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

Stefano Berardi

Abstract

We explore the possibility of defining a circular syntax directly for λ-abstraction and simple types, instead of interpreting λ-abstraction through combinators first, then inserting a circular syntax afterwards. We introduce our circular syntax as a fixed point operator defined by cases.

We prove the expected results for this circular simply typed λ-calculus, which we call CT-λ: *(i)* every closed term of type N reduces to some numeral; *(ii)* strong normalization for all reductions sequences outside all fixed points; *(iii)* strong normalization for "fair" reductions; *(iv)* Church-Rosser property; and *(i)* the equivalence between circular syntax and ordinary syntax for Gödel system $\mathbb{T}$.

Our motivation is that terms of CT-λ are much shorter than the equivalent terms written in $\mathbb{T}$. Beside, the circular syntax of CT-λ, using binders instead of combinators, could be more familiar for researchers working in the field of Type Theory.

1 Introduction

We will introduce $\mathbb{A}$, a set of infinite λ-terms with a circular syntax. The types of $\mathbb{A}$ 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 and they play a minor role in our paper.

The terms of $\mathbb{A}$ 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 type variables, then the trees in $\mathbb{A}$ 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

We introduce more sets of terms required as intermediate steps in the definition of GTC and CT-λ.

1. WTyped $\subseteq \mathbb{A}$, the set of well-typed terms, is the of terms having a unique type
2. Reg is the set of terms of $\mathbb{A}$ 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 regular. Terms of GTC denote total functionals.

We will prove that CT-λ is a decidable subset of Reg. CT-λ is a new 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. The 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 lists on $\{1,2\}$ and related notions.

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* $m' \in \mathrm{List}(\{1,2\})$*.*

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 subterm 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* $\mathcal{T}$ *and* $\mathtt{Lbl}(\mathcal{T}, l) = \phi(l)$ *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* $l = (i)$*.*

Remark that 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, i.e.: $\mathtt{Lbl}(\mathcal{T}, \mathtt{nil}) = l$, and with: $\mathcal{T}\lceil (i) = \mathcal{T}_i$ for all $1 \leq i \leq n$.

Now we define the types of $\mathbb{\Lambda}$, then the terms 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, types for short.*
2. `Type` *is the set of simples types.*
3. *We suppose be given a set* `var` *of all pairs* x^T *of a variable name* x *and a type* $T \in$ `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* `S` *has a unique child* $l \star (1)$*.*
3. *a node labeled* `ap`*,* `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-term of* u*, an immediate subterm if it is an immediate subtree.*

Assume $l \in L = \{x^T, \lambda x^T., \mathtt{ap}, 0, \mathtt{S}, \mathtt{cond}\}$. We use the operation $l\mathcal{T}_1 \dots \mathcal{T}_n$ 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 regular terms.

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*.*
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.*
5. *A variable* x^T *is free in* t *if there is some* $l \in t$ *labeled* x^T *in* T*, and no* $m \leq l$ *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. 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 $(t, t, t, \dots, t)$ and $(t, t, t, \dots, 0)$. There is a unique infinite sub-term chain, which is $(t, t, t, \dots)$.

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 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* $\Gamma = ({x_1}^{A_1} : A_1, \dots, x_n^{A_n} : A_n)$ *of pairwise distinct variables, each assigned to its type superscript* $A_1, \dots, A_n \in \mathtt{Type}$*.*
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* $\mathrm{FV}(\Gamma) = \{{x_1}^{A_1}, \dots, {x_n}^{A_n}\}$*. 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* $\Gamma = ({x_1}^{A_1} : A_1, \dots, x_n^{A_n} : A_n)$ *,* $\Gamma' = (x'_1 : A'_1, \dots, x'_n : A'_{n'})$ *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 *if* $x_i^{A_i} = x^T$*. 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 $\phi : \{1, \dots, n\} \to \{1, \dots, n'\}$ such that $x_i = x'_{\phi(i)}$ and $A_i = A'_{\phi(i)}$ for all $i \in \{1, \dots, n\}$. 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 define typing rules for terms of $\mathbb{\Lambda}$ and the subset $\mathtt{WTyped}$ 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 rule $\mathtt{ap}$ 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 importat for checking termination of the computation of $t(u)$, as we will explain later.

We also 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.

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 $t[\vec{x}]$ can be obtained by writing $(\lambda \vec{x}.t[\vec{x}])(\vec{x'})$. 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.

Definition 2.7 (Typing rules of $\mathbb{A}$) *Assume $\Gamma = {x_1}^{A_1} : A_1, \ldots, x_n^{A_n} : A_1$ is a context.*

1. *A rule is a list of $n+1$ typing judgements: $\Gamma \vdash t_1 : A_1, \ldots, \Gamma \vdash t_n : A_n, \Gamma \vdash t : A$. The first n sequents are the premises of the rule, the last sequent is the conclusion of the rule.*
2. *We read the rule above: "if $\Gamma \vdash t_1 : A_1, \ldots, \Gamma \vdash t_n : A_n$ then $\Gamma \vdash t : A$".*

We list the typing rules of $\mathbb{A}$.

1. `weak`*-rule (Weakening). 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 $\Gamma \vdash f : A \to B$ and $(x^A : A) \in \Gamma$ then $\Gamma \vdash f(x^A) : B$.*
5. $\mathtt{ap}_{\neg v}$*-rule. If $\Gamma \vdash f : A \to B$ and $\Gamma \vdash a : A$ and a is not a variable then $\Gamma \vdash f(a) : B$.*
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 $\mathbb{A}$ as

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

A rule is uniquely determined from its conclusion, provided we know whether the rule is a weakening or not.

