Some advances in the theory of computable functionals TCF^+ and its implementation

Iosif Petrakis

Joint, on-going work with H. Schwichtenberg

LMU München - Mathematisches Institut

FOMCAF 2013, University of Padova, 09.01.2013

A road to TCF$^+$

1. **Hilbert** (1926).
2. **Gödel**'s **T** (1958).
3. **Kleene** and **Kreisel** (1959).
4. **Platek** (1966).
5. **Scott**'s **LCF** (1969).
6. **Plotkin**'s **PCF** (1977).
7. The Munich logic group (mainly **H. Schwichtenberg** and **U. Berger**, 1990-2012) developed a theory of computable functionals, TCF, a variant of HA$^\omega$ (use of minimal or intuitionistic logic), the terms of which extend both Gödel's **T** and Plotkin's PCF.
8. TCF$^+$ is an extension of TCF. Its **object-terms** are the terms of TCF, representing the functionals, and its **approximation-terms** represent their finite approximations. In that way TCF$^+$ is better adjusted to the common (non-flat) Scott model than TCF.

Why TCF$^+$?

Theorem (Kreisel 1959)

A compact (finitely generated) functional can be extended to a total one, i.e.,

$$\forall_{U \in \mathrm{Con}_\rho} \exists_{x \in \mathbb{T}_\rho} (U \subseteq x).$$

We would like to give a completely formal proof of DT revealing its computational content.
But for that we need to express in a formal language both functionals (ideals) **and** formal neighborhoods

Why TCF^+?

1. A central motivation for TCF^+ is the paradigm of the **point-free topology**.

Why TCF^+?

1. A central motivation for TCF^+ is the paradigm of the **point-free topology**.
2. Within higher type computability this directs to an as much as possible reconstruction of the study of the "ideal", abstract functionals (points) through the study of their concrete and finite approximations (tokens and formal neighborhoods).

Why TCF$^+$?

1. A central motivation for TCF$^+$ is the paradigm of the **point-free topology**.
2. Within higher type computability this directs to an as much as possible reconstruction of the study of the "ideal", abstract functionals (points) through the study of their concrete and finite approximations (tokens and formal neighborhoods).
3. Instrumental to this will be the use of **information systems** instead of abstract domains for the description of the Scott model. An important tool of the point-free approach is the notion of an **approximable mapping**, the point-free version of a function between domains.

Algebraic Domains and Information Systems

1. An **algebraic domain** $(D, \leq, \perp, D_0)$ is a consistently complete, algebraic cpo and it is the result of investigating the structure of the domains arising in the Scott model of PCF.

Algebraic Domains and Information Systems

1. An **algebraic domain** $(D, \leq, \perp, D_0)$ is a consistently complete, algebraic cpo and it is the result of investigating the structure of the domains arising in the Scott model of PCF.
2. Scott's **information system** $\mathcal{A} = (A, \mathrm{Con}, \vdash)$: A are the *tokens*, $\mathrm{Con} \subseteq \mathcal{P}^{\mathsf{fin}}(A)$ is the set of **formal neighborhoods**, and $\vdash \subseteq \mathrm{Con} \times A$ is the **entailment** relation, where $U \vdash a$ means "the information in U is sufficient to compute the bit of data a".

Algebraic Domains and Information Systems

1. An **algebraic domain** $(D, \leq, \perp, D_0)$ is a consistently complete, algebraic cpo and it is the result of investigating the structure of the domains arising in the Scott model of PCF.
2. Scott's **information system** $\mathcal{A} = (A, \mathrm{Con}, \vdash)$: A are the *tokens*, $\mathrm{Con} \subseteq \mathcal{P}^{\mathrm{fin}}(A)$ is the set of **formal neighborhoods**, and $\vdash \subseteq \mathrm{Con} \times A$ is the **entailment** relation, where $U \vdash a$ means "the information in U is sufficient to compute the bit of data a".
3. $$\begin{aligned} &1.\ \ \mathrm{Con}(U) \to V \subseteq U \to \mathrm{Con}(V),\\ &2.\ \ \mathrm{Con}(\{a\}),\\ &3.\ \ \mathrm{Con}(U) \to U \vdash a \to \mathrm{Con}(U \cup \{a\}),\\ &4.\ \ \mathrm{Con}(U) \to a \in U \to U \vdash a,\\ &5.\ \ \mathrm{Con}(U) \to \mathrm{Con}(V) \to U \vdash V \to V \vdash a \to U \vdash a. \end{aligned}$$

Ideals of an information system

1. An **ideal** $x \subseteq A$ is consistent,

$$U \subseteq^{\text{fin}} x \to \mathrm{Con}(U),$$

and deductively closed

$$x \supseteq U \to U \vdash a \to a \in x.$$

Ideals of an information system

1. An **ideal** $x \subseteq A$ is consistent,

$$U \subseteq^{\text{fin}} x \to \mathrm{Con}(U),$$

and deductively closed

$$x \supseteq U \to U \vdash a \to a \in x.$$

2. If $U \in \mathrm{Con}$, then

$$\overline{U} = \{a \in A : U \vdash a\}$$

is a **compact** ideal.

