# A framework for proof certificates in finite state exploration

Quentin Heath, Dale Miller

# A framework for proof certificates in finite state exploration

Quentin Heath and Dale Miller
Inria Saclay–Île-de-France LIX, École polytechnique

Model checkers use automated state exploration in order to prove various properties such as reachability, non-reachability, and bisimulation over state transition systems. While model checkers have proved valuable for locating errors in computer models and specifications, they can also be used to prove properties that might be consumed by other computational logic systems, such as theorem provers. In such a situation, a prover must be able to trust that the model checker is correct. Instead of attempting to prove the correctness of a model checker, we ask that it outputs its "proof evidence" as a formally defined document—a proof certificate—and that this document is checked by a trusted proof checker. We describe a framework for defining and checking proof certificates for a range of model checking problems. The core of this framework is a (focused) proof system that is augmented with premises that involve "clerk and expert" predicates. This framework is designed so that soundness can be guaranteed independently of any concerns for the correctness of the clerk and expert specifications. To illustrate the flexibility of this framework, we define and formally check proof certificates for reachability and non-reachability in graphs, as well as bisimulation and non-bisimulation for labeled transition systems. Finally, we describe briefly a reference checker that we have implemented for this framework.

## 1 Introduction

Model checkers are one way in which logic is implemented. While one of the strengths of model checkers is to aid in the discovery of counterexamples and errors in specifications [6], they can also be used to prove theorems. Furthermore, such theorems might be of interest to other computational logic systems such as more general theorem provers. One then encounters the problem of whether or not such a theorem prover is willing to trust that model checker or at least a particular theorem it proves. Formally verifying a model checker might be both extremely hard to do and undesirable especially if that checker is being revised and improved. A more plausible option might be to have a model checker output its "proof evidence" as a document (a *certificate*). If that proof certificate can be formally checked by a trusted checker, one might then be willing to use the theorem in a theorem prover.

Of course, model checkers are asked to solve many kinds of problems so their proof evidence might take many different forms, ranging from decision procedures to paths in graphs, bisimulations, traces, and winning strategies. If we need to have trusted checkers for all these different kinds of proof evidence, then maybe we have not really improved the situation of trust. Here, we contribute to the *foundational proof certificate* (FPC) effort [13] by providing a framework for defining the semantics of a range of proof evidence that naturally arises in model checking. Such a formal semantic model for proof evidence allows anyone to build a proof checker of *any* formally defined evidence. Furthermore, it is possible to have an implementation of the entire framework of FPC so that this one system could check a wide range of proof evidence.

While this paper has a number of parallels with FPCs for first-order logic in [5], that work was limited to first-order logic *without* fixed points and, as a result, that work was not directly applicable to topics of model checking and inductive and co-inductive theorem proving.

Submitted to:

# 2 Proof theory for fixed points and certificates

Having proof certificates that are foundational here means that we need to find proof theoretic descriptions of model checking. We shall now describe a few recent developments in proof theory that we bring together in this paper. Of course, the topic of model checking is mature and varied. In order to lay down a convincing and direct proof theory for model checking, we eschew many of its more advanced topics—e.g., predicate abstraction and partial order reduction—for later consideration.

## 2.1 Fixed points as defined predicates

One of the earliest applications of sequent calculus to computational logic was to provide an execution model for logic programming [15]. That analysis, however, supported only the "open world assumption" of logic programming: negation-as-finite-failure was not touched by that work. Schroeder-Heister [19] and Girard [10] showed how sequent calculus could be extended with inference rules for fixed points (or *defined* predicates), thereby embracing important aspects of the *closed world assumption* and negation-as-finite-failure. The key additions to sequent calculus were rules for unfolding fixed point expressions as well as dealing with equality over the Herbrand universe. A series of papers [8, 12, 16] added induction and co-induction to the sequent calculi for intuitionistic and classical logics. Those papers have been used to design the Bedwyr model checker [4, 20] and the Abella interactive theorem prover [3].

Fixed point expressions will be written as $\mu B\bar{t}$ or $\nu B\bar{t}$, where $B$ is an expression representing a higher-order abstraction, and $\bar{t}$ is a list of terms. The unfolding of the fixed point expression $\mu B\bar{t}$ is written as $B(\mu B)\bar{t}$. It is important to understand that we shall treat both $\mu$ (least fixed point operator) and $\nu$ (greatest fixed point operator) as logical connectives since they will have introduction rules: they are also de Morgan duals of each other.

## 2.2 Fixed points in linear logic

Surprisingly, it is linear logic and not intuitionistic or classical logics per se that is most relevant to our exposition on model checking. The logic MALL (*multiplicative additive linear logic*) is an elegant, small logic that is, in and of itself, not appropriate for formalizing mathematics and computer science since it is not capable of modeling unbounded behaviors (for example, it is decidable). While Girard extended MALL with the "exponentials" (! and ?) [9], Baelde [2] extended it by adding the least ($\mu$) and greatest ($\nu$) fixed points operators as logical connectives. The resulting logic, called $\mu$MALL, forms the proof theoretic foundation of this paper.

To make the use of linear logic easier to swallow for those more familiar with model checking, we adopt the following shallow changes to its presentation. First, we use a two-sided sequent calculus instead of the one-sided calculus used for $\mu$MALL. While this change will double the size of our proof system, it will make inference rules look more familiar. Second, we replace the linear logic connectives with familiar connectives (although with annotations). In particular, we replace $\otimes$, &, $\oplus$ and their units $\mathbf{1}$, $\top$ and $\mathbf{0}$ with $\wedge^+$, $\wedge^-$, $\vee$, $t^+$, $t^-$ and $f^+$, respectively. (Truth functionally, the two versions of these operators are equivalent: their differences only influence the structure of focused proofs.) We also replace the negatively biased false $\bot$ with $f^-$, and instead of the multiplicative disjunction $A \parr B$, we use the implication $A^\bot \supset B$: the de Morgan dual of $A \supset B$ is $A \wedge^+ B^\bot$. Negation is written as $\cdot \supset f^-$.

In addition, we consider $\mu$ as positive and $\nu$ as negative; this arbitrary choice has been shown to give a convenient natural interpretation to the structure of focused proofs [2]. We therefore have the negative connectives $f^-$, $\supset$, $t^-$, $\wedge^-$, $\forall$, $\neq$ and $\nu$, and the positive connectives $t^+$, $\wedge^+$, $f^+$, $\vee$, $\exists$, $=$ and $\mu$.

### 2.3 Focused proof systems

In order to have the kind of control we need to support a definable notion of proof certificate, we make use of a *focused proof system*. Such sequent calculus proof systems are built from alternating phases which allow us to define flexible proof building protocols that can be used to drive proof search. During the *asynchronous* phase of proof building, simple (invertible) computations build a proof and during the *synchronous* phase, information needed for the construction of a proof (such as which branch of a disjunction to prove) must be found.

Focusing requires polarizing all formulas as being either negative or positive. A formula is negative or positive according to its top-level connective, and it is *purely negative* (resp. *purely positive*) when its connectives are positive if, and only if, they occur under an odd (resp. even) number of implications. Notice that the de Morgan dual of a positive (resp. purely positive) formula is a negative (resp. purely negative) formula. We call a formula *bipolar* when it is made of purely negative (resp. positive) subformulas occurring under an even (resp. odd) number of implications in a purely negative context.

