Strongly Normalising Cyclic Data Computation by Iteration Categories of Second-Order Algebraic Theories

Makoto Hamana

Department of Computer Science,

Gunma University, Japan

FSCD'16

Porto University, Portuguese.

Computing with Cyclic Datatypes

- Cyclic data structures in functional programming
- In Haskell, one can define a cyclic list by

```
clist = 2:1:clist
```

- Why is this technique correct? — lazy evaluation

```
head clist  ⇝  2
```

- Not completely safe
- E.g. what is the sum of all elements of `clist`?

```
sum clist  ⇝  non-termination
```

- The "encoding" does not give complete understanding of cyclic datatypes

This Work

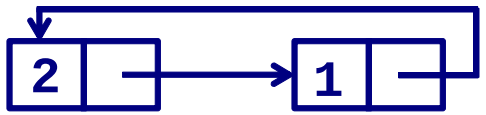
- Complete axiomatisation using
  - second-order algebraic theory [Fiore et al.'10]
  - iteration category [Bloom-Esik'93]
- Do not rely on lazy evaluation
- Strongly normalising fold combinator
- A framework for syntax and semantics of cyclic datatypes

Our Framework

```
ctype CList where
  [] : CList
  :: : CNat,CList  ⟶  CList
with axioms AxCy
```

```
ctype CNat where
  0 : CNat
  S : CNat ⟶ CNat
with axioms AxCy
```

cy(x.2::1::x)



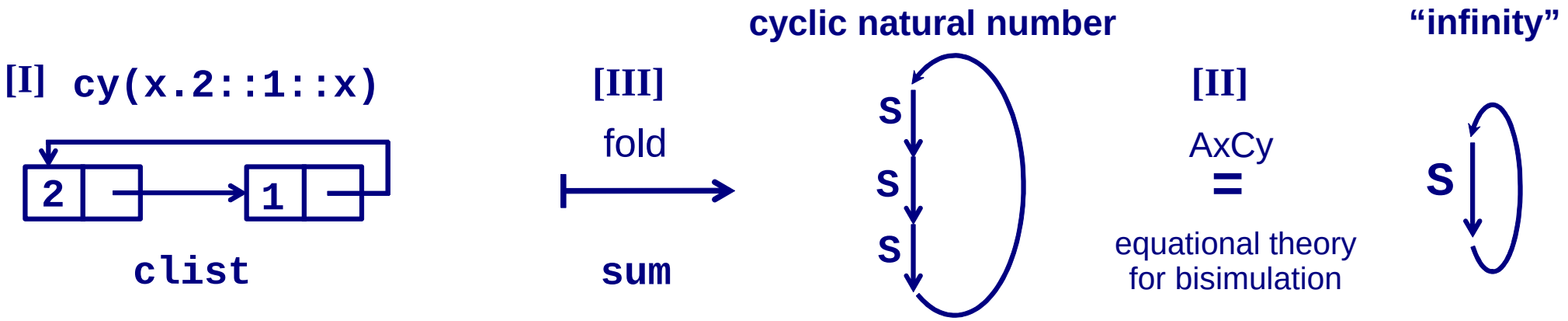
clist

Specification

```
sum : CList ⟶ CNat
spec sum ([])    = 0
     sum (k::t) = plus(k, sum (t))

fun sum t = fold (0, k,x.plus(k,x)) t
```

Example




```
fun sum t = fold (0, k,x.plus(k,x)) t
```

$\texttt{sum(cy(x.S}^2\texttt{(0)::S(0)::x))} \longrightarrow^+ \texttt{cy(x.sum(x.S}^2\texttt{(0)::S(0)::x)@x)}$
$\longrightarrow^+ \texttt{cy(x.S(S(S(x))))}$

- ▷ How to understand?
- ▷ Second-order equational logic and axioms AxCy to equate cyclic data
- ▷ $\Leftrightarrow$ decidable bisimulation

Cartesian Second-Order Algebraic Theory

- Base types $a, b, c, \ldots$
- Signature Σ

$$f : (\overrightarrow{a_1} \to \overrightarrow{b_1}), \ldots, (\overrightarrow{a_m} \to \overrightarrow{b_m}) \to c_1, \ldots, c_n.$$

