Circular Proofs in First-Order Linear Logic with Least and Greatest Fixed Points

Farzaneh Derakhshan
Philosophy Department
Carnegie Mellon University
fderakhs@andrew.cmu.edu

Frank Pfenning
Computer Science
Carnegie Mellon University
fp@cmu.edu

Abstract

Inductive and coinductive structures are everywhere in mathematics and computer science. The induction principle is well known and fully exploited to reason about inductive structures like natural numbers and finite lists. To prove theorems about coinductive structures such as infinite streams and infinite trees we can appeal to bisimulation or the coinduction principle. Pure inductive and coinductive types however are not the only data structures we are interested to reason about. In this paper we present a calculus to prove theorems about mutually defined inductive and coinductive data types. Derivations are carried out in an infinitary sequent calculus for first order intuitionistic multiplicative additive linear logic with fixed points. We enforce a condition on these derivations to ensure their cut elimination property and thus validity. Our calculus is designed to reason about linear properties but we also allow appealing to first order theories such as arithmetic, by adding an adjoint downgrade modality. We show the strength of our calculus by proving several theorems on (mutual) inductive and coinductive data types.

Keywords circular proofs, first order linear logic, fixed points, (co)induction, simultaneous induction and coinduction, recursive definitions, cut elimination

1 Introduction

The induction principle is well known and presented in the literature in many different contexts. Computer scientists use this principle to reason about inductive data types such as natural numbers and finite lists. To show properties of coinductive data types, e.g. streams and infinite trees, a dual principle of coinduction is needed. In the literature bisimulation has been used effectively to prove equality of structures defined as greatest fixed points. To prove properties other than equality for coinductive data types one needs to use the somewhat less familiar coinduction principle[6, 11, 13]. Kozen and Silva established a practical proof principle to produce sound proofs by coinduction [12]. However for data types mutually defined by induction and coinduction these separate principles are insufficient. One recent approach in type theory integrates induction and coinduction by pattern and copattern matching and explicit well-founded induction on ordinals[1], following a number of earlier representations of induction and coinduction in type theory [2]. Here, we pursue a different line of research in *linear logic* with fixed points. In this paper we introduce a sequent calculus to reason about linear predicates defined as nested least and greatest fixed points. Instead of applying induction and coinduction principles directly, we follow the approach of Brotherston et al. [7] to allow circularity in derivations. We use cyclic reasoning in the context of first order intuitionistic multiplicative additive linear logic extended with least and greatest fixed points. To ensure soundness of the proofs we impose a validity condition on our derivations.

Fortier et al. introduced an infinitary sequent calculus for propositional singleton logic with fixed points, where antecedent and succedent consist of exactly one formula [10, 18]. Adding circularity comes with the cost of losing the cut elimination property. To recover this property they introduced a guard condition that ensures soundness of possibly infinite derivations. They provide a cut elimination algorithm and show its productivity on derivations satisfying their guard condition. Fortier and Santocanale's result has been generalized by Baelde et al. [3, 9] for propositional MALL with fixed points.

In this paper, we extend Fortier et al.'s results to *first order* multiplicative additive linear logic with fixed points. Our notion of validity is adapted from its counterpart in their system. We introduce a similar cut elimination algorithm and prove its local termination on valid derivations with a dual approach. It is worth mentioning that our calculus is essentially different from the finitary one introduced by Baelde for the first order MALL with fixed points [4] since we allow for circularity.

We will show with several examples that our calculus is strong enough to prove many (mutual) inductive and coinductive theorems. To make the examples concise we may use pattern matching for defining inductive predicates [7, 17]. Our underlying system is designed to reason about linear structures. However, some properties of linear structures rely on first order non-linear theories such as theory of arithmetic or order theory. To be able to prove these properties as well we extend our calculus by mixing linear and structural formulas. Our approach is to use a restricted version of the adjoint logic presented by Pfenning et al. [5, 14]. The restriction is that only linear formulas can depend on non-linear ones and not vice versa. Thus we only add the *downgrade* operator (⇃) that embeds a nonlinear formula into a linear one to our language. In this way we can isolate the reasoning about nonlinear properties to the pure structural part, and use any sound nonlinear theory in a modular way.

In summary, the main contributions of this paper is to introduce an infinitary sequent calculus for first order multiplicative additive linear logic with (mutual) least and greatest fixed points. We provide a validity condition on derivations that ensures the cut elimination property. Our calculus is a tool to reason about a rich signature of mutually defined inductive and coinductive predicates and also allows using nonlinear first order theories. We show its strength by providing several examples including properties defined as nested least and greatest fixed points.

2 First order intuitionistic linear logic with fixed points

The syntax of formulas in the first order intuitionistic multiplicative additive linear logic with fixed points ($FIMALL^{\infty}_{\mu,\nu}$) follows the grammar

$$A \quad ::= \quad 1 \mid 0 \mid \top \mid A \otimes A \mid A \multimap A \mid A \oplus A \mid A \,\&\, A \mid \exists x.\, A(x) \mid \forall x.\, A(x) \mid s = t \mid T(\bar{t})$$

where s, t stand for terms[1] and x, y for term variables. $T(\bar{t})$ is a predicate variable defined using least and greatest fixed points in a *signature* Σ.

$$\Sigma ::= \cdot \mid \Sigma, T(\bar{x}) =^i_\mu A \mid \Sigma, T(\bar{x}) =^i_\nu A$$

The subscript a of a fixed point $T(\bar{x}) =^i_a$ determines its polarity. If $a = \mu$, then predicate $T(\bar{x})$ is of positive polarity and if $a = \nu$ it is of negative polarity. We represent inductively defined predicates (e.g., the property of being a natural number) as fixed points with positive polarity and coinductively defined predicates (e.g., the lexicographic order on streams) as fixed points with negative polarity. Here we restrict Σ to the definitions in which each predicate occurs only in positive (variant) or negative(contravariant) positions, i.e. we do not allow mixed positions[15, 16].

The superscript $i \in \mathbb{N}$ is the relative priority of $T(\bar{x})$ in the signature Σ with the condition that if $T_1(\bar{x}) =^i_a A, T_2(\bar{x}) =^i_b B \in \Sigma$, then $a = b$. Similar to prior work ([8, 10]) we use priority on predicates to define the validity condition on infinite derivations.

Example 2.1. Let signature Σ_1 be

$$\begin{array}{lll}
\mathtt{Nat}(x) & =^1_\mu & (\exists y.(x = sy) \otimes \mathtt{Nat}(y)) \oplus ((x = z) \otimes 1) \\
\mathtt{Even}(x) & =^2_\mu & (\exists y.(x = sy) \otimes \mathtt{Odd}(y)) \oplus ((x = z) \otimes 1) \\
\mathtt{Odd}(x) & =^2_\mu & (\exists y.(x = sy) \otimes \mathtt{Even}(y))
\end{array}$$

where positive predicates Nat, Even, and Odd refer to the properties of being natural, even, and odd numbers respectively. We interpret it as Nat having a higher priority relative to Even and Odd.

$$\frac{}{A \vdash A}\ \mathtt{fwd} \qquad \frac{\Gamma \vdash A \quad \Gamma', A \vdash C}{\Gamma, \Gamma' \vdash C}\ \mathtt{Cut}$$

$$\frac{}{\cdot \vdash 1}\ 1R \qquad \frac{\Gamma \vdash C}{\Gamma, 1 \vdash C}\ 1L$$

$$\frac{\Gamma \vdash A_1 \quad \Gamma' \vdash A_2}{\Gamma, \Gamma' \vdash A_1 \otimes A_2}\ \otimes R \qquad \frac{\Gamma, A_1, A_2 \vdash B}{\Gamma, A_1 \otimes A_2 \vdash B}\ \otimes L$$

$$\frac{\Gamma, A_1 \vdash A_2}{\Gamma \vdash A_1 \multimap A_2}\ \multimap R \qquad \frac{\Gamma \vdash A_1 \quad \Gamma', A_2 \vdash B}{\Gamma, \Gamma', A_1 \multimap A_2 \vdash B}\ \multimap L$$

$$\frac{\Gamma \vdash A_k \quad k \in I}{\Gamma \vdash \oplus\{l_i : A_i\}_{i \in I}}\ \oplus R \qquad \frac{\Gamma, A_i \vdash B \quad \forall i \in I}{\Gamma, \oplus\{l_i : A_i\}_{i \in I} \vdash B}\ \oplus L$$

$$\frac{\Gamma \vdash A_i \quad \forall i \in I}{\Gamma \vdash \&\{l_i : A_i\}_{i \in I}}\ \& R \qquad \frac{\Gamma, A_k \vdash B \quad k \in I}{\Gamma, \&\{l_i : A_i\}_{i \in I} \vdash B}\ \& L$$

$$\frac{\Gamma \vdash P(t)}{\Gamma \vdash \exists x.P(x)}\ \exists R \qquad \frac{\Gamma, P(x) \vdash B}{\Gamma, \exists x.P(x) \vdash B}\ \exists L$$

$$\frac{\Gamma \vdash P(x)}{\Gamma \vdash \forall x.P(x)}\ \forall R \qquad \frac{\Gamma, P(t) \vdash B}{\Gamma, \forall x.P(x) \vdash B}\ \forall L$$

$$\frac{\Gamma \vdash [\bar{t}/\bar{x}]A \quad T(\bar{x}) =_\mu A}{\Gamma \vdash T(\bar{t})}\ \mu_T R \qquad \frac{\Gamma, [\bar{t}/\bar{x}]A \vdash B \quad T(\bar{x}) =_\mu A}{\Gamma, T(\bar{t}) \vdash B}\ \mu_T L$$

$$\frac{\Gamma \vdash [\bar{t}/\bar{x}]A \quad T(\bar{x}) =_\nu A}{\Gamma \vdash T(\bar{t})}\ \nu_T R \qquad \frac{\Gamma, [\bar{t}/\bar{x}]A \vdash B \quad T(\bar{x}) =_\nu A}{\Gamma, T(\bar{t}) \vdash B}\ \nu_T L$$

$$\frac{}{\cdot \vdash s = s}\ = R \qquad \frac{\Gamma[\theta] \vdash B[\theta] \quad \theta \in \mathtt{mgu}(t, s)}{\Gamma, s = t \vdash B}\ = L$$

Figure 1. Infinitary calculus for first order linear logic with fixed points

A judgment in $FIMALL^{\infty}_{\mu,\nu}$ is of the form $\Gamma \vdash_\Sigma A$ where Γ is a set of formulas and Σ is the signature. We omit Σ from the judgments, since it never changes throughout a proof. The infinitary sequent calculus for this logic is given in Figure 1, in which we generalize $\oplus$ and $\&$ to be n-ary connectives $\oplus\{l_j : A_j\}_{j \in I}$ and $\&\{l_i : A_i\}_{j \in I}$. The binary disjunction and conjunction are defined as $A \oplus B = \oplus\{\pi_1 : A, \pi_2 : B\}$ and $A \& B = \&\{\pi_1 : A, \pi_2 : B\}$. Constants 0 and $\top$ defined as the nullary version of these connectives: $0 = \oplus\{\}$ and $\top = \&\{\}$.

