An Interactive Realizability Semantics for non-constructive proofs

Summer School
"Réalisabilité à Chambéry"
14-17 June, 2011

Toward a model for Classical Logic
through parallel computations
and non-monotonic learning

Stefano Berardi http://www.di.unito.it/~stefano/
C.S. Dept. Torino University

Acknowledgements

- I thank **P. Hyvernat** for inviting me to give a talk to **Chambery Summer School on Realization**, and **C. Raffalli** and **T. Hirschowitz** for organizing the event.
- I thank **F. Aschieri**, **G. Birolo** and **U. de' Liguoro** for checking an earlier version of this talk, and for suggesting several improvements.

If there are mistakes left in these slides they are of the author ...

2

Abstract

- Brower-Heyting-Kolmogorov-Realizability Semantics, from now on **BHK-Realizability**, takes a constructive mathematical proof of the existence of an individual with a given property and ***automatically extracts a certified functional program*** computing the individual. *Extracted programs are readable and may be improved.*
- A proof using Classical Logic (say, by contradiction) may still be interpreted as a program, but in a larger language, including extra features like ***continuation or A-translation***. *Extracted program are often unreadable and hard to improve.*
- Our goal is to define a Interactive Realizability Semantics of Classical Logic, **I-Realizability** for short, based over the idea of ***Learning*** in the limit, in the sense of Gold [Go], which **interprets classical proofs as parallel, non-deterministic programs**, *more readable and easier to improve*.

3

§ 1. Introduction: comparing the Functional paradigm and the Learning paradigm

We outline:

- *the principles of BHK Realizability, interpreting proofs without Excluded Middle as "constructions" in a typed functional language;*
- *the principle of Interactive Realizability, interpreting Excluded Middle as a a learning operator.*

This section only compare the respective ideas, and includes no formal definition. For an introduction to BHK realizability we refer to [Lo], Part 1.

4

Realizability Semantics and functional programs

- In BHK Realizability Semantics, all proofs of B with an hypothesis A are interpreted by recursive maps **f:A→B**, called **realizers**. They take an individual a:A ("a of type A") and return an individual f(a):B ("f(a) of type B"). Proof axioms are interpreted by primitive maps, proof rules are interpreted by compositions of such maps.
- Properties of realizers f:A→B are described by specifications of the form **∀x∈P.f(x)∈Q**, with P⊆A, Q⊆B properties of individuals of type A, B.
- Realizability Semantics defines realizers in a **functional language**, a typed lambda calculus extended with primitive for data types and recursion, called **system *T***.

5

Realizability Semantics and Excluded Middle

- Excluded Middle for a predicate A(x) over natural numbers is the axiom: **EM_A = ∀x.(A(x)∨¬A(x))**
- **EM** is the schema {EM_A | A(x) arithmetical formula}
- A realizer r of EM_A in BHK Realizability Semantic is a map taking some m∈N and returning a triple <b,r_1,r_2>, such that b is Boolean, and if **b=True** then r_1 is a realizer of **A(x)**, if **b=False** then r_2 is a realizer of **¬A(x)**.
- Thus, if there is a realizer of EM_A in BHK Realizability Semantic, then the existence of a realizer for **A(x) is a decidable predicate**. This requirement forbids the existence of a BHK realizer of EM_A for most arithmetical predicates A(x): **EM is false** in BHK Realizability

6

Interactive Realizability Semantics and Monotonic Learning

- In this talk we introduce a more general notion of realizability, **Interactive Realizability**, which interprets classical proofs using **programs learning in the limit** in the sense of Gold [Go], and **monotonic** learning.
- We assume having a countable set Atom of **atoms of information**, a set S of consistent sets of such atoms, and a global **knowledge state** s∈S, common to all realizers.
- Proofs of B with assumption A are interpreted by recursive maps f=<f_1,f_2>: A×S→B×P_{fin}(Atom) we call **"interactive realizers".** An interactive realizer takes a global state s∈S, some a:A, and returns some f_1(a,s):B and some finite set of atoms f_2(a,s)=X⊆Atom, **to be added to s∈S.** We interpret adding X to s as a form of **"monotonic learning".**

7

The interaction between a realizer and the knolwedge state

- Whenever f_2(a,s)=X≠∅ (there is **something to learn**), we recompute **f_2(a,s')=X'⊆Atom** in the new state s' obtained adding X to s. We define in this way some increasing chain **s⊂s'⊂s''⊂...** of states, and we assume that $f_2(a,s^{(n)})$=∅ for some n (that eventually the realizer f has **nothing left to learn**).
- Properties of maps **f=<f_1,f_2>:A×S→B×P_{fin}(Atom)** are described by specifications of the form **∀x∈P. (X=∅)⇒f(x)∈Q**, with P⊆A, Q⊆B. Whenever **X=∅**, that is, "f has nothing left to learn", f behaves **like a construction of BHK-Realizability**, otherwise f extends the knowledge state s by adding X to s.

8

Realizability Semantics and 1-Excluded Middle

EM_1 (1-Excluded Middle) = $\{\forall x.(\exists y. P(x,y) \vee \forall y.P^\perp(x,y))$ | P(x,y) decidable, $P^\perp$ complement of P}

- EM_1 is an axiom schema stronger than constructive Arithmetic, but weaker than EM [Ak]. There is **no realizer of EM_1** in BHK Realizability Semantics.
- There is **a realizer of EM_1 in the I-Realizability.** EM_1 is interpreted as a learning program, a construction of a more general kind than those considered in BHK Realizability.
- In order to interpret full EM as a learning program, we have to consider **non-monotonic learning** (not included in this talk), in which sometimes atoms **are removed from the knowledge state**. In monotonic learning we may only add atoms.

9

§ 2. BHK Realizability Semantics

- *We introduce Goedel's system **T** and a version BHK-(Brouwer-Heyting-Kolmogorov) Realizability in which realizers are terms of **T**.*
- *In the next section, we compare BHK Realizability with the I-Realizability Semantics.*

10

Goedel's system *T* a simply typed λ-calculus

- Goedel's system ***T*** is a **simply typed lambda calculus**, having as atomic types the Data Types:

  Unit={unit}, Bool={True,False}, N={0,1,2,3,..},

  L={nil, cons(n,nil), cons(n,cons(m,nil)), ...} ($n,m \in N$)
- Types of ***T*** are closed under product types $T \times U$ and arrow types $T \to U$. If u:U in ***T***, then $\lambda x^T.u:T \to U$ in ***T***.
- Constants of ***T*** are **if, unit, True, False, 0, Succ,** and primitive recursion **rec_N, rec_L** over integers and lists, n-ple **<.,...,.>** and the i-th projection π_i, with the suitable typing (see [Bo] for more details).
- BHK Realizability Semantics takes an arithmetical proof and turns it into a program written in system ***T.***

11

A Realizability Interpretation of Formulas.

- Let A be any arithmetical formula. We define the type |A| of the realizers of A by induction over A. Let T={Unit,Bool,N,L}.
- $|P(t_1,...,t_m)|$ = Unit
- $|A_1 \wedge A_2|$ = $|A_1| \times |A_2|$
- $|A_1 \vee A_2|$ = $Bool \times |A_1| \times |A_2|$
- $|A_1 \to A_2|$ = $|A_1| \to |A_2|$
- $|\forall x \in T.A|$ = $T \to |A|$
- $|\exists x \in T.A|$ = $T \times |A|$

12

The extraction method implemented in Coq: BHK's Realizability

BHK's realizability is a way of associating to each closed arithmetical formula A all possible programs t:|A| of ***T*** which ***realize*** what the formula ***says.*** We write **t|-A** for "t realizes A", and we call t a BHK **realizer** of A.

Definition (Realizers). Let t be a term of Goedel's system ***T***.

1. **t |- P(t_1,...,t_n)** iff P(t_1,...,t_n) = **True**
2. **t |- A∧B** iff π_0t |- A and π_1t |- B
3. **t |- A→B** iff for all u, if u |- A, then t(u) |- B
4. **t |-** ∀**xA** iff for all n∈N, t(n) |- A[n/x]
5. **t |- A∨B iff π_0t = True, π_1t |- A, or π_0t = False, π_2t |- B**
6. **t |- ∃xA iff π_0t = n and π_1t |- A[n/x]**

13

Informative clauses in BHK's Realizability

- **Clauses 1** of BHK Realizability says that a proof of an atomic formula **P(t_1,...,t_n)** carries no information but the fact that **P(t_1,...,t_n)** is true, and corresponds to a trivial program.
- **Clauses 2-4** of BHK Realizability (**A∧B, A→B,** ∀**xA**) move the information from a realizer to another one, but they produce no new information.
- The clause **3** for **t |- A→B** has the typical form ∀u∈{a |- A}. t(u) ∈{b|- B} used in functional languages.
- **Clause 5** produces some new information: **True∈Bool** whenever the left-hand-side of **A∨B** is realizable, **False∈Bool** whenever the right-hand-side of **A∨B** is realizable.
- **Clause 6** produces some new information: some **n∈N** such that **A[n/x]** is realizable.