Ideals of an information system

1. An **ideal** $x \subseteq A$ is consistent,

$$U \subseteq^{\text{fin}} x \to \mathrm{Con}(U),$$

   and deductively closed

$$x \supseteq U \to U \vdash a \to a \in x.$$

2. If $U \in \mathrm{Con}$, then

$$\overline{U} = \{a \in A : U \vdash a\}$$

   is a **compact** ideal.
3. $(|\mathcal{A}|, \subseteq, |\mathcal{A}|_0, \overline{\emptyset})$ is a domain, and each domain "is" the ideals of an information system.

Free non-flat algebras as data types

1. The algebras ι considered here are of the form $\mu_\xi(K_0, \ldots, K_{l-1})$, where each constructor K_j is of type of the form
$$\vec{\rho} \to (\vec{\sigma_\nu} \to \xi)_{\nu<n} \to \xi.$$

Free non-flat algebras as data types

1. The algebras ι considered here are of the form $\mu_\xi(K_0, \dots, K_{l-1})$, where each constructor K_j is of type of the form
$$\vec{\rho} \to (\vec{\sigma_\nu} \to \xi)_{\nu<n} \to \xi.$$
2. $\mathbf{B} := \mu_\xi(\xi, \xi)$ (booleans).

Free non-flat algebras as data types

1. The algebras ι considered here are of the form $\mu_\xi(K_0, \dots, K_{l-1})$, where each constructor K_j is of type of the form
$$\vec{\rho} \to (\vec{\sigma_\nu} \to \xi)_{\nu < n} \to \xi.$$
2. $\mathbf{B} := \mu_\xi(\xi, \xi)$ (booleans).
3. $\mathbf{N} := \mu_\xi(\xi, \xi \to \xi)$ (natural numbers).

Free non-flat algebras as data types

1. The algebras ι considered here are of the form $\mu_\xi(K_0, \ldots, K_{l-1})$, where each constructor K_j is of type of the form
$$\vec{\rho} \to (\vec{\sigma_\nu} \to \xi)_{\nu<n} \to \xi.$$
2. $\mathbf{B} := \mu_\xi(\xi, \xi)$ (booleans).
3. $\mathbf{N} := \mu_\xi(\xi, \xi \to \xi)$ (natural numbers).
4. $\mathbf{D} := \mu_\xi(\xi, \xi \to \xi \to \xi)$ (derivations).

Free non-flat algebras as data types

1. The algebras ι considered here are of the form $\mu_\xi(K_0, \dots, K_{l-1})$, where each constructor K_j is of type of the form
$$\vec{\rho} \to (\vec{\sigma_\nu} \to \xi)_{\nu<n} \to \xi.$$
2. $\mathbf{B} := \mu_\xi(\xi, \xi)$ (booleans).
3. $\mathbf{N} := \mu_\xi(\xi, \xi \to \xi)$ (natural numbers).
4. $\mathbf{D} := \mu_\xi(\xi, \xi \to \xi \to \xi)$ (derivations).
5. $\mathbf{O} := \mu_\xi(\xi, \xi \to \xi, (\mathbf{N} \to \xi) \to \xi)$ (ordinals).

Free non-flat algebras as data types

1. The algebras ι considered here are of the form $\mu_\xi(K_0, \dots, K_{l-1})$, where each constructor K_j is of type of the form
$$\vec{\rho} \to (\vec{\sigma_\nu} \to \xi)_{\nu<n} \to \xi.$$
2. $\mathbf{B} := \mu_\xi(\xi, \xi)$ (booleans).
3. $\mathbf{N} := \mu_\xi(\xi, \xi \to \xi)$ (natural numbers).
4. $\mathbf{D} := \mu_\xi(\xi, \xi \to \xi \to \xi)$ (derivations).
5. $\mathbf{O} := \mu_\xi(\xi, \xi \to \xi, (\mathbf{N} \to \xi) \to \xi)$ (ordinals).
6. Types ρ, σ, τ: from algebras ι by $\rho \to \sigma$, or
$$\rho = \tau_1 \to \dots \to \tau_n \to \iota.$$

The set-theoretic information systems $(\mathbf{C}_\rho)_\rho$

We simultaneously define $C_\iota, C_{\rho\to\sigma}, \mathrm{Con}_\iota, \mathrm{Con}_{\rho\to\sigma}, \vdash_\iota$ and $\vdash_{\rho\to\sigma}$.