- Metavariables $(\mathrm{T}, \mathrm{S}, \ldots)$ $\cdots$ variables of at most first-order

$$\mathrm{T} : \overrightarrow{a} \to b$$

- Variables $\cdots$ variables of order 0
- Syntax of meta-terms (Aczel [1978], Klop's CRS [1980])

$$t ::= x \mid x.t \mid f(t_1, \ldots, t_n) \mid \mathrm{T}[t_1, \ldots, t_n]$$

Cyclic Datatypes

Default constructors

Tuple	$\langle -, \cdots, - \rangle$	:	$(\overrightarrow{c_1}), \ldots, (\overrightarrow{c_n}) \to \overrightarrow{c_1}, \ldots, \overrightarrow{c_n}$
Cycle constructor	cy	:	$(\overrightarrow{c} \to \overrightarrow{c}) \to \overrightarrow{c}$
Composition	$\diamond$	:	$(\overrightarrow{a} \to \overrightarrow{c}), \overrightarrow{a} \to \overrightarrow{c}$

E.g.

```
cy(x.S(0)::x))
```

Axioms AxCy for Cycles

(sub) $\quad (\vec{y}.\mathrm{T}[\vec{y}]) \diamond \langle \mathrm{S}_1, \ldots, \mathrm{S}_n \rangle = \mathrm{T}[\mathrm{S}_1, \ldots, \mathrm{S}_n]$

(SP) $\quad \langle (\vec{y}.y_1) \diamond \mathrm{T}, \ldots, (\vec{y}.y_n) \diamond \mathrm{T} \rangle = \mathrm{T}$

(dinat) $\quad \mathrm{cy}(\vec{x}.\mathrm{S}[\mathrm{T}[\vec{x}]]) = (\vec{z}.\mathrm{S}[\vec{z}]) \diamond \mathrm{cy}(\vec{z}.\mathrm{T}[\mathrm{S}[\vec{z}]])$

(Bekič) $\quad \mathrm{cy}(\vec{x},\vec{y}.\langle \widehat{\mathrm{T}}, \widehat{\mathrm{S}} \rangle) = \langle \mathrm{cy}(\vec{x}.(\vec{y}.\widehat{\mathrm{T}}) \diamond \mathrm{cy}(\vec{y}.\widehat{\mathrm{S}})), \mathrm{cy}(\vec{y}.(\vec{x}.\widehat{\mathrm{S}}) \diamond \mathrm{cy}(\vec{x}.(\vec{y}.\widehat{\mathrm{T}}) \diamond \mathrm{cy}(\vec{y}.\widehat{\mathrm{S}}))) \rangle$

(CI) $\quad \mathrm{cy}(\vec{y}.\langle \mathrm{T}[\rho_1], \ldots, \mathrm{T}[\rho_m] \rangle) = \langle \mathrm{cy}(y.\tilde{\mathrm{T}}), \ldots, \mathrm{cy}(y.\tilde{\mathrm{T}}) \rangle$

Note

▷ "cy" is a (Conway) parameterised fixed point operator

(fix) $\quad \mathrm{cy}(\vec{x}.\mathrm{S}[\vec{x}]) = (\vec{z}.\mathrm{S}[\vec{z}]) \diamond \mathrm{cy}(\vec{z}.\mathrm{S}[\vec{z}])$

Example (1): Cyclic Lists modulo Bisimulation

▷ `ctype CList where`

`[] : CList`

`:: : CNat,CList → CList`

`with axioms` AxCy

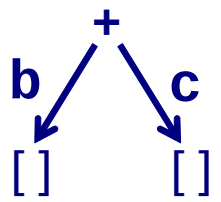
▷ AxCy captures the intended meaning of cyclic lists

$$\mathsf{cy}(x.2 :: x) = \quad 2 :: \mathsf{cy}(x.2 :: x) \quad = \quad 2 :: 2 :: \mathsf{cy}(x.2 :: x)$$

Example (2): Cyclic Sharing Trees modulo Bisimulation

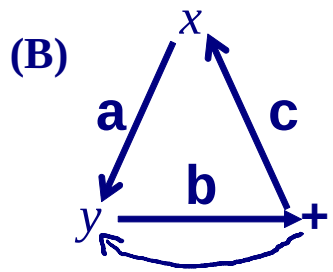
```
ctype CTree where
   f : CTree ⟶ CTree      (f =a,b,c,p,q,...)
   [] : CTree
   + : CTree,CTree ⟶ CTree
with axioms AxCy,AxBr
```

(A)



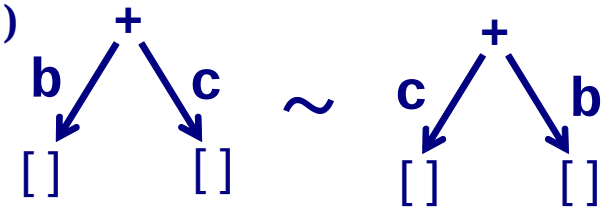
b([])+c([])

