Proof search for full intuitionistic propositional logic through
a coinductive approach for polarized logic

José Espírito Santo (Centre of Mathematics, University of Minho),
Ralph Matthes (CNRS, Institut de Recherche en Informatique de Toulouse),
Luís Pinto (Centre of Mathematics, University of Minho)

August 3, 2020

Abstract
The approach to proof search dubbed “coinductive proof search”, and previously developed
by the authors for implicational intuitionistic logic, is in this paper extended to polarized
intuitionistic logic. As before, this includes developing a coinductive description of the search
spaces generated by a sequent, an equivalent inductive syntax describing the same space, and
decision procedures for inhabitation problems in the form of predicates defined by recursion
on the inductive syntax. The polarized logic can be used as a platform from which proof
search for other logics is understood. We illustrate the technique with LJT, a focused sequent
calculus for full intuitionistic propositional logic (including disjunction). For that, we have to
work out the “negative translation” of LJT into the polarized logic (that sees all intuitionistic
types as negative types), and verify that the translation gives a faithful representation of proof
search in LJT as proof search in the polarized logic.

1 Introduction and Motivation

An approach to proof search dubbed “coinductive proof search” has been developed by the authors
[EMP13, EMP16]. The approach is based on three main ideas: (i) the Curry-Howard paradigm of
representation of proofs (by typed $\lambda$-terms) is extended to solutions of proof-search problems (a
solution is a run of the proof search process that, if not completed, does not fail to apply bottom-
up an inference rule, so it may be an infinite object); (ii) two typed $\lambda$-calculi are developed for
the effect, one being obtained by a co-inductive reading of the grammar of proof terms, the other
being obtained by enriching the grammar of proof terms with a formal fixed-point operator to
represent cyclic behaviour, the first calculus acting as the universe for the mathematical definition
of concepts pertaining to proof search (e.g., the existence of solutions for a given logical sequent),
the second calculus acting as the finitary setting where algorithmic counterparts of those concepts
can be found; (iii) formal (finite) sums are employed throughout to represent choice points, so
even entire solution spaces are represented, both coinductively and finitarily.

The approach was developed systematically for intuitionistic implicational logic, delivering
new solutions to inhabitation and counting problems, and proofs of the state-of-the-art coherence
theorems, in the simply typed $\lambda$-calculus [ESMP19]. But the approach also showed to be a vehicle
for the investigation of new questions, like the various concepts of finiteness suggested by proof
search [EMP19].

The goal of this paper is to extend this approach to full (in the sense of containing all connectives),
polarized, intuitionistic propositional logic [Sim14, San16], and proof search in a full-fledged
focused sequent calculus. Polarized logic can be used as a platform from which proof search for
other logics is understood [LM09]. We extend our approach to polarized logic with a view to
obtaining results about proof search for full, intuitionistic propositional logic.

Coinductive proof search extends smoothly to polarized logic, with an occasional insight coming
from focusing: formal sums are not needed in the inversion phases, and the infinity of solutions
must go infinitely often through stable sequents. Only the luxuriant syntax puts a notational challenge, and we make a proposal for that. This smoothness is a sign of the robustness of the approach, that was originally designed and tested in a relatively simple logic (that, however, unlike the propositional logic of Horn clauses, has to deal with a dynamic context). As a result of coinductive analysis, we obtain decidability of provability in the polarized logic, with our typical two-staged decision procedure: a function that calculates the finitary representation (in the calculus with formal fixed points) of the solution space of the given logical sequent, composed with a syntax-directed, recursive predicate that tests the existence of proofs/inhabitants.

As said, from the results about the polarized logic, we can extract results for other logics. We illustrate the technique with LJT, a focused sequent calculus for full intuitionistic propositional logic (including disjunction) \cite{Her95b,DP96}. For that, we define the “negative translation” of LJT into the polarized logic (that sees all intuitionistic formulas as negative formulas). While the translation of formulas is mostly dictated by polarity, there are subtle problems with a definition of the translation of proof terms without knowing the logical sequent they witness (see the definitions of DLV(t) and atomic and positive spines in Section 3). Soundness of a translation is its first aim, but we also need to verify that the translation gives a faithful representation of proof search in LJT as proof search in the polarized logic. This is an interesting result in itself, and benefits from the language of proof terms developed for polarized logic in \cite{San16}.

Plan of the paper. The presentation of polarized logic from \cite{San16} is reviewed in Section 2 and named PIPL. Before embarking on the coinductive proof search for PIPL (Sections 4 and 5), we show why we can study LJT via PIPL in Section 6. Applications to full intuitionistic logic are extracted in Section 6, Section 7 concludes.

2 Background on the system PIPL of polarized propositional logic

We introduce a variant of cut-free $\lambda_G^\pm$ \cite{San16} called PIPL (for polarized intuitionistic propositional logic).

Formulas of PIPL are as follows (unchanged from $\lambda_G^\pm$):

\[
\begin{align*}
\text{(formulas)} & \quad A ::\ N \mid P \\
\text{(negative)} & \quad N, M ::\ C \mid a^- \\
\text{(composite negative)} & \quad C ::\ \uparrow P \mid P \supset N \mid N \land M \\
\text{(positive)} & \quad P, Q ::\ a^+ \mid \downarrow N \mid \perp \mid P \lor Q
\end{align*}
\]

Here, we assume a supply of (names of) atoms, denoted typically by $a$—the markers $-$ and $+$ for polarity are added to the atom (name) as superscripts, giving rise to negative resp. positive atoms. The symbols $\perp$, $\land$ and $\lor$ obviously stand for falsity, conjunction and disjunction, $\supset$ stands for implication, and $\uparrow$ and $\downarrow$ are polarity shifts (as they are commonly denoted in the literature). We call right formulas or R-formulas positive formulas and negative atoms. The set of formulas is thus partitioned in two ways: into negative and positive formulas, and into composite negative and right formulas. The second partitioning plays an important role in PIPL, more than in $\lambda_G^\pm$. We also use the notion of left formulas or L-formulas: they are either negative formulas or positive atoms.

Proof terms of PIPL are organized in five syntactic categories as follows:

\[
\begin{align*}
\text{(values)} & \quad v ::\ z \mid \text{thunk}(t) \mid \text{inl}^i(v) \\
\text{(terms)} & \quad t ::\ [e] \mid [\text{cothunk}(p)] \mid v :: s \mid i :: s \\
\text{(co-values/spines)} & \quad s ::\ nil \mid \text{cothunk}(p) \mid v :: s \mid i :: s \\
\text{(co-terms)} & \quad p ::\ z^{a^+} . e \mid x^{N}. e \mid \text{abort}^A \mid [p_1, p_2] \\
\text{(stable expressions)} & \quad e ::\ \text{dlv}(t) \mid \text{ret}(v) \mid \text{coret}(x, s)
\end{align*}
\]

where $i \in \{1, 2\}$, and $z$ and $x$ range over countable sets of variables assumed to be disjoint, called positive resp. negative variables. The syntax deviates from $\lambda_G^\pm$ \cite{San16} Figure 4] in the following ways: the letters to denote values and covalues are now in lower case, the two expressions to type
We also consider sequents without proof-term annotations, i.e., $\Gamma \vdash \cdot$ The rules, given in Fig. 1, are the obvious adaptations of the ones in [San16, Figures 1–3] (omitting $\rho, \rho'$ etc. will range over the cut rules are absent, and the last form of values (the injections) and $\text{abort}$ come with type information, as well as the binding occurrences of variables in the first two forms of co-terms—all the other syntax elements do not introduce variable bindings, in particular, there is no binding in $\lambda p$ or $\text{coret}(x, s).$ Often we refer to all proof terms of PIPL as expressions, and use letter $T$ to range over expressions in this wide sense ($T$ being reminiscent of terms, but not confined to the syntactic category $t$). To shorten notation, we communicate $\langle t_1, t_2 \rangle$ and $[p_1, p_2]$ as $\langle t_i \rangle_i$ and $[p_i]_i$, respectively.

We also use the typical letters for denoting elements of the syntactic categories as sorts: let $S := \{v, t, s, p, e\}$ be their set, and use letter $\tau$ to denote any element of $S$.

Since proof terms of PIPL come with some extra type information as compared to $\lambda^\pm_G$, the typing rules will be adjusted accordingly. The typing relation will also be slightly reduced: it is assumed that the Focus rule of $\lambda^\pm_G$ (the one typing the $\text{coret}$ construction for proof terms) only applies if the right-hand side formula is an R-formula. This also means that focus negative left sequents can be restricted to R-formulas on the right-hand side, which we therefore do in PIPL.

