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

Stefano Berardi, Ugo de' Liguoro, Daisuke Kimura, Makoto Tatsuta

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 $\mathbb{T}$ have been proposed. In this paper 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. A circular syntax using binders instead of combinators could be more familiar for researchers working in the field of Type Theory.

We introduce our circular syntax as a fixed point operator defined by cases through a test on 0. 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 *(v)* the equivalence between circular syntax and ordinary syntax for Gödel system $\mathbb{T}$.

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 of terms, 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 term variables and 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 (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 $\subseteq \mathbb{A}$, the set of well-typed terms, is the of set 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 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 $\mathbb{A}$ 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 $\mathbb{A}$, then the terms and the contexts of $\mathbb{A}$.

Definition 2.3 (Types of $\mathbb{A}$)

1. The types of $\mathbb{A}$ *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. Type *is the set of simples types.*
3. *We suppose be given a set* var*, consisting of all pairs* $x^T = (x, T)$ *of a variable name* x *and a type* $T \in$ Type.

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

Definition 2.4 (Terms of $\mathbb{A}$) *The terms of* $\mathbb{A}$ *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-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 $\mathbb{A}$. If $t, u \in \mathbb{A}$ then $x^T, 0, \lambda x^T.t, \mathtt{ap}(t, u), \mathtt{S}(t), \mathtt{cond}(t, u) \in \mathbb{A}$ are terms with the root labeled l, and with immediate subterms among t, u. Conversely, each $v \in \mathbb{A}$ 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 $t, u \in \mathbb{A}$.

Now we define the regular terms $\mathbb{A}$.

Definition 2.5 (Regular terms of $\mathbb{A}$)

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. Reg *is the set of terms* $t \in \mathbb{A}$ *such that* $\mathtt{SubT}(t)$ *is finite. We call the terms of* 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* Num *for the set of numeral in* $\mathbb{A}$.
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.

Num is the representation inside $\mathbb{A}$ 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 $\mathbb{A}$. 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.

Example 2.1 *An example of regular term which is an infinite tree: the term* $t = \mathtt{cond}(0, t) \in \mathbb{A}$*. 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 $\mathbb{A}$. 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 $\mathbb{A}$)

1. *A context of* $\mathbb{A}$ *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* $A_1, \ldots, A_n \in$ Type.
2. *We denote the empty context with* nil*. We write* 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* $\mathtt{Stat} = \mathtt{Ctxt} \times \mathbb{A} \times \mathtt{Type}$ *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* $t \in \mathbb{A}$ *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* $\mathbb{A}$*, 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* $\mathbb{A}$*, 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.

2.3 Typing Rules

We define typing rules for terms of $\mathbb{A}$ 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 $t[\vec{x}]$ 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. 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 $\mathbb{A}$*: 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* $\mathbb{A}$ *as*

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

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

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 $\Gamma_1 \vdash t_1 : A_1, \ldots, \Gamma_p \vdash t_p : A_p$, 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 $r \in \texttt{Rule}$ of $\mathbb{A}$.

The next definition is the formal definition of proof. We mention that we will introduce a restriction we call *Almost-left-finite* on proof trees. The restriction Almost-left-finite is used to prevent trivial proofs. For instance, we want to forbid a proof cyclically switching the variables x and y in the context of a term, by using the `weak`-rule (Weakening + Exchange rule):

$$\dfrac{\dfrac{\ldots}{(y^U : U, x^T : T) \vdash t : A}\ \texttt{weak}}{(x^T : T, y^U : U) \vdash t : A}\ \texttt{weak}$$

These proof trees are meaningless and should be ruled out. The Almost-Left-Finite condition will ask that all leftmost branches in any sub-proof are finite, in particular that no infinite branch made of `weak`-rules exists.

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 $A \in$* `Type`

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

1. *$\phi(\texttt{nil}) = \Gamma \vdash t : A$.*
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 $r \in$ `Rule` *of $\mathbb{A}$ we have*

$$\dfrac{\Delta_1 \vdash u_1 : B_1 \quad \cdots \quad \Delta_p \vdash u_p : B_p}{\Delta \vdash u : B}\ r$$

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*

We can check that a proof is almost-left-finite if and only if all leftmost paths of all sub-proofs are finite.

In the case of a regular proof, we can decide in polynomial time in the number of sub-proofs whether a proof is almost-left-finite. Here is the algorithm. Suppose a possibly infinite proof has n subproofs. For each sub-proof we follow the leftmost path, after n steps either we reach some rule without premises (either a `var`-rule or a 0-rule), or we loop. Then:

1. If for all sub-proofs we stop then the proof is almost-left-finite
2. If for some sub-proof we loop, then there is infinite leftmost path and the proof is *not* almost-left-finite.

We estimate the computation time to $O(n)$ for each of the n sub-proofs, and the total computation time to $O(n^2)$.

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

Definition 2.10 (Well-typed term of $\mathbb{A}$)

1. *$\Gamma \vdash t : A$ is true if and only if $\Pi : \Gamma \vdash t : A$ for some Π.*

2. $t \in \mathbb{A}$ 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 $\Gamma \supsetsim \mathrm{FV}(t)$.*

3. `WTyped` is the set of well-typed $t \in \mathbb{A}$.

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

Proof By induction on the sum of the length of the leftmost branch in Π and Π'. These lengths are finite because Π and Π' are assumed to be almost-left-finite. Let r be the last rule of Π and r' be the last rule of Π'.

1. Assume $r = \texttt{weak}$. Then Π is obtained from a proof $\Pi_1 : \Gamma_1 \vdash t : A$. By induction hypothesis on Π_1 and Π' we conclude that $A = A'$.
2. Assume $r \neq \texttt{weak}$ and $r' = \texttt{weak}$. Then Π' is obtained from a proof $\Pi'_1 : \Gamma_1 \vdash t : A$. By induction hypothesis on Π and Π'_1 we conclude that $A = A'$.
3. Assume $r \neq \texttt{weak}$ and $r' \neq \texttt{weak}$. Then both r and r' are the typing rule for the outermost constructor of t and for some $h = 0, 1, 2$ the proof Π has premises $\Pi_1, \ldots, \Pi_h$ and the proof Π' has premises $\Pi'_1, \ldots, \Pi'_h$ (for the same h). We distinguish one case for each constructor.

   (1) Assume $t = x^T$. Then $r = r' = \texttt{var}$-rule and $A = A' = T$.

   (2) Assume $t = 0^N$. Then $r = r' = 0$-rule and $A = A' = N$.

   (3) Assume $t = \texttt{S}(u)$. Then $r = r' = \texttt{S}$-rule and $\Pi_1 : \Gamma \vdash u : N$ and $\Pi_2 : \Gamma' \vdash u : N$ and $A = A' = N$.

   (4) Assume $t = f(u)$. Then $r = r' = \texttt{ap}$-rule and $\Pi_1 : \Gamma \vdash f : U \to A$ and $\Pi_2 : \Gamma' \vdash f : U' \to A'$. By induction hypothesis on Π_1 and Π'_1 we deduce that $(U \to A) = (U' \to A')$, in particular that $A = A'$.

   (5) Assume $t = \lambda x^T.b$. Then $r = r' = \lambda$-rule and $\Pi_1 : \Gamma \vdash b : B$ and $\Pi_2 : \Gamma' \vdash b : B'$. By induction hypothesis on Π_1 and Π'_1 we deduce that $B = B'$, in particular that $A = T \to B = T \to B' = A'$.

   (6) Assume $t = \texttt{cond}(f, g)$. Then $r = r' = \texttt{cond}$-rule and $\Pi_1 : \Gamma \vdash f : A$ and $\Pi_2 : \Gamma' \vdash f : A'$. By induction hypothesis on Π_1 and Π'_1 we deduce that $A = A'$.

We provide some examples of well-typed and not well-typed terms. Some term in $\mathbb{A}$ has no type, like the application $0(0)$ of the non-function 0.

Example 2.2 *We provide a term in $\mathbb{A}$ having more than one type. Let $t = u(0)$ and $u = \texttt{cond}(t, u)$. t has an infinite lefmost branch $t, u, t, u, \ldots$, therefore t is not almost-left-finite and it has no almost-left-finite typing proof. Unicity fails and we can prove $\vdash t : A$ for all types $A \in \texttt{Type}$. The subterms of t are $\{t, u, 0\}$ and a proof $\Pi : \emptyset \vdash t : A$ is*

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

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 proved that if *all* subterms of a term have the leftmost branch finite, then the term has at most one type.

Claim (existence of a polynomial-time typing algorithm). We claim without proof that if a term t is regular then we can decide if an almost-left-finite proof $\Pi : \Gamma \vdash A$ exists. If an almost-left-finite proof Π exists then at least one of almost-left-finite proofs is regular and we can compute it. We first check whether the term is almost-left-finite in time $O(n^2)$. If it is not we reject the term. Otherwise we start the standard recursive typing algorithm, until a type has been inferred consistently for each of the finitely many subterms, or some type inconsistency has been found. In the first case we accept the term and we return the time, in the second case we reject the term and we return no type. The computation requires quadratic time in the size of the graph representing the term.

2.4 Reduction Rules

Our goal is to provide a set of well-formed term for $\mathbb{A}$ 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 $\mathbb{A}$.

Definition 2.12 (reduction rules for $\mathbb{A}$)

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 context and transitive closure of* $\mapsto_\beta$ *and* $\mapsto_{\mathtt{cond}}$
4. $t \sim u$ *if and only if there is some* $v \in \mathbb{A}$ *such that* $t \mapsto v$ *and* $u \mapsto v$.
5. *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`*).*
6. *We say that* $t \mapsto_{\mathtt{safe}} u$*, or that* t *reduces safely to* u*, if we only reduce on nodes of* `cond`*-depth* $= 0$.
7. *A term is safe-normal if all its redexes (if any) have* `cond`*-depth* > 0.

Example 2.3 *Let* $u = \mathtt{cond}.(0, (\lambda z.u)(z))$*, 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 are in the right-hand-side of a* `cond`*. However, 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.*

The reason for forbiding reductions in the right-hand-side $\mathtt{cond}(f,g)$ is that through branches crossing the right-hand-side of some `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 required to normalize closed terms of type N. They are the reductions *applied on maximal* `cond`*-expression.* For the moment, we skip all reductions 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 (not all) "unsafe" reductions.

Example 2.4 *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 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 we have a unique reduction path of length* $(n+1)$ *from* $v(n)$ *to* 1.

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.

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, while B_3 is an unnamed argument (it will be denoted by some bound variable in t). 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: in this case all integers $1, 2, 3$ are indexes of some N-argument, in general we can have argument with type different from N.

Remark that for an open term $t[x_1, x_2]$ we list among the "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.

Below is the formal definition of argument types for a term.

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 connection between N-arguments of subterms of t in a proof $\Pi : t : \Gamma \vdash A$ of $\mathbb{A}$. The connection describes how an input moves through the infinite unfolding of the term. 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.

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

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* $\texttt{ap}_{\neg v}$*-rule. We assume that* a *is not a variable.*

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

Remark that the first unnamed argument of f (colored in **old gold**) is in connection with no argument in the rule $\texttt{ap}_{\neg v}$. 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 $\texttt{ap}_v$.

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

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

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 connection

The definition of connection requires to define an injection

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

We set $\texttt{ins}(p, x) = x + 1$ if $x \geq p + 1$, and $\texttt{ins}(p, x) = x$ 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 (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 $\texttt{ap}_{\neg v}$ (i.e., a is not a variable) and $t' = f$ then $k' = \texttt{ins}(n, k)$. If $t' = a$ then $k' = k$ and $k' \leq n$.*
3. *if $t = f(x_i^{A_i})$ is obtained by a rule $\texttt{ap}_v$ and $t' = f$ then $k' = \texttt{ins}(n, k)$or $k' = n + 1$ and $k = i$.*
4. *if $t = \texttt{cond}(f, g)$ and $t' = f$ then $k' = \texttt{ins}(n, k)$. If $t' = g$ then $k' = k$.*

We require $k = k'$ in all other cases, which are: S, *λ.*
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. In the case of the rule $\texttt{ap}_v$, an argument can be connected to two arguments above it.