Focusing also relies on the sequents having additional storage zones on each side of the turnstile, where formulas can be stored and left untouched by logical inference rules. For instance, the usual one-sided focused presentation of $\mu$MALL [2] has one of these zones, similarly to the focused proof system for linear logic given by Andreoli [1]. A two-sided subsystem of $\mu$MALLF, called $\mu F$, makes use of two storage zones, noted $\mathscr{N}$ and $\mathscr{P}$, which are lists of, respectively, negative and positive formulas. (Appendix B contains an example of a $\mu F$ proof.) Between the arrows and the turnstile, are the contexts $\Gamma$ and $\Delta$: these are lists of formulas in (unfocused) $\Uparrow$-sequents, and sets of up to one formula in (focused) $\Downarrow$-sequents. The sequents of the $\mu F$ system are therefore:

| | |
|---|---|
| $\mathscr{N} \Uparrow \Gamma \vdash \Delta \Uparrow \mathscr{P}$ | unfocused, similar to the $\mu$MALLF sequent $\vdash \mathscr{N}^{\perp}, \mathscr{P} \Uparrow \Gamma^{\perp}, \Delta$ |
| $\Downarrow A \vdash$ | left-focused, similar to $\vdash \Downarrow A^{\perp}$ |
| $\vdash A \Downarrow$ | right-focused, similar to $\vdash \Downarrow A$ |

### 2.4 Foundational proof certificates

If we think of the implementers of computational logic systems (*e.g.*, model checking systems) as our clients, our job in this project is to formally check our client's proof evidence for formal correctness. Our approach is to have this evidence translated into a sequent calculus proof. Of course, we would not dream of asking our clients to supply a sequent calculus proof in the first place: such proofs are often huge, too messy, and too esoteric. Instead, we want to take from our clients objects with which they are familiar (*e.g.*, paths, simulations, *etc.*) and find flexible and high-level means to have our framework extract information from those objects in order to trace out a complete formal sequent calculus proof.

To this intent, Figures 1 and 2 present $\mu F^{a}$, which is a version of $\mu F$ augmented with a term $\Xi$ (encoding an actual certificate) as well as with *clerk* and *expert* predicates (examples of which we provide soon). This augmentation has two components. First, every sequent (either $\Uparrow$ or $\Downarrow$) is given an extra argument we write as $\Xi$. Thus, sequents now display as

$$\Xi : \mathscr{N} \Uparrow \Gamma \vdash \Delta \Uparrow \mathscr{P}, \qquad \Xi : \Downarrow A \vdash, \qquad \text{and} \qquad \Xi : \vdash A \Downarrow .$$

Second, every inference rule is given an additional premise. In all cases, this premise is an atomic formula with either a clerk or expert predicate as its head symbol: if the conclusion of the inference rule is a $\Downarrow$-sequent, then the premise atom uses an expert predicate (noted $\star_e(\dots)$ for the rule $\star$); otherwise, the conclusion is an $\Uparrow$-sequent and the atom uses a clerk predicate (noted $\star_c(\dots)$).

In the case of the clerk rules, the premise atom relates the $\Xi$ value of the concluding sequent with the corresponding value of $\Xi$ for all premises: *e.g.*,

$$\frac{\Xi_1 : \mathscr{N} \Uparrow A_1, \Gamma \vdash \Delta \Uparrow \quad \Xi_2 : \mathscr{N} \Uparrow A_2, \Gamma \vdash \Delta \Uparrow \quad \vee_c(\Xi_0, \Xi_1, \Xi_2)}{\Xi_0 : \mathscr{N} \Uparrow A_1 \vee A_2, \Gamma \vdash \Delta \Uparrow} \vee_L$$

In this way, the certificate $\Xi$, intended to aid in the proof of the concluding sequent, can be transformed into two certificates that are used to prove the two premise sequents. We refer to the predicates used in the asynchronous phase as clerks since these predicates do not need, in general, to examine the actual information in the proof certificate (except for the induction and co-induction rules, there is no consumption of information during the asynchronous phase). Instead, the clerks are responsible for keeping track of how a proof is unfolding: for example, $\Xi_1$ might be a copy of $\Xi_0$ but with the fact that checking has moved to the left premise instead of the right premise.

Experts are responsible for extracting information from a certificate. For example, $\mu F^a$ contains the inference rule

$$\frac{\Xi_1 : \vdash C\, t \Downarrow \quad \exists_e(\Xi_0, \Xi_1, t)}{\Xi_0 : \vdash \exists x. C\, x \Downarrow} \exists_R$$

Notice here that the exists-expert $\exists_e(\cdot,\cdot,\cdot)$ not only computes the continuation certificate $\Xi_1$ but also a term $t$ to be used to witness the existential variable.

The exact nature of both certificate terms $\Xi$ and of the clerk and expert predicates is not important to guarantee soundness of this system. That is, no matter how certificates, clerks, and experts are specified, if there is a proof in $\mu F^a$ then there is a proof in $\mu F$ of the same sequent, which can be obtained by deleting from the proof in $\mu F^a$ all references to $\Xi$, including the additional premises. Notice also that experts are not required to act particularly expertly: it is entirely possible for the $\exists_e(\Xi_0, \Xi_1, t)$ premise to functionally determine one $t$ from $\Xi_0$, or to relate all terms $t$ to $\Xi$. In the latter case, the actual value of $t$ used in a successful $\mu F^a$ proof is determined from other aspects of the proof checking process (typically implemented using unification).

## 3 A proof system underlying model checking

FPCs were first proposed in [5, 13] in the context of first-order logic and were used successfully to define and check proof evidence in the form of resolution refutations, Herbrand instances (expansion trees), natural deduction ($\lambda$-terms), Frege proofs, *etc.* We shall now adapt this approach to formally define the semantics of a range of proof evidence that can arise in simple but real model checking problems.

We shall later illustrate just how such a formal semantics can be provided for the following four kinds of proof evidence. These particular examples have been selected for their universality: numerous problems in model checking are related to them.

1. The fact that two nodes are related by the transitive closure of a graph's adjacency relation can be witnessed by an *explicit path* through the graph.
2. The fact that two nodes are *not* related by transitivity can be witnessed by pointing out that the *reachable set* of one does not contain the other.
3. Given an LTS (labeled transition system), the fact that two nodes are similar/bisimilar can be witnessed by a set of pairs called *simulation/bisimulation*.
4. If two nodes in an LTS are *not* bisimilar, then there is a *Hennessy-Milner logic (HML) formula* that is satisfied by one but not by the other.

ASYNCHRONOUS CONNECTIVE INTRODUCTIONS

$$\frac{\Xi_1\theta : \mathscr{N}\theta \Uparrow \Gamma\theta \vdash \Delta\theta \Uparrow \quad =_c^s(\Xi_0,\Xi_1)}{\Xi_0 : \mathscr{N} \Uparrow s = t, \Gamma \vdash \Delta \Uparrow} =_L^s\dagger \qquad \frac{\Xi_1\theta : \mathscr{N}\theta \Uparrow \vdash \Uparrow \quad \neq_c^f(\Xi_0,\Xi_1)}{\Xi_0 : \mathscr{N} \Uparrow \vdash s \neq t \Uparrow} \neq_R^f\dagger$$

$$\frac{\Xi_1 : \mathscr{N} \Uparrow \Gamma \vdash \Delta \Uparrow \quad t_c^+(\Xi_0,\Xi_1)}{\Xi_0 : \mathscr{N} \Uparrow t^+, \Gamma \vdash \Delta \Uparrow} t_L^+ \qquad \frac{\Xi_1 : \mathscr{N} \Uparrow \vdash \Uparrow \quad f_c^-(\Xi_0,\Xi_1)}{\Xi_0 : \mathscr{N} \Uparrow \vdash f^- \Uparrow} f_R^-$$