There are five forms of sequents, one for each syntactic category $\tau$ of proof terms (the full names and the rationales of the categories are found in [San16]):

| Rule Type          | Formulation                                                                 |
|--------------------|-----------------------------------------------------------------------------|
| (focus negative left) | $\Gamma \vdash \cdot$                                                      |
| invert positive left       | $\Gamma \vdash v : P$                                                     |
| (stable)              | $\Gamma \vdash t : N$                                                     |

The rules, given in Fig. 1 are the obvious adaptations of the ones in [San16] Figures 1–3 (omitting the cut rules), given the more annotated syntax and the mentioned restrictions to R-formulas in some places. We recall that $\Gamma$ is a context made of associations of variables with left formulas that respect polarity, hence these associations are either $\cdot : a^+$ or $x : N$ (in other words, positive variables are assigned atomic types only). The extra annotations ensure uniqueness of typing in that, given the shown context, type and term information, there is at most one formula that can replace any of the placeholders in $\Gamma \vdash \cdot$, $\Gamma \vdash v : \cdot$, $\Gamma \vdash p : \cdot \Rightarrow \cdot$, $\Gamma \vdash t : \cdot$, and $\Gamma \vdash e : \cdot$. We also consider sequents without proof-term annotations, i.e., $\Gamma \vdash \cdot$, $\Gamma \vdash \cdot \Rightarrow \cdot$, $\Gamma \vdash t : \cdot$, and $\Gamma \vdash e : \cdot$, that we will call logical sequents. The letters $\rho, \rho'$ etc. will range over
\( \Gamma \vdash R \), with an R-formula on the right-hand side. Those will be called R-stable sequents. (Such logical sequents cannot be proven by a proof term of the form \( \text{dv}(t) \).) Results about all forms of sequents can sometimes be presented uniformly, with the following notational device: If \( \sigma \) is any logical sequent and \( T \) a proof term of the suitable syntactic category, let \( \sigma(T) \) denote the sequent obtained by placing “\( T \)” properly into \( \sigma \), e.g., if \( \sigma = (\Gamma \mid P \Rightarrow A) \), then \( \sigma(p) = (\Gamma \mid p : P \Rightarrow A) \). (the parentheses around sequents are often used for better parsing of the text). We sometimes indicate the syntactic category \( \tau \) of \( T \) as upper index of \( \sigma \), e.g., an arbitrary logical sequent \( \Gamma \vdash A \) is indicated by \( \sigma^\tau \).

We also use the set \( S \) of sorts to give a more uniform view of the different productions of the grammar of PIPL proof terms. E.g., we consider \( \text{thunk}(\cdot) \) as a unary function symbol, which is typed/sorted as \( t \to v \), to be written as \( \text{thunk}(\cdot) : t \to v \). As another example, we see co-pairing as a binary function symbol \( [\cdot, \cdot] : p, p \Rightarrow p \). This notational device does not take into account variable binding, and we simply consider \( z^{\sigma^\tau} \cdot \) as a unary function symbol for every \( z \) and every \( \sigma \). The positive variables \( z \) have no special role either in this view, so they are all nullary function symbols (i.e., constants) with sort \( v \). Likewise, for every negative variable \( x \), \( \text{coret}(x, \cdot) \) is a unary function symbol sorted as \( s \to e \). We can thus see the definition of proof terms of PIPL as based on an infinite signature, with function symbols \( f \) of arities \( k \leq 2 \). The inductive definition of proof terms of PIPL can then be depicted in the form of one rule scheme:

\[
\begin{array}{c}
f : \tau_1, \ldots, \tau_k \to \tau \\
T_1 : \tau_i, 1 \leq i \leq k \\
f(T_1, \ldots, T_k) : \tau
\end{array}
\]

Later we will write \( f(T_1)_i \) in place of \( f(T_1, \ldots, T_k) \) and assume that \( k \) is somehow known. Instead of writing the \( k \) hypotheses \( T_i : \tau_i \), we will then just write \( \forall i, T_i : \tau_i \).

3 System \( LJT \) of intuitionistic logic with all propositional connectives

One of the interests of polarized logic is that it can be used to analyze other logics \([LM09]\)—this is also true of PIPL. We will use PIPL to analyze intuitionistic propositional logic (IPL), specifically a focused sequent calculus for IPL named \( LJT \). The best known variant of \( LJT \) is the one for implication only \([Her95a]\), but here we need a variant including conjunction and disjunction as well \([Her95b, DP96]\). We will call our own variant \( LJT_R \).

Formulas of \( LJT_R \) are as follows:

\[
\begin{align*}
\text{(intuitionistic formulas)} & \quad A, B ::= A \lor B \mid A \land B \mid R \\
\text{(right intuitionistic formulas)} & \quad R ::= a \mid \bot \mid A \lor B
\end{align*}
\]

where \( a \) ranges over atoms, of which an infinite supply is assumed. A positive intuitionistic formula, \( P_i \), is a non-atomic right intuitionistic formula.

Proof terms of \( LJT_R \) are organized in three syntactic categories as follows:

\[
\begin{align*}
\text{(terms)} & \quad t ::= \lambda x^A.t \mid \{t_1, t_2\} \mid e \\
\text{(expressions)} & \quad e ::= xs \mid \text{in}_i^A(t) \\
\text{(spines)} & \quad s ::= \text{nil} \mid t :: s \mid i :: s \mid \text{abort}^R \mid [x_1^{A_1} . e_1, x_2^{A_2} . e_2]
\end{align*}
\]

where \( i \in \{1, 2\} \), and \( x \) ranges over a countable set of variables. We will refer to \( e_1 \) and \( e_2 \) in the latter form of spines as arms. Proof terms in any category are ranged over by \( T \).

There are three forms of sequents, \( \Gamma \Rightarrow t : A \) and \( \Gamma \vdash e : R \) and \( \Gamma[s : A] \vdash R \), where, as usual, \( \Gamma \) is a context made of associations of variables with formulas. Therefore, a logical sequent \( \sigma \) in \( LJT_R \) may have three forms: \( \Gamma \Rightarrow \) and \( \Gamma \vdash R \) and \( \Gamma[A] \vdash R \). The latter two forms require a right formula to the right of the turnstile. The full definition of the typing rules of \( LJT_R \) is given in Fig. 2. As for PIPL, the annotations guarantee that there is at least one formula that can replace the placeholders in \( \Gamma \Rightarrow t : \cdot \) and \( \Gamma \vdash e : \cdot \) and \( \Gamma[s : A] \vdash \cdot \).

The characteristic feature of the design of \( LJT_R \) is the restriction of the type of spines to right formulas. Since the type of \( \text{nil} \) is atomic, spines have to be “long”; and the arms of spines
cannot be lambda-abstractions nor pairs, which is enforced by restricting the arms of spines to be expressions, rather than general terms: this is the usefulness of separating the class of expressions from the class of terms. In the typing rules, the restriction to right formulas is generated at the select rule (the typing rule for $xs$); and the long form is forced by the identity axiom (the typing rule for nil) because it applies to atoms only.

We remark that the select rule is also found under the name $Cont$ (from contraction) \cite{Her95a, Her95b}. A corresponding rule named $select$ is found in \cite{DP96}. The identity axiom found in the literature \cite{Her95a, Her95b, DP96} applies to arbitrary formulas. We could not find in the literature the restriction of cut-free LJT we consider here, but Ferrari and Fiorentini \cite{FF19} consider a presentation of IPL that enforces a similar use of right formulas, in spite of being given in natural deduction format and without proof terms. It is easy to equip this natural deduction system with proof terms and map it into LJT$_R$: the technique is fully developed in \cite{San16} for polarized logic, but goes back to \cite{DP96}. Since the just mentioned system \cite{FF19} is complete for provability, so is LJT$_R$.

System LJT$_R$ can be embedded in PIPL. We define the negative translation $(\cdot)^*: LJT_R \rightarrow PIPL$ in Fig. 3 comprising a translation of formulas and a translation of proof terms.

The translation of formulas uses an auxiliary translation of right intuitionistic formulas $R: R^o$ is a right formula (and specifically, $P^o$ is a positive formula). An intuitionistic formula $A$ is mapped to a negative formula $A^*$, hence the name of the translation. At the level of proof terms: terms (resp. spines, expressions) are mapped to terms (resp. spines, stable expressions). Definitions like $e^* = \top e^\tau$, if $e$ is atomic $e^* = [e^*]$, if $e$ is positive $(\text{abort}^R)^* = \text{cothunk} (\text{abort}^R)$ $(t :: s)^* = \text{thunk}(t^*) :: s^*$ $(i :: s)^* = i :: s^*$
DLV("e") = e and DLV(t) = dv(t) otherwise. Its derived typing rule is that \( \Gamma \succeq \text{DLV}(t) : N \) follows from \( \Gamma \Rightarrow t : N \).

The translation of proof terms is defined for legal proof terms in \( \text{LJT}_R \) only: \( T \) is legal if every expression \( e \) occurring in \( T \) is either atomic or positive; an expression \( xs \) is atomic (resp. positive) if \( s \) is atomic (resp. positive), whereas an injection is positive; and a spine \( s \) is atomic (resp. positive) if every “leaf” of \( s \) is nil or \( \text{abort}^a \) (resp. an injection or \( \text{abort}^e \)). Only when translating a legal \( T \) can we apply the definition of \( (\cdot)^* \) to \( e \) as a term.

Formally, the inductive definition of atomic and positive spines is as follows:

- \( \text{nil} \) is atomic; \( \text{abort}^a \) is atomic; if \( s \) is atomic, then \( t :: s \) and \( i :: s \) are atomic; if, for each \( i = 1,2 \), \( e_i = y_is_i \) and \( s_i \) is atomic, then \([x_1^{A_1}e_1,x_2^{A_2}e_2] \) is atomic.
- \( \text{abort}^p \) is positive; if \( s \) is positive, then \( t :: s \) and \( i :: s \) are positive; if, for each \( i = 1,2 \), \( e_i = y_is_i \) and \( s_i \) is positive, or \( e_i = \text{in}_i^A(t) \), then \([x_1^{A_1}e_1,x_2^{A_2}e_2] \) is positive.

Suppose \( \Gamma[s : A] \succeq R \) is derivable. If \( R = a \) (resp. \( R = P \)) then \( s \) is atomic (resp. positive). Hence any typable proof term of \( \text{LJT}_R \) is legal. Moreover, if \( \Gamma \vdash e : R \) then if \( e \) is atomic, \( R = a \) and if \( e \) is positive, \( R = P \).

The negative translation is easily seen to be injective. In order to state other properties of the translation, we define the logical PIPL sequent \( \sigma^* \) for every logical \( \text{LJT}_R \) sequent \( \sigma^* \): \( (\Gamma \Rightarrow A)^* = (\Gamma^* \Rightarrow A^*) \) and \( (\Gamma \vdash R)^* = (\Gamma^*[A^*] \vdash R^*) \).