14

BHK Realizers interprets proofs without EM as constructions

- There is a general procedure taking an arithmetical constructive proof of A (i.e., a proof <u>without</u> EM), and producing a BHK realizer of A, a program whose ideas mirrors the ideas of the proof. See Appendix for a sketch, [Re], § 1.2 for more details, and [Bo] for a full account.
- If the proof uses Peano Induction, then we decided to express the BHK realizer belongs to the system ***T***.
- If the proof also uses **induction over well-founded** decidable relations, we express the BHK realizer in the system ***T* + fixed point operator**.

15

BHK Realizers allow to compute the witness of an existential statement

- A constructive proof of ∀x∃y.P(x,y), with P any formula, is interpreted by some **r|-∀x∃y.P(x,y)**, which takes some value a for x and return some value b for y such that P(a,b).
- Such a b is called a "witness" of **∃y.P(a,y)**.
- For instance, if L is the type of lists over N, a proof of **∀l∈L.∃m∈L.(Perm(l,m)∧Sorted(m))** is interpreted by a realizer which is a sorting algorithm ([Re], § 2.1).
- The particular sorting algorithm we obtain depends on the idea of the proof: there are proofs corresponding to **InsertSort, MergeSort**, ...

16

BHK Realizers does not interpret EM_1

- We cannot interpret in BHK Realizability Semantics an arithmetical proof including EM.
- The reason is that the boolean and the natural number in Clauses 5, 6 are computed by recursive maps from the parameters of the formula.
- This forbids realizers of some instance **∀x.(∃y.P(x,y) ∨ ∀y.P⊥(x,y))** of EM_1, for some decidable P(x,y).
- Indeed, any realizer of EM_1 should provide a map taking some n∈N and returning True if **∃y.P(n,y)** is realizable, False if **∀y.P⊥(n,y)** is realizable. By Turing's proof of undecidability of the Halting problem, there is no such a map for some **decidable** P(x,y).

17

§ 3. Interactive Realizability Semantics

- *We introduce Goedel's system* ***T*** *extended with knowledge states, then Interactive Realizability Semantics, I-Realizability for short.*
- *We compare I-Realizability with BHK-Realizability Semantics.*

18

The set Atom of atoms of information

- Assume D_1,..,D_n,D are data types in {Unit,Bool, N, L}.
- Let d=d_1,...,d_n. An **atom** is any sequence <P,d,d>, with P:D_1,..,D_n,D→Bool any closed term of ***T***, and d_1∈D_1, ..., a_n∈D_n, d∈D, such that P(d,d)=True in ***T***.
- An atom <P,d,d> includes the information: **∃x∈D.P(d,x) = True** is true, and provides an example of some d∈D such that P(d,d)=True. Such a d∈D is called a **witness** of ∃x∈D.P(d,x)=True.
- We denote with **Atom** the set of all atoms. A set s of atoms is **consistent** if it includes at most one witness for any existential statement ∃x∈D.P(d,x)=True. A set s of atoms is **complete** if it it includes exactly one witness for any existential statement ∃x∈D.P(d,x)=True. Any complete set is infinite and is not recursive.

19

The set S of knowledge states

- **S** is the set of finite consistent sets of atoms.
- S_∞ is the set of (possibly infinite) consistent sets of atoms.
- P_{fin}**(Atom)** is the set of (possibly inconsistent) finite sets of atoms.
- Any s = {<P_1,d_1,d_1>, ..., <P_k,d_k,d_k>} ∈ S includes the information that finitely many **∃x∈D.P(d,x)=True** are true, and exactly one witness for each of them.
- If s∈S includes no witness for ∃x∈D.P(d,x)=True, we say that s "guesses" **∀x∈D.P(d,x)=False**.
- This **"guess"** may be used during the computation of a realizer, but often turns out to be false during the same computation.

20

Merging sets of atoms

- The **merging** of a consistent **$s \in S$** and of **$X \in P_{fin}$(Atom)** is some $s' \subseteq s \cup X$ obtained by selecting one atom <P,d,d>$\in$X for each **$\exists x \in D.P(\underline{d},x)$=True** having no witness in s (such that <P,d,e>$\notin$s for all e$\in$D), and adding it to s.
- **An example of merging.** Let X = {<P,d,d>, <P,d,d'>}.

1. If <P,d_1,...,d_n,e>$\notin$s for all e$\in$D, then the two possible merging of s, X are s'=s$\cup${<P,d,d>} and s'= s$\cup${<P,d,d'>}. We select and add to s one witness for **$\exists x \in D.P(\underline{d},x)$=True**.
2. If <P,d,e>$\in$s for some e$\in$D, then the only possible merging of s, X is s'=s. We do not add a witness for **$\exists x \in D.P(\underline{d},x)$=True** to s, because we already have one.

Merging corresponds to an **(apparent) conflict** in a parallel computation, when two processes having the **same goal** try **to write over the same memory**. It does not matter which process wins: **the goal is fulfilled in any case.** 21

Monotonic Learning and knowledge states

- A program with **monotonic learning** has a state s$\in S_{fin}$, and uses all information and assumptions from s.
- Whenever the program finds some example P(d,d)=True which **falsifies** an assumption $\forall$x$\in$D.P(d,x)=False of s, **it merges the one-element set {<P,d,d>} with s** and restarts **all subcomputations which used this wrong assumption**.
- This idea of monotonic learning is better expressed using processes executed **in parallel** and **non-deterministically**. However, in order to compare I-Realizability with BHK-Realizability, we express monotonic learning programs in some **extension of Goedel's system *T***. The relation between learning programs and functional programs may be made formal in term of **Monads** [Be]. 22

An extension T_S of Goedel's system *T* with knowledge states

We add to Goedel's system ***T*** and to the language of arithmetic the following constants.

- **Atomic types: S** denoting the set of finite consistent sets of atoms, and **P_{fin}(Atom)**, denoting the set of finite sets of atoms.
- One **costant** s for each s$\in$S, and one **constant** X for each X$\in P_{fin}$(Atom)
- The **union** map U:P_{fin}(Atom), P_{fin}(Atom)$\rightarrow P_{fin}$(Atom)

For any P:D_1,..,D_n,D$\rightarrow$Bool closed term of ***T*** we add:

- the **Skolem map**: ϕ_P:S,D_1,..,$D_n \rightarrow$D,
- the **oracle** χ_P: S,D_1,..,$D_n \rightarrow$Bool
- the **update map**: Add_P: S,D_1,..,D_n,D$\rightarrow P_{fin}$(Atom) 23

Reduction rules for T_S

T_S is defined by adding to ***T*** the algebraic reductions corresponding to the following equations:

1. U(X,Y) = **X$\cup$Y**
2. Add_P(s,d,d) = **{<P,d,d>}$\in P_{fin}$(Atom)** if P(d,d) = True and <P,d,d>$\in$s for no d$\in$D, **=$\emptyset \in P_{fin}$(Atom)** otherwise.
3. ϕ_P(s,d)=**d$\in$D** if <P,d,d>$\in$s, =some dummy value **$d_0 \in$D** otherwise.
4. χ_P(s,d)**=True** if <P,d,d>$\in$s for some d$\in$D, **=False** otherwise.

- U is union map, Add_P adds atoms to the knowledge state.
- ϕ_P is a Skolem map providing a witness for **$\exists x \in D.P(\underline{d},x)$ = True** if any exists **in s**, χ_P is an oracle deciding whether **$\exists x \in D.P(\underline{d},x)$=True** is true **using s**. The maps ϕ_P, χ_P, **are relativized to some s$\in$S**.

24

Some examples for χ_P and ϕ_P

- Let **s={<P,0,13>, <P,13,205>}**. Assume **P:N,N→Bool** is a binary closed term of T_S.
- We have χ_P**(s,0)=True** and ϕ_P**(s,0) =13,** because **<P,0,13>∈s**.
- We have χ_P**(s,13)=True** and ϕ_P**(s,13)=205,** because **<P,13,205>∈s**.
- We have χ_P**(s,205)=False** and ϕ_P**(s,205)=**some dummy value∈N, because **<P,205,m>∉s** for all m∈N.
- Even if χ_P**(s,205)=False**, we might have **∃x∈N.P(205,x)=True** because, say, **P(205,133)=True** but s "does not know it": by this we mean: **<P,205,m>∉s** for all m∈N.

25

The Skolem maps and the oracles of T_S may be wrong

- The Skolem maps ϕ_P(s,d) and the oracle χ_P(s,d) of T_S have an extra argument s, they use the information and the guesses from s, and they are **computable**, while ordinary Skolem maps are not.
- The price to pay is that ϕ_P(s,d) may fail to produce a witness for **∃x∈D.P(d,x)=True** even if some exists, in the case no such witness is available in s.
- For the same reason, we may have χ_P(s,d)=False even if **∃x∈D.P(d,x)=True** is true, if no witness for such statement is available in s.
- The outputs of ϕ_P, χ_P rely on the guesses made by s, which may turn out **to be wrong**. However, thorugh a learning mechanism, a realizer of a of simply existential formula in T_S will eventually return a **correct witnesses** for the formula.