(B)



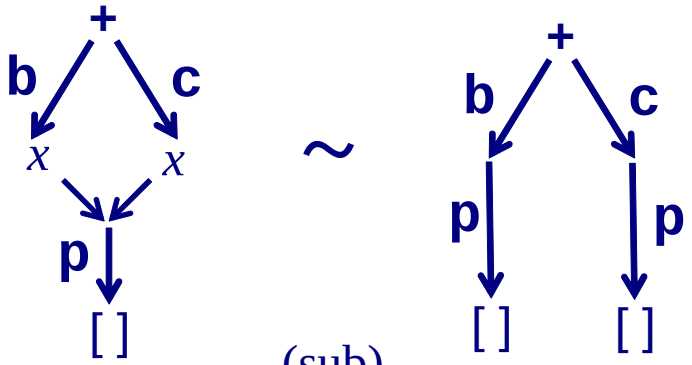
cy(x.a(cy(y.b(c(x+y)))))

(C)



b([])+c([]) = c([])+b([])

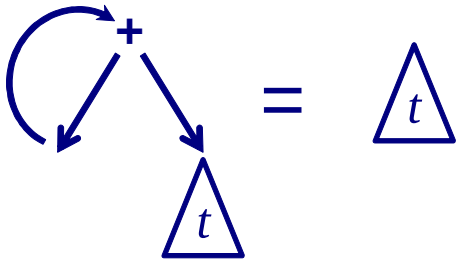
(F)



(x.b(x)+c(x)) o p([]) =(sub) b(p([]))+c(p([]))

Axioms AxBr for branching

(del)	$\mathrm{cy}(x^c.\mathrm{T} + x)$	$=$	T
(unitL)	$[] + \mathrm{T}$	$=$	T
(unitR)	$\mathrm{T} + []$	$=$	T
(assoc)	$(\mathrm{S} + \mathrm{T}) + \mathrm{U}$	$=$	$\mathrm{S} + (\mathrm{T} + \mathrm{U})$
(comm)	$\mathrm{S} + \mathrm{T}$	$=$	$\mathrm{T} + \mathrm{S}$
(degen)	$\mathrm{T} + \mathrm{T}$	$=$	T



axiom (del)

Prop. (Completeness for bisimulation) [H. FICS'15]

$\Gamma \vdash s = t \ : \mathtt{CTree}$ is derivable from AxCy and AxBr

. iff

if s and t are bisimilar.

Categorical Semantics of Cyclic Datatypes

- Why categorical semantics? — to obtain fold combinator
- **Categorical structure**

  Iteration category

  = traced cartesian category [Joyal et al.'96][Hasegawa-Hyland'97]

  + Bloom-Ésik's Commutative Identities axiom

- **Reference**

Bloom, Ésik. *Iteration Theories*, EATCS Monographs. 1993.

Axioms AxCy

(sub) $\quad (\vec{\boldsymbol{y}}.\mathrm{T}[\vec{\boldsymbol{y}}]) \diamond \langle \mathrm{S}_1, \ldots, \mathrm{S}_n \rangle = \mathrm{T}[\mathrm{S}_1, \ldots, \mathrm{S}_n]$

(SP) $\quad \langle (\vec{\boldsymbol{y}}.\boldsymbol{y}_1) \diamond \mathrm{T}, \ldots, (\vec{\boldsymbol{y}}.\boldsymbol{y}_n) \diamond \mathrm{T} \rangle = \mathrm{T}$

(dinat) $\quad \mathsf{cy}(\vec{\boldsymbol{x}}.\mathrm{S}[\mathrm{T}[\vec{\boldsymbol{x}}]]) = (\vec{\boldsymbol{z}}.\mathrm{S}[\vec{\boldsymbol{z}}]) \diamond \mathsf{cy}(\vec{\boldsymbol{z}}.\mathrm{T}[\mathrm{S}[\vec{\boldsymbol{z}}]])]$

(Bekič) $\quad \mathsf{cy}(\vec{\boldsymbol{x}}, \vec{\boldsymbol{y}}. \langle \widehat{\mathrm{T}}, \widehat{\mathrm{S}} \rangle) = \langle \mathsf{cy}(\vec{\boldsymbol{x}}. (\vec{\boldsymbol{y}}.\widehat{\mathrm{T}}) \diamond \mathsf{cy}(\vec{\boldsymbol{y}}.\widehat{\mathrm{S}})),$
$\mathsf{cy}(\vec{\boldsymbol{y}}. (\vec{\boldsymbol{x}}.\widehat{\mathrm{S}}) \diamond \mathsf{cy}(\vec{\boldsymbol{x}}. (\vec{\boldsymbol{y}}.\widehat{\mathrm{T}}) \diamond \mathsf{cy}(\vec{\boldsymbol{y}}.\widehat{\mathrm{S}}))) \rangle$