$$\frac{\Xi_1 : \mathscr{N} \Uparrow A_1, A_2, \Gamma \vdash \Delta \Uparrow \quad \wedge^+{}_c(\Xi_0,\Xi_1)}{\Xi_0 : \mathscr{N} \Uparrow A_1 \wedge^+ A_2, \Gamma \vdash \Delta \Uparrow} \wedge^+{}_L \qquad \frac{\Xi_1 : \mathscr{N} \Uparrow A_1 \vdash A_2 \Uparrow \quad \supset_c(\Xi_0,\Xi_1)}{\Xi_0 : \mathscr{N} \Uparrow \vdash A_1 \supset A_2 \Uparrow} \supset_R$$

$$\frac{=_c^f(\Xi_0)}{\Xi_0 : \mathscr{N} \Uparrow s = t, \Gamma \vdash \Delta \Uparrow} =_L^f\ddagger \qquad \frac{\neq_c^s(\Xi_0)}{\Xi_0 : \mathscr{N} \Uparrow \vdash s \neq t \Uparrow} \neq_R^s\ddagger$$

$$\frac{f_c^+(\Xi_0)}{\Xi_0 : \mathscr{N} \Uparrow f^+, \Gamma \vdash \Delta \Uparrow} f_L^+ \qquad \frac{t_c^-(\Xi_0)}{\Xi_0 : \mathscr{N} \Uparrow \vdash t^- \Uparrow} t_R^-$$

$$\frac{\Xi_1 : \mathscr{N} \Uparrow A_1, \Gamma \vdash \Delta \Uparrow \quad \Xi_2 : \mathscr{N} \Uparrow A_2, \Gamma \vdash \Delta \Uparrow \quad \vee_c(\Xi_0,\Xi_1,\Xi_2)}{\Xi_0 : \mathscr{N} \Uparrow A_1 \vee A_2, \Gamma \vdash \Delta \Uparrow} \vee_L$$

$$\frac{\Xi_1 : \mathscr{N} \Uparrow \vdash A_1 \Uparrow \quad \Xi_2 : \mathscr{N} \Uparrow \vdash A_2 \Uparrow \quad \wedge^-{}_c(\Xi_0,\Xi_1,\Xi_2)}{\Xi_0 : \mathscr{N} \Uparrow \vdash A_1 \wedge^- A_2 \Uparrow} \wedge^-{}_R$$

$$\frac{\Xi_1\, y : \mathscr{N} \Uparrow C y, \Gamma \vdash \Delta \Uparrow \quad \exists_c(\Xi_0,\Xi_1)}{\Xi_0 : \mathscr{N} \Uparrow \exists x.Cx, \Gamma \vdash \Delta \Uparrow} \exists_L \qquad \frac{\Xi_1\, y : \mathscr{N} \Uparrow \vdash C y \Uparrow \quad \forall_c(\Xi_0,\Xi_1)}{\Xi_0 : \mathscr{N} \Uparrow \vdash \forall x.Cx \Uparrow} \forall_R$$

SYNCHRONOUS CONNECTIVE INTRODUCTIONS

$$\frac{\neq_e^f(\Xi_0)}{\Xi_0 : \Downarrow t \neq t \vdash} \neq_L^f \qquad \frac{=_e^s(\Xi_0)}{\Xi_0 : \vdash t = t \Downarrow} =_R^s \qquad \frac{f_e^-(\Xi_0)}{\Xi_0 : \Downarrow f^- \vdash} f_L^- \qquad \frac{t_e^+(\Xi_0)}{\Xi_0 : \vdash t^+ \Downarrow} t_R^+$$

$$\frac{\Xi_1 : \vdash A_1 \Downarrow \quad \Xi_2 : \Downarrow A_2 \vdash \quad \supset_e(\Xi_0,\Xi_1,\Xi_2)}{\Xi_0 : \Downarrow A_1 \supset A_2 \vdash} \supset_L \qquad \frac{\Xi_1 : \vdash A_1 \Downarrow \quad \Xi_2 : \vdash A_2 \Downarrow \quad \wedge^+{}_e(\Xi_0,\Xi_1,\Xi_2)}{\Xi_0 : \vdash A_1 \wedge^+ A_2 \Downarrow} \wedge^+{}_R$$

$$\frac{\Xi_1 : \Downarrow A_i \vdash \quad \wedge^-{}_e(\Xi_0,\Xi_1,i)}{\Xi_0 : \Downarrow A_1 \wedge^- A_2 \vdash} \wedge^-{}_L \qquad \frac{\Xi_1 : \vdash A_i \Downarrow \quad \vee_e(\Xi_0,\Xi_1,i)}{\Xi_0 : \vdash A_1 \vee A_2 \Downarrow} \vee_R$$

$$\frac{\Xi_1 : \Downarrow Ct \vdash \quad \forall_e(\Xi_0,\Xi_1,t)}{\Xi_0 : \Downarrow \forall x.Cx \vdash} \forall_L \qquad \frac{\Xi_1 : \vdash Ct \Downarrow \quad \exists_e(\Xi_0,\Xi_1,t)}{\Xi_0 : \vdash \exists x.Cx \Downarrow} \exists_R$$

STRUCTURAL RULES

$$\frac{\Xi_1 : N \Uparrow \Gamma \vdash \Delta \Uparrow \quad \mathrm{store}_L(\Xi_0,\Xi_1)}{\Xi_0 : \Uparrow N, \Gamma \vdash \Delta \Uparrow} \mathrm{S}_L \qquad \frac{\Xi_1 : \Uparrow \vdash \Uparrow P \quad \mathrm{store}_R(\Xi_0,\Xi_1)}{\Xi_0 : \Uparrow \vdash P \Uparrow} \mathrm{S}_R$$

$$\frac{\Xi_1 : \Downarrow N \vdash \quad \mathrm{decide}_L(\Xi_0,\Xi_1)}{\Xi_0 : N \Uparrow \vdash \Uparrow} \mathrm{D}_L \qquad \frac{\Xi_1 : \vdash P \Downarrow \quad \mathrm{decide}_R(\Xi_0,\Xi_1)}{\Xi_0 : \Uparrow \vdash \Uparrow P} \mathrm{D}_R$$

$$\frac{\Xi_1 : \Uparrow P \vdash \Uparrow \quad \mathrm{release}_L(\Xi_0,\Xi_1)}{\Xi_0 : \Downarrow P \vdash} \mathrm{R}_L \qquad \frac{\Xi_1 : \Uparrow \vdash N \Uparrow \quad \mathrm{release}_R(\Xi_0,\Xi_1)}{\Xi_0 : \vdash N \Downarrow} \mathrm{R}_R$$

**Figure 1:** The $\mu F_0^a$ proof system. (This proof system is best viewed using color).
$y$ stands for a fresh eigenvariable, $s$ and $t$ for terms, $N$ for a negative formula, $P$ for a positive formula, and $C$ for the abstraction of a formula over a variable.
The † proviso requires that $\theta$ is the *mgu* of $s$ and $t$, and the ‡ proviso requires that $s$ and $t$ are not unifiable.

