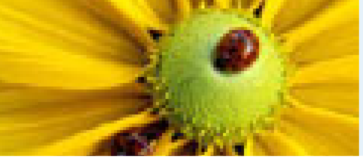
Reversivity, Reversibility and Retractability

Nikolai N. Nepejvoda

July 9, 2012



Brief history 1

Brief history 1
Brief history 2
Brief history 3
Brief history 4
Constructivism as a tool for CS and Informatics
Retractability
Reversibility
Reversivity
Summary

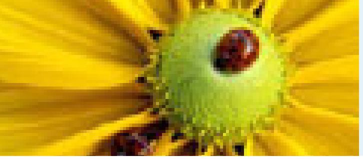
Landauer, von Neumann: Reversivity
Thermodynamic lower bound for information processing is

Generalized Landauer —

von Neumann principle

$$E_{diss} \geqslant T \times k_B \times \ln P$$

k_B is the Bolzmann's constant, P is the number of states of atomic computing element.



Brief history 1

Landauer, von Neumann: Reversivity
Thermodynamic lower bound for information processing is

Generalized Landauer —

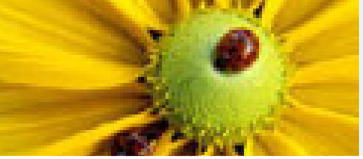
von Neumann principle

$$E_{diss} \geqslant T \times k_B \times \ln P$$

k_B is the Bolzmann's constant, P is the number of states of atomic computing element.
Landauer 1961: to avoid this limit is possible only if our actions are *invertible*

Brief history 2

Bennett 1973: Reversibility Possibility to undo any action

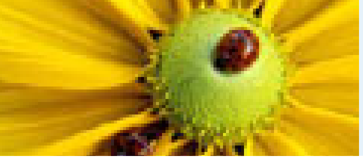


Brief history 2

Bennett 1973: Reversibility Possibility to undo any action
It is possible to emulate any Turing machine by reversible one for the cost of extra time and garbage

$$\text{Time} > 3^k \cdot 2^{O\left(\frac{T}{2^k}\right)} \qquad \text{Store} > S \cdot (1 + O(k)) \tag{1}$$

where k can be chosen between 1 and $\log_2 T$.

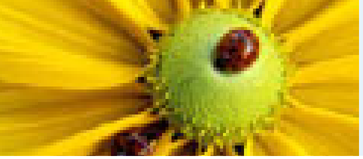


Brief history 2

Bennett 1973: Reversibility Possibility to undo any action
It is possible to emulate any Turing machine by reversible one for the cost of extra time and garbage

$$\text{Time} > 3^k \cdot 2^{O\left(\frac{T}{2^k}\right)} \qquad \text{Store} > S \cdot (1 + O(k)) \tag{1}$$

where k can be chosen between 1 and $\log_2 T$.
Reversibility is not full invertibility: we cannot undo which is not done. Thus reversibility has no relation to LvN principle.

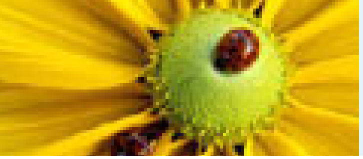


Brief history 3

H. Axelsen, R. Glück 2011: Reversibility is not Turing complete

Brief history 3

H. Axelsen, R. Glück 2011: Reversibility is not Turing complete
By reversible Turing machine we can compute exactly all injective computable function



Brief history 3

H. Axelsen, R. Glück 2011: Reversibility is not Turing complete

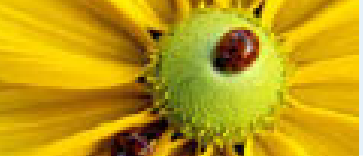
By reversible Turing machine we can compute exactly all injective computable function

There exists an universal reversible Turing machine

Brief history 3

H. Axelsen, R. Glück 2011: Reversibility is not Turing complete
By reversible Turing machine we can compute exactly all injective computable function
There exists an universal reversible Turing machine

T. Toffoli 1980
There is an invertible function $\mathbf{bool}^3 \longrightarrow \mathbf{bool}^3$ (Toffoli gate) which is a basis for all invertible Boolean functions



Brief history 3

H. Axelsen, R. Glück 2011: Reversibility is not Turing complete
By reversible Turing machine we can compute exactly all injective computable function
There exists an universal reversible Turing machine

T. Toffoli 1980
There is an invertible function $\mathbf{bool}^3 \to \mathbf{bool}^3$ (Toffoli gate) which is a basis for all invertible Boolean functions
Different gates are proposed now and extensively studied algorithms to build reversible extensions of usual boolean functions from those gates

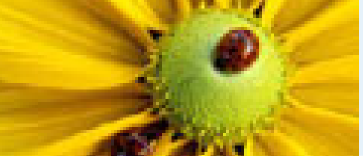
Brief history 4

Retractability: a good companion

Brief history 4

Retractability: a good companion

People extensively studied different method of program inversions

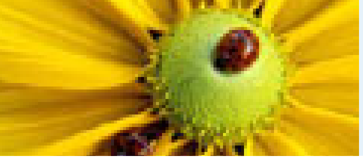


Brief history 4

Retractability: a good companion

People extensively studied different method of program inversions

We not always are to invert a whole program functional. Usually it is sufficient to *retract* some results up to their reasons.



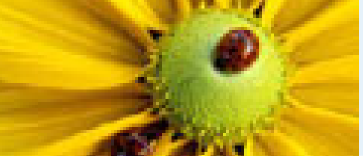
Brief history 4

Retractability: a good companion

People extensively studied different method of program inversions

We not always are to invert a whole program functional. Usually it is sufficient to *retract* some results up to their reasons.

One of practically used kinds of program retraction is error analysis.



Brief history 4

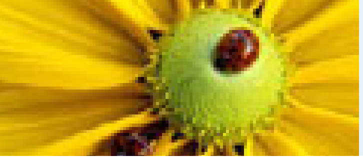
Retractability: a good companion

People extensively studied different method of program inversions

We not always are to invert a whole program functional. Usually it is sufficient to *retract* some results up to their reasons.

One of practically used kinds of program retraction is error analysis.

Practically we need rather to restore conditions than values



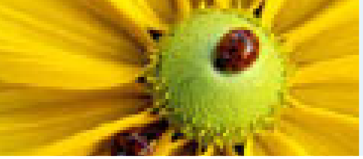
Constructivism as a tool for CS and Informatics

Constructive understanding

Our statements are considered as problems which are to be solved in such a way that ideal abstract but effective construction can be extracted from this solution

Constructive understanding

Our statements are considered as problems which are to be solved in such a way that ideal abstract but effective construction can be extracted from this solution
There are no logical values. Statement is to be *realized* and different proofs can give different realizations.



Constructive understanding

Our statements are considered as problems which are to be solved in such a way that ideal abstract but effective construction can be extracted from this solution

There are no logical values. Statement is to be *realized* and different proofs can give different realizations.