Example 2.2. Consider signature Σ_1 and predicates Even and Odd defined in Example 2.1. The following derivation is

[1]We do not specify a grammar for terms; all terms are of the only type U.

a finite proof of one (sz) being an odd number.

$$\dfrac{\dfrac{}{\cdot \vdash \mathsf{sz} = \mathsf{sz}}\ {=}R \qquad \dfrac{\dfrac{\dfrac{\dfrac{}{\cdot \vdash \mathsf{z} = \mathsf{z}}\ {=}R \quad \dfrac{}{\cdot \vdash 1}\ 1R}{\cdot \vdash (\mathsf{z} = \mathsf{z}) \otimes 1}\ \otimes R}{\cdot \vdash (\exists y.(x = \mathsf{s}y) \otimes \mathsf{Odd}(y)) \oplus ((\mathsf{z} = \mathsf{z}) \otimes 1)}\ \oplus R}{\cdot \vdash \mathsf{Even}(\mathsf{z})}\ \mu_{\mathsf{Even}}R}{\dfrac{\dfrac{\cdot \vdash (\mathsf{sz} = \mathsf{sz}) \otimes \mathsf{Even}(\mathsf{z})}{\cdot \vdash \exists y.(\mathsf{sz} = \mathsf{s}y) \otimes \mathsf{Even}(y)}\ \exists R}{\cdot \vdash \mathsf{Odd}(\mathsf{sz})}\ \mu_{\mathsf{Odd}}R}\ \otimes R$$

The calculus in Figure 1 is infinitary, meaning that it allows building infinite derivations as well. The infinite derivations we are interested in, are those we can represent in a finite way. A *circular derivation* is the finite representation of an infinite one in which we can identify each open subgoal with an identical interior judgement. In the first order context we may need to use a substitution rule right before a circular edge to make the subgoal and interior judgment exactly identical [7]:

$$\dfrac{\Gamma \vdash B}{\Gamma[\theta] \vdash B[\theta]}\ \mathtt{subst}_\theta$$

We can transform a circular derivation to its underlying infinite derivation in a productive way by deleting the $\mathtt{subst}_\theta$ rule and the circular edge. We need to instantiate the derivation to which the circular edge pointed with substitution θ. This instantiation exists and does not change the structure of derivation by Lemma A.1 in the Appendix.

Example 2.3. Consider Signature Σ_1 and predicates $\mathtt{Nat}$, $\mathtt{Even}$, and $\mathtt{Odd}$ defined in Example 2.1. Figure 2 represents a circular derivation for $\mathtt{Even}(x) \vdash \mathtt{Odd}(\mathsf{s}\,x)$. Π is the finite derivation given in Example 2.2.

We can interpret the proof in Example 2.3 as an inductive proof where its circular edge corresponds to applying the induction hypothesis. In the next two examples we represent two coinductive proofs in our circular calculus. Both examples are adapted from Kozen and Silva [12].

Example 2.4. Define Σ_2 to consist of a single predicate with negative polarity $\sim (x, y) =^1_\nu (\mathsf{hd}\, x = \mathsf{hd}\, y) \,\&\, \sim (\mathsf{tl}\, x, \mathsf{tl}\, y)$. Predicate $\sim (x, y)$ can be read as a bisimulation between streams x and y. We present a circular derivation for $\sim$ being symmetric in Figure 3.

Example 2.5. We can reason about the properties of stream operations in our calculus as well. Consider three operations merge, split_1 and split_2. Operation merge receives two streams and merge them into a single stream by alternatively outputting an element of each. Operations split_1 and split_2 receive a stream x as an input and return the odd and even elements of it, respectively. We define these operations as negative predicates in our language. Define signature Σ_3 as

$$\begin{array}{lcl}
\mathtt{Merge}(x, y, z) & =^1_\nu & (\mathsf{hd}\, z = \mathsf{hd}\, x \,\&\, \mathtt{Merge}\,(y, \mathsf{tl}\, x, \mathsf{tl}\, z)) \\
\mathtt{Split}_1(x, y) & =^1_\nu & (\mathsf{hd}\, y = \mathsf{hd}\, x \,\&\, \mathtt{Split}_2(\mathsf{tl}\, x, \mathsf{tl}\, y)) \\
\mathtt{Split}_2(x, y) & =^1_\nu & (1 \,\&\, \mathtt{Split}_1(\mathsf{tl}\, x, y))
\end{array}$$

The derivation given in Figure 4 shows that operations merge and split_i are inverses: Split a stream x into two streams y_1 and y_2 using split_1 and split_2, respectively, then merge y_1 and y_2. The result is x.

3 Pattern Matching

It may not be feasible to present a large piece of derivation fully in the calculus of Figure 1. For the sake of brevity, we may represent predicates of positive polarity in the signature using pattern matching and build equivalent derivations based on that signature [7, 17]. In all the examples we use pattern matching for it should be clear how to transform the signature and derivations into our main logical system.

Example 3.1. Redefine predicates $\mathtt{Even}$, $\mathtt{Odd}$, and $\mathtt{Nat}$ in Example 2.1 by pattern matching in Signature Σ'_1 as:

$$\begin{array}{lcl@{\qquad}lcl}
\mathtt{Nat}(\mathsf{z}) & =^1_\mu & 1 & \mathtt{Nat}(sy) & =^1_\mu & \mathtt{Nat}(y) \\
\mathtt{Odd}(\mathsf{z}) & =^1_\mu & 0 & \mathtt{Odd}(sy) & =^1_\mu & \mathtt{Even}(y) \\
\mathtt{Even}(\mathsf{z}) & =^1_\mu & 1 & \mathtt{Even}(sy) & =^1_\mu & \mathtt{Odd}(y)
\end{array}$$

The circular derivation in Example 2.3 can be simplified in the following way:

$$[1]\quad \dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{\cdot \vdash 1}\ 1R}{\cdot \vdash \mathtt{Even}(\mathsf{z})}\ \mu R}{\cdot \vdash \mathtt{Odd}(\mathsf{s}\,\mathsf{z})}\ \mu R}{1 \vdash \mathtt{Odd}(\mathsf{s}\,\mathsf{z})}\ 1L}{\dagger\,\mathtt{Even}(\mathsf{z}) \vdash \mathtt{Odd}(\mathsf{s}\,\mathsf{z})}\ \mu L
\qquad
[2]\quad \dfrac{\dfrac{\overset{\star}{\mathtt{Odd}(x) \vdash \mathtt{Even}(s\,x)}}{\mathtt{Odd}(x) \vdash \mathtt{Odd}(s\,s\,x)}\ \mu R}{\dagger\,\mathtt{Even}(s\,x) \vdash \mathtt{Odd}(\mathsf{s}\,\mathsf{s}\,x)}\ \mu L$$

$$[3]\quad \dfrac{\dfrac{}{0 \vdash \mathtt{Odd}(\mathsf{z})}\ 0L}{\star\,\mathtt{Odd}(z) \vdash \mathtt{Even}(\mathsf{s}\,\mathsf{z})}\ \mu L
\qquad
[4]\quad \dfrac{\dfrac{\overset{\dagger}{\mathtt{Even}(x) \vdash \mathtt{Odd}(s\,x)}}{\mathtt{Even}(x) \vdash \mathtt{Even}(s\,s\,x)}\ \mu R}{\star\,\mathtt{Odd}(s\,x) \vdash \mathtt{Even}(\mathsf{s}\,\mathsf{s}\,x)}\ \mu L$$

By the definition of signature Σ'_1, the pattern of x in $\mathtt{Odd}(x)$ is either of the form $\mathsf{s}\,y$ or z. At the subgoal marked with $\star$ in subderivation 2, we form a branch similar to the $\oplus L$ rule to cover all possible patterns of x; we continue with subderivations 3 and 4. With the same reasoning at the subgoal marked with $\dagger$ in the subderivation 4 we form a branch with subderivations 1 and 2.

A major contribution of this paper is to give a criterion for validity of theorems proved by simultaneous induction and coinduction. In the next example we see an interplay between positive and negative fixed points in the derivation. Define predicate $\mathtt{run}(x, t)$ to represent computation of a stream processor, where x is the list of operations we want to compute. Operations in x can be either a *skip* or a *put*$\langle x \rangle$. Operation *skip* simply skips one step and does not contribute to the output stream t. Operation *put*$\langle x \rangle$ puts element z as the head of the output stream t and inserts a new list of operations x to the original list of operations. After computing *skip* the length of remaining operations in x goes down by one. So we can define $\mathtt{run}(\mathit{skip}; x, t)$ inductively as a positive predicate. *put*$\langle x \rangle$ increases the length of the operations, but produces an element of the output stream. So

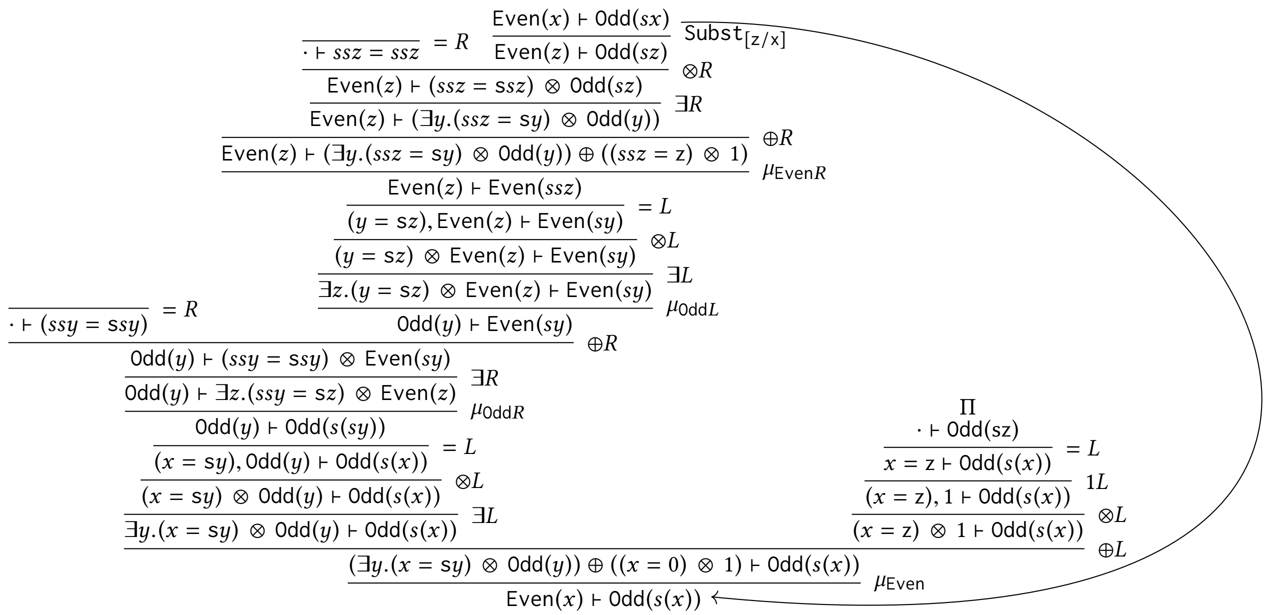


Figure 2. Successor of every even number is odd.

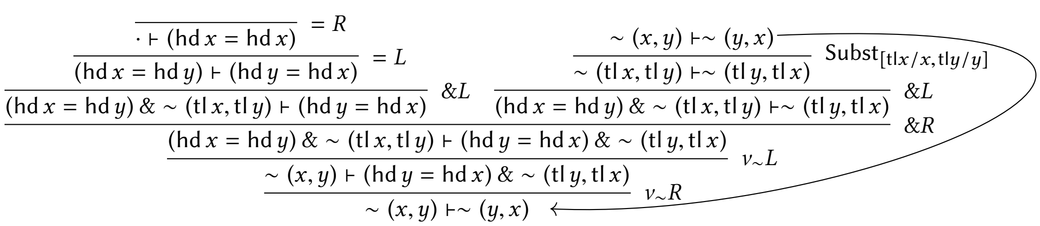


Figure 3. Relation $\sim$ defined on streams is symmetric.