26

Terms and formulas having a "hole" of type S

- We call each s∈S a (finite) ***"knowledge state".***
- We call S-terms and S-formulas all terms **t[.]** and formulas **A[.]** having a free variable **(.):S** as unique subterm of type S.
- We denote by **A[s]** the result of replacing (.) with some **s:S.** The constant s is the only subterm of type S in t[s], A[s], and it represents the **current knowledge state** of the realizer t[.] and of the formula A[.].

Interactive Realizability (w.r.t. a knowledge state s)

- For each arithmetical formula we define a type ||A|| for the interactive realizers of A. The definition is the same as in BHK, but in the case of an atomic formula, in which we choose: $||P(t_1,\ldots,t_m)|| = P_{fin}(\text{Atom})$. Interactive realizers of atomic formulas are (possibly inconsistent) **sets of atoms**, while BHK realizers of atomic formulas are dummy constants.
- For any S-term t, S-formula A such that t:||A||, we define a realizability notion $\mathbf{t} \Vdash_s \mathbf{A}$, to be read: "**t** realizes **A** w.r.t. to a knowledge state **s∈S".** We call it **"Interactive Realizability".**
- The goal of a realizer **t** w.r.t. the knowledge state **s∈S** is to **interact** with the global knowledge state **s**, extending it in order to make the formula **A** true.

Interactive Realizability (w.r.t. a knowledge state s)

The definition of Interactive Realizability is by induction over the S-formula **A**. Differences with BHK's Realizability are marked **red**.

1. **t ||-$_s$ P(t_1,…,t_k)** iff **t[s]–s = ø implies P(t_1,…,t_k)[s] = True**
2. **t ||-$_s$ A∧B** iff π_0**t ||-$_s$ A** and π_1**t ||-$_s$ B**
3. **t ||-$_s$ A→B** iff for all **u ||-$_s$ A** we have **tu ||-$_s$ B**
4. **t ||-$_s$** ∀**xA** iff for all **n**∈**N** we have **tn ||-$_s$ A[n/x]**
5. **t ||-$_s$ A∨B** iff either π_0**t[s]=True** and π_1**t ||-$_s$ A,** or π_0**t[s]=False** and π_2**t ||-$_s$ B**
6. **t ||-$_s$** ∃**xA** iff π_0**t[s]=n** and π_1**t ||-$_s$ A[n/x]**

t ||- A iff ∀**s**∈**S. t ||-$_s$ A**

29

Atomic formulas in Interactive Realizability

- The clause **1** for **t ||-$_s$ P(t_1,…,t_k)** has the form ∀**s**∈**S. t[s]-s=ø implies P(t_1,…,t_k)[s]= True**. It is the only clause different from BHK Realizability.
- Clause 1 defines the following loop, which we call **the learning loop**: the realizer t merges some **∅⊂X⊆t[s]-s** with s, forming s', then some ∅⊂X'⊆t[s']-s' with s', forming s", and so forth, producing some increasing chain **s⊂s'⊂s''⊂…** of states.
- If and when we have ∅=t[$s^{(n)}$]-$s^{(n)}$ (no fresh atoms are added to $s^{(n)}$) we reached some state $s^{(n)}$ in which, according to clause 1, we have **P(t_1,…,t_k)[$s^{(n)}$]= True**. We may prove that if a realizer is **extracted from a proof**, then **t[s]∩s=∅** for all **s**∈**S**, in this case **the loop ends when ∅=t[$s^{(n)}$].**

30

The Fixed Point Theorem

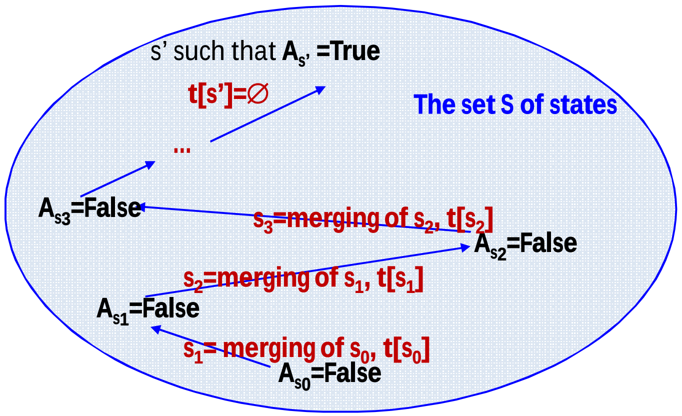
- We may prove (using the fact that the map s:S|→t[s]: P_{fin}(Atom) is continuous w.r.t. the Scott topology over S) the following Fixed Point result, which guarantees termination of the learning loop.

Fixed Point Theorem. Assume t[.]:P_{fin}(Atom) is any S-term. Then any sequences s⊂s'⊂s''⊂… of states defined by $s^{(i)}$⊂ $s^{(i+1)}$ ⊆ some merging of $s^{(i)}$, t[$s^{(i)}$] for all i, ***terminates in*** ***∅=t[$s^{(n)}$]-$s^{(n)}$, for some n.***

In the next slide we represent one possible learning loop associated to a realizer t[.] validating an atomic formula A. In this particular loop we add the **maximum possible of atoms** at each step. By Fixed Point Theorem, though, we are not forced to add the maximum of atoms at each step.

31

A possible learning loop for a realizer t[.] of an atomic formula A



Informative clauses in Interactive Realizability

- **Clauses 2-4** of Interactive Realizability (**$A \wedge B$, $A \rightarrow B$, $\forall xA$**) move some information from a realizer to another one, but they produce no new information.
- **Clause 5** produces some new information: if **$\pi_0 t[s]=True \in Bool$** then the left-hand-side of **$A \vee B$** is realizable, if **$\pi_0 t[s]=False \in Bool$** then the right-hand-side of **$A \vee B$** is realizable.
- The value **$\pi_0 t[s]$** (and the side of **$A \vee B$** which we realize) may change as the knowledge state increase. By a continuity argument, in any increasing chain $s \subset s' \subset s'' \subset \ldots$ of states, **$\pi_0 t[s]$ is eventually stationary** either to **True** or to **False**. In general, **$\pi_0 t[s]$** is not stationary **to the same value True (or False)** on all sequences.

Informative clauses in Interactive Realizability

- **Clause 6** produces some new information: some witness **$\pi_0 t[s]=n \in N$** of **$\exists xA$** (some **$n \in N$** such that **$A[n/x]$** is realizable).
- The witness **$\pi_0 t[s] \in N$** of **$\exists xA$** may change as the knowledge state increases. By a continuity argument, in any increasing chain $s \subset s' \subset s'' \subset \ldots$ of states, **$\pi_0 t[s]$ is eventually stationary** to some value **n_0** (not to the same **n_0** on all sequences, though).
- The term **$\pi_0 t[s]$** has a multi-value limit, one for each increasing chain $s \subset s' \subset s'' \subset \ldots$ of states.
- These different limit values arise in different computations, therefore are not in contradiction each other.

Interactive Realizers interprets proofs with EM1 as learning programs

- We may interpret proofs using EM_1 into Interactive Realizers of system $\boldsymbol{T}_S$ (of **$\boldsymbol{T}_S$ + fixed point operators** if the proof uses well-founded induction). The procedure is **almost the same** interpreting arithmetical constructive proof of A into BHK realizer of A. There are two differences:

1. We change BHK interpretation of **"atomic rules"**, that is, of all rules having atomic premises and conclusion. For instance: reflexivity, symmetry and transitivity of equality.
2. We produce **an interactive realizer of EM_1**.

- We explain these changes in the next slides. We refer to the Appendix for a sketch of the interpretation of proofs into interactive realizers, and to [As],[As3] for a full account.

The interactive realizer associated to an atomic rule

$$\frac{\begin{matrix}\ldots & & \ldots \\ r_1[s] \,||\!- P_1(\underline{t}_1) & \ldots & r_n[s] \,||\!- P_m(\underline{t}_m)\end{matrix}}{r_1[s]\ U \ldots U\ r_n[s] \,||\!- P(\underline{t})}$$

- **Why is it correct to take the union of all realizers?** In order to reach a state in which $P(\underline{t})$ is true it is enough to reach a state in which $P_1(\underline{t})$, ..., $P_n(\underline{t})$ are true, i.e., a state s in which ***$r_1[s] = \ldots = r_n[s] = \emptyset \in P_{fin}(Atom)$***.
- If we define ***$r[s] = r_1[s]\ U \ldots U\ r_n[s]$***, when $r[s]=\emptyset$ we have ***$r_1[s] = \ldots = r_n[s] = \emptyset \in P_{fin}(Atom)$***, therefore $P_1(\underline{t})$, ..., $P_n(\underline{t})$ are true, hence $P(\underline{t})$ is true. Thus, $r[s] \,||\!- P(\underline{t})$.

How does a realizer work for an atomic rule?