(CI) $\quad \mathsf{cy}(\vec{\boldsymbol{y}}.\langle \mathrm{T}[\boldsymbol{\rho}_1], \ldots, \mathrm{T}[\boldsymbol{\rho}_m] \rangle) = \langle \mathsf{cy}(\boldsymbol{y}.\tilde{\mathrm{T}}), \ldots, \mathsf{cy}(\boldsymbol{y}.\tilde{\mathrm{T}}) \rangle$

where $\quad \boldsymbol{\rho}_i$ is a permutation of $\vec{\boldsymbol{y}}$, $\quad \tilde{\mathrm{T}} = \mathrm{T}[\boldsymbol{y}, \ldots, \boldsymbol{y}]$

Categorical Semantics and Completeness

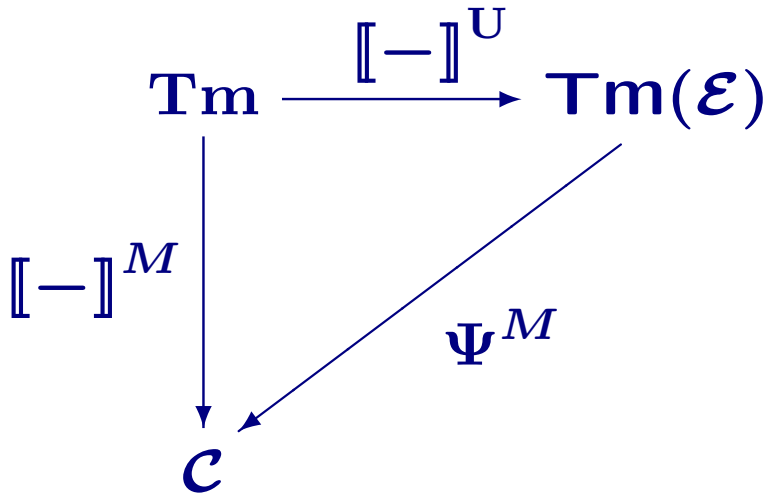
Thm. $\Gamma \vdash s = t \ : \overrightarrow{c}$ is derivable from $\mathcal{E}$ iff $[\![s]\!]^M_{\mathcal{C}} = [\![t]\!]^M_{\mathcal{C}}$ holds for all iteration categories $\mathcal{C}$ and all $(\Sigma, \mathcal{E})$-structures in $\mathcal{C}$.

Category $\mathbf{Tm}(\mathcal{E})$

- ▷ object: sequence of types $b_1, \ldots, b_n$
- ▷ arrow: equivalence class $[\Gamma \vdash t \ : \overrightarrow{c}]_{\mathcal{E}}$ of judgments modulo axioms $\mathcal{E} = \mathrm{AxCy} \cup \mathrm{AxBr}$