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

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

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

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

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 $\texttt{ap}_v$-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 $\texttt{Graph}(\Pi)$ whose nodes are all pairs (k, u), with u subterm of t and k index of some N-argument of u. $\texttt{Graph}(\Pi)$ 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 $\texttt{Graph}(\Pi)$.

Definition 3.3 (Trace for well-typed terms in $\mathbb{A}$) *Assume Π is any typing proof of $t \in$* 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 $\mathbb{A}$ 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 $t \in \mathbb{A}$ 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:*

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

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

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

We colored in old gold one sequence of atoms 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* $\texttt{It}$ *of* $\texttt{CT-}\lambda$ *computing the iteration of maps on* N*. The term* $\texttt{It}$ *is a normal term of type* $(N \to N), N, N \to N$ *such that* $\texttt{It}(f, a, n) = f^n(a)$ *for all numeral* $n \in \texttt{Num}$*. We have to solve the equations:*

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

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

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

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

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

Proof The term $\texttt{It}$ 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 $\texttt{It}$ in the unique infinite path π of $\texttt{It}$. The trace τ moves from $\texttt{It} = \lambda f.\lambda a.\texttt{it}$ to the first unnamed argument of the sub-term $\lambda a.\texttt{it}$, then to $a : N$ in the context of $\texttt{it} = \texttt{cond}(a, \lambda x.f(\texttt{it}(x)))$. Then the infinite path π and moves in this order to: $\lambda x.f(\texttt{it}(x))$, $f(\texttt{it}(x))$, $\texttt{it}(x)$, $\texttt{it}$ In the meanwhile, τ progresses from $\texttt{cond}$ to the second argument $\lambda x.f(\texttt{it}(x))$ of $\texttt{cond}$, then moves to x in the context of $f(\texttt{it}(x))$, then to x in the context $\texttt{it}(x)$, and eventually to x in the context of $\texttt{it}$. In this moment the infinite path π loops from $\texttt{it}$ to $\texttt{it}$. At each loop the trace τ progresses once. We conclude that τ infinitely progresses.

The proof above includes a type inference from the term $\texttt{it}$ 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 $\Gamma = (f : N \to N, a : N)$.

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

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

Example 4.3 *We will define a term* 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 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

$$\begin{aligned} \mathtt{base} \quad &= \quad [\mathtt{it}(x)] \quad = \quad \mathtt{cons}(\mathtt{it}(x), \mathtt{nil}) \\ \mathtt{ind} \quad &= \quad \mathtt{it}(x)@\mathtt{in}(\mathtt{S}(x))(y) \quad = \quad \mathtt{cons}(\ \mathtt{it}(x),\ \mathtt{in}(\mathtt{S}(x))(y)\ ) \end{aligned}$$

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

Proof The term In is well-typed and regular by definition. We check the global trace condition. Any infinite path π either moves to it, for which we already checked the global trace condition, or cyclically moves from in to in. We follow the unique infinite trace τ of the last unnamed argument N of In inside this infinite path. The trace τ moves to the last unnamed argument N of $\mathtt{in} : N, N \to N$, 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 in. From in the trace τ loops. Each time τ moves from in to in then τ progresses once. We conclude that τ infinitely progresses.

The proof above includes a type inference from the term 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 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 β, 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, In is some term of CT-λ 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 In are not regular.

$$
\dfrac{\dfrac{\dfrac{\dfrac{\cdots}{\Gamma, x:N \vdash \mathtt{cons}(\mathtt{it}(x)) : \alpha \to N}}{\Gamma, x:N \vdash \mathtt{base} : \alpha}\,\mathtt{ap}_v \qquad \dfrac{\dfrac{\dfrac{\cdots}{\Gamma, x:N, y:N \vdash \mathtt{cons}(\mathtt{it}(x)) : \alpha \to N} \quad \dfrac{\dfrac{\dfrac{\dfrac{\cdots}{\Gamma \vdash \mathtt{in} : N, \mathrm{N} \to \alpha \quad (\dagger)}}{\Gamma, x:N, y:N \vdash \mathtt{in} : N, \mathrm{N} \to \alpha}\,\mathtt{weak}}{\Gamma, x:N, y:N \vdash \mathtt{in}(\mathtt{S}(x)) : \mathrm{N} \to \alpha}\,\mathtt{ap}_{\neg v}}{\Gamma, x:N, y:\mathrm{N} \vdash \mathtt{in}(\mathtt{S}(x))(y) : \alpha}\,\mathtt{ap}_v}{\Gamma, x:N, y:\mathrm{N} \vdash \mathtt{ind} : \alpha}\,\mathtt{ap}_{\neg v}}{\Gamma, x:N \vdash \lambda y.\mathtt{ind} : \mathrm{N} \to \alpha}\,\lambda}{\Gamma, x:N \vdash \mathtt{cond}(\mathtt{base}, \mathtt{ind}) : \mathrm{N} \to \alpha \quad (\spadesuit)}\,\mathtt{cond}}{\Gamma \vdash \mathtt{in} : N, \mathrm{N} \to \alpha \quad (\dagger)}\,\lambda
$$

Figure 4.3

5 An example of "interactive use" of CT-λ

Our thesis is that we can have two very similar definitions f_1, f_2 of the same map ϕ by a regular well-typed term of $\mathbb{A}$. However, f_1 does not satisfies the Global Trace Condition, therefore $f_1 \not\in$ CT-λ is *not* automatically recognized as total, while f_2 satisfies the Global Trace condition and is automatically recognized as total.

Why does it happen? The global trace condition for CT-λ requires to follow the trace of some type -Nvariable, for instance it can follow the trace of the input x^N for $g(x)$, but it cannot follow the trace of a non-variable u used as input for $g(u)$. This means that for the global trace algorithm can follow the trace of x in $(\lambda x^N.g(x))(u)$, while cannot follow the trace of u in $g(u)$. If x is infinitely decreasing the globla trace algorithm can deduce termination in the first case, cannot deduce termination in the second case.

In a sense, the GTC-algorithm is *interactive* for CT-λ: when we believe that the fact that $u : N$ is infinitely decreasing in every infinite computation, we should to assign to u a local name x^N, in order to order to the GTC-algorithm to follow the trace of x^N in any computation. This way of using the algorithm is similar to the way of using the Size-Change-Termination in Neil Jones.

5.1 The Ackermann Function

The Ackermann function is a good example if we want to test the global trace condition in a case in which it is difficult to prove the convergence of a map. We check that the global trace condition can prove that the Ackermann function is convergent provided we use local names to denote the sub-expressions which are infinitely progressing toward 0.

The Ackermann function **m***Ack* : $N, N \to N$ is a map whose convergence can be proved in**PA**, but only if we use induction over formulas with at least two unbounded quantifiers. For a similar reason, **m***Ack* can be defined in system $\mathbb{T}$, but only if we use primitive recursion over terms whose type has degree ≥ 2. **m***Ack* is not primitive recursive.

Here is a fixed point definition of **m***Ack* in CT-λ using a nested conditional.

$$
\begin{aligned}
\mathbf{m}\mathit{Ack}(0, n) &= n + 1 \\
\mathbf{m}\mathit{Ack}(m + 1, 0) &= \mathbf{m}\mathit{Ack}(m, 1) \\
\mathbf{m}\mathit{Ack}(m + 1, n + 1) &= \mathbf{m}\mathit{Ack}(m, \mathbf{m}\mathit{Ack}(m + 1, n))
\end{aligned}
$$

We can prove that ack is convergent by induction on the lexicographic order on (m, n). In the next subsections we try to prove convergence of ack using GTC.

5.2 A term representation ack1 *without* the global trace condition

We first give the first representation ack1 of Ackermann function. In this article, we sometimes write $f(x, y)$ instead of $f(x)(y)$, for readability.

Definition 5.1 (`ack1`) **m** *We define* `ack1` *by the following equation.*

$$\mathtt{ack1} = \lambda\mathbf{mN}.\mathit{cond}(\mathtt{S}\mathbf{N}, \lambda m'.\mathit{cond}(\mathtt{ack1}(m',1), \lambda n'.\mathtt{ack1}(m', \mathtt{ack1}(\mathtt{S}m', n')))(\mathbf{N}))(\mathbf{m})$$

Checking the behavior of `ack1`

$$\begin{aligned}\mathtt{ack1}(0,n) &\mapsto \mathrm{cond}(\mathtt{S}n, \lambda m'.\mathrm{cond}(\mathtt{ack1}(m',1), \lambda n'.\mathtt{ack1}(m', \mathtt{ack1}(\mathtt{S}m', n')))(n))(0)\\ &\mapsto \mathtt{S}n\end{aligned}$$

$$\begin{aligned}\mathtt{ack1}(\mathtt{S}m,0) &\mapsto \mathrm{cond}(\mathtt{S}0, \lambda m'.\mathrm{cond}(\mathtt{ack1}(m',1), \lambda n'.\mathtt{ack1}(m', \mathtt{ack1}(\mathtt{S}m', n')))(0))(\mathtt{S}m)\\ &\mapsto (\lambda m'.\mathrm{cond}(\mathtt{ack1}(m',1), \lambda n'.\mathtt{ack1}(m', \mathtt{ack1}(\mathtt{S}m', n')))(0))(m)\\ &\mapsto \mathrm{cond}(\mathtt{ack1}(m,1), \lambda n'.\mathtt{ack1}(m', \mathtt{ack1}(\mathtt{S}m', n')))(0)\\ &\mapsto \mathtt{ack1}(m,1)\end{aligned}$$

$$\begin{aligned}\mathtt{ack1}(\underline{\mathtt{S}m},\mathtt{S}n) &\mapsto \mathrm{cond}(\mathtt{SS}n, \lambda m'.\mathrm{cond}(\mathtt{ack1}(m',1), \lambda n'.\mathtt{ack1}(m', \mathtt{ack1}(\mathtt{S}m', n')))(\mathtt{S}n))(\underline{\mathtt{S}m})\\ &\mapsto (\lambda m'.\mathrm{cond}(\mathtt{ack1}(m',1), \lambda n'.\mathtt{ack1}(m', \mathtt{ack1}(\mathtt{S}m', n')))(\mathtt{S}n))(\underline{m})\\ &\mapsto \mathrm{cond}(\mathtt{ack1}(m,1), \lambda n'.\mathtt{ack1}(m, \mathtt{ack1}(\mathtt{S}\underline{m}, n')))(\mathtt{S}n)\\ &\mapsto (\lambda n'.\mathtt{ack1}(m, \mathtt{ack1}(\mathtt{S}\underline{m}, n')))(n)\\ &\mapsto \mathtt{ack1}(m, \mathtt{ack1}(\mathtt{S}\underline{m}, n))\end{aligned}$$

The point of `ack1` is the last case. The term $\underline{\mathtt{S}m}$ of the first line and $\mathtt{S}\underline{m}$ of the last line are slightly diffelent: $\underline{\mathtt{S}m}$ is decomposed into $\underline{m}$ by the cond-reduction, then $\mathtt{S}\underline{m}$ is constructed by substituting m' of $\mathtt{S}m'$ by $\underline{m}$.

5.3 `ack1` $: N \to N \to N$ is not in `GTC`

Claim `ack1` $: N \to N \to N$ is well-typed, but does not satisfy GTC in the system.