...   ...

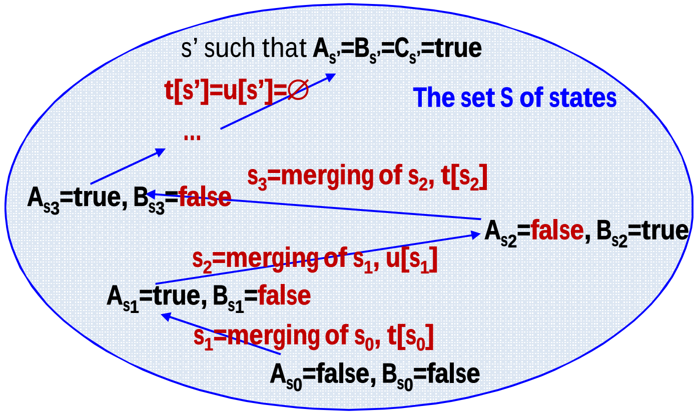
***t[s]*: A**   ***u[s]*: B**

***t[s] U u[s]*: C**

- The realizer ***r[s] = t[s] U u[s]*** searches for some s such that r[s]=∅, that is, ***t[s]=u[s]=∅*** , in order to validate the atomic formulas A and B at the same time, and C as a consequence.
- The search for s such that ***t[s]=u[s]=∅*** terminates in finitely many steps by the Fixed Point Theorem. However, this search may be more complex than just searching for some s such that ***t[s]=∅*** . For instance, if we look first for a state in which ***t[s]=∅ and A is true*** it might be that ***u[s]≠∅ and B false***, and conversely (see next slide for an example).

37

A possible *learning loop* for a realizer r[.]=t[.]Uu[.] of the conclusion of an atomic rule



The interactive realizer of EM_1

- A realizer E_P |\|- **∀x.(∃y.P(x,y) ∨ ∀y.P⊥(x,y))** of an instance of EM_1 may be defined by

  $E_P[s](x)$ = <χ_P(s,x), <ϕ_P(s,x), ∅>, λy.Add_P(s,x,y)>
- Given any value n for x, the realizer $E_P[s](n)$ returns the truth value $\chi_P(s,n)$, that is, the assumption made by s about the truth value of **∃y.P(n,y)**, and a realizer either of **∃y.P(n,y)**, or of **∀y.P⊥(n,y)**, according to the truth value of $\chi_P(s,n)$.
- ***Assume χ_P(s,n)=True.*** Then s has some atom <P,n,m> proving **∃y.P(n,y)**. In this case <ϕ_P(s,n),∅> is a realizer of **∃y.P(n,y)**: indeed, m=ϕ_P(s,n) is a witness of **∃y.P(n,y)**, and ∅:P_{fin}(Atom) is a realizer of **P(n,m)**, because P(n,m) is true.

39

The learning loop for EM_1

- ***Assume χ_P(s,n)=False.*** Then s has no atom <P,n,m> proving **∃y.P(n,y)**, and λy.Add_P(s,n,y) is a realizer of **∀y.P⊥(n,y)**, that is, for any m, Add_P(s,n,m) realizes **P⊥(n,m)**. Indeed:

1. If m is a witness of **∃y.P(n,y)**, then the realizer E_P learns that **∃y.P(n,y)** is true. **Add_P(s,n,m)** returns the singleton **{<P,n,m>} ∈ P_{fin}(Atom),** to be merged to the state s.
2. If m is no witness, then P(n,m) is false, and therefore **P⊥(n,m)** is trivially realizable by **Add_P(s,n,m)=∅ ∈ P_{fin}(Atom).**

Remark that the behaviour of each instance $E_P(n)$ of the realizer E_P is quite simple. $E_P[.](n)$ may add at most one atom <P,n,m> to the knowledge state s.

40

Program extraction in Interactive Realizability

- Any proof of ∀x∃**y.P(x,y)**, with P **atomic**, and using EM_1, is interpreted by some **r||-∀x∃y.P(x,y)**, which takes some value a for x, and returns some value **b[s]=π_0r(a)[s]** for y and some state-extending operator **t[s]=π_1r(a)[s]**:P_{fin}(Atom).
- After a finite loop of state-extending operations, we may reach some s such that t[s]=∅: for such an s, the value of b[s] is a **witness** of ∃**y.P(a,y)**.
- Interactive Realizability provides a model of the fragment EM_1 of classical logic in which **all connectives, including ∨, ∃, are interpreted as in BHK Realizability**. This is not the case with all other constructive interpretations of classical logic.
- The model is a conservative extension of BHK Realizability model for formulas ∃**y.P(a,y)** with P atomic.

41

The interpretation of ∃y.P(x,y) for a non-atomic P

- Any proof of ∀x∃**y.P(x,y)**, with P **not atomic**, and using EM_1, is interpreted by some **r ||- ∀x∃y.P(x,y)**, which takes some value a for x, and returns some value **b[s]=π_0r(a)[s]** for y and some realizer **t[s]=π_1r(a)[s]** of **P(a,b)**.
- By a continuity argument, we may prove that b[s] stabilizes to some limit value v on all increasing chains s⊂s'⊂s''⊂... of states (not to the same value v on all sequences, though).
- We may prove that v is a **witness** of ∃**y.P(a,y)** only if the knowledge state s = $\cup_{n\in N} s^{(n)} \in S_\infty$ limit of the chain is **complete**. The limit v over the chain is **not computable** from the input a when P is not atomic.

42

What is the use of a witness "in the limit"?

- A realizer **r ||- ∀x∃y.P(x,y)** for a non-atomic P has an interest in computations, even if it provides a witness **b[s]=π_0r(a)[s]** only in the limit of an increasing chain s⊂s'⊂s''⊂... of states, and even if this limit is not computable in general.
- Indeed, assume that we use ∀x∃**y.P(x,y)** as a Lemma to prove a goal ∃**z.Q(z)**, with Q atomic. Then, by a continuity argument, we may prove that we only need to know the value of b[s] over some finite state s∈S in order to compute a witness c for ∃**z.Q(z)**. We do not have to compute the limit of b[s], we **only have to know some "approximation" of b[s]** in some finite state in order to fulfill our goal ∃**z.Q(z)**.

43

Summing up

- Interactive Realizability w.r.t. a knowledge state interprets a classical proof of an existential statement ∃**y.P(a,y)** with P atomic as a realizer finding a witness, and using a knowledge state **s∈S** increasing with time.
- Whenever the proof, by Excluded Middle, uses the truth value of a formula **∃y.P(n,y)** which is not known in s, the realizer makes a **guess∀y.P$^\perp$(n,y)** about this truth value.
- If and when the realizer finds a witness m for the opposite statement **∃y.P(n,y)**, it **merges {<P,n,m>} with the current knowledge state s**. Then it removes all subcomputations built over the guess **∀y.P$^\perp$(n,y)**.
- **Program extracted from classical proofs are associated to many state-extending operators C, C', C", ...**

44

Comparing Realizability and Games models for Classical Logic

- Interactive Realizability is a Realizability model of EM_1 and monotonic learning. Interactive Realizability **originates from Game-Theoretical model of EM_1 and monotonic learning**, which uses the idea of 1-backtracking [As2].
- **"Backtracking"** in Game Theory is the possibility for a player of coming back finitely many times to a previous position of the play and changing his/her move from it. Adding backtracking to Game Theory allows us to model full Classical Logic [Coq].
- **"1-Backtracking"** is a restricted form of backtracking, when coming back to a previous move is an **irreversible choice**. 1-backtracking models the fragment EM_1 of EM [Be1], [Be2], [Be3].

45

"Retracting" and Classical Logic: a mathematical study

- The common ground between Interactive Realizability and Game Theory with backtracking is the possibility of **"retracting a previous choice"**: retracting a guess in Interactive Realizability, retracting a previous move in Game Theory with backtracking.
- The notion of **retracting may be studied as a mathematical notion**, without any reference to Realizability, nor to Game Theory.
- It turns out that retracting is a suitable notion for defining a **constructive model of Predicative Classical Arithmetic** [Be4] and of **non-monotonic learning**.

46

Bibliography

[Ak] Y. Akama, S. Berardi, S. Hayashi, U. Kohlenbach, ***An Arithmetical Hierarchy of the Law of Excluded Middle and Related Principles***. LICS 2004, pages 192-201.

[As] F. Aschieri and S. Berardi. ***A Realization Semantics for EM1-Arithmetic.*** TLCA, 2009.

[As2] F. Aschieri. ***Learning Based Realizability for HA + EM1 and 1-Backtracking Games: Soundness and*** **Completeness.** Submitted to APAL.

[As3] F. Aschieri. ***Learning, Realizability and Games in Classical Arithmetic.*** Ph. D. thesis, Torino, 2011.

