Lambda terms and maps, formally (II)

Catherine Dubois[1] and Alain Giorgetti[2]

[1] Samovar, ENSIIE, Evry, France
[2] FEMTO-ST, Univ. Bourgogne Franche-Comté, Besançon, France

Computational Logic and Applications
Paris, 24-25 May 2018

Motivations

- Recent bijections between lambda terms and maps
  - Closed linear lambda terms (CLLT) and almost trivalent maps with bivalent root vertex (ATM_2)

    Olivier Bodini, Danièle Gardy, and Alice Jacquot.
    Asymptotics and random sampling for BCI and BCK lambda terms.
    Theoretical Computer Science, 502(0):227 – 238, 2013.
  - Normal planar lambda terms and planar maps

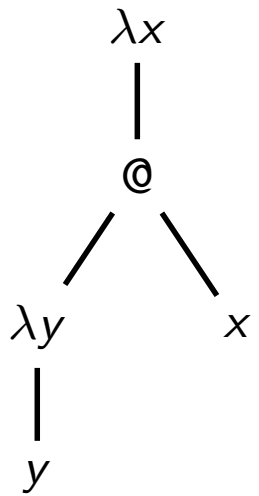
    Noam Zeilberger and Alain Giorgetti.
    A correspondence between rooted planar maps and normal planar lambda terms.
    Logical Methods in Computer Science, 11:1–39, 2015.
- Early formalization
  - Prolog specification
  - Formal proofs with a proof assistant (Coq)
  - Random and bounded-exhaustive testing

Closed linear lambda terms (CLLT)

- λ terms

$$M, N ::= \lambda X.\, M \mid M\ N \mid X$$

- CLLT (or BCI terms): closed λ terms where each variable is used exactly once (linearity)
- Example: $\lambda x.\, (\lambda y.\, y)\ x$
- Size: number $n \geq 0$ of applications
- $n + 1$ variables
- $n + 1$ abstractions

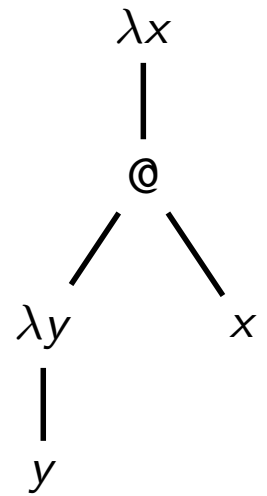


Closed linear lambda terms (CLLT)

- λ terms

$$M, N ::= \lambda X.\, M \mid M\ N \mid X$$

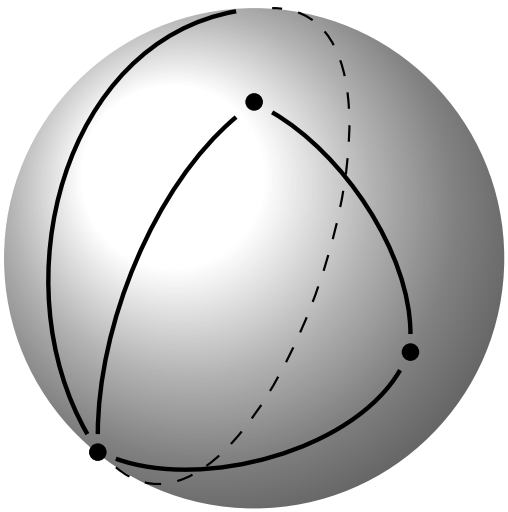
- CLLT (or BCI terms): closed λ terms where each variable is used exactly once (linearity)
- Example: $\lambda x.\, (\lambda y.\, y)\ x$
- Size: number $n \geq 0$ of applications
- $n + 1$ variables
- $n + 1$ abstractions
- Counted by size (modulo α-equivalence) by the OEIS sequence A062980

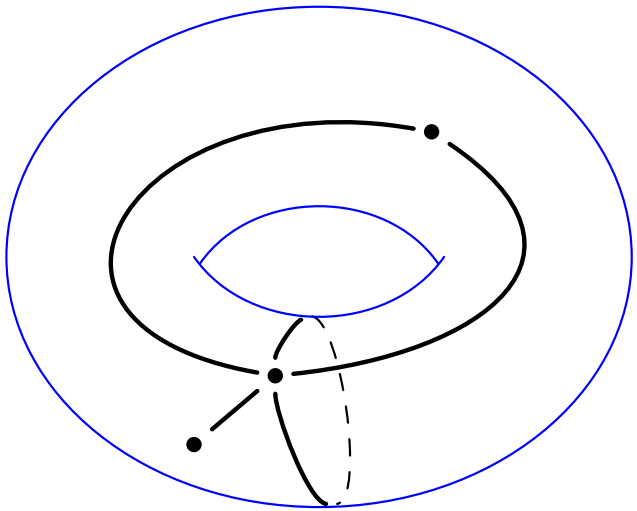


1 5 60 1,105 27,120 828,250 30,220,800 ...

What is a map?

