Coalgebraic Semantics for Logic Programming via Saturation

No Author Given

No Institute Given

Abstract. TODO

1 Introduction

In [KMP10] Komendantskaya, McCusker and Power introduced a coalgebraic semantics for ground (i.e. variable-free) logic programs. Given a set At of ground atoms, a program $\mathbb{P}$ with atoms from At is encoded as a coalgebra $p\colon At \to \mathcal{P}_f\mathcal{P}_f(At)$ on **Set**, where $\mathcal{P}_f$ is the finite powerset functor. The idea is that each atom $A \in At$, now considered as a *goal*, is associated with the set $p(A)$ where each element corresponds to a clause $H\colon\!-\, B_1, \ldots, B_k$ of $\mathbb{P}$ whose head H *unifies with* A, i.e. (in the ground case) $H = A$. Each such element is itself a set, consisting of the atoms $B_1, \ldots, B_k$ in the body of the clause.
The main result of [KMP10] concerns the construction of a coalgebra $\overline{p}\colon At \to \mathbb{C}(\mathcal{P}_f\mathcal{P}_f)(At)$, where $\mathbb{C}(\mathcal{P}_f\mathcal{P}_f)$ is the *cofree comonad* on $\mathcal{P}_f\mathcal{P}_f$. Given an atomic goal $A \in At$, the object $\overline{p}(A) \in \mathbb{C}(\mathcal{P}_f\mathcal{P}_f)(At)$, obtained by iteratively applying the map p, can be depicted as the *and-or parallel derivation tree* for A [GC94,GPA+01,GBM+07].
This framework is extended in [KP11b] to arbitrary logic programs. Differently from the ground case, the presence of variables requires p to deal with *substitution instances* of atoms. On the other hand, one wants to maintain the primary feature of the ground case, which is the explicit and-or parallelism exhibited by the notion of derivation tree associated with the coalgebra p. Unfortunately, for arbitrary logic programs and-or parallel derivation trees are not guaranteed to represent sound derivations [GC94]. The problem lies in the presence of variable dependencies and the use of unification, which make derivations for logic programs inherently *sequential* processes [DKM84].
Komendantskaya and Power [KP11b] obviate to this difficulty by shaping a variant of and-or parallel derivation trees - called *coinductive forest* - where unification is restricted to the case of *term matching*. Contrary to unification, the term-matching algorithm is *parallelizable* [DKM84], meaning that the and-or parallelism exhibited by coinductive forests does not lead to unsound derivations. This comes at price of having a semantics which is not *compositional*, in the sense that, for some atom A and substitution θ,

$$\|A\theta\| \neq \|A\|\overline{\theta}$$

where $\|A\|$ is the coinductive forest associated to A, $A\theta$ denotes the result of applying θ to A and $\|A\|\overline{\theta}$ denotes the result of applying θ to each atom in $\|A\|$.
In this paper we propose a different approach to the semantics of arbitrary logic programs, where compositionality is achieved through *saturation* techniques [CIT,CIT]. In our terminology, saturating an atom means to take all its (also non-ground) substitution instances. The term-matching approach in [KP11b] only operates a saturation on the program side, by trying to match a given goal with all possible substitution instances of heads in the program. What we suggest is to saturate also on the goal side, and shift the term matching to a correspondence between substitution instances of the goal and substitution instances of heads in the program.

$$\begin{array}{ccc} \text{Atom A} & & \text{Heads in the program} \\ \Big\downarrow{\scriptstyle saturation} & & \Big\downarrow{\scriptstyle saturation} \\ \text{subst. instances of A} & \xleftrightarrow[\text{term-matching}]{} & \text{subst. instances of heads} \end{array} \tag{1}$$

- ▶ Other features and description of our proposal to be added at a later stage. In particular:
  - Motivation concerning the categorical formalization (e.g. $\mathbf{Set}^{\mathcal{L}_\Sigma^{op}}$ is a (co)complete category and allow use to rephrase the constructions of the ground case, whereas this is not possible in the category of locally ordered endofunctors and lax natural transformations considered by Power).
  - Remark that our saturated semantics yields a sound notion of derivation, by reduction to Power's semantics.

2 Coalgebraic Semantics for Logic Programming

In this section we recall the coalgebraic framework for logic programming, as introduced in [KMP10] for the ground case and in [KP11b,KP11a] for the general case.
For this purpose, first we fix some terminology and notation, mainly concerning category theory and logic programming. Given a category $\mathbf{C}$, we denote with $|\mathbf{C}|$ the category with the same objects as $\mathbf{C}$ but no arrows apart the identities. With a little abuse of notation, we also write $o \in |\mathbf{C}|$ to indicate that o is an object of $\mathbf{C}$. If $\mathbf{C}$ is small, the set of arrows from o_1 to o_2, with $o_1, o_2 \in |\mathbf{C}|$, is denoted with $\mathbf{C}[o_1, o_2]$. A $\mathbf{C}$-indexed presheaf is any functor $\mathcal{G}$ of type $\mathbf{C} \to \mathbf{Set}$. We indicate with $\mathbf{Set}^{\mathbf{C}}$ the category of $\mathbf{C}$-indexed presheaves and natural transformations between them. Given an endofunctor $\mathcal{F}\colon \mathbf{C} \to \mathbf{C}$, an $\mathcal{F}$-coalgebra on an object $o \in |\mathbf{C}|$ is any arrow $p\colon o \to \mathcal{F}(o)$. We denote with $\mathbf{CoAlg}(\mathcal{F})$ the category of $\mathcal{F}$-coalgebrae and coalgebra morphisms .

Fil: La notazioen deve essere uniforme. Ad esempio o sempre $\mathcal{KF}$ o sempre $\mathcal{K}(\mathcal{F})$.

Fil: Io userei F e G per i presheaves e non $\mathcal{F}$ e $\mathcal{G}$

Fil: Pure i predicati hanno una arieta': formano una segnatura. Pertanto e' impreciso dire che prendiamo un insieme di predicati.

Fab: Seguo la convenzione di Power di tenere distinta una segnatura Σ di

For our purposes a *signature* Σ is a set of function symbols $f, g, h, \ldots$ each equipped with a fixed arity. Given a countably infinite list $Var = \{x_1, x_2, x_3, \ldots\}$ of variables, we define the set $Ter(\Sigma)$ of *terms* over Σ and Var as expected. The (contravariant) *Lawvere Theory* associated with Σ and Var is a category $\mathcal{L}_\Sigma^{op}$ where objects are natural numbers, with $n \in |\mathcal{L}_\Sigma^{op}|$ representing variables $x_1, x_2, \ldots, x_n$ from Var. Given $n, m \in |\mathcal{L}_\Sigma^{op}|$, the set $\mathcal{L}_\Sigma^{op}[n, m]$ consists of all m-tuples $\langle t_1, \ldots, t_m \rangle$ of terms from $Ter(\Sigma)$ where only variables among $x_1, x_2, \ldots, x_n$ occur. We call *substitutions* the arrows of $\mathcal{L}_\Sigma^{op}$ and use Greek letters $\theta, \sigma, \tau, \ldots$ to denote them.