FIXED-POINT RULES

$$\frac{\Xi_1\,\bar{y}\colon \Uparrow B S \bar{y} \vdash S\bar{y} \Uparrow \quad \Xi_2 : \mathscr{N} \Uparrow S\bar{t}, \Gamma \vdash \Delta \Uparrow \quad \mathrm{ind}(\Xi_0, \Xi_1, \Xi_2, S)}{\Xi_0 : \mathscr{N} \Uparrow \mu B \bar{t}, \Gamma \vdash \Delta \Uparrow}\ \mu$$

$$\frac{\Xi_1 : \mathscr{N} \Uparrow \vdash S\bar{t} \Uparrow \quad \Xi_2\,\bar{y}\colon \Uparrow S\bar{y} \vdash B S \bar{y} \Uparrow \quad \text{co-ind}(\Xi_0, \Xi_1, \Xi_2, S)}{\Xi_0 : \mathscr{N} \Uparrow \vdash \nu B \bar{t} \Uparrow}\ \nu$$

$$\frac{\Xi_1 : \mathscr{N} \Uparrow B(\mu B)\bar{t}, \Gamma \vdash \Delta \Uparrow \quad \mu\text{-unfold}_L(\Xi_0, \Xi_1)}{\Xi_0 : \mathscr{N} \Uparrow \mu B \bar{t}, \Gamma \vdash \Delta \Uparrow}\ \mu_L \qquad \frac{\Xi_1 : \mathscr{N} \Uparrow \vdash B(\nu B)\bar{t} \Uparrow \quad \nu\text{-unfold}_R(\Xi_0, \Xi_1)}{\Xi_0 : \mathscr{N} \Uparrow \vdash \nu B \bar{t} \Uparrow}\ \nu_R$$

$$\frac{\Xi_1 : \Downarrow B(\nu B)\bar{t} \vdash \quad \nu\text{-unfold}_L(\Xi_0, \Xi_1)}{\Xi_0 : \Downarrow \nu B \bar{t} \vdash}\ \nu_L \qquad \frac{\Xi_1 : \vdash B(\mu B)\bar{t} \Downarrow \quad \mu\text{-unfold}_R(\Xi_0, \Xi_1)}{\Xi_0 : \vdash \mu B \bar{t} \Downarrow}\ \mu_R$$

**Figure 2:** The $\mu F^a$ proof system results from adding these rules to $\mu F^a_0$.
$\bar{y}$ stands for an list of fresh eigenvariables, $\bar{t}$ for an list of terms, and $B$ for the abstraction of a formula over a predicate and a variable list.

### 3.1 Core proof system

Figures 1 and 2 contain the rules for the augmented focused proof systems $\mu F^a_0$ and $\mu F^a$. One could obtain the non-augmented systems $\mu F_0$ and $\mu F$ by ignoring the certificates (annotated $\Xi$ variables) and the clerk and expert premises; the resulting rules would be no more than (slightly restricted) two-sided versions of the $\mu$MALLF rules. The various clerk and expert predicates are named and displayed in their corresponding inference rules. Notice that those inference rules that involve the use of eigenvariables ($\exists_L$, $\forall_R$, $\mu$ and $\nu$) require the associated clerk predicates to return abstractions over certificates: in this way, premise certificates can be applied to the eigenvariables.

A key element of our proof theoretic treatment of model checking via $\mu F^a$ is the fact that focused sequents contain only one formula. This fact entails that $\mu F^a$ can only be complete with respect to $\mu$MALLF on a fragment where derivations satisfy this constraint. In particular, the $\mathscr{N}$ and $\mathscr{P}$ zones must never contain more than one formula, and never both at the same time. This can be ensured at least for the $\mu F^a_0$ subsystem by the following restriction on formulas.

**Definition 1** (switchable formula, switchable occurrence)**.** *A $\mu F^a$ formula is* switchable *if*

- *whenever a subformula $C \wedge^+ D$ occurs negatively (under an odd number of implications), either $C$ or $D$ is purely positive;*
- *whenever a subformula $C \supset D$ occurs positively (under an even number of implications), either $C$ is purely positive or $D$ is purely negative.*

*An occurrence of a formula $B$ is* switchable *if it appears on the right-hand side (resp. left-hand side) and $B$ (resp. $B \supset f^-$) is switchable.*

Notice that both a purely positive formula and its de Morgan dual are switchable. The follow theorem is proved by a simple induction on the structure of $\mu F^a_0$ proofs.

**Theorem 1** (switchability)**.** *Let $\Pi$ be a $\mu F^a_0$ derivation of either $\Uparrow A \vdash \Uparrow$ or $\Uparrow \vdash A \Uparrow$, where the occurrence of $A$ is switchable. Every sequent in $\Pi$ that is the conclusion of a rule that switches phases (either a decide or a release rule) contains exactly one occurrence of a formula and that occurrence is switchable.*

Theorem 1 states that an invariant of the $\mu F_0^a$ proof system (for switchable theorems) is that the number of non-purely asynchronous formulas (*i.e.* non-purely positive from $\mathscr{N}$ and $\Gamma$, and non-purely negative from $\mathscr{P}$ and $\Delta$) is one or less. Keeping sequents mostly asynchronous allows the asynchronous phase to deal with most of the context: that way the synchronous phase is left with a single, meaningful formula. (The structure of focused proofs based on switchable formulas is similar to the structure of *simple games* in the game-theoretic analysis of focused proofs in [7, Section 4].) While the restriction to switchable formulas provides a match to the model checking problems we develop here, that restriction is not needed for using clerks and experts (the examples in [5] involve non-switchable formulas).

### 3.2 Encoding of recursively defined predicates

In order to exploit the properties of $\mu F_0^a$ in model checking problems, we need them to extend to $\mu F^a$ by adding fixed-point rules. As those rules make use of the higher-order variables $S$ (an invariant which is either a pre-fixed point or a post-fixed point) and $B$ (the body of a predicate definition), they cannot be used freely without violating Theorem 1. We propose the following constraints on $\mu F^a$ proofs of switchable formulas so as to have exactly one formula per sequent when phases are switched:

- "arithmetic" restriction: $S$ and $B$ are purely positive (resp. negative);
- "model checking" restriction: $S$ is purely negative (resp. positive), and the context does not trigger synchronous rules ($\mathscr{N}$ is empty, $\Gamma$ is purely positive and $\Delta$ is purely negative).

The former restriction would allow to extend the scope of the framework by handling simple theorems involving inductive definitions (*e.g.* about natural numbers), but is not treated here. The latter restriction better suits our needs (since an asynchronous context fits the spirit of model checking) and is respected by all our examples.

**Example 2.** *Horn clauses (in the sense of Prolog) can be encoded as purely positive fixed point expressions. For example, here is the Horn clause logic program (using the λProlog syntax, the* `sigma Y\` *construction encodes the quantifier ∃Y) for specifying the graph in Figure 3 and its transitive closure:*

```
step a b.   step b c.    step c b.
path X Y :- step X Y.    path X Z :- sigma Y\ step X Y, path Y Z.
```

*We can translate the* `step` *relation into the binary predicate $\cdot \longrightarrow \cdot$ defined by*

$$\mu\left(\lambda A\lambda x\lambda y.\,((x=a)\wedge^{+}(y=b))\vee((x=b)\wedge^{+}(y=c))\vee((x=c)\wedge^{+}(y=b))\right)$$

*which only uses positive connectives. Likewise,* `path` *can be encoded as path:*

$$\mu\left(\lambda A\lambda x\lambda z.\,x\longrightarrow z\vee(\exists y.\,x\longrightarrow y\wedge^{+}A\,y\,z)\right)$$

In general, it is sensible to view any purely positive least fixed point expression as a predicate specified by Horn clauses. (For example, SOS rules for CCS are easily seen as Horn clauses.)