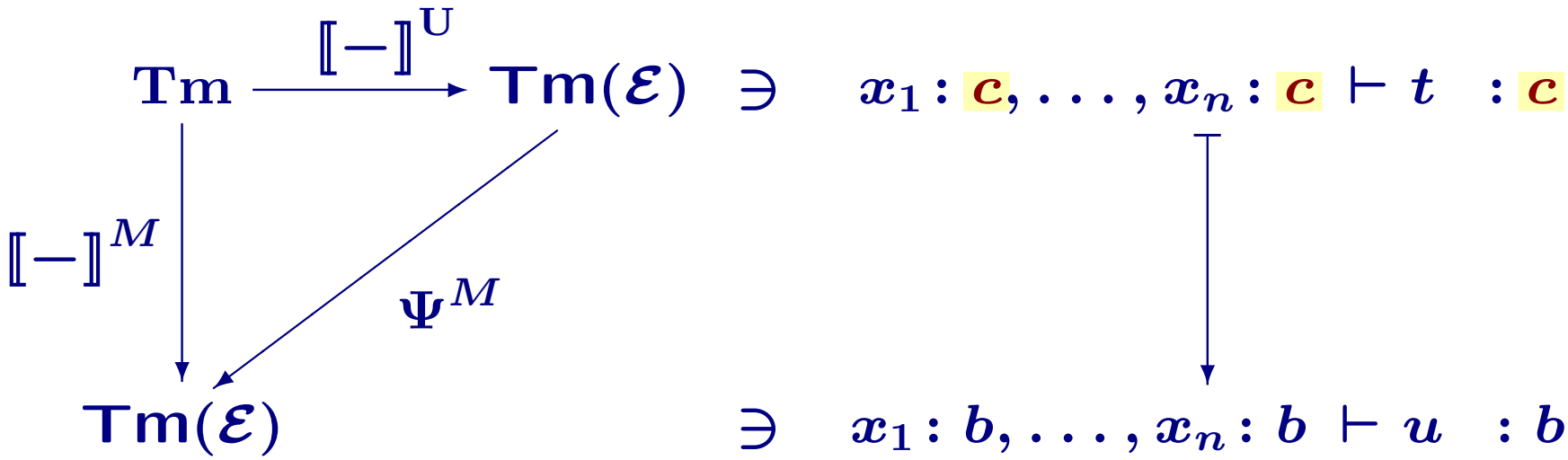
Universality

Thm. For a $(\Sigma, \mathcal{E})$-structure M in an iteration category $\mathcal{C}$, there exists a unique iteration functor $\Psi^M : \mathbf{Tm}(\mathcal{E}) \longrightarrow \mathcal{C}$ that preserves $(\Sigma, \mathcal{E})$-structures.



Extract "Fold" from Universality

Thm. For a $(\Sigma, \mathcal{E})$-structure M in an iteration category $\mathcal{C}$, there exists a unique iteration functor $\Psi^M : \mathbf{Tm}(\mathcal{E}) \longrightarrow \mathcal{C}$ that preserves $(\Sigma, \mathcal{E})$-structures.



- ▷ where the $(\Sigma, \mathcal{E})$-structure M interprets a type c as type b
- ▷ Ψ^M gives a translation from the cyclic datatype c to b (e.g. `CList` to `CNat`)
- ▷ Extract the law of fold by formalising Ψ^M

Second-Order Algebraic Theory for fold

$$\mathsf{fold}(\boldsymbol{E},\ \overrightarrow{\boldsymbol{y}^{\boldsymbol{c}}} \Vdash \boldsymbol{y_i}) = \overrightarrow{\boldsymbol{y}^{\boldsymbol{b}}} \Vdash \boldsymbol{y_i} \quad (\text{for } \boldsymbol{y_i} \in \{\overrightarrow{\boldsymbol{y}}\})$$

$$\mathsf{fold}(\boldsymbol{E},\ \overrightarrow{\boldsymbol{y}} \Vdash \langle\,\rangle) = \overrightarrow{\boldsymbol{y}} \Vdash \langle\,\rangle$$

$$\mathsf{fold}(\boldsymbol{E},\ \overrightarrow{\boldsymbol{y}} \Vdash \langle \mathrm{S}[\overrightarrow{\boldsymbol{y}}], \mathrm{T}[\overrightarrow{\boldsymbol{y}}]\rangle) = \overrightarrow{\boldsymbol{y}} \Vdash \langle \mathsf{app}(\mathsf{fold}(\boldsymbol{E},\ \overrightarrow{\boldsymbol{y}} \Vdash \mathrm{S}[\overrightarrow{\boldsymbol{y}}]), \overrightarrow{\boldsymbol{y}}),\ \mathsf{app}(\mathsf{fold}(\boldsymbol{E},\ \overrightarrow{\boldsymbol{y}} \Vdash \mathrm{T}[\overrightarrow{\boldsymbol{y}}]), \overrightarrow{\boldsymbol{y}}$$

$$\mathsf{fold}(\boldsymbol{E},\ \overrightarrow{\boldsymbol{y}} \Vdash \mathsf{cy}(\overrightarrow{\boldsymbol{x}}.\mathrm{T}[\overrightarrow{\boldsymbol{y}}, \overrightarrow{\boldsymbol{x}}])) = \overrightarrow{\boldsymbol{y}} \Vdash \mathsf{cy}(\overrightarrow{\boldsymbol{x}}.\mathsf{app}(\mathsf{fold}(\boldsymbol{E},\ \overrightarrow{\boldsymbol{y}}, \overrightarrow{\boldsymbol{x}} \Vdash \mathrm{T}[\overrightarrow{\boldsymbol{y}}, \overrightarrow{\boldsymbol{x}}]), \overrightarrow{\boldsymbol{y}}, \overrightarrow{\boldsymbol{x}}))$$