47

Bibliography

[Be] S. Berardi and U. de' Liguoro. ***Interactive realizers. A new approach to program extraction from non constructive proofs.*** To appear on TOCL 2011:

http://tocl.acm.org/accepted/451deLiguoro.pdf

Interactive Realizers and Monads (Preliminary and simpler version of the journal paper, 2010):

http://www.di.unito.it/~deligu/papers/InterRealMonads.pdf

[Be1] S. Berardi, T. Coquand, and S. Hayashi. ***Games with 1-backtracking.*** APAL, 2010.

[Be2] S. Berardi and M. Tatsuta. ***Positive Arithmetic Without Exchange Is a Subclassical Logic.*** In Zhong Shao, editor, APLAS, volume 4807 of Lecture Notes in Computer Science, pages 271-285. Springer, 2007.

48

Bibliography

[Be3] S. Berardi and Y. Yamagata. ***A Sequent Calculus for Limit ComputableMathematics.*** APAL, 153(1-3):111-126, 2008.

[Be4] S Berardi, U. de' Liguoro: ***Toward the interpretation of non-constructive reasoning as non-monotonic learning***. Inf. Comput. 207(1): 63-81 (2009)

[Bo] L. Boerio ***"Optimizing Programs Extracted from Proofs"***. Ph. D. Thesis, C. S. Dept. Turin University, 1997: **http://www.di.unito.it/~stefano/BoerioPhD-OptimizingProgramsExtractedFromProofs.ps**

[Coq] T. Coquand. ***A Semantics of Evidence for Classical Arithmetic.*** JSL, 60(1):325-337, 1995.

Bibliography

[Go] E. M. Gold, ***Limiting Recursion***, Journal of Symbolic Logic 30, p. 28-48 (1965)

[Lo] J. Longley, ***Realizability: a short course***, Summer School "Réalisabilité à Chambéry", 2011. Part 1.

[Re] ***Realizability: Extracting Programs from proofs,*** Summer School of Bertinoro, Italy, 2007, § 1.2 and § 2.1:

http://www.di.unito.it/~stefano/Berardi-ProgramExtraction-SummerSchool-Bertinoro2007.ppt

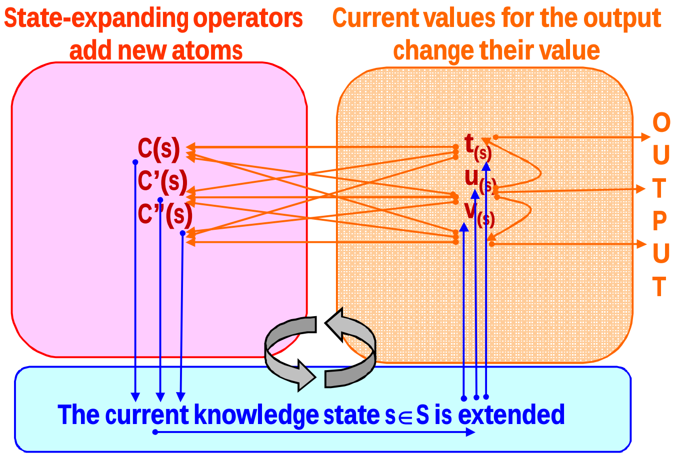
§ 4. Learning and Parallel computations: an example

- *We introduce some classical proofs of simple existential statements, and we use Interactive Realizability in order to extract a non-trivial program mirroring the ideas from the proof.*
- *We stress that we if allow non-determinism and parallelism in our interpretation, we may extract different and subtler programs from the same proofs.*

Learning and Parallel computations

- Evaluating the **learning loop** associated to an interactive realizers require the study of a **parallel computation**.
- The reason is that such a realizer may be the **union** of state-extending operators C, C', C", ..., which may **return at the same time** different and possibly alternative witnesses to be added to our knowledge base.
- We obtain a different result if we add **one witness** at the time, sequentially, or **many witnesses in parallel**.
- Two witnesses of the same statement may be **in (apparent) conflict** with each other, and may require a non-deterministic choice. However, we may prove: if we start from a logically correct proof we obtain a **correct**, **terminating** and **deadlock-free** parallel computation.

A picture of the learning loop



The Minimum Principle

- Assume f:N→N is any map. A **minimum point** of f is any x∈N such that f(x)≤f(y). The minimum principle is the statement **Min=∃x∀y.f(x)≤f(y)**, that is, **"f has a minimum point"**.
- A BHK realizer of Min should define some computable functional F[f], taking a parameter f:N→N, and returning some minimum point n=F[f] of f.
- By a continuity argument, we may show that a computable functional F should produce a minimum point n **out of finitely many values of f.** This is impossible for some f. Thus, there is no such an F, and no BHK realizer of Min.
- We describe now a classical proof of Min, then the interactive realizer of Min extracted from it.

54

A classical proof of Min

- Assume f:N→N is any map. We prove **Min=∃x∀y.f(x)≤f(y)** using EM_1, by induction over the well-founded relation **P(x,y)≡(f(x)>f(y)).**
- **A Proof of Min by EM_1 and induction over P.** We assume that if f(y)<f(x) for some y, then Min holds, and we have to prove Min. We use EM_1 on P and x: **∃y.f(x)>f(y) ∨ ∀y.f(x)≤f(y)**, and case reasoning. ***Left-hand-side***. If **∃y.f(x)>f(y)**, we pick some y such that f(x)>f(y), we apply the induction hypothesis on y, and we deduce Min. ***Right-hand-side.*** If **∀y.f(x)≤f(y)**, then x is a minimum point of f. **Q.E.D..**
- If we express this proof in Natural Deduction, then we apply the translation sketched in the Appendix, we obtain an interactive realizer **r[s] = <μ_0[s], λy∈N.C(y)[s]>**, with **μ_0:N**, and **C||-∀y.f(μ_0)≤f(y)** a realizer.

55

An iteractive realizer of Min

We define the interactive realizer **r[s] = <μ_0[s], λy∈N.C(y)[s]>**, with **μ_0:N**, and **C||-∀y.f(μ_0)≤f(y)**. Let **P(x,y)≡(f(x)>f(y)).**

1. The axiom EM_1 on P, x is translated by **χ_P(s,x)**.
2. Case reasoning is translated by **if(χ_P(s,x), .,.)**.
3. In the case **∃y.f(x)>f(y)** (when **χ_P(s,x)=True**) we pick an y such that **f(x)>f(y)** with the Skolem map **ϕ_P(s,x)**, then we translate ind. hyp. by a **recursive call μ(ϕ_P(s,x)).**
4. In the case **∀y.f(x)≤f(y)** (when **χ_P(s,x)=False**), x is a minimum point of f, and we return **x**.

The realizer C(m) of **f(μ_0)≤f(m)** is an instance of the *right-hand-side of EM_1*, and is equal to update map **Add_P(s,μ_0,m)**. In the case **f(μ_0)>f(m)** (the guess **∀y.f(μ_0)≤f(y) is wrong**) **C(m)[s]** adds the atom **<P,n,m>** with **n=μ_0[s]** to the knowledge state s.

56

Defining a Realizer of the Minimum Principle

- Let $P(x,y) \equiv (f(x)>f(y))$. We define r||-Min by **$r[s] = <\mu_0[s], \lambda y \in N.C(y)[s]>$**, with **$\mu_0$:N**, and **$C$||-$\forall y.f(\mu_0) \leq f(y)$** a realizer.

1. **$\mu(x)[s] = if(\chi_P(s,x), \mu(\phi_P(s,x)),x)$:N**
2. **$\mu_0=\mu(0)$:N**
3. **$C(y)[s] = Add_P(s,\mu_0,y)$ ||- $f(\mu_0)\leq f(y)$**

- Let **s={<P,0,13>, <P,13,205>}**. In §2 we checked that: **$\chi_P(s,0)$=True, $\phi_P(s,0)$=13, $\chi_P(s,13)$=True, $\phi_P(s,13)$=205** and **$\chi_P(s,205)$=False**. Thus, **$\mu_0=\mu(0)$** =(by $\chi_P(s,0)$=True) **$\mu(\phi_P(s,0))$ = $\mu(13)$** =(by $\chi_P(s,13)$=True) **$\mu(\phi_P(s,13))$ = $\mu(205)$** =(since $\chi_P(s,205)$=False) **205**. s “guesses” that 205 is a minimum point of f because s includes no witness for **$\exists y.f(205)>f(y)$**.
- Let f(205)>f(133). Then **C(205)[s]={<P,205,133>}**: C finds some counterexample to the “guess” of s and adds it to s.

57

Discussing the Realizer of the Minimum Principle

- The component **μ_0**:N returns some **$\mu_0[s]$**, which is a minimum point of f **w.r.t. the knowledge state s**: s makes the “guess” **$\forall y.P^{\perp}(\mu_0[s],y)$**, that is, **$\forall y.f(\mu_0[s])\leq f(y)$**, because it has no evidence of the opposite.
- However, the guess made by s may be wrong, in this case **$\mu_0[s]$** is no minimum point of f, and we have **$f(\mu_0[s])>f(p)$** for some p.
- The realizer **C(m)[s]**:P_{fin}(Atom) asks for some m∈N. In the case we have **$f(\mu_0[s])>f(m)$**, then C(m)[s] adds the atom **$<P,\mu_0[s],m>$** to the knowledge state s: it “learns that **$\mu_0[s]$** is wrong”.