An *alphabet* $\mathcal{A}$ consists of a signature Σ, a list of variables $Var = \{x_1, x_2, x_3, \ldots\}$ and a set of predicate symbols $P, P_1, P_2, \ldots$ each assigned an arity. Given a predicate symbol P of arity n and terms $t_1, \ldots, t_n$, $P(t_1, \ldots, t_n)$ is a *formula* (also called an *atom*). We use Latin capital letters $A, B, \ldots$ for atoms. An atom $A = P(t_1, \ldots, t_n)$ is *ground* if each t_i does not contain any variable. The set At consists of all atoms constructed from the symbols of the alphabet $\mathcal{A}$. Given a substitution $\theta = \langle t_1, \ldots, t_m \rangle \colon n \to m$ and an atom A with variables among $x_1, \ldots, x_n$, we adopt the standard notation of logic programming in denoting with $A\theta$ the atom obtained by replacing x_i with t_i in A, for each $i \leq m$. Also we use the notation $\{A_1, \ldots, A_n\}\theta$ as a shorthand for $\{A_1\theta, \ldots, A_n\theta\}$.

A *logic program* $\mathbb{P}$ based on alphabet $\mathcal{A}$ consists of a finite set of clauses C noted as $H \colon\!-\, B_1, \ldots, B_k$, with $k < \omega$. Both H and $B_1, \ldots, B_k$ are atoms from At, where H is called the *head* of C and $B_1, \ldots, B_k$ form the *body* of C. We say that $\mathbb{P}$ is *ground* if all clauses only contain ground atoms.

- How much detail in the preliminaries of logic programming?
- Think about some fixed terminology/notation on trees. To be defined: depth of a tree.

2.1 The Ground Case

We recall the modeling of variable-free programs through coalgebrae, as first presented in [KMP10]. For the sequel we fix an alphabet $\mathcal{A}$, a set At of ground atoms based on $\mathcal{A}$ and a ground logic program $\mathbb{P}$ with clauses given by atoms from At. The behavior represented by $\mathbb{P}$ is captured by a coalgebra $p \colon At \to \mathcal{P}_f\mathcal{P}_f(At)$ on **Set**, where $\mathcal{P}_f$ is the finite powerset functor and p defined as follows:

$$p \colon\ A \ \mapsto\ \{\{B_1, \ldots, B_n\} \mid H \colon\!-\, B_1, \ldots, B_n \text{ is a clause of } \mathbb{P} \text{ and } A = H\}. \tag{2}$$

The idea is that each atom $A \in At$ is associated with the set $p(A)$ where each element corresponds to a clause $H \colon\!-\, B_1, \ldots, B_k$ of $\mathbb{P}$ whose head H *unifies with* A, i.e. (in the ground case) $H = A$. Each such element is itself a set, consisting of the atoms $B_1, \ldots, B_k$ in the body of the clause.

Example 1. A program $\mathbb{P}$, the associated coalgebra and the value on an atom A.

Fab: Personalmente mi piace una spiegazione 'lenta' e chiara del caso ground, ma il limite di pagine con ogni probabilit imporr di accorciare, e questa sezione mi pare un buon punto dove puoi provare a togliere particolari.

The set $p(A)$ of example 1 can be displayed as a finite tree with two sorts of nodes, occurring at alternated depths. At depth 0 and 2 we have *and-nodes*, which are labeled with atoms from At. At depth 1 we have *or-nodes*, which corresponds to clauses whose head matches a given goal and remain unlabeled.
The "boolean" terminology for the two sorts of nodes comes from an operational reading of this tree: the root is labeled with the atomic goal A which one wants to refute in the program $\mathbb{P}$; each and-node represents an atomic subgoal which has to be refuted in order to refute A; each or-node (with an and-node, say labeled with B, as parent) represents an alternative way to try to refute B.
Each depth of the tree represents a stage in this computation: the presence of multiple nodes at the same depth provides an explicit representation of *and-parallelism* (depths where and-nodes occur) and *or-parallelism* (depths where or-nodes occur) in our derivation strategy for the goal A. These are two main ways in which parallelism arises in logic programming, corresponding respectively to simultaneous proof-search of multiple atomic goals and parallel exploration of multiple attempts to refute the same goal [GC94]. As mentioned in the introduction, and-or parallel implementations of logic programming have a standard representation of computations as *parallel and-or derivation trees.*

Definition 1. *Let $\mathcal{A}$ be an alphabet. Given a logic program $\mathbb{P}$ and an atom A based on $\mathcal{A}$, the and-or parallel derivation tree for A in $\mathbb{P}$ is the possibly infinite tree T satisfying the following properties:*

1. *Each node in T is either an and-node or an or-node.*
2. *Each or-node is labeled with $\bullet$.*
3. *Each and-node is labeled with one atom on the alphabet $\mathcal{A}$.*
4. *The root of T is an and-node labeled with A.*
5. *For every and-node s in T, let A' be its label. If A' is unifiable with exactly m distinct clauses $C_1, \ldots, C_m$ in $\mathbb{P}$ via mgu's $\theta_1, \ldots, \theta_m$, then s has exactly m children given by or-nodes, such that, for every $i \in m$, if $C_i = H^i :- B^i_1, \ldots, B^i_k$, then the ith or-node has k and-nodes $t_1, \ldots, t_k$ as children, where t_j is labeled with $B^i_j\theta$ for $1 \leq j \leq k$.*

Fab: Q: Is the associated coalgebra a natural transformation in the non-ground case?

In the ground case considered here, the mgu's $\theta_1, \ldots, \theta_m$ of the above definition are always identities. Given a goal A, the object $p(A) \in \mathcal{P}_f\mathcal{P}_f(At)$ represents the prefix of depth 2 of the parallel and-or derivation tree of A in $\mathbb{P}$. The full tree for A is recovered as an element of $\mathbb{C}(\mathcal{P}_f\mathcal{P}_f)(At)$, where $\mathbb{C}(\mathcal{P}_f\mathcal{P}_f)$ is the *cofree comonad* on $\mathcal{P}_f\mathcal{P}_f$, standardly provided by the following construction [Wor99].

Construction 1 *We first construct the* cofree $\mathcal{P}_f\mathcal{P}_f$-coalgebra *on At. In this aim, consider the terminal sequence for the functor $At \times \mathcal{P}_f\mathcal{P}_f(-)\colon \mathbf{Set} \to \mathbf{Set}$, which consists of sequences of objects X_α and arrows $\delta_\alpha\colon X_{\alpha+1} \to X_\alpha$, defined by induction on α as follows.*

$$X_\alpha := \begin{cases} At & \alpha = 0 \\ At \times \mathcal{P}_f\mathcal{P}_f(X_\beta) & \alpha \text{ is a successor ordinal } \beta + 1 \end{cases}$$

$$\delta_\alpha := \begin{cases} \pi_1 & \alpha = 0 \\ Id_{At} \times \mathcal{P}_f\mathcal{P}_f(\delta_\beta) & \alpha \text{ is a successor ordinal } \beta + 1 \end{cases}$$