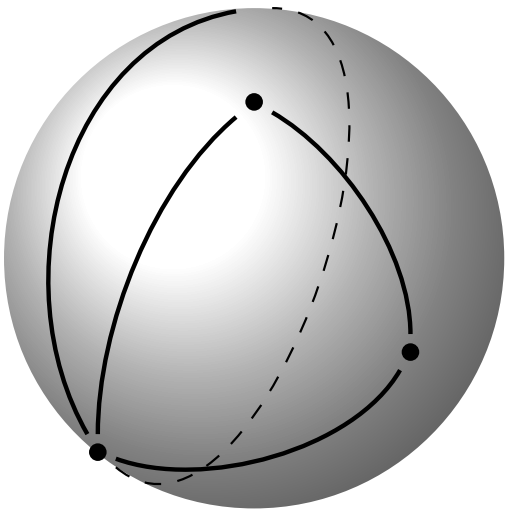
A (topological) map is a connected graph (loops and multiple edges allowed) drawn on a surface

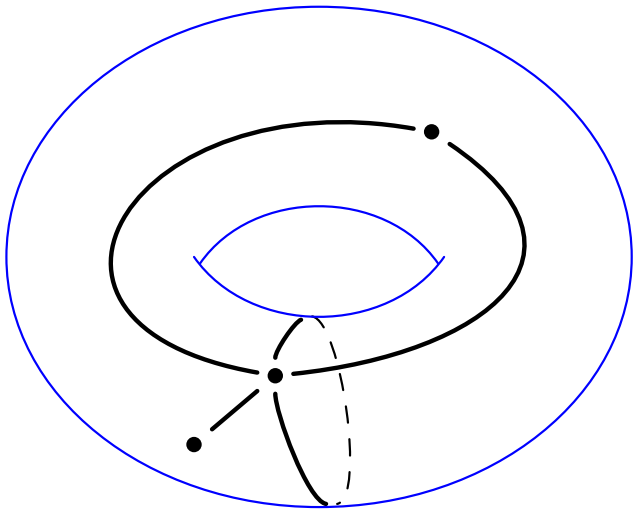




What is a map?

A (topological) map is a connected graph (loops and multiple edges allowed) drawn on a surface

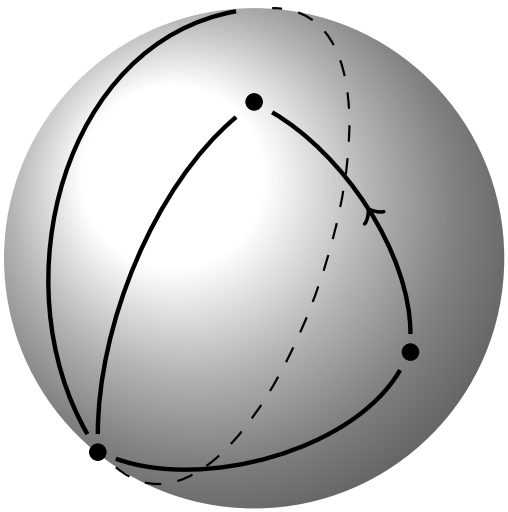


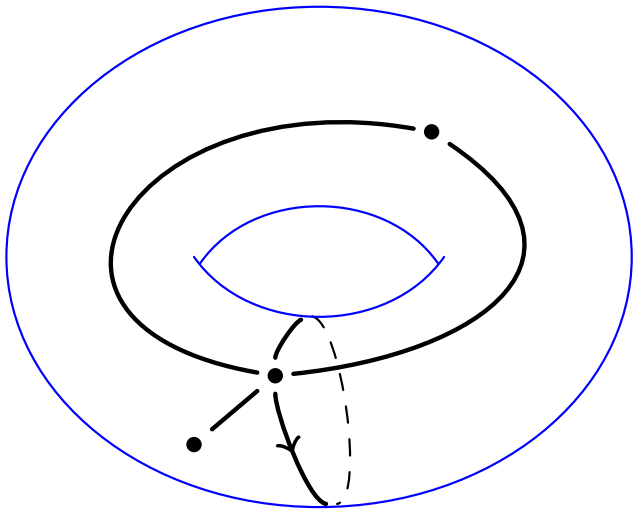


- Studied (generated, counted, etc) up to isomorphism (orientation-preserving surface diffeomorphism & underlying graph isomorphism) $\rightsquigarrow$ Finitely many distinct maps with a given number of edges

What is a map?

A (topological) map is a connected graph (loops and multiple edges allowed) drawn on a surface





- Studied (generated, counted, etc) up to isomorphism (orientation-preserving surface diffeomorphism & underlying graph isomorphism) $\rightsquigarrow$ Finitely many distinct maps with a given number of edges
- Rooted: First *dart* (half-edge) distinguished (root) $\rightsquigarrow$ no inner symmetry

Bijection under formal study

- Almost trivalent map (ATM): non-root vertices have degree 3
- ATM_2: ATM with bivalent root vertex (degree 2)

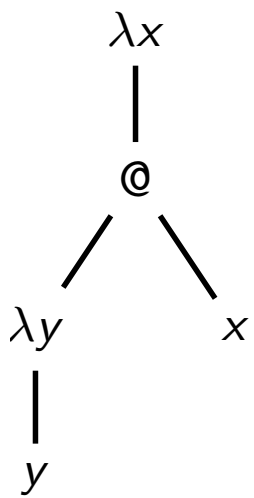
Bijection under formal study

- Almost trivalent map (ATM): non-root vertices have degree 3
- ATM_2: ATM with bivalent root vertex (degree 2)

Theorem [BodiniGardyJacquot13]

Closed linear lambda terms (CLLT) with $n+1$ variables and ATM_2 with $2n+1$ vertices are in bijection.