1. A **ground-type token**, $a \in C_\iota$, is a type correct constructor expression $Ca_1^* \ldots a_n^*$, where each a_i^* is an *extended token*, i.e., a *proper* token or the special symbol $*_\iota$ which carries no information.
2. An **arrow-type token**, $a \in C_{\rho\to\sigma}$, is a pair (U, b), where $U \in \mathrm{Con}_\rho$ and $b \in C_\sigma$.
3. A **ground-type formal neighborhood**, $U \in \mathrm{Con}_\iota$, is a finite set of tokens in C_ι starting with the same constructor $C^{\tau_1\to\ldots\to\tau_n\to\iota}$, i.e.,

$$U = \{Ca_{(1)1}^* \ldots a_{(1)n}^*, \ldots, Ca_{(k)1}^* \ldots a_{(k)n}^*\},$$

for some $k \in \mathbb{N}$, such that, for each $1 \leq l \leq n$,

$$U_l = \{a_{(i)l}^* : a_{(i)l}^* \text{ is a proper token in } C_{\tau_l} \wedge 1 \leq i \leq k\} \in \mathrm{Con}_{\tau_l}.$$

The set-theoretic information systems $(\mathbf{C}_\rho)_\rho$

1. An **arrow-type formal neighborhood**, $W \in \mathrm{Con}_{\rho\to\sigma}$, is a finite set of tokens in $C_{\rho\to\sigma}$, i.e., $W = \{(U_i, b_i) : i \in I\}$, for some finite set I, such that

$$\forall_{J\subseteq I}\Big(\bigcup_{j\in J} U_j \in \mathrm{Con}_\rho \to \{b_j : j \in J\} \in \mathrm{Con}_\sigma\Big).$$

2. If $U = \{Ca^*_{(1)1}\ldots a^*_{(1)n}, \ldots, Ca^*_{(k)1}\ldots a^*_{(k)n}\}$ is a ground-type formal neighborhood such that $k \geq 1$ and $C^{\tau_1\to\ldots\to\tau_n\to\iota}$ is a constructor, then

$$\{Ca^*_{(1)1}\ldots a^*_{(1)n}, \ldots, Ca^*_{(k)1}\ldots a^*_{(k)n}\} \vdash_\iota C'\vec{a^*} \leftrightarrow C = C' \wedge \forall_l (U_l \vdash_{\tau_l} a^*_l),$$

where U_l is defined as above and $U \vdash *$ is always true.

3. If $W = \{(U_i, b_i) : i \in I\}$ is an arrow-type formal neighborhood, then

$$W \vdash_{\rho\to\sigma} (V, b) \leftrightarrow WV := \{b_i : V \vdash_\rho U_i\} \vdash_\sigma b.$$

The set-theoretic information systems $(\mathbf{C}_\rho)_\rho$

Theorem (H.S, I.P 2012)

The structure $\mathbf{C}_\rho = (C_\rho, \mathrm{Con}_\rho, \vdash_\rho)$ *is an information system, for each type* ρ.

Proof.

Since the definition of $\mathbf{C}_\rho$ is given by recursion on the height of the syntactic expressions involved, the proof is also given w.r.t. this height. It is simple for the ground types, but for the arrow types it uses simultaneous general induction in a non trivial way. Note that $WV \in \mathrm{Con}_\sigma$ is not obvious and has to be proved. □

Proposition: The information system $\mathbf{C}_\rho$ is **coherent** (U is consistent iff each pair of its elements is consistent), for each ρ.

The starting point of TCF^+

1. We want to reproduce in the syntactic level of TCF^+ the systems $\mathbf{C}_\rho$ avoiding their set-theoretic character. First we work with the generic algebra of derivations $\mathbf{D}$ defining the **syntactic information systems $\mathbf{SC}_\mathbf{D}$** and $\mathbf{SC}_{\mathbf{D}\to\mathbf{D}}$.

The starting point of TCF^+

1. We want to reproduce in the syntactic level of TCF^+ the systems $\mathbf{C}_\rho$ avoiding their set-theoretic character. First we work with the generic algebra of derivations $\mathbf{D}$ defining the **syntactic information systems** $\mathbf{SC}_\mathbf{D}$ and $\mathbf{SC}_{\mathbf{D}\to\mathbf{D}}$.
2. In that way not only a part of set-theoretic mathematics has a formal and constructive counterpart, but also its formalization makes possible the implementation of all related notions and results to a proof assistant like **Minlog**.

The starting point of TCF$^+$

1. We want to reproduce in the syntactic level of TCF$^+$ the systems $\mathbf{C}_\rho$ avoiding their set-theoretic character. First we work with the generic algebra of derivations **D** defining the **syntactic information systems** $\mathbf{SC_D}$ and $\mathbf{SC_{D \to D}}$.
2. In that way not only a part of set-theoretic mathematics has a formal and constructive counterpart, but also its formalization makes possible the implementation of all related notions and results to a proof assistant like **Minlog**.
3. The main tool of this constructivization is the use of **inductive definitions** in the spirit of inductive mathematics (**Brouwer, Martin-Löf, Sambin, Coquand**).

The algebra **D** in Minlog

1. (add-alg "tokstar"
   '("CS" "tokstar")
   '("CZ" "tokstar")
   '("CC" "tokstar => tokstar => tokstar"))

The algebra **D** in Minlog

1. (add-alg "tokstar"
   '("CS" "tokstar")
   '("CZ" "tokstar")
   '("CC" "tokstar => tokstar => tokstar"))
2. We represent finite sets of tokens as objects of type "list tokstar".

The algebra **D** in Minlog

1. (add-alg "tokstar"
   '("CS" "tokstar")
   '("CZ" "tokstar")
   '("CC" "tokstar => tokstar => tokstar"))
2. We represent finite sets of tokens as objects of type "list tokstar".
3. We define **membership** of a token in a list as a program constant "**in**":

   "a in (Nil tokstar)" "False"
   "a in (b::bs)" "a=b orb a in bs".