58

What is the use of a witness “in the limit”?

- The interactive realizer **r ||- $\exists x \forall y.f(x)\leq f(y)$** provides a witness for a non-atomic property **$\forall y.f(x)\leq f(y)$.** It has an interest for computations, even if it provides a witness **$\mu_0[s]$** only in the limit of an increasing chain $s \subset s' \subset s'' \subset \ldots$ of states, and even if this limit is not computable in general.
- In the rest of the talk, we use **$\exists x \forall y.f(x)\leq f(y)$** as a Lemma to prove goals of the form **$\exists z.Q(z)$**, with Q atomic. Then, by a continuity argument, we may prove that we only need to know the value of **$\mu_0[s]$** over some finite state s∈S in order to compute a witness c for **$\exists z.Q(z)$**. We will not have to compute the limit of **$\mu_0[s]$**, we will **only have to know some “approximation” of $\mu_0[s]$** in some finite state.

59

An Interactive Realizer for the corollary $\exists x.f(x) \leq f(g_1(x))$ of Min (by T. Coquand)

- Let $P(x,y) \equiv (f(x)>f(y))$. Assume f, g_1:N→N. Consider the unequation (c_1) **$f(n) \leq f(g_1(n))$**. The existence of a solution of (c_1) is a corollary of Min, if we set n=minimum point **μ_0** of f.
- Let C1=C($g_1(\mu_0)$):P_{fin}(Atom). Then **$<\mu_0$,C1>** is an interactive realizer of **$\exists x.f(n)\leq f(g_1(n))$** interpreting the classical proof of existence of a solution. C1 is a state-extending operator, adding the atom **$<P,n,g_1(n)>$** to s whenever **$n=\mu_0[s]$** and **$f(n)>f(g_1(n))$** (i.e., **c_1 is false**). In the new state s’, **$\mu_0[s'] \neq n$.**
- In the next picture, we fix a random choice of f, g_1, then we draw **the only possible computation** finding a solution of the unequation (c_1) using the operator C1. Whenever **c_1 is false**, we write **c1NO.** There is a unique state-extending operator, therefore the computation is **deterministic**.

60

A sample computation for the realizer C1

Let $P(x,y) \equiv (f(x)>f(y))$

Start

c1NO — $f(0)>f(g_1(0))$ — *Suppose $g_1(0)=13$*
$s=\emptyset$, $\mu_0[s]=0$, $C1[s]=\{\langle P,0,13\rangle\}$

c1NO — $f(13)>f(g_1(13))$ — *Suppose $g_1(13)=205$*
$s=\{\langle P,0,13\rangle\}$, $\mu_0[s]=13$, $C1[s]=\{\langle P,13,205\rangle\}$

c1NO — $f(205)>f(g_1(205))$ — *Suppose $g_1(205)=133$*
$s=\{\langle P,0,13\rangle, \langle P,13,205\rangle\}$, $\mu_0[s]=205$,
$C1[s]=\{\langle P,205,133\rangle\}$

133 — $f(133)\le f(g_1(133))$ — *End of computation*
$\mu_0[s]=133$, $s=\{\langle P,0,13\rangle, \langle P,13,205\rangle, \langle P,205,133\rangle\}$,
$C1[s]=\emptyset$

The computation returns only <u>one</u> possible solution: n=133

A sample computation tree for a sequential non-deterministic Realizer

- Let $P(x,y) \equiv (f(x)>f(y))$. Assume f, g_1, g_2, g_3, g_4 :$N\rightarrow N$. Consider the 4-equations system:

  (c_i) **$f(n) \le f(g_i(n))$** (i=1,...,4)
- The existence of a solution for the system $(c_1)\wedge\ldots\wedge(c_4)$ is a corollary of Min, if we set n=minimum point μ_0 of f.
- The interactive realizer interpreting the classical proof of existence of a solution is **$\langle\mu_0,C_1U\ldots UC_4\rangle$**, with **$C_i = C(g_i(\mu_0))[s]$** for i=1,...,4. C_i requires to add the atom **$\langle P, \mu_0[s], g_i(\mu_0[s])\rangle$** to the current state s, whenever it is true (whenever **$f(\mu_0[s]) > f(g_i(\mu_0[s]))$** is true, i.e., **$c_i$ is false**).
- In the next picture, we draw **the tree of all possible computations** finding a solution of this system, using C1,...,C4. Whenever **c_i is false**, we write **ciNO.**

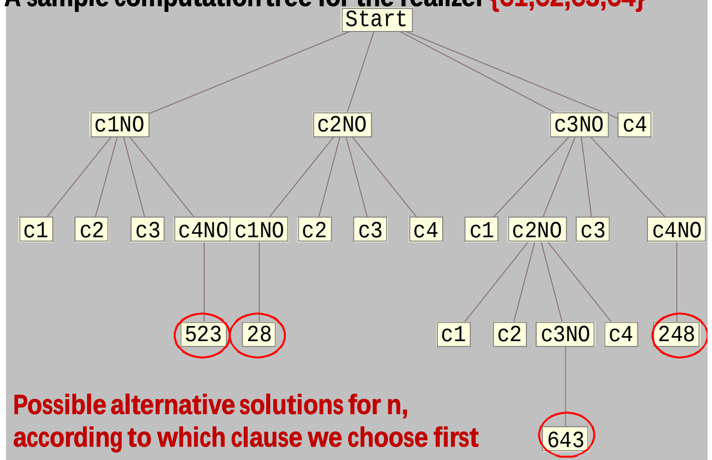
62

A sample computation tree for a sequential non-deterministic Realizer

- Each Ci tries and make **the subgoal (c_i)** true: whenever the current value **$n=\mu_0[s]$** for the minimum of f is wrong, Ci adds the atom **$\langle P, n, g_i(n)\rangle$** to the knowledge state s. As a result, s increases to s', and the current value **n** for the minimum **is replaced by $g_i(n)=\mu_0[s']$.**
- We have a tree of possible computation because the computation is **non-deterministic**. When **more than one c_i is false** we choose which C_i to apply, by choosing one node of the form **ciNO**: the tree forks. The computation is **sequential**: we **can never apply in parallel** Ci,Cj, because two atoms **$\langle P,n,g_i(n)\rangle$, $\langle P,n,g_j(n)\rangle$** define different witness for **$\exists y.f(n)>f(y)$**, hence are inconsistent each other.

63

A sample computation tree for the realizer {C1,C2,C3,C4}



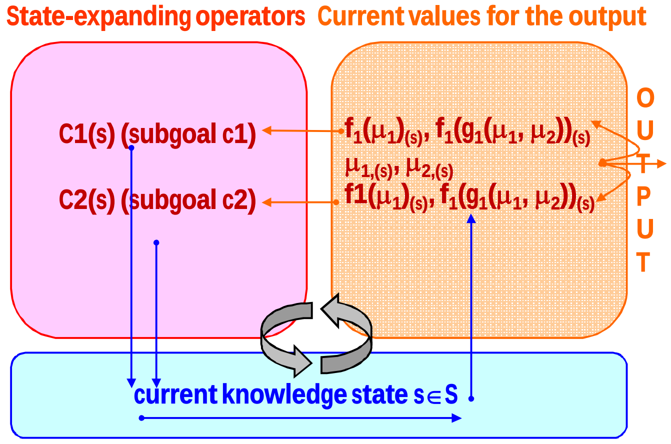
Possible alternative solutions for n, according to which clause we choose first

A realizer corresponding to a parallel program

- **Thesis**: $\forall f_1,f_2:N\to N.\ \forall g_1,g_2:N,N\to N.\ \exists n,m\in N$ s.t.
  - **(c1)** $f_1(n) \le f_1(g_1(n,m))$
  - **(c2)** $f_2(m) \le f_2(g_2(n,m))$
- **Proof (using Min of f_1 and f_2).** n=minimum point μ_1 of f_1, m=minimum point μ_2 of f_2.
- The realizer associated to the proof is $<\mu_1,<\mu_2,C1UC2>>$ with $Ci=C(g_i(\mu_1,\mu_2))$. The current values of n,m rely on **guesses** made by the state s.

1. C1 tries and make **the subgoal (c1)** true: whenever the current value n for the minimum of f_1 is wrong, C1 adds $f_1(n)>f_1(g_1(n,m))$ to the knowledge state. As a result, the current value **n** for the minimum **is replaced by g1(n,m).**
2. C2 tries and make **the subgoal (c2)** true, in the same way.