**Proposition 1 (Soundness)** For all \( T = t,e,s \) in \( \text{LJT}_R \): if \( \sigma(T) \) is derivable in \( \text{LJT}_R \) then \( \sigma^*(T^*) \) is derivable in PIPL.

**Proof** By simultaneous induction on derivations for \( \sigma(T) \). \( \Box \)

For the converse property (faithfulness), we need to understand better the image of the negative translation, which we will call the *-fragment* of PIPL. Consider the following subclass of formulas in PIPL:

\[
\begin{align*}
(\text{positive } \sigma^* \text{-formulas}) & \\
M, N & ::= a^- | \uparrow P | \downarrow N \cup M | N \land M \\
(\text{positive } \sigma \text{-formulas}) & \\
P & ::= \bot | \downarrow N \lor \downarrow M
\end{align*}
\]

The positive \( \sigma \)-formulas are separated because they are useful to define \( \sigma \)-formulas \( R \), which are either atoms \( a^- \) or positive \( \sigma \)-formulas \( P \). A *-formula \( N \) is a negative formula; a positive \( \sigma \)-formula \( P \) is a positive formula; a \( \sigma \)-formula \( R \) is a right formula. The negative translation, at the level of formulas, is a bijection from intuitionistic formulas to *-formulas, from positive intuitionistic formulas to positive \( \sigma \)-formulas; and from right intuitionistic formulas to \( \sigma \)-formulas. The respective inverse maps are denoted \(| \cdot | \): they just erase the polarity shifts and the minus sign from atoms.

If we are interested in deriving in PIPL logical sequents of the form \( \sigma^* \) only, then some obvious cuts can be applied to the grammar of proof terms of PIPL:

- As to values, there are no \( z \)'s, and thunks can only occur inside injections, so that \( \text{in}_i^P(\text{thunk}(t)) \) is the only possible form of value.
- As to terms, \( \lambda \)-abstraction can be reduced to the form \( \lambda(x^N,e) \).
- As to spines, the form \( v :: s \) can be constrained to the form \( \text{thunk}(t) :: s \).
- As to co-terms, the only possible forms are \( \text{abort}^R \) and \([x_1^{N_1}e_1,x_2^{N_2}e_2] \).

But then, values are only used in returns, and co-terms are only used in co-thunks, so those two classes may disappear, yielding the following grammar \( \mathcal{G} \) of *-proof terms:

\[
\begin{align*}
(\text{*-terms}) & \\
t & ::= [e] | [\epsilon] \supset \lambda(x^N,e)(t_1,t_2) \\
(\text{*-spines}) & \\
s & ::= \text{nil} | \text{cothunk}(\text{abort}^R) | \text{cothunk}([x_1^{N_1}e_1,x_2^{N_2}e_2]) | \text{thunk}(t) :: s | i :: s \\
(\text{*-expressions}) & \\
e & ::= \text{dlv}(t) | \text{ret}(\text{in}_i^P(\text{thunk}(t))) | \text{coret}(x,s)
\end{align*}
\]
A legal *-proof term is one where $\text{dlv}(t)$ is only allowed as the body of a $\lambda$-abstraction. Legal expressions are generated by a restricted variant of the grammar above: $\text{dlv}(t)$ is forbidden as a *-expression per se, but, as a compensation, we introduce a second form of $\lambda$-abstraction, $\lambda(x^N.\text{dlv}(t))$.

There is a forgetful map from legal *-terms (resp. legal *-spines, legal *-expressions) to terms (resp. spines, expressions) of $LJT_R$, given in Fig. 4 that essentially erases term decorations. The negative translation only generates legal *-proof terms; and, since the negative translation is just a process of decoration, the forgetful map is left inverse to it: $|T^*| = T$.

**Proposition 2 (Faithfulness)** For all $T$ in PIPL: if $\sigma^*(T)$ is derivable in PIPL, then $T$ is legal and $\sigma(|T|)$ is derivable in $LJT_R$ and $|T^*| = T$.

**Proof** By simultaneous induction on $T = t, s, r$ as generated by the grammar $\mathcal{G}$ above. □

By faithfulness and injectivity of the negative translation, the implications in Proposition 1 are in fact equivalences. Moreover, we obtain the following reduction of counting and inhabitation problems:

**Corollary 3 (Reduction of problems)**

1. There is a bijection between the set of those $T \in LJT_R$ such that $\sigma(T)$ is derivable in $LJT_R$ and the set of those $T' \in \text{PIPL}$ such that $\sigma^*(T')$ is derivable in PIPL.

2. There is $T \in LJT_R$ such that $\sigma(T)$ is derivable in $LJT_R$ iff there is $T' \in \text{PIPL}$ such that $\sigma^*(T')$ is derivable in PIPL.

**Proof** We prove the first item. The negative translation is the candidate for the bijection. Due to Proposition 1 it maps from the first set to the second. We already observed that the translation is injective. Proposition 2 guarantees that the translation is also surjective. The second item is an immediate consequence of the first. □

## 4 Coinductive approach to proof search in the polarized system PIPL

In this section, we adapt our coinductive approach to proof search from implicational intuitionistic logic to PIPL. Due to the high number of syntactic categories and different constructors for proof terms, we use the extra notational devices from the end of Section 2 to ensure a uniform presentation of mostly similar rules that appear in definitions. Our previous development sometimes departs from such a uniformity, which is why we also widen the grammar of “forests”. This in turn asks for a mathematically more detailed presentation of some coinductive proofs that are subtle but lie at the heart of our analysis. (For reasons of limited space, that presentation was moved into Appendix A.0)
4.1 Search for inhabitants in PIPL, coinductively

System PIPL\(^{co}\) extends the proof terms of PIPL in two directions: there is a coinductive reading of the rules of the grammar of proof terms, and formal sums are added to the grammar as means to express alternatives. This general idea is refined when applied to the focused system PIPL: the coinductive reading will be attached to stable expressions; and the formal sums are not added to the categories of (co)terms, since they serve to represent the inversion phase in proof search, where choice is not called for.

The expressions in the wide sense of PIPL\(^{co}\) are called forests and ranged by the letter \(T\). They comprise five categories introduced by the simultaneous coinductive definition of the sets \(\psi^\omega, \tau^\omega, s^\omega, p^\omega, \) and \(e^\omega\). However, we will continue to use the sorts \(\tau\) taken from the set \(S\) that was introduced for PIPL. This allows us to maintain the function-symbol view of PIPL with the same symbols \(f\) that keep their typing/sorting. As said, only for the classes of values, spines and expressions, we also add finite (heterogeneous) sums, denoted with the multiary function symbols \(\Sigma^\tau\) for \(\tau \in \{v, s, e\}\). The definition of the set of forests, i.e., the expressions (in a wide sense) of PIPL\(^{co}\) can thus be expressed very concisely as being obtained by only two rule schemes:

\[
\begin{align*}
& f : T_1, \ldots, T_k \to \tau & \forall i, T_i : \tau & \text{coinductive if } \tau = e \\
& f(T_1, \ldots, T_k) : \tau & \sum_i T_i : \tau & \tau \in \{v, s, e\}
\end{align*}
\]

The doubly horizontal line indicates a possibly coinductive reading. As a first step, we read all these inference rules coinductively, but in a second step restrict the obtained infinitary expressions to obey the following property: infinite branches must go infinitely often through the \(e\)-formation rules coming from PIPL, i.e., those depicted as unary function symbols \(f : \tau_1 \to e\) (also called the inherited \(e\)-formation rules—those for \(\mathrm{dv}(\cdot), \mathrm{ret}(\cdot)\) and \(\mathrm{coret}(x, \cdot)\)).

This can be expressed as the parity condition (known from parity automata where this is the acceptance condition) based on priority 2 for any rule for those \(f : \tau_1 \to e\) and priority 1 for all the others. The parity condition requires that the maximum of the priorities seen infinitely often on a path in the (forest) construction is even, hence infinite cycling through the other syntactic categories and the summing operation for \(e\)-expressions is subordinate to infinite cycling through the inherited \(e\)-formation rules. Put less technically, we allow infinite branches in the construction of forests, but infinity is not allowed to come from infinite use solely of the “auxiliary” productions (for \(\tau \neq e\)) or the additional sum operator for \(e\), thus, in particular ruling out infinite pairing with angle brackets, infinite copairing with brackets or infinite spine composition by way of one of the :: constructors—all of which would never correspond to typable proof terms—and also ruling out infinite stacks of finite sums.

Sums \(\sum_i T_i\) are required to be finite and therefore may also be denoted by \(T_1 + \ldots + T_k\), leaving \(\tau\) implicit. We write \(\emptyset\) (possibly with the upper index \(\tau\) that obviously cannot be inferred from the summands) for empty sums. Sums are treated as sets of alternatives (so they are identified up to associativity, commutativity and idempotency—that incorporates \(\alpha\)-equivalence (this is still a \(\lambda\)-calculus, the presentation with function symbols \(f\) is a notational device) and bisimilarity coming from the full coinductive reading in the first step of the construction).

A rich class of examples of forests is provided by Def. (\ref{def:forests}) below.

We now define an inductive notion of membership, hence restricting the notion we had in our previous papers on implicational logic.

**Definition 4 (membership)** A PIPL-expression \(T\) is a member of a forest \(T'\) when the predicate \(\text{mem}(T, T')\) holds, which is defined inductively as follows.

\[
\begin{align*}
& \forall i, \text{mem}(T_i, T'_i) & \text{mem}(T, T'_j) & \text{mem}(f(T_1, \ldots, T_k), f(T'_1, \ldots, T'_k)) & \text{for some } j
\end{align*}
\]

The intuition of this definition is obviously that the sums expressed by \(\sum_i T_i\) represent alternatives out of which one is chosen for a concrete member. It can equally be seen as a recursive definition by recursion over \(T_0\).
For all the five categories of logical sequents $\sigma$, this property holds since we tacitly assume that the sum operators are tagged with the respective syntactic category.