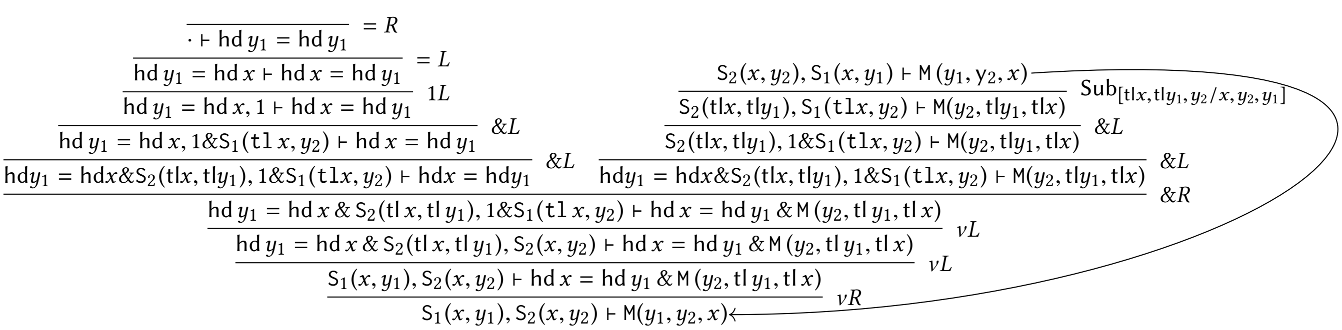


Figure 4. Operations $\mathtt{Merge}(\mathsf{M})$ and $\mathtt{Split_i}(\mathsf{S_i})$ are inverses.

$\mathsf{run}(put\langle x\rangle; y, t)$ needs to be defined as a negative predicate rather than a positive one.

Example 3.2. Define the signature Σ_4 to be

$$\begin{array}{lll}
\mathsf{run}(\cdot\,, t) & =^1_\mu & 1 \\
\mathsf{run}(skip; x, t) & =^1_\mu & \mathsf{run}(x, t) \\
\mathsf{run}(put\langle x\rangle; y, t) & =^1_\mu & \mathsf{nrun}\,(x, y, t) \\
\mathsf{nrun}(x, y, t) & =^2_\nu & \mathsf{hd}\, t = \mathsf{z} \,\&\, \mathsf{run}(x; y, \mathtt{tl}\, t)
\end{array}$$

The equivalent signature without pattern matching is

$$\begin{array}{lll} \mathsf{run}(x,t) & =^1_\mu & \oplus\{\mathsf{e}: x = \cdot \otimes 1, \\ & & \mathsf{s}: \exists x'.x = skip; x' \otimes \mathsf{run}(x',t), \\ & & \mathsf{p}: \exists x'.\exists y.x = put(x'); y \otimes \mathsf{nrun}(x',y,t)\} \\ \mathsf{nrun}(x,y,t) & =^2_\nu & \mathsf{hd}\, t = \mathsf{z} \,\&\, \mathsf{run}(x;y,\mathsf{tl}\, t) \end{array}$$

Here we define $\mathsf{run}(put\langle x\rangle; y, t)$ in two steps to follow the rules of definition by pattern matching. We can abbreviate this definition to one step as:

$$\mathsf{run}(put\langle x\rangle; y, t) =^2_\nu \mathsf{hd}\, t = \mathsf{z} \,\&\, \mathsf{run}(x; y, \mathsf{tl}\, t)$$

We want to prove that a run of any list of operations x produces a (possibly infinite) list of elements z.

$$\begin{array}{lll} \mathsf{zlist}(t) & =^1_\mu & 1 \oplus \mathsf{ztream}(t) \\ \mathsf{ztream}(t) & =^2_\nu & \mathsf{hd}\, t = \mathsf{z} \,\&\, \mathsf{zlist}\,(\mathsf{tl}\, t) \end{array}$$

We give circular derivations for both (†) $\mathsf{run}(x,t) \vdash \mathsf{zlist}(t)$ and (★) $\mathsf{nrun}(x,y,t) \vdash \mathsf{ztream}(t)$ in Figure 5 to show the interplay between coinductive and inductive predicates.

4 A Validity Condition

Adding fixed point rules to the calculus comes with the price of losing the cut elimination property. Infinite derivations in this calculus do not necessarily enjoy the cut elimination property and thus are called *pre-proofs* instead of *proofs*. We introduce a validity condition on derivations such that the cut elimination property holds for the derivations satisfying it. Our condition is adapted from the Guard condition introduced by Fortier and Santocanale [10] for singleton logic. We annotate formulas with position variables $\mathbf{x}, \mathbf{y}, \mathbf{z}$ and track their generations α, β to capture evolution of a formula in a derivation. With this annotation we can keep track of behaviour of any particular formula throughout the whole derivation. Our validity condition requires that at least one formula in every infinite branch behaves in a way that justifies validity of that branch.

A basic judgment in the annotated calculus is of the form $\Delta \vdash_\Omega \mathbf{z}^\beta : C$ where $\Delta = \cdot \mid \mathbf{x}^\alpha : A, \Delta$. The set Ω keeps the relation between different generation of position variables in a derivation. We will use the set Ω to define our validity condition. Figure 6 shows the calculus annotated with position variable generations and their relations. A new generation of a position variable is introduced when a fixed point rule applies on it. The relation of a new generation to its priors is determined by the role of the rule that introduces it in (co)induction. μL rule breaks down an inductive antecedent and νR produces a coinductive information. They both take a step toward termination/productivity of the proof: we put the new generation of the position variable they introduce to be less than the prior ones in the given priority. Their counterpart rules μR and νL, however, do not contribute to termination/productivity. They break the relation between the new generation and its prior ones for the given priority.

In the *Cut* rule we introduce a fresh position variable of generation zero, $\mathbf{w}^0$. Since it refers to appearance of a new formula, we put it to be incomparable to other position variables. We do not consider $\mathbf{w}^0$ as a continuation of $\mathbf{z}^\beta$ in the rule $\otimes R$ either; we need to restrict left branching on succedent position variables to prove Theorem 5.2.[2] The fresh position variable $\mathbf{w}^0$ introduced in $\multimap R$ (resp. $\multimap L$) rule switches its polarity from right to left (resp. left to right) so it cannot be equal to $\mathbf{z}^\beta$ (resp. $\mathbf{y}^\alpha$).[3] As none of the above reasons hold for $\mathbf{w}^0$ in $\otimes L$, we keep its relation with $\mathbf{y}^\alpha$ in Ω.

Definition 4.1. For a given signature Σ, define *snapshot* of an annotated position variable $\mathbf{x}^\alpha$ as a list $\mathsf{snap}(\mathbf{x}^\alpha) = [\mathbf{x}^\alpha_i]_{i<n}$, where n is the maximum priority in Σ.

The list $\mathsf{snap}(\mathbf{x}^\alpha)$ stores the information of the fixed point unfolding rules applied on previous generations of position variable $\mathbf{x}^\alpha : A$ in a derivation.

Example 4.2. For signature Σ_1 defined in Example 2.1:

$$\begin{array}{lll} \mathsf{Nat}(x) & =^1_\mu & (\exists y.(x = \mathsf{s}y) \otimes \mathsf{Nat}(y)) \oplus ((x = \mathsf{z}) \otimes 1) \\ \mathsf{Even}(x) & =^2_\mu & (\exists y.(x = \mathsf{s}y) \otimes \mathsf{Odd}(y)) \oplus ((x = \mathsf{z}) \otimes 1) \\ \mathsf{Odd}(x) & =^2_\mu & (\exists y.(x = \mathsf{s}y) \otimes \mathsf{Even}(y)) \end{array}$$

and position variables $\mathbf{x}^\alpha$ and $\mathbf{z}^\beta$ in the judgment $\mathbf{x}^\alpha : \mathsf{Odd}(x) \vdash \mathbf{z}^\beta : \mathsf{Even}(\mathsf{s}x)$ we have $\mathsf{snap}(\mathbf{x}^\alpha) = [\mathbf{x}^\alpha_i]_{i<2} = [\mathbf{x}^\alpha_1, \mathbf{x}^\alpha_2]$ and $\mathsf{snap}(\mathbf{z}^\beta) = [\mathbf{z}^\beta_i]_{i<2} = [\mathbf{z}^\beta_1, \mathbf{z}^\beta_2]$.

Having the relation between annotated position variables in Ω, we can define a partial order on snapshots of annotated position variables. We write

$$\mathsf{snap}(\mathbf{x}^\alpha) = [\mathbf{x}^\alpha_1 \cdots \mathbf{x}^\alpha_n] <_\Omega [\mathbf{z}^\beta_1 \cdots \mathbf{z}^\beta_n] = \mathsf{snap}(\mathbf{z}^\beta)$$

if the list $[\mathbf{x}^\alpha_1 \cdots \mathbf{x}^\alpha_n]$ is less than $[\mathbf{z}^\beta_1 \cdots \mathbf{z}^\beta_n]$ by the lexicographic order defined by the transitive closure of the relations in Ω.

Example 4.3. Let $\Omega = \{\mathbf{x}^\alpha_1 = \mathbf{z}^\beta_1, \mathbf{x}^\alpha_2 < \mathbf{z}^\gamma_2, \mathbf{z}^\gamma_2 < \mathbf{z}^\beta_2\}$. For $\mathsf{snap}(\mathbf{x}^\alpha)$ and $\mathsf{snap}(\mathbf{z}^\beta)$ defined over signature Σ_1 in Example 4.2, we have $\mathsf{snap}(\mathbf{x}^\alpha) = [\mathbf{x}^\alpha_1, \mathbf{x}^\alpha_2] <_\Omega [\mathbf{z}^\beta_1, \mathbf{z}^\beta_2] = \mathsf{snap}(\mathbf{z}^\beta)$.

We adapt the definitions of left μ-trace and right ν-trace from Fortier and Santocanale to our own settings.

Definition 4.4. An infinite branch of a derivation is a *left μ-trace* if for infinitely many position variables $\mathbf{x1}^{\alpha_1}, \mathbf{x2}^{\alpha_2}, \cdots$ appearing as antecedents of judgments in the branch we can form an infinite chain of inequalities

$$\mathsf{snap}(\mathbf{x1}^{\alpha_1}) >_{\Omega_1} \mathsf{snap}(\mathbf{x2}^{\alpha_2}) >_{\Omega_2} \cdots.$$

[2] This restriction aligns with the computational interpretation of linear logic as session types.

[3] One can maintain their relation despite the polarity change by introducing shifts in the language. We reserve this for a further work.