The algebra **D** in Minlog

1. (add-alg "tokstar"
   '("CS" "tokstar")
   '("CZ" "tokstar")
   '("CC" "tokstar => tokstar => tokstar"))
2. We represent finite sets of tokens as objects of type "list tokstar".
3. We define **membership** of a token in a list as a program constant "**in**":

   "a in (Nil tokstar)" "False"
   "a in (b::bs)" "a=b orb a in bs".
4. We define inclusion of one list to another one as a program constant "**InDot**":

   "InDot (Nil tokstar) as" "True"
   "InDot (a::as) bs" "a in bs andb InDot as bs".

Definition of consistency in **D**

First we define **consistency between two tokens** as the program constant "**con**" of type "tokstar => tokstar => boole"

"con CS b" "True"
"con a CS" "True"
"con CZ CZ" "True"
"con CZ (CC a b)" "False"
"con (CC a b) CZ" "False"
"con (CC a b) (CC c d)" "con a c andb con b d".

Definition of consistency in **D**

Next we define **consistency between a token and a list** as the program constant "**Altcon**" of type "tokstar => list tokstar => boole"

"Altcon a (Nil tokstar)" "True"
"Altcon a (b::as)" "con a b andb Altcon a as").

Finally we define the **consistency of a list** as the program constant "**Con**" of type "list tokstar => boole":

"Con(Nil tokstar)" "True"
"Con(a::as)" "Altcon a as andb Con(as)".

The proof of **totality** of these constants is direct and makes no use of general induction.

Properties of consistency in **D**

Theorem

1. **Reflexivity of con**: *"allnc a(TotalTokstar a → con a a)"*.
2. **Commutativity of con**: *"allnc a1(TotalTokstar a1 → allnc a2(TotalTokstar a2 → (con a1 a2) = (con a2 a1)))"*.
3. **Axiom2**: *"allnc a(TotalTokstar a → Con(a::(Nil tokstar)))"*.
4. **Reflexivity of Con**: *"allnc a(TotalTokstar a → Con(a::a::(Nil tokstar)))"*.
5. **Commutativity of Con**: *"allnc a1(TotalTokstar a1 → allnc a2(TotalTokstar a2 → Con(a1::a2::(Nil tokstar)) = Con(a2::a1::(Nil tokstar))))"*.
6. **Axiom1**: *"allnc as1(TotalList as1 → allnc as2(TotalList as2 → Con as1 → InDot as2 as1 → Con as2))"*.

Entailment in **D**

1. "**argOne**" (argTwo) is of type "list tokstar => list tokstar":
   "argOne (Nil tokstar)" "(Nil tokstar)"
   "argOne (CS::as)" "argOne as"
   "argOne (CZ::as)" "argOne as"
   "argOne ((CC a b)::as)" "a::(argOne as)".

Entailment in **D**

1. "**argOne**" (argTwo) is of type "list tokstar => list tokstar":

   "argOne (Nil tokstar)" "(Nil tokstar)"
   "argOne (CS::as)" "argOne as"
   "argOne (CZ::as)" "argOne as"
   "argOne ((CC a b)::as)" "a::(argOne as)".

2. "**Comp**" is of type "list tokstar => boole":

   "Comp(Nil tokstar)" "False"
   "Comp(CS::as)" "Comp as"
   "Comp(CZ::as)" "Comp as"
   "Comp(CC a b::as)" "True".

Entailment in **D**

1. "**argOne**" (argTwo) is of type "list tokstar => list tokstar":

   "argOne (Nil tokstar)" "(Nil tokstar)"
   "argOne (CS::as)" "argOne as"
   "argOne (CZ::as)" "argOne as"
   "argOne ((CC a b)::as)" "a::(argOne as)".

2. "**Comp**" is of type "list tokstar => boole":

   "Comp(Nil tokstar)" "False"
   "Comp(CS::as)" "Comp as"
   "Comp(CZ::as)" "Comp as"
   "Comp(CC a b::as)" "True".

3. "**Ent**" is of type "list tokstar => tokstar => boole":

   "Ent as CS" "True"
   "Ent as CZ" "CZ in as"
   "Ent as (CC a b)" "Comp(as) andb Ent (argOne(as)) a andb
   Ent (argTwo(as)) b".

Properties of entailment in **D**

Theorem

1. **Preaxiom4**: *"allnc a(TotalTokstar a → allnc as(TotalList as → a in as → Ent as a))".*
2. **Axiom4**: *"allnc a(TotalTokstar a → allnc as(TotalList as →* **Con(as)** *→ a in as → Ent as a))".*
3. **Prepreaxiom3**: *"allnc as(TotalList as → allnc a(TotalTokstar a → allnc b(TotalTokstar b → Con as → b in as → Ent as a → con a b)))".*
4. **Preaxiom3**: *"allnc as(TotalList as → allnc a(TotalTokstar a → Con as → Ent as a → Altcon a as))".*
5. **Axiom3:** *"allnc as(TotalList as → allnc a(TotalTokstar a → Con as → Ent as a → Con (a::as)))".*

