Submitted to MFPS 2025

CT-λ, a cyclic simply typed λ-calculus with fixed points*

Stefano Berardi Ugo de' Liguoro Daisure Kimura Koji Nakazawa Makoto Tatsuta

[a] *C.S. Dept., Turin University, Torino, Italy*

[b] *Department of Information Science, Toho University, Japan*

[c] *Graduate School of Informatics, Nagoya University, Japan*

[d] *National Institute of Informatics, Sokendai, Tokyo, Japan*

Abstract

A motivation for having a circular syntax for Gödel System $\mathbb{T}$ is that terms in a circular syntax are much shorter than the equivalent terms written in $\mathbb{T}$, yet they can be checked mechanically. Several circular syntax for $\mathbb{T}$ (that is, infinite regular terms with the global trace condition) have been proposed, like CT by Anupam ([**?**], **S**4.5, page 20), but without an explicit use of λ-abstraction. In this paper we explore the possibility of defining a circular syntax directly for λ-abstraction and simple types, which we call CT-λ. We introduce our circular syntax as a fixed point operator defined by cases through a test on 0. We prove for CT-λ: *(i)* every closed term of type N reduces to some numeral; *(ii)* strong normalization for all reductions sequences never reducing inside a fixed point operator. *(iii)* the existence of a limit for all infinite reductions.

Keywords: Please list keywords from your paper here, separated by commas.

1 Introduction

We will introduce $\mathbb{\Lambda}$, a set of infinite λ-terms with a circular syntax. The types of $\mathbb{\Lambda}$ are: the atomic type N, possibly type variables $\alpha, \beta, \ldots$, and all types $A \to B$ for any types A, B. Type variables are only used to provide examples of terms and play a minor role in our paper.

The terms of $\mathbb{\Lambda}$ are all possibly infinite trees representing expressions defined with 0,S,ap (application), variables x^T (with a type superscript T), λ (the binder for defining maps), and cond, the arithmetic conditional (i.e., the test on zero). If we have no term variables and no type variables, then the trees in $\mathbb{\Lambda}$ represent partial functionals on N, provided we add reduction rules transforming closed terms of type N in notations for natural numbers.

In this paper will consider two sets of terms:

1. the set CT-λ of well-typed terms in a circular syntax, which are equivalent to the set of terms in Gödel system $\mathbb{T}$.
2. The set of terms GTC, satisfying a condition called global trace condition, which are possibly non-recursive terms used to provide semantics for CT-λ (and $\mathbb{T}$, of course)

We introduce more sets of terms only used as intermediate steps in the definition of the semantic GTC and the syntax CT-λ.

1. WTyped $\subseteq \mathbb{\Lambda}$, the set of well-typed terms, is the of set terms having a unique type

* General thanks to everyone who should be thanked.

2. `Reg` is the set of terms of $\mathbb{\Lambda}$ which are regular trees (i.e., having finitely many subtrees). They are possibly infinite terms which are finitely presented by a finite graph possibly having cycles.
3. `GTC` $\subseteq$ `WTyped` will be defined as the set of well-typed circular λ-terms satisfying the global trace condition (and possibly non-recursive, as we said). Terms of `GTC` denote total functionals.

We will prove that `CT-`λ is a decidable subset of `Reg`. `CT-`λ is a new variant of the existing circular version of Gödel system $\mathbb{T}$. Differently from all previous circular versions of $\mathbb{T}$, our system `CT-`λ uses binders instead of combinators. As we anticipated, circular syntax has the advange of writing much shorter terms while preserving decidability of termination. Besides, by introducing a circular syntax with binders, we hope to provide a circular syntax more familiar to researchers working in the field of Type Theory.

We will prove the expected results for the circular syntax `CT-`λ: strong normalization for reductions not in the right-hand side of any `cond` and Church-Rosser. We will prove normalization in the limit if we use reductions which are "fair": fair reductions can reduce *inside* the right-hand side of some `cond`, but they never forget entirely the task of reducing *outside* all such subterms.

Eventually, we will prove that the closed terms `CT-`λ (those without free variables) represent exactly the total computable functionals definable in Gödel system $\mathbb{T}$.

2 The set of infinite λ-terms

We define the set $\mathbb{\Lambda}$ of infinite circular terms, the subset `WTyped` of well-typed terms and a reduction relation for them. Infinite terms are labeled binary trees, therefore we have to define binary trees first, and before them the lists on $\{1,2\}$ and more related notions.

2.1 Lists and Binary Trees

Definition 2.1 [Lists on $\{1,2\}$]

1. We denote with $\mathrm{List}(\{1,2\})$ the set of all lists of elements of $\{1,2\}$: $(), (1), (1), (1,1), (1,2), \ldots$. We call the elements of $\mathrm{List}(\{1,2\})$ just lists for short.
2. If $i_1, \ldots, i_n \in \{1,2\}$, we write $(i_1, \ldots, i_n)$ for the list with elements $i_1, \ldots, i_n$.
3. We write `nil` for the empty list, or $()$.
4. If $l = (i_1, \ldots, i_n)$, $m = (j_1, \ldots, j_m)$ and $l, m \in \mathrm{List}(\{1,2\})$ we define
$$l \star m = (i_1, \ldots, i_n,\ j_1, \ldots, j_m) \in \mathrm{List}(\{1,2\})$$
5. The prefix order $\leq$ on lists is defined by $l \leq m$ if and only if $m = l \star m'$ for some $m' \in \mathrm{List}(\{1,2\})$.

Binary trees are represented by set of lists.

Definition 2.2 [Binary trees]

1. A binary tree, just a tree for short, is any set $T \subseteq \mathrm{List}(\{1,2\})$ of lists including the empty list and closed by prefix: `nil` $\in T$ and for all $l, m \in \mathrm{List}(\{1,2\})$ if $m \in T$ and $l \leq m$ then $l \in T$.
2. `nil` is called the root of T, any $l \in T$ is called a node of T and if $l \in T$, then any $l \star (i) \in T$ is called a child of l in T.
3. A node l of T is a leaf of T if l is an maximal element of T w.r.t. the prefix order (i.e., if l has no children).
4. If T is a tree and $l \in T$ we define $T\lceil l = \{m \in \mathrm{List}(\{1,2\}) | l \star m \in T\}$. $T\lceil l$ is a tree and we say that $T\lceil l$ is a subtree of T in the node l, and for each $m \in T\lceil l$ we say that $l \star m$ is the corresponding node in T.
5. If $l = (i)$ we say that T_l is an immediate sub-tree of T.
6. A binary tree labeled on a set L is any pair $\mathcal{T} = (T, \phi)$ with $\phi : T \to L$. We call $|\mathcal{T}| = T$ the set of nodes of $\mathcal{T}$. For all $l \in T$ we call $\mathtt{Lbl}(\mathcal{T}, l) = \phi(l)$ the label of l in $\mathcal{T}$.

7. We define $\mathcal{T}\lceil l = (T\lceil l, \phi_l)$ and $\phi_l(m) = l\star m$. We call any $\mathcal{T}\lceil l$ a labeled subtree of $\mathcal{T}$ in the node l, an immediate sub-tree if $l = (i)$.

The labeling of a node $m \in |(\mathcal{T}\lceil l)|$ is the same label of the corresponding node $l\star m$ of $|\mathcal{T}|$.
Let $n = 0, 1, 2$. Assume $\mathcal{T}_1, \ldots, \mathcal{T}_n$ are trees labeled on L and $l \in L$. Then we define

$$\mathcal{T} = l(\mathcal{T}_1, \ldots, \mathcal{T}_n)$$

as the unique tree labeled on L with the root labeled l and with: $\mathcal{T}\lceil(i) = \mathcal{T}_i$ for all $1 \leq i \leq n$.

2.2 Types, Terms and Contexts

Now we define the types of $\mathbb{\Lambda}$, then the terms and the contexts of $\mathbb{\Lambda}$.

Definition 2.3 [Types of $\mathbb{\Lambda}$]

1. The types of $\mathbb{\Lambda}$ are: the type N of natural numbers, an infinite list $\alpha, \beta, \ldots$ of type variables, and with A, B also $A \to B$. We call them simple types, just *types* for short.
2. $\mathtt{Type}$ is the set of simples types.
3. We suppose be given a set $\mathtt{var}$, consisting of all pairs $x^T = (x, T)$ of a variable name x and a type $T \in \mathtt{Type}$.

Terms of $\mathbb{\Lambda}$ are labeled binary trees.

Definition 2.4 [Terms of $\mathbb{\Lambda}$] The terms of $\mathbb{\Lambda}$ are all binary trees t with set of labels $L = \{x^T, \lambda x^T., \mathtt{ap}, 0, \mathtt{S}, \mathtt{cond}\}$ such that:

1. a node l labeled x^T or 0 is a leaf of t (no children).
2. a node l labeled $\lambda x^T.$ or $\mathtt{S}$ has a unique child $l\star(1)$.
3. a node labeled $\mathtt{ap}$, $\mathtt{cond}$ has two children $l\star(1)$, $l\star(2)$.

We say that t is a sub-term of u if t is a labeled sub-tree of u, an immediate subterm if it is an immediate sub-tree.