In the following, weakening is implicitly applied.

$$\dfrac{\dfrac{\dfrac{\dfrac{}{\vdash n : N}}{\vdash \mathtt{S}n : N} \quad \dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\overset{(\dagger1)}{\vdash \mathtt{ack1} : \mathbf{N} \to N \to N}}{m' : N \vdash \mathtt{ack1} : \mathbf{N} \to N \to N}}{m' : \mathbf{N} \vdash \mathtt{ack1}(m') : N \to N}\;(\mathrm{ap_v}) \quad \dfrac{}{\vdash 1 : N}}{m' : \mathbf{N} \vdash \mathtt{ack1}(m',1) : N} \quad \dfrac{\dfrac{\dfrac{\dfrac{\overset{(\dagger2)}{\vdash \mathtt{ack1} : \mathbf{N} \to N \to N}}{m' : N \vdash \mathtt{ack1} : \mathbf{N} \to N \to N}}{m' : \mathbf{N} \vdash \mathtt{ack1}(m') : N \to N}\;(\mathrm{ap_v}) \quad \dfrac{\dfrac{\overset{(\dagger3)}{\vdash \mathtt{ack1} : N \to \mathbf{N} \to N} \quad \dfrac{\dfrac{}{m' : N \vdash m' : N}}{m' : N \vdash \mathtt{S}m' : N}}{m' : N, n' : N \vdash \mathtt{ack1}(\mathtt{S}m') : \mathbf{N} \to N}\;(\mathrm{ap_{\neg v}})}{m' : N, n' : \mathbf{N} \vdash \mathtt{ack1}(\mathtt{S}m', n') : N}\;(\mathrm{ap_v})}{m' : \mathbf{N}, n' : \mathbf{N} \vdash \mathtt{ack1}(m', \mathtt{ack1}(\mathtt{S}m', n')) : N}}{m' : \mathbf{N} \vdash \lambda n'.\mathtt{ack1}(m', \mathtt{ack1}(\mathtt{S}m', n')) : \mathbf{N} \to N}}{n : N, m' : \mathbf{N} \vdash \mathrm{cond}(\mathtt{ack1}(m',1), \lambda n'.\mathtt{ack1}(m', \mathtt{ack1}(\mathtt{S}m', n'))) : \mathbf{N} \to N}\;(\mathrm{cond})}{n : \mathbf{N}, m' : \mathbf{N} \vdash \mathrm{cond}(\mathtt{ack1}(m',1), \lambda n'.\mathtt{ack1}(m', \mathtt{ack1}(\mathtt{S}m', n')))(n) : N}\;(\mathrm{ap_v})}{n : \mathbf{N} \vdash \lambda m'.\mathrm{cond}(\mathtt{ack1}(m',1), \lambda n'.\mathtt{ack1}(m', \mathtt{ack1}(\mathtt{S}m', n')))(n) : \mathbf{N} \to N}}{m : N, n : \mathbf{N} \vdash \mathrm{cond}(\mathtt{S}n, \lambda m'.\mathrm{cond}(\mathtt{ack1}(m',1), \lambda n'.\mathtt{ack1}(m', \mathtt{ack1}(\mathtt{S}m', n')))(n)) : \mathbf{N} \to N}\;(\mathrm{cond})$$

$$\dfrac{\dfrac{m : N, n : \mathbf{N} \vdash \mathrm{cond}(\mathtt{S}n, \lambda m'.\mathrm{cond}(\mathtt{ack1}(m',1), \lambda n'.\mathtt{ack1}(m', \mathtt{ack1}(\mathtt{S}m', n')))(n)) : \mathbf{N} \to N}{m : \mathbf{N}, n : \mathbf{N} \vdash \mathrm{cond}(\mathtt{S}n, \lambda m'.\mathrm{cond}(\mathtt{ack1}(m',1), \lambda n'.\mathtt{ack1}(m', \mathtt{ack1}(\mathtt{S}m', n')))(n))(m) : N}\;(\mathrm{ap_v})}{\vdash \mathtt{ack1} : \mathbf{N} \to \mathbf{N} \to N\ (\dagger)}$$

5.4 Second term representation: `ack2`

We give the second representation `ack2` of Ackermann function.

Definition 5.2 (`ack2`) **m** *We define* `ack2` *by the following equation.*

$$\mathtt{ack2} = \lambda\mathbf{mN}.\mathit{cond}(\mathtt{S}\mathbf{N}, \lambda m'.\mathit{cond}(\mathtt{ack2}(m',1), \lambda n'.\mathtt{ack2}(m', \mathtt{ack2}(\mathbf{m}, n')))(\mathbf{N}))(\mathbf{m})$$

Checking the behavior of `ack2`

We check only the last case.

$$\begin{aligned}
\mathtt{ack2}(\underline{\mathtt{S}m}, \mathtt{S}n) &\mapsto \mathtt{cond}(\mathtt{SS}n, \lambda m'.\mathtt{cond}(\mathtt{ack2}(m',1), \lambda n'.\mathtt{ack2}(m', \mathtt{ack2}(\underline{\mathtt{S}m}, n')))(\mathtt{S}n))(\underline{\mathtt{S}m})\\
&\mapsto (\lambda m'.\mathtt{cond}(\mathtt{ack2}(m',1), \lambda n'.\mathtt{ack2}(m', \mathtt{ack2}(\underline{\mathtt{S}m}, n')))(\mathtt{S}n))(\underline{m})\\
&\mapsto \mathtt{cond}(\mathtt{ack2}(m,1), \lambda n'.\mathtt{ack2}(m, \mathtt{ack2}(\underline{\mathtt{S}m}, n')))(\mathtt{S}n)\\
&\mapsto (\lambda n'.\mathtt{ack2}(m, \mathtt{ack2}(\underline{\mathtt{S}m}, n')))(n)\\
&\mapsto \mathtt{ack2}(m, \mathtt{ack2}(\underline{\mathtt{S}m}, n))
\end{aligned}$$

The point of ack2 is that $\underline{\mathtt{S}m}$ at the first line and the one at the last line are exactly the same.

5.5 $\mathtt{ack2} : N \to N \to N$ is not in GTC (with (η))

Claim $\mathtt{ack2} : N \to N \to N$ is well-typed, but does not satisfy GTC in the system with (η).

$$\dfrac{\dfrac{\dfrac{\vdash n : N}{\vdash \mathtt{S}n : N} \quad \dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{(\dagger 1)}{\vdash \mathtt{ack2} : N \to N \to N}}{m' : N \vdash \mathtt{ack2}(m') : N \to N}\,(\eta) \quad \dfrac{}{\vdash 1 : N}}{m' : N \vdash \mathtt{ack2}(m', 1) : N} \quad \dfrac{\dfrac{\dfrac{\dfrac{(\dagger 2)}{\vdash \mathtt{ack2} : N \to N \to N}}{m' : N \vdash \mathtt{ack2}(m') : N \to N}\,(\eta) \quad \dfrac{\dfrac{\dfrac{(\dagger 3)}{\vdash \mathtt{ack2} : \textcolor{red}{\mathbf{N}} \to \textcolor{olive}{\mathbf{N}} \to N}}{m : \textcolor{red}{\mathbf{N}} \vdash \mathtt{ack2}(m) : \textcolor{olive}{\mathbf{N}} \to N}\,(\eta)}{m : \textcolor{red}{\mathbf{N}}, n' : \textcolor{olive}{\mathbf{N}} \vdash \mathtt{ack2}(m, n') : N}\,(\eta)}{m : \textcolor{red}{\mathbf{N}}, m' : N, n' : \textcolor{olive}{\mathbf{N}} \vdash \mathtt{ack2}(m', \mathtt{ack2}(m, n')) : N}}{m : \textcolor{red}{\mathbf{N}}, m' : N \vdash \lambda n'.\mathtt{ack2}(m', \mathtt{ack2}(m, n')) : \textcolor{olive}{\mathbf{N}} \to N}}{m : \textcolor{red}{\mathbf{N}}, m' : N \vdash \mathtt{cond}(\mathtt{ack2}(m', 1), \lambda n'.\mathtt{ack2}(m', \mathtt{ack2}(m, n'))) : \textcolor{olive}{\mathbf{N}} \to N}\,(\mathrm{cond})}{m : \textcolor{red}{\mathbf{N}}, n : \textcolor{olive}{\mathbf{N}}, m' : N \vdash \mathtt{cond}(\mathtt{ack2}(m', 1), \lambda n'.\mathtt{ack2}(m', \mathtt{ack2}(m, n')))(n) : N}\,(\eta)}{m : \textcolor{red}{\mathbf{N}}, n : \textcolor{olive}{\mathbf{N}} \vdash \lambda m'.\mathtt{cond}(\mathtt{ack2}(m', 1), \lambda n'.\mathtt{ack2}(m', \mathtt{ack2}(m, n')))(n) : N \to N}}{m : \textcolor{red}{\mathbf{N}}, n : \textcolor{olive}{\mathbf{N}} \vdash \mathtt{cond}(\mathtt{S}n, \lambda m'.\mathtt{cond}(\mathtt{ack2}(m', 1), \lambda n'.\mathtt{ack2}(m', \mathtt{ack2}(m, n')))(n)) : N \to N}\,(\mathrm{cond}) \quad \dfrac{}{m : \textcolor{red}{\mathbf{N}} \vdash m : N}}{\dfrac{m : \textcolor{red}{\mathbf{N}}, n : \textcolor{olive}{\mathbf{N}} \vdash \mathtt{cond}(\mathtt{S}n, \lambda m'.\mathtt{cond}(\mathtt{ack2}(m', 1), \lambda n'.\mathtt{ack2}(m', \mathtt{ack2}(m, n')))(n))(m) : N}{\vdash \mathtt{ack2} : \textcolor{red}{\mathbf{N}} \to \textcolor{olive}{\mathbf{N}} \to N \ (\dagger)}}\,(\mathrm{ap})$$

The problem is with the use of (ap). This cuts the trace chasing of $\textcolor{red}{\mathbf{N}}$. The infinite path $(\dagger) \rightsquigarrow (\dagger 1) \rightsquigarrow (\dagger) \rightsquigarrow (\dagger 1) \rightsquigarrow \cdots$ does not contains progressing trace.

5.6 $\mathtt{ack2} : N \to N \to N$ is in GTC

Proposition 5.3 *$\mathtt{ack2} : N \to N \to N$ has a proof that satisfies* GTC *in the system with $(\mathrm{ap_v})$.*