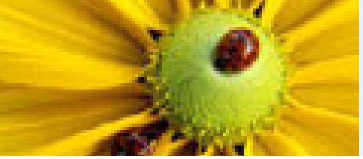
Effectivity is not treated as absolute notion of Turing completeness. We are to construct our result by admissible for the ptoblem tools and by admissible spending of resources

Constructive paradigm

There are no universal methods and silver bullets. When somebody claims that the method can solve everything this persdon is a crook or fanatic or politician or simply a lying advertiser.

Constructive paradigm

There are no universal methods and silver bullets. When somebody claims that the method can solve everything this persdon is a crook or fanatic or politician or simply a lying advertiser.
We are to choose the best tools fitting our problems.



Constructive paradigm

There are no universal methods and silver bullets. When somebody claims that the method can solve everything this persdon is a crook or fanatic or politician or simply a lying advertiser.

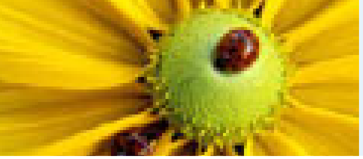
We are to choose the best tools fitting our problems.

Tools for different domains and for different systems of values can be incompatible and using them “in interoperable manner” is a mortal trick.

Example: Curry paradox (1930). Logic is incompatible with λ-calculus.

Constructive paradigm

There are no universal methods and silver bullets. When somebody claims that the method can solve everything this persdon is a crook or fanatic or politician or simply a lying advertiser. We are to choose the best tools fitting our problems. Tools for different domains and for different systems of values can be incompatible and using them “in interoperable manner” is a mortal trick. Example: Curry paradox (1930). Logic is incompatible with λ-calculus. This does not prevent to use different constructive tools in different modules of a single system.



Constructive rationalism

Brief history 1
Brief history 2
Brief history 3
Brief history 4
Constructivism as a tool for CS and Informatics
Constructive understanding
Constructive paradigm
Constructive rationalism
Intuitionistic logic
Intuitionistic logic 2
Intuitionistic logic 3
Restricted constructions
Restricted constructions 2
Retractability
Reversibility
Reversivity
Summary

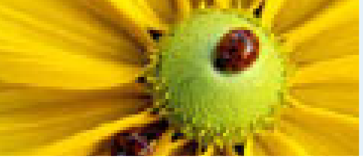
Constructivism is really another form of rational thinking which is alternative to usual “Aristotelian” one.

Constructive rationalism

Constructivism is really another form of rational thinking which is alternative to usual “Aristotelian” one.
We seek solutions instead of “THE HOLY ABSOLUTE TRUTH”

Constructive rationalism

Constructivism is really another form of rational thinking which is alternative to usual "Aristotelian" one.
We seek solutions instead of "THE HOLY ABSOLUTE TRUTH"
Thus we are versatile and allow other people think differently



Constructive rationalism

Constructivism is really another form of rational thinking which is alternative to usual “Aristotelian” one.

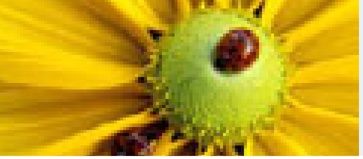
We seek solutions instead of “THE HOLY ABSOLUTE TRUTH”

Thus we are versatile and allow other people think differently

Thus we are ruthless and intolerant because way of thinking of a person makes his values, goals and prejudices explicit. We try to oppose those who uses inadequate tools for dirty purposes. Each person has first of all responsibility and only if he/her is responsible he/her can claim rights.

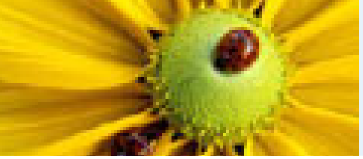
Intuitionistic logic

L. E. J. Brouwer (1908)



Intuitionistic logic

L. E. J. Brouwer (1908)
The language is the same as for classical logic



Intuitionistic logic

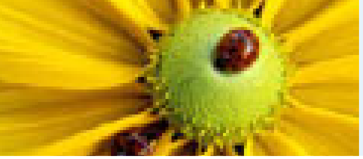
L. E. J. Brouwer (1908)

The language is the same as for classical logic

Formulas are understood as problems

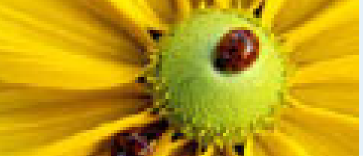
Intuitionistic logic

L. E. J. Brouwer (1908)
The language is the same as for classical logic
Formulas are understood as problems
We are interested in ideal mental constructions.
Our only restriction is that their execution is to be finite and use finite information on arguments.



Intuitionistic logic

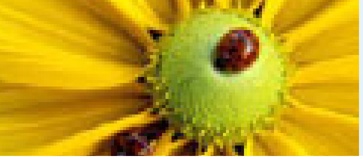
L. E. J. Brouwer (1908)
The language is the same as for classical logic
Formulas are understood as problems
We are interested in ideal mental constructions.
Our only restriction is that their execution is to be finite and use finite information on arguments.
Formal system is the classical logic without $A \vee \neg A$.



Intuitionistic logic

Brief history 1
Brief history 2
Brief history 3
Brief history 4
Constructivism as a tool for CS and Informatics
Constructive understanding
Constructive paradigm
Constructive rationalism
Intuitionistic logic
Intuitionistic logic 2
Intuitionistic logic 3
Restricted constructions
Restricted constructions 2
Retractability
Reversibility
Reversivity
Summary

L. E. J. Brouwer (1908)
The language is the same as for classical logic
Formulas are understood as problems
We are interested in ideal mental constructions.
Our only restriction is that their execution is to be finite and use finite information on arguments.
Formal system is the classical logic without $A \vee \neg A$.
Removing irrelevant supposition ‘We know all’ we get a stronger system which includes the whole classical logic as a isomorphic image (A. Glivenko, 1929)

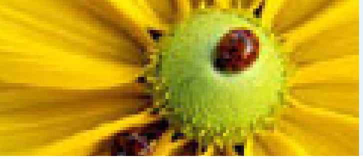


Intuitionistic logic

L. E. J. Brouwer (1908)
The language is the same as for classical logic
Formulas are understood as problems
We are interested in ideal mental constructions.
Our only restriction is that their execution is to be finite and use finite information on arguments.
Formal system is the classical logic without $A \vee \neg A$.
Removing irrelevant supposition ‘We know all’ we get a stronger system which includes the whole classical logic as a isomorphic image (A. Glivenko, 1929)

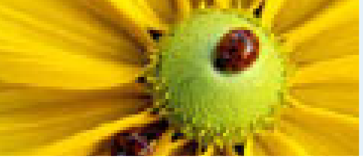
Intuitionistic logic 2

There are new possibilities: to express ignorance and to use it as a positive factor; to express in a short and concise way complex conditions on used tools; to analyse a level of constructivity of theorems and solutions.



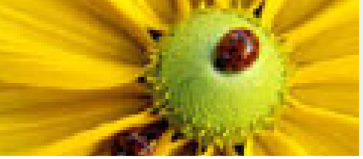
Intuitionistic logic 2

There are new possibilities: to express ignorance and to use it as a positive factor; to express in a short and concise way complex conditions on used tools; to analyse a level of constructivity of theorems and solutions.
But there are no possibilities to express that our resources are restricted and take into account the main resource restriction.