Assume $l \in L = \{x^T, \lambda x^T., \mathtt{ap}, 0, \mathtt{S}, \mathtt{cond}\}$. We use the operation $l(\mathcal{T}_1 \ldots \mathcal{T}_n)$ on labeled trees to define new terms in $\mathbb{\Lambda}$. If $t, u \in \mathbb{\Lambda}$ then $x^T, 0, \lambda x^T.t, \mathtt{ap}(t, u), \mathtt{S}(t), \mathtt{cond}(t, u) \in \mathbb{\Lambda}$ are terms with the root labeled l, and with immediate subterms among t, u. Conversely, each $v \in \mathbb{\Lambda}$ is in one of the forms: $x^T, 0, \lambda x^T.t$, $\mathtt{ap}(t, u)$, $\mathtt{S}(t)$, $\mathtt{cond}(t, u)$ for some $t, u \in \mathbb{\Lambda}$.
Now we define the regular terms $\mathbb{\Lambda}$.

Definition 2.5 [Regular terms of $\mathbb{\Lambda}$]

1. We write $\mathtt{SubT}(t)$ for the (finite or infinite) set of subterms of t. Subterms are coded by the nodes of t, different nodes can code the same subterm.
2. $\mathtt{Reg}$ is the set of terms $t \in \mathbb{\Lambda}$ such that $\mathtt{SubT}(t)$ is finite. We call the terms of $\mathtt{Reg}$ the *regular terms*.
3. As usual, we abbreviate $\mathtt{ap}(t, u)$ with $t(u)$.
4. When $t = \mathtt{S}^n(0)$ for some natural number $n \in \mathbb{N}$ we say that t is a numeral. We write $\mathtt{Num}$ for the set of numeral in $\mathbb{\Lambda}$.
5. A variable x^T is free in t if there is some $l \in t$ labeled x^T in T, and *no* $m < l$ *labeled* $\lambda x^T.$ in t. $\mathrm{FV}(t)$ is the set of free variables of t.

$\mathtt{Num}$ is the representation inside $\mathbb{\Lambda}$ of the set $\mathbb{N}$ of natural numbers. All numerals are finite trees of Λ. All finite well-typed typed λ-terms we can define with the rules above are finite terms of $\mathbb{\Lambda}$. Regular terms can be represented by the subterm relation restricted to $\mathtt{SubT}(t)$: this relation defines a graph with possibly cicles. Even if $\mathtt{SubT}(t)$ is finite, t can be an infinite tree.

We do not assume implicit renaming of bound variables, namely α-equivalence because the typing system (given later) does not have renaming rule as a primitive rule.

Example 2.1 *An example of regular term which is an infinite tree: the term* $t = \mathtt{cond}(0, t) \in \mathbb{\Lambda}$. *The set* $\mathtt{SubT}(t) = \{t, 0\}$ *of subterms of* t *is finite, therefore* t *is a regular term. However,* t *is an infinite tree (it includes itself as a subtree). The immediate sub-term chains of* t *are all* $(t, t, t, \ldots, t)$ *and* $(t, t, t, \ldots, 0)$. *There is a unique infinite sub-term chain, which is* $(t, t, t, \ldots)$.

In order to define the type of an infinite term, we first define contexts and sequences for any term of $\mathbb{\Lambda}$. A context is a list of type assignments to variables: our variables already have a type superscript, so in fact a type assignment $x^T : T$ is *redundant* for our variables (not for our terms). Yet, we add an assignment relation $x^T : T$ for uniformity with the notation $x : T$ in use in Type Theory.

Definition 2.6 [Contexts of $\mathbb{\Lambda}$]

1. A context of $\mathbb{\Lambda}$ is any finite list $\Gamma = ({x_1}^{A_1} : A_1, \ldots, x_n^{A_n} : A_n)$ of *pairwise distinct variables*, each assigned to its type superscript $A_1, \ldots, A_n \in \mathtt{Type}$.
2. We denote the empty context with $\mathtt{nil}$. We write $\mathtt{Ctxt}$ for the set of all contexts.
3. A sequent is the pair of a context Γ and a type A, which we write as $\Gamma \vdash A$. We write $\mathtt{Seq} = \mathtt{Ctxt} \times \mathtt{Type}$ for the set of all sequents.
4. A typing judgement is the list of a context Γ, a term t and a type A, which we write as $\Gamma \vdash t : A$. We write $\mathtt{Stat} = \mathtt{Ctxt} \times \mathbb{\Lambda} \times \mathtt{Type}$ for the set of all typing judgements.
5. We write $\mathrm{FV}(\Gamma) = \{{x_1}^{A_1}, \ldots, {x_n}^{A_n}\}$. We say that Γ is a context for $t \in \mathbb{\Lambda}$ and we write $\Gamma \vdash t$ if $\mathrm{FV}(t) \subseteq \mathrm{FV}(\Gamma)$.
6. If $\Gamma = ({x_1}^{A_1} : A_1, \ldots, x_n^{A_n} : A_n)$, $\Gamma' = (x'_1 : A'_1, \ldots, x'_n : A'_{n'})$ are context of $\mathbb{\Lambda}$, then we write
   (1) $\Gamma \subsetneq \Gamma'$ if for all $(x^A : A)$: $(x^A : A) \in \Gamma \Rightarrow (x^A : A) \in \Gamma'$
   (2) $\Gamma \sim \Gamma'$ if for all $(x^A : A)$: $(x^A : A) \in \Gamma \Leftrightarrow (x^A : A) \in \Gamma'$
7. If Γ is a context of $\mathbb{\Lambda}$, then $\Gamma \setminus \{x^T : T\}$ is the context obtained by removing $x_i^{A_i} : A_i$ from Γ for all $x_i^{A_i} = x^T$. If $x \notin \mathrm{FV}(\Gamma)$ then $\Gamma \setminus \{x^T : T\} = \Gamma$.

We have $\Gamma \subsetneq \Gamma'$ if and only if if there is a (unique) map $\phi : \{1, \ldots, n\} \to \{1, \ldots, n'\}$ such that $x_i = x'_{\phi(i)}$ and $A_i = A'_{\phi(i)}$ for all $i \in \{1, \ldots, n\}$. We have $\Gamma \sim \Gamma'$ if and only if $\Gamma \subsetneq \Gamma'$ and $\Gamma \supsetneq \Gamma'$ if and only if Γ, Γ' are permutation each other if and only if the map ϕ above is a bijection. We do not identify two contexts which are one a permutation of the other, therefore we will need a rule to move from one to the other.

We define the term $t[u/x]$ obtained by substituting the infinite term u for the free variable x in t avoiding variable captures. In order to define the capture avoiding substitution under the assumption of not having implicit α-equivalence, we implicitly fix a well-order on variables and define the simultaneous substitution, written $t[u_1/x_1, \ldots, u_n/x_n]$ or $t[\boldsymbol{u}/\boldsymbol{x}]$, namely it satisfies the following:

- $x_i[u_1/x_1, \ldots, u_n/x_n] = u_i$ and $y[\boldsymbol{u}/\boldsymbol{x}] = y$, where $y \notin \boldsymbol{x}$.
- $(f(a))[\boldsymbol{u}/\boldsymbol{x}] = f[\boldsymbol{u}/\boldsymbol{x}](a[\boldsymbol{u}/\boldsymbol{x}])$.
- $(\lambda z^T.b)[\boldsymbol{u}/\boldsymbol{x}] = \begin{cases} \lambda z^T.(b[(\boldsymbol{u}/\boldsymbol{x})_{\overline{z}}]), & \text{if } z \notin \mathrm{FV}(\boldsymbol{u}), \\ \lambda z'^T.(b[(\boldsymbol{u}/\boldsymbol{x})_{\overline{z}}, z'/z]), & \text{otherwise,} \end{cases}$
  where $(\boldsymbol{u}/\boldsymbol{x})_{\overline{z}}$ is the one obtained by removing u/z (for some u) from $\boldsymbol{u}/\boldsymbol{x}$, and z' is the least variable (with respect to the well-order) such that $z' \notin \mathrm{FV}(b, \boldsymbol{u})$.
- $0[\boldsymbol{u}/\boldsymbol{x}] = 0$.
- $\mathtt{S}(t)[\boldsymbol{u}/\boldsymbol{x}] = \mathtt{S}(t[\boldsymbol{u}/\boldsymbol{x}])$.
- $\mathtt{cond}(f, g)[\boldsymbol{u}/\boldsymbol{x}] = \mathtt{cond}(f[\boldsymbol{u}/\boldsymbol{x}], g[\boldsymbol{u}/\boldsymbol{x}])$.

Formally we define the substitution as follows.

Definition 2.7 [Substitution for terms of $\mathbb{\Lambda}$] A set $\theta = \{u_1/x_1, \ldots, u_n/x_n\}$ is called a *substitution* if $x_1, \ldots, x_n$ are pairwise distinct variables. It is called *renaming* if $u_1, \ldots, u_n$ are pairwise distinct variables. We define $\theta(y)$ by u_i if $y = x_i$, and by y otherwise. We write θ, θ' if $\theta \cup \theta'$ is a substitution. The substitution $\theta_{\overline{x}}$ is defined as the one obtained by removing u/x (for some u) from θ.