**Example 3.** *Let the ternary predicate $\cdot \dot{\longrightarrow} \cdot$ describe a labeled transition system. It can be defined as a purely positive fixed point expression of the form*

$$\mu\left(\lambda A\lambda p\lambda a\lambda q.\,\textstyle\bigvee_i((p=u_i)\wedge^{+}(a=v_i)\wedge^{+}(q=w_i))\right)$$

*and the simulation and bisimulation relations can be defined as the following greatest fixed point expressions (note: the second contains both $\wedge^{-}$ and $\wedge^{+}$). Both of these formulas are switchable.*

$$\nu\left(\lambda S\lambda p\lambda q.\,\forall a\forall p'.\,p\xrightarrow{a}p'\supset\exists q'.\,q\xrightarrow{a}q'\wedge^{+}S\,p'\,q'\right) \tag{sim}$$

$$\begin{aligned}\nu\big(\lambda B\lambda p\lambda q.\,&(\forall a\forall p'.\,p\xrightarrow{a}p'\supset\exists q'.\,q\xrightarrow{a}q'\wedge^{+}B\,p'\,q')\\ &\wedge^{-}(\forall a\forall q'.\,q\xrightarrow{a}q'\supset\exists p'.\,p\xrightarrow{a}p'\wedge^{+}B\,q'\,p')\big)\end{aligned} \tag{bisim}$$

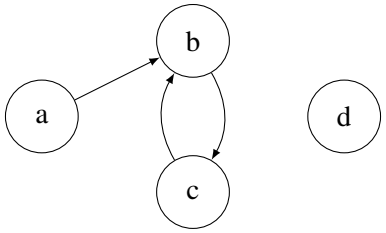


**Figure 3:** (Un)reachability problem

### 3.3 Common proof certificates

The presentation of an FPC now involves the following three steps.

1. Describe how unpolarized formulas should be polarized.
2. Describe the structure of certificates $\Xi$. This can be done, for example, by describing the signature of constructors for certificates.
3. Define the clerk and expert predicates.

To ease steps 2 and 3, we define the following certificate constructors (shown together with their types), which describe generic focused proof behaviors. The associated clauses can be included into any subsequent clerks and experts definitions.

The **stop** : `cert` certificate authorizes no search; it is to be used as a continuation certificate for other certificate constructors.

The **sync** : `cert->cert` certificate constructor authorizes $\mu F^a$ to conduct an unbounded synchronous search for a proof before handing the search over to a continuation certificate. It has no clerks and its experts run an exhaustive non-deterministic search for $\vee$ and $\exists$. The experts for the right rules are:

$$
\begin{array}{ll}
=^s_e(\mathsf{sync}(\Xi)). & \vee_e(\mathsf{sync}(\Xi),\mathsf{sync}(\Xi),1). \\
\wedge^+{}_e(\mathsf{sync}(\Xi),\mathsf{sync}(\Xi),\mathsf{sync}(\Xi)). & \vee_e(\mathsf{sync}(\Xi),\mathsf{sync}(\Xi),2). \\
\forall T.\, \exists_e(\mathsf{sync}(\Xi),\mathsf{sync}(\Xi),T). & \mu\text{-unfold}_R(\mathsf{sync}(\Xi),\mathsf{sync}(\Xi)). \\
\text{release}_R(\mathsf{sync}(\Xi),\Xi). &
\end{array}
$$

The **async** : `cert->cert` certificate constructor is the dual of sync; it handles an asynchronous phase and has no experts apart from the decide rules. The clerks for the left rules are:

$$
\begin{array}{ll}
=^s_c(\mathsf{async}(\Xi),\mathsf{async}(\Xi)). & \vee_c(\mathsf{async}(\Xi),\mathsf{async}(\Xi),\mathsf{async}(\Xi)). \\
\wedge^+{}_c(\mathsf{async}(\Xi),\mathsf{async}(\Xi)). & \\
\exists_c(\mathsf{async}(\Xi),\lambda x.\,\mathsf{async}(\Xi)). & \mu\text{-unfold}_L(\mathsf{async}(\Xi),\mathsf{async}(\Xi)). \\
\text{store}_L(\mathsf{async}(\Xi),\mathsf{async}(\Xi)). & \text{decide}_L(\mathsf{async}(\Xi),\Xi).
\end{array}
$$

**bipole**$_n$ : `cert` is actually short-hand for a chain of $n$ $\mathsf{async}(\mathsf{sync}(\cdot))$ before a final stop. It is used for bounded-depth search when simple search strategies would otherwise not terminate. We also write **bipole** : `cert` for $\mathsf{bipole}_1 = \mathsf{async}(\mathsf{sync}(\mathsf{stop}))$.

The **decproc** : `cert` constructor is short-hand for $\mathsf{bipole}_\infty$, the unbounded version of $\mathsf{bipole}_n$. It is a general purpose decision procedure used for automated and unguided proving. Its rules are similar to those from sync and async, and can be obtained via the equivalence $\mathsf{decproc} = \mathsf{async}(\mathsf{sync}(\mathsf{decproc}))$.

The two constructors **inv** and **co-inv** : `(i->i->bool)->cert->cert` each take an explicit predicate $S$ as parameter. It is expected to be proved to be an invariant with the help of bipole.

$$\forall S.\,\mathsf{ind}(\mathsf{inv}(S,\Xi),\lambda\bar{x}.\,\mathsf{bipole},\Xi,S) \qquad \forall S.\,\mathsf{co\text{-}ind}(\mathsf{co\text{-}inv}(S,\Xi),\Xi,\lambda\bar{x}.\,\mathsf{bipole},S)$$

We now turn our attention to describing how to formally define the four kinds of proof evidence mentioned earlier in Section 3. Some of the constructors defined above will be used in those definitions.

# 4 Examples: certificates for graphs

We use the notations from Example 2 to define $\cdot \longrightarrow \cdot$ and *path*.

## 4.1 Lists as reachability certificates

The natural choice for a certificate of the proof of $\vdash \mathit{path}(x,y)$ is an explicit path, *i.e.* a list of nodes starting right after $x$ and ending right before $y$. In fact, this list $L$ can be used directly as the proof certificate. Aside from the initial $\mathsf{store}_R$, no clerks are invoked in the process of checking this particular FPC. The following clauses defining the experts only use the provided information to instantiate the logical variables of the proof.

$$\forall X\forall L.\,\exists_e(X :: L, L, X). \qquad \forall L.\,\wedge^+{}_e(L,\mathsf{sync}(\mathsf{stop}),L). \qquad \forall L.\,\mathsf{decide}_R(L,L).$$
$$\forall X\forall L.\,\vee_e(X :: L, X :: L, 2). \qquad \forall L.\,\vee_e(\mathit{nil},\mathsf{sync}(\mathsf{stop}),1). \qquad \forall L.\,\mu\text{-}\mathsf{unfold}_R(L,L).$$

In this setting, the $\mathsf{sync}(\mathsf{stop})$ certificate will terminate quickly since it is only searching through the term that defines $\cdot \longrightarrow \cdot$.

**Example 4.** *In Figure 3, $(c)$ is reachable from $(a)$, as witnessed by certificates like $[b]$, $[b;c;b]$, etc.*

## 4.2 Invariants as non-reachability certificates

The non-reachability problem comes in two forms: if there are no loops in the graph, then a simple check of the set of nodes reachable from the first node provides a simple decision procedure; if there are loops, then induction is needed.

In the first case, the decision procedure can directly be translated as an FPC for proving $\vdash \neg\mathit{path}(x,y)$.

**Example 5.** *In Figure 3, $(a)$ is not reachable from $(d)$, as witnessed by $\mathsf{async}(\mathsf{stop})$.*

On the other hand, if the underlying graph has loops, then the rules of Figure 1 only do not allow proof search to terminate. As the body $B$ of the *path* expression (*i.e.*, the displayed formula without $\mu$) is purely positive, a bipole can prove that a chosen purely negative predicate $S$ containing no fixed point expressions is an induction invariant (bipole: $\Uparrow B\,S\,x\,y \vdash S\,x\,y \Uparrow$), which means that we can use the certificate constructor $\mathsf{inv}(S,\cdot)$. Then we use another bipole as continuation certificate for this constructor to check that the invariant is adequate for the refutation of $\mathit{path}(t,u)$ (bipole: $\Uparrow S\,t\,u \vdash \cdot \Uparrow$).

Here, the invariant can be chosen so as to represent the fact of *not* belonging to the set $\mathscr{T} \times \{u\}$, where $\mathscr{T}$ is the reachable set of $\{t\}$.