Intuitionistic logic 2

There are new possibilities: to express ignorance and to use it as a positive factor; to express in a short and concise way complex conditions on used tools; to analyse a level of constructivity of theorems and solutions.
But there are no possibilities to express that our resources are restricted and take into account the main resource restriction.
This logic was created as a logic of ideal mental construction and ideally fits to this mental and real domain



Intuitionistic logic 2

There are new possibilities: to express ignorance and to use it as a positive factor; to express in a short and concise way complex conditions on used tools; to analyse a level of constructivity of theorems and solutions.

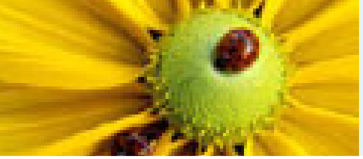
But there are no possibilities to express that our resources are restricted and take into account the main resource restriction.

This logic was created as a logic of ideal mental construction and ideally fits to this mental and real domain

Thus Yessenin-Volpin proposed in 1960 to consider logics for restricted constructions.

Intuitionistic logic 3

NOTE. If we do not insert natural numbers, induction or fixed point intuitionistic logic gives very effective solutions which are linear in time and space **modulo** primitive functions. (Nepejvoda 1979)



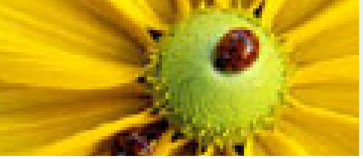
Intuitionistic logic 3

Brief history 1
Brief history 2
Brief history 3
Brief history 4
Constructivism as a tool for CS and Informatics
Constructive understanding
Constructive paradigm
Constructive rationalism
Intuitionistic logic
Intuitionistic logic 2
Intuitionistic logic 3
Restricted constructions
Restricted constructions 2
Retractability
Reversibility
Reversivity
Summary

NOTE. If we do not insert natural numbers, induction or fixed point intuitionistic logic gives very effective solutions which are linear in time and space **modulo** primitive functions. (Nepejvoda 1979)

This is one of partial cases of the common principle:

Worst enemies of a good systems are new possibilities



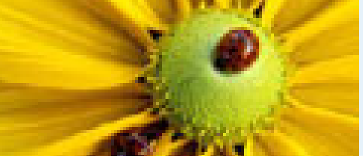
Intuitionistic logic 3

NOTE. If we do not insert natural numbers, induction or fixed point intuitionistic logic gives very effective solutions which are linear in time and space **modulo** primitive functions. (Nepejvoda 1979)

This is one of partial cases of the common principle:

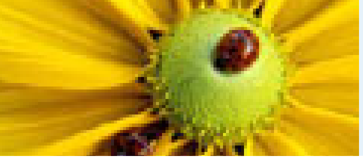
Worst enemies of a good systems are new possibilities

Thus let us do not criticize a system for it cannot do something (e.g. express a factorial) It must work perfectly on its native domain.



Restricted constructions

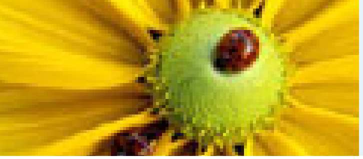
Yessenin-Volpin could not imagine how drastically changed logic after we take into account that “finite in theory means infinite in practice”



Restricted constructions

Yessenin-Volpin could not imagine how drastically changed logic after we take into account that "finite in theory means infinite in practice"

If teoretician says: "This is possible in principle" practitioner must understand: "This is practically impossible"

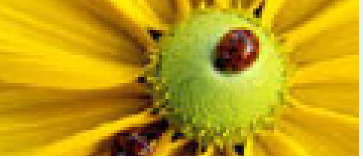


Restricted constructions

Yessenin-Volpin could not imagine how drastically changed logic after we take into account that “finite in theory means infinite in practice”

> If teoretician says: “This is possible in principle” practitioner must understand: “This is practically impossible”

1983: Nilpotent logic of restricted time (N. Nepejvoda)



Restricted constructions

Yessenin-Volpin could not imagine how drastically changed logic after we take into account that "finite in theory means infinite in practice"

> If teoretician says: "This is possible in principle" practitioner must understand: "This is practically impossible"

1983: Nilpotent logic of restricted time (N. Nepejvoda)
1988: Linear logic of restricted money (J.-Y. Girard)

Restricted constructions

Yessenin-Volpin could not imagine how drastically changed logic after we take into account that "finite in theory means infinite in practice"

> If teoretician says: "This is possible in principle" practitioner must understand: "This is practically impossible"

1983: Nilpotent logic of restricted time (N. Nepejvoda)
1988: Linear logic of restricted money (J.-Y. Girard)
2008: Reversive logic of invertible actions (N. Nepejvoda & A. Nepejvoda)

Restricted constructions 2

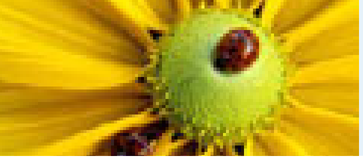
All logics of restricted constructions are very non-classical and mutually inconsistent

Restricted constructions 2

All logics of restricted constructions are very non-classical and mutually inconsistent

Nilpotent (aka automate or flowchart): No constructive conjunctions (no parallelism in automate) $A \Rightarrow A$ is true only if A is always false.

Propositional fragment has simple formalisms and is easily decidable

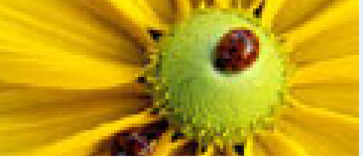


Restricted constructions 2

All logics of restricted constructions are very non-classical and mutually inconsistent

Nilpotent (aka automate or flowchart): No constructive conjunctions (no parallelism in automate) $A \Rightarrow A$ is true only if A is always false. Propositional fragment has simple formalisms and is easily decidable

Linear: all classical, intuitionistic and much more connectives. Propositional fragment is undecidable. No $A \Rightarrow A\&A$



Restricted constructions 2

All logics of restricted constructions are very non-classical and mutually inconsistent

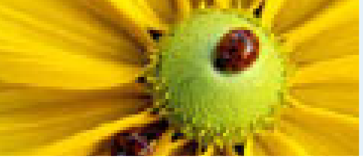
Nilpotent (aka automate or flowchart): No constructive conjunctions (no parallelism in automate) $A \Rightarrow A$ is true only if A is always false. Propositional fragment has simple formalisms and is easily decidable

Linear: all classical, intuitionistic and much more connectives. Propositional fragment is undecidable. No $A \Rightarrow A\&A$

Reversive: no constructive disjunctions.

Paraconsistent. No $A \Rightarrow A\&A$, $A\&A \Rightarrow A$.

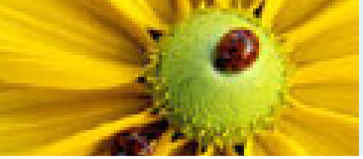
Retractability



Zaslavsky logic

There are only constructive connectives $\Rightarrow \vee \& \sim \forall \exists$. Their semantic is defined through two notions of realizability: positive and negative one. This logic is called intuitionistic symmetric logic.