For α a limit ordinal, X_α is given as a limit of the sequence and a function $\delta_\alpha \colon X_\alpha \to X_\beta$ is given for each $\beta < \alpha$ by universal mapping property of X_α.
Both $\mathcal{P}_c$ and $\mathcal{P}_f$ are readily seen to be mono-preserving accessible functors. Then by theorem [Wor99, Th. 7] we know that the sequence given above converges to a limit X_γ such that $X_\gamma \cong X_{\gamma+1}$. Since $X_{\gamma+1}$ is defined as $At \times \mathcal{P}_f\mathcal{P}_f X_\gamma$, there is a projection function $\pi_2 \colon X_{\gamma+1} \to \mathcal{P}_f\mathcal{P}_f X_\gamma$ which makes $\pi_2 \circ \delta_\gamma^{-1} \colon X_\gamma \to \mathcal{P}_f\mathcal{P}_f X_\gamma$ the cofree $\mathcal{P}_f\mathcal{P}_f$-coalgebra *on At. Since $\mathcal{P}_f\mathcal{P}_f$ is accessible, the functor $\mathcal{R} \colon$ **Set** $\to$ **CoAlg**$(\mathcal{P}_f\mathcal{P}_f)$ sending a set to its cofree $\mathcal{P}_f\mathcal{P}_f$-coalgebra is right adjoint to the forgetful functor $\mathcal{V} \colon$ **CoAlg**$(\mathcal{P}_f\mathcal{P}_f) \to$ **Set**. Then $\mathcal{V}\mathcal{R} \colon$ **Set** $\to$ **Set** is the* cofree comonad *on $\mathcal{P}_f\mathcal{P}_f$, which we denote with $\mathbb{C}(\mathcal{P}_f\mathcal{P}_f)$.*

As the elements of the cofree comonad on $\mathcal{P}_f$ are standardly presented as finitely branching trees [Wor99], one can depict elements of $\mathbb{C}(\mathcal{P}_f\mathcal{P}_f)(At)$ as finitely branching trees with two sorts of nodes occurring at alternated depth (the $\mathcal{P}_f\mathcal{P}_f(-)$ component of the functor) and where one of the sorts is labeled (the $At \times -$ component). We now define a $\mathbb{C}(\mathcal{P}_f\mathcal{P}_f)$-coalgebra $\overline{p}$ structure on At.

Construction 2 *Given the ground program $\mathbb{P}$ based on atoms from At, let $p \colon At \to \mathcal{P}_f\mathcal{P}_f(At)$ be the coalgebra associated with $\mathbb{P}$. We define a cone $\{p_\alpha \colon At \to X_\alpha\}_{\alpha<\gamma}$ on the terminal sequence of construction 1 as follows:*

$$p_\alpha := \begin{cases} Id_{At} & \alpha = 0 \\ Id_{At} \times ((\mathcal{P}_f\mathcal{P}_f(p_\beta) \circ p) & \alpha \text{ is a successor ordinal } \beta + 1 \end{cases}$$

For α a limit ordinal, a function $p_\alpha \colon At \to X_\alpha$ is provided by the universal mapping property of the limit X_α. Then in particular $X_\gamma = \mathbb{C}(\mathcal{P}_f\mathcal{P}_f)(At)$ yields a function $\overline{p} \colon At \to \mathbb{C}(\mathcal{P}_f\mathcal{P}_f)(At)$.

Example 2. Same A, $\mathbb{P}$ of the previous example. Representation of $\overline{p}(A)$ as a tree.

Given an atomic goal $A \in At$, the tree $\overline{p}(A) \in \mathbb{C}(\mathcal{P}_f\mathcal{P}_f)(At)$ is built by iteratively applying the map p, first to A, then to each atom in $p(A)$, and so on. For each $m < \omega$, the arrow p_m can be described as the mapping of A into its parallel and-or derivation tree up to depth m. Then the limit $\overline{p}$ of all such approximations provides the full parallel and-or derivation tree of A, as stated by the following proposition.

Proposition 1 ([KMP10]). *Given an alphabet $\mathcal{A}$ and a ground logic program $\mathbb{P}$ based on $\mathcal{A}$, let $\overline{p}$ be defined from $\mathbb{P}$ according to construction 2. Then for every atom A the value $\overline{p}(A)$ represents the parallel and-or derivation tree of A in $\mathbb{P}$.*

2.2 The General Case

We recall the extension of the coalgebraic semantics to arbitrary (i.e. possibly with variables) logic programs, as introduced in [KP11b,KP11a]. As mentioned in the introduction, a natural way to do that would be to generalize the definition of coalgebrae p and $\overline{p}$ in such a way that the associated notion of computation is represented by parallel and-or derivation trees (definition 1). Unfortunately, in presence of variables these trees are not guaranteed to represent sound derivations. The problem lies in the presence of variable dependencies and the use of unification, which make derivations for logic programs inherently *sequential* processes [DKM84].

Example 3. Unsound and-or tree: the LIST program and the goal $List(cons(x, cons(y, x)))$.

In [KP11b] Komendantskaya and Power obviate to this difficulty by shaping a variant of and-or trees - called *coinductive forests* - where unification is restricted to the case of *term matching*.

Definition 2. *Let $\mathcal{A}$ be an alphabet. Fix a logic program $\mathbb{P}$, an atom A based on $\mathcal{A}$ and a number $k \leq \omega$. The* k-coinductive forest *for A in $\mathbb{P}$ is the possibly infinite tree T satisfying properties 1-4 of definition 1 and property 5 replaced by the following:*

- *for every and-node s in T, let A' be its label. Children of s are or-nodes in 1-1 correspondence with clauses $C = H :- B_1, \ldots, B_n$ of $\mathbb{P}$ such that $A' = H\theta$ for some substitution θ and $B_1\theta, B_n\theta$ have variables among $x_1, \ldots, x_k$. For each such child t, let $C = H :- B_1, \ldots, B_n$ and θ respectively the clause and the substitution associated with t. Then t has exactly n and-nodes $t_1, \ldots, t_n$ as children, where t_j is labeled with $B_j\theta$ for $1 \leq j \leq n$.*

Fab: Non ho gi messo questi remark perch non sono ancora sicuro che questa sia la nozione pi 'giusta' da utilizzare nell'articolo, tra quelle introdotte da Power.

▶ Possible remarks/footnote to include:
- why do we parameterize the definition to k? The reader will wonder about that.
- Our coinductive forests are not sets of coinductive trees as defined in [KP11a] but they 'glue' them together.