$$\dfrac{\dfrac{\dfrac{\dfrac{\vdash n : N}{\vdash \mathtt{S}n : N} \quad \dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{(\dagger 1)}{\vdash \mathtt{ack2} : \textcolor{blue}{\mathbf{N}} \to N \to N}}{m' : N \vdash \mathtt{ack2} : \textcolor{blue}{\mathbf{N}} \to N \to N}}{m' : \textcolor{blue}{\mathbf{N}} \vdash \mathtt{ack2}(m') : N \to N}\,(\mathrm{ap_v}) \quad \dfrac{}{\vdash 1 : N}}{m' : \textcolor{blue}{\mathbf{N}} \vdash \mathtt{ack2}(m', 1) : N} \quad \dfrac{\dfrac{\dfrac{\dfrac{\dfrac{(\dagger 2)}{\vdash \mathtt{ack2} : \textcolor{blue}{\mathbf{N}} \to N \to N}}{m' : N \vdash \mathtt{ack2} : \textcolor{blue}{\mathbf{N}} \to N \to N}}{m' : \textcolor{blue}{\mathbf{N}} \vdash \mathtt{ack2}(m') : N \to N}\,(\mathrm{ap_v}) \quad \dfrac{\dfrac{\dfrac{\dfrac{\dfrac{(\dagger 3)}{\vdash \mathtt{ack2} : \textcolor{red}{\mathbf{N}} \to \textcolor{olive}{\mathbf{N}} \to N}}{m : N \vdash \mathtt{ack2} : \textcolor{red}{\mathbf{N}} \to \textcolor{olive}{\mathbf{N}} \to N}}{m : \textcolor{red}{\mathbf{N}} \vdash \mathtt{ack2}(m) : \textcolor{olive}{\mathbf{N}} \to N}\,(\mathrm{ap_v})}{m : \textcolor{red}{\mathbf{N}}, n' : \textcolor{olive}{\mathbf{N}} \vdash \mathtt{ack2}(m) : \textcolor{olive}{\mathbf{N}} \to N}}{m : \textcolor{red}{\mathbf{N}}, n' : \textcolor{olive}{\mathbf{N}} \vdash \mathtt{ack2}(m, n') : N}\,(\mathrm{ap_v})}{m : \textcolor{red}{\mathbf{N}}, m' : \textcolor{blue}{\mathbf{N}}, n' : \textcolor{olive}{\mathbf{N}} \vdash \mathtt{ack2}(m', \mathtt{ack2}(m, n')) : N}}{m : \textcolor{red}{\mathbf{N}}, m' : \textcolor{blue}{\mathbf{N}} \vdash \lambda n'.\mathtt{ack2}(m', \mathtt{ack2}(m, n')) : \textcolor{olive}{\mathbf{N}} \to N}}{m : \textcolor{red}{\mathbf{N}}, n : N, m' : \textcolor{blue}{\mathbf{N}} \vdash \mathtt{cond}(\mathtt{ack2}(m', 1), \lambda n'.\mathtt{ack2}(m', \mathtt{ack2}(m, n'))) : \textcolor{olive}{\mathbf{N}} \to N}\,(\mathrm{cond})}{m : \textcolor{red}{\mathbf{N}}, n : \textcolor{olive}{\mathbf{N}}, m' : \textcolor{blue}{\mathbf{N}} \vdash \mathtt{cond}(\mathtt{ack2}(m', 1), \lambda n'.\mathtt{ack2}(m', \mathtt{ack2}(m, n')))(n) : N}\,(\mathrm{ap_v})}{m : \textcolor{red}{\mathbf{N}}, n : \textcolor{olive}{\mathbf{N}} \vdash \lambda m'.\mathtt{cond}(\mathtt{ack2}(m', 1), \lambda n'.\mathtt{ack2}(m', \mathtt{ack2}(m, n')))(n) : \textcolor{blue}{\mathbf{N}} \to N}}{m : \textcolor{red}{\mathbf{N}}, n : \textcolor{olive}{\mathbf{N}} \vdash \mathtt{cond}(\mathtt{S}n, \lambda m'.\mathtt{cond}(\mathtt{ack2}(m', 1), \lambda n'.\mathtt{ack2}(m', \mathtt{ack2}(m, n')))(n)) : \textcolor{blue}{\mathbf{N}} \to N}\,(\mathrm{cond})}{m : \rlap{\textcolor{blue}{\mathbf{N}}}\textcolor{red}{\mathbf{N}}, n : \textcolor{olive}{\mathbf{N}} \vdash \mathtt{cond}(\mathtt{S}n, \lambda m'.\mathtt{cond}(\mathtt{ack2}(m', 1), \lambda n'.\mathtt{ack2}(m', \mathtt{ack2}(m, n')))(n))(m) : N}\,(\mathrm{ap_v})}{\vdash \mathtt{ack2} : \rlap{\textcolor{blue}{\mathbf{N}}}\textcolor{red}{\mathbf{N}} \to \textcolor{olive}{\mathbf{N}} \to N \ (\dagger)}$$

This proof satisfies the global trace condition. Note that:

- The path $(\dagger) \rightsquigarrow (\dagger 1)$ contains a progressing trace $\tau_1 = (\rlap{\textcolor{blue}{\mathbf{N}}}\textcolor{red}{\mathbf{N}}, \rlap{\textcolor{blue}{\mathbf{N}}}\textcolor{red}{\mathbf{N}}, \textcolor{blue}{\mathbf{N}}, \dots, \textcolor{blue}{\mathbf{N}})$.
- The path $(\dagger) \rightsquigarrow (\dagger 2)$ contains a progressing trace $\tau_2 = (\rlap{\textcolor{blue}{\mathbf{N}}}\textcolor{red}{\mathbf{N}}, \rlap{\textcolor{blue}{\mathbf{N}}}\textcolor{red}{\mathbf{N}}, \textcolor{blue}{\mathbf{N}}, \dots, \textcolor{blue}{\mathbf{N}})$.
- The path $(\dagger) \rightsquigarrow (\dagger 3)$ contains a progressing trace $\tau_3' = (\textcolor{olive}{\mathbf{N}}, \dots, \textcolor{olive}{\mathbf{N}})$ and a non-progressing trace $\tau_3 = (\rlap{\textcolor{blue}{\mathbf{N}}}\textcolor{red}{\mathbf{N}}, \rlap{\textcolor{blue}{\mathbf{N}}}\textcolor{red}{\mathbf{N}}, \textcolor{red}{\mathbf{N}}, \dots, \textcolor{red}{\mathbf{N}})$.

Take an infinite path π from this proof. If π passes through $(\dagger 1)$ (or $(\dagger 2)$) infinitely many times, take its infinitely progressing trace by combining τ_1, τ_2, and τ_3. If π passes through $(\dagger 1)$ and $(\dagger 2)$ finitely many times, namely it eventually becomes a loop of $(\dagger)$ and $(\dagger 3)$, take its infinitely progressing trace by combining τ_1, τ_2, and τ_3'.

6 Subject Reduction for Well-Typed Infinite Lambda Terms

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

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

In this section we prove that every infinite reduction sequence of GTC includes only finitely many "safe" reduction steps. This implies that in GTC the sequences made of only "safe" reductions are finite. That is, this implies strong normalization for "safe" reduction steps.

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 $t[a_1, \ldots, a_n]$ instead of $t[a_1/x_1, \ldots, a_n/x_n]$.

We define total well-typed terms by induction on the type, as in Tait's normalization proof.

We recall that $u \in \mathbb{A}$ is *finite for safe reduction* if and only if all infinite reduction sequences from t include only finitely many "safe" reduction steps (Def. 8.7).

Definition 7.1 (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 total assignment $[\vec{x}/\vec{v}]$ is any assignment of total terms to variables of the same type.
A term t is total by substitution if and only if: $t[\vec{x}/\vec{v}]$ is total for all total assignments $[\vec{x}/\vec{v}]$

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.

Definition 7.2 (The $\mathtt{S}$-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 7.1 *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 length.

However, we can prove that a decreasing sequence of $\prec$ terminates on a term finite for safe reductions.

Lemma 7.3 (The $\prec$-order) *Assume $t \in \mathtt{WTyped}$ is finite for safe reductions and of type N.*

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 terms finite for safe reductions and of type N.*

Proof

1. t is finite for safe-reductions, therefore σ only has finitely many reductions. From some $n \in N$ tehre are no more safe reductions from $\mathtt{S}^n(t_n)$. This implies that for some u and $f_1, g_1, \ldots, f_n, g_n$ we have $t_n = u[\mathtt{cond}(f_1, g_1), \ldots, \mathtt{cond}(f_n, g_n)]$ and all reductions from t_n are inside some $g_1, \ldots, g_n$. Then for all $m \in N$, $m \geq n$ we have $\mathtt{S}^m(t_m) = u[\mathtt{cond}(f_1, g'_1), \ldots, \mathtt{cond}(f_n, g'_n)]$ for some $g'_1, \ldots, g'_n$. This is a contradiction when m is larger than the number of symbols in u.
2. If $t_0 \mapsto \mathtt{S}(t_1)$, $t_1 \mapsto \mathtt{S}(t_2)$, ... from some t_0 finite for safe reductions, then there is some infinite sequence $\sigma : t = t_0 \mapsto \mathtt{S}(t_1) \mapsto \mathtt{S}^2(t_2) \mapsto \ldots$, contradicting point 1. above.

We will 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 7.4 (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 Π and $\vec{v} = (\vec{v_1}, \ldots, \vec{v_n}, \ldots)$ is any sequence of total assignments, one for each t_i. $\vec{v}$ 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 $v_j = v_k$*
2. *if j does not progress to k then $v_j \prec v_k$.*

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

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

Proposition 7.5 (Trace assignment) *Assume $\Pi : \Gamma \vdash t : A$ and there is some infinite path π of Π and some a trace-compatible assignment ρ to π.*

1. *Any trace in π progresses only finitely many times.*
2. *$t \notin$ `GTC`.*

Proof

1. By definition of trace-compatible assignment, whenever the trace progresses, the term associated to the progressing argument decreases by $\prec$. Whenever the trace does not progress, the argument remains the same. By $\prec$ well-founded (lemma 7.3.2.), a trace in π progresses only finitely many times, as we wished to show.
2. By point 1. above, no trace from any argument in any term of the branch π of `Tree`(t) progresses infinitely many times. We assumed that there is an infinite path π in Π. By definition of `GTC`, we conclude that $t \notin$ `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 7.6 *Assume $t, u, f, a \in$ `WTyped` 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 T be any atomic type, and $t[\vec{x}] : \vec{A} \to T$ be a term whose all free variables are $\vec{x} : \vec{B}$.*
   If for all total $\vec{u} : \vec{B}$, $\vec{a} : \vec{A}$ the term $t[\vec{u}]\vec{a} : N$ is finite for safe reductions, then the term $t[\vec{x}]$ is total by substitution.

Proof

1. We show *point 1.* by induction on A. We assume that $t : A$ and $t \mapsto u$. and t is total, in order to prove that u is total.By the subject reduction property, u has type A.
   (1) We show the *base case*, namely when $A = N, \alpha$ is an atomic type. By the assumption, u is total. By definition of t total for T atomic, all infinite reductions from t only include finitely many safe reductions. All infinite reductions from u can be extended to some infinite reduction from t, and therefore they only include finitely many safe reductions.
   (2) We show the *induction case*, namely when $A = (A_1 \to A_2)$. Take any arbitrary total term $a : A_1$ in order to prove that $u(a) : A_2$ is total. Then we have $t(a) \mapsto u(a)$ and $t(a) : A_2$ is total by the assumption that t is total. Hence $u(a)$ is total by $t(a) \mapsto u(a)$ and the induction hypothesis on A_2. We conclude that $u : A_1 \to A_2$ is total.
2. If $f : A \to B$, $a : A$ are total terms, then $f(a)$ is total by definition of total.
3. We prove that $x^T : T$ is total. We prove that for all total $\vec{a} : \vec{A}$, if $T = \vec{A} \to U$ then $x(\vec{a}) : U$ is total. The thesis follows if we take $\vec{a} =$ `nil`. We argue by induction on U.

(1) Assume U is atomic. Then every reduction on $x(\vec{a}) : U$ takes place in $\vec{a}$. By definition of total for an atomic type we have to prove that in all infinite reduction sequences from $x(\vec{a}) : U$ there are only finitely many safe reduction. All reductions on $x(\vec{a})$ take place on $\vec{a}$, and since each a_i in $\vec{a}$ is total only finitely many safe reductions are possible, as we wished.

(2) Assume $U = (A_1 \dots A_2)$. By definition of total for an arrow type we have to prove that for all total a we have $x(\vec{a}, a) : A_2$ total. This follows by induction hypothesis on A_2.

4. *We assume that $t : U$ is total in order to prove that t is finite for safe reductions.* We argue by induction on U.

(1) If U is atomic then the thesis is true by definition of total.

(2) Suppose $U = (A_1 \to A_2)$. By point 5. above, $x^{A_1} : A_1$ is total, therefore $t(x) : A_2$ is total and by induction hypothesis on A_2 any infinite reduction sequence from $t(x)$ only includes finitely many safe reductions. Any infinite reduction sequence $\sigma : t = t_0 \mapsto_1 t_1 \mapsto_1 t_2 \mapsto_1 \dots$ from t can be raised to an infinite reduction sequence $\tau : t = t_0(x) \mapsto_1 t_1(x) \mapsto_1 t_2(x) \mapsto_1 \dots \text{from} t(x)$ while preserving the fact that a reduction is safe. We conclude that σ only includes finitely many safe reductions.

5. We show *point 5.* 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 total for all vector of total terms $\vec{a'}$. Then $t[\vec{u}]a_0 : \vec{A'} \to N$ is total for all total $a_0 : A_0$ by the induction hypothesis on $\vec{A'} \to N$. By definition of total we deduce that $t[\vec{u}] : A_0, \vec{A'} \to N$ is total. We conclude that $t[\vec{x}]$ is total by substitution, as we wished to show.