Let t be a term of $\mathbb{\Lambda}$, which is the labeled tree (T_t, ϕ_t), and θ be $\{u_1/x_1, \ldots, u_n/x_n\}$, where each u_i is (T_{u_i}, ϕ_{u_i}). Then the term $t[\theta]$ is (T, ϕ), where $T = T_t \cup \{l \star m \mid \phi_t(l) = x_i \in \mathrm{FV}(t) \text{ and } m \in T_{u_i}\}$. For

each $l\star m$ in the latter part, $\phi(l\star m) = \phi_{u_i}(m)$. For $l \in T_t$, we inductively define $\phi(l)$ and θ_l (such that $\theta_l = \theta', \theta_{ren}$, where $\theta' \subseteq \theta$ and θ_{ren} is a renaming) as follows.

- $\theta_{\mathtt{nil}} = \theta$.
- If $\phi_t(l) = y^T \notin \mathrm{FV}(t)$, then $\phi(l) = \theta_l(y^T)$.
- If $\phi_t(l) = 0$, then $\phi(l) = 0$.
- If $\phi_t(l) = \mathtt{S}$, then $\phi(l) = \mathtt{S}$ and $\theta_{l\star(1)} = \theta_l$.
- If $\phi_t(l) \in \{\mathtt{ap}, \mathtt{cond}\}$, then $\phi(l) = \phi_t(l)$ and $\theta_{l\star(1)} = \theta_{l\star(2)} = \theta_l$.
- If $\phi_t(l) = \lambda z^T.$ and $z \notin \mathrm{FV}(\boldsymbol{u})$, then $\phi(l) = \lambda z^T.$ and $\theta_{l\star(1)} = (\theta_l)_{\overline{z}}$.
- If $\phi_t(l) = \lambda z^T.$ and $z \in \mathrm{FV}(\boldsymbol{u})$, then $\phi(l) = \lambda z'^T.$ and $\theta_{l\star(1)} = (\theta_l)_{\overline{z}}, z'/z$, where z' is the least variable such that $z' \notin \mathrm{FV}(t\lceil l\star(1), \boldsymbol{u})$.

We often abbreviate $t[\{u_1/x_1, \ldots, u_n/x_n\}]$ by $t[u_1/x_1, \ldots, u_n/x_n]$. For a context $\Gamma = (x_1 : A_1, \ldots, x_n : A_n)$ and a renaming θ, we write $\Gamma[\theta]$ for $(\theta(x_1) : A_1, \ldots, \theta(x_n) : A_n)$ if it is a context.

We have the following substitution lemma.

Lemma 2.8 (Substitution lemma) *$t[u/x][v/y] = t[t/y][u[v/y]/x]$ if $x \notin \mathrm{FV}(v)$.*

2.3 Typing Rules

We define typing rules for terms of $\mathbb{\Lambda}$ and the subset `WTyped` of well-typed terms. We consider a term well-typed if typing exists and it is unique for the term and for all its subterms.

The typing rules are the usual ones but for the *typing rule* `ap` *for an application* $t(u)$, which we split in two sub-rules, $\mathtt{ap}_v$ and $\mathtt{ap}_{\neg v}$, according if u is a variable or is not a variable. We need to insert this extra information t in typing because it is important for checking termination of the computation of $t(u)$, as we will explain later.

Remark that we introduced a unique *term notation* for application: we write $\mathtt{ap}(t, u)$ no matter if u is a variable or not, but we we split the typing rule for `ap` into two sub-cases, $\mathtt{ap}_v$ and $\mathtt{ap}_{\neg v}$.

We have a a single structural rule `weak` for extending a context Γ to a context $\Gamma' \supsetneq \Gamma$. When $\Gamma' \sim \Gamma$ (when Γ', Γ are permutation each other), the rule `weak` can be used for variable permutation. Variable renaming for a term $t[\boldsymbol{x}]$ can be obtained by writing $(\lambda \boldsymbol{x}.t[\boldsymbol{x}])(\boldsymbol{x'})$. Therefore we do not assume having a primitive rule for renaming. The lack of a renaming rule is a *difference with the circular syntax for inductive proofs*.

The reason of not having implicit α-conversion mentioned in the previous section comes from this assumption. For example, if we identify the α-equivalent terms $\lambda x^A.x$ and $\lambda z^A.z$, the left derivation is a correct proof, but the right one is not ($(z : A, z : A)$ is not a context). That is, the proof structure does not closed under identity of the α-equivalence.

$$\frac{\dfrac{}{z : A, x : A \vdash x : A}}{z : A \vdash \lambda x^A.x : A \to A} \qquad \frac{\dfrac{}{z : A, z : A \vdash z : A}}{z : A \vdash \lambda z^A.z : A \to A}$$

The provability of sequents is closed under the variable renaming and the α-equivalence. We will discuss this point later.

Definition 2.9 [Typing rules of $\mathbb{\Lambda}$] Assume $\Gamma = {x_1}^{A_1} : A_1, \ldots, x_n^{A_n} : A_1$ is a context. Let $p = 0, 1, 2$.

1. A rule is a list of $p+1$ typing judgements: $\Gamma \vdash t_1 : A_1, \ldots, \Gamma \vdash t_p : A_p, \Gamma \vdash t : A$. The first p sequents are the premises of the rule (at most two in our case), the last sequent is the conclusion of the rule.
2. We read the rule above: *"if $\Gamma \vdash t_1 : A_1, \ldots, \Gamma \vdash t_p : A_p$ then $\Gamma \vdash t : A$"*.

We list the typing rules of $\mathbb{\Lambda}$: the rules $\mathtt{ap}_v$ and $\mathtt{ap}_{\neg v}$ are restricted.

1. `weak`-rule (Weakening + Exchange). If $\Gamma \vdash t : T$ and $\Gamma \subsetneq \Gamma'$ then $\Gamma' \vdash t : T$
2. `var`-rule. If $x^A \in \Gamma$ then $\Gamma \vdash x^A : A$.
3. λ-rule. If $\Gamma, x^A : A \vdash b : B$ then $\Gamma \vdash \lambda x^A.b : A \to B$.
4. $\mathtt{ap}_v$-rule. If $\Gamma \vdash f : A \to B$ then $\Gamma \vdash f(x^A) : B$, provided $(x^A : A) \in \Gamma$.

5. $\mathtt{ap}_{\neg v}$-rule. If $\Gamma \vdash f : A \to B$ and $\Gamma \vdash a : A$ then $\Gamma \vdash f(a) : B$, provided a is *not* a variable
6. 0-rule. $\Gamma \vdash 0 : N$
7. S-rule. If $\Gamma \vdash t : N$ then $\Gamma \vdash \mathtt{S}(t) : N$.
8. cond-rule. If $\Gamma \vdash f : T$ and $\Gamma \vdash g : N \to T$ then $\Gamma \vdash \mathtt{cond}(f, g) : N \to T$.

We abbreviate $\mathtt{nil} \vdash t : A$ ($t : A$ in the empty context) with $\vdash t : A$. We write the set of rules of $\mathbb{\Lambda}$ as

$$\mathtt{Rule} = \{r \in \mathtt{Seq} \cup \mathtt{Seq}^2 \cup \mathtt{Seq}^3 | r \text{ instance of some rule of } \mathbb{\Lambda}\}$$

A rule is uniquely determined from its conclusion, provided we know whether the rule is a weakening or not. Two rules are equal if they have the same assumptions and the same conclusion.

Proposition 2.10 (Rules and subterms) *Assume $r, r' \in \mathtt{Rule}$ have conclusion $\Gamma \vdash t : A$, $\Gamma' \vdash t' : A'$ respectively.*

1. *If r, r' are non-weakening rules and $\Gamma \vdash t : A = \Gamma' \vdash' t' : A'$ then $r = r'$.*
2. *If r is not a weakening and the premises of r are some $\Gamma_1 \vdash t_1 : A_1, \ldots, \Gamma_p \vdash t_p : A_p$, then the list of immediate subterms of t is exactly $t_1, \ldots, t_p$.*

From the typing rules we define the proofs that a term is well-typed. Proofs are binary trees on the set Rule, each node is associated to a typing judgement which is a consequence of the typing judgements associated to the children node under some rule $r \in \mathtt{Rule}$ of $\mathbb{\Lambda}$.

The Almost-Left-Finite condition will ask that all leftmost branches in any sub-proof are finite, say, that no infinite leftmost branch made of weak-rules or of $\mathtt{ap}_{\neg v}$ exists. The idea is that when the main operation in a programming language is application, then the leftmost branch uniquely determines the type of the term and therefore it should be finite.