$$\mathsf{fold}(\boldsymbol{E},\ \overrightarrow{\boldsymbol{y}} \Vdash \boldsymbol{d}(\overrightarrow{\mathrm{A}}, \mathrm{T}_1[\overrightarrow{\boldsymbol{y}}], \ldots, \mathrm{T}_n[\overrightarrow{\boldsymbol{y}}])) = \overrightarrow{\boldsymbol{y}} \Vdash (\overrightarrow{\boldsymbol{x}}.\mathrm{E}_{\boldsymbol{d}}[\overrightarrow{\mathrm{A}}, \overrightarrow{\boldsymbol{x}}]) \diamond \langle \mathsf{app}(\mathsf{fold}(\boldsymbol{E},\ \overrightarrow{\boldsymbol{y}} \Vdash \mathrm{T}_1[\overrightarrow{\boldsymbol{y}}]), \overrightarrow{\boldsymbol{y}}$$

$$\mathsf{fold}(\boldsymbol{E},\ \boldsymbol{x} \Vdash (\overrightarrow{\boldsymbol{y}}.\mathrm{T}[\overrightarrow{\boldsymbol{y}}]) \diamond \mathrm{S}[\overrightarrow{\boldsymbol{x}}]) = \overrightarrow{\boldsymbol{x}} \Vdash \overrightarrow{\boldsymbol{y}}.\mathsf{app}(\mathsf{fold}(\boldsymbol{E},\ \overrightarrow{\boldsymbol{y}} \Vdash \mathrm{T}[\overrightarrow{\boldsymbol{y}}]), \overrightarrow{\boldsymbol{y}}) \diamond \mathsf{app}(\mathsf{fold}(\boldsymbol{E},\ \overrightarrow{\boldsymbol{x}} \Vdash \mathrm{S}[\overrightarrow{\boldsymbol{x}}]$$

$$\mathsf{app}(\overrightarrow{\boldsymbol{x}} \Vdash \mathrm{S}[\overrightarrow{\boldsymbol{x}}], \boldsymbol{z_1}, \ldots, \boldsymbol{z_m}) = \mathrm{S}[\boldsymbol{z_1}, \ldots, \boldsymbol{z_m}]$$

To obtain computation rules

▷ Include some theorems of AxCy and AxBr for simplification

▷ Orient equations

Second-Order Rewrite System FOLDr

Fold

$$\mathsf{fold}(\boldsymbol{E},\ \overrightarrow{\boldsymbol{y}^{\mathsf{Var}_{\boldsymbol{c}}}}.\ \mathsf{v}(\boldsymbol{y_i})) \to \overrightarrow{\boldsymbol{y}^{\mathsf{Var}_{\boldsymbol{b}}}}.\ \mathsf{v}(\boldsymbol{y_i})$$

$$\mathsf{fold}(\boldsymbol{E},\ \overrightarrow{\boldsymbol{y}}.\ \langle\,\rangle) \to \overrightarrow{\boldsymbol{y}}.\ \langle\,\rangle$$

$$\mathsf{fold}(\boldsymbol{E},\ \overrightarrow{\boldsymbol{y}}.\ \langle \mathrm{S}[\overrightarrow{\boldsymbol{y}}], \mathrm{T}[\overrightarrow{\boldsymbol{y}}]\rangle) \to \overrightarrow{\boldsymbol{y}}.\ \langle \mathsf{fold}(\boldsymbol{E},\ \overrightarrow{\boldsymbol{y}}.\ \mathrm{S}[\overrightarrow{\boldsymbol{y}}])@\overrightarrow{\boldsymbol{y}},\ \mathsf{fold}(\boldsymbol{E},\ \overrightarrow{\boldsymbol{y}}.\ \mathrm{T}[\overrightarrow{\boldsymbol{y}}])@\overrightarrow{\boldsymbol{y}}\rangle$$

$$\mathsf{fold}(\boldsymbol{E},\ \overrightarrow{\boldsymbol{y}}.\ \mathsf{cy}^{\boldsymbol{1}}(\boldsymbol{x}.\mathrm{T}[\overrightarrow{\boldsymbol{y}}, \boldsymbol{x}])) \to \overrightarrow{\boldsymbol{y}}.\ \mathsf{cy}^{\boldsymbol{1}}(\boldsymbol{x}.\mathsf{fold}(\boldsymbol{E},\ \overrightarrow{\boldsymbol{y}}, \boldsymbol{x}.\ \mathrm{T}[\overrightarrow{\boldsymbol{y}}, \boldsymbol{x}])@\overrightarrow{\boldsymbol{y}}, \boldsymbol{x})$$