Contrary to unification, the term-matching algorithm is *parallelizable* [DKM84], meaning that the explicit and-or parallelism exhibited by forests does not lead to unsound derivations [KP11a, Th. 4.8]. The intuitive reason is that, at each stage of the computation, one considers only the substitutions that do not apply to the current goal, meaning that also the "previous history" of the computation remains uncorrupted.
We now recall from [KP11b] the categorical formalization of this class of trees. First let us fix a signature Σ, a list $Var = \{x_1, x_2, \ldots\}$ of variables and an alphabet $\mathcal{A}$ on Σ and Var. A collection of atoms (for which we overload the notation used in the base case) based on $\mathcal{A}$ is modeled as a presheaf $At\colon \mathcal{L}_\Sigma^{op} \to \mathbf{Set}$. The index category is the (contravariant) *Lawvere Theory* $\mathcal{L}_\Sigma^{op}$ associated with Σ, as defined in the preliminaries. For each natural number $n \in |\mathcal{L}_\Sigma^{op}|$, $At(n)$ is defined as the set of atoms with variables among $x_1, \ldots, x_n$. Given an arrow $\theta \in \mathcal{L}_\Sigma^{op}(n, m)$, the

function $At(\langle t_1, \ldots, t_n \rangle)\colon At(n) \to At(m)$ is defined by substitution, i.e. $At(\theta)(A) := A\theta$.
The next step is to generalize the definition of the coalgebra p associated to a logic program $\mathbb{P}$. In this aim, recall from definition 2 how p should act on an atom A, for a fixed $k < \omega$:

$$A \mapsto \{\{B_1, \ldots, B_j\}\theta \mid H\colon\!- B_1, \ldots, B_j \text{ is a clause of } \mathbb{P}, \\ A = H\theta \text{ and} \\ B_1\theta, \ldots, B_j\theta \in At(k)\}. \quad (3)$$

For each clause $H\!:\!-B_1, \ldots, B_k$, there might be infinitely (but countably) many substitutions θ such that $A = H\theta$. Thus the object on the right-hand side of (3) will be associated with the functor $\mathcal{P}_c\mathcal{P}_f\colon \mathbf{Set} \to \mathbf{Set}$, where $\mathcal{P}_c$ and $\mathcal{P}_f$ are respectively the countable powerset functor and the finite powerset functor. In order to formalize this as a coalgebra on $At\colon \mathcal{L}_\Sigma^{op} \to \mathbf{Set}$, we need to lift the analysis from $\mathbf{Set}$ to $\mathbf{Set}^{\mathcal{L}_\Sigma^{op}}$. For this purpose, we consider liftings $\widehat{\mathcal{P}_c}\colon \mathbf{Set}^{\mathcal{L}_\Sigma^{op}} \to \mathbf{Set}^{\mathcal{L}_\Sigma^{op}}$ and $\widehat{\mathcal{P}_f}\colon \mathbf{Set}^{\mathcal{L}_\Sigma^{op}} \to \mathbf{Set}^{\mathcal{L}_\Sigma^{op}}$, standardly defined on presheaves $\mathcal{F}\colon \mathcal{L}_\Sigma^{op} \to \mathbf{Set}$ by precomposition respectively with $\mathcal{P}_c$ and $\mathcal{P}_f$. Then one would like to fix (3) as the definition of the k-component $p(k)$ of a natural transformation $p\colon At \to \widehat{\mathcal{P}_c}\widehat{\mathcal{P}_f}(At)$. The key problem with this formulation is that p would *not* be a natural transformation, as shown by the following example.

Example 4. The LIST program, the two goals $List(x)$, $List(Nil)$ and the substitution $x \mapsto Nil$.

In [KP11b] the authors overcome this difficulty by relaxing the naturality requirement. The presheaf At is re-defined as a functor $\widetilde{At}\colon \mathcal{L}_\Sigma^{op} \to Poset$ and p as a *lax natural transformation* $\widetilde{p}\colon \widetilde{At} \to \widetilde{\mathcal{P}_c}\widetilde{\mathcal{P}_c}(At)$, where $\widetilde{\mathcal{P}_c}\widetilde{\mathcal{P}_f}\colon Lax(\mathcal{L}_\Sigma^{op}, Poset) \to Lax(\mathcal{L}_\Sigma^{op}, Poset)$ is the extension of $\widehat{\mathcal{P}_c}\widehat{\mathcal{P}_f}\colon \mathbf{Set}^{\mathcal{L}_\Sigma^{op}} \to \mathbf{Set}^{\mathcal{L}_\Sigma^{op}}$ to $Poset$-valued functors. Then $\widetilde{p}$ becomes a $\widetilde{\mathcal{P}_c}\widetilde{\mathcal{P}_c}$-coalgebra in the category $Lax(\mathcal{L}_\Sigma^{op}, Poset)$ of locally ordered functors $\mathcal{F}\!:\!\mathcal{L}_\Sigma^{op} \to Poset$ and lax natural transformations.
We refer to [KP11b, Ch.4] for further details. The lax approach allows the coalgebra $\widetilde{p}$ to be defined according to (3) on each component $n \in |\mathcal{L}_\Sigma^{op}|$. However, it has also introduce several shortcomings. Unlike the categories $\mathbf{Set}$ and $\mathbf{Set}^{\mathcal{L}_\Sigma^{op}}$, $Lax(\mathcal{L}_\Sigma^{op}, Poset)$ is neither complete nor cocomplete, meaning that a cofree comonad on $\widetilde{\mathcal{P}_c}\widetilde{\mathcal{P}_c}$ cannot be retrieved through the standard constructions 1 and 2 that were used in the ground case. Moreover, the category of $\widetilde{\mathcal{P}_c}\widetilde{\mathcal{P}_c}$-coalgebrae becomes problematic, because the commutativity property of coalgebra maps does not cohere well with the laxness of arrows in $Lax(\mathcal{L}_\Sigma^{op}, Poset)$. These two issues force the formalization of non-ground logic program to use quite different (and more sophisticated) categorical tools than the ones employed for the ground case.
Even more significantly, the lax coalgebra map $\widetilde{p}^\sharp$ induces a semantics which is not *compositional*. The same square of example 4 still does not commute in the lax setting, meaning that the actions of applying a substitution and making a derivation step produce different results according to the order in which we perform them.

Fab: unnatural, sophisticated...choose the appropriate connotation.

- Insert further motivations? Cfr. also the end of the Introduction
- Perhaps we could state the following Adequacy theorem: n-coinductive forest for $A \in At(n) = \overline{p}(n)(A)$. Depends if it is useful/essential...

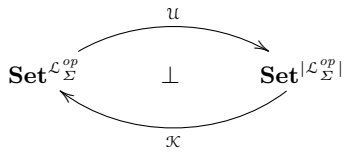
3 A Saturated Semantics for Logic Programming

Fab: Decidere se le notazioni At, p, $\overline{p}$ vengono sovrascritte dal caso ground a quello generale, oppure ne usiamo di nuove. Per ora sono sovrascritte. Genera troppa confusione?

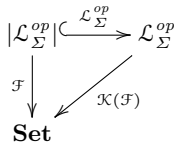
Motivated by the observations of the previous section, we propose a different approach to the semantics of non-ground programs, based on *saturation* techniques [CIT,CIT].

- More introduction/intuition on saturation, depending on what is already explained in the introduction.

For such purpose, we consider the following adjunction between presheaf categories.



The left adjoint $\mathcal{U}$ is the forgetful functor, given by precomposition with the inclusion functor $\iota\colon |\mathcal{L}_\Sigma^{op}| \hookrightarrow \mathcal{L}_\Sigma^{op}$. As shown in [ML98, X, Th. 1, Cor. 2], $\mathcal{U}$ has a right adjoint $\mathcal{K}\colon \mathbf{Set}^{|\mathcal{L}_\Sigma^{op}|} \to \mathbf{Set}^{\mathcal{L}_\Sigma^{op}}$ sending each object of $\mathbf{Set}^{|\mathcal{L}_\Sigma^{op}|}$ to its *Right Kan Extension* along ι.