Definition 2.11 [Well-typed term of $\mathbb{\Lambda}$] Assume $\Pi = (T, \phi)$ is a binary tree labeled on Rule. Assume Γ is a context of t (i.e, $\mathrm{FV}(t) \subseteq \Gamma$) and $A \in \mathtt{Type}$

Then we write $\Pi : \Gamma \vdash t : A$, and we say that Π is a proof of $\Gamma \vdash t : A$ if:

1. $\phi(\mathtt{nil}) = \Gamma \vdash t : A$.
2. Assume that
   (1) $l \in T$ is labeled with $\phi(l) = \Delta \vdash u : B$.
   (2) The children of l in T are labeled $\phi(l\star(1)) = \Delta_1 \vdash u_1 : B_1, \ldots, \phi(l\star(p)) = \Delta_p \vdash u_p : B_p$
   Then for some rule $r \in \mathtt{Rule}$ of $\mathbb{\Lambda}$ we have
   $$\frac{\Delta_1 \vdash u_1 : B_1 \quad \cdots \quad \Delta_p \vdash u_p : B_p}{\Delta \vdash u : B}\ r$$
3. A path π in a proof tree Π is *almost-leftmost* if there are only finitely many rules with two premises in π such that π does *not* choose the leftmost premise.
4. A proof Π is Almost-left-finite if all almost-lefmost paths π in Π are finite

Eventually we define the well-typed terms of $\mathbb{\Lambda}$ using almost-left-finite proofs.

Definition 2.12 [Well-typed term of $\mathbb{\Lambda}$]

1. $\Gamma \vdash t : A$ is true if and only if $\Pi : \Gamma \vdash t : A$ for some Π.
2. $t \in \mathbb{\Lambda}$ is well-typed if and only if $\Pi : \Gamma \vdash t : A$ for some *almost-left-finite* proof Π, for some A and some context $\Gamma \supseteq \mathrm{FV}(t)$.
3. WTyped is the set of well-typed $t \in \mathbb{\Lambda}$.

Well-typed terms are closed by substitution.

We prove that the type assigned to a term by an Almost-Left-Finite proof is *unique*: if two Almost-Left-Finite proofs assign two types T, T' to the same possibly infinite term t, then $T = T'$.

Proposition 2.13 (Uniqueness of almost-left-finite type) *If $t \in \mathbb{\Lambda}$, $\Pi : \Gamma \vdash t : A$ and $\Pi' : \Gamma' \vdash t : A'$ are two almost-left-finite proofs then $A = A'$.*

2.4 Reduction Rules

Our goal is to provide a set of well-formed term for $\mathbb{\Lambda}$ and interpret them as partial functionals. Some terms, those satisfying the global trace condition (to be introduced later) will be total functionals. Our first step is to provide reduction rules for $\mathbb{\Lambda}$.

Definition 2.14 [reduction rules for $\mathbb{\Lambda}$]

1. $\mapsto_\beta$: $(\lambda x^A.b)(a) \mapsto_\beta b[a/x]$
2. $\mapsto_{\mathtt{cond}}$: $\mathtt{cond}(f,g)(0) \mapsto_{\mathtt{cond}} f$ and $\mathtt{cond}(f,g)(\mathtt{S}(t)) \mapsto_{\mathtt{cond}} g(t)$.
3. $\mapsto$ is the contraction of one β or `cond` redex: the context closure of $\mapsto_\beta$ and $\mapsto_{\mathtt{cond}}$.
4. $\mapsto^*$ is the reflexive and transitive closure of $\mapsto$ (zero or more reductions).
5. $\mapsto^+$ is the transitive closure of $\mapsto$ (one or more reductions).
6. The `cond`-depth of a node $l = (i_1, \ldots, i_n) \in t$ is the number of $1 \leq h < n$ such that $(i_1, \ldots, i_h)$ has label `cond` and $i_{h+1} = 2$ (the number of `cond` such that l occurs in the right-hand-side of such `cond`).
7. The n-safe level of t is the set of nodes of t of depth $\leq n$.
8. We say that $t \mapsto_{n,\mathtt{safe}} u$, or that t n-safely reduces to u, if we reduce t to u by contracting a single redex in the n-safe level of t (i.e. of `cond`-depth $\leq n$).
9. A term is n-safe-normal if all its redexes (if any) have `cond`-depth $> n$.
   We abbreviate "0-safe normal" with just "safe normal".

Example 2.2 *Let $u = \mathtt{cond}.(0, (\lambda z.u)(z))$, where we omitted the type superscript of z because it is irrelevant. Then u is 0-safe-normal (safe normal for short), because all redexes in u are of the form $(\lambda z.u)(z)$ and are in the right-hand-side of a* `cond`*. However u has infinitely many nodes which are β-redexes. Indeed, the tree form of u has the following branch:*

$$u, \quad (\lambda z.u)(z), \quad \lambda z.u, \quad u, \quad \ldots$$

This branch is cyclic, infinite, and it includes infinitely many β-redexes, all equal to the same term $(\lambda z.u)(z)$.

Through branches crossing the right-hand-side of some `cond` we will-express fixed-point equations. Reductions on fixed-point equations can easily loop. However, in order to produre the output of a computation we only need to reduce closed terms of type N to a numeral. We will prove that for this taks we only need 0-safe reductions, those in the the right-hand-side of *no* `cond`, and that 0-safe reductions strongly normalize.

Example 2.3 *This is an example of safe* `cond`*-reductions from a term $v(n)$ to a normal form. Assume $n \in \mathtt{Num}$ is any numeral and $v = \mathtt{cond}(1, v)$. There are only finitely many reductions from $v(n)$, and they are all 0-safe. $v(n)$* `cond`*-reduces to $v(n-1)$, then we loop: $v(n-1)$* `cond`*-reduces to $v(n-2)$ and so forth. After n* `cond`*-reductions we get $v(0)$. With one last* `cond`*-reduction we get 1 and we stop. Thus, the term $v(n)$ strongly normalizes to 1 in $(n+1)$-steps, in fact $v(n)$ has a unique reduction path, having length $(n+1)$ and terminating in 1.*

There are meaningful terms without a normal form, but we can get as close as we want to a normal form In fact, we will prove that for all n, all terms have some n-safe normal form, that is, with no redexes of `cond`-depth $\leq n$.

We will also prove that terms have a limit normal form, obtained by taking the limit of the n-safe normal forms of the term.

3 The trace of the infinite λ-terms

We define a notion of trace for possibly infinite λ-terms, describing how an input of type N is used when computing the output. The first step toward a trace is defining a notion of *connection* between arguments of type N in the proof that t is well-typed. To this aim, we need the notions of *list of argument types* and of *index of atomic types* for a term.

Definition 3.1 [List of argument types of a term] Assume that $\boldsymbol{A} = A_1, \ldots, A_n$, $\boldsymbol{B} = B_{n+1}, \ldots, B_{n+m}$, $\Gamma = \{\boldsymbol{x} : \boldsymbol{A}\}$, and $\Gamma \vdash t : \boldsymbol{B} \to N$.

1. The *list of argument types* of t is $\boldsymbol{C} = \boldsymbol{A}, \boldsymbol{B}$.
2. $A_1, \ldots, A_n$ are the *named arguments*, with names $x_1, \ldots, x_n$.
3. $B_{n+1}, \ldots, B_{n+m}$ are the *unnamed arguments*.
4. An *index of an N-argument* of t is any $j \in \{1, \ldots, n+m\}$ such that $C_j = N$.

The definition of connection requires to define an injection

$$\mathtt{ins} : N, N \to N$$

We set $\mathtt{ins}(p, x) = x + 1$ if $x \geq p + 1$, and $\mathtt{ins}(p, x) = x$ if $x \leq p$. The role of $\mathtt{ins}$ is inserting one fresh index $p + 1$ for a new argument type higher in the proof connected to no argument in the conclusion.

Definition 3.2 [Connection in a proof of $\mathbb{\Lambda}$] Assume $\boldsymbol{A} = A_1, \ldots, A_n$, $\boldsymbol{B} = B_1, \ldots, B_m$, $\Gamma = \boldsymbol{x} : \boldsymbol{A}$, and $\Gamma \vdash t : \boldsymbol{B} \to N$.

For each N-argument k in t, for $t' = t$ or t' immediate subterm of t each N-argument k' in t' we define the relation: "k', t' the successor of k, t". We require:

1. if t is obtained by a rule $\mathtt{weak}$ from $t' = t$, described by a map $\phi : \Gamma \to \Delta$, then $k = \phi(k')$.
2. if $t = f(a)$ is obtained by a rule $\mathtt{ap}_{\neg v}$ (i.e., a is not a variable) and $t' = f$ then $k' = \mathtt{ins}(n, k)$. If $t' = a$ then $k' = k$ and $k' \leq n$.
3. if $t = f(x_i^{A_i})$ is obtained by a rule $\mathtt{ap}_v$ and $t' = f$ then $k' = \mathtt{ins}(n, k)$or $k' = n + 1$ and $k = i$.
4. if $t = \mathtt{cond}(f, g)$ and $t' = f$ then $k' = \mathtt{ins}(n, k)$. If $t' = g$ then $k' = k$.

We require $k = k'$ in all other cases, which are: $\mathtt{S}$, λ.
No connection is defined for the rules $\mathtt{var}$, 0.

Below we include some examples. We draw in the same color two N-argument which are in connection. In the case of the rule $\mathtt{ap}_v$, an argument can be connected to two arguments above it.