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 $l_n = (e_1, \dots, e_{n-1}) \in |\Pi|$ 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 GTC are total by substitution. If we consider the substitution of a variable with itself (a variable is a total term by 7.6.5.), we will deduce that all all terms of GTC are total, hence finite for safe reductions, hence strongly normalizing for safe reductions.

Theorem 7.7 *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 $\pi = (e_1, e_2, \dots)$ of Π with no infinite progressing trace.*

Proof Assume that t is not total by substitution. Let $\vec{x} : \vec{D}$ and $\vec{A} \to U$ for some *atomic* U be Γ and A, respectively. By Proposition 7.5.2. it is enough to prove that Π has some infinite path $\pi = (e_1, e_2, \dots)$ and some trace-compatible assignment ρ for π. By case 5. of Lemma 7.6, there exist total terms $\vec{a} : \vec{A}$ and $\vec{d} : \vec{D}$ such that there is an infinite reduction from $t[\vec{d}]\vec{a}$ with infinitely many safe reductions. By induction on $i \in \mathbb{N}$, for each i, we construct a path $l_i = (e_1, \dots, e_{i-1}) \in |\Pi|$ and a total assignment $\vec{v_i} = (\vec{d_i}, \vec{a_i})$ such that $(\vec{v_1}, \dots, \vec{v_i})$ is a trace-compatible assignment for the node l_i.

We recall that "trace compatible in i" means: if $i-1$ is a progress point, namely if $t_{i-1} = \mathtt{cond}(f, g)$ and $e_i = 2$ (hence $t_i = g$), then $\vec{a_i} = \mathtt{S}(m'), \vec{a'}$ (the first unnamed argument of $\mathtt{cond}(f, g) : N \to A$ is $S(m')$) while $\vec{d_i} = \vec{d_i}, m'$ (the last named argument of g is 1 unit smaller).

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 total terms such that $t[\vec{d}](\vec{a})$ is not total. Trace compatibility is vacuous because the branch l_1 does not contain two nodes.

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

1. The case of weak, namely $\mathtt{Lbl}(\Pi, l_i) = \Gamma' \vdash t_{i+1} : \vec{A_i} \to N$ is obtained from the induction hypothesis for $\Gamma \vdash t_i : \vec{A_i} \to N$. We have:

(1) $t_i = t_{i+1}$.

(2) $\Gamma = x_1 : D_1, \dots, x_n : D_n$, $\Gamma' = x'_1 : D'_1, \dots, x'_m : D'_m$, and $\Gamma \subsetneq \Gamma'$ with an injection $\phi : \{1, \dots, n\} \to \{1, \dots, m\}$ between contexts.

(3) $x_i = x'_{\phi(i)}$ and $D_i = D'_{\phi(i)}$ for all $i \in \{1, \dots, n\}$.

By the induction hypothesis, $t_i[d_{i,\phi(1)}/x_1, \ldots, d_{i,\phi(n)}/x_n]\vec{a_i} = t_i[d_{i,1}/x'_1, \ldots, d_{i,m}/x'_m]\vec{a_i}$ is not total, where $\vec{d_i} = d_{i,1} \ldots d_{i,m}$. Then we define $l_{i+1} = l_i\star(1)$ taking the unique child node of l_i in Π, and we also define $\overrightarrow{d_{i+1}}$ and $\overrightarrow{a_{i+1}}$ by $d_{i,\phi(1)} \ldots d_{i,\phi(n)}$ and $\vec{a_i}$, respectively. This is an assignment with total terms. $((\vec{d_i}, \vec{a_i}), (\overrightarrow{d_{i+1}}, \overrightarrow{a_{i+1}}))$ is trace compatible for (t_i, t_{i+1}): if two arguments are connected then they are assigned to the same term.

2. The case of $\mathtt{var}$-rule, namely $\mathtt{Lbl}(\Pi, l_i) = \Gamma \vdash x : D$ for some $x_{i,k} : D_{i,k} = x : D \in \Gamma$ cannot be, because $t_i[\vec{d_i}/\vec{x_i}] = d_{i,k}$ is total by assumption on $\vec{d}$.
3. The case of 0-rule, namely $\mathtt{Lbl}(\Pi, l_i) = \Gamma \vdash 0 : N$, cannot be. Indeed, $t_i = 0$ is total because 0 is a numeral.
4. The case of $\mathtt{S}$-rule, namely $\mathtt{Lbl}(\Pi, l_i) = \Gamma \vdash t_i : N = \Gamma \vdash \mathtt{S}(u) : N$ for some u is obtained from our assumptions on $\Gamma \vdash u : N$. In this case $\vec{a_i}$ is empty, $t_{i+1} = u$, and by the induction hypothesis $\mathtt{S}(u)[\vec{d_i}] : N$ has an infinite reduction with infinitely many safe reductions $\mathtt{S}(u) \mapsto_1 \mathtt{S}(u_1) \mapsto_1 \mathtt{S}(u_2) \mapsto_1 \ldots$, all taking place on u. Then, by the definition of total, $u[\vec{d_i}] = t_{i+1}[\vec{d_i}]$ is not total, because the $u \mapsto_1 u_1 \mapsto_1 u_2 \mapsto_1 \ldots$ is an an infinite reduction with infinitely many safe reductions.
   We define $e_i = 1$, the index of the unique child node of $l_{i+1} = l_i\star(1)$ in the $\mathtt{S}$-rule, and we also define $\overrightarrow{d_{i+1}} = \vec{d_i}$ and $\overrightarrow{a_{i+1}} = ()$. This is an assignment with total terms and trace compatible for (t_i, t_{i+1}): if two arguments are connected then they are assigned to the same term.
5. The case of the $\mathtt{ap}_{\neg v}$-rule, namely $\mathtt{Lbl}(\Pi, l_i) = \Gamma \vdash t_i : \vec{A} \to N$, with $t_i = f[\vec{x}](u[\vec{x}])$ for some f and u. The premises of the $\mathtt{ap}_{\neg v}$-rule are $\Gamma \vdash f[\vec{x}] : B \to \vec{A} \to N$ and $\Gamma \vdash u[\vec{x}] : B$, where u is <u>not</u> a variable. By the induction hypothesis, there is some infine reduction from $t_i[\vec{d_i}]\vec{a_i} = f[\vec{d_i}](u[\vec{d_i}])\vec{a_i} : N$ with infinitely many safe-reductions. We argue by case on the statement: $u[\vec{d_i}] : B$ *is total.*

   (1) We first consider the subcase: $u[\vec{d_i}] : B$ *is total.* We define $b = u[\vec{d_i}]$, then $l_{i+1} = l_i\star(1)$, taking the first premise of the rule, and we define $\overrightarrow{d_{i+1}} = \vec{d_i}$ and $\overrightarrow{a_{i+1}} = b, \vec{a_i}$. This is an assignment with total values, and providing an infinite reduction with infinitely many safe reductions, as expected. The connection from $(\vec{d_i}, \vec{a_i})$ to $(\overrightarrow{d_{i+1}}, \overrightarrow{a_{i+1}}) = (\vec{d_i}, b, \vec{a_i})$ 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 b 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 total.* Let $B = \vec{C} \to N$. By lemma 7.6.5. there is a sequence of values $\vec{c} : \vec{C}$ and an infinite reductions from $u[\vec{d_i}]\vec{c} : N$ with infinitely many safe reductions. Define $l_{i+1} = l_i\star(2)$ taking the second premise of the rule, and define $\overrightarrow{d_{i+1}} = \vec{d_i}$ and $\overrightarrow{a_{i+1}} = \vec{c}$. This is an assignment with total terms providing an infinite reductions with infinitely many safe reductions, as expected. The connection from $(\vec{d_i}, \vec{a_i})$ to $(\overrightarrow{d_{i+1}}, \overrightarrow{a_{i+1}}) = (\vec{d_i}, \vec{c})$ 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, and no unnamed arguments are connected.

6. The case of $\mathtt{ap}_v$-rule, namely $\mathtt{Lbl}(\Pi, l_i) = \Gamma \vdash f[\vec{x}](x) : \vec{A} \to N$ is obtained from $\Gamma \vdash f[\vec{x}] : D, \vec{A} \to N$, where $\Gamma = x_1 : D_1, \ldots, x_m : D_m$ and $x_k : D_k = x : D \in \Gamma$. By the induction hypothesis, there is an infinite reduction from $f[\vec{d_i}]d_{i,k}\vec{a_i} : N$ with infinitely many safe reductions, where $\vec{d_i} = (d_{i,1}, \ldots, d_{i,m})$. Define $l_{i+1} = l_i\star(1)$ as the unique child of l_i in Π, and also define $\overrightarrow{d_{i+1}} = \vec{d_i}$ and $\overrightarrow{a_{i+1}} = d_{i,k}, \vec{a_i}$. This is an assignment with total terms providing an infinite reductions with infinitely many safe reductions, as expected. The connection from $(\vec{d_i}, \vec{a_i})$ to $(\overrightarrow{d_{i+1}}, \overrightarrow{a_{i+1}}) = (\vec{d_i}, d_{i,k}, \vec{a_i})$ is trace compatible: all connected N-arguments in $\vec{d_i}, \vec{a_i}$ and $\overrightarrow{d_{i+1}}, \overrightarrow{a_{i+1}}$ are the same. The only difference between the two assignments is that, if $D_k = N$, the value $d_{i,k}$ of type N for the variable x_k in $t_i[\vec{x}] = f[x_1, \ldots, x_k, \ldots, x_m](x_k)$ is duplicated to the term $d_{i,k}$ assigned to the first unnamed argument of $f[\vec{x}]$.
7. The case of λ-rule, namely when a sequent $\mathtt{Lbl}(\Pi, l_{i+1}) = \Gamma \vdash t_i : A, \vec{A} \to N$ with $t_i = \lambda x^A.u[\vec{x}, x]$ is obtained from $\Gamma, x : A \vdash t_{i+1}[\vec{x}, x^A] : \vec{A} \to N$, where $t_{i+1}[\vec{x}, x] = u[\vec{x}, x]$. By the induction hypothesis we have an infinite reduction sequence σ from $t_i[\vec{d_i}]\vec{a_i} = (\lambda x.(u[\vec{d_i}, x]))\vec{a_i} : N$ with infinitely many safe reductions, where $\vec{a_i} = a, \vec{b}$. Define $l_{i+1} = l_i\star(1)$ as the unique child of l_i in Π, and $\overrightarrow{d_{i+1}} = a, \vec{d_i}$ and $\overrightarrow{a_{i+1}} = \vec{b}$. This is an assignment with total terms. The connection from $(\vec{d_i}, \vec{a_i})$ to $(\overrightarrow{d_{i+1}}, \overrightarrow{a_{i+1}}) = (\vec{d_i}, a,\ \vec{b})$ is trace compatible: all connected N-argument of $t_i = \lambda x^A.u[\vec{x}, x]$ and

$t_{i+1} = u[\vec{x}, x]$ are the same, except for the first unnamend argument a of $\vec{a_i}$ which is moved to the last named argument of $u[\vec{x}, x]$, with name x.
We have to prove that there is some infinite reduction from $t_{i+1}[\vec{d_{i+1}}](\vec{a_{i+1}})$ with infinitely many safe reductions. We argue by case.

(1) Suppose all reductions in the infinite reduction σ are inside $\lambda x.(u[\vec{d_i}, x])$ or $\vec{a_i}$. Then there are finitely many safe reductions in $\vec{a_i}$ because $\vec{a_i}$ is total. Thus, there are infinitely many safe reductions on the part of σ taking place on $\lambda x.(u[\vec{d_i}, x])$. We conclude that there is an infinite reduction from $u[\vec{d_i}, x]$ with infinitely many safe reduction. This is true for $u[\vec{d_i}, a]$, too, because reductions and safe reductions are closed by substitution of x with a.