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{\cdot \vdash 1}\;1R}{\cdot \vdash 1 \oplus \mathtt{ztream}(t)}\;\oplus R}{\cdot \vdash \mathtt{zlist}(t)}\;\mu_{\mathtt{zlist}}R}{1 \vdash \mathtt{zlist}(t)}\;1L}{\dagger\, \mathsf{run}(\cdot, t) \vdash \mathtt{zlist}(t)}\;\mu_{\mathsf{run}}L
\qquad
\dfrac{\overset{\dagger}{\mathsf{run}(x,t) \vdash \mathtt{zlist}(t)}}{\dagger\, \mathsf{run}((skip;x), t) \vdash \mathtt{zlist}(t)}\;\mu_{\mathsf{run}}L
\qquad
\dfrac{\dfrac{\dfrac{\overset{\star}{\mathsf{nrun}\,(x,y,t) \vdash \mathtt{ztream}(t)}}{\mathsf{nrun}\,(x,y,t) \vdash 1 \oplus \mathtt{ztream}(t)}\;\oplus R}{\mathsf{nrun}\,(x,y,t) \vdash \mathtt{zlist}(t)}\;\mu_{\mathtt{zlist}}R}{\dagger\, \mathsf{run}(put\langle x\rangle; y, t) \vdash \mathtt{zlist}(t)}\;\mu_{\mathsf{run}}L$$

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{\mathsf{hd}\,t = \mathsf{z} \vdash \mathsf{hd}\,t = \mathsf{z}}\;\mathtt{ID}}{\mathsf{hd}\,t = \mathsf{z} \,\&\, \mathsf{run}(x;y,\mathsf{tl}\,t) \vdash \mathsf{hd}\,t = \mathsf{z}}\;\&L}{\mathsf{nrun}\,(x,y,t) \vdash \mathsf{hd}\,t = \mathsf{z}}\;\nu_{\mathsf{nrun}}L
\quad
\dfrac{\dfrac{\overset{\dagger}{\mathsf{run}(x;y,\mathsf{tl}\,t) \vdash \mathtt{zlist}(\mathsf{tl}\,t)}}{\mathsf{hd}\,t = \mathsf{z} \,\&\, \mathsf{run}(x;y,\mathsf{tl}\,t) \vdash \mathtt{zlist}(\mathsf{tl}\,t)}\;\&L}{\mathsf{nrun}\,(x,y,t) \vdash \mathtt{zlist}(\mathsf{tl}\,t)}\;\nu_{\mathsf{nrun}}L}{\mathsf{nrun}\,(x,y,t) \vdash \mathsf{hd}\,t = \mathsf{z} \,\&\, \mathtt{zlist}(\mathsf{tl}\,t)}\;\&R}{\star\, \mathsf{nrun}\,(x,y,t) \vdash \mathtt{ztream}(t)}\;\nu_{\mathtt{ztream}}R$$

Figure 5. run produces a possibly infinite list of elements z

$$\dfrac{}{\mathbf{x}^\alpha : A \vdash_\Omega \mathbf{z}^\beta : A}\;fwd
\qquad
\dfrac{\Delta \vdash_\Omega \mathbf{w}^0 : A \quad \Delta', \mathbf{w}^0 : A \vdash_\Omega \mathbf{z}^\beta : C}{\Delta, \Delta' \vdash_\Omega \mathbf{z}^\beta : C}\;Cut$$

$$\dfrac{}{\cdot \vdash_\Omega \mathbf{z}^\beta : 1}\;1R
\qquad
\dfrac{\Delta \vdash_\Omega \mathbf{z}^\beta : C}{\Delta, \mathbf{y}^\alpha : 1 \vdash_\Omega \mathbf{z}^\beta : C}\;1L$$

$$\dfrac{\Delta \vdash_\Omega \mathbf{w}^0 : A_1 \quad \Delta' \vdash_\Omega \mathbf{z}^\beta : A_2}{\Delta, \Delta' \vdash_\Omega \mathbf{z}^\beta : A_1 \otimes A_2}\;\otimes R
\qquad
\dfrac{\Delta, \mathbf{w}^0 : A_1, \mathbf{y}^\alpha : A_2 \vdash_{\Omega \cup \{\mathbf{w}^0 = \mathbf{y}^\alpha\}} \mathbf{z}^\beta : B}{\Delta, \mathbf{y}^\alpha : A_1 \otimes A_2 \vdash_\Omega \mathbf{z}^\beta : B}\;\otimes L$$

$$\dfrac{\Delta, \mathbf{w}^0 : A_1 \vdash_\Omega \mathbf{z}^\beta : A_2}{\Delta \vdash_\Omega \mathbf{z}^\beta : A_1 \multimap A_2}\;\multimap R
\qquad
\dfrac{\Delta \vdash_\Omega \mathbf{w}^0 : A_1 \quad \Delta', \mathbf{y}^\alpha : A_2 \vdash_\Omega \mathbf{z}^\beta : B}{\Delta, \Delta', \mathbf{y}^\alpha : A_1 \multimap A_2 \vdash_\Omega \mathbf{z}^\beta : B}\;\multimap L$$

$$\dfrac{\Delta \vdash_\Omega \mathbf{z}^\beta : A_k \quad k \in I}{\Delta \vdash_\Omega \mathbf{z}^\beta : \oplus\{l_i : A_i\}_{i \in I}}\;\oplus R
\qquad
\dfrac{\Delta, \mathbf{y}^\alpha : A_i \vdash_\Omega \mathbf{z}^\beta : B \quad \forall i \in I}{\Delta, \mathbf{y}^\alpha : \oplus\{l_i : A_i\}_{i \in I} \vdash_\Omega \mathbf{z}^\beta : B}\;\oplus L$$

$$\dfrac{\Delta \vdash_\Omega \mathbf{z} :^\beta A_i \quad \forall i \in I}{\Delta \vdash_\Omega \mathbf{z}^\beta : \&\{l_i : A_i\}_{i \in I}}\;\&R
\qquad
\dfrac{\Delta, \mathbf{y}^\alpha : A_k \vdash_\Omega \mathbf{z}^\beta : B \quad k \in I}{\Delta, \mathbf{y}^\alpha : \&\{l_i : A_i\}_{i \in I} \vdash_\Omega \mathbf{z}^\beta : B}\;\&L$$

$$\dfrac{\Delta \vdash_\Omega \mathbf{z}^\beta : P(t)}{\Delta \vdash_\Omega \mathbf{z}^\beta : \exists x.P(x)}\;\exists R
\qquad
\dfrac{\Delta, \mathbf{y}^\alpha : P(x) \vdash_\Omega \mathbf{z}^\beta : B}{\Delta, \mathbf{y}^\alpha : \exists x.P(x) \vdash_\Omega \mathbf{z}^\beta : B}\;\exists L$$

$$\dfrac{\Delta \vdash_\Omega P :: \mathbf{z}^\beta : P(x)}{\Delta \vdash_\Omega \mathbf{z}^\beta : \forall x.P(x)}\;\forall R
\qquad
\dfrac{\Delta, \mathbf{y}^\alpha : P(t) \vdash_\Omega \mathbf{z}^\beta : B}{\Delta, \mathbf{y}^\alpha : \forall x.P(x) \vdash_\Omega \mathbf{z}^\beta : B}\;\forall L$$

$$\dfrac{\begin{array}{c}\Omega' = \Omega \cup \{\mathbf{z}_i^{\beta+1} = \mathbf{z}_i^\beta \mid i \neq j\} \\ \Delta \vdash_{\Omega'} \mathbf{z}^{\beta+1} : [\bar{t}/\bar{x}]A \quad T(\bar{x}) =_\mu^j A\end{array}}{\Delta \vdash_\Omega \mathbf{z}^\beta : T(\bar{t})}\;\mu R
\qquad
\dfrac{\begin{array}{c}\Omega' = \Omega \cup \{\mathbf{y}_i^{\alpha+1} = \mathbf{y}_i^\alpha \mid i \neq j\} \cup \{\mathbf{y}_j^{\alpha+1} < \mathbf{y}_j^\alpha\} \\ \Delta, \mathbf{y}^{\alpha+1} : [\bar{t}/\bar{x}]A \vdash_\Omega \mathbf{z}^\beta : B \quad T(\bar{x}) =_\mu^j A\end{array}}{\Delta, \mathbf{y}^\alpha : T(\bar{t}) \vdash_\Omega \mathbf{z}^\beta : B}\;\mu L$$

$$\dfrac{\begin{array}{c}\Omega' = \Omega \cup \{\mathbf{z}_i^{\beta+1} = \mathbf{z}_i^\beta \mid i \neq j\} \cup \{\mathbf{z}_j^{\beta+1} < \mathbf{z}_j^\beta\} \\ \Delta \vdash_{\Omega'} \mathbf{z}^{\beta+1} : [\bar{t}/\bar{x}]A \quad T(\bar{x}) =_\nu^j A\end{array}}{\Delta \vdash_\Omega \mathbf{z}^\beta : T(\bar{t})}\;\nu R
\qquad
\dfrac{\begin{array}{c}\Omega' = \Omega \cup \{\mathbf{y}_i^{\alpha+1} = \mathbf{y}_i^\alpha \mid i \neq j\} \\ \Delta, \mathbf{y}^{\alpha+1} : [\bar{t}/\bar{x}]A \vdash_\Omega \mathbf{z}^\beta : B \quad T(\bar{x}) =_\nu^j A\end{array}}{\Delta, \mathbf{y}^\alpha : T(\bar{t}) \vdash_\Omega \mathbf{z}^\beta : B}\;\nu L$$

$$\dfrac{}{\cdot \vdash_\Omega \mathbf{z}^\beta : (s = s)}\;= R
\qquad
\dfrac{\Delta[\theta] \vdash_\Omega \mathbf{z}^\beta : B[\theta] \quad \theta \in \mathsf{mgu}(t, s)}{\Delta, \mathbf{y}^\alpha : (s = t) \vdash_\Omega \mathbf{z}^\beta : B}\;= L$$

Figure 6. Infinitary calculus annotated with position variables and their generations.

Dually, an infinite branch of a derivation is a *right ν-trace* if for infinitely many position variables $\mathbf{y1}^{\beta_1}, \mathbf{y2}^{\beta_2}, \cdots$ appearing as the succedents of judgments in the branch, we can form an infinite chain of inequalities

$$\mathsf{snap}(\mathbf{y1}^{\beta_1}) >_{\Omega_1} \mathsf{snap}(\mathbf{y2}^{\beta_2}) >_{\Omega_2} \cdots$$

.

Definition 4.5 (Validity condition for infinite derivations)**.** An infinite derivation is a *valid proof* if each of its infinite branches is either a left μ-trace or a right ν-trace. A circular *proof* has a valid underlying infinite derivation.

Example 4.6. We can rewrite derivation of Example 2.3 in the annotated calculus as in Figure 7. To check the validity of this derivation, it is enough to observe that

$$\mathsf{snap}(\mathbf{x}^2) = [\mathbf{x}_1^2, \mathbf{x}_2^2] <_{\Omega_6} [\mathbf{x}_1^0, \mathbf{x}_2^0] = \mathsf{snap}(\mathbf{x}^0).$$

Since the annotation of position variables is straightforward, for the sake of conciseness, we present future examples as circular derivations in the calculus of Figure 1. We also use pattern matching whenever possible. All derivations presented in this paper are valid by this definition. We leave it to the reader to check their validity.

5 A productive cut elimination algorithm

Fortier and Santocanale [10] introduced a cut elimination algorithm for infinite pre-proofs in singleton logic with fixed points. They proved that for infinite proofs satisfying their guard condition the algorithm is productive. In this section we adapt their cut elimination algorithm to $\mathit{FIMALL}^{\infty}_{\mu,\nu}$ and prove its productivity for valid derivations. The algorithm receives an infinite proof as an input and outputs a cut-free infinite proof. Since we are dealing with infinite derivations, to make the algorithm productive we need to push every cut away from the root with a lazy strategy (BFS). With this strategy we may need to permute two consecutive cuts which results into a loop. To overcome this problem, similar to Fortier and Santocanale and also Baelde and Miller [4] we generalize binary cuts to n-ary cuts using the notion of a *branching tape* the prior notion of *tape*.

Definition 5.1. A *branching tape* C is a finite list of sequents $\Delta \vdash \mathbf{w}^{\beta} : A$[4], such that

- Every two judgments $\Delta \vdash \mathbf{w}^{\beta} : A$ and $\Delta' \vdash \mathbf{w'}^{\beta'} : A'$ on the tape share at most one position variable $\mathbf{z}^{\alpha} : B$. If they share such position variable, we call them connected. Moreover, assuming that $\Delta \vdash \mathbf{w}^{\beta} : A$ appears before $\Delta' \vdash \mathbf{w'}^{\beta'} : A'$ on the list, we have $\mathbf{z}^{\alpha} : B \in \Delta'$ and $\mathbf{z}^{\alpha} : B = \mathbf{w}^{\beta} : A$.
- Each position variable $\mathbf{z}^{\beta}$ appears at most twice in a tape and if it appears more than once it connects two judgments.
- Every tape is connected and acyclic.