Given any presheaf $\mathcal{F}\colon |\mathcal{L}_\Sigma^{op}| \to \mathbf{Set}$, the functor $\mathcal{K}(\mathcal{F})\colon \mathcal{L}_\Sigma^{op} \to \mathbf{Set}$ is defined as follows on objects $n \in |\mathcal{L}_\Sigma^{op}|$ and arrows $\theta \in \mathcal{L}_\Sigma^{op}[n,m]$:

$$\begin{aligned} \mathcal{K}(\mathcal{F})(n) &:= \prod_{\theta \in \mathcal{L}_\Sigma^{op}[n,m]} \mathcal{F}(m) \\ \mathcal{K}(\mathcal{F})(\theta) &: \langle x_\tau \rangle_{\tau \in \mathcal{L}_\Sigma^{op}[n,m']} \mapsto \langle (x_{\sigma\circ\theta})_\sigma \rangle_{\sigma \in \mathcal{L}_\Sigma^{op}[m,m'']} \end{aligned}$$

Fab: Qualche idea migliore per la notazione delle tuple?

where $\langle x_\tau \rangle_{\tau \in \mathcal{L}_\Sigma^{op}[n,m']}$ is the notation for a tuple in $\prod_{\tau \in \mathcal{L}_\Sigma^{op}[n,m']} \mathcal{F}(m')$, with x_τ the element indexed by τ. The tuple $\langle (x_{\sigma\circ\theta})_\sigma \rangle_{\sigma \in \mathcal{L}_\Sigma^{op}[m,m']}$ can be intuitively read as follows: for each $\sigma \in \mathcal{L}_\Sigma^{op}[m,m'']$, we let the element indexed by σ to be the one which was indexed by $\sigma \circ \theta \in \mathcal{L}_\Sigma^{op}[n,m'']$ in the input tuple $\langle x_\tau \rangle_{\tau \in \mathcal{L}_\Sigma^{op}[n,m']}$.

Next we consider the instantiation of the unit of the adjunction to the presheaf $At\colon \mathcal{L}_\Sigma^{op} \to \mathbf{Set}$, as defined in the previous section. This is a natural transformation $\eta\colon At \to \mathcal{KU}(At)$ given as follows:

$$\begin{aligned}\eta_{At}(n)\colon \mathcal{U}At(n) &\to \prod_{\theta \in \mathcal{L}_\Sigma^{op}[n,m]} \mathcal{U}At(m)\\ A &\mapsto \langle \big(\mathcal{U}At(\theta)(A)\big)_\theta \rangle_{\theta \in \mathcal{L}_\Sigma^{op}[n,m]}. \end{aligned} \tag{4}$$

The set $\eta_{At}(n)$ represents the *saturation* of the goal A, i.e. it consists of all substitution instances $\mathcal{U}At(\theta)(A) = A\theta$ of A, each indexed by the corresponding $\theta \in \mathcal{L}_\Sigma^{op}[n,m]$.
We now include in the picture the saturation on the program side. For such purpose, consider the restriction of $\widehat{\mathcal{P}_c}\widehat{\mathcal{P}_f}\colon \mathbf{Set}^{\mathcal{L}_\Sigma^{op}} \to \mathbf{Set}^{\mathcal{L}_\Sigma^{op}}$ to an endofunctor in $\mathbf{Set}^{|\mathcal{L}_\Sigma^{op}|}$. When the context is unambiguous, this will be also denoted with $\widehat{\mathcal{P}_c}\widehat{\mathcal{P}_f}$, the difference between $\mathcal{L}_\Sigma^{op}$ and $|\mathcal{L}_\Sigma^{op}|$ being just that the latter has only identities as arrows. Fixed a logic program $\mathbb{P}$, define a natural transformation $p^\flat\colon \mathcal{U}At \to \widehat{\mathcal{P}_c}\widehat{\mathcal{P}_f}(\mathcal{U}At)$ in $\mathbf{Set}^{|\mathcal{L}_\Sigma^{op}|}$ as follows:

$$\begin{aligned}p^\flat(n)\colon At(n) &\to \widehat{\mathcal{P}_c}\widehat{\mathcal{P}_f}(\mathcal{U}At)(n)\\ A &\mapsto \{\{B_1,\dots,B_k\}\theta \mid H\colon\!-B_1,\dots,B_k \text{ is a clause of } \mathbb{P},\\ &\qquad\qquad A = H\theta \text{ and}\\ &\qquad\qquad B_1\theta,\dots,B_k\theta \in At(n)\}. \end{aligned} \tag{5}$$

Observe that $p^\flat$ is defined by term-matching according to equation (3), the only difference being that we consider the presheaf $\mathcal{U}At \in \mathbf{Set}^{|\mathcal{L}_\Sigma^{op}|}$ instead of $At \in \mathbf{Set}^{\mathcal{L}_\Sigma^{op}}$. As observed in the previous section, $p^\flat$ would fail to be a natural transformation if directly defined on At in $\mathbf{Set}^{\mathcal{L}_\Sigma^{op}}$. However, the naturality requirement is trivially satisfied in $\mathbf{Set}^{|\mathcal{L}_\Sigma^{op}|}$, because there is no arrow to test in $|\mathcal{L}_\Sigma^{op}|$ apart the identities.
We can now recover a natural transformation in $\mathbf{Set}^{\mathcal{L}_\Sigma^{op}}$ as follows:

$$At \xrightarrow{\eta_{At}} \mathcal{KU}(At) \xrightarrow{\mathcal{K}(p^\flat)} \mathcal{K}\widehat{\mathcal{P}_c}\widehat{\mathcal{P}_f}\mathcal{U}(At). \tag{6}$$

The natural transformation $\mathcal{K}(p^\flat)$ in the equation (6) is defined as an 'indexwise' application of $p^\flat$ on tuples from $\mathcal{KU}(At)$, that is:

$$\begin{aligned}\mathcal{K}(p^\flat)(n)\colon \mathcal{KU}(At)(n) &\to \mathcal{K}\widehat{\mathcal{P}_c}\widehat{\mathcal{P}_f}\mathcal{U}(At)(n)\\ \langle A_\theta \rangle_{\theta \in \mathcal{L}_\Sigma^{op}[n,m]} &\mapsto \langle \big(p^\flat(m)(A_\theta)\big)_\theta \rangle_{\theta \in \mathcal{L}_\Sigma^{op}[n,m]}\end{aligned}$$