The learning loop associated to C1, C2



A sample computation tree for a parallel non-deterministic realizer

- In the next picture we assume to be fixed some random maps f_1,f_2,g_1,g_2, and we draw the computation tree for the union realizers **C1 U C2** in a sample case.
- A node labelled **"c1NO"** represent a situation in which the subgoal c1 is false and we apply C1 to try and make c1 true. The same for a node labelled **"c2NO".**
- A node labelled **"c12NO"** represent a situation in which both subgoals c1,c2 are false and we apply C1 and C2 **in parallel** to try and make c1,c2 true. C1, C2, may be applied in parallel, because they produce atoms associated to different existential formulas $\exists y.f_1(n)>f_1(y)$, $\exists y.f_2(n)>f_2(y)$, hence always consistent each other.

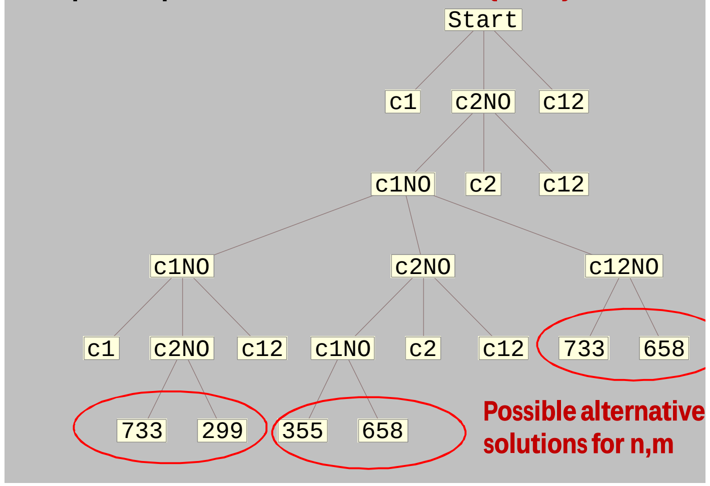
67

A sample computation tree for a parallel non-deterministic realizer (2)

- A node labelled **"c1"** represent a situation in which c1 is true and we cannot apply C1.
- A node labelled **"c2"** represent a situation in which c2 is true and we cannot apply C2.
- A node labelled **"c12"** represent a situation in which either c1 or c2 is true and cannot apply C1, C2 in parallel.
- A pair of leaves of the tree labeled with two integers, say, 733, 299, represent a situation in which the current values **n=733, m=299** for the minimum of f_1,f_2 solve the original problem (w.r.t. some f_1,f_2,g_1,g_2 fixed at random).

68

A sample computation tree for the realizer {C1,C2}



Appendix 1. The interpretation of proofs in the Realizability Semantics

- *We define a mapping sending any proof with EM_1 of A into an interactive realizer of A.*
- *With a minimum of changes the same procedure works for BHK Realizability for Intuitionistic Arithmetic.*

70

Interactive Realizability and BHK Realizability

- We define a map taking an arithmetical proof in natural deduction form of some formula φ, using EM_1 and returns some interactive realizer r|| - φ in Goedel's system $\boldsymbol{T}_S$ extended with states. For a full account we refer to [As], [As3].
- For a description of arithmetic in natural deduction form we refer to [Re].
- The definition of the realizer is by induction over the proof.
- If we change the clauses for atomic formula and we drop the realizer for EM_1 we obtain a procedure which maps an intuitionistic arithmetical proof of φ into a BHK Realizer r| - φ in Goedel's system ***T.***

71

Extending Realizability to more Data Types

- All what we will say applies not just to a language having types:

  T=Unit, Bool, N (Natural Numbers),

  L (Lists of Natural Numbers)
- but also to a language having types

  T=any Bohm-Berarducci Data Types
- We refer Boerio Ph.d [Bo] for a procedure transforming any intuitionistic proof of this extended language into a BHK realizer.

72

Dummy constants.

- For each simple type T of $\mathbf{T}_S$, we we will need some dummy element $dummy^T$:T (just d^T for short), to be used as default value for such type.
- We define d^T:T by induction over T.

1. $dummy^{P_{fin}(Atom)} = \varnothing$
2. $dummy^{Unit}$ = unit
3. $dummy^{Bool}$ = False
4. $dummy^N$ = 0
5. $dummy^L$ = nil
6. $dummy^{T \to U}$ = $\lambda x.\ dummy^U$
7. $dummy^{T \times U}$ = $<dummy^T, dummy^U>$

73

Realizability Interpretation of Formulas.

- Let φ be any closed formula and T={Unit,Bool,N,L}. We recall the definition of the simple type $||\varphi||$ for all interactive realizer r of φ. The definition of $||\varphi||$ is by induction over φ.
- $||P(t_1,\dots,t_m)||$ = $P_{fin}(Atom)$
- $||\varphi_1 \wedge \varphi_2||$ = $||\varphi_1|| \times ||\varphi_2||$
- $||\varphi_1 \vee \varphi_2||$ = $Bool \times ||\varphi_1|| \times ||\varphi_2||$
- $||\varphi_1 \to \varphi_2||$ = $||\varphi_1|| \to ||\varphi_2||$
- $||\forall x \in T.\varphi||$ = $T \to ||\varphi||$
- $||\exists x \in T.\varphi||$ = $T \times ||\varphi||$

We obtain the definition $|\varphi|$ of the type of a BHK realizer of φ if we write $|P(t_1,\dots,t_m)|$ = Unit in the atomic case

74

The Interactive Realizability Interpretation of proofs

- If $\underline{x}=x_1,\dots,x_n$ is a vector of variables of types $T_1,\dots, T_n$, then $|\varphi(\underline{x})| = T_1 \times \dots \times T_n \to |\varphi|$ is the type of all $r || \text{-} \varphi(\underline{x})$.
- Let $\Gamma=\{\varphi_1,\dots,\varphi_n\}$ be a set of assumptions and $\underline{x}=x_1,\dots,x_k$. We write $r || \text{-} (\Gamma | \text{-} \varphi(\underline{x}))$ for: r is an interactive realizer of $\varphi(\underline{x})$, depending on free variables in $\underline{x}$, and on the realizer variables $\xi_1 || \text{-} \varphi_1(\underline{x}), \dots, \xi_k || \text{-} \varphi_k(\underline{x})$.
- We may turn every proof of $\varphi(\underline{x})$, with free assumptions in Γ, possibly using EM_1, into some $r || \text{-} (\Gamma | \text{-} \varphi(\underline{x}))$. Definition is by induction on p, with one clause for each possible rule at the conclusion of p.
- If the proof of $\varphi(\underline{x})$ is purely intuitionistic, we may define a BHK realizer $r | \text{-} (\Gamma | \text{-} \varphi(\underline{x}))$ by changing the case of atomic formulas and removing the definition of a realizer of EM_1.

75

The Interactive Realizer for an Atomic rule

Atomic rules. If the proofs ends by some Atomic rule, then the realizers of the assumptions are state-extending operators, and we take their union to realize the conclusion.

$$\frac{\begin{array}{ccc} \dots & & \dots \\ r_1[s] || \text{-}\ P_1(\underline{t}_1) & \dots & r_m[s] || \text{-}\ P_m(\underline{t}_m) \end{array}}{r_1[s] \cup \dots \cup r_m[s] || \text{-}\ P(\underline{t})}$$

If $r_1[s] || \text{-}\ \Gamma | \text{-} P_1(\underline{t}_1), \dots, r_m[s] || \text{-}\ \Gamma | \text{-} P_m(\underline{t}_1)$***, then*** $r_1[s] \cup \dots \cup r_m[s] || \text{-}\ \Gamma | \text{-}\ P(\underline{t})$

76

The BHK Realizer for an Atomic rule

- ***Atomic rules.*** If the proofs ends by some Atomic rule, then r($\underline{x}$)=***unit***.

$$\frac{\begin{matrix} \ldots & & \ldots \\ \textit{unit} \vdash P_1(\underline{t}_1) & \ldots & \textit{unit} \vdash P_m(\underline{t}_m) \end{matrix}}{\textit{unit} \vdash P(\underline{t})}$$

- ***If*** $\textit{unit} \vdash \Gamma \vdash P_1(\underline{t}_1)$, ..., $\textit{unit} \vdash \Gamma \vdash P_m(\underline{t}_1)$, ***then*** $\textit{unit} \vdash \Gamma \vdash P(\underline{t})$

Interactive/BHK realizers for Conjunction

- ***Rules for*** $\wedge$
- ***Introduction rules:***

$$\frac{s_1 \Vdash \varphi \quad s_2 \Vdash \psi}{\langle s_1,s_2 \rangle \Vdash \varphi \wedge \psi}$$

- ***If*** $s_1 \Vdash \Gamma \vdash \varphi$ ***and*** $s_2 \Vdash \Gamma \vdash \psi$ ***then*** $\langle s_1,s_2 \rangle \Vdash \Gamma \vdash \varphi \wedge \psi$

Interactive/BHK realizers for Conjunction

- ***Elimination rules:***

$$\frac{s \Vdash \varphi \wedge \psi}{\pi_1(s) \Vdash \varphi} \qquad \frac{s \Vdash \varphi \wedge \psi}{\pi_2(s) \Vdash \psi}$$

- ***If*** $s \Vdash \Gamma \vdash \varphi \wedge \psi$, ***then*** $\pi_1(s) \Vdash \Gamma \vdash \varphi$ ***and*** $\pi_2(s) \Vdash \Gamma \vdash \psi$