$$\mathsf{fold}(\boldsymbol{E},\ \overrightarrow{\boldsymbol{y}}.\ \boldsymbol{d}(\overrightarrow{\mathrm{A}}, \mathrm{T}_{\boldsymbol{1}}[\overrightarrow{\boldsymbol{y}}], \ldots, \mathrm{T}_{\boldsymbol{n}}[\overrightarrow{\boldsymbol{y}}])) \to \overrightarrow{\boldsymbol{y}}.\ (\overrightarrow{\boldsymbol{x}}.\mathrm{E}_{\boldsymbol{d}}[\overrightarrow{\mathrm{A}}, \overrightarrow{\boldsymbol{x}}]) \diamond \langle\, \mathsf{fold}(\boldsymbol{E},\ \overrightarrow{\boldsymbol{y}}.\ \mathrm{T}_{\boldsymbol{1}}[\overrightarrow{\boldsymbol{y}}])@\overrightarrow{\boldsymbol{y}}, \ldots\rangle$$

$$\mathsf{fold}(\boldsymbol{E},\ \overrightarrow{\boldsymbol{x}}.\ (\overrightarrow{\boldsymbol{y}}.\mathrm{T}[\overrightarrow{\boldsymbol{y}}]) \diamond \mathrm{S}[\overrightarrow{\boldsymbol{x}}]) \to \overrightarrow{\boldsymbol{x}}.\ (\overrightarrow{\boldsymbol{y}}.\mathsf{fold}(\boldsymbol{E},\ \overrightarrow{\boldsymbol{y}}.\ \mathrm{T}[\overrightarrow{\boldsymbol{y}}])@\overrightarrow{\boldsymbol{y}})\ \diamond\ \mathsf{fold}(\boldsymbol{E},\ \overrightarrow{\boldsymbol{x}}.\ \mathrm{S}[\overrightarrow{\boldsymbol{x}}])@\overrightarrow{\boldsymbol{x}}$$

Bekič and cycle cleaning

$$\mathsf{cy}^{\boldsymbol{m+n}}(\overrightarrow{\boldsymbol{x}}, \overrightarrow{\boldsymbol{y}}.\ \langle\, \widehat{\mathrm{T}},\ \widehat{\mathrm{S}}\,\rangle) \to \langle\, \mathsf{cy}^{\boldsymbol{m}}(\overrightarrow{\boldsymbol{x}}.\ (\overrightarrow{\boldsymbol{y}}.\widehat{\mathrm{T}}) \diamond \mathsf{cy}^{\boldsymbol{n}}(\overrightarrow{\boldsymbol{y}}.\widehat{\mathrm{S}})),\ \mathsf{cy}^{\boldsymbol{n}}(\overrightarrow{\boldsymbol{y}}.\ (\overrightarrow{\boldsymbol{x}}.\widehat{\mathrm{S}}) \diamond \mathsf{cy}^{\boldsymbol{m}}(\overrightarrow{\boldsymbol{x}}.\ (\overrightarrow{\boldsymbol{y}}.\widehat{\mathrm{T}}) \diamond \mathsf{cy}^{\boldsymbol{n}}(\overrightarrow{\boldsymbol{y}}.\widehat{\mathrm{S}})))\rangle$$

$$\mathsf{cy}(\overrightarrow{\boldsymbol{y}}.\mathrm{T}) \to \mathrm{T}$$

$$\mathsf{cy}(\boldsymbol{x}.\mathsf{v}(\boldsymbol{x})) \to [\,]$$

$$\mathsf{cy}(\boldsymbol{x}.\mathrm{T} + \mathsf{v}(\boldsymbol{x})) \to \mathrm{T}$$

Composition

$$(\overrightarrow{\boldsymbol{y}}.\ \mathsf{v}(\boldsymbol{y_i})) \diamond \langle \overrightarrow{\mathrm{S}} \rangle \to \mathrm{S}_{\boldsymbol{i}}$$

$$(\overrightarrow{\boldsymbol{y}}.\ \boldsymbol{d}(\overrightarrow{\boldsymbol{x_1}}.\ \mathrm{T}_{\boldsymbol{1}}[\overrightarrow{\boldsymbol{y}}, \overrightarrow{\boldsymbol{x_1}}], \ldots)) \diamond \langle \overrightarrow{\mathrm{S}} \rangle \to \boldsymbol{d}(\overrightarrow{\boldsymbol{x}}.\ (\overrightarrow{\boldsymbol{y}}.\ \mathrm{T}_{\boldsymbol{1}}[\overrightarrow{\boldsymbol{y}}, \overrightarrow{\boldsymbol{x_1}}]) \diamond \langle \overrightarrow{\mathrm{S}} \rangle, \ldots, (\overrightarrow{\boldsymbol{y}}.\ \mathrm{T}_{\boldsymbol{n}}[\overrightarrow{\boldsymbol{y}}, \overrightarrow{\boldsymbol{x_n}}]) \diamond \langle \overrightarrow{\mathrm{S}} \rangle \rangle$$