Example: $\lambda x.\,(\lambda y.\,y)\;x$, $n = 1$



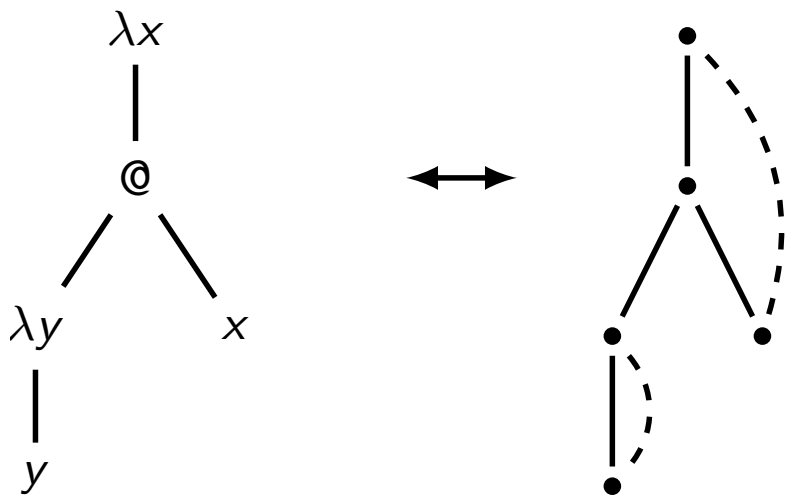
Bijection under formal study

- Almost trivalent map (ATM): non-root vertices have degree 3
- ATM_2: ATM with bivalent root vertex (degree 2)

Theorem [BodiniGardyJacquot13]

Closed linear lambda terms (CLLT) with $n+1$ variables and ATM_2 with $2n+1$ vertices are in bijection.

Example: $\lambda x.\,(\lambda y.\,y)\;x$, $n = 1$



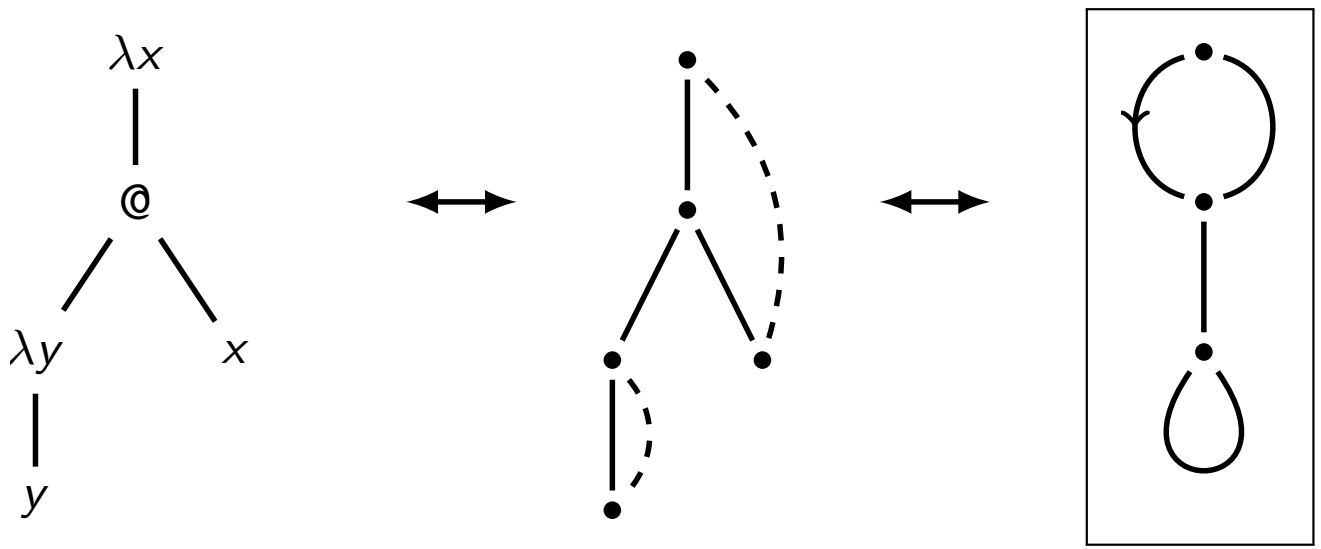
Bijection under formal study

- Almost trivalent map (ATM): non-root vertices have degree 3
- ATM_2: ATM with bivalent root vertex (degree 2)

Theorem [BodiniGardyJacquot13]

Closed linear lambda terms (CLLT) with $n+1$ variables and ATM_2 with $2n+1$ vertices are in bijection.

Example: $\lambda x.\,(\lambda y.\,y)\;x$, $n = 1$



Issues for formalization

How to formalize (almost) (trivalent) maps?

Issues for formalization

How to formalize (almost) (trivalent) maps? $\rightsquigarrow$ see Part 4

Issues for formalization

How to formalize (almost) (trivalent) maps? $\rightsquigarrow$ see Part 4

How to formalize closed linear λ terms?

Issues for formalization

How to formalize (almost) (trivalent) maps? $\rightsquigarrow$ see Part 4

How to formalize closed linear λ terms?

- $M, N ::= \lambda X.\, M \mid M\ N \mid X$ as a Coq inductive type

```
Definition variable := nat.

Inductive lam : Set :=
  Abs : variable → lam → lam
| App : lam → lam → lam
| Var : variable → lam.
```

- $+$ closure property (no free variable)
- $+$ linearity property (no duplicate in the list of used variables)
- $+$ α-equivalence

☹ Complex dependent type (structural properties $+$ quotient)

Issues for formalization

How to formalize (almost) (trivalent) maps? $\rightsquigarrow$ see Part 4

How to formalize closed linear λ terms?

- $M, N ::= \lambda X.\, M \mid M\ N \mid X$ as a Coq inductive type

```
Definition variable := nat.

Inductive lam : Set :=
  Abs : variable → lam → lam
| App : lam → lam → lam
| Var : variable → lam.
```

- $+$ closure property (no free variable)
- $+$ linearity property (no duplicate in the list of used variables)
- $+$ α-equivalence

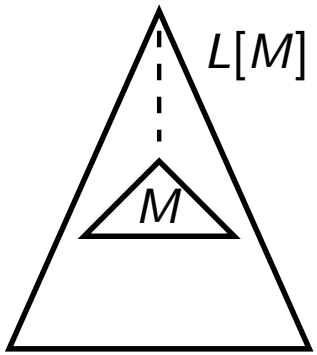
☹ Complex dependent type (structural properties $+$ quotient)

$\rightsquigarrow$ Recursive characterization of CLLT, with simpler constraints?