**Example 6.** *In Figure 3, $(d)$ is not reachable from $(b)$, as witnessed by $\mathsf{inv}(S,\mathsf{bipole})$, where the invariant $S$ (built from the set $\{b,c\} \times \{d\}$) is*

$$\lambda x\lambda y.\,((x = b \wedge^+ y = d) \vee (x = c \wedge^+ y = d)) \supset f^-$$

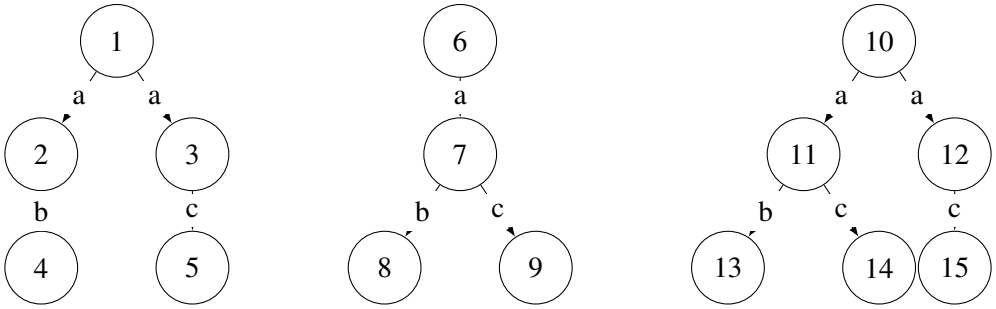


**Figure 4:** Non-(bi)similar noetherian labeled transition systems

# 5 Examples: certificates for labeled transition systems

Bisimilarity and similarity are important relationships in the domain of process calculus and model checking. To illustrate how these can be captured as FPCs, we first restrict our attention to the existence of a simulation between finite labeled transition systems; bisimilarity is then addressed by expanding on this presentation. We define $\cdot \dot{\longrightarrow} \cdot$ (for the LTS), *sim* (for simulated-by) and *bisim* (for bisimulated-by) as seen in Example 3.

## 5.1 Invariants as simulation certificates

We shall consider two cases: one where the underlying transition system is noetherian and one where it is not. An LTS is said to be *noetherian* if there is no infinite sequence of transitions $p_1 \xrightarrow{a_1} p_2 \xrightarrow{a_2} \cdots$ (in the setting of finite LTSs, this is equivalent to the absence of loops).

In the noetherian case, there is a decision procedure to determine whether or not one process is simulated by another: one simply attempts to incrementally check simulation at every point. This systematic search can be described using the clerks and experts of the decproc certificate, which allows a proof to be built from any number of bipoles (one for each unfolding of the simulation predicate, which formula is itself bipolar).

**Example 7.** *In Figure 4, the process* (1) *is simulated by the process* (6)*, as witnessed by the certificate decproc.*

In the more general (possibly non-noetherian) setting, we need to recall the formal definition of the simulation relation as a set. A binary relation $S$ is a simulation if whenever $\langle p,q\rangle \in S$ and whenever $p \xrightarrow{a} p'$ holds, then there exists a $q'$ such that $q \xrightarrow{a} q'$ holds and $\langle p',q'\rangle \in S$. We say that process $p$ is simulated by process $q$ if there is a simulation $S$ such that $\langle p,q\rangle \in S$.

Let $S$ be a finite set of pairs and let $\hat{S}$ be the purely positive expression $\lambda x \lambda y.\ \bigvee_{\langle p,q\rangle \in S}(x = p \wedge^+ y = q)$. As the body $B$ of the *sim* expression is a bipolar formula, a bipole can prove the closure condition for (finite) simulations (bipole: $\Uparrow \hat{S}\,x\,y \vdash B\,\hat{S}\,x\,y \Uparrow$), so we can use the certificate constructor co-inv$(\hat{S},\cdot)$. Once again, we use another bipole as continuation certificate to complete the proof that $p$ is simulated by $q$ (bipole: $\Uparrow \cdot \vdash \hat{S}\,p\,q \Uparrow$).

**Example 8.** *According to Figure 5, the set* $\{(21,23),(22,24)\}$ *is a simulation and, therefore, the process* (21) *is simulated by the process* (23)*. This corresponds to the following certificate*

$$\mathit{co\text{-}inv}\left(\lambda x \lambda y.\,(x = 21 \wedge^+ y = 23) \vee (x = 22 \wedge^+ y = 24), \mathit{bipole}\right).$$

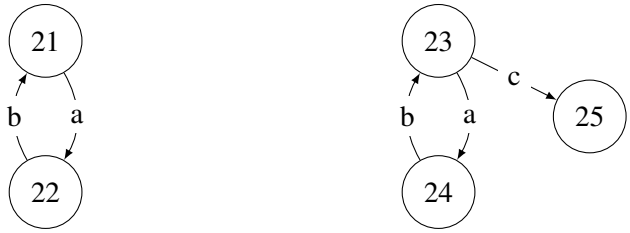


**Figure 5:** Non-noetherian labeled transition systems

Providing an entire invariant as part of a proof certificate or restricting to the case when an invariant is finite certainly limits what kinds of simulation relationships can be proved. In general, invariants will not be finite and, even when they are, they are large. It is for reasons such as this that there has been a great deal of work on bisimulation-up-to [17, 18]: generally, it is possible to discover and check a closure property of a much smaller relationship and then via various meta-theoretic properties, ensure that such closure properties entail the existence of a proper (bi)simulation.

## 5.2 Assertions as non-simulation certificates

Hennessy and Milner [11] provided a characterization of bisimulation in terms of an assertion language over modal operators $[a]$ and $\langle a\rangle$. The characterization states that two processes are bisimilar if and only if they satisfy the same assertion formulas. Thus, if $p$ and $q$ are not bisimilar, there is some assertion formula $A$ which is true for $p$ and not for $q$. Formally, we write $p \models A$ and $q \not\models A$.

It is possible to use such assertion formulas directly as proof certificates in the simpler and related problem of the absence of simulation, *i.e.* for theorems of the form $\vdash \neg sim(p,q)$. In that case, the assertion language needs only the diamond modality $\langle\cdot\rangle$ as well as the conjunction. More formally, let *Act* be a set of actions. The restricted set of assertions over *Act* is given by the recurrence $A := \bigwedge_{i\in I}\langle a_i\rangle A_i$, where $I$ is a finite set and $a_i \in Act$; that is, we have a strict alternation of (indexed) conjunctions and the diamond modality. The statement $p \models \bigwedge_{i\in I}\langle a_i\rangle A_i$ means that, for every $i \in I$, there exists a $q_i$ such that $p \xrightarrow{a_i} q_i$ and $q_i \models A_i$. We shall choose to write *true* for empty conjunctions and we can drop $\bigwedge_{i\in I}$ when $I$ is a singleton. Thus, $\langle a\rangle true$ stands for $\bigwedge_{i\in\{\bullet\}}\langle a\rangle\bigwedge_{j\in\{\}}\langle b_{i,j}\rangle A_{i,j}$.