(2) Suppose there is some reduction in σ contracting the first β-redex. Then $(\lambda x.(u[\vec{d_i}, x]))a\vec{b}$ reduces first to some $(\lambda x.v[x])a'\vec{b'}$, then to $v[a']\vec{b'}$, and continues with some reduction including infinitely many safe reductions. From $u[\vec{d_i}, x] \mapsto v[x]$ we deduce $u[\vec{d_i}, a] \mapsto v[a]$, then

$$t_{i+1}[\vec{d_i}, a]\vec{b} \quad = \quad u[\vec{d_i}, a]\vec{b} \quad \mapsto \quad v[a]\vec{b} \quad \mapsto \quad v[a']\vec{b'}$$

Then we continue with some reduction including infinitely many safe reductions.

8. The case of $\mathtt{cond}$-rule, namely a sequent $\mathtt{Lbl}(\Pi, l_i) = \Gamma \vdash t_i : N, \vec{A} \to N$ having $t_i[\vec{x}] = \mathtt{cond}(f[\vec{x}], g[\vec{x}])$, and obtained from $\Gamma \vdash f[\vec{x}] : \vec{A} \to N$ and $\Gamma \vdash g[\vec{x}] : N, \vec{A} \to N$. By the induction hypothesis, there is some infinite reduction σ sequence from $t_i[\vec{d_i}]m\vec{b} = \mathtt{cond}(f[\vec{d_i}], g[\vec{d_i}])m\vec{b} : N$ with infinitely many safe reductions, where $\vec{a_i} = m, \vec{b}$.
Suppose the reduction sequence σ never contracts the leftmost $\mathtt{cond}$-redex, then infinitely many reductions take place on $f[\vec{d_i}]$, m, $\vec{b}$, but not on $g[\vec{d_i}]$ (inside g they would not be safe). m, $\vec{b}$ are total, only finitely many reduction on them are possible, therefore we have a contradiction.
Suppose the reduction sequence contracts the leftmost $\mathtt{cond}$-redex, then $t_i[\vec{d_i}]m\vec{b} \mapsto \mathtt{cond}(u, v)m$”$\vec{b'}$, with m” $= 0, \mathtt{S}(m')$, Then in the first sub-case $\mathtt{cond}(u, v)m$” $\mapsto u\vec{b'}$, in the second sub-case $\mathtt{cond}(u, v)m$” $\mapsto vm'\vec{b'}$. After this $\mathtt{cond}$-reduction we have infinitely many safe reductions. We prove our thesis by cases on m”.

   (1) *Suppose* m” $= 0$. We choose $l_{i+1} = l_i\star(1)$ and $\vec{d_{i+1}} = \vec{d_i}$, $\vec{a_{i+1}} = \vec{b}$. This is an assignment with total terms and $f[\vec{d_{i+1}}](\vec{b})$ reduces to $u\vec{b'}$, then we have infinitely many safe reductions.
   We have trace-compatibility because: 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) *Suppose* m” $= \mathtt{S}(m')$. We choose $l_{i+1} = l_i\star(2)$ and $\vec{d_{i+1}} = \vec{d_i}$, $\vec{a_{i+1}} = m', \vec{b}$. We have trace-compatibility because: 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 unnamed argument m of $t_i[\vec{x}]$ which is connected the first unnamed argument of $g[\vec{x}]$. This is fine because in the second premise of a $\mathtt{cond}$ the trace progresses and $m' \prec m$, because $m \mapsto m$” $= \mathtt{S}(m')$.

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 strong normalization result for safe reductions on terms of $\mathtt{GTC}$ (on all terms which satisfy the global trace condition).

Corollary 7.8 *1. Assume* $\Gamma \vdash t : A$. *Then* $t \in \mathtt{GTC}$ *implies that* t *is total.*
2. Assume $\Gamma \vdash t : A$. *Then* $t \in \mathtt{GTC}$ *implies that* t *strongly safe-normalizes*

Safe-normalization on closed terms of $\mathtt{GTC}$ of type N produce a numeral:

Proposition 7.9 (Closed Safe-Normal terms of Type N) *If* $t \in \mathtt{GTC}$, *t is closed and safe-normal then* $t \in \mathtt{Num}$ *(t is a numeral)*

8 Infinite Church-Rosser for Infinite Lambda Terms

In this section prove a result for the safe part of a term, which we call "*unicity of the safe part of the safe normal form*". By this we mean: for all $t, u, v \in \mathbb{A}$, if $t \mapsto u$ and $t \mapsto v$ and u, v are safe-normal then u, v are equal outside the right-hand side of all $\texttt{cond}$-sub-terms.

Our first idea (wrong) is to prove a full Church-Rosser property for $\mathbb{A}$: for all $t, u, v \in \mathbb{A}$, if $t \mapsto u$ and $t \mapsto v$ then for some $w \in \mathbb{A}$ we have $u \mapsto w$ and $v \mapsto w$. This property is false: for some $t \in \mathbb{A}$, finding a common reduction of u, v takes infinitely many steps. This even in the case $t \in \texttt{CT-}\lambda$, as the next example shows.

Example 8.1 (Failure of Church-Rosser for $\texttt{CT-}\lambda$) *Let $b = \texttt{cond}(x^N, b) : N \to N$ a normal form and $t = (\lambda x^N.b)(r) : N \to N$, where $r = (\lambda x^N.x^N)(3)$ is some redex.*

We have $b[r/x](n) \mapsto r \mapsto 3$ for all numerals n, therefore t and $\lambda\_.3$ are extensionally equal, however $t \not\mapsto \lambda\_.3$. We have $t \in \texttt{CT-}\lambda$. Indeed,

1. *t is regular by construction.*
2. *We have $t \in \texttt{GTC}$, because the unique infinite path of t is $t, \lambda x^N.b, b, b, b, \ldots$, and the unique unnamed argument of $b : N \to N$ in the path progresses infinitely many times.*

Now consider the reductions: $t \mapsto b[r/x^N]$ and $t \mapsto (\lambda x^N.b)(3)$. We expect $b[3/x^N]$ as common normal form. But we have $b[r/x^N] = \texttt{cond}(r, b[r/x^N]$, that is, we have replicated the redex r infinitely many times in $b[r/x^N]$. Therefore to reduce $b[r/x^N]$ to $b[3/x^N]$ takes infinitely many steps, and for no finite reduction we have $b[r/x^N] \mapsto b[3/x^N]$.

We proved that Church-Rosser is false for $\texttt{CT-}\lambda$.

In order to recover Church-Rosser we have to consider a more general notion of reduction $\mapsto_X$, which allow to reduce *infinitely many redexes in one step*: all those in a *decidable* set X of redexes of t, and possibly all current redexes This reduction can generate new redexes of course.

We will prove that $\mapsto_X$ is confluent: namely, we will prove that if $t \mapsto_X u$ and $t \mapsto_Y v$, then for some w, Z, T we will have $u \mapsto_Z w$ and $v \mapsto_T w$. We call this property Infinite Church-Rosser. Infinite Church-Rosser will imply the unicity of the safe part of the safe normal form.

Our first problem is that infinite reductions can easily loop, therefore Infinite Church-Rosser is stated for the set $\mathbb{A}_\perp$ of terms with possibly *undefinite* subterms. This is not an obstacle for the goals of this paper, as we will see.

We formally state Infinite Church-Rosser as follows. "*For all $t, u, v \in \mathbb{A}_\perp$, all decidable sets X, Y of redexes of t, if $t \mapsto_X u$ and $t \mapsto_Y v$, then for some $w \in \mathbb{A}_\perp$, some decidable set Z, T of redexes of u, v we have $u \mapsto_Z w$ and $v \mapsto_T w$ and $t \mapsto_{X \cup Y} w$*".

We represent a decidable set X of positions of redexes, and in fact any decidable set of sub-terms by a map $\phi_X : |t| \to \{\text{true}, \text{false}\}$ such that $l \in X$ if and only if $\phi_X(l) = \text{true}$. Our first step is to precise how ϕ_X changes when redexes in X are moved or duplicated.

Definition 8.1 (Substitution, subterms and labels) *Suppose $t, u \in \mathbb{A}_\perp$ and X is a decidable set of redexes of t and Y a decidable set of redexes of u.*

1. *$Z = [Y/x]$ is a a decidable set of redexes of $t[u/x]$ defined as: for all $l \in |t|$, $m|u|$: if l is a free occurrence of x we set $\phi_Z(l{\star}m) = \phi_Y(m)$, otherwise $\phi_Z(l{\star}m) = \text{false}$.*
2. *$X[Y/x] = X \cup [Y/x]$.*
3. *If $n = 0, 1, 2$ and $t = c(t_1) \ldots (t_n)$ and X a decidable set of redexes of t, then for all $1 \leq i \leq n$ we define a set X_i of redexes in t_i by $\phi_{X_i}(l) = \phi_X((i){\star}l)$.*
4. *If $n = 0, 1, 2$ and $t = c(t_1) \ldots (t_n)$ and for all $1 \leq i \leq n$ X_i is a decidable set of redexes of t_i, then we define a set X of redexes in t by $\phi_X(\texttt{nil}) = \text{false}$ and $\phi_X((i){\star}l) = \phi_{X_i}(l)$.*

We define the *unique* $u \in \mathbb{A}_\perp$ such that $t \mapsto_X u$ as the limit of a map $\rho(t, X, n)$ for $n \to \infty$. $\rho(t, X, n)$ starts with the undefined value $\perp$, then either holds the value $\perp$ forever, or at some step the root of the term $\rho(t, X, n)$ becomes some constructor of $\mathbb{A}$ and never changes again. At each step, if t itself is a redex in the set X then we reduce it, if t is not a redex in X then we move to the subterms of t. In both case we update X accordingly to some set of labels X'. We update any other set Z of labels in t in the same way to some Z'. We introduce a map $\sigma(t, X, n, Z)$ computing the value for a set Z of redexes after n steps. In particular we have $X' = \sigma(t, X, 1, X)$.

Definition 8.2 (Infinite reductions) *Assume $t \in \mathbb{A}_\perp$ and X, Z are decidable sets of redexes of t. Let $X_1, Y_1, \dots$ as in Def. 8.1.*

We set $\rho(t, X, 0) = \perp$. If $\phi_X(t) = \text{true}$, then we set:

1. *If $t = (\lambda x^T.b)(a)$, then*
   (1) $\rho(t, X, n+1) \quad\;\; = \rho(b[a/x], X', n)$
   (2) $\sigma(t, X, n+1, Z) = \sigma(b[a/x], X, n,\ Z')$
   where $Z' = (Z_1)_1[Z_2/x]$.
2. *If $t = \mathtt{cond}(f, g)(0)$, then*
   (1) $\rho(t, X, n+1) \quad\;\; = \rho(f, X', n)$
   (2) $\sigma(t, X, n+1, Z) = \sigma(f, X, n, Z')$
   where $Z' = (Z_1)_1$.
3. *If $t = \mathtt{cond}(f, g)(\mathtt{S}(u))$, then*
   (1) $\rho(t, X, n+1) \quad\;\; = \rho(g(u), X', n)$
   (2) $\sigma(t, X, n+1, Z) = \sigma(g(u), X, n, Z')$
   where $Z' = \mathtt{ap}((Z_1)_2, (Z_2)_1)$.

Assume $\phi_X(t) = \text{false}$ and $t = c(t_1)\dots(t_h)$ for some $h = 0, 1, 2$ some $t_1, \dots, t_h \in \mathbb{A}_\perp$. Then we set

1. *$\rho(t, X, n+1) \quad\;\; = c(\rho(t_1, X_1, n)\dots(\rho(t_h, X_h, n))$*
2. *$\sigma(t, X, n+1, Z) = c(\sigma(t_1, X_1, n, Z_1)\dots\sigma(t_h, X_h, n, Z_h))$ and $Z' = c(Z_1)\dots(Z_h) = Z$.*

We define $\rho(t, X) = \lim_{n\to\infty} \rho(t, X, n)$ and $\sigma(t, X, Z) = \lim_{n\to\infty} \sigma(t, X, n, Z)$ and $t \mapsto_X \rho(t, X)$.