The *conclusion* $\mathsf{conc}_{\mathcal{M}}$ of a branching tape $\mathcal{M}$ is a sequent $\Delta \vdash \mathbf{x}^{\alpha} : A$ such that

- there is a sequent $\Delta' \vdash \mathbf{x}^{\alpha} : A$ in the tape that $\mathbf{x}^{\alpha} : A$ does not connect it to any other sequent in the tape.
- For every $\mathbf{y}^{\beta} : B \in \Delta$ there is a sequent $\Delta', \mathbf{y}^{\beta} : B \vdash \mathbf{z}^{\gamma} : C$ on the tape such that $\mathbf{y}^{\beta} : B$ does not connect it to any other sequent in the tape.

We call Δ the set of *leftmost formulas* of $\mathcal{M}$: $\mathsf{lft}(\mathcal{M})$. And $x^{\alpha} : A$ is the *rightmost formula* of tape $\mathcal{M}$: $\mathsf{rgt}(\mathcal{M})$.

The conclusion of a branching tape always exists and is unique. An n-ary cut is a rule formed from a tape $\mathcal{M}$ and its conclusion $\mathsf{conc}_{\mathcal{M}}$: $\dfrac{\mathcal{M}}{\mathsf{conc}_{\mathcal{M}}}\ \mathit{nCut}$

We generalize Fortier and Santaconale's set of primitive operations to account for $\mathit{FIMALL}^{\infty}_{\mu,\nu}$. They closely resemble the reduction rules given by Doumane [9]. Figure 8 depicts a few interesting Internal and External reductions [5].

Our cut elimination algorithm is given as Algorithms 1 and 2. The output of the algorithm is a tree labelled by $\{0, 1\}$. For each node $w \in \{0, 1\}^*$ of the tree it also identifies the corresponding sequent, $s(w)$, and the rule applied on the node, $r(w)$. The algorithm *Treat* reduces the sequence in a branching tape with internal reductions until either a left rule is applied on one of its leftmost formulas or a right rule is applied on its rightmost formula. While this condition holds, the algorithm applies a *flip* rule on a leftmost/rightmost formula of the tape. The flipping step is always productive since it pushes a cut one step up. It suffices to show that the treating part is terminating to prove productivity of the algorithm.

Theorem 5.2. *For every input tape M, computation of $\mathit{Treat}(M)$ halts.*

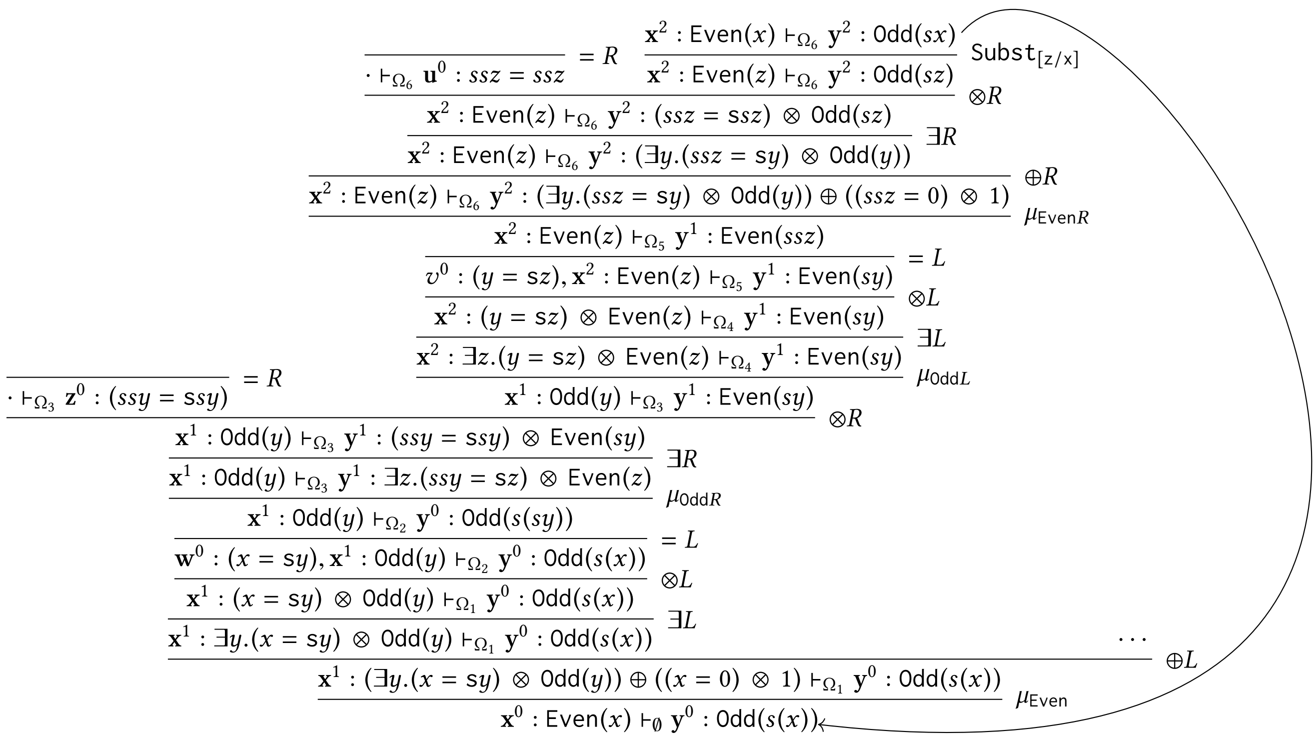
Proof. By a proof similar to FS, except that we use μ-threads instead of ν-threads to show that $\mathit{Treat}(M)$ does not have an infinite computation tree. Assume for the sake of contradiction that $\mathit{Treat}(M)$ has an infinite computation tree Ψ. We prove the following three contradictory statements, where Here $<$ and $\bigwedge$ are defined according to the lexicographic order on the tree Ψ.

(i) The greatest infinite branch of Ψ is a μ-branch.
(ii) Let E be a nonempty collection of μ-branches and let $\gamma = \bigwedge E$. Then γ is a μ-branch.
(iii) If β is a μ-branch, then there exists another μ-branch $\beta' < \beta$.

The complete proof is given in the Appendix. □

[4]For brevity we elide the set Ω in the judgments.

[5] $C_{\Delta'_1}$ in the fourth operation of Figure 8 is a subset of the tape C connected to Δ'_1. By definition of tape, two sets $C_{\Delta'_1}$ and $C_{\Delta'_2}$ partition C.



$$\Omega_1 = \{\mathbf{x}^1_2 < \mathbf{x}^0_2, \mathbf{x}^1_1 = \mathbf{x}^0_1\},\ \Omega_2 = \Omega_1 \cup \{\mathbf{w}^0_2 = \mathbf{x}^1_2, \mathbf{w}^0_1 = \mathbf{x}^1_1\},\ \Omega_3 = \Omega_2 \cup \{\mathbf{y}^1_1 = \mathbf{y}^0_1\},\ \Omega_4 = \Omega_3 \cup \{\mathbf{x}^2_2 < \mathbf{x}^1_2, \mathbf{x}^2_1 = \mathbf{x}^1_1\},$$
$$\Omega_5 = \Omega_4 \cup \{v^0_2 = \mathbf{x}^2_2, v^0_1 = \mathbf{x}^2_1\}, \text{ and } \Omega_6 = \Omega_5 \cup \{\mathbf{y}^2_1 = \mathbf{y}^1_1\}.$$

Figure 7. Successor of an even number is odd in the annotated calculus.

6 Adding finite structural derivations

In the proof of theorems about linear structures, we may want to appeal to some first order (structural) theories such as arithmetic. To allow a restricted form of such structural (that is, nonlinear) reasoning in our system we add nonlinear formulas A_s to the linear system as

$$A_l ::= \cdots \mid \downarrow A_s$$

labeling all linear propositions as A_l. We do not prescribe the exact syntax for nonlinear proposition since our development is parametric in those (subject to a few assumptions) and may vary with the particular application.

Nonlinear judgments are of the form $\Psi \Vdash A_s$. The only restriction we put on the calculus for pure structural judgments is to have the cut elimination property. We generalize linear judgments to $\Psi; \Delta \vdash_\Omega x^\alpha : A_l$ where Ψ is a structural context (i.e., it allows weakening and contraction) and Δ a linear context. Judgments of the rules in Figure 6 are refined by adding an extra structural context Ψ to them. This change is straightforward since none of these rules affect the structural context Ψ. We add the following three rules to Figure 6 to connect linear and structural reasoning.

$$\frac{\Psi, A_s; \Delta \vdash_\Omega \mathbf{z}^\beta : B_l}{\Psi; \Delta, x^\alpha :\downarrow A_s \vdash_\Omega \mathbf{z}^\beta : B_l} \downarrow L \qquad \frac{\Psi \Vdash B_s}{\Psi; \cdot \vdash_\Omega x^\alpha :\downarrow B_s} \downarrow R$$

$$\frac{\Psi \Vdash A_s \quad \Psi, A_s; \Gamma \vdash_\Omega B_l}{\Psi; \Gamma \vdash_\Omega B_l} \mathsf{Cut}_{sl}$$

In both $\downarrow L$ and $\downarrow R$ rules, when position variable x^α becomes structural we simply delete it. The Cut_{sl} rule does not create any fresh position variable either. So our validity condition remains intact after adding the structural component.

There are three different cut rules in this system:

$$\frac{\Psi \Vdash B_s \quad \Psi, B_s \Vdash A_s}{\Psi \Vdash A_s} \mathsf{Cut}_{ss} \qquad \frac{\Psi \Vdash A_s \quad \Psi, A_s; \Gamma \vdash_\Omega B_l}{\Psi; \Gamma \vdash_\Omega B_l} \mathsf{Cut}_{sl}$$

$$\dfrac{C_1 \quad \dfrac{\Delta'_1 \vdash \mathbf{u}^0 : A_1 \quad \Delta'_2 \vdash \mathbf{z}^\beta : A_2}{\Delta' \vdash \mathbf{z}^\beta : A_1 \otimes A_2}\otimes R \quad C_2 \quad \dfrac{\Delta'', \mathbf{u}^0 : A_1, \mathbf{z}^\beta : A_2 \vdash \mathbf{w}^\alpha : B}{\Delta'', z^\beta : A_1 \otimes A_2 \vdash \mathbf{w}^\alpha : B}\otimes L \quad C_3}{\Delta \vdash \mathbf{v} : C}\, nCut \xRightarrow{\text{Reduce}} \dfrac{C_1 \quad \Delta'_1 \vdash \mathbf{u}^0 : A_1 \quad \Delta'_2 \vdash \mathbf{z}^\beta : A_2 \quad C_2 \quad \Delta'', \mathbf{u}^0 : A_1, \mathbf{z}^\beta : A_2 \vdash \mathbf{w}^\alpha : B \quad C_3}{\Delta \vdash \mathbf{v} : C}\, nCut$$

$$\dfrac{C_1 \quad \dfrac{\Delta', \mathbf{u}^0 : A_1 \vdash \mathbf{z}^\beta : A_2}{\Delta' \vdash \mathbf{z}^\beta : A_1 \multimap A_2}\multimap R \quad C_2 \quad \dfrac{\Delta''_1 \vdash \mathbf{u}^0 : A_1 \quad \Delta''_2, \mathbf{z}^\beta : A_2 \vdash \mathbf{w}^\alpha : B}{\Delta'', \mathbf{z}^\beta : A_1 \multimap A_2 \vdash \mathbf{w}^\alpha : B}\multimap L \quad C_3}{\Delta \vdash \mathbf{v} : C}\, nCut \xRightarrow{\text{Reduce}} \dfrac{C_1 \quad C_2 \quad \Delta''_1 \vdash \mathbf{u}^0 : A_1 \quad \Delta', \mathbf{u}^0 : A_1 \vdash \mathbf{z}^\beta : A_2 \quad \Delta''_2, \mathbf{z}^\beta : A_2 \vdash \mathbf{w}^\alpha : B \quad C_3}{\Delta \vdash \mathbf{v} : C}\, nCut$$