Intuitively, $\mathcal{K}(p^\flat)(n)$ models term matching between the substitution instances of a goal $A \in At(n)$ and the substitution instances of clauses in the program (the action of $p^\flat$). The former are recovered from A through the natural transformation η_{At}. The resulting composition $\mathcal{K}(p^\flat) \circ \eta_{At}\colon At \to \mathcal{K}\widehat{\mathcal{P}_c}\widehat{\mathcal{P}_f}\mathcal{U}(At)$ is a coalgebra for the functor $\mathcal{K}\widehat{\mathcal{P}_c}\widehat{\mathcal{P}_f}\mathcal{U}\colon \mathbf{Set}^{\mathcal{L}_\Sigma^{op}} \to \mathbf{Set}^{\mathcal{L}_\Sigma^{op}}$. We overload the notation of the ground case by denoting with p the coalgebra $\mathcal{K}(p^\flat) \circ \eta_{At}$ and we also abbreviate $\mathcal{K}\widehat{\mathcal{P}_c}\widehat{\mathcal{P}_f}\mathcal{U}$ with $\mathcal{S}$. The idea is to let $p\colon At \to \mathcal{S}(At)$ encode our logic

program $\mathbb{P}$, with $\mathcal{S}$ playing the same role of $\mathcal{P}_f\mathcal{P}_f$ in the ground case. In order to show how this meets the specification given in (1), we unfold the meaning of p in terms of its components η_{At} and $p^\flat$:

$$\begin{aligned} p(n)\colon At(n) &\to \mathcal{S}(At)(n) \\ A &\mapsto \langle \{\{B_1, \ldots, B_k\}\tau \mid H\colon\!- B_1, \ldots, B_k \text{ is a clause of } \mathbb{P}, \\ &\qquad\qquad A\theta = H\tau \text{ and} \\ &\qquad\qquad B_1\tau, \ldots, B_k\tau \in At(m)\}\rangle_{\theta \in \mathcal{L}_\Sigma^{op}[n,m]}. \end{aligned} \tag{7}$$

Observe that p is indeed an arrow of $\mathbf{Set}^{\mathcal{L}_\Sigma^{op}}$: it satisfies the naturality requirement, since both $\mathcal{K}(p^\flat)$ and η_{At} are natural transformations. In this way, we achieve that property of 'commuting with substitutions' that was precluded by the term-matching approach, as shown by the following rephrasing of example 4.

Example 5. Example 4 rephrased and eventually commuting.

Another benefit of saturated semantics is that the coalgebra $p\colon At \to \mathcal{S}(At)$ 'lives' in a (co)complete category which behaves (pointwise) as **Set**. This allows to follow the same steps of the ground case, explicitly constructing a a coalgebra for the cofree comonad $\mathbb{C}(\mathcal{S})$ as a straightforward generalization of constructions 1 and 2. In this aim, we first need to show the following property.

Proposition 2. *The terminal sequence for $At \times \mathcal{S}(-)$ converges to a terminal coalgebra.*

Proof. By [Wor99, Th. 7], it suffices to prove that $\mathcal{S}$ is an accessible mono-preserving functor. Since these properties are preserved by composition, we show them separately for each component of $\mathcal{S}$:

- $\mathcal{K}$ and $\mathcal{U}$ are a pair of adjoint functors between accessible categories, whence they are accessible themselves ([AR94, Prop. 2.23]). Moreover, they are both right adjoints: in particular, $\mathcal{U}$ is right adjoint to the left Kan extension along $i\colon |\mathbf{C}| \hookrightarrow \mathbf{C}$. It follows that both preserve limits, whence they preserve monos.
- The functors $\widehat{\mathcal{P}_c}\colon \mathbf{Set}^{|\mathcal{L}_\Sigma^{op}|} \to \mathbf{Set}^{|\mathcal{L}_\Sigma^{op}|}$ and $\widehat{\mathcal{P}_f}\colon \mathbf{Set}^{|\mathcal{L}_\Sigma^{op}|} \to \mathbf{Set}^{|\mathcal{L}_\Sigma^{op}|}$ are defined as liftings from **Set** to $\mathbf{Set}^{|\mathcal{L}_\Sigma^{op}|}$ of $\mathcal{P}_c\colon \mathbf{Set} \to \mathbf{Set}$ and $\mathcal{P}_f\colon \mathbf{Set} \to \mathbf{Set}$. As observed in construction 1, $\mathcal{P}_c$ and $\mathcal{P}_f$ are both accessible functors on **Set** which also preserve pullbacks. Since (co)limits in presheaf categories are computed objectwise, this means that also $\widehat{\mathcal{P}_c}$ and $\widehat{\mathcal{P}_f}$ have these properties. In particular, preservation of pullbacks implies preservation of monos.

Construction 3 *The terminal sequence for the functor $At \times \mathcal{S}(-)\colon \mathbf{Set}^{\mathcal{L}_\Sigma^{op}} \to \mathbf{Set}^{\mathcal{L}_\Sigma^{op}}$ consists of a sequence of objects X_α and arrows $\delta_\alpha\colon X_{\alpha+1} \to X_\alpha$, which are defined just as in construction 1, with $\mathcal{S}$ replacing $\mathcal{P}_f\mathcal{P}_f$.*
By proposition 2, this sequence converges to a limit X_γ such that $X_\gamma \cong X_{\gamma+1}$ and X_γ is the carrier of the cofree $\mathcal{S}$-coalgebra on At.

By accessibility of $\mathcal{S}$, the *cofree comonad* $\mathbb{C}(\mathcal{S})$ on $\mathcal{S}$ exists and maps At into X_γ given as in construction 3. A $\mathbb{C}(\mathcal{S})$-coalgebra $\overline{p}\colon At \to X_\gamma$ (for which we overload the notation used in the ground case) is defined as follows.

Construction 4 *Let $\{p_\alpha\colon At \to X_\alpha\}_{\alpha<\gamma}$ be a cone on the terminal sequence of construction 3, defined as in construction 2 with $\mathcal{S}$ replacing $\mathcal{P}_f\mathcal{P}_f$ and p defined according to* (7)*. Analogously to the ground case, $X_\gamma = \mathbb{C}(\mathcal{S})(At)$ yields a function $\overline{p}\colon At \to X_\gamma$ which is a $\mathbb{C}(\mathcal{S})$-coalgebra structure on At.*

As in the ground case, the coalgebra $\overline{p}$ is constructed as an iterative application of p: we call *saturated derivation tree* the associated notion of computation, which is defined as follows.

Definition 3. *Let $\mathcal{A}$ be an alphabet. Fix a logic program $\mathbb{P}$ and an atom A based on $\mathcal{A}$. The* saturated derivation tree *for A in $\mathbb{P}$ is the possibly infinite tree T satisfying properties 1-4 of definition 1 and property 5 replaced by the following:*

- *for every and-node s in T, let A' be its label. Children of s are or-nodes in 1-1 correspondence with clauses $C = H\text{:-}B_1,\ldots,B_n$ of $\mathbb{P}$ such that $A'\theta = H\tau$ for some substitution τ, θ. For each such child t, let $C = H\text{:-}B_1,\ldots,B_n$, θ and τ respectively the clause and the substitutions associated with t. Then t has exactly n and-nodes $t_1,\ldots,t_n$ as children, where t_j is labeled with $B_j\tau$ for $1 \leq j \leq n$.*

We can now state an adequacy theorem, in analogy with the one provided for the ground case (proposition 1).

Theorem 1. *Given an alphabet $\mathcal{A}$ and a logic program $\mathbb{P}$ based on $\mathcal{A}$, let $\overline{p}$ be defined from $\mathbb{P}$ according to construction 4. Then, for every $n < \omega$ and $A \in At(n)$, the value $\overline{p}(n)(A)$ represents the saturated derivation tree of A in $\mathbb{P}$.*

- If true, we could remark that our adequacy theorem (contrary to the one in [KP11a, Th. 4.5] for coinductive forests) does not need to bound the breadth, by effect of $\mathcal{K}$.