Proposition 8.3 (Reduction and union) *Suppose $t, u \in \mathbb{A}_\perp$ and X are decidable sest of redexes of t and Y are decidable sets of redexes of u and $l \in |t|$. Let $X', Y', \dots$ as in Def. 8.1.*

1. *$[Z \cup T/x] = [Z/x] \cup [T/x]$*
2. *$(X \cup Y)[Z \cup T/x] = X[Z/x] \cup Y[T/x]$*
3. *$(X \cup Y)_l = X_l \cup Y_l$*
4. *$c(X_1 \cup Y_1)\dots(X_n \cup Y_n) = c(X_1)\dots(X_n) \cup c(Y_1)\dots(Y_n)$*
5. *$(X \cup Y)' = X' \cup Y'$*

We will prove that for all $t, u, v \in \mathbb{A}_\perp$, all decidable sets X, Y of redexes of t we have $\rho(\rho(t, X), Z) = \rho(t, X \cup Y)$ for $Z = \sigma(t, X, Y)$.

We order $\mathbb{A}_\perp$ with $t \leq u$ if and only if $|t| \subseteq |u|$ and for all $l \in |t|$ either l has label $\perp$ in $|t|$ or l has the same label in $|t|$ and $|u|$.

We prove $\rho(\rho(t, X), Z) \leq \rho(t, X \cup Y)$ (left-to-right implication) and $\rho(t, X \cup Y) \leq \rho(\rho(t, X), Z)$ (right-to-left implication).

Lemma 8.4 (Infinite Church-Rosser (left-to-right)) *For all $t, u, v \in \mathbb{A}_\perp$, all decidable sets X, Y of redexes of t: $\rho(\rho(t, X), Z) \leq \rho(t, X \cup Y)$ for $Z = \sigma(t, X, Y)$.*

Proof We have to prove that $\rho(\rho(t, X), Z) = \rho(t, X \cup Y)$ for $Z = \sigma(t, X, Y)$.

Let us abbreviate $Y_{(n)} = \sigma(t, X, n, Y)$ and $Y_{(\omega)} = \lim_{n\to\infty} Y_{(n)} = Z$. then $\rho(\rho(t, X), Z) = \rho(\rho(t, X), Y_{(\omega)}))$ is the limit of $\rho(\rho(t, X, n), Y_{(n)}, m)$ for $n, m \to \infty$.

We prove that for all $n, m \in N$ there is some $p \in N$ such that $\rho(\rho(t, X, n), Y_{(n)}, m) \leq \rho(t, X \cup Y, p)$, and conversely that for all $p \in N$ there are $n, m \in N$ such that $\rho(t, X \cup Y, p) \leq \rho(\rho(t, X, n), Y_{(n)}, m)$. It will follow that if a node is defined in $\rho(\rho(t, X), Z)$ then it is defined in $\rho(t, X \cup Y)$ and with the same constructor, and conversely.

$\rho(\rho(t, X, n), Y_{(n)}, m) = \perp$ we are done, suppose $\rho(\rho(t, X, n), Y_{(n)}, m) > \perp$. If $\phi_X(t) = \text{true}$, then we have:

1. If $t = (\lambda x^T.b)(a)$, then $\rho(t,X,n) = \rho(b[a/x],X',n-1)$, and by induction hypothesis $\rho(\rho(t,X,n),Y_{(n)},m) = \rho(\rho(b[a/x],X',n-1),Y'_{(n-1)},m) \leq \rho(b[a/x],X'\cup Y',p) = \rho(b[a/x],(X\cup Y)',p) = \rho((\lambda x^T.b)(a),X\cup Y,p+1)$.
2. If $t = \mathtt{cond}(f,g)(0)$, then $\rho(t,X,n+1) = \rho(f,X',n)$, and by induction hypothesis $\rho(\rho(t,X,n),Y_{(n)},m) = \rho(\rho(f,X',n-1),Y'_{(n-1)},m) \leq \rho(f,X'\cup Y,p) = \rho(f,X'\cup Y',p) = \rho(f,(X\cup Y)',p) = \rho(t,X\cup Y,p+1)$.
3. If $t = \mathtt{cond}(f,g)(\mathtt{S}(u))$, then $\rho(t,X,n+1) = \rho(g(u),X',n)$ and by induction hypothesis $\rho(\rho(t,X,n),Y_{(n)},m) = \rho(\rho(f,X',n-1),Y'_{(n-1)},m) \leq \rho(f,X'\cup Y',p) = \rho(f,(X\cup Y)',p) = \rho(t,X\cup Y,p+1)$.

Assume $\phi_X(t) = \text{false}$. Then we have $\rho(t,X,n) = c(\rho(t_1,X_1,n-1)\dots(\rho(t_h,X_h,n-1))$ and $X = c(X_1)\dots c(X_h)$.

Suppose $\phi_Y(t) = \text{true}$.

1. If $t = (\lambda x^T.b)(a)$, $\rho(t,X,n) = (\lambda x^T.b')(a')$ then $\rho((\lambda x^T.b')(a'),Y_{(n)},m) = \rho(b'[a'/x],Y'_{(n-1)},n-1)$, and by induction hypothesis $\rho(b'[a'/x],Y'_{(n-1)},n-1) \leq \rho(b'[a'/x],X\cup Y',p) = \rho(b[a/x],X'\cup Y',p) = \rho(b[a/x],(X\cup Y)',p) = \rho((\lambda x^T.b)(a),X\cup Y,p+1)$.
2. If $t = \mathtt{cond}(f,g)(0)$, $\rho(t,X,n) = \mathtt{cond}(f',g')(0)$ then $\rho(t,X,n) = \rho(f,X',n-1)$, and by induction hypothesis $\rho(\rho(t,X,n),Y_{(n)},m) = \rho(\rho(f,X',n-1),Y'_{(n-1)},m) \leq \rho(f,X'\cup Y,p) = \rho(f,X'\cup Y',p) = \rho(f,(X\cup Y)',p) = \rho(t,X\cup Y,p+1)$.
3. If $t = \mathtt{cond}(f,g)(\mathtt{S}(u))$, $\rho(t,X,n) = \mathtt{cond}(f',g')(\mathtt{S}(u'))$ then $\rho(t,X,n) = \rho(g(u),X',n-1)$ and by induction hypothesis $\rho(\rho(t,X,n),Y_{(n)},m) = \rho(\rho(g(u),X',n-1),Y'_{(n-1)},m) \leq \rho(g(u),X'\cup Y',p) = \rho(g(u),(X\cup Y)',p) = \rho(g(u),X\cup Y,p+1)$.

Assume $\phi_X(t) = \text{false}$ and $\phi_Y(t) = \text{false}$ and $t = c(t_1)\dots(t_h)$ for some $h = 0,1,2$. Then $\rho(\rho(t,X,n),Y_{(n)},m) = c(\rho(\rho(t_1,X_1,n-1),(Y_{(n)})_1,m-1),\dots,\rho(\rho(t_h,X_h,n-1),(Y_{(n)})_h,m-1) = c(\rho(\rho(t_1,X_1,n),(Y_1)_{(n-1)},m),\dots,\rho(\rho(t_h,X_h,n),(Y_h)_{(n-1)},m) \leq c(\rho(t_1,X_1\cup Y_1,p_1),\dots,\rho(t_h,X_h\cup Y_h,p_h) = c(\rho(t_1,(X\cup Y)_1,p_1),\dots,\rho(t_h,(X\cup Y)_h,p_h) \leq c(\rho(t_1,(X\cup Y)_1,p),\dots,\rho(t_h,(X\cup Y)_h,p) = \rho(t,X\cup Y,p)$ for $p = \max(p_1,\dots,p_h)$.

The proof of the opposite implication is similar.

Lemma 8.5 (Infinite Church-Rosser (right-to-left)) *For all $t,u,v \in \mathbb{A}_\perp$, all decidable sets X, Y of redexes of t: $\rho(t,X\cup Y) \leq \rho(\rho(t,X),Z)$ for $Z = \sigma(t,X,Y)$.*

Proof ....

Theorem 8.6 (Infinite Church-Rosser) *For all $t,u,v \in \mathbb{A}_\perp$, all decidable sets X, Y of redexes of t:*

1. *if $t \mapsto_X u$ and $t \mapsto_{X\cup Y} w$ and $Z = \sigma(t,X,Y)$ then $u \mapsto_Z w$.*
2. *if $t \mapsto_X u$ and $u \mapsto_Y v$, then for some $w \in \mathbb{A}_\perp$, for some decidable sets Z, T of redexes of u,v we have $u \mapsto_Z w$ and $v \mapsto_T w$.*

Proof

1. Assume if $t \mapsto_X u$ and $t \mapsto_{X\cup Y} w$ and $Z = \sigma(t,X,Y)$, in order to prove $u \mapsto_Z w$. Then $u = \rho(t,X)$ and $w = \rho(t,X\cup Y)$ and we have to prove $\rho(u,Z) = w$. This follows from $\rho(\rho(t,X),Z) = \rho(t,X\cup Y)$ (lemmas and).
2. Assume $t \mapsto_X u$ and $u \mapsto_Y v$. There is some $w \in \mathbb{A}_\perp$ such that $t \mapsto_{X\cup Y} w$. From the previous point we have $u \mapsto_Z w$ for $Z = \sigma(t,X,Y)$ and $v \mapsto_T w$ for $T = \sigma(t,Y,X)$

We define the safe trunk of a term as the part of the term which we can normalize with safe reductions only. In the rest of this section we will prove that Infinite Church-Rosser implies that if the safe trunk exists then it is unique.

Infinite Church-Rosser and Safe strong Normalization together imply that after finitely many steps all safe reductions reach the same safe trunk.

Definition 8.7 (Safe Trunk of a term) *Assume $t \in \mathbb{A}$.*

1. *The safe trunk of t is any expression $u[\mathtt{cond}(f_1,\cdot),\dots,\mathtt{cond}(f_n,\cdot)]$ such that for some $g_1,\dots,g_n$ we have $v = u[\mathtt{cond}(f_1,g_1),\dots,\mathtt{cond}(f_n,g_n)]$ safe normal and $t \mapsto v$.*
2. *t is finite for safe reduction if and only if all infinite reduction sequences from t include only finitely many "safe" reduction steps.*

Lemma 8.8 (Safe Trunk of a term) *Assume t is finite for safe reductions. If the safe-trunk of t exists then it is unique.*

Proof Assume t is finite for safe reductions in order to prove that the safe-trunk of t is unique.

Assume that $u[\mathtt{cond}(f_1,\cdot),\dots,\mathtt{cond}(f_n,\cdot)]$ and $u'[\mathtt{cond}(f'_1,\cdot),\dots,\mathtt{cond}(f'_{n'},\cdot)]$ are safe-trunks for t, in order to prove that $u = u'$ and $n = n'$.

Then for some $g_1,\dots,g_n$ and some $g'_1,\dots,g'_n$ we have that $v = u[\mathtt{cond}(f_1,g_1),\dots,\mathtt{cond}(f_n,g_n)]$ and $v' = u'[\mathtt{cond}(f'_1,g'_1),\dots,\mathtt{cond}(f'_n,g'_{n'})]$ are safe-normal forms of t and all cond-expressions shown are maximal. The decomposition of each safe-normal form v is therefore unique: if $v = u"[\mathtt{cond}(f"_1,g"_1),\dots,\mathtt{cond}(f"_n,g"_{n"})]$ then $u = u"$ and $n = n"$ and $f_1 = f"_1$, …, $f_n = f"_n$. Each reduction from v, v' takes place in some $g_1,\dots,g'_{n'}$.