Proposition 2.8 (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 weakening rules and $\Gamma \vdash t : A = \Gamma' \vdash' t' : A'$ then $r = r'$.*
2. *If r, r' are non-weakening rules and $\Gamma \vdash t : A = \Gamma' \vdash' t' : A'$ then $r = r'$.*
3. *If r is not a weakening and the premises of r are some $\Gamma_1 \vdash t_1 : A_1, \ldots, \Gamma_n \vdash t_n : A_n$, then the list of immediate subterms of t is exactly $t_1, \ldots, t_n$.*

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 $\mathbb{A}$.

This is the formal definition of proof.

Definition 2.9 (Well-typed term of $\mathbb{A}$) *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)

   (3) The children of l in T are labeled $\Delta_1 \vdash u_1 : B_1 = \phi(l\star(1)), \ldots, \Delta_n \vdash u_n : B_n = \phi(l\star(n))$

Then for some rule $r \in \texttt{Rule}$ *of* $\mathbb{\Lambda}$ *we have*

$$\frac{\Delta_1 \vdash u_1 : B_1 \quad \dots \quad \Delta_n \vdash u_1 : B_1}{\Delta \vdash u : B}\ r$$

Eventually we define the well-typed terms of $\mathbb{\Lambda}$.

Definition 2.10 (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 unique* A *and some context* $\Gamma \supseteq \mathrm{FV}(t)$.
3. $\texttt{WTyped}$ *is the set of well-typed* $t \in \mathbb{\Lambda}$.

We provide some examples of well-typed and not well-typed terms. Some term in $\mathbb{\Lambda}$ has no type, like the application $0(0)$ of the non-function 0. Some term in $\mathbb{\Lambda}$ has more than one type. For instance if $t = u(0)$ and $u = \texttt{cond}(t, u)$ we can prove $\vdash t : A$ for all types $A \in \texttt{Type}$. The subterms of t are $\{t, u, 0\}$ and a proof $\Pi :\vdash t : A$ is

$$\dfrac{\dfrac{\dfrac{\dots}{\vdash t : A} \qquad \dfrac{\dots}{\vdash u : N \to A}}{\vdash u : N \to A}\ \texttt{cond} \qquad \dfrac{}{\vdash 0 : N}\ 0}{\vdash t : A}\ \texttt{ap}_{\neg v}$$

Formally, the typing proof $\Pi = (T, \phi) :\vdash t : A$ is defined by $|\Pi| = |t|$ and:

1. $\texttt{Lbl}(\Pi, l) = (\vdash A)$ for all $l \in |t|$ such that $t\lceil l = t$,
2. $\texttt{Lbl}(\Pi, l) = (\vdash N \to A)$ for all $l \in |t|$ such that $t\lceil l = u$,
3. $\texttt{Lbl}(\Pi, l) = (\vdash N)$ for all $l \in |t|$ such that $t\lceil l = 0$.

A term with at least two types has the leftmost branch infinite, as it is the case for t above. We will prove that if *all* subterms of a term have the leftmost branch finite, then the term has at most one type. If the term is also regular then we can decide if the typing proof exists and if it exists we can compute it, in quadratic time in the size of the graph representing the term. Well-typed terms are closed by substitution.

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 2.11 (reduction rules for $\mathbb{\Lambda}$)

1. $\mapsto_\beta$*:* $(\lambda x^A.b)(a) \mapsto_\beta b[a/x]$
2. $\mapsto_{\texttt{cond}}$*:* $\texttt{cond}(f, g)(0) \mapsto_{\texttt{cond}} f$ *and* $\texttt{cond}(f, g)(\mathtt{S}(t)) \mapsto_{\texttt{cond}} g(t)$.
3. $\mapsto$ *is the context and transitive closure of* $\mapsto_\beta$ *and* $\mapsto_{\texttt{cond}}$
4. $t \sim u$ *if and only if there is some* $v \in \mathbb{\Lambda}$ *such that* $t \mapsto v$ *and* $u \mapsto v$.
5. *The* $\texttt{cond}$*-depth of a node* $l = (i_1, \dots, i_n) \in t$ *is the number of* $1 \le h < n$ *such that* $(i_1, \dots, i_h)$ *has label* $\texttt{cond}$ *and* $i_{h+1} = 2$ *(such that* l *occurs in the right-hand-side of a* $\texttt{cond}$*).*
6. *We say that* $t \mapsto_{\texttt{safe}} u$*, or that* t *reduces safely to* u*, if we only reduce on nodes of* $\texttt{cond}$*-depth* $= 0$.
7. *A term is safe-normal if all its redexes (if any) have* $\texttt{cond}$*-depth* > 0.

An example. Let $u = \texttt{cond}.(0, (\lambda z.u)(z))$, where we omitted the type superscript of z because it is irrelevant. Then u is safe-normal, because all redexes in u are of the form $(\lambda z.u)(z)$ and in the right-hand-side of a $\texttt{cond}$. However, the tree form of u has the following branch:

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

This branch is cyclic, infinite, and it includes infinitely many β-redexes.

The reason for forbiding reductions in the right-hand-side $\texttt{cond}(f, g)$ is that through branches crossing the right-hand-side of some $\texttt{cond}$ we will-express fixed-point equations. Reductions on fixed-point

equations can easily loop, and we consider them "unsafe". For this reason, we first considered the minimum possible of reductions of the form: $\mathtt{cond}(f,g)(0)\mapsto_{\mathtt{cond}} f$ and $\mathtt{cond}(f,g)(\mathtt{S}(t))\mapsto_{\mathtt{cond}} g[t/x]$: only those which are applied on maximal `cond`-expression. We considered no reduction inside the second argument of `cond`.

In a second moment, by adding a restriction of *fairness* on reduction strategies, we will be able to recover strong normalization for most "unsafe" reductions.