- $\langle a, b\rangle \circledR^{+} A\&B \equiv a\circledR^{+} A \wedge b\circledR^{+} B$; $\langle i, c\rangle \circledR^{-} A\&B \equiv (i = 1 \wedge c\circledR^{-} A)$ or $(i = 2 \wedge c\circledR^{-} B)$;



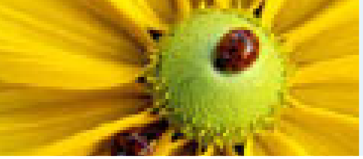
Zaslavsky logic

There are only constructive connectives $\Rightarrow \vee \& \sim \forall \exists$. Their semantic is defined through two notions of realizability: positive and negative one. This logic is called intuitionistic symmetric logic.

- $\langle a, b\rangle \circledR^{+} A \& B \equiv a \circledR^{+} A \wedge b \circledR^{+} B$;
  $\langle i, c\rangle \circledR^{-} A \& B \equiv (i = 1 \wedge c \circledR^{-} A)$ or
  $(i = 2 \wedge c \circledR^{-} B)$;
- $\langle i, c\rangle \circledR^{+} A \& B \equiv (i = 1 \wedge c \circledR^{+} A)$ or
  $(i = 2 \wedge c \circledR^{+} B)$;
  $\langle a, b\rangle \circledR^{-} A \vee B \equiv a \circledR^{-} A \wedge b \circledR^{-} B$;

Zaslavsky logic 2

- $\langle f, g\rangle \circledR^{+} A \Rightarrow B \equiv \forall a\,(a \circledR^{+} A \supset !(a\ f) \wedge (a\ f) \circledR^{+} B) \wedge \forall b\,(b \circledR^{-} B \supset !(b\ g) \wedge (b\ g) \circledR^{-} A)$;
  $\langle a, b\rangle \circledR^{-} A \Rightarrow B \equiv a \circledR^{+} A \wedge b \circledR^{-} B$;

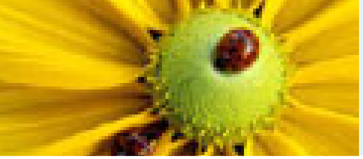


Zaslavsky logic 2

- $\langle f, g\rangle \circledR^{+} A \Rightarrow B \equiv \forall a\,(a \circledR^{+} A \supset !(a\ f) \wedge (a\ f) \circledR^{+} B) \wedge \forall b\,(b \circledR^{-} B \supset !(b\ g) \wedge (b\ g) \circledR^{-} A);$
  $\langle a, b\rangle \circledR^{-} A \Rightarrow B \equiv a \circledR^{+} A \wedge b \circledR^{-} B;$
- $a \circledR^{+} \sim A \equiv a \circledR^{-} A;$
  $a \circledR^{-} \sim A \equiv a \circledR^{+} A;$

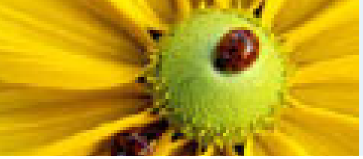
Zaslavsky logic 2

- $f \circledR^{+} \forall x\, A(x) \equiv$ for all a $(a \in U \supset !(a\ f) \wedge (a\ f) \circledR^{+} A(a))$; $\langle u, a\rangle \circledR^{-} \forall x\, A(x) \equiv$ exists u $(u \in U \wedge a \circledR^{-} A(u))$;



Zaslavsky logic 2

- $f \circledR^{+} \forall x\, A(x) \equiv$ for all a $(a \in U \supset !(a\ f) \wedge (a\ f)\circledR^{+} A(a))$;
  $\langle u, a\rangle \circledR^{-} \forall x\, A(x) \equiv$ exists u $(u \in U \wedge a \circledR^{-} A(u))$;
- $\langle u, a\rangle \circledR^{+} \exists x\, A(x) \equiv$ exists u $(u \in U \wedge a \circledR^{+} A(u))$;
  $f \circledR^{-} \exists x\, A(x) \equiv$ for all a $(a \in U \supset !(a\ f) \wedge (a\ f)\circledR^{-} A(a))$;

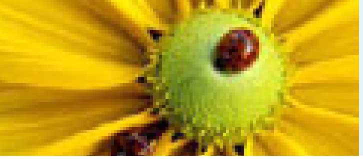


Sample applied theory

Let the following theory fragment describes some packages in functional language

$$
\begin{array}{ll}
\forall x\,((A(x) \Rightarrow N(x)), & \varphi\,Ⓡ\,\forall y\,(N(y) \Rightarrow\sim \exists x\, M(\\
\multicolumn{2}{l}{g\,Ⓡ\,\forall x(C(x) \Rightarrow L(x) \vee E(x) \vee M(x)),} \\
\forall x\,(L(x) \Rightarrow D(x)), & \forall x\,(H(x) \Rightarrow T(x, (x\ f)))
\end{array}
$$

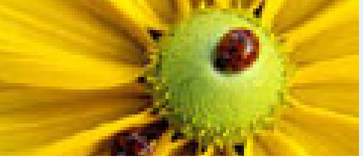
which is a part of a constructive theory describing some packages of programs



Our goal

Let we proved a formula

$$\begin{aligned}&\forall x\,(A(x)\ \&\\ &\quad(\forall x\,(C(x)\Rightarrow D(x)\vee E(x))\Rightarrow \exists y\,H(y))\\ &\Rightarrow \exists z\,T(y,z))\end{aligned}$$

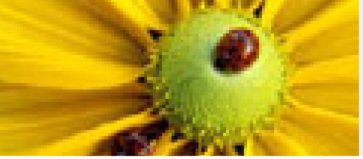


Our goal

Let we proved a formula

$$\begin{aligned}&\forall x\,(A(x)\ \&\\ &\quad(\forall x\,(C(x) \Rightarrow D(x) \vee E(x)) \Rightarrow \exists y\, H(y))\\ &\Rightarrow \exists z\, T(y,z))\end{aligned}$$

Proof consists of two parts: forward (computation) and backwards (analysis)



Forward proof

$* \ A(z), \ \forall x \, (C(x) \Rightarrow D(x) \vee E(x)) \Rightarrow \exists y \, H(y),$
z is arbitrary

> $N(z)$
>
> $\sim \exists x \, M(x)$
>
> $* \ C(u), \ u$ is arbitrary
>
> > $L(u) \vee E(u) \vee M(u)$
> >
> > $\sim M(u)$
> >
> > $* \ L(u) \qquad * \ E(u)$
> >
> > > $D(u)$
>
> $\forall x \, (C(x) \Rightarrow D(x) \vee E(x))$
>
> $H(c_1)$
>
> $T(z, (c_1 \ f))$

Backward proof

$*\ \sim T(y,z),\ y, z$ are arbitrary

$\sim A(x) \vee \sim (\forall x\,(C(x) \Rightarrow D(x) \vee E(x)) \Rightarrow \exists y\, H($

$*\ \sim (\forall x\,(C(x) \Rightarrow D(x) \vee E(x)) \Rightarrow \exists y\, H(y))$

