in Greibach normal form) with an associated rewriting $q_1 \rightarrow q_2$ of the control state. So, an accepting run (*i.e.* one that empties the stack) of a PDA can be viewed as a parse tree generated by the corresponding grammar, the non-terminals of the tree being equipped with control states; the run can then be extracted from the tree through a leftmost-outermost traversal. In addition, we label the leaves with the target control states, and call such trees *PDA parse trees.*

We define the *type* of a PDA parse tree as a triple (q, A, q') where A is the non-terminal at the root of the tree, q is its associated control state, and q' is the control state labelling the rightmost leaf. Intuitively, the type of a parse tree denotes that if the PDA is in control state q and the stack consists of A, then upon emptying the stack the PDA is in control state q'. The notion of type can be lifted to a tree node through the sub-tree rooted at this node. Notice that sub-trees of the same type can be substituted for each other in a PDA parse tree in a context-free fashion, *i.e.* without violating the PDA parse tree formation rules.

For every sufficiently long accepting run of a given PDA, the corresponding parse tree must repeat some node type along some path from the root to some leaf. Substituting for each other the induced sub-trees gives rise to the same kind of pumping behaviour as in the standard view explained above. In terms of the accepting run itself, the result follows. □

Note that the above proposition can be generalised to arbitrary sub-runs z of M by relaxing state q_0 and non-terminal S to be arbitrary, and by appending any non-terminal string $\mathcal{Z} \in \mathit{\Gamma}^*$ at the end of all stacks of the original run.