For a forest $T$, we call finite extension of $T$, which we denote by $E_{\text{fin}}(T)$, the set of the (finite) members of $T$, i.e., $E_{\text{fin}}(T) = \{ T_0 \mid \text{mem}(T_0, T) \}$. Properties of special interest in this paper are: $\text{exfin}(T)$ iff $T \in \tau^{co}_{\Sigma}$, and $\text{nofin}$ the complement of $\text{exfin}$. These predicates play an important role in Section 5. Analogously to our previous work [ESMPT10], we inductively characterize $\text{exfin}$, and we coinductively characterize $\text{nofin}$:

**Definition 5 (exfin and nofin)**

\[
\begin{array}{c}
\forall i. \text{exfin}(T_i) \quad \text{exfin}(f(T_i)) \quad \text{exfin}(\sum_i T_i) \\
\frac{}{\text{exfin}(\sum_i T_i)} \\
\end{array}
\]

\[
\begin{array}{c}
\forall i. \text{nofin}(T_i) \quad \text{nofin}(f(T_i)) \quad \text{nofin}(\sum_i T_i) \\
\frac{}{\text{nofin}(\sum_i T_i)} \\
\end{array}
\]

In Appendix A, it is shown that $\text{exfin} = \text{exfinext}$ and $\text{nofin} = \text{nofinext}$, in other words, we indeed have an inductive resp. coinductive characterization, and as immediate consequence $\text{exfin}$ and $\text{nofin}$ are complementary predicates.

Now, we are heading for the infinitary representation of all inhabitants of any logical sequent $\sigma$ of PIPL as a forest whose members are precisely those inhabitants (to be confirmed in Prop. 8). For all the five categories of logical sequents $\sigma^+$, we define the associated solution space $S(\sigma^+)$ as a forest, more precisely, an element of $\tau^{co}_{\Sigma}$. This is supposed to represent the space of solutions generated by an exhaustive and possibly non-terminating search process applied to that given logical sequent $\sigma^+$. This is by way of the following simultaneous coinductive definition. It is simultaneous for the five categories of logical sequents. For each category, there is an exhaustive case analysis on the formula argument.

**Definition 6 (Solution spaces)**

As announced in the preceding paragraph, we define a forest $S(\sigma^+) \in \tau^{co}_{\Sigma}$ for every logical sequent $\sigma^+$, by simultaneous coinduction for all the $\tau \in S$. The definition is found in Fig. 3, where in the clauses for $S(\Gamma \mid a^+ \Rightarrow A)$ resp. $S(\Gamma \mid \downarrow N \Rightarrow A)$, the variables $z$ resp. $x$ are supposed to be “fresh”, and since the names of bound variables are considered as immaterial, there is no choice involved in this inversion phase of proof search, as is equally the case for $S(\Gamma \Rightarrow \cdot)$—as should be expected for inversion rules.

**Lemma 7 (Well-definedness of $S(\sigma)$)** For all logical sequents $\sigma$, the definition of $S(\sigma)$ indeed produces a forest.
**Proof** Well-definedness is not at stake concerning productivity of the definition since, clearly, every corecursive call is under a constructor. As is directly seen in the definition, the syntactic categories are respected. Only the parity condition requires further thought. In Appendix A.2 we prove it by showing that all the “intermediary” corecursive calls to $\mathcal{S}(\sigma)$ in the calculation of $\mathcal{S}(\Gamma \vdash A)$—which is the only case that applies inherited $\epsilon$-formation rules—lower the “weight” of the logical sequent, until a possible further call to some $\mathcal{S}(\Gamma' \vdash A')$.

In general, the members of a solution space are exactly the inhabitants of the sequent:

**Proposition 8 (Adequacy of the coinductive representation)** For each $\tau \in S$, logical sequent $\sigma^\tau$ and $T$ of category $\tau$, $\text{mem}(T, \mathcal{S}(\sigma))$ iff $\sigma(T)$ (proof by induction on $T$).

The following definition is an immediate adaptation of the corresponding definition in [ESMP19].

**Definition 9 (Inessential extension of contexts and R-stable sequents)**

1. $\Gamma \leq \Gamma'$ iff $\Gamma \subseteq \Gamma'$ and $|\Gamma| = |\Gamma'|$, with the set $|\Delta| := \{L \mid \exists x, (xz : L) \in \Delta\}$ of assumed types of $\Delta$ for an arbitrary context $\Delta$ (where we write $xz$ for an arbitrary variable). In other words, $\Gamma \leq \Gamma'$ if $\Gamma'$ only has extra bindings w. r. t. $\Gamma$ that come with types that are already present in $\Gamma$.

2. $\rho \leq \rho'$ iff for some $\Gamma \leq \Gamma'$ and for some right-formula $R$, $\rho = (\Gamma \vdash R)$ and $\rho' = (\Gamma' \vdash R)$.

If $\rho \leq \rho'$, we seek to transform the forest $\mathcal{S}(\rho)$ into the forest $\mathcal{S}(\rho')$: both represent the entire search space for solutions, and $\rho'$ offers extra proof possibilities just by the availability of extra names of hypotheses that are already present in $\rho$. In essence, whatever is done with such a hypothesis in the source forest ought to be done in the target forest with each of the new names instead of the original one in addition, and this independently for each occurrence of that hypothesis. This operation can be defined not only for some $\mathcal{S}(\rho)$ but for any forest $T$, and its result is denoted by $[\rho'/\rho]T$. However, we only need to know that $\mathcal{S}(\rho')$ and $[\rho'/\rho]\mathcal{S}(\rho)$ are indeed the same forest modulo $\alpha$-equivalence, bisimulation and our view of sums as sets of alternatives. More details are found in Appendix A.3.

### 4.2 Search for inhabitants in $\text{PIPL}^\infty$, inductively

We are going to present a finitary version of $\text{PIPL}^\infty$ in the form of a system $\text{PIPL}^\infty_{\text{fp}}$ of finitary forests that are again generically denoted by letter $T$. We are again making extensive use of our notational device introduced in Section 2. The letter $f$ ranges over the function symbols in this specific view on $\text{PIPL}$. Summation is added analogously as for $\text{PIPL}^\infty$, and there are two more constructions for the category of expressions.

\[
\begin{align*}
\frac{f : \tau_1, \ldots, \tau_k \to \tau}{f(T_1, \ldots, T_k) : \tau} & \quad \frac{\forall \iota. T_i : \tau_i}{\forall \iota. T_i : \tau} & \quad \frac{\forall \iota. T_i : \tau}{\sum_i T_i : \tau} & \quad \frac{\tau \in \{v, s, e\}}{X^\rho : e} & \quad \frac{T : e}{\text{gfp} T : e}
\end{align*}
\]

where $X$ is assumed to range over a countably infinite set of fixpoint variables and $\rho$ ranges over R-stable sequents, as said before. The conventions regarding sums $\sum_i$ in the context of forests are also assumed for finitary forests. We stress that this is an all-inductive definition, and that w. r. t. $\text{PIPL}$, the same finite summation mechanism is added as for $\text{PIPL}^\infty_{\text{co}}$, but that the coinductive generation of stable expressions is replaced by formal fixed points whose binding and bound/free variables are associated with R-stable sequents $\rho$ whose proof theory is our main aim.

Below are some immediate adaptations of definitions in our previous paper [ESMP19]. However, they are presented in the new uniform notation. Moreover, the notion of guardedness only arises with the now wider formulation of finitary forests that allows fixed-point formation for any finitary forest of the category of stable expression.

For a finitary forest $T$, let $\text{FPV}(T)$ denote the set of freely occurring typed fixed-point variables in $T$, which can be described by structural recursion:

\[
\begin{align*}
\text{FPV}(f(T_i)) & = \text{FPV}(\sum_i T_i) = \bigcup_i \text{FPV}(T_i) & \text{FPV}(X^\rho) & = \{X^\rho\} \\
\text{FPV}(\text{gfp} X^\rho T) & = \text{FPV}(T) \setminus \{X^\rho \mid \rho \text{ R-stable sequent and } \rho \leq \rho'\}
\end{align*}
\]
Notice the non-standard definition that considers $X^\rho$ also bound by $\text{gfp} X^\rho$, as long as $\rho \leq \rho'$. This special view on binding necessitates to study the following restriction on finitary forests: A finitary forest is said well-bound if, for any of its subterms $\text{gfp} X^\rho.T$ and any free occurrence of $X^\rho$ in $T$, $\rho \leq \rho'$.

**Definition 10 (Interpretation of finitary forests as forests)** For a finitary forest $T$, the interpretation $[T]$ is a forest given by structural recursion on $T$:

\[
\begin{align*}
[f(T_1, \ldots, T_k)] &= f([T_1], \ldots, [T_k]) \\
[T_1 + \ldots + T_k] &= [T_1] + \ldots + [T_k] \\
[X^\rho] &= S(\rho) \\
[\text{gfp} X^\rho.T] &= [T]
\end{align*}
\]

This definition may look too simple to handle the interpretation of bound fixed-point variables adequately, and in our previous paper [EMP16, Lemma 52] we called an analogous definition “simplified semantics” to stress that point. However, as in that previous paper, we can study those finitary forests for which the definition is “good enough” for our purposes of capturing solution spaces:

**Definition 11 (Proper finitary forest)** A finitary forest $T$ is proper if for any of its subterms $T'$ of the form $\text{gfp} X^\rho.T''$, it holds that $[T'] = S(\rho)$.

To any free occurrence of an $X^\rho$ in $T$ is associated a depth: for this, we count the function symbols on the path from the occurrence to the root and notably do not count the binding operation of fixed-point variables and the sum operations. So, $X^\rho$ only has one occurrence of depth 0, likewise for $Y^\rho$ in $\text{gfp} X^\rho.Y^\rho$.