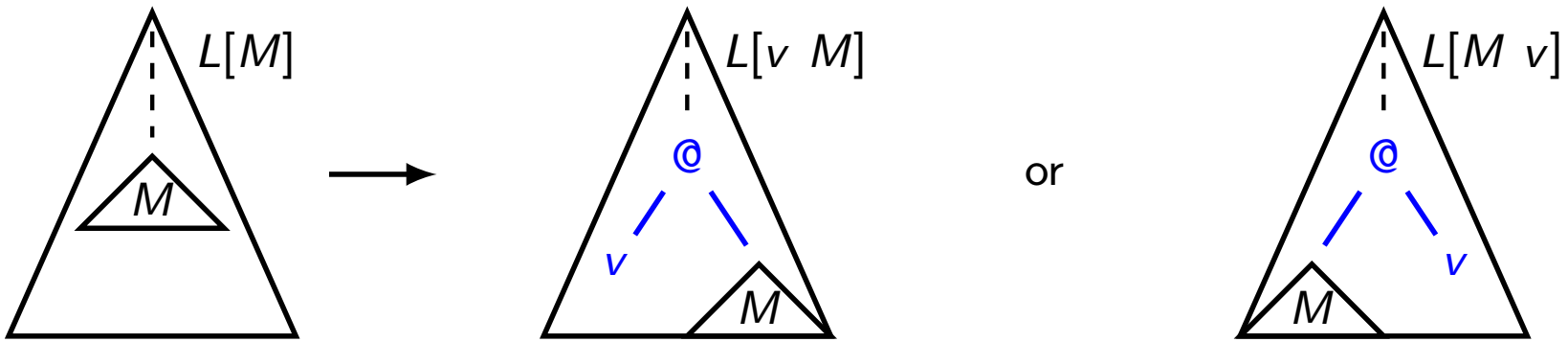
Outline

Outline

Blooming

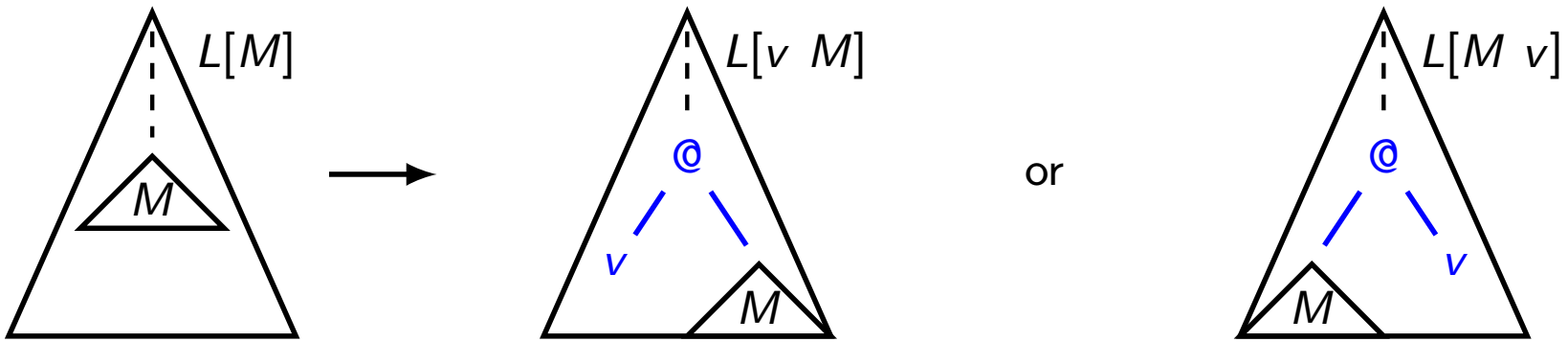


Blooming



Blooming the lambda term L with the variable v is replacing some subterm M of L by $(v\ M)$ or $(M\ v)$

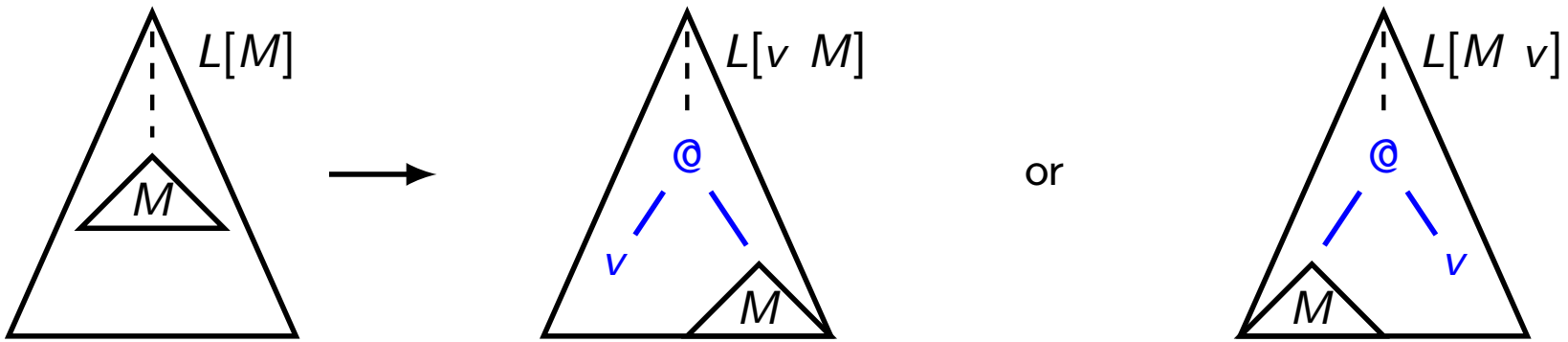
Blooming



Blooming the lambda term L with the variable v is replacing some subterm M of L by $(v\ M)$ or $(M\ v)$