Interactive/BHK realizers for Disjunction

- ***Rules for*** $\vee$***.*** Let T=True, F=False, and _, _', be the dummy elements of type $||\varphi||$, $||\psi||$ (of type $|\varphi|$, $|\psi|$ in the case of a BHK-realizer)
- ***Introduction rules:***

$$\frac{r \Vdash \varphi}{\langle True,r,\_' \rangle \Vdash \varphi \vee \psi} \qquad \frac{s \Vdash \psi}{\langle False,\_,s \rangle \Vdash \varphi \vee \psi}$$

- ***If*** $r \Vdash \Gamma \vdash \varphi$ ***then*** $\langle True,r,\_' \rangle \Vdash \Gamma \vdash \varphi \vee \psi$
- ***If*** $s \Vdash \Gamma \vdash \psi$ ***then*** $\langle False,\_,s \rangle \Vdash \Gamma \vdash \varphi \vee \psi$

Interactive/BHK realizers for Disjunction

- *Elimination rules for* $\vee$*.* Let
  u = if (i=True) then s(a) else t(b)

Then

$$\dfrac{\langle i,a,b\rangle \,||\!-\, \varphi \vee \psi \qquad \begin{matrix} \xi \,||\!-\, \varphi \\ \dots \\ s(\xi) \,||\!-\, \theta \end{matrix} \qquad \begin{matrix} \eta \,||\!-\, \psi \\ \dots \\ t(\eta) \,||\!-\, \theta \end{matrix}}{u \,||\!-\, \theta}$$

- *If* r||- Γ|-$\varphi \vee \psi$ *and* s(ξ)||- Γ,ξ:φ|-θ *and* t(η)||- Γ,η:ψ|-θ, *then* u||- Γ|-θ

Interactive/BHK realizers for Implication

- *Rules for* $\rightarrow$*. Introduction rule:*

$$\dfrac{\begin{matrix} \xi \,||\!-\, \varphi \\ \dots \\ s(\xi) \,||\!-\, \psi \end{matrix}}{\lambda \xi.s(\xi) \,||\!-\, \varphi \rightarrow \psi}$$

- *If* s(ξ)||- Γ, ξ:φ |- ψ, *then* λξ.s(ξ)||- Γ|- $\varphi \rightarrow \psi$

Interactive/BHK realizers for Implication

- *Elimination rule:*

$$\dfrac{r \,||\!-\, \varphi \rightarrow \psi \qquad s \,||\!-\, \varphi}{r(s) \,||\!-\, \psi}$$

- *If* r||- Γ|- $\varphi \rightarrow \psi$ *and* s||- Γ|- φ, *then* r(s)||- Γ|- ψ.

Interactive/BHK realizers for Existential

- *Rules for* $\exists$*: Introduction rule.*

$$\dfrac{\begin{matrix} \Gamma \\ \dots \\ r \,||\!-\, \varphi[t/x] \end{matrix}}{\langle t,r\rangle \,||\!-\, \exists x \in T.\varphi}$$

- *If* r||- Γ|-φ*[t/x] for some* t, *then* <t,r>||- Γ|-$\exists x \in T.\varphi$

Interactive/BHK realizers for Existential

- ***Rules for ∃: Elimination rule.***

$$\frac{\begin{array}{cc} \Gamma & \Gamma, \xi ||\!- \varphi \\ \ldots & \ldots \\ \langle i,a\rangle ||\!- \exists x\in T.\varphi & t(x,\xi) ||\!- \psi \end{array}}{t(i,a) ||\!- \psi}$$

- *Provided* ***$x \notin FV(\Gamma,\psi)$.***
- ***If $\langle i,a\rangle ||\!- \Gamma |\!- \exists x\in T.\varphi$, $t(x,\xi) ||\!- \Gamma,\xi{:}\varphi |\!- \psi$, and $x \notin FV(\Gamma,\psi)$, then $t(i,a) ||\!- \Gamma |\!- \psi$***

85

Interactive/BHK realizers for Universal

- ***Rules for ∀: Introduction rule.***

$$\frac{\begin{array}{c} \Gamma \\ \ldots \\ r ||\!- \varphi \end{array}}{\lambda x.r ||\!- \forall x\in T.\varphi}$$

- *Provided* ***$x \notin FV(\Gamma)$***
- ***If $r ||\!- \Gamma |\!- \varphi$ and $x \notin FV(\Gamma)$, then $\lambda x.r ||\!- \Gamma |\!- \forall x\in T.\varphi$***

86

Interactive/BHK realizers for Universal

- ***Rules for ∀: Elimination rule.***

$$\frac{\begin{array}{c} \Gamma \\ \ldots \\ f ||\!- \forall x\in T.\varphi \end{array}}{f(t) ||\!- \varphi[t/x]}$$

- ***If $f ||\!- \Gamma |\!- \forall x\in T.\varphi$, then $f(t) ||\!- \Gamma |\!- \varphi[t/x]$ for all t***

87

Interactive/BHK realizers for Induction on Natural Numbers

- ***The Induction Axiom for the type N=Natural Numbers:***

  Ind: $\forall x\in N.(\varphi[0/x] \to \forall x\in N.(\varphi \to \varphi[x+1/x]) \to \varphi)$
- ***The realizer Rec has type:***

  $N \to ||\varphi|| \to (N \to ||\varphi|| \to ||\varphi||) \to ||\varphi||$

BHK realizers have $|\varphi|$ in the place of $||\varphi||$.

- Let $n{:}N$, $r ||\!- \varphi[0/x]$ and $s ||\!- \forall x\in N.(\varphi \to \varphi[x+1/x])$.
- We define $Rec(n,r,s) ||\!- \varphi[n/x]$ by primitive recursion:

1. Rec(0,r,s) = r
2. Rec(n+1,r,s) = s(n,Rec(n,r,s))

88

Interactive/BHK realizers for Induction for Induction on Lists

- ***Induction Axiom for the type L=Lists is:***

$$Ind_L: \forall l \in L.(\varphi[nil/l] \rightarrow \forall l \in L, x \in N.(\varphi \rightarrow \varphi[cons(x,l)/l]) \rightarrow \varphi)$$

- ***We abbreviate $A \rightarrow B \rightarrow \ldots \rightarrow C$ by $A,B,\ldots \rightarrow C$.***
- ***The realizer Rec_L has type:***

$$L, ||\varphi||, (L, N, ||\varphi|| \rightarrow ||\varphi||) \rightarrow ||\varphi||$$

BHK realizers have $|\varphi|$ in the place of $||\varphi||$.

- Let $m:L$, $r \,||\!-\, \varphi[nil/l]$, $s \,||\!-\, \forall l \in L, x \in N.(\varphi \rightarrow \varphi[cons(x,l)/l])$
- We define $Rec_L(m,r,s) \,||\!-\, \varphi[m/l]$ by primitive recursion:

1. $Rec_L(nil,r,s) = r$
2. $Rec_L(cons(n,l),r,s) = s(l,n,Rec_L(l,r,s))$

89

Interactive/BHK realizers for Well-founded Induction

- *Assume R is an atomic arithmetical formula defining some well-founded relation (i.e., there is no infinite R-chain).* ***The Well-founded Induction Axiom for R is:***

$$WInd: (\forall y \in N.(\forall z \in N.R(y,z) \rightarrow \varphi[z/x]) \rightarrow \varphi[y/x]) \rightarrow \forall x \in N.\varphi$$

- ***The realizer W has type (with $|\varphi|$ or $||\varphi||$):***

$$(N \rightarrow (N \rightarrow P_{fin}(Atom) \rightarrow ||\varphi||) \rightarrow ||\varphi||) \rightarrow N \rightarrow ||\varphi||$$

- Let $r \,||\!-\, \forall y \in N.(\forall z \in N.R(z,y) \rightarrow \varphi[z/x]) \rightarrow \varphi[y/x]$ and $n:N$.
- We define $W(r,n) \,||\!-\, \varphi[n/x]$ by fixed point:

$$W(r,n) = r(n, \lambda m{:}N.\lambda s{:}P_{fin}(Atom).W(r,m)) : ||\varphi||$$

The realizer belongs to T_S **+ fixed point operators.** Terms of this system are convergent if we reduce only closed terms which are not in the minor branch of an "if".

90

The Interactive realizer of EM_1

- An interactive realizer $E_P \,||\!-\, \forall x.(\exists y.P(x,y) \vee \forall y.P^{\perp}(x,y))$ of an instance of EM_1 may be defined as in § 3, by

$$E[s](x) = \langle \chi_P(s,x), \langle \phi_P(s,x), \emptyset \rangle, \lambda y.Add_P(s,x,y) \rangle \,||\!-$$
$$\Gamma \,|\!-\, \forall x.(\exists y.P(x,y) \vee \forall y.P^{\perp}(x,y))$$

There is no BHK realizer for EM_1, instead.

91

Talk given at Technolac