$\sim H(x),\ x$ is arbitrary

$\exists x\,(C(x)\ \&\!\sim D(x)\ \&\!\sim E(x))$

$L(c_2) \vee E(c_2) \vee M(c_2)$

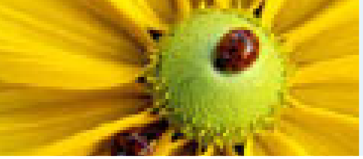
$\sim L(c_2) \qquad \sim E(c_2) \qquad *\ \sim A(y)$

$M(c_2)$

$\sim N(y)$

$\sim A(y)$

$\sim A(y)$



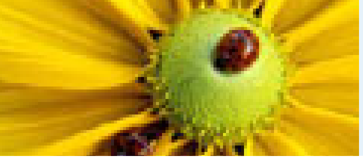
Program and analysis

Here our direct program is

$$\Phi : \ \textbf{func}\ (\textbf{obj}, \textbf{func}(\textbf{func}(\textbf{obj})\textbf{void} \oplus \textbf{void})\ \textbf{obj})\ \textbf{obj}$$

$$\lambda x, \Psi. ((\lambda x. \textbf{case}\ (x\ \ g)$$

$$\textbf{in}\ 1 : 1, 2 : 2, 3 : \ \textbf{error esac}\ \ \Psi)\ \ f)$$



Program and analysis

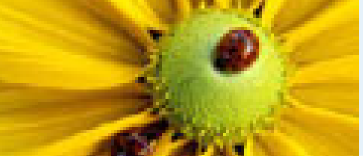
Here our direct program is

$$\Phi : \ \textbf{func}\ (\textbf{obj}, \textbf{func}(\textbf{func}(\textbf{obj})\textbf{void} \oplus \textbf{void})\ \textbf{obj})\ \textbf{obj}$$

$$\lambda x, \Psi. \left((\lambda x.\, \textbf{case}\ (x\ \ g) \right.$$

$$\left. \textbf{in}\ 1:1, 2:2, 3:\ \ \textbf{error esac}\ \ \Psi)\ \ f\right)$$

If its result is wrong, an error is in A. The reason of this trouble is probably a wrong value of x which formally does not enter into a resulting program.



Ghosts

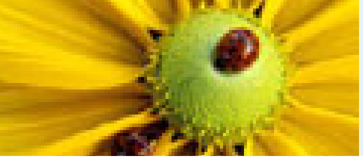
Moreover here we have an interesting duality. G. S. Tseytin pointed out in 1970 that program values are not sufficient to analyze a program. Program is surrounded by *ghosts* which are necessary to understand and to transform a program but are at least useless during its computation. During retraction ghosts become computable entities while values of direct program become ghosts.

Slabs

There is a dual notion: a *slab*. This is what is not needed logically but is inserted from some side reasons: lack of constructions in PL, 'effectivity' and so on. For example (x,y):=(y,x+y) we are forced to express like

z:=x; x:=y; y:=x+z;

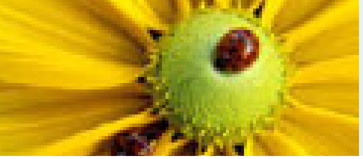
Reversibility



A semigroup

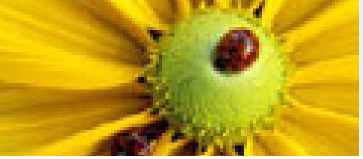
An algebraic definition of reversibility

Let X be an enumerated set. Let $\mathfrak{C}(X,X)$ be a set of all total computable functions $f : X \to X$. A semigroup $R \subset \mathfrak{C}(X,X)$ having a neutral element $e = \lambda x.x$ and having a right inverse f^{-1} for each f (i.e. such f^{-1} that $f \circ f^{-1} = e$) is called *reversible computability* upon set of objects X.



Shortcoming to incoming

Because reversibility has no connection to Landauer limit we don’t need to assure undoing down to atomic actions in reversible computing because reversibility is needed only for external reasons (say many legal and business program must be able to reconstruct the state of the system for any previous time moment). Hence *a reversible program can use modules written in irreversible manner if we grant undoing of their results*.



Shortcoming to incoming 2

From this point we can see strategic mistakes made in the design of reversible language Janus. For example, there is a brilliant invention of Janus authors that each unary function f is extended up to its reversible extension

$$(x\ y\ g) = \langle x * (y\ f), y\rangle$$

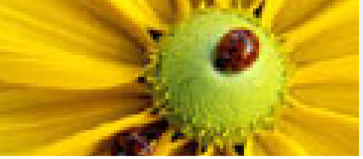
where $\forall x, y, z\,(x * z = y * z \supset x = y)$. They showed that each unary function can be extended in such manner.

Shortcoming to incoming 3

This excellent shot had a wrong goal and is missed. Of course it is too much for reversibility but too less for reversivity (it grants only undoing).

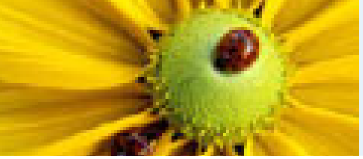
Shortcoming to incoming 3

This excellent shot had a wrong goal and is missed. Of course it is too much for reversibility but too less for reversivity (it grants only undoing). But excellent ideas are always useful though not always where they had been proposed. A. Nepejvoda yesterday stated connections of r.e. with simple proofs.



Challenging claim

There is no need of reversible programming language. All needed can be formulated as clear and easily checked automatically discipline of programming in traditional language.



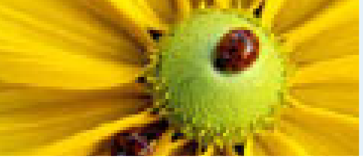
Reversivity

Constructive reversive logic (CRL)

For a mathematical semantic we consider an arbitrary group G. One more important step was proposed and successfully developed by J.-Y. Girard in his linear logic (using commutative monoid to represent money-spending actions). For our case it sounds as follows:

States are the same group as actions.

Thus G is called both *the group of actions* and *the group of states*.



Language of CRL

CRL is a propositional logic. The primitives of reversive logic language are propositional symbols A, B, $C\ldots$, five connectives of classical logic ($\supset$, $\equiv$, $\wedge$, $\vee$, $\neg$) called here *descriptive connectives*, four constructive logical connectives $\Rightarrow, \&, \sim, E$. E is null-ary, $\neg$ and $\sim$ are unary, all others are binary.

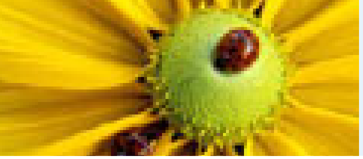
Classical and constructive connectives are fully interoperable and can be mixed arbitrarily. This is not the case in other constructive logics of restricted constructions.

Informal semantic of CRL

Let *signature* Σ be a nonempty set of propositional symbols.
Classical connectives are read and understood in standard way. $\Rightarrow$ reads "can be transformed", $A\&B$ reads "*sequential conjunction*" or "A then B"[1], $\sim A$ is a preventive negation which can be read in different contexts as "undo A" or "prevent A".

---

[1] Of course we can read this "and" in the sense of famous Kleene's examples: "Mary married and born a child", "Mary born a child and married".