The abstraction of the blooming a closed linear λ-term (CLLT) with a fresh variable is a CLLT

Blooming



Blooming the lambda term L with the variable v is replacing some subterm M of L by $(v\ M)$ or $(M\ v)$

The abstraction of the blooming a closed linear λ-term (CLLT) with a fresh variable is a CLLT

Characterization of CLLT:

$$T, L[M] ::= \lambda X.\, X \mid \lambda X.\, L[X\ M] \mid \lambda X.\, L[M\ X] \mid T\ L$$

(+ distinct variables)

Bloom datatype

$$T, L[M]_i ::= \lambda X.\, X \mid \lambda X.\, L[X\ M]_i \mid \lambda X.\, L[M\ X]_i \mid T\ L$$

Coq type:

```
Inductive bloom: nat → Type :=
| B0: bloom 0
| B1: ∀ m (s: bool) i, i ≤ 3*m+1 → bloom m → bloom (S m)
| B2: ∀ n1 n2, bloom n1 → bloom n2 → bloom (S (n1+n2)).
```

Bloom datatype

$$T, L[M]_i ::= \lambda X.\, X \mid \lambda X.\, L[X\ M]_i \mid \lambda X.\, L[M\ X]_i \mid T\ L$$

Coq type:

```
Inductive bloom: nat → Type :=
| B0: bloom 0
| B1: ∀ m (s: bool) i, i ≤ 3*m+1 → bloom m → bloom (S m)
| B2: ∀ n1 n2, bloom n1 → bloom n2 → bloom (S (n1+n2)).
```

Simpler (sized) dependent type for CLLT (just a linear constraint on a natural number) ☺

Exhaustive generation with Prolog

```
bloom(b0,0).
bloom(b1(S,I,B),N) :- N > 0, M is N-1, bloom(B,M),
 in(S,0,1), Imax is 3*N-2, in(I,0,Imax).
bloom(b2(B1,B2),N) :- N > 0, M is N-1, in(N1,0,M),
 bloom(B1,N1), N2 is M-N1, bloom(B2,N2).
```

`bloom(`B`,`N`)` holds iff B is a bloom with N unary and binary nodes (size)

Exhaustive generation with Prolog

```
bloom(b0,0).
bloom(b1(S,I,B),N) :- N > 0, M is N-1, bloom(B,M),
 in(S,0,1), Imax is 3*N-2, in(I,0,Imax).
bloom(b2(B1,B2),N) :- N > 0, M is N-1, in(N1,0,M),
 bloom(B1,N1), N2 is M-N1, bloom(B2,N2).
```

`bloom(`B`,`N`)` holds iff B is a bloom with N unary and binary nodes (size)

☺ Exhaustive generator for free:

```
?- bloom(B,0).
B = b0 ;
```

```
?- bloom(B,1).
B = b1(1, 1, b0) ;
B = b1(1, 0, b0) ;
B = b1(0, 1, b0) ;
B = b1(0, 0, b0) ;
B = b2(b0, b0) ;
```

```
?- bloom(B,2).
B = b1(1, 4, b1(1, 1, b0)) .
B = b1(1, 3, b1(1, 1, b0)) ;
B = b1(1, 2, b1(1, 1, b0)) ;
B = b1(1, 1, b1(1, 1, b0)) ;
...
B = b2(b1(1, 1, b0), b0) ;
B = b2(b1(1, 0, b0), b0) ;
B = b2(b1(0, 1, b0), b0) ;
...
```

Property checking by counting

```
:- compile(measure).
```

swipl library written by Valerio Senni [_, Senni, GASCom'12]

Property checking by counting

```
:- compile(measure).
```

swipl library written by Valerio Senni [_, Senni, GASCom'12]

- First numbers

  ```
  ?- iterate(0,6,bloom).
  ```

  $\rightsquigarrow$ 1,5,60,1105,27120,828250,30220800 (in less than 1 minute)
  Blooms are counted by the OEIS sequence A062980

Property checking by counting

```
:- compile(measure).
```

swipl library written by Valerio Senni [_, Senni, GASCom'12]

- First numbers

  ```
  ?- iterate(0,6,bloom).
  ```

  ⇝ 1,5,60,1105,27120,828250,30220800 (in less than 1 minute)
  Blooms are counted by the OEIS sequence A062980

- The size of a bloom is its number of unary and binary nodes

  ```
  size(b0,0).
  size(b1(_,_,B),NP1) :- size(B,N), NP1 is N+1.
  size(b2(B1,B2),N) :- size(B1,N1), size(B2,N2), N is N1+N2+1.

  bloom12([B,N],N) :- bloom(B,N), size(B,N).

  ?- iterate(0,6,bloom12).
  ```

  ⇝ same sequence

Property-based testing in Coq: QuickChick

- QuickChick: plugin for Coq for testing conjectures (B. Pierce, C. Hriţcu, L. Lampropoulos, Z. Paraskevopoulou, `https://github.com/QuickChick`, 2014)
  - inspired by QuickCheck for Haskell (2000)
  - similar tools exist for Agda, Isabelle, PVS, FoCaLiZe ...