Example 3.1 *Atom connection for $\mathtt{ap}_v$. We assume that x is a variable.*

$$\frac{x_1 : \mathbf{N}, x_2 : \mathbf{N}, x_3 : \mathbf{N} \vdash f : \mathbf{N}, \mathbf{N} \to N}{x_1 : \mathbf{N}, x_2 : \mathbf{N}, x_3 : \mathbf{N} \vdash f(x_3) : \mathbf{N} \to N}\ (\mathtt{ap}_v)$$

Remark that the last variable in the conclusion of the rule is in connection with the first unnamed argument of f (colored in **old gold**) in the premise, and with the variable with the same name in the premise: there are two connections.

Example 3.2 *Atom connection for λ-rule. We assume that x is a variable.*

$$\frac{x_1 : \mathbf{N}, x_2 : \mathbf{N}, x : \mathbf{N} \vdash b : N}{x_1 : \mathbf{N}, x_2 : \mathbf{N} \vdash \lambda x.b : \mathbf{N} \to N}\ (\mathtt{ap})$$

Remark that the first unnamed argument of $\lambda x.b$ (colored in **old gold**) in the conclusion is in connection with the last variable type in the premise of the rule.

Summing up, when we move up in a $\mathtt{ap}_v$-rule, the type of last free variable corresponds to the type of the first unnamend argument. When we move up in a λ-rule it is the other way round.

The connection on N-arguments in a proof $\Pi : \Gamma \vdash t : A$ defines a graph $\mathtt{Graph}(\Pi)$ whose nodes are all pairs (k, u), with u subterm of t and k index of some N-argument of u. $\mathtt{Graph}(\Pi)$ is finite when t is regular. A trace represents the movement of an input information through the infinite unfolding of a tree. Formally, we define a trace as a (finite or infinite) path in the graph $\mathtt{Graph}(\Pi)$.

Definition 3.3 [Trace for well-typed terms in $\mathbb{\Lambda}$] Assume Π is any typing proof of $t \in \mathtt{WTyped}$.

1. A path of Π is any finite or infinite branch $\pi = (i_0, \ldots, i_n, \ldots)$ of Π.

2. Assume $\pi = (t_0, \ldots, t_n, \ldots)$ is a path of Π, finite or infinite. A finite or infinite *trace* τ of π in Π is a list $\tau = ((k_m, t_m), \ldots, (k_n, t_n), \ldots)$ such that for all $i = m, \ldots, n, \ldots$:
   (1) k_i is an index of some N-argument of t_i
   (2) if $i+1$ is an index of τ then (k_{i+1}, t_{i+1}) is connected with (k_i, t_i) in Π.

The starting point of a trace could be different from the starting point of the path to which the trace belongs.

4 The circular system CT-λ

We introduce a condition call *global trace condition* for all proofs Π that the term t is well-typed, then the subset GTC of the set WTyped of well-typed terms of $\mathbb{\Lambda}$ satisfying the global trace condition. GTC is a set of total recursive functionals. We use GTC as a semantics for the set CT-λ of circular λ-terms we will define.

CT-λ will be the regular terms of CT-λ. CT-λ is a decidable set. For the terms of CT-λ we will prove strong normalization, church-rosser for the "safe" part of a term, and the fact that every closed normal term of type N is a numeral. As a consequence, all terms CT-λ will be interpreted as total functionals.

From the N-argument connection we now define the global trace condition and the set of terms of GTC and of CT-λ.

Definition 4.1 [Global trace condition and terms of CT-λ] Assume $\tau = ((k_m, t_m), \ldots, (k_n, t_n), \ldots)$ is a trace of a path $\pi = (t_1, \ldots, t_n, \ldots)$ of $t \in$ WTyped. Assume $i = m, \ldots, n$.

1. τ is progressing in i if $t_i = \mathtt{cond}(f, g)$ for some f, g, and k_i is the index of the first *unnamed* argument the cond-rule, otherwise τ is not progressing in i.
2. t satisfies the global trace condition if for some typing proof Π, of t, for all infinite paths π of t in Π, there is some infinitely progressing path τ in π and Π. GTC is the set of well-typed terms $t \in \mathbb{\Lambda}$ satisfying the global trace condition.
3. CT-λ = GTC $\cap$ Reg: the cyclic λ-terms of CT-λ are all well-typed terms which are regular trees (having finitely many subtrees), and which satisfy the global trace condition.

The notion of global trace condition is defined through a universal quantification on proof $\Pi : \Gamma \vdash t : A$, and proofs Π are possibly infinite objects and they form a non-countable sets. Therefore we would expect that the global trace condition is not decidable. Surprisingly, global trace is decidable in polynomial space in t. The reason is that is some proof satisfies the global trace condition then all proofs do, and for regular terms there is a typing proof which is a finite graph and for which the global trace condition is decidable.

We believe that the global trace condition is decidable quite fast for realistic λ-terms. Another nice feature of the global trace condition is that for $t \in$ GTC we require that there is some proof $\Pi : \Gamma \vdash t : A$ whose infinite paths all have some infinite progressing trace. This proof is almost-left-finite, because all leftmost paths from a sub-proof have no progress point and therefore are finite. We conclude that $t \in$ GTC implies that $t \in$ WTyped: we can assign a unique type to t with a "meaningful proof", that is, an almost-left-finite finite proof.

We include now some examples of cyclic λ-terms. We recall that they have to satisfy GTC, therefore they are almost-left-finite, and by definition they are regular.

4.1 The sum map

A first example of term of CT-λ. We provide an infinite regular term Sum computing the sum on N. In this example, the type superscript N of variables x^N, z^N is omitted.

Example 4.1 *We set* $\mathtt{Sum} = \lambda x.\mathtt{cond}(x, \lambda z.\mathtt{S}(\mathtt{Sum}(x)(z)))$. Sum *is a regular term because it has finitely many subterms:*

$$\mathtt{Sum},\quad \mathtt{cond}(x, \lambda z.\mathtt{S}(\mathtt{Sum}(x)(z))),\quad x,\quad \lambda z.\mathtt{S}(\mathtt{Sum}(x)(z)),\quad \mathtt{S}(\mathtt{Sum}(x)(z)),\quad \mathtt{Sum}(x)(z),\quad \mathtt{Sum}(x),\quad z$$

$\mathtt{Sum}$ *is well-typed by the following derivation, in which we have a back edge from the* † *above to the* † *below.*

$$\dfrac{\dfrac{}{x : N \vdash x : N}\ \mathtt{var} \qquad \dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\vdots}{\vdash \mathtt{Sum} : N, \mathbf{N} \to N \quad (\dagger)}}{x : N, z : \mathbf{N} \vdash \mathtt{Sum} : N, \mathbf{N} \to N}\ \mathtt{weak}}{x : N, z : \mathbf{N} \vdash \mathtt{Sum}(x) : \mathbf{N} \to N}\ \mathtt{ap}_v}{x : N, z : \mathbf{N} \vdash \mathtt{Sum}(x)(z) : N}\ \mathtt{ap}_v}{x : N, z : \mathbf{N} \vdash \mathtt{S}(\mathtt{Sum}(x)(z)) : N}\ \mathtt{S}}{x : N \vdash \lambda z.\mathtt{S}(\mathtt{Sum}(x)(z)) : \mathbf{N} \to N}\ \lambda}{x : N \vdash \mathtt{cond}(x, \lambda z.\mathtt{S}(\mathtt{Sum}(x)(z))) : \mathbf{N} \to N \quad (\spadesuit)}\ \mathtt{cond}}{\vdash \mathtt{Sum} : N, \mathbf{N} \to N \quad (\dagger)}\ \lambda$$

We colored in **old gold** one sequence of atoms $\mathbf{N}$: this is one trace of the proof. The colored trace is the unique infinite trace, it is cyclic and it is infinitely progressing. We marked ♠ the only progress point of the only infinite trace. The progress point is repeated infinitely many times.

4.2 The Iterator

Example 4.2 *We define a term* $\mathtt{It}$ *of* $\mathtt{CT}$-λ *computing the iteration of maps on* N*. The term* $\mathtt{It}$ *is a normal term of type* $(N \to N), N, N \to N$ *such that* $\mathtt{It}(f, a, n) = f^n(a)$ *for all numeral* $n \in \mathtt{Num}$*. We have to solve the equations:*

1. $\mathtt{It}(f, a, 0) \sim a$
2. $\mathtt{It}(f, a, \mathtt{S}(t)) \sim f(\mathtt{It}(f, a, t))$

where f*, a abbreviate* $f^{N \to N}$ *and* a^N*. We solve them with* $\mathtt{It} = \lambda f, a.\mathtt{it}$ *where*

$$\mathtt{it} = \mathtt{cond}(a, \lambda x.f(\mathtt{it}(x))) : N \to N$$

is a term in the context $\Gamma = (f : N \to N, a : N)$*.*

Proposition 4.2 $\mathtt{It} \in \mathtt{Reg} \cap \mathtt{WTyped} \cap \mathtt{GTC} = \mathtt{CT}$-$\lambda$.