Some of the clerks and experts needed for this interpretation of an assertion as a certificate are listed below; the rest of the definition can be taken from the async constructor.

$$\forall(a_i)_i\forall(A_i)_i\forall j.\,\mathrm{decide}_L(\textstyle\bigwedge_i\langle a_i\rangle A_i, \langle a_j\rangle A_j). \qquad \forall a\forall A.\,\forall_e(\langle a\rangle A, A, a).$$

$$\forall A.\,\supset_e(A, \mathsf{sync}(\mathsf{stop}), A). \qquad \forall T\forall A.\,\forall_e(A, A, T).$$

$$\forall a\forall A.\,\nu\text{-unfold}_L(\langle a\rangle A, \langle a\rangle A). \qquad \forall A.\,\mathrm{release}_L(A, A).$$

**Example 9.** *In Figure 4, the process* (6) *is not simulated by the process* (1)*: if* $\Xi$ *is the assertion formula* $\langle a\rangle(\langle b\rangle true \wedge \langle c\rangle true)$*, then* $6 \models \Xi$ *but* $1 \not\models \Xi$*.*

## 5.3 Assertions as non-bisimilarity certificates

It is possible to extend the FPC described in Section 5.2 to account for the absence of bisimulation in addition to the absence of simulation. As bisimilarity is finer than similarity, this will require a richer class of assertion formulas. The fact that it is a symmetric relation suggests that assertions should contain negations.

We could use full Hennessy-Milner logic (*i.e.* any arbitrary mix of $\langle\cdot\rangle$, $[\cdot]$, $\vee$ and $\wedge$ or, equivalently, $\langle\cdot\rangle$, $\wedge$ and $\neg$), but instead we choose the smaller but equivalent set of assertions defined by the following recurrence.

$$A := \bigwedge_{i\in I} B_i$$
$$B := \langle a_i\rangle A_i \mid \neg(\langle a_i\rangle A_i)$$

It can be shown that this set characterizes the same relation as full Hennessy-Milner logic. The statement $p \models \bigwedge_{i\in I} B_i$ means that, for every $i \in I$, $p \models B_i$; the statement $p \models \langle a\rangle A$ means that there exists a $q$ such that $p \xrightarrow{a} q$ and $q \models A$; and the statement $p \models \neg(\langle a\rangle A)$ means that $p \not\models \langle a\rangle A$.

Very little more is needed to extend the FPC to handle this certificate. We need to make sure that, in addition to certificates with a top-level $\langle\cdot\rangle$, decide$_L$ and $\nu$-unfold$_L$ allow (and propagate) certificates with a top-level $\neg\langle\cdot\rangle$. We also need an expert to consume $\neg$, and an expert to handle the additional $\wedge^-$ connective (see the definition of bisimilarity from Example 3). If we give these two roles to the same new expert, namely $\wedge^-{}_e$, the link between reflexivity and negations in the assertions appears clearly.

The resulting set of clerks and experts for theorems of the form $\vdash \neg bisim(p,q)$ is the following.

$$\forall A.\,\text{store}_L(A,A). \qquad \forall(B_i)_i\forall j.\,\text{decide}_L(\textstyle\bigwedge_i B_i, B_j).$$
$$\forall B.\,\nu\text{-unfold}_L(B,B). \qquad \forall a\forall A.\,\wedge^-{}_e(\ \langle a\rangle A, \langle a\rangle A, 1).$$
$$\forall a\forall A.\,\forall_e(\langle a\rangle A, A, a). \qquad \forall a\forall A.\,\wedge^-{}_e(\neg\langle a\rangle A, \langle a\rangle A, 2).$$
$$\forall T\forall A.\,\forall_e(A,A,T). \qquad \forall A.\,\supset_e(A,\mathsf{sync}(\mathsf{stop}),A).$$
$$\forall A.\,\text{release}_L(A,A).$$
$$\forall A.\,\exists_c(A,\lambda x.A). \qquad \forall A.\,\wedge^+{}_c(A,A).$$
$$\forall A.\,\mu\text{-unfold}_L(A,A). \qquad \forall A.\,\vee_c(A,A,A).$$
$$\forall A.\,=^s_c(A,A).$$

This FPC extension is conservative, in that it can still check a certificate for non-simulation.

**Example 10.** *In Figure 4, the processes* (6) *and* (10) *are similar but not bisimilar: if* $\Xi$ *is the generalized assertion formula* $\langle a\rangle\neg\langle b\rangle$true*, then* $10 \models \Xi$ *but* $6 \not\models \Xi$.

## 6 A reference proof checker

The framework for foundational proof certificates described in [13, 5] was based on proof theory without fixed point definitions. In that setting, a standard logic programming language (in that case, $\lambda$Prolog [14]) was an ideal prototyping language for implementing and testing FPCs. The FPCs described in this paper are not so easily implemented in standard logic programming languages since the unification of eigenvariables must be done alongside the usual unification of "logic variables" that makes proof reconstruction possible. The implementation of $\lambda$Prolog, for example, considers eigenvariables as constants during unification.

We have built a prototype proof checker for testing the FPCs described in this paper using the Bedwyr extension to logic programming [21, 4]. That system, originally designed to tackle various kinds of

model checking problems, provides the necessary unification for logic and eigenvariables along with backtracking search and support for $\lambda$-terms, $\lambda$-conversion and higher-order pattern unification.

One could have imagined implementing the non-augmented proof system $\mu F_0$ directly and in a sense, this is already done by Bedwyr itself. For example, if $(\mu B\bar{t})$ is a purely positive fixed point encoding a Prolog predicate, when the system is given the sequent $\vdash (\mu B\bar{t}) \Downarrow$, it would emulate the Prolog search. Similarly, if it is given the sequent $\Uparrow (\mu B\bar{t}) \vdash \Uparrow$, it would emulate a finite failing proof search. But, as anyone familiar with Prolog-style depth-first search knows, such proof search is limited in its effectiveness. For example, if one is attempting to prove that there is or is not a path between two points, a cycle in the underlying graph can make the search non-terminating. Bedwyr handles this with a loop-detection mechanism that can be embedded in the rules from Figure 2, making it a partial implementation of $\mu F$.

However, our goal with the $\mu F$ proof system is not to use it by itself, but together with clerks and experts, as the engine (as a "kernel") for checking already existing proof evidence. Since the logic of the existing Bedwyr system has no native support for proof objects, we implemented $\mu F^a$ as an "object logic", without using some native features such as loop-detection. The Bedwyr specification files that we use (available directly at `http://slimmer.gforge.inria.fr/bedwyr/pcmc/`, or from the authors' homepages) are rather direct translations of the inference rules in Figures 1 and 2 as well as of the various FPCs listed in the previous few sections. It has thus been easy for us to experiment and test FPCs.

While we have found the Bedwyr system to be useful for prototyping a proof checker, our proposal for FPC is not tied to any one particular implementation. Instead, the framework is defined using inference rules (such as found in Figures 1 and 2). Any system that can implement the logical principles required by such inference rules can be used as a proof checking FPC kernel.

# 7 Conclusion

We have taken the basic structure of *foundational proof certificates* that had been developed elsewhere for first-order logic and described how it could be imposed on a logic based on fixed points. The resulting logic is much richer (think of the difference between first-order logic and first-order arithmetic) and additional logic principles need to be accounted for in the description of proof certificates.

In the areas of model checking that we have discussed, proof evidence is often taken to be, say, a path through a graph, a set of pairs of nodes (satisfying certain closure conditions), or a Hennessy-Milner logic assertion formula. We have illustrated how each of these familiar objects can be easily transformed into hints to guide a proof checker though the construction of a detailed and complete sequent calculus proof. The architecture of focused proof systems and the clerk and expert predicates allow this conceptual gap (between familiar proof evidence and sequent calculus proofs) to be bridged in a flexible and natural fashion.

We have also provided a novel look at the proof theory foundations of model checking systems by basing our entire project on the $\mu$MALL variant of linear logic and on the notion of *switchable formulas*. This latter notion seems to provide an interesting demarcation between the logically simpler notion of model checking and the more general notion of (inductive and co-inductive) deduction.

**Acknowledgment.** We thank the reviewers for their detailed and useful comments on an earlier draft of this paper. This work has been funded by the ERC Advanced Grant ProofCert.