- `Lemma L: ∀ x: T, precondition x → conclusion x`

  where `precondition` and `conclusion : T → Prop` are logical predicates
- How to test lemma `L` (before proving)?
  1. Define a random generator `gen_T` of values of type `T` that satisfy the property `precondition`
  2. Prove that the random generator is correct ⇝ we test it
  3. Turn `conclusion` into a Boolean function `conclusionb : T → bool`
  4. Run the test on, e.g., 10000 test cases:

     ```
     QuickCheck (forAll gen_T (fun x ⇒ conclusionb x))
     ```

  5. To reduce the trusted base, prove that `conclusion` and `conclusionb` are semantically equivalent

Random generation of blooms with QuickChick

Random generator of values of type `(bloom n)` (follows the inductive definition of the type `bloom`)

```
Program Fixpoint gen_bloom_n (n : nat)
      {measure n}: G (bloom n) :=
 match n with
   0   ⇒ returnGen b0
 | S m ⇒ oneof ...
    [ (bindGen' (choose (0, m)) (fun e1 H ⇒
        liftGen2 (@b2 m e1 H) (gen_bloom_n e1)
                              (gen_bloom_n (m - e1))));
      (bindGen' (choose (0, 3*m+1)) (fun i H ⇒
        bindGen genBool (fun s ⇒
        liftGen (@b1 m s i _) (gen_bloom_n m))))
    ]
 end.
```

Random generator validation

How to validate the random generator?

Random generator validation

How to validate the random generator?

By testing some properties, like the ones checked with Prolog (e.g., the size of a bloom is its number of unary and binary nodes)

```
QuickCheck (sized (fun n ⇒
  forAll (gen_bloom_n n) (fun t ⇒
      beq_nat (size n t) n))).
+++ Passed 10000 tests (0 discards)
```

`sized`: iterates over different values for `n`

Random generator validation

How to validate the random generator?

By testing some properties, like the ones checked with Prolog (e.g., the size of a bloom is its number of unary and binary nodes)

```
QuickCheck (sized (fun n ⇒
  forAll (gen_bloom_n n) (fun t ⇒
      beq_nat (size n t) n))).
+++ Passed 10000 tests (0 discards)
```

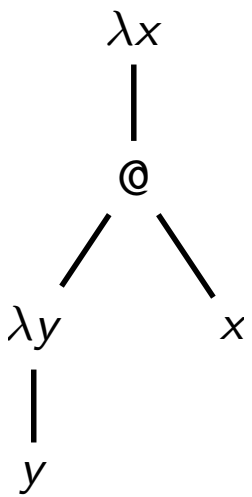
`sized`: iterates over different values for `n`

Properties also proven in Coq (easy, by induction)

```
Lemma bloom_size: ∀ n (t : bloom n), size n t = n.
```

Blooms in the middle

λ terms



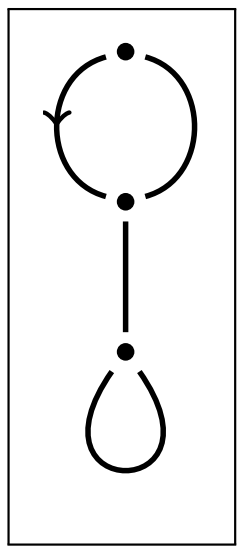
bloomlam ⟵
lambloom ⟶

Blooms

$b_1(?, ?, b_0)$

bloomatm2 ⟶
atm2bloom ⟵

ATM_2



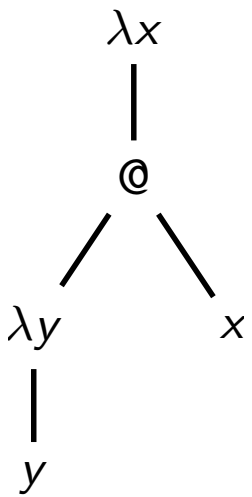
Blooms in the middle, with rooted trivalent maps

Rooted trivalent maps (RTM): all vertices have degree 3

Blooms in the middle, with rooted trivalent maps

Rooted trivalent maps (RTM): all vertices have degree 3

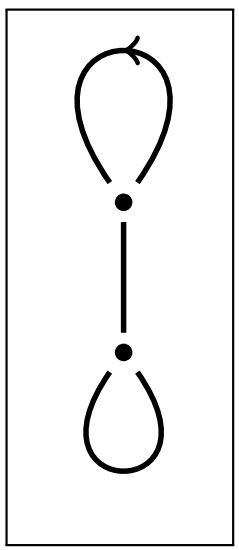
λ terms | Blooms | RTM



bloomlam ⟵ lambloom ⟶

$b_1(1, 0, b_0)$

bloomrtm ⟶ rtmbloom ⟵



Outline

CLLT: Coq definition

```
Inductive lam : Set :=
  Abs : variable → lam → lam
| App : lam → lam → lam
| Var : variable → lam.

Definition is_closed t := (fv t) = [ ].
Definition is_lin t := noDup (uvar t) ∧ nbAbs t = nbVar t
Definition is_cllt t :=
   distinctabs t ∧ is_closed t ∧ is_lin t.
```

A CLLT is a λ-term closed with distinct bound variables such that the list of used variables has no duplicates and such that the number of abstractions is equal to the number of variables

CLLT: Coq definition

```
Inductive lam : Set :=
  Abs : variable → lam → lam
| App : lam → lam → lam
| Var : variable → lam.

Definition is_closed t := (fv t) = [ ].
Definition is_lin t := noDup (uvar t) ∧ nbAbs t = nbVar t
Definition is_cllt t :=
   distinctabs t ∧ is_closed t ∧ is_lin t.
```