The commutativity property of the map p, as described in example 5, can be lifted to a commutativity property of $\overline{p}$, in virtue of its definition in terms of p. This allows us to state the following proposition.

Theorem 2 (Compositionality). *Given an atomic goal $A \in At(n)$, let $\|\cdot\|$ denote the map associating to A its saturated derivation tree $\overline{p}(n)(A)$. Let $\theta \in \mathcal{L}_\Sigma^{op}[n,m]$ be a substitution. Then we have the following:*

$$\|A\theta\| = \|A\|\overline{\theta}$$

where the object $\|A\|\overline{\theta}$ on the right-hand side is a shorthand for the application of $\mathbb{C}(\mathcal{S})(At)(\theta)\colon \mathbb{C}(\mathcal{S})(At)(n) \to \mathbb{C}(\mathcal{S})(At)(m)$ to $\|A\| \in \mathbb{C}(\mathcal{S})(At)(n)$.

Proof. ▶ Give in few words the idea of how $\mathbb{C}(\mathcal{S})(At)(\theta)$ acts on $\|A\|$, as a 'nodewise' application of θ to the whole tree $\|A\| \in \mathbb{C}(\mathcal{S})(At)(n)$. This should convince the reader of the statement.

4 Desaturation

As outlined in the introduction, one of the main features of Komendatskaya and Power's coinductive forests is to represent (sound) and-or parallel derivation of goals. In [KP11a] this notion of computation leads to a resolution algorithm exploiting the two forms of parallelism. Motivated by these developments, we include coinductive forests in our framework, showing how they can be obtained as a "desaturation" of saturated derivation trees. This also provides a correctness and completeness theorem of saturated derivation trees with respect to *SLD*-resolution, as a consequence of the analogous result shown for coinductive forests in [KP11a, Th.4.8].
For this purpose, we recall from the previous section that the notion of behavior associated with coinductive forests can be encoded as a $\widehat{\mathcal{P}_c\mathcal{P}_f}$-coalgebra in $\mathbf{Set}^{|\mathcal{L}_\Sigma^{op}|}$ according to (5), which trivially satisfies the naturality requirement by discreteness of $|\mathcal{L}_\Sigma^{op}|$. Just as $\mathbf{Set}^{\mathcal{L}_\Sigma^{op}}$, the category $\mathbf{Set}^{|\mathcal{L}_\Sigma^{op}|}$ is (co)complete and locally presentable, meaning that we can provide the cofree comonad $\mathcal{C}(\widehat{\mathcal{P}_c\mathcal{P}_f})\colon \mathbf{Set}^{|\mathcal{L}_\Sigma^{op}|} \to \mathbf{Set}^{|\mathcal{L}_\Sigma^{op}|}$ following the same steps of construction 3, with $\widehat{\mathcal{P}_c\mathcal{P}_f}\colon \mathbf{Set}^{|\mathcal{L}_\Sigma^{op}|} \to \mathbf{Set}^{|\mathcal{L}_\Sigma^{op}|}$ in place of $\mathcal{S}\colon \mathbf{Set}^{\mathcal{L}_\Sigma^{op}} \to \mathbf{Set}^{\mathcal{L}_\Sigma^{op}}$. We state its definition for later reference.

Construction 5 *The terminal sequence for the functor* $\mathcal{U}At \times \widehat{\mathcal{P}_c\mathcal{P}_f}(-)\colon \mathbf{Set}^{|\mathcal{L}_\Sigma^{op}|} \to \mathbf{Set}^{|\mathcal{L}_\Sigma^{op}|}$ *consists of sequences of objects* Y_α *and arrows* $\lambda_\alpha\colon Y_{\alpha+1} \to Y_\alpha$*, defined by induction on* α *as follows.*

$$Y_\alpha := \begin{cases} \mathcal{U}At & \alpha = 0 \\ \mathcal{U}At \times \widehat{\mathcal{P}_c\mathcal{P}_f}(Y_\beta) & \alpha \text{ is a successor ordinal } \beta+1 \end{cases}$$

$$\lambda_\alpha := \begin{cases} \pi_1 & \alpha = 0 \\ Id_{\mathcal{U}At} \times \widehat{\mathcal{P}_c\mathcal{P}_f}(\lambda_\beta) & \alpha \text{ is a successor ordinal } \beta+1 \end{cases}$$

For α *a limit ordinal,* Y_α *and* λ_α *are defined as expected. As stated in proposition 2,* $\widehat{\mathcal{P}_c\mathcal{P}_f}$ *is a mono-preserving accessible functors. Then by theorem [Wor99, Th. 7] we know that the sequence given above converges to a limit* Y_χ *such that* $Y_\chi \cong Y_{\chi+1}$ *and* Y_χ *is the value of* $\mathcal{C}(\widehat{\mathcal{P}_c\mathcal{P}_f})\colon \mathbf{Set}^{|\mathcal{L}_\Sigma^{op}|} \to \mathbf{Set}^{|\mathcal{L}_\Sigma^{op}|}$ *on* $\mathcal{U}At$*, where* $\mathcal{C}(\widehat{\mathcal{P}_c\mathcal{P}_f})$ *is the cofree comonad on* $\widehat{\mathcal{P}_c\mathcal{P}_f}$ *induced by the terminal sequence analogously to construction 1.*

We now have a $\mathcal{L}_\Sigma^{op}$-indexed presheaf $\mathcal{C}(\mathcal{S})(At)$ and a $|\mathcal{L}_\Sigma^{op}|$-indexed presheaf $\mathcal{C}(\widehat{\mathcal{P}_c\mathcal{P}_f})(At)$, representing respectively the space of saturated

derivation trees and of coinductive forests. The next construction provides a translation from the former to the latter notion of computation, in the form of a natural transformation $\overline{d}\colon \mathcal{U}\big(\mathcal{C}(\mathcal{S})(At)\big) \to \mathcal{C}(\widehat{\mathcal{P}_c}\widehat{\mathcal{P}_f})(At)$ in $\mathbf{Set}^{|\mathcal{L}_\Sigma^{op}|}$. The key ingredient of this translation is provided by the *counit* ϵ of the saturation adjunction $\mathcal{U} \dashv \mathcal{K}$. Given a presheaf $\mathcal{F}\colon |\mathcal{L}_\Sigma^{op}| \to \mathbf{Set}$, this is a natural transformation $\epsilon_{\mathcal{F}}\colon \mathcal{K}\mathcal{U}(\mathcal{F}) \to \mathcal{F}$ in $\mathbf{Set}^{|\mathcal{L}_\Sigma^{op}|}$ defined as follows:

$$\epsilon_{\mathcal{F}}(n)\colon \prod_{\theta \in \mathcal{L}_\Sigma^{op}[n,m]} \mathcal{F}(m) \to \mathcal{F}(n)$$
$$\langle x_\theta \rangle_{\theta \in \mathcal{L}_\Sigma^{op}[n,m]} \mapsto x_{Id_n}.$$

where x_{Id_n} is the element of the input tuple which is indexed by the identity substitution $Id_n \in \mathcal{L}_\Sigma^{op}[n,n]$. In the logic programming perspective, the intuition is that, while the unit of the adjunction provided the saturation of a goal, the counit reverses the process. It takes the saturation of a goal and gives back the substitution instance given by the identity substitution, that is, the goal itself.
The construction of $\overline{d}$ is supplied as a nodewise application of this "desaturation" process to the saturated derivation trees in $\mathcal{U}\big(\mathcal{C}(\mathcal{S})(At)\big) = \mathcal{C}(\mathcal{S})(At)$.