## References


[1] Jean-Marc Andreoli (1992): *Logic Programming with Focusing Proofs in Linear Logic*. J. of Logic and Computation 2(3), pp. 297–347.

[2] David Baelde (2012): *Least and greatest fixed points in linear logic*. ACM Trans. on Computational Logic 13(1).

[3] David Baelde, Kaustuv Chaudhuri, Andrew Gacek, Dale Miller, Gopalan Nadathur, Alwen Tiu & Yuting Wang (2014): *Abella: A System for Reasoning about Relational Specifications*. Journal of Formalized Reasoning 7(2).

[4] David Baelde, Andrew Gacek, Dale Miller, Gopalan Nadathur & Alwen Tiu (2007): *The Bedwyr system for model checking over syntactic expressions*. In F. Pfenning, editor: 21th Conf. on Automated Deduction (CADE), LNAI 4603, Springer, New York, pp. 391–397.

[5] Zakaria Chihani, Dale Miller & Fabien Renaud (2013): *Foundational proof certificates in first-order logic*. In Maria Paola Bonacina, editor: CADE 24: Conference on Automated Deduction 2013, LNAI 7898, pp. 162–177.

[6] Edmund M. Clarke, Orna Grumberg & Doron A. Peled (1999): *Model Checking*. The MIT Press, Cambridge, Massachusetts.

[7] Olivier Delande, Dale Miller & Alexis Saurin (2010): *Proof and refutation in MALL as a game*. Annals of Pure and Applied Logic 161(5), pp. 654–672.

[8] Andrew Gacek, Dale Miller & Gopalan Nadathur (2011): *Nominal abstraction*. Information and Computation 209(1), pp. 48–73.

[9] Jean-Yves Girard (1987): *Linear Logic*. Theoretical Computer Science 50, pp. 1–102.

[10] Jean-Yves Girard (1992): *A Fixpoint Theorem in Linear Logic*. An email posting to the mailing list linear@cs.stanford.edu.

[11] Matthew Hennessy & Robin Milner (1985): *Algebraic Laws for Nondeterminism and Concurrency*. JACM 32(1), pp. 137–161.

[12] Raymond McDowell & Dale Miller (2000): *Cut-elimination for a logic with definitions and induction*. Theoretical Computer Science 232, pp. 91–119.

[13] Dale Miller (2011): *A proposal for broad spectrum proof certificates*. In J.-P. Jouannaud & Z. Shao, editors: CPP: First International Conference on Certified Programs and Proofs, LNCS 7086, pp. 54–69.

[14] Dale Miller & Gopalan Nadathur (2012): *Programming with Higher-Order Logic*. Cambridge University Press.

[15] Dale Miller, Gopalan Nadathur, Frank Pfenning & Andre Scedrov (1991): *Uniform Proofs as a Foundation for Logic Programming*. Annals of Pure and Applied Logic 51, pp. 125–157.

[16] Dale Miller & Alwen Tiu (2005): *A proof theory for generic judgments*. ACM Trans. on Computational Logic 6(4), pp. 749–783.

[17] Robin Milner (1989): *Communication and Concurrency*. Prentice-Hall International.

[18] Damien Pous & Davide Sangiorgi (2011): *Enhancements of the bisimulation proof method*. In Davide Sangiorgi & Jan Rutten, editors: Advanced Topics in Bisimulation and Coinduction, Cambridge University Press, pp. 233–289.

[19] Peter Schroeder-Heister (1993): *Rules of Definitional Reflection*. In M. Vardi, editor: 8th Symp. on Logic in Computer Science, IEEE Computer Society Press, IEEE, pp. 222–232.

[20] Alwen Tiu, Gopalan Nadathur & Dale Miller (2005): *Mixing Finite Success and Finite Failure in an Automated Prover*. In: Empirically Successful Automated Reasoning in Higher-Order Logics (ESHOL'05), pp. 79–98.

[21] (2015): *The Bedwyr model checker*. Available at `http://slimmer.gforge.inria.fr/bedwyr/`.

# A On the augmented focusing proof system $\mu F^a$

It should be noted that, although the system presents two left rules for the connective $=$, one with the clerk $=^s_c$ for success and one with the clerk $=^f_c$ for failure, the implementation is usually expected to have one single unification facility that, given an equation, will or will not succeed, and which is tied to one single clerk. If the unification fails, the rule succeeds immediately without generating a premise certificate $\Xi_1$, and the constraint on the conclusion certificate $\Xi_0$ is actually the same as for success. Hence the clerk $=^f_c$ can be defined as an existential closure of $=^s_c$. Likewise, $\neq^s_c$ can be defined in terms of $\neq^f_c$.

$$=^f_c(\cdot) \equiv (\exists \Xi_1 . =^s_c(\cdot, \Xi_1)) \qquad \neq^s_c(\cdot) \equiv (\exists \Xi_1 . \neq^f_c(\cdot, \Xi_1))$$

It is also possible to remove the truth and falsity connectives, as we expect to have the equivalences

$$t^+ \equiv (a = a) \qquad f^+ \equiv (a = b)$$
$$f^- \equiv (a \neq a) \qquad t^- \equiv (a \neq b)$$

if $a$ and $b$ are distinct constants, hence the following:

$$t^+_e \equiv =^s_e \qquad t^+_c \equiv =^s_c \qquad f^+_c \equiv =^f_c$$
$$f^-_e \equiv \neq^f_e \qquad f^-_c \equiv \neq^f_c \qquad t^-_c \equiv \neq^s_c$$

Last, it is customary to leave clerks and experts out of rules with no premises (*i.e.* $t^+_e$, $=^s_e$, $f^-_e$, $\neq^f_e$, $f^+_c$, $=^f_c$, $t^-_c$ and $\neq^f_c$). This has the same effect as setting them to be always true.

The system presented in Figures 1 and 2 does not have these simplifications, but the Bedwyr-based implementation does.

# B A simple example of a $\mu F$ proof

The following proof can be seen as the justification that $\{1,3\} \subseteq \{1,2,3\}$. In particular, encode these two sets as the predicates (i.e., abstractions over formula):

$$\lambda x[x = 1 \vee x = 3] \quad \text{and} \quad \lambda x[x = 1 \vee x = 2 \vee x = 3].$$

The sequent calculus proof of inclusion can then be written as the following focused proof.

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{\vdash 1 = 1 \Downarrow}}{\vdash 1 = 1 \vee 1 = 2 \vee 1 = 3 \Downarrow}}{\Uparrow \vdash \Uparrow 1 = 1 \vee 1 = 2 \vee 1 = 3}}{\Uparrow \vdash 1 = 1 \vee 1 = 2 \vee 1 = 3 \Uparrow}}{\Uparrow x = 1 \vdash x = 1 \vee x = 2 \vee x = 3 \Uparrow} \quad \dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{\vdash 3 = 3 \Downarrow}}{\vdash 3 = 1 \vee 3 = 2 \vee 3 = 3 \Downarrow}}{\Uparrow \vdash \Uparrow 3 = 1 \vee 3 = 2 \vee 3 = 3}}{\Uparrow \vdash 3 = 1 \vee 3 = 2 \vee 3 = 3 \Uparrow}}{\Uparrow x = 3 \vdash x = 1 \vee x = 2 \vee x = 3 \Uparrow}}{\Uparrow x = 1 \vee x = 3 \vdash x = 1 \vee x = 2 \vee x = 3 \Uparrow}}{\Uparrow \vdash [x = 1 \vee x = 3] \supset [x = 1 \vee x = 2 \vee x = 3] \Uparrow}}{\Uparrow \vdash \forall x.[x = 1 \vee x = 3] \supset [x = 1 \vee x = 2 \vee x = 3] \Uparrow}$$