**Definition 12 (Guarded finitary forest)** A finitary forest $T$ is guarded if for any of its subterms $T'$ of the form $\text{gfp} X^\rho.T''$, it holds that every free occurrence of a fixed-point variable $X^\rho$ that is bound by this fixed-point constructor has a depth of at least 1 in $T''$.

**Definition 13 (Finitary solution spaces for PIPL)** Let $\Xi := \overline{X_1, \ldots, X_n}$ be a vector of $m \geq 0$ declarations $(X_i : \rho_i)$ where no fixed-point variable name and no sequent occurs twice. The specification of the finitary forest $F(\sigma; \Xi)$ is as follows. If for some $1 \leq i \leq m$, $\rho_i =: (\Gamma_i \vdash R_i)$ $\leq \sigma$ (i.e., $\sigma = \Gamma \vdash R_i$ and $\Gamma_i \leq \Gamma$), then $F(\sigma; \Xi) = X^\rho_i$, where $i$ is taken to be the biggest such index (notice that the produced $X_i$ will not necessarily appear with the $\rho_i$ associated to it in $\Xi$). Otherwise, $F(\sigma; \Xi)$ is as follows. The only two cases where the fixed-point constructor is used are displayed in Fig. 6. The other cases are specified as the respective cases of the definition for the solution function $S$ (Def. 6), where the co-recursive calls of $S$ are replaced by recursive calls of $F$. In all those other clauses, there is no change in parameter $\Xi$. In the last (resp. penultimate) clause displayed in Fig. 6, if $\Xi = \emptyset$, then $\Xi' = \Xi$; if $\Xi = (\Xi', X : \rho)$ with $\rho = \Gamma' \vdash R$, then $\Xi' = (\Xi', X : \rho')$, with $\rho' = \Gamma', x : N \vdash R$ (resp. $\Gamma', z : a^+ \vdash R$). (The complete definition is found in Appendix A.4.)

$F(\sigma)$ denotes $F(\sigma; \Xi)$ with empty $\Xi$. Analogously to the proof for the similar result for implicative logic [EMP16, Lemma 52], one can show that $F(\sigma)$ is well-defined (the above recursive definition terminates)—some details are given in Appendix A.5.

**Theorem 14 (Equivalence of representations for PIPL)** Let $\sigma$ be a logical sequent and $\Xi$ as in Def. 13 so that $F(\sigma; \Xi)$ exists (in particular, this holds for empty $\Xi$).
Figure 7: EF\(_P\) and NEF\(_P\) predicates

\[
\begin{array}{c|c|c|c}
\text{P(}\rho\text{)} & \forall i, \text{EF}_P(T_i) & \text{EF}_P(T_j) & \text{for some } j \\
\hline
\text{EF}_P(\text{X}^\rho) & \text{EF}_P(f^*(T_i)) & \text{EF}_P(\sum_i T_i) & \\
\hline
\neg P(\rho) & \text{NEF}_P(T_j) & \text{for some } j & \forall i, \text{NEF}_P(T_i) \\
\text{NEF}_P(\text{X}^\rho) & \text{NEF}_P(f^*(T_i)) & \text{NEF}_P(\sum_i T_i) & \\
\end{array}
\]

1. \(F(\sigma; \Xi)\) is guarded.
2. For any \(X : \rho \in \Xi\), if \(X^\rho \) occurs in \(F(\sigma; \Xi)\), then \(\rho \leq \rho'\); hence, \(F(\sigma)\) is well-bound.
3. \(F(\sigma; \Xi)\) is proper.
4. \([F(\sigma; \Xi)] = S(\sigma)\); hence the coinductive and the finitary representations are equivalent.

**Proof** The proof is by structural induction on \(F(\sigma; \Xi)\). Items [1] and [2] are proved independently (the former is an easy induction, the latter needs auxiliary recursive unfolding of \(F(\sigma; \Xi)\) in one subcase of each of the two context-expanding rules). As in the proof of [ESMP19 Thm. 19], item [3] uses item [4] which can be proved independently, but some effort is saved if the two items are proved simultaneously. □

## 5 The inhabitation problem in the polarized system PIPL

We adapt to PIPL our method to decide type emptiness (provability) for the implicational fragment [ESMP19]. The presentation will look very different due to our notational device. Because of the wider notion of finitary forests that does not ensure guardedness through the grammar, some subtle technical refinements are needed in the proofs, which will be most clearly seen in the proof of Prop. [10] which is however only found in Appendix [A.13].

We turn to the representation of solution spaces through finitary forests \(T\) and consider a parameterized predicate \(EF_P(T)\) where the parameter \(P\) is a predicate on logical sequents. \(P = \emptyset\) is already an important case, in fact the main case of interest in the present paper.

In the following, we write \(f^*\) to stand for a function symbol \(f\) or the prefix \(\text{gfp}\text{X}^\rho\), of a finitary forest, the latter being seen a special unary function symbol.

The definition of this (parameterized) predicate \(EF_P\) is inductive and presented in the first line of Fig. [7] although, as in [ESMP19], it is clear that it could equivalently be given by a definition by recursion over the term structure. Thus, the predicate \(EF_P\) is decidable if \(P\) is.

The following can be proven by routine induction on \(T\) (barely more than an application of de Morgan’s laws).

**Lemma 15** For all \(T \in \text{PIPL}^{\text{gfp}}_{\Sigma^P}\), \(\text{NEF}_P(T) \iff EF_P(T)\) does not hold.

**Proposition 16 (Finitary characterization)**

1. If \(P \subseteq \text{exfin}\circ S\) and \(EF_P(T)\) then \(\text{exfin}([T])\).

2. Let \(T \in \text{PIPL}^{\text{gfp}}_{\Sigma^P}\) be well-bound, guarded and proper. If \(\text{NEF}_P(T)\) and for all \(X^\rho \in \text{FPV}(T)\), \(\text{exfin}(S(\rho))\) implies \(P(\rho)\), then \(\text{nofin}([T])\).

3. For any \(T \in \text{PIPL}^{\text{gfp}}_{\Sigma^P}\) well-bound, guarded, proper and closed, \(\text{EF}_0(T) \iff \text{exfin}([T])\).

**Proof** [1] is proved by induction on the predicate \(EF_P\) (or, equivalently, on \(T\)). The base case for fixpoint variables needs the proviso on \(P\), and all other cases are immediate by the induction hypothesis (notice the special case for \(f^*\) that is even more simple).

[2] This needs a special notion of depth of observation for the truthfulness of \(\text{nofin}\) for forests. A more refined statement has to keep track of this observation depth in premise and conclusion,
even taking into account the depth of occurrences of the bound fixed-point variables of $T$. This is presented with details in Appendix A.6.

For $P = \emptyset$ resp. for closed $T$, the extra condition on $P$ in part 1 resp. part 2 is trivially satisfied. Therefore, we only need that $\text{exfin}$ and $\text{nofin}$ are complements, as are $\text{NEF}_P$ and $\text{EF}_P$. □

**Theorem 17 (Deciding the existence of inhabitants in PIPL)** A logical sequent $\sigma$ of PIPL is inhabited iff $\text{exfin}(\mathcal{S}(\sigma))$ iff $\text{EF}_0(\mathcal{F}(\sigma))$. Hence "$\sigma$ is inhabited" is decided by deciding $\text{EF}_0(\mathcal{F}(\sigma))$. In other words, the inhabitation problem for PIPL is decided by the computable predicate $\text{EF}_0 \circ \mathcal{F}$.

**Proof** The first equivalence follows by Prop. 8 and $\text{exfin} = \text{exfinext}$. The second equivalence follows from Prop. 10 using all items of Theorem 13. Computability comes from termination of the recursive specification of $\mathcal{F}(\sigma)$ for all $\sigma$ and the equivalence of the inductively defined $\text{EF}_0$ with a recursive procedure over the term structure of its argument. □

As for Thm. 24 in our previous paper [ESMP19], the theorem opens the way to using Prop. 16 with $P := \text{EF}_0 \circ \mathcal{F}$, a path we followed for the decision of type finiteness for implicational logic.

## 6 Applications to IPL

We give a brief indication of how PIPL and its properties established before can be used to study proof search in intuitionistic logic.

The minimum we can immediately do is to use the reduction of inhabitation problems, obtained in Section 3 to show that decidability of inhabitation in IPL follows from decidability of inhabitation in PIPL. Given $\sigma$ in $LJT_R$, $\sigma$ is inhabited in $LJT_R$ iff $\sigma^*$ is inhabited in PIPL (Cor. 3); iff $\text{exfin}(\mathcal{S}(\sigma^*))$ (Prop. 8 and $\text{exfin} = \text{exfinext}$); iff $\text{EF}_0(\mathcal{F}(\sigma^*))$ (Thm. 17).

The obtained algorithm is thus $\text{EF}_0(\mathcal{F}(\sigma^*))$. It is a two-stage process. First, $\mathcal{F}$ calculates the representation of the full solution space; second the predicate $\text{EF}_0(\cdot)$ does the specific decision. Despite elegance, the obtained procedure, of course, does not give a competitive algorithm to decide full IPL, because its first stage is not optimized. Also, note that if only decision of IPL is sought, variants of LJT like the systems $MJ^\text{Hist}$ [How98] or $Nbu$ [FP19] offer a base for more efficient decision, as, in particular, context-expanding rules are blocked when the formula to be added to the context is already in there (like in total discharge convention). On the other hand, in view of the Curry-Howard isomorphism, one often asks for more information about the set of inhabitants (such as cardinality at most 1, finiteness, enumeration), and our current setting is ready to accommodate such questions. This can be done in two ways: either continuing to work through PIPL, or by translating the key tools, infinitary and finitary representations of the solution space of $LJT_R$, once and for all in terms entirely contained in $LJT_R$. We sketch how the latter can be done.