The proof above includes a type inference from the term $\mathtt{it}$ to itself. We draw the type inference in the picture below. We have a back edge from the † on the top to the † in the bottom, and we marked ♠ the only progress point, which is cyclically repeated in τ. We abbreviate $\Gamma = (f : N \to N, a : N)$.

$$\dfrac{\dfrac{}{\Gamma \vdash a : N}\ \mathtt{var} \qquad \dfrac{\dfrac{\dfrac{}{\Gamma, x : N \vdash f : N \to N}\ \mathtt{var} \qquad \dfrac{\dfrac{\dfrac{\vdots}{\Gamma \vdash \mathtt{it} : \mathbf{N} \to N \quad (\dagger)}}{\Gamma, x : N \vdash \mathtt{it} : \mathbf{N} \to N}\ \mathtt{weak}}{\Gamma, x : \mathbf{N} \vdash \mathtt{it}(x) : N}\ \mathtt{ap}_v}{\Gamma, x : \mathbf{N} \vdash f(\mathtt{it}(x)) : N}\ \mathtt{ap}}{\Gamma \vdash \lambda x.f(\mathtt{it}(x)) : \mathbf{N} \to N}\ \lambda}{\Gamma \vdash \mathtt{it} : \mathbf{N} \to N \quad (\spadesuit, \dagger)}\ \mathtt{cond}$$

4.3 The Interval Map

A third example. We simulate lists with two variables $\mathtt{nil} : \alpha$ and $\mathtt{cons} : N, \alpha \to \alpha$. We recursively define a notation for lists by $[] = \mathtt{nil}$, $a@l = \mathtt{cons}(a, l)$ and $[a, \boldsymbol{a}] = a@[\boldsymbol{a}]$. We add no elimination rules for lists, though, only the variables $\mathtt{nil}$ and $\mathtt{cons}$. Elimination rules are not required in our example.

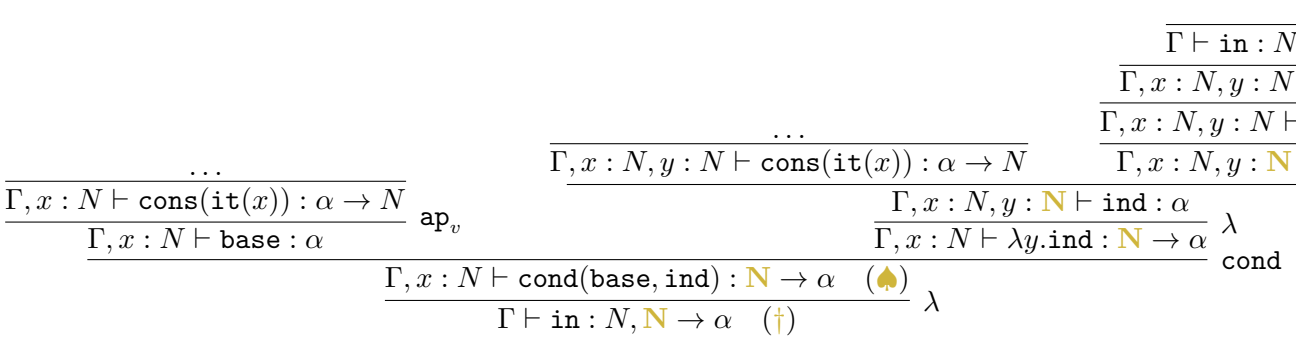

Figure 4.3

Example 4.3 *We will define a term* $\mathtt{In}$ *with one argument* $f : N \to N$ *and three argument* $a, x, y : N$ *(we skip all type superscripts), such that*

$$\mathtt{In}(f, a, n, m) \;\sim\; [f^n(a), f^{n+1}(a), \ldots, f^{n+m}(a)] \;:\; \alpha$$

for all numeral $n, m \in \mathtt{Num}$*. We have to solve the recursive equations*

$$\mathtt{In}(f, a, n, 0) \sim [f^n(a)] \quad \textit{and} \quad \mathtt{In}(f, a, n, \mathtt{S}(m)) \sim f^n(a) @ \mathtt{In}(f, a, \mathtt{S}(n), m).$$

Assume $\mathtt{it} = \mathtt{cond}(a, \lambda x.f(\mathtt{it}(x))) : N \to N$ *is the term in the context* $(f : N \to N, a : N)$ *defined in the previous sub-section: in particular,* $\mathtt{it}(n) \sim f^n(a)$ *for all* $n \in \mathtt{Num}$*. We solve the recursive equation for* $\mathtt{In}$ *with* $\mathtt{In} = \lambda f, a.\mathtt{in}$*, where*

$$\mathtt{in} : N, N \to \alpha$$

is a term in the context $(f : N \to N, a : N)$ *defined by*

$$\mathtt{in} \quad = \quad \lambda x.\mathtt{cond}(\mathtt{base}, \lambda y.\mathtt{ind}),$$

where the base case and the inductive case are

$$\mathtt{base} \quad = \quad [\mathtt{it}(x)] \quad = \quad \mathtt{cons}(\mathtt{it}(x), \mathtt{nil})$$

$$\mathtt{ind} \quad = \quad \mathtt{it}(x) @ \mathtt{in}(\mathtt{S}(x))(y) \quad = \quad \mathtt{cons}(\ \mathtt{it}(x),\ \mathtt{in}(\mathtt{S}(x))(y)\ )$$

Proposition 4.3 $\mathtt{In} \in \mathtt{Reg} \cap \mathtt{WTyped} \cap \mathtt{GTC} = \mathtt{CT}\text{-}\lambda$.

The proof above includes a type inference from the term $\mathtt{in}$ to itself. We draw the type inference in the figure 4.3. In this figure, we include a back edge from the † on the top to the † in the bottom, and we marked ♠ the only progress point, which is cyclically repeated in τ. We abbreviate $\Sigma = (\mathtt{nil} : \alpha, \mathtt{cons} : N, \alpha \to N)$ and $\Gamma = \Sigma, (f : N \to N, a : N)$.

If we carefully examine the term $\mathtt{In}$, we can guess several results of this paper. We have infinitely many nested β-reduction $(\lambda x.\ldots)(\mathtt{S}(x))$. We can remove all of them in a single step. Inside the β-redex number k we obtain a sub-term $\mathtt{it}[\mathtt{S}(x)/x]\ldots[\mathtt{S}(x)/x]$ (substitution repeated k times). The result is $\mathtt{it}[\mathtt{S}^k(x)/x]$. The nested substitution produce infinitely many pairwise different sub-terms $\mathtt{it}(\mathtt{S}^k(x))$ for all $k \in N$. We need infinitely many steps to normalize all $\mathtt{it}(\mathtt{S}^k(x))$ to $f^k(I)$, even if we allow to reduce all $\beta, \mathtt{cond}$-redexes at the same time. Also the normal form is not regular: it contains all terms $f^k(\mathtt{it}(x))$ for $k \in N$, hence infinitely many pairwise different terms.

In conclusion, $\mathtt{In}$ is some term of $\mathtt{CT}\text{-}\lambda$ which can be safely normalized, but which cannot be fully normalized in finite time, not even if we allow infinite parallel reductions without any "safety" restriction. The normal form is produced *only in the limit* and it is *not regular*. If we allow to reduce infinitely many nested existing β-redexes in one step, also the intermediate steps of the infinite reduction of $\mathtt{In}$ are not regular.

5 Subject Reduction for Well-Typed Infinite Lambda Terms

We show the subject reduction for well-typed terms of $\mathbb{\Lambda}$, and also show the global trace condition is preserved by reductions. We first introduce some auxiliary notations for proofs of them.

Let X be a set of variables of the form x^T. We write Γ_X for the sublist of Γ consisting of all $x^T : T$ such that $x^T \in X$. Let Γ and Δ be contexts of $\mathbb{\Lambda}$. The merged context $\Gamma \sharp \Delta$ is defined by $\Gamma \star (\Delta_{\mathrm{FV}(\Delta)\setminus\mathrm{FV}(\Gamma)})$.

Let θ be a renaming, and $S_1 = \Gamma_1 \vdash t_1 : \boldsymbol{B_1} \to N$ and $S_2 = \Gamma_2[\theta] \vdash t_2[\theta] : \boldsymbol{B_2} \to N$ be sequents. Let k_1 and k_2 be indexes of N-arguments of t_1 and t_2, respectively. Then we write $(k_1, S_1) \sim^{\theta}_{\mathtt{idx}} (k_2, S_2)$ (or $(k_1, t_1) \sim^{\theta}_{\mathtt{idx}} (k_2, t_2)$ for short) if they are respectively indexes of named arguments for some y and $\theta(y)$, or are respectively those of unnamed arguments at the same position i in $\boldsymbol{B_1}$ and $\boldsymbol{B_2}$, namely $k_1 = |\Gamma_1| + i$ and $k_2 = |\Gamma_2| + i$, where $|\Gamma_1|$ and $|\Gamma_2|$ are the sizes of Γ_1 and Γ_2. Note that the index equivalent to k_1 is unique (if it exists), namely $(k_1, t_1) \sim^{\theta}_{\mathtt{idx}} (k_2, t_2)$ and $(k_1, t_1) \sim^{\theta}_{\mathtt{idx}} (k_2', t_2)$ implies $k_2 = k_2'$. We write $(k_1, t_1) \sim_{\mathtt{idx}} (k_2, t_2)$ if θ is the identity renaming.