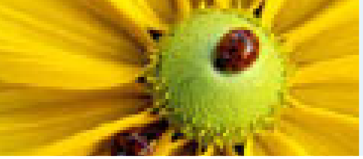
Formal semantic of CRL

Realization of a formula in the interpretation I. The set of realizations for A is denoted $\circledR A$.

1. $a \circledR A \triangleq a \in \zeta(A)$ where A is propositional letter and $A \in \Sigma$.

2. Classical connectives are standard. E.g. $a \circledR (A \wedge B) \triangleq a \circledR A$ and $a \circledR B$.

3. $a \circledR (A \Rightarrow B) \triangleq \forall b \in G \; (b \circledR A \supset b \circ a \circledR B)$. Thus a transforms solutions of A into solutions of B.

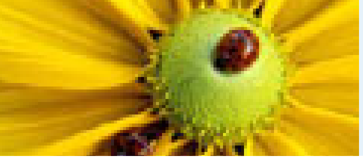


Formal semantic of CRL 2

4 $a \circ b \circledR (A \,\&\, B) \triangleq a \circledR A \wedge b \circledR B$. A solution of B is applied to a solution of A.

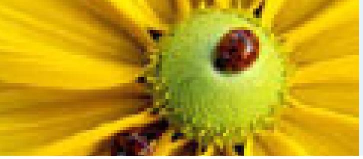
5 $a \circledR \sim A \triangleq a^{-1} \circledR A$. a undoes a solution of A or prevents it.

6 $a \circledR E \triangleq a = e$.



CRL and programming

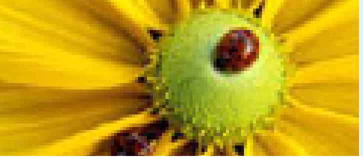
Here we have no constructive disjunction. If introduced it demands an ≪interleaving product≫ of groups: a group of all products $a_1 \circ b_1 \circ \cdots \circ a_n \circ b_n$ where a_i are from realizations of A and b_i are from one of B. This destroys finiteness and means that conditionals demand increasing memory. Analyzing constructions of Fredkin and Toffoli we see that it is.



CRL and programming 2

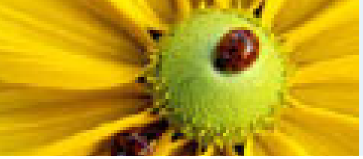
Brief history 1
Brief history 2
Brief history 3
Brief history 4
Constructivism as a tool for CS and Informatics
Retractability
Reversibility
Reversivity
Constructive reversive logic (CRL)
Language of CRL
Informal semantic of CRL
Formal semantic of CRL
Formal semantic of CRL 2
CRL and programming
CRL and programming 2
Gains of group semantics
Sketch: Botik language 1
Sketch: Botik language 2
Sketch: Botik language 3

So pure reversive programming language is to be without conditionals and loops but from the very beginning functional one. In practice we are to use irreversive operations (at least initializing and result writing) and very restricted use of conditionals and loops. Of course there are no recursions and reversive language is not Turing-complete. Atomic computing elements for reversive computer are to be group-valued not binary.



Gains of group semantics

Composition of group elements $a \circ b$ can be understood by any of three ways:



Gains of group semantics

Brief history 1
Brief history 2
Brief history 3
Brief history 4
Constructivism as a tool for CS and Informatics
Retractability
Reversibility
Reversivity
Constructive reversive logic (CRL)
Language of CRL
Informal semantic of CRL
Formal semantic of CRL
Formal semantic of CRL 2
CRL and programming
CRL and programming 2
Gains of group semantics
Sketch: Botik language 1
Sketch: Botik language 2
Sketch: Botik language 3

Composition of group elements $a \circ b$ can be understood by any of three ways:

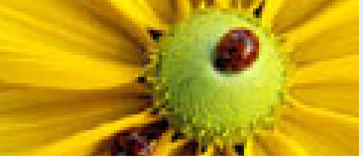
1. We perform the state-transfoming action a then the action b;

Gains of group semantics

Composition of group elements $a \circ b$ can be understood by any of three ways:

1. We perform the state-transfoming action a then the action b;

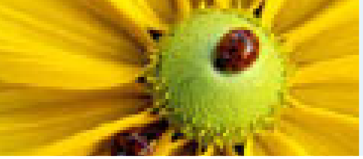
2. We apply the function b to a;



Gains of group semantics

Composition of group elements $a \circ b$ can be understood by any of three ways:

1. We perform the state-transfoming action a then the action b;
2. We apply the function b to a;
3. We construct a composition of functions a and b.

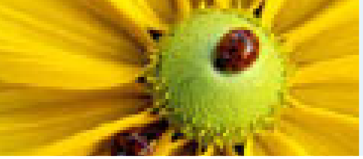


Gains of group semantics

Composition of group elements $a \circ b$ can be understood by any of three ways:

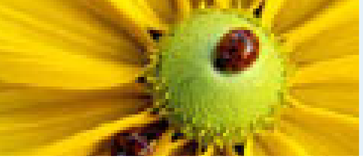
1. We perform the state-transfoming action a then the action b;
2. We apply the function b to a;
3. We construct a composition of functions a and b.

All those interpretations are compatible and fully interoperable. This is the main peculiarity of group as a space of elements and actions.



Sketch: Botik language 1

Program consists of header, definitions section, input section, program body and output section.
Heading is:
PROGRAM ⟨Program_name⟩ Output section is
OUTPUT
write ⟨variable list⟩
END OUTPUT



Sketch: Botik language 2

Definitions section begins by a string `DEFINITIONS`, and ends by `END DEFINITIONS`.

Here all names and all explicit subgroups are defined Subgroup definition has one of two forms

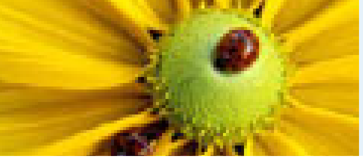
GROUP STANDARD # Only one group and it is defined externally

All atoms except boolean are from this group

Several data types:

GROUP g1,g2: EXTERNAL, ck: [0..k], tn: TRANSPOSITION[n]

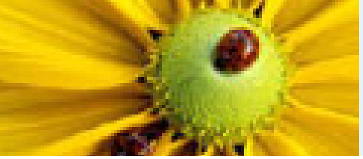
In modeling admissible elementary types are cyclic groups, permutation groups and direct products of Boolean.



Sketch: Botik language 3

Semidirect product construction is a central here. $D \rtimes P$ is defined through a homomorphism $\varphi : P \to \operatorname{Aut} D$ with the following operation:

$$\langle d_1, p_1\rangle \circ \langle d_2, p_2\rangle = \langle d_1 \circ (d_2\ (p_2\ \varphi)), p_1 \circ p_2\rangle$$



Sketch: Botik language 3a

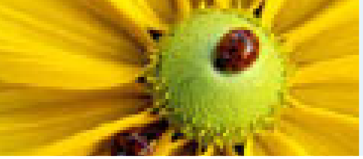
Brief history 1
Brief history 2
Brief history 3
Brief history 4
Constructivism as a tool for CS and Informatics
Retractability
Reversibility
Reversivity
Constructive reversive logic (CRL)
Language of CRL
Informal semantic of CRL
Formal semantic of CRL
Formal semantic of CRL 2
CRL and programming
CRL and programming 2
Gains of group semantics
Sketch: Botik language 1
Sketch: Botik language 2
Sketch: Botik language 3

A semdirect product usually is given implicitly by a list of some variables of the same type: (a,b,c). It means that to compute new values of those variables could be used other from the same list but each only once on each step. Here is an example:

```
var c=(a,b);
. . .
{c;(b,E);(E,-a)}
```

Sketch: Botik language 4

Atoms can be variables, plain atoms and constants. Variables can be changed during execution. Initial values of variables and simple atoms are given in input section. Constants get values in definitions section. There is one constant of any type: E.
One cyclic variables can be declared as guarded. When it becomes 0, program is ended.