We include one example of safe `cond`-reductions of a term $v(n)$ to a normal form. Assume $n \in \mathtt{Num}$ is any numeral and $v = \mathtt{cond}(0, v)$. There are only finitely many reductions from $v(n)$ instead, and they are all 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 0 and we stop. Thus, the term $v(n)$ strongly normalizes in $(n+1)$-steps, in fact we have a unique reduction path of length $(n+1)$ from $v(n)$.

3 The trace of the cyclic λ-terms

We introduce the set `GTC` of the set `WTyped` of well-typed terms of $\mathbb{\Lambda}$ satisfying a condition call *global trace condition* for all proofs Π that the term t is well-typed.

In order to define the global trace condition, we have to 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.

We sketch the notion of connection through an example. Assume

$$x_1{}^{A_1} : A_1, x_2{}^{A_2} : A_2 \vdash t[x_1, x_2] : B_3 \to N$$

Then the list of argument types of t is A_1, A_2, B_3. A_1, A_2 are arguments with names x_1, x_2, and B_3 is an unnamed argument. The index of an N-argument of t is any $j \in \{1,2,3\}$ such that $A_j = N$ or $B_j = N$ respectively.

Remark that for an open term $t[x_1, x_2]$ we list as "argument types" also the types A_1, A_2 of the free variables. We motivate this terminology: in a sense, t is an abbreviation of the closed term $t' = \lambda x_1{}^{A_1}, x_2{}^{A_2}.t : (A_1, A_2, B_3 \to N)$, and the argument types of t' are A_1, A_2, B_3 and they include A_1, A_2.

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

1. *The list of argument types of* t *is* $\vec{C} = \vec{A}, \vec{B}$.
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* $C_j = N$.

We now define the atom connection between N-argument of subterms of t in a proof $\Pi : t : \Gamma \vdash A$ of $\mathbb{\Lambda}$. The definition of atom connection for a syntax including the binder λ is the main contribution of this paper. Before providing the general definition, we discuss the notion of connection through examples. We draw in the same color two N-argument which are in connection.

Example 3.1 An example of atom connection for some instance of the `weak`-rule.

$$\frac{x_1 : \mathbf{N}, x_2 : \mathbf{N} \vdash t : \mathbf{N} \to N}{x_1 : \mathbf{N}, x_2 : \mathbf{N}, x_3 : \mathbf{N} \vdash t : \mathbf{N} \to N}\ (\mathtt{weak})$$

Remark that the type N of the variable x_3, colored in **old gold** and introduced by weakening, is in connection with no type in the rule `weak`.

Example 3.2 An example of atom connection for some instance of the $\mathtt{ap}_{\neg v}$-rule. We assume that a is *not* a variable.

$$\frac{x_1 : \mathbf{N}, x_2 : \mathbf{N} \vdash f : \mathbf{N}, \mathbf{N} \to N \qquad x_1 : \mathbf{N}, x_2 : \mathbf{N} \vdash a : N}{x_1 : \mathbf{N}, x_2 : \mathbf{N} \vdash f(a) : \mathbf{N} \to N}\ (\mathtt{ap}_{\neg v})$$

Remark that the first unnamed argument of f (colored in old gold) is in connection with no argument in the rule $\mathtt{ap}_{\neg v}$. The reason is that in the term $f(a)$, the first argument of f receives a value from the value a which is local to the term $f(a)$, does not receive a value from outside the term. However, the first argument of f can be in connection with some argument higher in the typing proof.

We postpone examples with the rule $\mathtt{ap}_v$.

Example 3.3 An example of atom connection for the rule `cond`.

$$\frac{x_1 : \mathbf{N}, x_2 : \mathbf{N} \vdash f : N \qquad x_1 : \mathbf{N}, x_2 : \mathbf{N}, x : \mathbf{N} \vdash g : N}{x_1 : \mathbf{N}, x_2 : \mathbf{N} \vdash \mathtt{cond}(f, g) : \mathbf{N} \to N}\ (\mathtt{cond})$$

Remark that the first unnamed argument of f (colored in old gold) is in connection the type of the free variable x in the second premise (the premise with g), but it is in connection with no argument in the first premise (the premise with f).

3.1 A formal definition of atom connection

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 `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 (Atom connection in a proof of $\mathbb{A}$) *Assume $\vec{A} = A_1, \ldots, A_n$, $\vec{B} = B_1, \ldots, B_m$, $\Gamma = \vec{x} : \vec{A}$, and $\Gamma \vdash t : \vec{B} \to N$.*

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* `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 $t = \mathtt{cond}(f, g)$ 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: `S`, λ, $\mathtt{ap}_v$.
No connection is defined for the rules `var`, 0.

Below we include some examples. We draw in the same color two N-argument which are in connection.

Example 3.4 Atom connection for $\mathtt{ap}_v$. We assume that x is a variable.

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

Remark that the first unnamed argument of f (colored in old gold) in the premise is in connection with the last variable type in the conclusion of the rule.

Example 3.5 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. 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 $\mathbb{A}$) *Assume Π is any typing proof of $t \in \texttt{WTyped}$.*

1. *A path of Π is any finite or infinite branch $\pi = (i_0, \dots, i_n, \dots)$ of Π.*
2. *Assume $\pi = (t_0, \dots, t_n, \dots)$ is a path of Π, finite or infinite. A finite or infinite trace τ of π in Π is a list $\tau = ((k_m, t_m), \dots, (k_n, t_n), \dots)$ such that for all $i = m, \dots, n, \dots$:*
   (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.

4 The circular system CT-λ

We define a subset CT-λ of the set WTyped of well-typed term, by adding the global trace condition and regularity. For the terms of CT-λ we will prove strong normalization, church-rosser for terms of type N, and the fact that every 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 define the global trace condition and the set of terms of CT-λ.

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

1. *τ is progressing in i if $t_i = \texttt{cond}(f, g)$ 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.*
3. *Terms of CT-λ are all well-typed terms which are regular trees (having finitely many subtrees), and 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 should be decidable quite fast for realistic λ-terms.