In this paper we do not implicitly identify α-equivalent terms, in order to make simpler to state the global trace condition. For this reason, proofs of this section requires some delicate treatment of variables.

We consider the following restricted λ rule (called λ') as follows:

- λ'-rule. If $\Gamma, x^A : A \vdash b : B$ and $\mathrm{FV}(\Gamma) = \mathrm{FV}(\lambda x.b)$, then $\Gamma \vdash \lambda x^A.b : A \to B$.

The next lemma says that if $\Gamma \vdash t : A$ is provable, then $t : A$ can be shown in a restricted context $\Gamma_{\mathrm{FV}(t)}$ even if the rule λ is restricted to λ'.

Lemma 5.1 *Assume $\Pi : \Gamma \vdash t : A$. Then $\Pi' : \Gamma_{\mathrm{FV}(t)} \vdash t : A$ for some Π' that may have instances of the rule λ' and not have those of the rule λ. Moreover if the global trace condition holds for Π, then it also holds for Π'.*

Lemma 5.2 (Substitution lemma) *Assume that $\Pi_u : \Delta \vdash u : A$ and $\Pi_t : \Gamma, x : A \vdash t : B$ hold. Then there exists Π° such that $\Pi^\circ : \Delta \sharp \Gamma \vdash t[u/x] : B$. Moreover, if both Π_u and Π_t satisfy the global trace condition, then Π° also satisfies it.*

Lemma 5.3
1. *If $\Pi : \Gamma \vdash f(a) : A$ and Π satisfies the global trace condition, then there exist Π_1, Π_2 and B such that $\Pi_1 : \Gamma \vdash f : B \to A$, $\Pi_2 : \Gamma \vdash a : B$, and both Π_1 and Π_2 satisfy the global trace condition.*
2. *If $\Pi : \Gamma \vdash \lambda x^T.b : A$, where $x \notin \mathrm{FV}(\Gamma)$, and Π satisfies the global trace condition, then there exist Π_1 and B such that $\Pi_1 : \Gamma, x : T \vdash b : B$ and $A = T \to B$, and Π_1 satisfies the global trace condition.*
3. *If $\Pi : \Gamma \vdash \mathtt{cond}(f, g) : A$ and Π satisfies the global trace condition, then there exist Π_1, Π_2 and B such that $\Pi_1 : \Gamma \vdash f : B$, $\Pi_2 : \Gamma \vdash g : N \to B$, $A = N \to B$, and both Π_1 and Π_2 satisfy the global trace condition.*
4. *If $\Pi : \Gamma \vdash \mathtt{S}(t) : A$ and $\mathtt{S}(t) \in \mathtt{GTC}$, then there exists Π_1 such that $\Pi_1 : \Gamma \vdash t : N$, $A = N$, and Π_1 satisfies the global trace condition.*

Theorem 5.4 (Subject reduction) *Assume that $\Pi_t : \Gamma \vdash t : A$, Π_t satisfies the global trace condition, and $t \mapsto u$. Then there exists Π_u such that $\Pi_u : \Gamma \vdash u : A$ and Π_u satisfies the global trace condition.*

Using the subject reduction theorem, variable renaming is shown to be admissible in our system.

Proposition 5.5
1. *Let θ be a renaming. If $\Pi : \Gamma \vdash t : A$ and $\Gamma[\theta]$ is a context, then $\Pi' : \Gamma[\theta] \vdash t[\theta] : A$ for some Π'. Moreover, if Π satisfies the global trace condition, so is Π'.*
2. *If $\Pi : \Gamma \vdash t : A$ and t' is an α-equivalent term of t, then $\Pi' : \Gamma \vdash t' : A$ for some Π'. Moreover, if Π satisfies the global trace condition, so is Π'.*

6 Terms with the Global Trace Condition are Finite for Safe Reductions

In this section we prove that for all $n \in N$, every infinite reduction sequence π from some term of $\mathtt{GTC}$ includes only finitely many "0-safe" (safe) reduction steps: from some point on, all reduction steps are unsafe.

From this, we will prove that in GTC the reduction sequences made of only "n-safe" reductions from some point on stop, therefore are finite. Namely, this implies strong normalization for "n-safe" reduction steps: no matter how we reduce within the safe level of a term, eventually no reductions are left, therefore we obtained some safe normal form.

We introduce the property of being "finite for n-safe reductions".

Definition 6.1 Assume $t \in \mathtt{WTyped}$. We say that t is *finite for n-safe reductions* if and only if all infinite reduction paths include only finitely many n-safe reductions (in all infinite reduction paths, from some point on all reductions are *not* n-safe).

In the following, we explicitly write $t[x_1, \ldots, x_n]$, when each free variable in a term t is some x_i, and, under this notation, we also write $t[a_1, \ldots, a_n]$ for $t[a_1/x_1, \ldots, a_n/x_n]$.

It is enough to consider the case of 0-safe reductions, the general case will follow. As we previously said, we abbreviate "0-safe" with "safe". We have to prove that all terms in GTC are finite for safe reductions. For, we define total well-typed terms by induction on the type, as in Tait's normalization proof.

Definition 6.2 [Total well-typed terms of $\mathbb{A}$] Let $t \in \mathtt{WTyped}$ and $t : T$. We define "t is total of type T" by induction on T

1. Let T be any atomic type: $T = N$ or $= \alpha$ for some type variable α. Then t is total of type T if and only if t is finite for safe reductions.
2. Let T be any arrow type $A \to B$. Then f is total of type T if and only if for all total a of type A we have $f(a)$ total of type B.

A term $t[\boldsymbol{x}]$ with free variables $\boldsymbol{x} : \boldsymbol{B}$ is *total by substitution* if and only if: $t[\boldsymbol{v}]$ is total for all totals $\boldsymbol{x} : \boldsymbol{B}$.

Let U be an atomic type, $t[\boldsymbol{x}] : \boldsymbol{A} \to U$, and $\boldsymbol{x} : \boldsymbol{B}$. A *total assignment* of $t[\boldsymbol{x}]$ is $(\boldsymbol{v}, \boldsymbol{a})$, where $\boldsymbol{v} : \boldsymbol{B}$ and $\boldsymbol{a} : \boldsymbol{A}$ are total terms, and $t[\boldsymbol{v}](\boldsymbol{a})$ is total.

By definition, any total term is well-typed, in particular it has exactly one type.

We define a well-founded relation predecessor relation on terms finite for safe reductions and of type N. We recall that $\mapsto$ denotes one reduction step (contraction of a single redex which is a subterm), $\mapsto^*$ denotes zero or more reduction steps and $\mapsto^+$ denotes one or more reduction steps.

Definition 6.3 [The S-order] Assume $t, u \in \mathtt{WTyped}$ and $t, u : N$ (possibly open terms). Then: $(u \prec t) \Leftrightarrow (t \mapsto^* \mathtt{S}(u))$

Example 6.1 *If $t = \mathtt{S}^2(x)$ then $x \prec \mathtt{S}(x) \prec t$. If $t = \mathtt{S}(t)$ then $t \prec t$ and $\prec$ is not well-founded on t:*

We did not prove Church-Rosser yet, therefore we ignore whether all decreasing sequences of $\prec$ have the same However, we can prove that if t is a term finite for safe reductions, then any $\prec$-decreasing sequence from t terminates.

Lemma 6.4 (The $\prec$-order) *Assume $t \in \mathtt{WTyped}$ has type N, and t is finite for safe reductions.*

1. *There is no infinite sequence $\sigma : t = t_0 \mapsto^* \mathtt{S}(t_1) \mapsto^* \mathtt{S}^2(t_2) \mapsto^* \ldots$*
2. *$\prec$ is well-founded on total terms of type N.*

We will prove that if t is not total, then we can assign total terms to the sub-terms of t in an infinite path of a proof $\Pi : \Gamma \vdash t : A$ in a way compatible with traces.

Definition 6.5 [Trace-compatible Assignment] Assume $\pi = (\Gamma_1 \vdash t_1 : A_1, \ldots, \Gamma_n \vdash t_n : A_n, \ldots)$ is any finite or infinite branch of a typing proof Π and $\boldsymbol{v} = (r_1, \ldots, r_n, \ldots)$ is any sequence of total assignments, where, for each i, $r_i = (r_{i,1}, \ldots, r_{i,m_i})$ is one for t_i. $\boldsymbol{v}$ is *trace-compatible* in an index i of π if and only if it satisfies the following condition:

for all j index of an N-argument of t_i, all k index of an N-argument of t_{i+1}, if j is connected to k then:

1. if j progresses to k then $r_{i+1,k} \prec r_{i,j}$
2. if j does not progress to k then $r_{i+1,k} = r_{i,j}$.

$\boldsymbol{v}$ is *trace-compatible* if it is trace-compatible in all i.

Now we will prove that if an infinite branch π of a proof tree $\Pi : \Gamma \vdash t : A$ has a trace-compatible assignment made of total terms, then all traces σ of π progress only finitely many times and the term t is not in $\mathtt{GTC}$ (t does not satisfy the global trace condition).