Properties of entailment in **D**

Theorem

1. "$allnc\ as(TotalList\ as \rightarrow allnc\ bs(TotalList\ bs \rightarrow EntList\ as\ bs \rightarrow EntList\ (argOne(as))\ (argOne(bs))))$".
2. **Preaxiom5**: "$allnc\ a(TotalTokstar\ a \rightarrow allnc\ as(TotalList\ as \rightarrow allnc\ bs(TotalList\ bs \rightarrow EntList\ as\ bs \rightarrow Ent\ bs\ a \rightarrow Ent\ as\ a)))$".
3. **Axiom5:** "$allnc\ a(TotalTokstar\ a \rightarrow allnc\ as(TotalList\ as \rightarrow allnc\ bs(TotalList\ bs \rightarrow \mathbf{Con\ as} \rightarrow \mathbf{Con\ bs} \rightarrow EntList\ as\ bs \rightarrow Ent\ bs\ a \rightarrow Ent\ as\ a)))$".

Remark 1: We proved the preaxioms 4 and 5 without the consistency hypotheses.
Remark 1: All proofs are without the use of general induction.

The algebra “list tokstar yprod tokstar”

1. The arrow-tokens are pairs $w = (as, a) = (lft(w), rht(w))$.

The algebra "list tokstar yprod tokstar"

1. The arrow-tokens are pairs $w = (as, a) = (lft(w), rht(w))$.
2. The finite sets of arrow-tokens are lists ws of arrow-tokens.

The algebra "list tokstar yprod tokstar"

1. The arrow-tokens are pairs $w = (as, a) = (lft(w), rht(w))$.
2. The finite sets of arrow-tokens are lists *ws* of arrow-tokens.
3. "**One**" is of type "list (list tokstar yprod tokstar) => list (list tokstar)":

   "One(Nil (list tokstar yprod tokstar))" "(Nil (list tokstar))"
   "One(w::ws)" "lft(w)::One(ws)".

The algebra "list tokstar yprod tokstar"

1. The arrow-tokens are pairs $w = (as, a) = (lft(w), rht(w))$.
2. The finite sets of arrow-tokens are lists *ws* of arrow-tokens.
3. "**One**" is of type "list (list tokstar yprod tokstar) => list (list tokstar)":

   "One(Nil (list tokstar yprod tokstar))" "(Nil (list tokstar))"
   "One(w::ws)" "lft(w)::One(ws)".
4. "**Two**" is of type "list (list tokstar yprod tokstar) => list tokstar":

   "Two (Nil (list tokstar yprod tokstar))" "(Nil tokstar)"
   "Two(w::ws)" "rht(w)::Two(ws)"

Consistency in **D** $\rightarrow$ **D**

1. "**arrcons**" is of type "(list tokstar yprod tokstar) => (list tokstar yprod tokstar) => boole":

   "arrcons u w" "(Con(lft(u)++lft(w)) imp (con rht(u) rht(w)))".

Consistency in **D** → **D**

1. "**arrcons**" is of type "(list tokstar yprod tokstar) => (list tokstar yprod tokstar) => boole":

   "arrcons u w" "(Con(lft(u)++lft(w)) imp (con rht(u) rht(w)))".

2. "**arrAltcons**" is of type "(list tokstar yprod tokstar) => list(list tokstar yprod tokstar) => boole":

   "arrAltcons w (Nil (list tokstar yprod tokstar))" "True"
   "arrAltcons w (u::ws)" "arrcons w u andb arrAltcons w ws".

Consistency in D → D

1. "**arrcons**" is of type "(list tokstar yprod tokstar) => (list tokstar yprod tokstar) => boole":

   "arrcons u w" "(Con(lft(u)++lft(w)) imp (con rht(u) rht(w)))".

2. "**arrAltcons**" is of type "(list tokstar yprod tokstar) => list(list tokstar yprod tokstar) => boole":

   "arrAltcons w (Nil (list tokstar yprod tokstar))" "True"
   "arrAltcons w (u::ws)" "arrcons w u andb arrAltcons w ws".

3. "**arrCons**" is of type "list (list tokstar yprod tokstar) => boole":

   "arrCons(Nil (list tokstar yprod tokstar))" "True"
   "arrCons(w::ws)" "arrAltcons w ws andb arrCons(ws)".

Properties of consistency in $\mathbf{D} \to \mathbf{D}$

Theorem

1. **Reflexivity of arrcons**.
2. **Commutativity of arrcons**.
3. **arrAxiom2**.
4. **Reflexivity of arrCons**.
5. **Commutativity of arrCons**.
6. **arrAxiom1**: *"allnc ws1(TotalList ws1 → allnc ws2(TotalList ws2 → arrCons ws1 → arrInDot ws2 ws1 → arrCons ws2))"*.