Let $LJT_{\Sigma}^{\text{co}}$ (resp. $LJT_{\Sigma}^{\text{gfp}}$) be the system whose forests (resp. finitary forests) are obtained by extending the grammar of proof terms of $LJT_R$ with sums (resp. sums, the fixed-point construction and fixed-point variables) for categories $\tau \in \{e, s\}$ (resp. $\tau \in \{e\}$), similarly to the extensions of PIPL to $\text{PIPL}_{\Sigma}^{\text{co}}$ and $\text{PIPL}_{\Sigma}^{\text{gfp}}$ (including the precise reading of the grammars of forests, which in this case requires that infinite branches go infinitely often through inherited $e$-formation rules. For the help of the reader, the grammar obtained for $LJT_{\Sigma}^{\text{gfp}}$ is presented in Appendix A.7.

For a logical sequent $\sigma$ and a vector of declarations $\Xi = X : \rho$ in $LJT_R$, let $\mathcal{S}(\sigma)$ (resp. $\mathcal{F}(\sigma; \Xi)$) be the $LJT_{\Sigma}^{\text{co}}$-forest (resp. $LJT_{\Sigma}^{\text{gfp}}$-finitary forest) resulting from the compositions $|\mathcal{S}(\sigma^*)|$ (resp. $|\mathcal{F}(\sigma^*; \Xi^*)|$), where the forgetful maps are extended from logical $\beta$-proof terms in the obvious way (in particular, $|\sum T_i| = \sum |T_i|$, $|X^e| = X^e$, $|gfp X^s\cdot T| = gfp (X^s\cdot |T|$). As before, $\mathcal{F}(\sigma) = \mathcal{F}(\sigma; \emptyset)$ is a well-defined function. The result of calculating $\mathcal{F}(\sigma; \Xi)$, for $\Xi := X_i : \rho_i$, is the following: If for some $i$, $\rho_i =: (\Gamma_i \vdash R_i) \leq \sigma$ (i.e., $\sigma = \Gamma \vdash R$ and $\Gamma_i \subseteq \Gamma$), then $\mathcal{F}(\sigma; \Xi) = X_i^\sigma$, where $i$ is taken to be the biggest such index; otherwise, the specification is found in Fig. 8 (and, for space reasons, in Appendix A.7), where $\Xi'$ (occurring once in the first clause and twice in the last clause) is given resp. by: if $\Xi = \emptyset$, then $\Xi' = \Xi$; if $\Xi = (\Xi'' , X : \rho)$ with $\rho = \Gamma' \vdash R'$, then $\Xi' = (\Xi'' , X : \rho')$ with $\rho' = \Gamma' , x : A \vdash R'$ (resp. $\Gamma'' , x_1 : A_1 \vdash R'$, resp. $\Gamma'' , x_2 : A_2 \vdash R'$).
We extended “coinductive proof search” to polarized intuitionistic logic, as presented in a slight which other logics (and their properties) can be decomposed. We illustrated this view with Variants of this translation were previously mentioned or sketched [Zei08, LM09], here we give a variation of the system proposed by the first author [San16], obtaining the basic results about this specific approach to proof search, and a result about proof search in polarized logic.

The interpretation \([T]\) of a finitary forest \(T \in \text{LJT}_{\Sigma}^{gfp}\) is defined by structural recursion on \(T\), analogously to \(\text{PIPL}_{\Sigma}^{gfp}\). In particular, \([X^p] = S(\rho)\), and \([gfp \, X^p, e] = [e]\). Then, \([[F(\sigma^*)]] = [F(\sigma)]\) holds (for which one proves more generally \([[F(\sigma^*; \Xi^*)]] = [F(\sigma; \Xi)]\), by an easy induction on \(F(\sigma; \Xi)\)), which, together with the Equivalence Thm. for \(\text{PIPL}\) yields:

**Theorem 18 (Equivalence of representations for \(\text{LJT}_{\Sigma}\))** \( [[F(\sigma)] ] = S(\sigma)\).

### 7 Final remarks

We extended “coinductive proof search” to polarized intuitionistic logic, as presented in a slight variation \(\text{PIPL}\) of the system proposed by the first author [San16], obtaining the basic results about the equivalence of the coinductive and finitary representation of solution spaces, and decidability of the logic through a recursive predicate defined over the finitary syntax. The latter is both a result about this specific approach to proof search, and a result about proof search in polarized logic.

But polarized logic may also be seen as a mere platform, that is, a low-level logic in terms of which other logics (and their properties) can be decomposed. We illustrated this view with \(\text{LJT}_{\Sigma}\), a focused proof system for intuitionistic logic, and the negative translation of \(\text{LJT}_{\Sigma}\) into \(\text{PIPL}\). Variants of this translation were previously mentioned or sketched [Zei08, LM09], here we give a full treatment as a translation between languages of proof terms. By composing the properties of the negative translation with the results about polarized logic, we extracted results about—and tools for—proof search in intuitionistic logic (including notably disjunction). In fact, we worked out a positive translation of \(\lambda\)JQ [DL07] as well, but have no space to show it. This opens the way to the study of inhabitation problems relative to call-by-value \(\lambda\)-terms, and for that, the results obtained here about \(\text{PIPL}\) will be reused.

As we mentioned before, the parameter \(P\) in Prop. 16 could be set differently, to capture other decision problems, e.g., problems involving the concept of solution rather than inhabitant. As a further direction for future work, we plan to extend to \(\text{PIPL}\) such decidability results obtained before [ESMP19, EMP19] in the context of implicational logic.

### References

[DL07] Roy Dyckhoff and Stéphane Lengrand. Call-by-value lambda-calculus and LJQ. *J. Log. Comput.*, 17(6):1109–1134, 2007.

[DP96] Roy Dyckhoff and Luís Pinto. A permutation-free sequent calculus for intuitionistic logic. Technical report, St Andrews University Computer Science Research Report CS/96, August 1996.

[EMP13] José Espírito Santo, Ralph Matthes, and Luís Pinto. A coinductive approach to proof search. In David Baelde and Arnaud Carayol, editors, *Proceedings Workshop on Fixed
A Appendix with some more technical details

A.1 On the characterization of predicates on forests in Section 4.1

Lemma 19 Given a forest $T$, $\text{exfin}(T)$ iff $\text{nofin}(T)$ does not hold.

Proof This is plainly an instance of the generic result in the style of De Morgan’s laws that presents inductive predicates as complements of coinductive predicates, by a dualization operation on the underlying clauses.

The following lemma shows that the predicate $\text{exfin}$ corresponds to the intended meaning in terms of the finite extension. Additionally, the lemma shows that the negation of $\text{exfin}$ holds exactly for the forests which have no (finite) members.
Lemma 20 (Coinductive characterization) Given a forest $T$. Then, \text{exfin}(T)$ is non-empty, i.e., \text{exfin} = \text{exfinext}$ as sets of forests.

**Proof** Follows immediately from the fact:

\[
\text{exfin}(T) \text{ iff } \text{mem}(T_0, T) \text{ for some } T_0.
\]

The left to right implication is proved by induction on \text{exfin}. (Recall \text{exfin} is a predicate on forests, but is defined inductively.) The right to left implication can be proved via the equivalent statement

\[
\text{for all } T_0, \text{ mem}(T_0, T) \text{ implies } \text{exfin}(T)
\]

which follows by induction on PIPL proof terms $T_0$. For the case of membership in sums, it is necessary to decompose them (thanks to priority 1) until membership in an expression $f(T_i)$ is reached so that the argument for the first inductive clause of membership applies. \hfill \square

A.2 On well-definedness of infinitary representation in Section [4.1]

This section is dedicated to the proof of Lemma [7].

It remains to check the parity condition. As mentioned in the main text, this comes from the observation that all the “intermediary” corecursive calls to $S(\sigma)$ in the calculation of $S(\Gamma \vdash A)$—which is the only case that applies inherited $e$-formation rules—lower the “weight” of the logical sequent, until a possible further call to some $S(\Gamma' \vdash A')$.

**Definition 21 (weight)** Weight of a formula: $w(\langle \cdot , a^+ \rangle) := 0$, $w(a^-) := 1$, and for composite formulas, add the weights of the components and add the following for the extra symbols: $w(\langle \cdot \rangle, \land) := 0$, $w(\lor) := 1$, $w(\uparrow) := 2$, $w(\triangledown) := 3$. Then $w(N) \geq 1$ and $w(P) \geq 0$.

Weight of context $\Gamma$: the sum of the weights of all the formulas associated with the variables.

Weight of logical sequent: $w(\Gamma \vdash A) := w(\Gamma) + w(A)$, $w(\Gamma \Rightarrow N) := w(\Gamma) + w(N) - 1 \geq 0$. $w(\Gamma \vdash \{P\}) := w(\Gamma) + w(P)$, $w(\Gamma \vdash P \Rightarrow A) := w(\Gamma) + w(P) + w(A) + 1$, $w(\Gamma \vdash R) := w(\Gamma) + w(N) + w(R)$. Then for all $\sigma$, $w(\sigma) \geq 0$.

In preparation of Section [A.3] we even show the following more general statement:

**Lemma 22** Every direct corecursive call in the definition of $S(\sigma)$ to some $S(\sigma')$ for neither $\sigma$ nor $\sigma'$ R-stable sequents lowers the weight of the logical sequent.