$$\dfrac{C_1 \quad \dfrac{\Delta' \vdash \mathbf{z}^{\beta+1} : [\bar{t}/\bar{x}]A \quad T(\bar{x}) =_\mu A}{\Delta' \vdash \mathbf{z}^\beta : T(\bar{t})}\mu R \quad C_2 \quad \dfrac{\Delta'', \mathbf{z}^{\beta+1} : [\bar{t}/\bar{x}]A \vdash \mathbf{w}^\alpha : B \quad T(\bar{x}) =_\mu A}{\Delta'', \mathbf{z}^\beta : T(\bar{t}) \vdash \mathbf{w}^\alpha : B}\mu L \quad C_3}{\Delta \vdash \mathbf{v} : C}\, nCut \xRightarrow{\text{Reduce}} \dfrac{C_1 \quad \Delta' \vdash \mathbf{z}^{\beta+1} : [\bar{t}/\bar{x}]A \quad C_2 \quad \Delta'', \mathbf{z}^{\beta+1} : [\bar{t}/\bar{x}]A \vdash \mathbf{w}^\alpha : B \quad C_3}{\Delta \vdash \mathbf{v} : C}\, nCut$$

$$\dfrac{C_1 \quad \dfrac{\Delta' \vdash \mathbf{z}^\beta : P(t)}{\Delta' \vdash \mathbf{z}^\beta : \exists x.P(x)}\exists R \quad C_2 \quad \dfrac{\overset{\Pi'}{\Delta'', \mathbf{z}^\beta : P(x) \vdash \mathbf{w}^\alpha : B}}{\Delta'', \mathbf{z}^\beta : \exists x.P(x) \vdash \mathbf{w}^\alpha : B}\exists L \quad C_3}{\Delta \vdash \mathbf{v} : C}\, nCut \xRightarrow{\text{Reduce}} \dfrac{C_1 \quad \Delta' \vdash \mathbf{z}^\beta : P(t) \quad C_2 \quad \overset{\Pi'[t/x]}{\Delta'', \mathbf{z}^\beta : P(t) \vdash \mathbf{w}^\alpha : B} \quad C_3}{\Delta \vdash \mathbf{v} : C}\, nCut$$

$$\dfrac{C_1 \quad \dfrac{}{\cdot \vdash \mathbf{z}^\beta : s = s} = R \quad C_2 \quad \dfrac{\Delta'' \vdash \mathbf{w}^\alpha : B}{\Delta'', \mathbf{z}^\beta : s = s \vdash \mathbf{w}^\alpha : B} = L \quad C_3}{\Delta \vdash \mathbf{v} : C}\, nCut \xRightarrow{\text{Reduce}} \dfrac{C_1 \quad C_2 \quad \Delta'' \vdash \mathbf{w}^\alpha : B \quad C_3}{\Delta \vdash \mathbf{v} : C}\, nCut$$

$$\dfrac{C \quad \dfrac{\Delta'_1 \vdash \mathbf{u}^0 : A_1 \quad \Delta'_2 \vdash \mathbf{z}^\beta : A_2}{\Delta'_1, \Delta'_2 \vdash \mathbf{z}^\beta : A_1 \otimes A_2}\otimes R}{\Delta_1, \Delta_2 \vdash \mathbf{z}^\beta : A_1 \otimes A_2}\, nCut \xRightarrow{\text{RFLIP}} \dfrac{\dfrac{C_{\Delta'_1} \quad \Delta'_1 \vdash \mathbf{u}^0 : A_1}{\Delta_1 \vdash \mathbf{u}^0 : A_1}\, nCut \quad \dfrac{C_{\Delta'_2} \quad \Delta'_2 \vdash \mathbf{z}^\beta : A_2}{\Delta_2 \vdash \mathbf{z}^\beta : A_2}\, nCut}{\Delta_1, \Delta_2 \vdash \mathbf{z}^\beta : A_1 \otimes A_2}\otimes R$$

$$\dfrac{C_1 \quad \dfrac{\Delta', \mathbf{u}^0 : A_1, \mathbf{z}^\beta : A_2 \vdash \mathbf{w}^\alpha : B}{\Delta', \mathbf{z}^\beta : A_1 \otimes A_2 \vdash \mathbf{w}^\alpha : B}\otimes L \quad C_2}{\Delta, \mathbf{z}^\beta : A_1 \otimes A_2 \vdash \mathbf{v} : C}\, nCut \xRightarrow{\text{LFLIP}} \dfrac{\dfrac{C_1 \quad \Delta', \mathbf{u}^0 : A_1, \mathbf{z}^\beta : A_2 \vdash \mathbf{w}^\alpha : B \quad C_2}{\Delta, \mathbf{u}^0 : A_1, \mathbf{z}^\beta : A_2 \vdash \mathbf{u}^0 : A_1}\, nCut}{\Delta, \mathbf{z}^\beta : A_1 \otimes A_2 \vdash \mathbf{v} : C}\otimes L$$

$$\dfrac{C_1 \quad \dfrac{\Delta'[\theta] \vdash \mathbf{w}^\alpha : B'[\theta] \quad \theta \in \mathtt{mgu}(t, s)}{\Delta', \mathbf{z}^\beta : s = t \vdash \mathbf{w}^\alpha : B} = L \quad C_2}{\Delta, \mathbf{z}^\beta : s = t \vdash \mathbf{w}^\alpha : B}\, nCut \xRightarrow{\text{LFLIP}} \dfrac{\dfrac{C_1[\theta] \quad \Delta'[\theta] \vdash \mathbf{w}^\alpha : B'[\theta] \quad C_2[\theta]}{\Delta[\theta] \vdash \mathbf{w}^\alpha : B[\theta]}\, nCut \quad \theta \in \mathtt{mgu}(t, s)}{\Delta, \mathbf{z}^\beta : s = t \vdash \mathbf{w}^\alpha : B} = L$$

$$\dfrac{C_1 \quad \dfrac{}{\mathbf{x}^\alpha : A \vdash \mathbf{w}^\gamma : A} ID \quad C_2}{\Delta \vdash \mathbf{z}^\beta : C}\, nCut \xRightarrow{\text{ID-Elim}} \dfrac{C_1 \quad C_2[\mathbf{x}^\alpha / \mathbf{w}^\gamma]}{\Delta \vdash \mathbf{z}^\beta : C}\, nCut$$

Figure 8. Primitive operations

$$\dfrac{\Psi; \Delta \vdash_\Omega \mathbf{w}^0 : A \quad \Psi; \Delta', \mathbf{w}^0 : A \vdash_\Omega \mathbf{z}^\beta : C}{\Psi; \Delta, \Delta' \vdash_\Omega \mathbf{z}^\beta : C}\, \mathrm{Cut}_{ll}$$

By assumption, Cut_{ss} can be eliminated. It is enough to eliminate Cut_{ll} and Cut_{sl} rules in a productive way. We define a generalized n-ary cut to account for the two latter two cut rules.

Definition 6.1. A *generalized branching tape* $\mathcal{M}_\Psi$ is of the form $\mathcal{S}_\Psi \mid C_\Psi$ where C_Ψ is a branching tape by Definition 5.1 and $\mathcal{S}_\Psi$ is a set of structural judgments $\Psi \Vdash A_s$. For each structural judgment $\Psi \Vdash A_s \in \mathcal{S}$, formula A_s appears in the structural context of at least one judgment in C_Ψ. For a judgment $\Psi'; \Delta \vdash B_l$ in C_Ψ, we have $\Psi' = \Psi, \Phi$, where all

Algorithm 1: Cut elimination algorithm

Initialization: $\Lambda \leftarrow \emptyset; Q \leftarrow [(\epsilon, [v])]$; v is the root sequent. $\rho(s)$ is the rule applied on formula annotated with position variable s, it can either be an ID, Cut, a L rule, or a R rule. $\mathsf{lft}(M)$ and $\mathsf{rgt}(M)$ are defined in Definition 5.1. The FLip rules will return a rule that they permuted down, the sequent corresponding to that, and a list *List* of one or two tapes.

```
while Q ≠ ∅ do
    (w, M) ← pull(Q);
    Λ ← Λ ∪ {w};
    M ← Treat(M);
    if |M| = 1 and ρ(lft(M)) = ID then
        (r(w), s(w), List) ← IdOut(M);
    else
        if ρ(lft(M)) ∈ L then
            (r(w), s(w), List) ← LFlip(M);
        else
            if ρ(rgt(M)) ∈ R then
                (r(w), s(w), List) ← RFlip(M);
            end
        end
        if List = [M′] then
            push((w0, M′), Q);
        else
            if List = [M′0, M′1] then
                push((w0, M′0), Q);
                push((w1, M′1), Q;
            end
        end
    end
end
```

formulas in Φ are the succedent of a structural judgment in $\mathcal{S}_\Psi$.

The generalized n-ary cut rule is

$$\frac{\mathcal{M}_\Psi}{\Psi; \Delta \vdash \mathbf{x}^\alpha : B_l}\ nCut$$

Where $\Delta \vdash \mathbf{x}^\alpha : B_l$ is the conclusion of the linear part of the tape as defined in Definition 5.1. We add one reduction step, a rule to merge a Cut_{sl} rule to the tape, and two flips to the set of primitive operations (Figure 9). We keep all the primitive operations for the purely linear system. To preserve the invariants of the (generalized) branching tape in some of the operations we silently remove the structural sequents in which their succedent is not used in any linear judgment. In the Reduce, Merge−Cut, and RFlip rules we know that $\Psi_1 = \Psi, \Phi$, where all elements in Φ appear as a succedent in S_Ψ. By cut elimination for pure linear judgments we get a proof for $\Psi \Vdash A_s$. In LFlip rule we use admissibility of Weakening

Algorithm 2: Treat Function

Initialization: M is a branching tape. i and j in $\mathsf{Reduce}(M, i, j)$ are the index of the two sequents in tape on which the reduction rules are applied. Similarly i in $\mathsf{Merge}(M, i)$ and idElim is the index of the sequent in the tape on which the corresponding rule is applied. $\rho'(i)$ is the rule applied on the i-th sequent of the tape, it can either be an ID, Cut, a L rule, or a R rule.

```
while ρ(lft(M)) ∉ L and ρ(rgt(M)) ∉ R do
    if |M| > 1 and ∃i ∈ M : ρ′(i) = ID then
        M ← IdElim(M, i);
    else
        if ∃i ∈ M : ρ′(i) = Cut then
            M ← Merge(M, i);
        else
            if ∃i, j.ρ′(i) ∈ R & ρ′(j) ∈ L then
                M ← Reduce(M, i, j);
            end
        end
    end
end
```

for the structural context: We can prove coinductively that if there is a derivation for $\Psi; \Delta \vdash \mathbf{x}^\alpha : A_l$ in our calculus, there is also a derivation for $\Psi, B_s; \Delta \vdash \mathbf{x}^\alpha : A_1$ with the same structure.

The next example shows how to use a structural context to prove a property of infinite streams.

Example 6.2 (Lexicographic order on streams)**.** We define the lexicographic order on streams [12] in signature Σ_5 as a negative predicate

$$x \leq y \quad =^1_\nu \quad \downarrow(\mathsf{hd}x < \mathsf{hd}y) \oplus (\downarrow(\mathsf{hd}x = \mathsf{hd}y) \,\&\, \mathsf{tl}x \leq \mathsf{tl}y).$$

where the relation $<$ is a transitive partial order on the elements of streams. Our goal is to show that relation $\leq$ is transitive by using the (structural) first order theory of orders ($\mathbb{O}$). In Figure 10 we show two branches of this proof, the rest of the proof can be completed in a similar way.