5 Examples of terms of CT-λ

5.1 The sum map

A first example of term of CT-λ. In this example, the type superscript N of variables x^N, z^N is omitted. We provide an infinite regular term Sum computing the sum on N, defined by $\texttt{Sum} = \lambda x.\texttt{cond}(x, \lambda z.\texttt{S}(\texttt{Sum}(x)(z)))$. This term is regular because it has finitely many subterms:

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

This term is well-typed by the following circular derivation, in which we have a back edge from the † above to the † below. We colored in old gold one sequence of atoms **N** (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.

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

5.2 The Iterator

A second example. 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 5.1 $\mathtt{It} \in \mathtt{Reg} \cap \mathtt{WTyped} \cap \mathtt{GTC} = \mathtt{CT}$-$\lambda$.

Proof The term $\mathtt{It}$ is well-typed and regular by definition. We check the global trace condition.
We follow the unique infinite trace τ of the last unnamed argument N of $\mathtt{It}$ in the unique infinite path π of $\mathtt{It}$. The trace τ moves to the first unnamed argument of $\lambda a.\mathtt{it}$, then to $a : N$ in the context of $\mathtt{it}$. Then the infinite path π moves in this order to: $\lambda x.f(\mathtt{it}(x))$, $f(\mathtt{it}(x))$, $\mathtt{it}(x)$, $\mathtt{it}$ In the meanwhile, τ progresses and moves to the first unnamed argument, then moves to x in the context of $f(\mathtt{it}(x))$, then to x in the context $\mathtt{it}(x)$, and eventually to x in the context of $\mathtt{it}$. Now the infinite path π loops, and at each loop the trace τ progresses once. We conclude that τ infinitely progresses.

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 $\Gamma = (f : N \to N, a : N)$.

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

5.3 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 $[a, \vec{a}] = a@[\vec{a}]$. We add no elimination rules for lists, though, only the variables $\mathtt{nil}$ and $\mathtt{cons}$. Elimination rules are not required in our example.

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

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

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

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

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

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

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

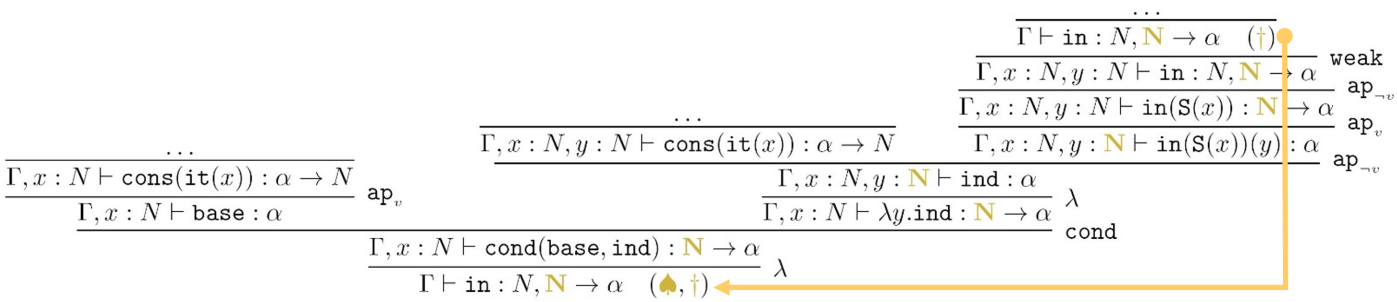

Figure 5.3

Proposition 5.2 $\mathtt{In} \in \mathtt{Reg} \cap \mathtt{WTyped} \cap \mathtt{GTC} = \mathtt{CT}\text{-}\lambda$.

Proof The term $\mathtt{In}$ is well-typed and regular by definition. We check the global trace condition. Any infinite path π either moves to $\mathtt{it}$, for which we already checked the global trace condition, or cyclically moves from $\mathtt{in}$ to $\mathtt{in}$. We follow the unique infinite trace τ of the last unnamed argument N of $\mathtt{In}$ inside this infinite path. The trace τ moves to the last unnamed argument N of $\mathtt{in} : N, \mathbf{N} \to N$, then to the last unnamed argument of $\mathtt{cond}(\ [\mathtt{it}(x)],\ \lambda y.\mathtt{it}(x)@(\lambda x.\mathtt{in})(\mathtt{S}(x))(y)\ )$. In this step τ progresses, and moves to the first unnamed argument of $\lambda y.\mathtt{it}(x)@(\lambda x.\mathtt{in})(\mathtt{S}(x))(y)\ )$, then to $y : N$ in the context of $\mathtt{it}(x)@(\lambda x.\mathtt{in})(\mathtt{S}(x))(y)\ )$. After one $\mathtt{ap}_v$ rule, the trace τ reaches the unique unnamed argument of $(\lambda x.\mathtt{in})(\mathtt{S}(x))$, then the last unnamed argument τ of $\mathtt{in}$. From $\mathtt{in}$ the trace τ loop. Each time τ moves from $\mathtt{in}$ to $\mathtt{in}$ then τ progresses once. We conclude that τ infinitely progresses.

The proof above includes a type inference from the term $\mathtt{in}$ to itself. We draw the type inference in the figure 5.3. 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 $\Gamma = \Sigma, (f : N \to N, a : N)$.

More comments about the term $\mathtt{In}$. 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.

6 Subject reduction for terms of $\mathtt{CT}\text{-}\lambda$ of type N

. . . . . .
(Here we should fill this part) . . . . . .