**Proof** We have to show the following inequalities:

\[
\begin{align*}
\text{if } & w(\Gamma \vdash C) > w(\Gamma \Rightarrow C) \text{ (the rule introducing \text{dlv}(\cdot) is easy to overlook but not needed for the proof of Lemma [7]): this is why } \cdot \Rightarrow \cdot \text{ to weigh less} \\
& w(\Gamma \\downarrow N \Rightarrow A) > w(\Gamma, x : N \Rightarrow A): w(\downarrow) = 0 \text{ suffices} \\
& w(\Gamma \vdash [\downarrow N]) > w(\Gamma \Rightarrow N): w(\downarrow) = 0 \text{ suffices} \\
& w(\Gamma \vdash \{P_1 \lor P_2\}) > w(\Gamma \vdash \{P_1\}): \text{ trivial since } w(\lor) > 0 \\
& w(\Gamma \Rightarrow a^-) > w(\Gamma \Rightarrow a^-) \text{ is not to be shown (and is wrong) since we hit the class of R-stable sequents} \\
& w(\Gamma \Rightarrow \triangledown P) > w(\Gamma \Rightarrow P): \text{ this works since } \triangledown \text{ weighs more (given that } \cdot \Rightarrow \cdot \text{ weighs less), but } \\
& \text{ this inequality is not needed either} \\
& w(\Gamma \Rightarrow N_1 \land N_2) > w(\Gamma \Rightarrow N_1): \text{ since both logical sequent weights are unfavourably} \\
& \text{ modified, the weight of } \land \text{ has to be so high} \\
& w(\Gamma \Rightarrow \downarrow P \Rightarrow R) > w(\Gamma \Rightarrow P \Rightarrow R): \text{ this is why } \downarrow \text{ has to weigh more (given that } \cdot \Rightarrow \cdot \text{ weighs more} \\
& w(\Gamma \vdash [P \Rightarrow R]) \text{ and } w(\Gamma \Rightarrow [P \Rightarrow R]): \text{ both are trivial since } w(\triangledown) > 0 \\
& w(\Gamma \vdash N_1 \land N_2 \Rightarrow R) > w(\Gamma \Rightarrow N_1 \Rightarrow R): \text{ since } w(N_3 \Rightarrow i) \geq 1 \\
& \text{ this lemma guarantees the parity condition for all } S(\sigma).
\end{align*}
\]
Figure 9: Corecursive equations for definition of decontraction

\[
\begin{align*}
[\Gamma'/\Gamma] f(T_1, \ldots, T_k) &= f([\Gamma'/\Gamma] T_1, \ldots, [\Gamma'/\Gamma] T_k) & \text{for } f \text{ neither } z \text{ nor } \text{coret}(x, \cdot) \\
[\Gamma'/\Gamma] \sum_i T_i &= \sum_i [\Gamma'/\Gamma] T_i \\
[\Gamma'/\Gamma] z &= z & \text{if } z \notin \text{dom}(\Gamma) \\
[\Gamma'/\Gamma] z &= \sum_{z' \in D_z} z' & \text{if } z \in \text{dom}(\Gamma) \\
[\Gamma'/\Gamma] \text{coret}(x, s) &= \text{coret}(x, [\Gamma'/\Gamma] s) & \text{if } x \notin \text{dom}(\Gamma) \\
[\Gamma'/\Gamma] \text{coret}(x, s) &= \sum_{x' \in D_s} \text{coret}(x', [\Gamma'/\Gamma] s) & \text{if } x \in \text{dom}(\Gamma)
\end{align*}
\]

A.3 On forest transformation for inessential extensions in Section 4.1

If \( \rho = (\Gamma \vdash R) \) and \( \rho' = (\Gamma' \vdash R) \), then the result \([\rho'/\rho] T\) of the decontraction operation applied to \( T \) is defined to be \([\Gamma'/\Gamma] T\), with the latter given as follows:

**Definition 23 (Decontraction)** Let \( \Gamma \leq \Gamma' \). For a forest \( T \) of \( \PiPL^\infty \), the forest \([\Gamma'/\Gamma] T\) of \( \PiPL^\infty \) is defined by corecursion in Fig. 9 where, for \( w \in \text{dom}(\Gamma) \), \( D_w := \{w\} \cup \{w' : (w': \Gamma(w')) \in (\Gamma' \setminus \Gamma)\} \). In other words, the occurrences of variables (in the syntactic way they are introduced in the forests) are duplicated for all other variables of the same type that \( \Gamma' \) has in addition.

**Lemma 24 (Solution spaces and decontraction)** Let \( \rho \leq \rho' \). Then \( S(\rho') = [\rho'/\rho] S(\rho) \).

**Proof** Analogous to the proof for implicational logic [EMP16]. Obviously, the decontraction operation for forests has to be used to define decontraction operations for all forms of logical sequents (analogously to the R-stable sequents, where only \( \Gamma \) varies). Then, the coinductive proof is done simultaneously for all forms of logical sequents. \( \square \)

A.4 Detailed specification of finitary solution spaces for \( \PiPL \) (Def. 13)

**Definition 25 (Finitary solution spaces for \( \PiPL \))** Let \( \Xi := \overrightarrow{X : \rho} \) be a vector of \( m \geq 0 \) declarations \((X_i : \rho_i)\) where no fixed-point variable name and no sequent occurs twice. The specification of the finitary forest \( \mathcal{F}(\sigma; \Xi) \) is as follows. If for some \( 1 \leq i \leq m \), \( \rho_i = (\Gamma_i \vdash R_i) \leq \sigma \) (i.e., \( \sigma = \Gamma \vdash R \) and \( \Gamma_i \leq \Gamma \)), then \( \mathcal{F}(\sigma; \Xi) = X_i^\sigma \), where \( i \) is taken to be the biggest such index (notice that the produced \( X_i \) will not necessarily appear with the \( \rho_i \) associated to it in \( \Xi \)). Otherwise, \( \mathcal{F}(\sigma; \Xi) \) is as displayed in Fig. 10. The vector of declarations \( \Xi' \) (used in the first resp. third clauses relative to co-terms) is given as follows: in case \( \Xi = 0 \), \( \Xi' = \Xi \); in case \( \Xi = (\Xi'', X : \rho') \), \( \Xi' = (\Xi'', X : \rho') \), with \( \rho' = \Gamma' \vdash R \) (resp. \( \Gamma' \vdash N \vdash R \)) when \( \rho = \Gamma' \vdash R \). Then, \( \mathcal{F}(\sigma) \) denotes \( \mathcal{F}(\sigma; \Xi) \) with empty \( \Xi \).

A.5 On termination of finitary representation in Section 4.2

We were careful in Definition 13 to speak of “specification” when presenting the recursive equations. We then mentioned that the proof of termination of an analogous function for implicational logic [EMP16] Lemma 52] when given the empty vector of fixed-point declarations can be adapted to establish also termination of \( \mathcal{F}(\sigma) \) (i.e., with empty \( \Xi \)). Here, we substantiate this claim.

The difficulty comes from the rich syntax of \( \PiPL \), so that the “true” recursive structure of \( \mathcal{F}(\rho; \Xi) \)—for R-stable sequents that spawn the formal fixed points—gets hidden through intermediary recursive calls with the other forms of logical sequents. However, we will now argue that all those can be seen as plainly auxiliary since they just decrease the “weight” of the problem to be solved.
Lemma 26 Every direct recursive call in the specification of \( F; \Xi \) to some \( F; \Xi' \) for neither \( \sigma \) nor \( \sigma' \) R-stable sequents lowers the weight of the first argument.

Proof This requires to check precisely the same inequations as in the proof of Lemma 22. \( \square \)

The message of the lemma is that the proof search through all the other forms of logical sequents (including the form \( \Gamma \vdash C \)) is by itself terminating. Of course, this was to be expected. Otherwise, we could not have “solved” them by a recursive definition in \( F \) where only R-stable sequents ask to be hypothetically solved through fixed-point variables.

The present argument comes from an analysis that is deeply connected to PIPL, it has nothing to do with an abstract approach of defining (infinitary or finitary) forests. As seen directly in the specification of \( F \), only by cycling finitely through the \( \text{dlv}(\cdot) \) construction is the context \( \Gamma \) extended in the arguments \( \sigma \) to \( F \). And the context of the last fixed-point variable in \( \Xi \) grows in lockstep.

It is trivial to observe that all the formula material of the right-hand sides lies in the same subformula-closed sets (see \( \text{EMP16} \)) as the left-hand sides (in other words, the logical sequents in the recursive calls are taken from the same formula material, and there is no reconstruction whatsoever).

Therefore, the previous proof for the implicational case \( \text{EMP16} \) Lemma 52 can be carried over without substantial changes. What counts are recursive calls with first argument an R-stable sequent for the calculation when the first argument is an R-stable sequent. In the implicational case, these “big” steps were enforced by the grammar for finitary forests (and the logical sequents...
The aim of this section is to prove Prop. 16.2. For this, we need an auxiliary concept with which we can formulate a refinement of that proposition. From the refinement, we eventually get Prop. 16.2.

We give a sequence of approximations to the coinductive predicate $\text{nofin}$ whose union exhausts the predicate. The index $n$ is meant to indicate to which observation depth of $T$ we can guarantee that $\text{nofin}(T)$ holds. For this purpose, we do not take into account the summation operation as giving depth. We present the notion as a simultaneous inductive definition:

\[
\begin{align*}
\text{nofin}_0(T) & \quad \text{nofin}_n(T_j) & \quad \text{nofin}_{n+1}(f(T_i)) & \quad \forall i. \text{nofin}_{n+1}(T_i) & \quad \text{nofin}_{n+1}(\sum T_i)
\end{align*}
\]

A guarantee up to observation depth $0$ does not mean that the root symbol is suitable but the assertion is just void. Going through a function symbol requires extra depth. The child has to be fine up to a depth that is one less. As announced, the summation operation does not provide depth, which is why this simultaneous inductive definition cannot be seen as a definition of $\text{nofin}_n$ by recursion over the index $n$.