We can define even and odd predicates alternatively using structural arithmetic formulas. In the next example we show how these alternative definitions can be deduced from the ones we introduced in Example 2.1.

Example 6.3. Define Signature Σ_6 to be

$$\begin{array}{lll@{\qquad}lll} \mathsf{Odd}(\mathsf{z}) & =^1_\mu & 0 & \mathsf{Odd}(sy) & =^1_\mu & \mathsf{Even}(y) \\ \mathsf{Even}(\mathsf{z}) & =^1_\mu & 1 & \mathsf{Even}(sy) & =^1_\mu & \mathsf{Odd}(y) \end{array}$$

$$\mathsf{E}(x) \;=^2_\mu\; \exists y.\downarrow(x = 2y) \qquad \mathsf{O}(x) \;=^2_\mu\; \exists y.\downarrow(x = 2y+1)$$

Put $\mathbb{P}$ to be the rules of arithmetic. We present circular derivations for $\star\ \mathbb{P}; \mathsf{Even}(x) \vdash \mathsf{E}(x)$ and $\dagger\ \mathbb{P}; \mathsf{Odd}(x) \vdash \mathsf{O}(x)$ in Figure 11. These derivations satisfy the validity condition since in every infinite branch infinitely many $\mu_{\mathsf{Odd}}L$ and $\mu_{\mathsf{Even}}L$ rules are applied on the antecedent.

$$\dfrac{\mathcal{S}_\Psi \mid C_{1\Psi} \quad \dfrac{\Psi_1 \Vdash A_s}{\Psi_1; . \vdash \mathbf{z}^\beta :\downarrow A_s}\downarrow R \quad C_{2\Psi} \quad \dfrac{\Psi_2, A_s; \Delta' \vdash \mathbf{w}^\alpha : B}{\Psi_2; \Delta', \mathbf{z}^\beta :\downarrow A_s \vdash \mathbf{w}^\alpha : B}\downarrow L \quad C_{3\Psi}}{\Psi; \Delta \vdash \mathbf{v} : C}\, nCut \quad \overset{\text{Reduce}}{\Longrightarrow} \quad \dfrac{\mathcal{S}_\Psi, \Psi \Vdash A_s \mid C_{1\Psi} \quad C_{2\Psi} \quad \Psi_2, A_s; \Delta' \vdash \mathbf{w}^\alpha : B \quad C_{3\Psi}}{\Psi; \Delta \vdash \mathbf{v} : C}\, nCut$$

$$\dfrac{\mathcal{S}_\Psi \mid C_{1\Psi} \quad \dfrac{\Psi_1 \Vdash A_s \quad \Psi_1, A_s; \Delta' \vdash \mathbf{w}^\alpha : B}{\Psi_1; \Delta', \mathbf{z}^\beta : B \vdash \mathbf{w}^\alpha : B}\,\mathsf{Cut}_{sl} \quad C_{2\Psi}}{\Psi; \Delta \vdash \mathbf{v} : C}\, nCut \quad \overset{\text{Merge-Cut}}{\Longrightarrow} \quad \dfrac{\mathcal{S}_\Psi, \Psi \Vdash A_s \mid C_{1\Psi} \quad \Psi_1, A_s; \Delta' \vdash \mathbf{w}^\alpha : B \quad C_{2\Psi}}{\Psi; \Delta \vdash \mathbf{v} : C}\, nCut$$

$$\dfrac{\mathcal{S}_\Psi \quad \mid \cdot \quad \dfrac{\Psi_1 \Vdash A_s}{\Psi_1; \cdot \vdash \mathbf{z}^\beta :\downarrow A_s}\downarrow R}{\Psi; \cdot \vdash \mathbf{z}^\beta :\downarrow A_s}\, nCut \quad \overset{\text{RFLip}}{\Longrightarrow} \quad \dfrac{\Psi \Vdash A_s}{\Psi; \cdot \vdash \mathbf{z}^\beta :\downarrow A_s}\downarrow R$$

$$\dfrac{\mathcal{S}_\Psi \mid C_{1\Psi} \quad \dfrac{\Psi_1, A_s; \Delta' \vdash \mathbf{w}^\alpha : B}{\Psi_1; \Delta', \mathbf{z}^\beta :\downarrow A_s \vdash \mathbf{w}^\alpha : B}\downarrow L \quad C_{2\Psi}}{\Psi; \Delta, \mathbf{z}^\beta :\downarrow A_s \vdash \mathbf{v} : C}\, nCut \quad \overset{\text{LFlip}}{\Longrightarrow} \quad \dfrac{\dfrac{\mathcal{S}_{\Psi, A_s} \mid C_{1\Psi, A_s} \quad \Psi_1, A_s; \Delta' \vdash \mathbf{w}^\alpha : B \quad C_{2\Psi, A_s}}{\Psi, A_s; \Delta \vdash \mathbf{v} : C}\, nCut}{\Psi; \Delta, \mathbf{z}^\beta :\downarrow A_s \vdash \mathbf{v} : C}\downarrow L$$

Figure 9. Primitive operations with structural component

$$\dfrac{\dfrac{\cdots \quad \dfrac{\dfrac{\cdots \quad \dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\mathit{Structural - Proof}}{\mathbb{O}, \mathsf{hd}x = \mathsf{hd}y, \mathsf{hd}y = \mathsf{hdz} \Vdash \mathsf{hd}x = \mathsf{hdz}}}{\mathbb{O}, \mathsf{hd}x = \mathsf{hd}y, \mathsf{hd}y = \mathsf{hdz}; \cdot \vdash \downarrow (\mathsf{hd}x = \mathsf{hdz})}\downarrow R}{\mathbb{O}, \mathsf{hd}x = \mathsf{hd}y; \downarrow (\mathsf{hd}y = \mathsf{hdz}) \vdash \downarrow (\mathsf{hd}x = \mathsf{hdz})}\downarrow L}{\mathbb{O}; \downarrow (\mathsf{hd}x = \mathsf{hd}y), \downarrow (\mathsf{hd}y = \mathsf{hdz}) \vdash \downarrow (\mathsf{hd}x = \mathsf{hdz})}\downarrow L \quad \dfrac{\overset{\star}{\mathbb{O}; x \leq y, y \leq z \vdash x \leq z}}{\mathbb{O}; \mathsf{tl}x \leq \mathsf{tl}y, \mathsf{tl}y \leq \mathsf{tlz} \vdash \mathsf{tl}x \leq \mathsf{tlz}}\,\mathtt{Subst}_{[\mathtt{tl}x, \mathtt{tl}y, \mathtt{tl}z/x,y,z]}}{\mathbb{O}; \downarrow (\mathsf{hd}x = \mathsf{hd}y) \,\&\, \mathsf{tl}x \leq \mathsf{tl}y, \downarrow (\mathsf{hd}y = \mathsf{hdz}) \,\&\, \mathsf{tl}y \leq \mathsf{tlz} \vdash (\downarrow (\mathsf{hd}x = \mathsf{hdz}) \& \mathsf{tl}x \leq \mathsf{tlz})}\,\&R, \&L, \&L}{\mathbb{O}; \downarrow (\mathsf{hd}x = \mathsf{hd}y) \,\&\, \mathsf{tl}x \leq \mathsf{tl}y, \downarrow (\mathsf{hd}y = \mathsf{hdz}) \,\&\, \mathsf{tl}y \leq \mathsf{tlz} \vdash \downarrow (\mathsf{hd}x < \mathsf{hdz}) \oplus (\downarrow (\mathsf{hd}x = \mathsf{hdz}) \& \mathsf{tl}x \leq \mathsf{tlz})}\oplus R}{\mathbb{O}; \downarrow (\mathsf{hd}x = \mathsf{hd}y) \,\&\, \mathsf{tl}x \leq \mathsf{tl}y, \downarrow (\mathsf{hd}y = \mathsf{hdz}) \,\&\, \mathsf{tl}y \leq \mathsf{tlz} \vdash x \leq z}}{\mathbb{O}; \downarrow (\mathsf{hd}x = \mathsf{hd}y) \,\&\, \mathsf{tl}x \leq \mathsf{tl}y, \downarrow (\mathsf{hd}y < \mathsf{hdz}) \oplus (\downarrow (\mathsf{hd}y = \mathsf{hdz}) \,\&\, \mathsf{tl}y \leq \mathsf{tlz}) \vdash x \leq z}\oplus L}{\mathbb{O}; \downarrow (\mathsf{hd}x = \mathsf{hd}y) \,\&\, \mathsf{tl}x \leq \mathsf{tl}y, y \leq z \vdash x \leq z}\,\nu_{\leq} L}{\mathbb{O}; \downarrow (\mathsf{hd}x < \mathsf{hd}y) \oplus (\downarrow (\mathsf{hd}x = \mathsf{hd}y) \& \mathsf{tl}x \leq \mathsf{tl}y), y \leq z \vdash x \leq z}\oplus L}{\star\, \mathbb{O}; x \leq y, y \leq z \vdash x \leq z}\,\nu_{\leq_{\leq}} L$$

Figure 10. Lexicographic order on streams is transitive.

7 Conclusion

In this paper we introduced an infinitary sequent calculus for first order intuitionistic multiplicative additive linear logic with fixed points. This system is mainly designed for linear reasoning but we also allow appealing to first order theories such as arithmetic, by adding an adjoint downgrade modality. Inspired by the work of Fortier and Santocanale [10] we provide an algorithm to identify valid proofs among all infinite derivations. We have provided several examples to show the strength of calculus in proving theorems about mutually inductive and coinductive data types.

One of our main motivations for introducing this calculus was to have a system for reasoning about programs behaviour. In particular we want to use this calculus to give a direct proof for the strong progress property of locally valid binary session typed processes [8]. The importance of a direct proof other than its elegance is that it can be adapted for a more general validity condition on processes without the need to prove cut elimination productivity for their underlying derivations.

The connection to the type theoretic approach by Abel et al [1] is an interesting item for future research. A first step in this general direction was taken by Sprenger and Dam [19] who justify cyclic inductive proofs using inflationary iteration.