A generalization

Theorem

If s is a total function of type $\alpha => \alpha => \mathrm{boole}$, such that s is reflexive and commutative, then

1. Alts, *of type* $\alpha => \mathrm{list}\ \alpha => \mathrm{boole}$ *defined by*
   $\mathrm{Alts}(a, \mathrm{Nil}\ \alpha) = \mathtt{tt}$,
   $\mathrm{Alts}(a, (b :: as) = s(a, b) \wedge_{\mathbf{B}} \mathrm{Alts}(a, as)$
   is total, reflexive, commutative and
   $$\mathrm{Alts}(a, as) \leftrightarrow \forall_{b \dot{\in} as}(s(a, b)).$$
2. *If S is of type* $\mathrm{list}\ \alpha => \mathrm{boole}$ *defined by*
   $S(\mathrm{Nil}\ \alpha) = \mathtt{tt}$,
   $S(a :: as) = \mathrm{Alts}(a, as) \wedge_{\mathbf{B}} S(as)$ *then* ,
   (i) $\forall_{as}(S(as) \leftrightarrow \forall_{a, b \dot{\in} as}(s(a, b)))$.
   (ii) $\forall as_1, as_2(S(as_1) \to as_2 \dot{\subseteq} as_1 \to S(as_2))$.

The Parallel-True-Sublist

"**PTS**" is of type "list boole => list tokstar => list tokstar" and from a given list of booleans and a list of tokens outputs the terms of the latter which correspond to the appearance of "True" in the former.

"PTS (Nil boole) as" "(Nil tokstar)"
"PTS As (Nil tokstar)" "(Nil tokstar)"
"PTS (True::As) (a::as)" "a::(PTS As as)"
"PTS (False::As) (a::as)" "PTS As as".

Entailment in $\mathbf{D} \to \mathbf{D}$

1. "**AltEntList**" is of type "list tokstar => list (list tokstar) => list boole":

   "AltEntList as (Nil (list tokstar))" "(Nil boole)"
   "AltEntList as (bs::ass)" "(EntList as bs)::(AltEntList as ass)".

Entailment in **D** $\rightarrow$ **D**

1. "**AltEntList**" is of type "list tokstar => list (list tokstar) => list boole":

   "AltEntList as (Nil (list tokstar))" "(Nil boole)"
   "AltEntList as (bs::ass)" "(EntList as bs)::(AltEntList as ass)".

2. "**App**" is of type "list (list tokstar yprod tokstar) => list tokstar => list tokstar":

   "App ws as" "PTS (AltEntList as (One(ws))) (Two(ws))".

Entailment in $\mathbf{D} \to \mathbf{D}$

1. "**AltEntList**" is of type "list tokstar => list (list tokstar) => list boole":

   "AltEntList as (Nil (list tokstar))" "(Nil boole)"
   "AltEntList as (bs::ass)" "(EntList as bs)::(AltEntList as ass)".

2. "**App**" is of type "list (list tokstar yprod tokstar) => list tokstar => list tokstar":

   "App ws as" "PTS (AltEntList as (One(ws))) (Two(ws))".

3. "**arrEnt**" is of type "list (list tokstar yprod tokstar) => list tokstar yprod tokstar => boole":

   "arrEnt ws w" "Ent (App ws (lft(w))) (rht(w))".

Properties of entailment in $\mathbf{D} \to \mathbf{D}$

Theorem

1. **Prearraxiom4**: *"allnc ws(TotalList ws → allnc w(TotalYprod w → w arrin ws → arrEnt ws w))".*
2. *"allnc bs1(TotalList bs1 → allnc bs2(TotalList bs2 → allnc ws(TotalList ws → allnc a(TotalTokstar a → EntList bs1 bs2 → Ent (App ws bs2) a → Ent (App ws bs1) a))))".*
3. **Preprearraxiom5**: *"allnc ws1(TotalList ws1 → allnc ws2(TotalList ws2 → allnc as(TotalList as → arrEntList ws1 ws2 → EntList (App ws1 as) (App ws2 as))))".*
4. **Prearraxiom5**: *"allnc ws1(TotalList ws1 → allnc ws2(TotalList ws2 → allnc w(TotalYprod w → arrEntList ws1 ws2 → arrEnt ws2 w → arrEnt ws1 w)))".*

Properties of entailment in $\mathbf{D} \to \mathbf{D}$

Theorem

1. "$allnc\ bs(TotalList\ bs \to allnc\ As(TotalList\ As \to allnc\ b(TotalTokstar\ b \to b\ in\ (PTS\ As\ bs) \to exr\ n(TotalNat\ n \wedge (n\ thof\ As) = True \wedge (n\ thof\ bs) = b))))$".
2. "$allnc\ ws(TotalList\ ws \to allnc\ as(TotalList\ as \to arrCons\ ws \to Con\ as \to Con(App\ ws\ as)))$".
3. **arrprepreaxiom3**: "$allnc\ u(TotalYprod\ u \to allnc\ ws(TotalList\ ws \to allnc\ as(TotalList\ as \to u\ arrin\ ws \to rht\ u\ in\ (App\ ws(as ++lft\ u)))))$".
4. **arrpreAxiom3**: "$allnc\ ws(TotalList\ ws \to allnc\ w(TotalYprod\ w \to arrCons\ ws \to arrEnt\ ws\ w \to arrAltcons\ w\ ws))$".
5. **arrAxiom3**: "$allnc\ ws(TotalList\ ws \to allnc\ w(TotalYprod\ w \to arrCons\ ws \to arrEnt\ ws\ w \to arrCons\ (w :: ws)))$".