7 Weak normalization for closed terms of CT-λ of type N

In this section we prove that every closed term of CT-λ (that is, well-typed, regular and with the global trace condition) normalizes with finitely many "safe" steps to some numeral $n \in N$. In the following, we explicitly write $t[x_1, \ldots, x_n]$, when each free variable in t is some x_i, and, under this notation, we also write $t[a_1, \ldots, a_n]$ instead of $t[a_1/x_1, \ldots, a_n/x_n]$. We recall that we denote with Num the set of all numerals, namely the set of all terms of the form $\mathtt{S}^n(0)$ for some $n \in \mathbb{N}$.

We define closed total terms as in Tait's normalization proof, and a subset of them, the values. The only different between values and closed total terms is that if a value has type N then it is a numeral.

Here is an informal definition. Values of type N are exactly the numerals. Total terms t of type N are the closed terms of type N evaluating in at least one reduction path to a numeral. Hence total terms t of type N include values of type N. Closed total terms and values of type $t : \alpha$ (with α type variable) coincide, both are all closed terms. Closed total terms and values of type $t : A \to B$ coincide, both are all the terms mapping values to total terms. An open term is total if all substitutions of free variables with values produce a *closed total* term.

Below is the formal definition of values, closed total terms and total terms.

Definition 7.1 (Values and total term) *We define values and closed total terms on values by induction on types.*

1. *A closed term $t : N$ or is a value if and only if $t \in \mathbb{N}$ (t is some numeral).*
2. *A closed term $t : N$ is closed total if and only if $t \mapsto_{\mathtt{safe}}{}^* u$ for some $u \in \mathbb{N}$.*
3. *Closed terms and values of type $t : \alpha$ (type variable) coincide, both are all closed terms.*
4. *A closed term $t : A \to B$ is a value if and only if $t(a)$ is closed total for any value $a : A$.*
5. *Closed terms and values of type $t : A \to B$ coincide.*

A term $t[\vec{x}] : C$ (i.e., whose free variables are in $\vec{x} : \vec{A}$), is total if and only if $t[\vec{a}]$ is closed total for any values $\vec{a}$ of type $\vec{A}$.

Assume $t[\vec{x}] : \vec{A} \to N$ is a well-typed term of $\mathbb{\Lambda}$ in the context $\Gamma = \vec{x} : \vec{B}$. A *value assignment* $\vec{v}$ for t in Γ is any vector $\vec{v} = \vec{u}, \vec{a} : \vec{B}\vec{A}$ of closed values. We can assign values to the sub-terms in a path of a proof in a way compatible with traces.

Definition 7.2 (Trace-compatible Assignment) *Assume $\pi = (t_1, \ldots, t_n) \in \mathtt{Tree}(t)$ is any chain of immediate subterms from $t_1 = t$, and $\vec{v} = (\vec{v_1}, \ldots, \vec{v_n})$ is any vector of value assigments, one for each t_i. $\vec{v}$ is trace-compatible if and only it satisfies the following condition:*

for all j index of an N-argument of t_i assigned to $\mathtt{S}^a(0)$, all k index of an N-argument of t_{i+1} assigned to $\mathtt{S}^b(0)$, if j is connected to k then:

1. *if j progresses to k then $a = b + 1$*
2. *if j does not progress to k then $a = b$.*

If an infinite path has a trace-compatible assignment, then all traces of the path progress only finitely many times.

Proposition 7.3 (Trace assigment) *Assume $\Pi : \Gamma \vdash t : A$ and π is an infinite path of Π and ρ is a value assignment to π. Assume $n \in \mathbb{N}$.*

1. *If an N argument j of some $t_i \in \pi$ has value $\mathtt{S}^n(0)$ in ρ, then a trace from j progresses at most n times.*
2. *$t \notin \mathtt{GTC}$.*

Proof 1. By definition of trace-compatible assignment, whenever the trace progresses, the numeral n associated to the progressing argument decreases by 1. Whenever the trace does not progress, the numeral n remains the same. Thus, a trace from j progresses at most n times, as we wished to show.

2. By point 1. above, no trace from any argument in any term of the branch π of $\mathtt{Tree}(t)$ progresses infinitely many times. By definition of GTC, we conclude that $t \notin \mathtt{GTC}$.

Closed total terms are closed by safe reductions and by application.

Lemma 7.4 *1. Let $t : A$ be a closed term and $t \mapsto_{\mathtt{safe}} u$. If u is total, then so is t.*

2. Let $f : A \to B$ and $a : A$ be closed total terms. Then so is $f(a)$.

3. Let $t[\vec{x}] : \vec{A} \to N$ be a term, whose all free variables are $\vec{x} : \vec{B}$, and $\vec{u} : \vec{B}$ and $\vec{a} : \vec{A}$ be closed values. If all $t[\vec{u}]\vec{a} : N$ are total, then $t[\vec{x}]$ is total.

Proof 1. We show *point 1.* by induction on A. By the subject reduction property, u has type A.

(1) We first show the first *base case*, namely when $A = N$. By the assumption, u is closed total. By definition of u closed total, we have $t \mapsto_{\mathtt{safe}} u \mapsto_{\mathtt{safe}}{}^{*} n$ for some $n \in \mathtt{Num}$. Hence $t : N$ is total.

(2) We first show the second *base case*, namely when $A = \alpha$. Both t and u are closed terms of type α, therefore both are closed total terms.

(3) We show the *induction case*, namely when $A = (A_1 \to A_2)$. Take arbitrary closed value $a : A_1$. Then we have $t(a) \mapsto_{\mathtt{safe}} u(a)$ and $u(a) : A_2$ is total by the assumption that u is total. Hence by the induction hypothesis $t(a)$ is total. We obtain that $t : A_1 \to A_2$ is total.