Sketch: Botik language 5

Arrays have a cyclic index, for example
    [pn] **array** [i] fib1, fib2
Here [pn] is a type of elements, numder of elements is defined by type of i. Thus array has an associated index variable.
Predicates are only unary and only on a cyclic group:
predicate [ck] pr
Here is a function:
function f1={**if** p1 **then** -a2; a3; a1
    else a4; -a1 **fi**; a1}

Sketch: Botik language 6

Input section

INPUT

...

END INPUT

Here all values of variables and predicates are to be given by read <list_of_names> or directly. Only in this section a value can be copied many times.

Sketch: Botik language 7

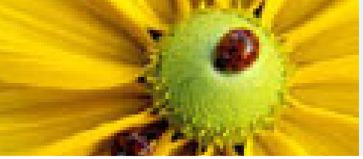
Program body is a sequence of segments. Segments are separated by ; or by , . Comma means that these segments are independent. Weak segment is

$$\left\{ \mathrm{v} \left\langle \begin{array}{c} \text{possible sequence of operators} \\ \text{of the same type,} \\ \text{divided by semicolons} \end{array} \right\rangle \right\}$$

Segment can be preceded by $-$ (inversion).

Sketch: Botik language 8

Segment is a weak segment without $-$. Its first element is whether a variable or a conditional with both alternatives are segments. This variable is the basic. All other elements are understood as operators changing the basic. After a segment there can be $-$, inverting action for basic variable.



Sketch: Botik language 8

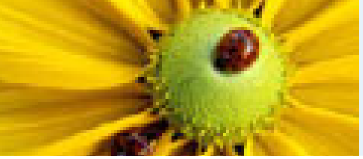
Types of segments

to N **do** t **od**

Loop segment

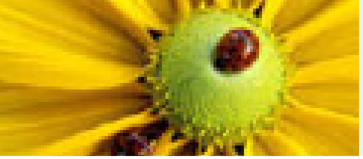
if P **then** t **else** r **fi**

Conditional segment (P is boolean, t, r are weak segments of the same type).



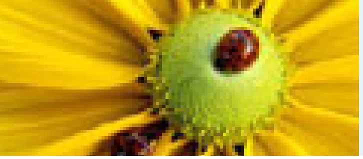
Classes of segments

Classes of segments
No loops and conditionals: pure.
No loops: conditional;
no conditionals: looping;
no conditionals inside loops and loops inside conditionals: safe;
otherwise: dangerous.



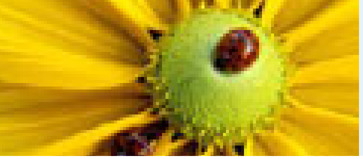
example program 1a

```
PROGRAM Action_directe
DEFINITIONS # All names used in a program
      are specified here
group standard
atom var c
atom a1, a2, a3, a4, a5, a6, a7
predicate p1, p2
function f={a1; if p1
|>then -a2; a3; a1
   else a4; -a1 fi}
function g={a1; to 51 do -a1 od}
function h={a1; a3; -a1}
END DEFINITIONS
```



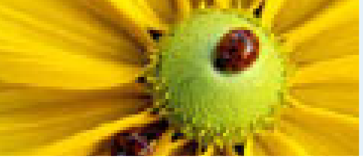
example program 1b

```
INPUT
   # initial values of all atoms and
   # predicates are given here;
   # usually they are computed
   # by external program
   # and transferred into
read c, a1, a2, a3, a4, a5, a6, a7
p1=¬(a4,a6)
# if the domain of a predicate
   # or the value of an atom
   # is fixed for all executions
   # it can be defined inside
   ...
END INPUT
```



example program 1c

```
{c;
-{to 14 do
   -g; h; a7;
   od; a2};
   # we take an inverse
#of the whole program block
if p2 then -f; h else f fi
f; -g; -a4; h;}-
# Direct action leads
# to opposite
# results than desired :)
OUTPUT # a substructure transferred
   #to external processor
   # is defined here
   write c
END OUTPUT
```



Problem with conditionals

if P **then** t **else** r **fi** Let G is a basic group of program commands, H is a group for alternatives. Then to compute this conditional we need a group $\mathbb{Z}_2 \times G \times G \times H$ with an operation

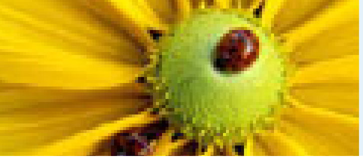
$$\langle z, a_1, b_1, c_1 \rangle \circ \langle 0, a_2, b_2, c_2 \rangle = \langle z, a_1 \circ a_2, b_1 \circ b_2, c_1 \circ$$

$$\langle z, a_1, b_1, c_1 \rangle \circ \langle 1, a_2, b_2, c_2 \rangle = \langle z \oplus 1, a_1 \circ b_2, b_1 \circ a_2, c_1 \circ c_2 \rangle \tag{2}$$

This can be described also as $(G \times G) \rtimes (\mathbb{Z}_2 \times H)$.

Some estimations

They hold during program translation!



Some estimations

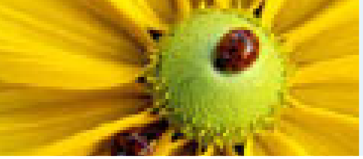
They hold during program translation!

1. pure programs do not change a group;

Some estimations

They hold during program translation!

1. pure programs do not change a group;
2. each written loop adds an additive constant to the number of the group elements;



Some estimations

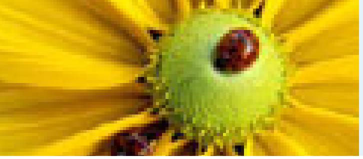
They hold during program translation!

1. pure programs do not change a group;
2. each written loop adds an additive constant to the number of the group elements;
3. each executed conditional (roughly speaking) doubles the number of elements in a group.

Some estimations

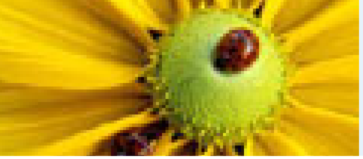
They hold during program translation!

1. pure programs do not change a group;
2. each written loop adds an additive constant to the number of the group elements;
3. each executed conditional (roughly speaking) doubles the number of elements in a group.