Construction 6 *Consider the image of the terminal sequence for $X_\gamma = \mathbb{C}(\mathcal{S})(At)$ (construction 3) under the forgetful functor $\mathcal{U}\colon \mathbf{Set}^{\mathcal{L}_\Sigma^{op}} \to \mathbf{Set}^{|\mathcal{L}_\Sigma^{op}|}$. We define a sequence of natural transformations $\{d_\alpha\colon \mathcal{U}(X_\alpha) \to Y_\alpha\}_{\alpha<\gamma}$ as follows.*

$$d_\alpha := \begin{cases} Id_{\mathcal{U}At} & \alpha = 0 \\ Id_{\mathcal{U}At} \times \big(\widehat{\mathcal{P}_c}\widehat{\mathcal{P}_f}(d_\beta) \circ \epsilon_{\widehat{\mathcal{P}_c}\widehat{\mathcal{P}_f}(\mathcal{U}X_\beta)}\big) & \alpha \text{ is a succ. ord. } \beta+1 \end{cases} \quad (8)$$

Here $\epsilon_{\widehat{\mathcal{P}_c}\widehat{\mathcal{P}_f}\mathcal{U}X_\beta}\colon \mathcal{U}\mathcal{K}\widehat{\mathcal{P}_c}\widehat{\mathcal{P}_f}\mathcal{U}X_\beta \to \widehat{\mathcal{P}_c}\widehat{\mathcal{P}_f}\mathcal{U}X_\beta$ is the instantiation at $\widehat{\mathcal{P}_c}\widehat{\mathcal{P}_f}\mathcal{U}X_\beta$ of the counit of the adjunction $\mathcal{U} \dashv \mathcal{K}$. Concerning the successor case, observe that $Id_{\mathcal{U}At} \times \big(\widehat{\mathcal{P}_c}\widehat{\mathcal{P}_f}(ds_\beta) \circ \epsilon_{\widehat{\mathcal{P}_c}\widehat{\mathcal{P}_f}(\mathcal{U}X_\beta)}\big)$ is in fact an arrow from $\mathcal{U}At \times \mathcal{U}\mathcal{K}\widehat{\mathcal{P}_c}\widehat{\mathcal{P}_f}(\mathcal{U}X_\beta)$ to $Y_{\beta+1} = \mathcal{U}At \times \widehat{\mathcal{P}_c}\widehat{\mathcal{P}_f}(Y_\beta)$. However, the former is isomorphic to $\mathcal{U}(X_{\beta+1}) = \mathcal{U}\big(At \times \mathcal{K}\widehat{\mathcal{P}_c}\widehat{\mathcal{P}_f}(\mathcal{U}X_\beta)\big)$, because $\mathcal{U}$ is a right adjoint (namely to the left Kan extension along $i\colon |\mathcal{L}_\Sigma^{op}| \hookrightarrow \mathcal{L}_\Sigma^{op}$) and thence it preserves limits.
For α a limit ordinal, a natural transformation $d_\alpha\colon \mathcal{U}(X_\alpha) \to Y_\alpha$ is provided by the limiting property of Y_α. In order to show that the limit case is well defined, observe that, for each limit ordinal $\alpha < \gamma$, for every $\beta < \alpha$, the following square commutes

$$\begin{array}{ccc} \mathcal{U}(X_\alpha) & \xrightarrow{d_\alpha} & Y_\alpha \\ {\scriptstyle \mathcal{U}(\delta_\alpha)}\uparrow & & \uparrow{\scriptstyle \lambda_\alpha} \\ \mathcal{U}(X_{\alpha+1}) & \xrightarrow[d_{\alpha+1}]{} & Y_{\alpha+1} \end{array}$$

as can be easily checked by transfinite induction, using the fact that $\epsilon_{\widehat{\mathcal{P}_c}\widehat{\mathcal{P}_f}(\mathcal{U}X_\beta)}$ *is a natural transformation, for each such* β*.*
We turn turn to the definition of a natural transformation $\overline{d}\colon \mathcal{U}\big(\mathcal{C}(\mathcal{S})(At)\big) \to \mathcal{C}(\widehat{\mathcal{P}_c}\widehat{\mathcal{P}_f})(At)$*. Observe that, since* $\widehat{\mathcal{P}_c}\widehat{\mathcal{P}_f}$ *is part of* $\mathcal{S}$*, the terminal sequence for the former cannot converge in less steps than the one for the latter. Therefore the ordinal* χ *is not bigger than* γ*, where* $Y_\chi = \mathcal{C}(\widehat{\mathcal{P}_c}\widehat{\mathcal{P}_f})(At)$ *is given as in construction 5. It follows that we also have a natural transformation* $\overline{d}\colon \mathcal{U}(X_\gamma) \to Y_\chi$*, given by the natural transformation* $d_\chi\colon \mathcal{U}(X_\chi) \to Y_\chi$ *together with the limiting property of* $\mathcal{U}(X_\gamma)$ *on* $\mathcal{U}(X_\chi)$*.*

▶ Section to be completed

5 Bonus tracks?

(A questo punto mi sa di no...)

6 Conclusions

References

[AR94] J. Adámek and J. Rosický. *Locally Presentable and Accessible Categories*. Cambridge University Press, 1994.

[DKM84] Cynthia Dwork, Paris C. Kanellakis, and John C. Mitchell. On the sequential nature of unification. *The Journal of Logic Programming*, 1(1):35 – 50, 1984.

[GBM+07] Gopal Gupta, Ajay Bansal, Richard Min, Luke Simon, and Ajay Mallya. Coinductive logic programming and its applications. In *ICLP*, pages 27–44, 2007.

[GC94] Gopal Gupta and Vtor Santos Costa. Optimal implementation of and-or parallel prolog. In *PARLE*, pages 71–92, 1994.

[GPA+01] Gopal Gupta, Enrico Pontelli, Khayri A.M. Ali, Mats Carlsson, and Manuel V. Hermenegildo. Parallel execution of prolog programs: a survey. *ACM Trans. Program. Lang. Syst.*, 23(4):472–602, July 2001.

[KMP10] Ekaterina Komendantskaya, Guy McCusker, and John Power. Coalgebraic semantics for parallel derivation strategies in logic programming. In *AMAST*, pages 111–127, 2010.

[KP11a] Ekaterina Komendantskaya and John Power. Coalgebraic derivations in logic programming. In *CSL*, pages 352–366, 2011.

[KP11b] Ekaterina Komendantskaya and John Power. Coalgebraic semantics for derivations in logic programming. In *CALCO*, pages 268–282, 2011.

[ML98] Saunders Mac Lane. *Categories for the Working Mathematician*. Graduate Texts in Mathematics. Springer, 2nd edition, September 1998.

[Wor99] James Worrell. Terminal sequences for accessible endofunctors. *Electronic Notes in Theoretical Computer Science*, 19, 1999.