2. We show *point 2.* by case reasoning. Assume that$f : A \to B$, $a : A$ are closed total terms, in order to prove that $f(a)$ is closed total.
Assume A is a variable type or an arrow type. Then $a : A$ is a value because values and closed total terms of type A coincide, therefore $f(a) : A$ is closed total by definition of closed total.
Assume $A = N$. Then $a \mapsto_{\mathtt{safe}} n$ for some $n \in \mathtt{Num}$ by definition of closed total. $f(n)$ is closed total by definition of closed total. $f(a)$ is closed total by $f(a) \mapsto_{\mathtt{safe}} f(n)$ and point 2. above.

3. We show *point 3.* by induction on the number $|\vec{A}|$ of elements of $\vec{A}$.

(1) The *base case* $|\vec{A}| = 0$ is immediately shown by the definition.

(2) We show the *induction case*. Let $\vec{A} = A_0, \vec{A'}$. Take arbitrary values $\vec{u} : \vec{B}$, $\vec{a'} : \vec{A'}$, and $a_0 : A_0$. By the assumption, we have that $t[\vec{u}]a_0\vec{a'} : N$ is closed total for all vector of values $\vec{a'}$. Then $t[\vec{u}]a_0 : \vec{A'} \to N$ is total by the induction hypothesis on $\vec{A'} \to N$. By definition $t[\vec{u}] : A_0, \vec{A'} \to N$ is closed total, and so by definition $t[\vec{x}]$ is total, as we wished to show.

Let $\Pi = (T, \phi)$ be a proof of $\Gamma \vdash t : A$ and e be a node of Π, that is, a list $e = (e_1, \ldots, e_n) \in |\Pi| = T$. We write $\Gamma_n \vdash t_n : A_n = \mathtt{Lbl}(\Pi, e)$ for the label of the node e. We want to prove that all terms of $\mathtt{GTC}$ are total.

Theorem 7.5 *Assume $\Pi : \Gamma \vdash t : A$ (hence $t \in \mathtt{WTyped}$). If t is not total, then $t \notin \mathtt{GTC}$, i.e.: there is some infinite path $(e_1, e_2, \ldots)$ of Π with no infinite progressing trace.*

Proof Assume that t is not total and $t : \vec{x} : \vec{D} \vdash \vec{A} \to N$ has a proof $\Pi = (T, \phi)$ in order to show that $t \notin \mathtt{GTC}$. By Proposition 7.3.2. it is enough to prove that Π has some infinite path $\pi = (e_1, e_2, \ldots)$ and some trace-compatible assignment ρ for π. By 3. of Lemma 7.4, there exist closed values $\vec{a} : \vec{A}$ and $\vec{d} : \vec{D}$ such that $t[\vec{d}]\vec{a}$ is not total. By induction on $i \in N$, we construct an index $e_i \in \{1, 2\}$ for each natural number i, and a value assignment $\vec{v_i} = (\vec{d_i}, \vec{a_i})$ such that $t_i = t\lceil(e_1, \ldots, e_{i-1})$ is a subterm of t and $(\vec{v_1}, \ldots, \vec{v_i})$ is a trace-compatible assignment for the node $l_i = (e_1, \ldots, e_{i-1})$. We have to find some $e_i \in \{1, 2\}$ such that:

(i) $t_i = t\lceil l_i$ is a subterm of t, $\Pi((l_i) = \vec{x_i} : \vec{D_i} \vdash t_i : \vec{A_i} \to N$, and e_i is any child node of $(e_1, \ldots, e_i)$ in $|\Pi|$;

(ii) t_i is not total;

(iii) $\vec{d_i} : \vec{D_i}$ and $\vec{a_i} : \vec{A_i}$ are closed values such that $t_i[\vec{d_i}]\vec{a_i}$ is not total;

(iv) the value assignment $\vec{d_1}; \vec{a_1}, \ldots, \vec{d_i}; \vec{a_i})$ is trace compatible. That is, if $i - 1$ is a progress point, namely if t_{i-1} is of the form $\mathtt{cond}(f, g)$ and $e_i = 2$ (hence t_i is g), then $\vec{a_i} = \mathtt{S}(m')\vec{a'}$, $\vec{d_i} = \vec{d_i}m'$.

We first define $(l_1, \vec{d_1}, \vec{a_1})$ for the root node t of Π. In this case $l_1 = \mathtt{nil}$ and $(\vec{d}, \vec{a})$ are values such that $t[\vec{d}](\vec{a})$ is not total. Points (i), (ii), (iii), (iv) are immediate.

Next, assume that $(e_i, \vec{d_i}, \vec{a_i})$ is already constructed. Then we define $(e_{i+1}, \vec{d_{i+1}}, \vec{a_{i+1}})$ by the case analysis on the last rule for the node e_{i+1} in Π.