A CLLT is a λ-term closed with distinct bound variables such that the list of used variables has no duplicates and such that the number of abstractions is equal to the number of variables

Redundant, but may help for proving and can be checked with Prolog

CLLT: Coq random generator

`(gen_cllt n rk fvar)` generates a λ-term with `n` abstractions named `rk`, `rk`+1, ... and free variables in `fvar`

- may fail (when $\texttt{n} = 0$ and $|\texttt{fvar}| > 0$)
- uses the QuickChick combinator `backtrack`

We test the outputs have distinct bound variables, no duplicates in the used variables, no free variables and as many abstractions as variable terms

```
QuickCheck (sized  (fun n ⇒
    forAll (gen_cllt n 0 []) is_clltb)).
++ Passed 10000 tests (0 discards)
```

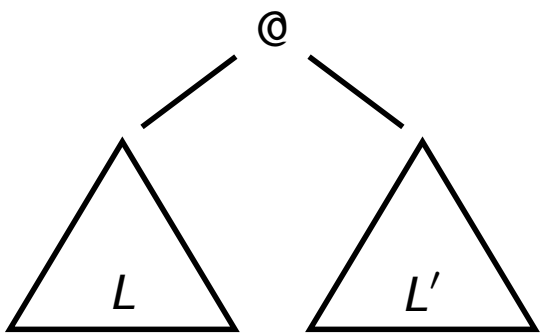
`is_clltb`: Boolean function that verifies its argument is a CLLT (proved as equivalent with the logical predicate `is_cllt`)

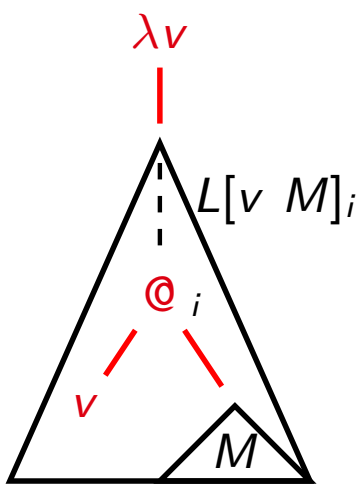
CLLT: Coq theorems

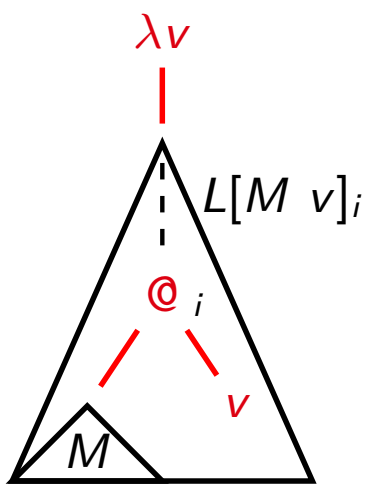
Application preserves closed linear terms (with distinct variables)

```
Lemma app_cllt: ∀ t1 t2,
 is_cllt t1 → is_cllt t2 →
 disjoint (abvar t1) (abvar t2) →
 is_cllt (App t1 t2).
```

From blooms to lambda terms







b_0	$\lambda x.x$
$b_2\ B\ B'$	where $B \rightsquigarrow L$ and $B' \rightsquigarrow L'$
$b_1\ s\ i\ B$	s = false s = true where $B \rightsquigarrow L$ and $L_{\mid i} = M$

From blooms to lambda terms: Prolog

For generation and counting

`bloomlam(B,L,V)` holds iff `L` is the lambda semantics of the bloom `B`, with `V` as first (highest) variable

- Table of correspondence
- Check by counting that the lambda terms `L` are closed and linear

```
bl([B,L],N) :- bloom(B,N), bloomlam(B,L,N), closed(L),
  uv(L,U), nodup(U), eq(L).

:- iterate(0,6,bl).
```

$\rightsquigarrow$ 1,5,60,1105,27120,828250,30220800 (`https://oeis.org/A062980`)

From blooms to lambda terms: Coq

- Definition

```
Fixpoint bloomlam_aux:
    ∀ n : nat, bloom n → variable → option lam :=
  ...

Definition bloomlam n t := bloomlam_aux n t n.
```

From blooms to lambda terms: Coq

- Definition

```
Fixpoint bloomlam_aux:
    ∀ n : nat, bloom n → variable → option lam :=
  ...

Definition bloomlam n t := bloomlam_aux n t n.
```

- Random property-based testing with QuickChick: the λ term obtained from a bloom is a closed linear λ term

```
QuickCheck (sized (fun n ⇒
  forAll (gen_bloom_n n) (fun t ⇒
   boolopt is_clltb (bloomlam n t)))).
+++ Passed 10000 tests (0 discards)
```

From blooms to lambda terms: Coq

- Definition

```
Fixpoint bloomlam_aux:
    ∀ n : nat, bloom n → variable → option lam :=
  ...

Definition bloomlam n t := bloomlam_aux n t n.
```

- Random property-based testing with QuickChick: the λ term obtained from a bloom is a closed linear λ term

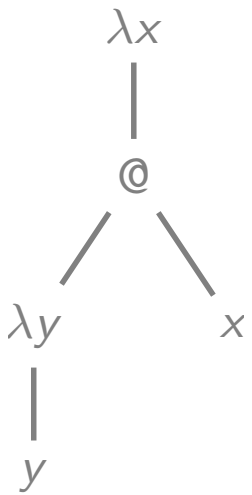
```
QuickCheck (sized (fun n ⇒
  forAll (gen_bloom_n n) (fun t ⇒
   boolopt is_clltb (bloomlam n t)))).
+++ Passed 10000 tests (0 discards)
```