Proposition 6.6 (Trace assignment) *Assume $\Pi : \Gamma \vdash t : A$ and there is some infinite path π of Π for which we have some total trace-compatible assignment ρ to π.*

1. *Any trace σ in π progresses only finitely many times.*
2. *$t \notin \mathtt{GTC}$.*

We now continue with our Tait's-style proof of Strong Normalization. We check that total terms are closed by reductions, by application and by variables. Any total term is finite for safe reductions.

Lemma 6.7 *Assume $t, u, f, a \in \mathtt{WTyped}$, $n \in N$ and A, B, T are types.*

1. *Let $t : A$ and $t \mapsto^* u$. If t is total, then u is total.*
2. *If $f : A \to B$ and $a : A$ are total terms, then $f(a)$ is total.*
3. *$x^T : T$ is total.*
4. *If t is total then t is finite for safe reductions.*
5. *Let U be any atomic type, and $t[\boldsymbol{x}] : \boldsymbol{A} \to U$ be a term whose all free variables are $\boldsymbol{x} : \boldsymbol{B}$.*
   If for all total $\boldsymbol{u} : \boldsymbol{B}$, $\boldsymbol{a} : \boldsymbol{A}$ the term $t[\boldsymbol{u}]\boldsymbol{a} : U$ is finite for safe reductions, then the term $t[\boldsymbol{x}]$ is total by substitution.

Let $\Pi = (T, \phi)$ be a proof of $\Gamma \vdash t : A$ and l_n be a node of Π, that is, a list of integers $l_n = (e_1, \ldots, e_{n-1}) \in |\Pi|$ for some $n \in N$. We write $\Gamma_n \vdash t_n : A_n = \mathtt{Lbl}(\Pi, l_n)$ for the sequent labelling the node l_n. We want to prove that all terms of $\mathtt{GTC}$ are total by substitution. If we consider the substitution of a variable with itself (a variable is a total term by 6.7.5.), we will deduce that all all terms of $\mathtt{GTC}$ are total, hence finite for safe reductions, hence strongly normalizing for safe reductions.

One problem in the proof of the theorem is that reduction does not commute with the second argument of substitution. That is, if $b \mapsto^* a$ we cannot deduce that $v[b] \mapsto^* v[a]$. The reason is that we could have infinitely many free occurrences of x in v, and it could take an infinite number of steps to reduce to a each b which has been replaced to x in v.

However, we can prove a weaker property: if there is some infinite reduction from $v[a]$, then there is some infinite reduction from $v[b]$, such that if the first reduction has infinitely many safe reduction steps, then the second reduction has infinitely many safe reduction steps, too.

To this aim, we first define a simulation relation from reductions from $v[a]$ to reductions from $v[b]$.

We recall that $\mapsto$, $\mapsto^*$, $\mapsto^+$ denote respectively one, zero or more, one or more reduction steps.

Lemma 6.8 (Safety-preserving Simulation) *Define a binary relation $R \subseteq \mathbb{\Lambda} \times \mathbb{\Lambda}$ by: tRu if and only if there are $v, a, b \in \mathbb{\Lambda}$ and a variable x such that:*

$$(b \mapsto^* a) \quad \text{and} \quad (t = v[a/x]) \quad \text{and} \quad (u = v[b/x])$$

Then R is a safety-preserving simulation on $\mathbb{\Lambda}$ between $\mapsto$ and $\mapsto^+$, namely:

1. *whenever $t, u, t' \in \mathbb{\Lambda}$, tRu and $t \mapsto t'$ then for some $u' \in \mathbb{\Lambda}$ we have $t'Ru'$ and $u \mapsto^+ u'$.*
2. *Besides, if $t \mapsto t'$ is safe then some step in $u \mapsto^+ u'$ is safe, too.*

Now we can prove the required property for reductions with infinitely many safe reduction steps.

Lemma 6.9 (Safe infinite reductions and Substitution) *Let $a \in \mathbb{\Lambda}$, $b \in \mathbb{\Lambda}$ and $v \in \mathbb{\Lambda}$. If $b \mapsto^+ a$ and there is some reduction with with infinitely many safe steps from $v[a/x]$, then there is some reduction with with infinitely many safe steps from $v[b/x]$.*

The Lemma above will be enough to prove our Main Theorem.

Theorem 6.10 (Main Theorem) *Assume* $\Pi : \Gamma \vdash t : A$ *(hence* $t \in \mathtt{WTyped}$*). If* t *is not total by substitution, then* $t \notin \mathtt{GTC}$*, i.e.: there is some infinite path* $\pi = (e_1, e_2, \ldots)$ *of* Π *with no infinite progressing trace.*

6.1 Strong Normalization for safe reductions in CT-λ *and Failure of the Closure under Limit for* CT-λ

From this theorem we derive the strong normalization result for safe reductions on terms of GTC and for all terms which satisfy the global trace condition.

Corollary 6.11 *Assume* $t \in \mathtt{CT}\text{-}\lambda$ *and* $n \in N$*.*

1. *t is total and all reduction sequences from t include only finitely many 0-safe reductions.*
2. *t is total and all reduction sequences from t include only finitely many n-safe reductions.*
3. *t strongly n-safe-normalizes, and in all infinite reductions from t from some point on all reductions are not n-safe.*
4. *All infinite reduction sequences in CT-λ from t have some limit in $\mathbb{\Lambda}$, a limit which is sometimes outside CT-λ.*

Note that CT-λ is *not* **closed under infinite reductions**: the limit of any infinite reduction of any term of CT-λ exists and it is unique in $\mathbb{\Lambda}$, yet it is often not in CT-λ because it is often not regular.

As an example, consider a term $f = \lambda x^N.\mathtt{cond}(x, f(\mathtt{S}x))$, which actually implements the addition. f is regular. f has a type $N, N \to N$ and we trace the second argument $\mathbf{N}$ of f. The unique infinite branch of f is

$$f : N, \mathbf{N} \to N, \quad \mathtt{cond}(x, f(\mathtt{S}x)) : \mathbf{N} \to N, \quad f(\mathtt{S}x) : \mathbf{N} \to N, \quad f : N, \mathbf{N} \to N, \quad \ldots$$

f is in GTC because in this unique infinite branch the trace of the second argument crosses the right-hand-side of a cond infinitely many times. Hence $f \in \mathtt{CT}\text{-}\lambda$. From $f(x) \mapsto \mathtt{cond}(n, f(\mathtt{S}n))$ we deduce that there is a infinite reduction sequence: $f(x) \mapsto \mathtt{cond}(x, f(\mathtt{S}(x))) \mapsto \mathtt{cond}(x, \mathtt{cond}(\mathtt{S}(x), f(\mathtt{S}^2(x)))) \mapsto \cdots$

The limit of $f(x)$ is $\mathtt{cond}(x, \mathtt{cond}(\mathtt{S}(x), \mathtt{cond}(\mathtt{S}^2(x), \cdots)))$, which is *not* regular, since it has infinitely many different subterms $x, \mathtt{S}(x), \mathtt{S}^2(x), \ldots$. Therefore the limit of $f(x)$ is not in CT-λ, even though each term finitely reachable from $f(x)$ is in CT-λ, and f is just the unary algorithm for the addition, because

$$f(\mathtt{S}^n(0))(\mathtt{S}^m(0)) \mapsto \mathtt{cond}(n, \mathtt{cond}(\mathtt{S}(n), \mathtt{cond}(\mathtt{S}^2(n), \cdots)))(\mathtt{S}^m(0)) \mapsto (\mathtt{S}^{n+m})(0)$$

This is an unfortunate but common situation for the limit of a sequence in a topology: finiteness of a notation (which in our case is expressed by *regularity* for trees) is easily lost under limit.

6.2 Closed Safe-Normal terms of CT-λ *of type N are Numeral*

This property is not straightforward. There are closed normal terms of N in $\mathbb{\Lambda}$, even regular terms, which are not numerals. An example is $t = \mathtt{cond}(0, \lambda x^N.1)(t)$ ([**?**], Remark 44, page 33). t has a unique type N. t is normal, because t is not 0 nor successor, therefore t cannot be reduced. However t is no numeral.

Remark that t is regular: the subterms of t are t itself and $\mathtt{cond}(0, \lambda x^N.1)$, 0, $\lambda x^N.1$, 1. However, t is not in GTC. Indeed, the rightmost path of t is an infinite path $t, t, t, t, \ldots$, never crossing a cond, and therefore never progressing.

We check that instead safe-normalization on closed terms of GTC of type N produce a numeral. As an intermediate step in this proof we need the notion of "sound" term.

Definition 6.12 [Sound terms] A term $t \in \mathtt{CT}\text{-}\lambda$ is sound if at least one of the following conditions holds.

1. t is open
2. t has some 0-safe redex
3. $t = \lambda x.u$ or $t = \mathtt{cond}(f, g)$ for some u, f, g and t has an arrow type
4. t is a numeral

Proposition 6.13 (Closed Safe-Normal terms of Type N) *1. All $t \in \mathtt{GTC}$ are sound*
2. If $t \in \mathtt{GTC}$, t is closed and 0-safe-normal and $t : N$ then $t \in \mathtt{Num}$ (t is a numeral)