1. The case of `weak`, namely $\mathtt{Lbl}(\Pi, l_{i+1}) = t[y_{\theta(1)}/x_1, \ldots, y_{\theta(n)}/x_n] : \Delta \vdash \vec{A_i} \to N$ is obtained from $\Gamma \vdash t : A$, where $\Gamma = x_1 : C_1, \ldots, x_n : C_n$, $\Delta = y_1 : B_1, \ldots, y_m : B_m$, $\theta : \{1, \ldots, n\} \to \{1, \ldots, m\}$ is an injection, and $x_i^{C_i} = y_{\theta(i)}^{B_{\theta(i)}}$ for all $1 \leq i \leq n$. By the induction hypothesis and (b), the conclusion $t[y_{\theta(1)}/x_1, \ldots, y_{\theta(n)}/x_n][d_{i,\theta(1)}/y_{\theta(1)}, \ldots, d_{i,\theta(n)}/y_{\theta(n)}]\vec{a_i} : N$ of `weak` is not total, where we have $\vec{d_i} = d_{i,1} \ldots d_{i,m}$. Then we define e_{i+1} as the unique parent node of e_i in the rule, and we also we define $\vec{d_{i+1}}$ and $\vec{a_{i+1}}$ by $d_{i,\theta(1)} \ldots d_{i,\theta(n)}$ and $\vec{a_i}$, respectively. We obtain (i), (ii), and (iii) for $i+1$, as expected. We also have (iv) since the connection, determined by θ, from $(d_{i,1} \ldots d_{i,m}; \vec{a_i})$ to $(\vec{d_{i+1}}; \vec{a_{i+1}}) = (d_{i,\theta(1)}, \ldots d_{i,\theta(n)}; \vec{a_i})$ is trace compatible.
2. The case of `var`-rule, namely $\mathtt{Lbl}(\Pi, l_{i+1}) = \Gamma \vdash x_j : C_j$ for some j, cannot be, because $t_i = [x_j/d_j] = d_j$ is closed total by assumption on $\vec{d}$.
3. The case of 0-rule, namely $\mathtt{Lbl}(\Pi, l_{i+1}) = \Gamma \vdash 0 : N$, cannot be. Indeed, $t_i = 0$ is total because 0 is a numeral.
4. The case of S-rule, namely $\mathtt{Lbl}(\Pi, l_{i+1}) = \Gamma \vdash t_i : N = \Gamma \vdash \mathtt{S}(u) : N$ for some u is obtained from our assumptions on $\Gamma \vdash \mathtt{S}(u) : N$. In this case $\vec{a_i}$ is empty, $t_{i+1} = u$, and by the induction hypothesis $\mathtt{S}(u)[\vec{d_i}]) : N$ is not closed total. Then, by the definition of closed total, $u[\vec{d_i}] = t_{i+1}[\vec{d_i}]$ is not closed total. We define $e_{i+1} = 1$, the index of the unique parent node of $l_{i+1} = (e_1, \ldots, e_n)$ in the S-rule, and we also define $\vec{d_{i+1}}$ and $\vec{a_{i+1}}$ by $\vec{d_i}$ and the empty list, respectively. We obtain (i), (ii), (iii), and (iv) for $i+1$, as expected.
5. The case of the $\mathtt{ap}_{\neg_v}$-rule, namely $\mathtt{Lbl}(\Pi, l_{i+1}) = \Gamma \vdash f[\vec{x}](u[\vec{x}]) : \vec{A} \to N$ for some f, u is obtained from $\Gamma \vdash f[\vec{x}] : B \to \vec{A} \to N$ and $\Gamma \vdash u[\vec{x}] : B$. u is not variable. By the induction hypothesis, $f[\vec{d_i}](u[\vec{d_i}])\vec{a_i} : N$ is not total. We argue by case on the statement: $u[\vec{d_i}] : B$ *is closed total.*

   (1) We first consider the subcase that $u[\vec{d_i}] : B$ *is closed total.* We define $a' : B$ by $a' = n \in \mathtt{Num}$ such that $u[\vec{d_i}] \mapsto_{\mathtt{safe}}{}^* n$ if $B = N$, by $a' = u[\vec{d_i}]$ otherwise. By lemma 7.4.1., a' is a value. Then define $e_{i+1} = 1$, the index of $f[\vec{x}]$, and define $\vec{d_{i+1}} = \vec{d_i}$ and $\vec{a_{i+1}} = a', \vec{a_i}$. Using Lemma 7.4.1., 7.4.2. we obtain (i), (ii), (iii) for $i+1$, as expected. We also have (iv) since the connection from $(\vec{d_i}; \vec{a_i})$ to $(\vec{d_{i+1}}; \vec{a_{i+1}}) = (\vec{d_i}; a', \vec{a_i})$ is trace-compatible: all connected N-argument of $t_i = f(u)[\vec{x}]$ and $t_{i+1}[\vec{x}] = f[\vec{x}]$ are the same, because the only fresh argument of $f[\vec{x}]$ is a' and no argument of $f(u)[\vec{x}]$ is connected to it.

   (2) Next we consider the subcase that $u[\vec{d_i}] : B$ *is not closed total.* By lemma 7.4.3. there is a sequence of values $\vec{a'}$ such that $u[\vec{d_i}]\vec{a'} : N$ is not total. Define $e_{i+1} = 2$, the index of $u[\vec{x}]$, and define $\vec{d_{i+1}} = \vec{d_i}$ and $\vec{a_{i+1}} = \vec{a'}$. We obtain (i), (ii), (iii) for $i+1$, as expected. We also have (iv) since the connection from $(\vec{d_i}; \vec{a_i})$ to $(\vec{d_{i+1}}; \vec{a_{i+1}}) = (\vec{d_i}; \vec{a'})$ is trace compatible: all connected N-argument of $t_i = f(u)[\vec{x}]$ and $t_{i+1} = u[\vec{x}]$ are in $\vec{d_i}$ and therefore are the same.