Split information systems

The entailment relation of a **split** information system is a subset of $\mathcal{P}^{\mathrm{fin}}(A) \times A$ (i.e., U is not necessarily consistent in $U \vdash a$) satisfying axioms 1-3 of an information system and

$$4'.\quad a \in U \to U \vdash a,$$

$$5'.\quad U \vdash V \to V \vdash a \to U \vdash a.$$

Theorem (I.P, H.S 2012)

The systems $\mathbf{C}_\rho$ are split information systems, for each type ρ.

Proof.

Simplified version of the proof that $\mathbf{C}_\rho$ is an information system. $\square$

Decidable ideals of **D**

1. "I" is a variable of type "tokstar => boole".

Decidable ideals of **D**

1. "I" is a variable of type "tokstar => boole".
2. "**In**" is of type "tokstar => (tokstar => boole) => boole":
   "a In I" "I a".

Decidable ideals of **D**

1. “l” is a variable of type “tokstar => boole”.
2. “**In**” is of type “tokstar => (tokstar => boole) => boole”:

   “a In l” “l a”.
3. “**Indot**” s of type “list tokstar => (tokstar => boole) => boole”))”

   “Indot (Nil tokstar) l” “True”
   “Indot (a::as) l” “a In l andb Indot as l”.

Decidable ideals of **D**

1. "l" is a variable of type "tokstar => boole".
2. "**In**" is of type "tokstar => (tokstar => boole) => boole":

   "a In l" "l a".
3. "**Indot**" s of type "list tokstar => (tokstar => boole) => boole"))"

   "Indot (Nil tokstar) l" "True"
   "Indot (a::as) l" "a In l andb Indot as l".
4. "**Ideal**" is of type "(tokstar => boole) => list tokstar => tokstar => boole":

   "Ideal l as a" "((Indot as l) imp (Con as)) andb ((Indot as l andb Ent as a) imp (a In l))".

Decidable ideals of **D**

1. "I" is a variable of type "tokstar => boole".
2. "**In**" is of type "tokstar => (tokstar => boole) => boole":

   "a In I" "I a".
3. "**Indot**" s of type "list tokstar => (tokstar => boole) => boole"))"

   "Indot (Nil tokstar) I" "True"
   "Indot (a::as) I" "a In I andb Indot as I".
4. "**Ideal**" is of type "(tokstar => boole) => list tokstar => tokstar => boole":

   "Ideal I as a" "((Indot as I) imp (Con as)) andb ((Indot as I andb Ent as a) imp (a In I))".

Theorem

"allnc as(TotalList as $\rightarrow$ Con as $\rightarrow$ allnc bs(TotalList bs $\rightarrow$ allnc a(TotalTokstar a $\rightarrow$ Ideal (Ent as) bs a)))".

Total tokens of **D**

1. "**totalTokstar**" is of type "tokstar => boole":
   "totalTokstar CS" "False"
   "totalTokstar CZ" "True"
   "totalTokstar (CC a b)" "totalTokstar a andb totalTokstar b".

Total tokens of **D**

1. "**totalTokstar**" is of type "tokstar => boole":

   "totalTokstar CS" "False"
   "totalTokstar CZ" "True"
   "totalTokstar (CC a b)" "totalTokstar a andb totalTokstar b".
2. "**Onetotalization**" is of type "tokstar => tokstar":

   "Onetotalization CS" "CZ"
   "Onetotalization CZ" "CZ"
   "Onetotalization (CC a b)" "CC (Onetotalization a) (Onetotalization b)".

Total tokens of **D**

1. "**totalTokstar**" is of type "tokstar => boole":

   "totalTokstar CS" "False"
   "totalTokstar CZ" "True"
   "totalTokstar (CC a b)" "totalTokstar a andb totalTokstar b".
2. "**Onetotalization**" is of type "tokstar => tokstar":

   "Onetotalization CS" "CZ"
   "Onetotalization CZ" "CZ"
   "Onetotalization (CC a b)" "CC (Onetotalization a) (Onetotalization b)".
3. "**sup**" is of type "tokstar => tokstar => tokstar":

   "sup CS b" "b" and "sup b CS" "b"
   "sup CZ CZ" "CZ"
   "sup CZ (CC a b)" "CS"
   "sup (CC a b) CZ" "CS"
   "sup (CC a b) (CC c d)" "CC (sup a c) (sup b d)".

Basic lemma

Lemma

1. *“allnc a(TotalTokstar a → Ent (Onetotalization a ::(Nil tokstar)) a)”.*
2. *“allnc a(TotalTokstar a → allnc b(TotalTokstar b → con a b → Ent (sup a b ::(Nil tokstar)) a))”.*
3. *“allnc a(TotalTokstar a → allnc b(TotalTokstar b → con a b → Ent (sup a b ::(Nil tokstar)) b))”.*
4. *“allnc a(TotalTokstar a → allnc b(TotalTokstar b → allnc c(TotalTokstar c → con a b → con b c → con a c → con (sup a b) (sup b c))))”.*

The program constant Sup

"**Sup**" is of type "tokstar => list tokstar => tokstar":

"Sup a (Nil tokstar)" "a"

"Sup a (b::bs)" "sup (sup a b) (Sup a bs)".