A Appendix

Lemma A.1 (Substitution). *For a valid derivation*

$$\dfrac{\Pi}{\Delta \vdash \mathbf{w}^\alpha : A}$$

$$
\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\textit{Structural-proof}}{\mathbb{P} \Vdash \mathsf{z} = 2\mathsf{z}}}{\mathbb{P}; \cdot \vdash\downarrow \mathsf{z} = 2\mathsf{z}}\ \downarrow R}{\mathbb{P}; \cdot \vdash \exists y. \downarrow (\mathsf{z} = 2y)}\ \exists R}{\mathbb{P}; \cdot \vdash \mathsf{E}(\mathsf{z})}\ \mu_{\mathsf{E}} R}{\mathbb{P}; 1 \vdash \mathsf{E}(\mathsf{z})}\ 1L}{\star\ \mathbb{P}; \mathsf{Even}(\mathsf{z}) \vdash \mathsf{E}(\mathsf{z})}\ \mu_{\mathsf{Even}} L
$$

$$
\dfrac{\dfrac{\dfrac{\dagger}{\mathbb{P}; \mathsf{Odd}(x) \vdash \mathsf{O}(x)} \quad \dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\textit{Structural}-\textit{Proof}}{\mathbb{P}, (x = 2w+1) \Vdash (sx = 2(sw))}}{\mathbb{P}, (x = 2w+1); \cdot \vdash\downarrow (sx = 2(sw))}\ \downarrow R}{\mathbb{P}; \downarrow (x = 2w+1) \vdash\downarrow (sx = 2(sw))}\ \downarrow L}{\mathbb{P}; \downarrow (x = 2w+1) \vdash \exists y. \downarrow (sx = 2y)}\ \exists R}{\mathbb{P}; \exists w. \downarrow (x = 2w+1) \vdash \exists y. \downarrow (sx = 2y)}\ \exists L}{\mathbb{P}; \mathsf{O}(x) \vdash \exists y. \downarrow (sx = 2y)}\ \mu_{\mathsf{O}} L}{\mathbb{P}; \mathsf{Odd}(x) \vdash \mathsf{E}(sx)}\ \mathsf{Cut}_{ll}}{\star\ \mathbb{P}; \mathsf{Even}(sx) \vdash \mathsf{E}(sx)}\ \mu_{\mathsf{Even}} L
$$

$$
\dfrac{\dfrac{\dfrac{\star}{\mathbb{P}; \mathsf{Even}(x) \vdash \mathsf{E}(x)} \quad \dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\textit{Structural-Proof}}{\mathbb{P}, (x = 2w) \Vdash (sx = 2w+1)}}{\mathbb{P}, (x = 2w); \cdot \vdash\downarrow (sx = 2w+1)}\ \downarrow R}{\mathbb{P}; \downarrow (x = 2w) \vdash\downarrow (sx = 2w+1)}\ \downarrow L}{\mathbb{P}; \downarrow (x = 2w) \vdash \exists y. \downarrow (sx = 2y+1)}\ \exists R}{\mathbb{P}; \exists w. \downarrow (x = 2w) \vdash \exists y. \downarrow (sx = 2y+1)}\ \exists L}{\mathbb{P}; \mathsf{E}(x) \vdash \exists y. \downarrow (sx = 2y+1)}\ \mu_{\mathsf{E}} L}{\mathbb{P}; \mathsf{Even}(x) \vdash \mathsf{O}(sx)}\ \mathsf{Cut}_{ll}}{\dagger\ \mathbb{P}; \mathsf{Odd}(sx) \vdash \mathsf{O}(sx)}\ \mu_{\mathsf{Odd}} L
$$

Figure 11. Structural definition of even and odd numbers.

in the infinite system and substitution θ, there is a valid derivation for

$$\dfrac{\Pi[\theta]}{\Delta[\theta] \vdash \mathbf{w}^{\alpha} : A[\theta]}$$

Where $\Pi[\theta]$ is the whole derivation Π or a prefix of it instantiated by θ.

Proof. The proof is by coinduction on the structure of

$$\Delta \vdash \mathbf{w}^{\alpha} : A.$$

The only interesting case is where we get to the $= L$ rule.

$$\dfrac{\begin{matrix}\Pi' \\ \Gamma[\theta'] \vdash B[\theta']\end{matrix} \quad \theta' \in \mathsf{mgu}(t, s)}{\Gamma, s = t \vdash B} = L$$

If the set $\mathsf{mgu}(t[\theta], s[\theta])$is empty then so is $\Pi[\theta]$. Otherwise if η is the single element of $\mathsf{mgu}(t[\theta], s[\theta])$, then for some substitution λ we have $\theta\eta = \theta'\lambda$, and we can form the rest of derivation for substitution λ as $\Pi'[\lambda]$ coinductively. □

Theorem 5.2. *For every input tape M, computation of $Treat(M)$ halts.*

Proof. We show that $Treat(M)$ does not have an infinite computation tree. Assume for the sake of contradiction that $Treat(M)$ has an infinite computation tree and gets into an infinite loop. We follow the proof by Fortier and Santocanale [10](FS) closely.

Put M_i for $i \geq 1$ to be the branching tape in memory before the i-th turn of the loop, with $M_1 = M$. We build the full trace T of the algorithm with essentially the same transition rules as in FS. In our algorithm the sequents subject to reduction may not be next to each other. The Reduce function needs to receive two indices in the tape to find the sequents for reduction. All reductions except those corresponding to $\otimes$ and $\multimap$ are non-branching(nb) and their transition rules are quite similar to the one introduced by FS.

- If $M_{n+1} = \mathsf{Reduce}_{nb}(M_n, i, j)$ then
  - $(n, k) \to^{\perp} (n+1, k)$ for $k \notin \{i, j\}$,
  - $(n, i) \to^{0} (n+1, i)$,
  - $(n, j) \to^{0} (n+1, j)$.

.

The reductions corresponding to $\otimes$ and $\multimap$, however, produce a branch and need to be defined separately:

- If $M_{n+1} = \mathsf{Reduce}_{\otimes}(M_n, i, j)$ then
  - $(n, k) \to^{\perp} (n+1, k)$ for $k < i$,
  - $(n, i) \to^{1} (n+1, i)$ and $(n, i) \to^{2} (n+1, i+1)$,
  - $(n, j) \to^{0} (n+1, j+1)$,
  - $(n, k) \to^{\perp} (n+1, k+1)$ for $i < k < j$ or $k > j$.

  .
- If $M_{n+1} = \mathsf{Reduce}_{\multimap}(M_n, i, j)$ then
  - $(n, k) \to^{\perp} (n+1, k)$ for $k < i$,
  - $(n, i) \to^{0} (n+1, j)$,
  - $(n, k) \to^{\perp} (n+1, k-1)$ for $i < k < j$,
  - $(n, j) \to^{1} (n+1, j-1)$ and $(n, j) \to^{2} (n+1, j+1)$,
  - $(n, k) \to^{\perp} (n+1, k+1)$ for $k < j$.

Transitions labelled by $\perp$ mean that the sequent has not evolved by a reduction rule, while other labels show that the sequent is evolved into one or two (in the case of branching rules) new sequents in the next tape. We get the real trace Ψ by collapsing the transitions labelled by $\perp$. Ψ is an infinite, finitely branching labelled tree with prefix order $\sqsubseteq$ and lexicographical order $<$. A branch in Ψ is a maximal path. The set of all branches of Ψ ordered lexicographically forms a complete lattice.

An infinite branch is a μ- branch (resp. ν-branch) if its corresponding derivation is a μ- trace (resp. ν-trace). By our validity condition Ψ satisfies the property that a ν-branch can only admit finitely many branches on its right side (it may include cuts, $\otimes$, or $\multimap$ reductions).

We prove the following three contradictory statements dual to FS:

(i) The greatest infinite branch of Ψ is a μ-branch:
The greatest infinite branch of Ψ exists by Konig's lemma and is either a μ- or a ν- branch. Assume it is a ν-branch. Then either it forms infinitely many branches on its right or there is an infinite branch greater than it. In both cases we can form a contradiction.

(ii) Let E be a nonempty collection of μ-branches. Then $\gamma = \bigwedge E$ is a μ-branch:
If $\gamma \in E$ then it is trivially true. Otherwise, by the way we constructed Ψ, it means that γ has infinitely many branches on its right and thus cannot be a ν branch.

(iii) If β is a μ-branch, then there exists another μ-branch $\beta' < \beta$:
β is a μ-branch so for infinitely many position variables $\mathbf{x1}^{\alpha_1}, \mathbf{x2}^{\alpha_2}, \cdots$ on the antecedents of β we can form an infinite chain of inequalities

$$\mathsf{snap}(\mathbf{x1}^{\alpha_1}) >_{\Omega_1^\beta} \mathsf{snap}(\mathbf{x2}^{\alpha_2}) >_{\Omega_2^\beta} \cdots .$$

There are two possibilities here:

(a) There is an infinite branch $\beta' < \beta$ with infinitely many position variables $\mathbf{xi}^{\alpha_i}, \mathbf{x\{i+1\}}^{\alpha_{i+1}}, \cdots$ as its succedents. Note that these position variables connect sequents in β to the sequents in β' infinitely many times. So every $\mu/\nu L$ rule in β reduces with a $\mu/\nu R$ rule in β'. This means that a μR rule with priority i is applied on the succedent of β' infinitely often but no priority $j < i$ has an infinitely many νR rule in β'.

(b) There is an infinite branch $\beta' < \beta$ with infinitely many branches on its right.

In both cases β' cannot be a ν-branch and thus is a μ-branch.

Items (i)-(iii) form a contradiction. We can form the nonempty collection E of all μ- branches in Ψ by (i). By (ii) we get $(\gamma = \bigwedge E) \in E$, which forms a contradiction with (iii). □

References

[1] Andreas Abel and Brigitte Pientka. 2016. Well-founded recursion with copatterns and sized types. *Journal of Functional Programming* 26 (2016).

[2] Andreas M Abel and Brigitte Pientka. 2013. Wellfounded recursion with copatterns: A unified approach to termination and productivity. In *ACM SIGPLAN Notices*, Vol. 48. ACM, 185–196.

[3] David Baelde, Amina Doumane, and Alexis Saurin. 2016. Infinitary proof theory: the multiplicative additive case. (2016).

[4] David Baelde and Dale Miller. 2007. Least and greatest fixed points in linear logic. In *International Conference on Logic for Programming Artificial Intelligence and Reasoning*. Springer, 92–106.

[5] P Nick Benton. 1994. A mixed linear and non-linear logic: Proofs, terms and models. In *International Workshop on Computer Science Logic*. Springer, 121–135.

[6] Michael Brandt and Fritz Henglein. 1998. Coinductive axiomatization of recursive type equality and subtyping. *Fundamenta Informaticae* 33, 4 (1998), 309–338.

[7] James Brotherston. 2005. Cyclic proofs for first-order logic with inductive definitions. In *International Conference on Automated Reasoning with Analytic Tableaux and Related Methods*. Springer, 78–92.

[8] Farzaneh Derakhshan and Frank Pfenning. 2019. Circular Proofs as Session-Typed Processes: A Local Validity Condition. *arXiv preprint arXiv:1908.01909* (2019).

[9] Amina Doumane. 2017. *On the infinitary proof theory of logics with fixed points*. Ph.D. Dissertation.

[10] Jérôme Fortier and Luigi Santocanale. 2013. Cuts for Circular Proofs: Semantics and Cut-Elimination. In *22nd Annual Conference on Computer Science Logic (CSL 2013)*, Simona Ronchi Della Rocca (Ed.). LIPIcs 23, Torino, Italy, 248–262.

[11] Claudio Hermida and Bart Jacobs. 1998. Structural induction and coinduction in a fibrational setting. *Information and computation* 145, 2 (1998), 107–152.

[12] Dexter Kozen and Alexandra Silva. 2017. Practical coinduction. *Mathematical Structures in Computer Science* 27, 7 (2017), 1132–1152.

[13] Milad Niqui and Jan Rutten. 2009. Coinductive Predicates as Final Coalgebras.. In *FICS*. 79–85.

[14] Frank Pfenning and Dennis Griffith. 2015. Polarized substructural session types. In *International Conference on Foundations of Software Science and Computation Structures*. Springer, 3–22.

[15] Benjamin C Pierce and C Benjamin. 2002. *Types and programming languages*. MIT press.

[16] John C. Reynolds. 1981. The Essence of Algol. In *Algorithmic Languages*, Jaco W. de Bakker and J. C. van Vliet (Eds.). North-Holland, Amsterdam, 345–372.

[17] Grigore Rosu. 2017. Matching Logic. *Logical Methods in Computer Science* 13 (2017).

[18] Luigi Santocanale. 2002. A calculus of circular proofs and its categorical semantics. In *International Conference on Foundations of Software Science and Computation Structures*. Springer, 357–371.

[19] Christoph Sprenger and Mads Dam. 2003. On the Structure of Inductive Reasoning: Circular and Tree-Shaped Proofs in the μCalculus. In *International Conference on Foundations of Software Science and Computation Structures*. Springer, 425–440.