By Infinite Church-Rosser (theorem 8.6) we deduce that v and v' are confluent outside the right-hand-side of cond-expressions, therefore for some $v"$ we have $v \mapsto_Z v"$ and $v' \mapsto_T v'"$. Since the reductions on v, v' take place in some $g_1,\dots,g'_{n'}$, we deduce that $v" = u[\mathtt{cond}(f_1,g"_1),\dots,\mathtt{cond}(f_n,g"_n)]$ and $v'" = u[\mathtt{cond}(f'_1,g'"_1),\dots,\mathtt{cond}(f'_n,g'"_{n'})]$ for some $g"_1,\dots,g'"_{n'}$. From the unicity of the decomposition of $v"$ with maximal cond-subterms we conclude that $u = u'$ and $n = n'$ and and $f_1 = f'_1$, …, $f_n = f'_n$, as wished.

References

[1] A Circular Version of Gödel's T and its abstraction complexity. Anupam Das. arXiv:2012.14421v2 [cs.LO] 16 Jan 2021.

9 Appendix: Equivalence Between Cyclic and non-Cyclic System T

In this section we prove that the two systems $\mathbb{T}$ and $\texttt{CT-}\lambda$ are each interpretable into the other. Both interpretations preserve the reduction relation, applications, 0 and $\texttt{S}$ and contexts.

In both interpretations we neglect the terms of $\texttt{CT-}\lambda$ including type variables in their type or in their context, because these terms have no corresponding in $\mathbb{T}$.

We first provide the interpretation from $\mathbb{T}$ to $\texttt{CT-}\lambda$, which has a short definition.

For any type T we define a term $\texttt{Rec} : T, (N, T \to T), N \to T$ such that $\texttt{Rec}(a, f, 0) = a$ and $\texttt{Rec}(a, f)(n + 1) = f(n, \texttt{Rec}(a, f, n))$, for all numeral $n \in \texttt{Num}$. The definition is $\texttt{Rec} = \lambda a, f.\texttt{rec}$ with $\texttt{rec} = \texttt{cond}(a, \lambda x^N.f(x, \texttt{rec}(x))) : N \to T$.

Proposition 9.1 *$\texttt{Rec}$ is a term of $\texttt{CT-}\lambda$.*

Proof

1. $\texttt{Rec}$ is regular by construction.
2. The only infinite path of $\texttt{Rec}$ loops from $\texttt{rec}$ to $\texttt{rec}$ infinitely many times, and it includes an infinitely progressing trace. Here it is: from the first unnamed argument of $\texttt{rec}$ to to the first unnamed argument of $\lambda x^N.f(x, \texttt{rec}(x))$, then to x^N in the context of $f(x, \texttt{rec}(x))$, to x^N in the context of $\texttt{rec}(x)$, and eventually to the first unnamed argument of $\texttt{rec}$ again.

If we replace each primitive recursion in $\mathbb{T}$ with the term $\texttt{Rec}$ in $\texttt{CT-}\lambda$ we define an interpretation from $\mathbb{T}$ into $\texttt{CT-}\lambda$, preserving reductions, applications, 0 and $\texttt{S}$ and contexts.

The rest of the section if but an early draft The opposite interpretation, from $\texttt{CT-}\lambda$ to $\mathbb{T}$, it has been defined by proof-theory for the combinatorial version of circular $\mathbb{T}$. For $\texttt{CT-}\lambda$, instead we will define an algorithm taking an infinite term in $\texttt{CT-}\lambda$, described as a finite circular tree, and returning a term in $\mathbb{T}$ with the required properties.

First, we need a notion of confluence and of extensional equality for functionals of $\texttt{CT-}\lambda$ and of $\mathbb{T}$.

We define $t \sim_{\beta,\texttt{rec}} u$ (a syntactical confluence for $\mathbb{T}$) if and and only if $t, u \in \mathbb{T}$ and t, u have the same type and for some $v \in \mathbb{T}$: $t \mapsto_{\beta,\texttt{rec}} v$ and $u \mapsto_{\beta,\texttt{rec}} v$.

We define an equivalence relation (an extensional equality for $\mathbb{T}$) $\sim_{\mathbb{T}}$ on $\mathbb{T}$, by induction on the type.

Definition 9.2 (An extensional equality on $\mathbb{T}$) *Assume $t, u \in \mathbb{T}$.*

1. *If $t, u : N$ and t, u are closed then we set: $(t \sim_{\mathbb{T}} u) \Leftrightarrow (t \sim_{\beta,\texttt{rec}} u)$.*
2. *If $t, u : A, \vec{A} \to N$ and t, u are closed then we set: $(t \sim_{\mathbb{T}} u) \Leftrightarrow \forall a \in \mathbb{T}.(a : A), (a \text{ closed}) \Rightarrow (t(a) \sim_{\mathbb{T}} u(a))$.*
3. *If $t, u : \vec{A} \to N$ and $\mathrm{FV}(t), \mathrm{FV}(u) \subseteq \vec{x}$ then we set:*

$$(t \sim_{\mathbb{T}} u) \Leftrightarrow \forall \vec{a} \in \mathbb{T}.(\vec{a} : \vec{A}), (\vec{a} \text{ closed}) \Rightarrow (t[\vec{a}/\vec{x}] \sim_{\mathbb{T}} u[\vec{a}/\vec{x}])$$

Then we can consider $\mathbb{T}$ as a structure $(\mathbb{T}/\sim_{\mathbb{T}}, 0, \texttt{S}, \texttt{ap})$ with natural numbers, functionals and extensional equality. In the same way we define an equivalence relation on terms of $\texttt{CT-}\lambda$ which denote functionals. We only consider terms whose type and context only include the atomic type N, and no type variables.

Definition 9.3 (An extensional equality on $\texttt{CT-}\lambda$) *Assume $t, u \in \texttt{CT-}\lambda$.*

1. *If $t, u : N$ and t, u are closed then we set: $(t \sim_{\texttt{CT-}\lambda} u) \Leftrightarrow (t \sim_{\texttt{CT-}\lambda} u)$.*
2. *If $t, u : A, \vec{A} \to N$ and t, u are closed then we set: $(t \sim_{\texttt{CT-}\lambda} u) \Leftrightarrow \forall a \in \texttt{CT-}\lambda.(a : A), (a \text{ closed}) \Rightarrow (t(a) \sim_{\texttt{CT-}\lambda} u(a))$.*
3. *If $t, u : \vec{A} \to N$ and $\mathrm{FV}(t), \mathrm{FV}(u) \subseteq \vec{x}$ then we set: $(t \sim_{\texttt{CT-}\lambda} u) \Leftrightarrow \forall \vec{a} \in \texttt{CT-}\lambda.(\vec{a} : \vec{A}), (\vec{a} \text{ closed}) \Rightarrow (t[\vec{a}/\vec{x}] \sim_{\texttt{CT-}\lambda} u[\vec{a}/\vec{x}])$.*

Our goal is now to prove that there is an embedding from the types of N-functionals in $(\mathtt{CT}\text{-}\lambda/\sim_{\mathtt{CT}\text{-}\lambda}, 0, \mathtt{S}, \mathtt{ap})$ to the entire $(\mathbb{T}/\sim_{\mathbb{T}}, 0, \mathtt{S}, \mathtt{ap})$. We ignore types of $\mathtt{CT}\text{-}\lambda$ including type variables, they have no corresponding in $\mathbb{T}$.

For each N-functional type T we have to define a map ϕ_T from terms of type T of $\mathtt{CT}\text{-}\lambda$ to terms of type T of $\mathbb{T}$. We abbreviate ϕ_T with ϕ and we require:

1. If $t \sim_{\mathtt{CT}\text{-}\lambda} u$ then $\phi(t) \sim_{\mathbb{T}} \phi(u)$
2. If $t : A \to B$, $u : A$ then $\phi(t(u)) \sim_{\mathbb{T}} \phi(t)(\phi(u))$
3. $\phi(0) \sim_{\mathbb{T}} 0$ and $\phi(\mathtt{S}(t)) \sim_{\mathbb{T}} \mathtt{S}(\phi(t))$

We will define an embedding from $\mathtt{CT}\text{-}\lambda$ to $\mathbb{T}$ extended with $+, \times$-types. There is an embedding from $\mathbb{T}$ extended with $+, \times$-types to $\mathbb{T}$ with only $N, \to$, therefore it is enough to embed $\mathtt{CT}\text{-}\lambda$ to $\mathbb{T}$ extended with $+, \times$-types.

We suppose be fixed a cyclic λ-term $t \in \mathtt{CT}\text{-}\lambda$, $t : T$, and we use several ingredients. First we consider all $\Gamma_i \vdash u_i : U_i$ for u_i sub-term of t, we turn it into a simple type $\Gamma_i \to U_i$, then we consider a single type $S = \Sigma_i \Gamma_i \to U_i$.

1. We suppose be given a map $\mathtt{trunk}_t : N \to S = \Sigma_i \Gamma_i \to U_i$, such that $\mathtt{trunk}_t(n)$ is the unfolding of t with all subterms $u : U$ in the level number n replaced by a dummy term 0_U. We use the type S in order to have a common type and context for all sub-terms u.
2. For each pair of subterms u, v of t and each path π from u to v. we suppose be given a one-to-many trace relation R_π between the indexes of N-arguments of u and of the N-arguments of v. We close the set of relations R_π by composition and we obtain a finite set (of exponential size in t).

By the global trace condition for each infinite composition of R_π there is some infinitely progressing trace. This means that there is some index i in the domain of R_π such that for some $n \in N$ we have $R^n_\pi(i, i)$ and the variable i progresses.

We consider any assignment $t' \equiv t[\vec{n}, \vec{x}](\vec{m}, \vec{y})$ of the arguments of t. The idea is to find map ϕ such that for all $p \geq \vec{n}, \vec{n}$ we have $\mathtt{trunk}_{t'}(m)$ stationary for all $m \geq \phi(p)$, therefore $t' = \mathtt{trunk}_{t'}(\phi(p))$.

$\phi_c(p)$ is the map computing the maximum number of nodes for an Erdos tree in c colors with height $\leq p$ and branching $\leq c$ (there is at most one child per color). Whenever an Erdos tree has at least a branch of length $cp + 1$ and c colors, then we have a monotonically colored sequence of length $cp + 1$, therefore at least one homogeneous set of length $p + 1$. If we take any composition of $\phi_c(p) + 1$ times some relations R_π, there is some homogeneous set of $(p + 1)$ elements decorated with a single R_π, and for some i some variable starting from some value $\leq p$ and decreasing p times.

This means that each branch terminates and the whole computation of $\mathtt{trunk}_{t'}(\phi(p))$ terminates.

For a c-color tree, the value of $\phi_c(p)$ is $1 + c + c^2 + \ldots + c^{p-1} = (c^p - 1)/(c - 1)$. Suppose $f : N \to N$ is some weakly increasing map. We want to prove that there is some Erdos tree including some branch including some f-homogeneous set: some homogeneous set with first point h followed by $f(h)$ more points. We want to define a functional in $\mathbb{T}$ such that all Erdos trees with $\geq F(f)$ nodes include some f-homogeneous set.

We call a Ramsey functional any functional $F(\vec{x}, c)$ associated to e any functional taking weakly increasing functionals $\vec{x}$, and returning an upper bound for the size of a c-color Erdos tree including some homogeneous set with first node l with value $\leq e(\vec{x}, \vec{F}, l)$, followed by $\leq e(\vec{x}, \vec{F}, l)$ more nodes, in which some variable i decreases by 1 each n steps (n number of sub-terms of the cyclic term).

We require that e is any primitive recursive functional, that is, e is defined by simply typed lambda calculus plus recursion on $\mathtt{Seq}(N)$, and $\vec{F}$ are Ramsey functionals.

We claim that all sub-terms of t have a computation time bounded by some Ramsey functional. If this is not the case, we define some infinite path with a infinitely progressing trace associated to some infinitely decreasing numeral, contradiction.

. . . .