By induction on the inductive definition, one can show that $\text{nofin}_n$ is antitone in $n$, i.e., if $\text{nofin}_{n+1}(T)$ then $\text{nofin}_n(T)$.

**Lemma 27** (Closure under decontraction of each $\text{nofin}_n$) Let $\rho \leq \rho'$ and $n \geq 0$. For all forests $T$, $\text{nofin}_n(T)$ implies $\text{nofin}_n(\rho'/\rho T)$.

**Proof** By induction on the inductive definition—we profit from not counting sums as providing depth.

**Lemma 28** (Inductive characterization of absence of members) Given a forest $T$. Then, $\text{nofin}(T)$ iff $\text{nofin}_n(T)$ for all $n$.

**Proof** From left to right, this is by induction on $n$. One decomposes (thanks to priority 1) the sums until one reaches finitely many expressions $f(T_i)_i$, to which the induction hypothesis applies. From right to left, thanks to $\text{nofin}_n$ being antitone, the definition above is more constraining than the expected one that changes the index in the last clause. And already that other definition would allow to go to the left-hand side.

For $T \in \text{PIPL}^{\Sigma}$, we write $\mathcal{A}_n(T)$ for the following assumption: For every free occurrence of some $X^\rho$ in $T$ (those $X^\rho$ are found in $\text{FPV}(T)$) such that $\neg P(\rho)$, there is an $n_0$ with $\text{nofin}_{n_0}(S(\rho))$ and $d+n_0 \geq n$ for $d$ the depth of the occurrence in $T$ as defined earlier, where sums and generations of fixed points do not contribute to depth.

Notice that, trivially $n' \leq n$ and $\mathcal{A}_n(T)$ imply $\mathcal{A}_{n'}(T)$.

**Lemma 29** (Ramification of Proposition 16.2) Let $T \in \text{PIPL}^{\Sigma}$ be well-bound, proper and guarded and such that $\mathcal{NEF}_P(T)$ holds. Then, for all $n \geq 0$, $\mathcal{A}_n(T)$ implies $\text{nofin}_n([T])$.

**Proof** By induction on the predicate $\mathcal{NEF}_P$ (which can also be seen as a proof by induction on finitary forests).

Case $T = X^\rho$. Then $[T] = S(\rho)$. Assume $n \geq 0$ such that $\mathcal{A}_n(T)$. By inversion, $\neg P(\rho)$, hence, since $X^\rho \in \text{FPV}(T)$ at depth $0$ in $T$, this gives $n_0 \geq n$ with $\text{nofin}_{n_0}(S(\rho))$. Since $\text{nofin}_{n_0}$ is antitone in $m$, we also have $\text{nofin}_{n_0}([\rho])$.

Case $T = \text{gfp} X^\rho. T_1$. $\mathcal{NEF}_P(T)$ comes from $\mathcal{NEF}_P(T_1)$. Let $N := [T] = [T_1]$. As $T$ is proper, $N = S(\rho)$. We do the proof by a side induction on $n$. The case $n = 0$ is trivial. So assume $n = n' + 1$ and $\mathcal{A}_n(T)$ and that we already know that $\mathcal{A}_{n'}(T)$ implies $\text{nofin}_{n'}(S(\rho))$. We have to
show \( \text{nofin}_n(S(\rho)) \), i.e., \( \text{nofin}_n([T_i]) \). We use the main induction hypothesis on \( T_1 \) with the same index \( n \). Hence, it suffices to show \( A_n(T_1) \). Consider any free occurrence of some \( Y' \) in \( T_1 \) such that \( \neg P(\rho') \). We have to show that there is an \( n_0 \) with \( \text{nofin}_{n_0}(S(\rho)) \) and \( d + n_0 \geq n \) for \( d \) the depth of the occurrence in \( T_1 \).

First sub-case: the considered occurrence is also a free occurrence in \( T \). Since we disregard fixed-point constructions for depth, \( d \) is also the depth in \( T \). Because of \( A_n(T) \), we get an \( n_0 \) as desired.

Second sub-case: the remaining case is with \( Y = X \) and, since \( T \) is well-bound, \( \rho \leq \rho' \).

As remarked before, \( A_n(T) \) gives us \( A_{n'}(T) \). The side induction hypothesis therefore yields \( \text{nofin}_{n'}(S(\rho)) \). By closure of \( \text{nofin}_n \) under decontraction, we get \( \text{nofin}_{n'}(\rho/\rho'S(\rho)) \), but that latter forest is \( S(\rho') \) by Lemma 24. By guardedness of \( T \), this occurrence of \( X' \) has depth \( d \geq 1 \) in \( T_1 \). Hence, \( d + n' \geq 1 + n' = n \).

Case \( T = f(T_1, \ldots, T_k) \) with a proper function symbol \( f \). Assume \( n \geq 0 \) such that \( A_n(T) \). There is an index \( j \) such that \( \text{NEF}_P(T) \) comes from \( \text{NEF}_P(T_j) \). Assume \( n \geq 0 \) such that \( A_n(T) \). We have to show that \( \text{nofin}_n([T_j]) \). This is trivial for \( n = 0 \). Thus, assume \( n = n' + 1 \). We are heading for \( \text{nofin}_{n'}([T_i]) \). We use the induction hypothesis on \( T_j \) (even with this smaller index \( n' \)). Therefore, we are left to show \( A_n(T_j) \). Consider any free occurrence of some \( X' \) in \( T_j \) such that \( \neg P(\rho) \), of depth \( d \) in \( T_j \). This occurrence is then also a free occurrence in \( T \) of depth \( d + 1 \) in \( T \). From \( A_n(T) \), we get an \( n_0 \) with \( \text{nofin}_{n_0}(S(\rho)) \) and \( d + n_0 \geq n \), hence with \( d + n_0 \geq n' \), hence \( n_0 \) is as required for showing \( A_{n'}(T_j) \).

Case \( T = \sum_i T_i \). \( \text{NEF}_P(T) \) comes from \( \text{NEF}_P(T_i) \) for all \( i \). Assume \( n \geq 0 \) such that \( A_n(T) \). We have to show that \( \text{nofin}_n([T_i]) \). This is trivial for \( n = 0 \). Thus, assume \( n = n' + 1 \) and fix some index \( i \). We have to show \( \text{nofin}_n([T_i]) \). We use the induction hypothesis on \( T_i \) (with the same index \( n \)). Therefore, we are left to show \( A_n(T_i) \). Consider any free occurrence of some \( X' \) in \( T_i \) such that \( \neg P(\rho) \), of depth \( d \) in \( T_i \). This occurrence is then also a free occurrence in \( T \) of depth \( d + 1 \) in \( T \). From \( A_n(T) \), we get an \( n_0 \) with \( \text{nofin}_{n_0}(S(\rho)) \) and \( d + n_0 \geq n \), hence \( n_0 \) is as required for showing \( A_{n_0}(T_i) \). (Of course, it is important that sums do not count for depth in finitary terms if they do not count for the index of the approximations to \( \text{nofin} \). Therefore, this proof case is so simple.)

We return to Prop. 16.2.

**Proof** Let \( T \in \PiPL^\text{gfp}_\Sigma \) be well-bound, proper and guarded, assume \( \text{NEF}_P(T) \) and that for all \( X' \in \text{FPV}(T) \), \( \text{exfin}(S(\rho)) \) implies \( P(\rho) \). We have to show \( \text{nofin}([T]) \). By Lemma 28 it suffices to show \( \text{nofin}_n([T]) \) for all \( n \). Let \( n \geq 0 \). By the just proven refinement, it suffices to show \( A_n(T) \). Consider any free occurrence of some \( X' \) in \( T \) such that \( \neg P(\rho) \), of depth \( d \) in \( T \). By contraposition of the assumption on \( \text{FPV}(T) \) and by the complementarity of \( \text{nofin} \) and \( \text{exfin} \), we have \( \text{nofin}(S(\rho)) \), hence by Lemma 28, \( \text{nofin}_n(S(\rho)) \), and \( d + n \geq n \), as required for \( A_n(T) \). 

A.7 Details on finitary representation for \( LJT_R \) in Section 6

The grammar obtained for \( LJT_R \) as described in the text is:

\[
\begin{align*}
(\text{terms}) & \quad t & ::= & \lambda x^A.t \mid \{t_1, t_2\} \mid e \\
(\text{stable expressions}) & \quad e & ::= & xs \mid \text{in}_A(t) \mid \sum_i e_i \mid X^\rho \mid \text{gfp}X^\rho.e \\
(\text{spines}) & \quad s & ::= & \text{nil} \mid t :: s \mid i :: s \mid \text{abort}^R \mid [x_1^{A_1}, e_1, x_2^{A_2}, e_2] \mid \sum_i s_i
\end{align*}
\]

We now give the omitted cases of the specification of \( \mathcal{F}(\sigma; \Xi) \), i.e., those that have to be added to Fig. 8:

\[
\begin{align*}
\mathcal{F}(\Gamma \implies A_1 \land A_2; \Xi) & = (\mathcal{F}(\Gamma \implies A_1; \Xi), \mathcal{F}(\Gamma \implies A_2; \Xi)) \\
\mathcal{F}(\Gamma \implies R; \Xi) & = (\mathcal{F}(\Gamma \implies R; \Xi)) \\
\mathcal{F}(\Gamma[A \sqcup B] \implies R; \Xi) & = \mathcal{F}(\Gamma \implies A; \Xi) \sqcup \mathcal{F}(\Gamma[B] \implies R; \Xi) \\
\mathcal{F}(\Gamma[A_1 \land A_2] \implies R; \Xi) & = \sum_{i \in \{1, 2\}} (i :: \mathcal{F}(\Gamma[A_i] \implies R; \Xi)) \\
\mathcal{F}(\Gamma[\bot] \implies R; \Xi) & = \text{abort}^R
\end{align*}
\]