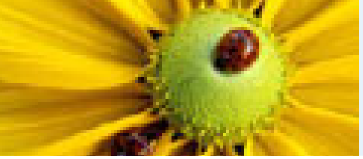
More sophisticated example

Let we try to apply the same actin very many times. This corresponds in group to compute $a \circ b^{\omega}$. Then we represent ω in Fibonacci system. This can be easily made by usual computer. Let number of bits in representation is k. Then we define and transfer to reversive program two predicates: `(i fib_odd)`, `(i fib_even)`. First one is 1 iff i is odd and the corresponding digit is equal to 1. `(i fib_even)` is the same for even indices.



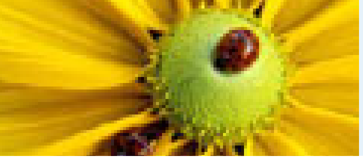
Large loop program

```
PROGRAM Fibonacci_power
DEFINITIONS
int atom n
GROUP tn: TRANSPOSITION[n]
tp atom var a,b,d
tp atom e
(tp,tp) var c is (a,b)
constant e=E
int atom k
int atom var i [0..k] guarded
boolean atom l; predicate [i] fib_odd, fib_even
END DEFINITIONS
```



Large loop program

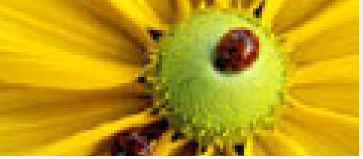
```
INPUT read a, k
b← a
i ← 1
l ← TRUE
d ← E
read fib_odd, fib_even
END INPUT
```



Large loop program

Brief history 1
Brief history 2
Brief history 3
Brief history 4
Constructivism as a tool for CS and Informatics
Retractability
Reversibility
Reversivity
Constructive reversive logic (CRL)
Language of CRL
Informal semantic of CRL
Formal semantic of CRL
Formal semantic of CRL 2
CRL and programming
CRL and programming 2
Gains of group semantics
Sketch: Botik language 1
Sketch: Botik language 2
Sketch: Botik language 3

```
to k do
    {c; if l then (e,a) else (b,e) fi};
    {d; if (i fib_odd) then
        a else if (i fib_odd) then b else e
    fi fi};
{i;1},
{l; true}
od
OUTPUT
write d
END OUTPUT
```

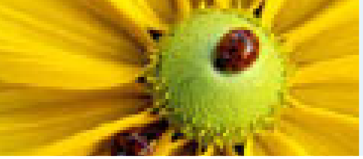


Large loop program

This program looks on the first glance hopelessly dangerous but transforming algebraic structures we really can get an effective algorithm to execute it do not losing its good properties.

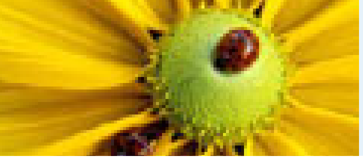
$\heartsuit$ $\smile$

Summary



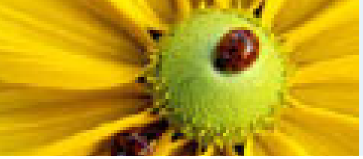
Thanks!

There are three substantially different but usually mixed notions of inverse computability. They need different tools and use different logics.



Thanks!

There are three substantially different but usually mixed notions of inverse computability. They need different tools and use different logics.
A reversive computation demands full invertibility of actions. Only it can grant minimization of heat pollution.

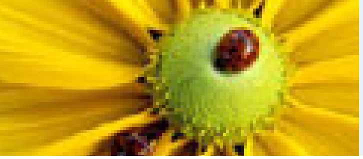


Thanks!

There are three substantially different but usually mixed notions of inverse computability. They need different tools and use different logics.

A reversive computation demands full invertibility of actions. Only it can grant minimization of heat pollution.

Reversive computability is not Turing-complete and a reversive processor can work only as specialized unit of an usual (for example von Neumann) computer.

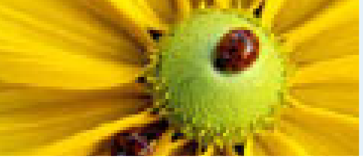


Thanks!

There are three substantially different but usually mixed notions of inverse computability. They need different tools and use different logics.

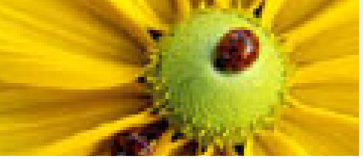
A reversive computation demands full invertibility of actions. Only it can grant minimization of heat pollution.

Reversive computability is not Turing-complete and a reversive processor can work only as specialized unit of an usual (for example von Neumann) computer.



Thanks!

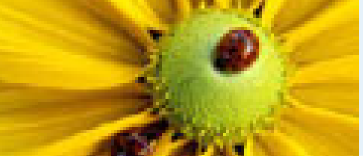
It is necessary to compute in a reversive program the algebraic structures of data types and of the whole data space before program compilation because each modification of programs changes all data structures in it. This algebraic computation can be somewhat sophisticated.



Thanks!

It is necessary to compute in a reversive program the algebraic structures of data types and of the whole data space before program compilation because each modification of programs changes all data structures in it. This algebraic computation can be somewhat sophisticated.

A reversible computing (unrestricted undoing) can be implemented in traditional computers by traditional programming languages as a discipline of programming.



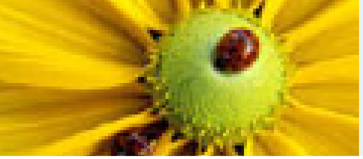
Thanks!

Brief history 1
Brief history 2
Brief history 3
Brief history 4
Constructivism as a tool for CS and Informatics
Retractability
Reversibility
Reversivity
Summary
Thanks!
Thanks!
Publications
Publications

It is necessary to compute in a reversive program the algebraic structures of data types and of the whole data space before program compilation because each modification of programs changes all data structures in it. This algebraic computation can be somewhat sophisticated.

A reversible computing (unrestricted undoing) can be implemented in traditional computers by traditional programming languages as a discipline of programming.

A program retraction (computation of precondition which hold or fail for the given result) can be made by means of almost traditional logic. During retraction values and ghosts are interchanged.



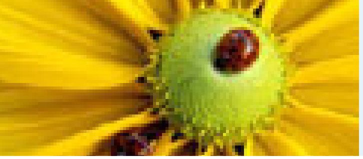
Publications

Непейвода Н.Н.: Уроки конструктивизма. Geidelberg: Lambert Academic Publishing, 98 pp. (2011)

Непейвода Н.Н.: Реверсивные конструктивные логики. Логические исследования, 15, 150–168 (2009)

Непейвода А. Н.: О сюрьективной импликации в реверсивной логике. VI Смирновские чтения по логике (2009)

Непейвода А. Н. Элементы реверсивных вычислений Управление большими системами труды VI всероссийской школы-семинара молодых ученых, Ижевск (2009)



Publications

Непейвода А. Н.: О реверсивной альтернативе традиционным вычислениям. Трехмерная визуализация научной, технической и социальной реальности. Технологии высокополигонального моделирования : труды Второй междунар. конф., Ижевск (2010).

Непейвода А. Н.: Функциональное программирование над группой. Системный анализ и семиотическое моделирование: труды первой всероссийской конференции, 2011, Казань (2011)