6. The case of $\mathtt{ap}_v$-rule, namely $\mathtt{Lbl}(\Pi, l_{i+1}) = \Gamma, x^N \vdash f[\vec{x}](x) : \vec{A} \to N$ is obtained from $f[\vec{x}] : \Gamma \vdash N \to \vec{A} \to N$. x is a variable. By the induction hypothesis, $f[\vec{d'}]d\vec{a_i} : N$ is not total, where $\vec{d_i} = \vec{d'}, d$ and d is the value of the variable x^N. Define $e_{i+1} = 1$ as the index of unique parent node of $l_{i+1} = (e_1, \ldots, e_i)$, and also define $\vec{d_{i+1}}$ by $\vec{d'}$ and define $\vec{a_{i+1}}$ by $d, \vec{a_i}$. We obtain (i), (ii), and (iii) for $i+1$, as expected. We also have (iv) since the connection from $(\vec{d_i}; \vec{a_i}) = (\vec{d'}, d; \vec{a_i})$ to $(d_{i+1}; \vec{a_{i+1}}) = (\vec{d'}; d\vec{a_i})$ is trace compatible: all connected N-arguments in $\vec{d_i}, \vec{a_i}$ and $\vec{d_{i+1}}, \vec{a_{i+1}}$ are the same. The only difference between the two assignments is that the value d of one variable x of type N in $t_i[\vec{x}] = f[\vec{x}](x)$ is duplicated to the value d of the first unnamed argument of $f[\vec{x}]$.
7. The case of λ-rule, namely $\mathtt{Lbl}(\Pi, l_{i+1}) = \Gamma \vdash \lambda x^A.u[\vec{x}, x] : A, \vec{A} \to N$ is obtained from $t_{i+1}[\vec{x}, x^A] : \Gamma, x^A \vdash \vec{A} \to N$, where $t_{i+1}[\vec{x}, x] = u[\vec{x}, x]$. By the induction hypothesis, $(\lambda x.(u[\vec{d_i}, x]))a\vec{a'} : N$ is not total, where $\vec{a_i} = a\vec{a'}$. Then, by Lemma 7.4, $t_{i+1}[\vec{d_i}, a]\vec{a'}$ is not total. We define $e_{i+1} = 1$ as the index of unique parent node of $l_{i+1} = (e_1, \ldots, e_i)$, and we also define $\vec{d_{i+1}} = \vec{d_i}, a$ and $\vec{a_{i+1}} = \vec{a'}$. We obtain (i), (ii), (iii) for $i+1$, as expected. We also have (iv) since the connection from $(\vec{d_i}; \vec{a_i}) = (\vec{d_i}; a\vec{a'})$ to $(\vec{d_{i+1}}; \vec{a_{i+1}}) = (\vec{d_i}a; \vec{a'})$ is trace-compatible: all connected argument in $\vec{d_i}, \vec{a_i}$ and $\vec{d_{i+1}}, \vec{a_{i+1}}$ are the same. The only difference between the two assignments is that the value d of the first unnamed argument of $t_i[\vec{x}]$ is moved to the value d of the last variable of type N of $t_{i+1}[\vec{x}, x] = u[\vec{x}, x]$. This is a kind of opposite of the movement we have in the $\mathtt{ap}_v$.

8. The case of cond-rule, namely $\mathtt{Lbl}(\Pi, l_{i+1}) = \mathtt{cond}(f[\vec{x}], g[\vec{x}, x]) : \Gamma \vdash C \to N, \vec{A} \to N$ is obtained from $f[\vec{x}] : \Gamma \vdash \vec{A} \to N$ and $g[\vec{x}] : \Gamma \vdash N, \vec{A} \to N$. By the induction hypothesis, $\mathtt{cond}(f[\vec{d_i}], g[\vec{d_i}, x])m\vec{a'} : N$ is not total, where $\vec{a_i} = m\vec{a'}$ and $m \in \mathtt{Num}$. We argue by cases on m.
   (1) We first consider the *subcase* $m = 0$. Define $e_{i+1} = 1$ the index of parent node whose term is $f[\vec{x}]$, and define $\vec{d_{i+1}} = \vec{d_i}$ and $\vec{a_{i+1}} = \vec{a'}$. We obtain (i), (ii), (iii) for $i + 1$, as expected. We also have (iv) since the connection from $(\vec{d_i}; \vec{a_i}) = (\vec{d_i}; 0\vec{a'})$ to $(\vec{d_{i+1}}; \vec{a_{i+1}}) = (\vec{d_i}; \vec{a'})$ is trace compatible: each argument of t_i is connected to some equal argument of $t_{i+1}[\vec{x}] = f[\vec{x}]$, the first argument of $t_i[\vec{x}]$ disappears but it is connected to no N-argument in $f[\vec{x}]$.
   (2) Next we consider the *subcase* $m = \mathtt{S}(m')$. Define $e_{i+1} = 2$ the index of the parent node whose term is $g[\vec{x}]$, and define $\vec{d_{i+1}} = \vec{d_i}, m'$ and $\vec{a_{i+1}} = \vec{a'}$. We obtain (i), (ii), (iii) for $i + 1$, as expected. We also have (iv) since the connection from $(\vec{d_i}; \vec{a_i}) = (\vec{d_i}; \mathtt{S}(m'), \vec{a'})$ to $(\vec{d_{i+1}}; \vec{a_{i+1}}) = (\vec{d_i}, m'; \vec{a'})$ is trace compatible: each argument of $t_i[\vec{x}]$ is connected to some equal argument of $t_{i+1}[\vec{x}] = g[\vec{x}]$, but for the first unnamend argument of $t_i[\vec{x}]$ which is connected the first unnamend argument of $g[\vec{x}]$. This is fine because in the second premise of a cond the trace progress. This requires that *the value m of the argument decreases by* 1, as indeed it is the case: the new value is m'.

By the above construction, we have an infinite path $\pi = (e_1, e_2, \ldots)$ in $|\Pi|$ and a trace-compatible assignment, as we wished to show.

From this theorem we derive the weak normalization result: every closed term of type N is closed total, therefore by definition it reduces to some numeral for at least one reduction path.

Corollary 7.6 *Assume $t : \Gamma \vdash A$,*

1. *$t \in \mathtt{GTC}$ implies that t is total.*
2. *For any closed $t : N$, there is numeral $n \in \mathtt{Num}$ such that $t \mapsto_{\mathtt{safe}}{}^* n$.*

8 Uniqueness of normal form for closed terms of CT-λ of type N

Incomplete section: uncomment the input command (below in the source code) to make this section visible

9 Infinite reductions

Incomplete section: uncomment the input command (below in the source code) to make this section visible

10 Normalization and Fairness

Incomplete section: uncomment the input command (below in the source code) to make this section visible

11 Equivalence between cyclic and non-cyclic system T

Incomplete section: uncomment the input command (below in the source code) to make this section visible