Theorem

"allnc as(TotalList as $\rightarrow$ allnc a(TotalTokstar a $\rightarrow$ allnc b(TotalTokstar b $\rightarrow$ Con (a::b::as) $\rightarrow$ con (sup a b) (Sup a as))))".

Density theorem in **D**

Theorem

1. **Prepredensity**: *"allnc bs(TotalList bs → allnc a(TotalTokstar a → Con (a::bs) → EntList ((Sup a bs)::(Nil tokstar)) (a::bs)))".*
2. **Predensity**: *"allnc as(TotalList as → Con as → exr a(TotalTokstar a ∧ totalTokstar a ∧ EntList (a::(Nil tokstar)) as))".*
3. **Density**: *"allnc as(TotalList as → Con as → exr a(TotalTokstar a ∧ totalTokstar a ∧ Indot as (Ent (a::(Nil tokstar)))))".*

Proof.

(2) is proved by (1), if we take **a = Onetotalization(Sup b bs)**, for a non-empty list b::bs. □

Conclusions

1. A non-trivial portion of mathematics within the set-theoretic Scott-model admits a constructive and formalizable treatment.
2. The implementation enterprise can reveal unexpected analogies. E.g., the analogies in the proofs of $\mathbf{SC_D}$ and $\mathbf{SC_{D \to D}}$ being information systems, which lead to more general results.
3. The implementation can produce new stronger versions of some propositions. E.g., $\mathbf{SC_D}$ and $\mathbf{SC_{D \to D}}$ are split information systems.
4. The implementation enterprise can reveal an expliciteness not always found outside the implementation point of view. E.g., we have a complete description of the total decidable ideal extending a formal neighborhood of $\mathbf{D}$.
5. Our implementation choices and results are applicable to any **specific** finitary algebra.

Next steps

1. To elaborate the implementation of a decidable approximable mapping R in order
2. To complete the implementation of density theorem in $\mathbf{D} \to \mathbf{D}$, which is a larger enterprise than the corresponding implementation in $\mathbf{D}$.
3. To generalize the above implementations to an **abstract** ground algebra (non-trivial).
4. To test TCF^+ with respect to other case studies (e.g., general form of Plotkin's definability theorem).

The up to now TCF$^+$-bibliography

1. S. Huber, B. Karadais and H. Schwichtenberg: Towards a formal theory of computability, in R. Schindler (ed.) *Ways of Proof Theory: Festschrift for W. Pohlers*, 251-276, Ontos Verlag, 2010.

The up to now TCF^{+}-bibliography

1. S. Huber, B. Karadais and H. Schwichtenberg: Towards a formal theory of computability, in R. Schindler (ed.) *Ways of Proof Theory: Festschrift for W. Pohlers*, 251-276, Ontos Verlag, 2010.
2. B. Karadais: Towards a Formal Theory of Computable Functionals, PhD Thesis, LMU, under completion, 2013.

The up to now TCF$^+$-bibliography

1. S. Huber, B. Karadais and H. Schwichtenberg: Towards a formal theory of computability, in R. Schindler (ed.) *Ways of Proof Theory: Festschrift for W. Pohlers*, 251-276, Ontos Verlag, 2010.
2. B. Karadais: Towards a Formal Theory of Computable Functionals, PhD Thesis, LMU, under completion, 2013.
3. H. Schwichtenberg and S. Wainer: *Proofs and Computations*, Perspectives in Logic. Assoc. Symb. Logic and Cambridge University Press, 2012.

The up to now TCF$^+$-bibliography

1. S. Huber, B. Karadais and H. Schwichtenberg: Towards a formal theory of computability, in R. Schindler (ed.) *Ways of Proof Theory: Festschrift for W. Pohlers*, 251-276, Ontos Verlag, 2010.
2. B. Karadais: Towards a Formal Theory of Computable Functionals, PhD Thesis, LMU, under completion, 2013.
3. H. Schwichtenberg and S. Wainer: *Proofs and Computations*, Perspectives in Logic. Assoc. Symb. Logic and Cambridge University Press, 2012.
4. C. Saile: A Theory of Computability in Higher Types, Bachelor-Thesis, LMU, 2013.

The up to now TCF$^+$-bibliography

1. S. Huber, B. Karadais and H. Schwichtenberg: Towards a formal theory of computability, in R. Schindler (ed.) *Ways of Proof Theory: Festschrift for W. Pohlers*, 251-276, Ontos Verlag, 2010.
2. B. Karadais: Towards a Formal Theory of Computable Functionals, PhD Thesis, LMU, under completion, 2013.
3. H. Schwichtenberg and S. Wainer: *Proofs and Computations*, Perspectives in Logic. Assoc. Symb. Logic and Cambridge University Press, 2012.
4. C. Saile: A Theory of Computability in Higher Types, Bachelor-Thesis, LMU, 2013.
5. I. Petrakis: A Theory of Computable Functionals and their Finite Approximations, Ph.D Thesis, LMU, very much under construction.