▷ Strongly normalising by [Blanqui RTA'00]'s the General Schema

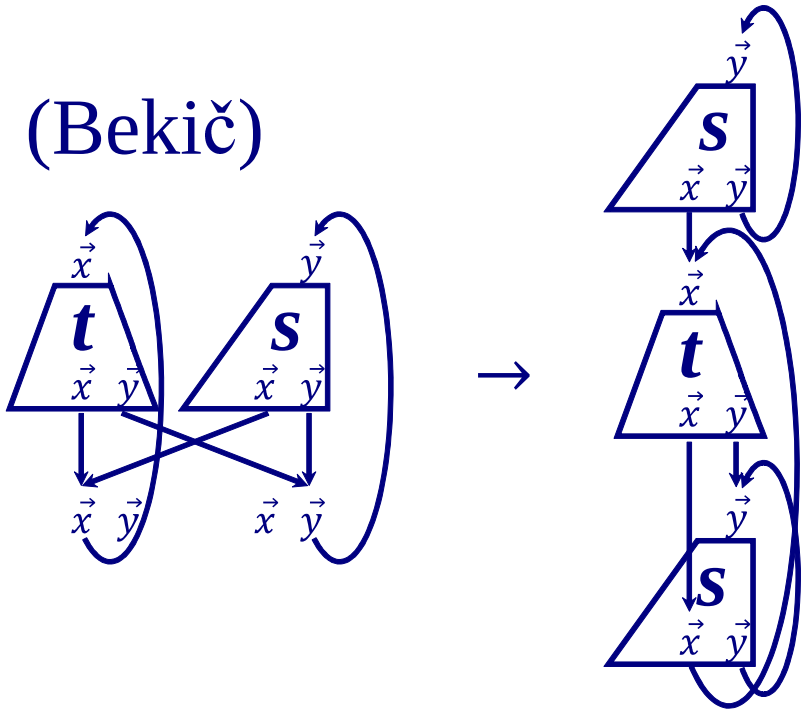
Fixed point law

We do not include

$$\text{(fix)} \quad \mathrm{cy}(\vec{x}.\mathrm{s}[\vec{x}]) \;\longrightarrow\; (\vec{z}.\mathrm{s}[\vec{z}]) \diamond \mathrm{cy}(\vec{z}.\mathrm{s}[\vec{z}])$$

in the rewrite system FOLDr

But include



Example: Tail of Cyclic List


```
ctail : CList ⟶ CList
spec ctail ([])       = []
     ctail (k::t)     = t
     ctail (cy(x. t))= ??
```

Example: Tail of Cyclic List


```
ctail : CList ⟶ CList
spec ctail ([])      = []
     ctail (k::t)    = t
     ctail (cy(x. t))= ??
```

▶ Primitive recursion by fold

. "paramorphism"

```
fun ctail(t) = π1 ◇ fold (<[],[]>, k.x.y.<y, k::y>) t
```

```
ctail(cy(x.1::2::x)) ⟶+ π1 ◇ cy(x.y. <2::y, 1::2::y>)
    ⟶+ π1 ◇ <cy(x.2::cy(y.1::2::y)), cy(y.1::2::y)>
    ⟶  cy(x.2::cy(y.1::2::y))
    ⟶  2::cy(y.1::2::y)        (Normal form)
```

▷ The step $\longrightarrow^+$ uses Bekič law

▷ Fixed point law is not used ▶ Not infinite unfolding

Summary

I. A framework for cyclic datatypes using second-order algebraic theories

II. Semantics: iteration categories

III. Extracted SN fold by rewrite system FOLDr

Further results

- ▷ FOLDr: Confluent on plain terms
- ▷ FOLDr: Confluent modulo bisimulation $\sim$ via Huet [1980]'s theorem using *local coherence modulo* $\sim$
- ▷ A complete model of graph database language UnCAL [Buneman'96] (with Matsuda and Asada) draft available