We get a new random CLLT generator by combining the `bloom` generator and the `bloomlam` function

Outline

Map semantics of blooms

λ terms



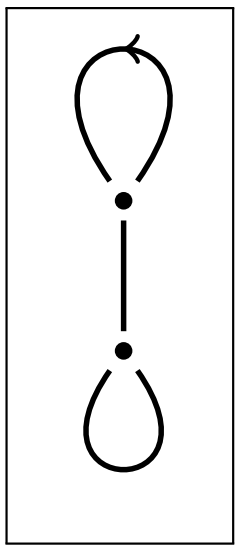
Blooms

bloomlam ⟵
⟶ lambloom

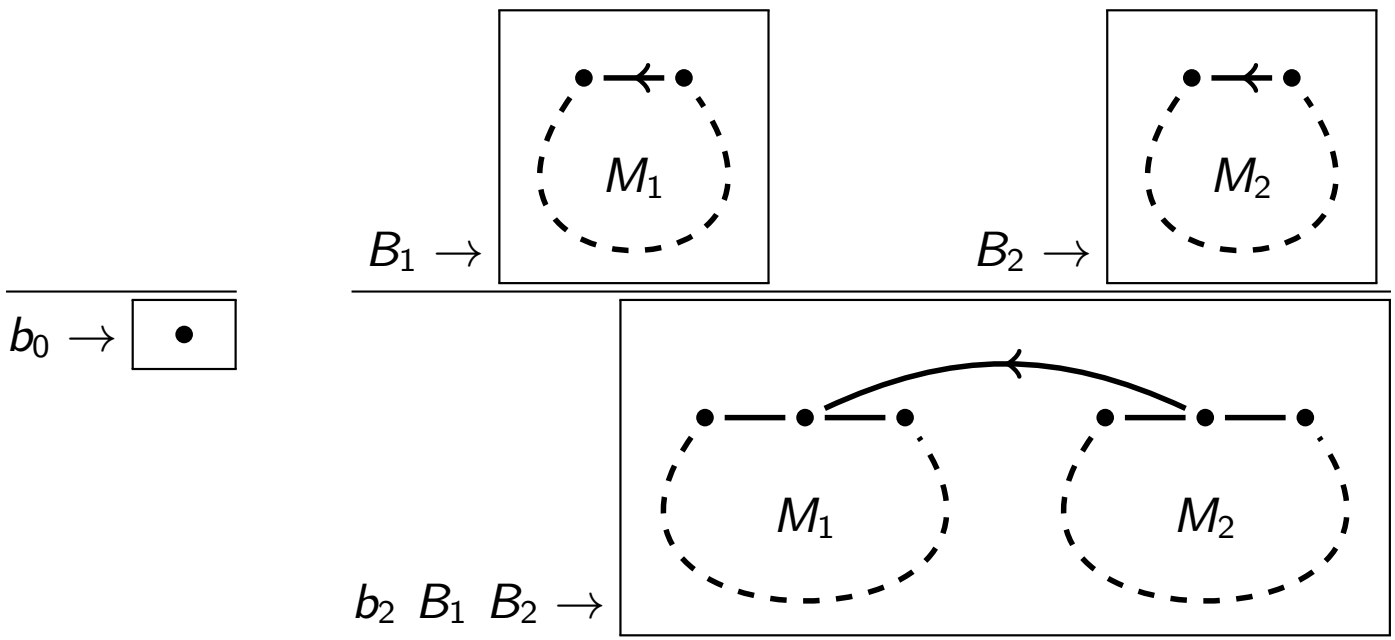
$b_1(1, 0, b_0)$

RTM

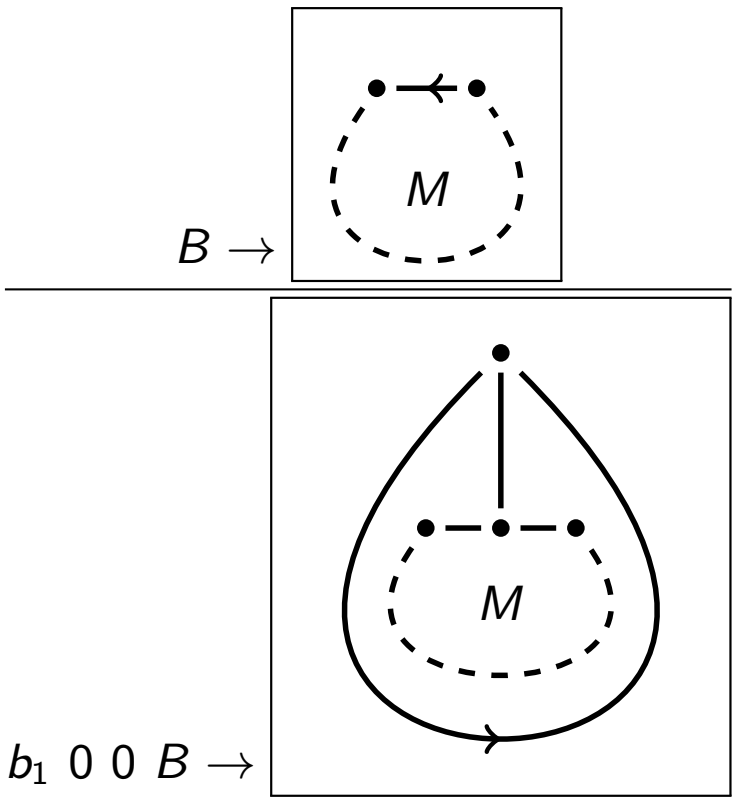
bloomrtm ⟶
⟵ rtmbloom



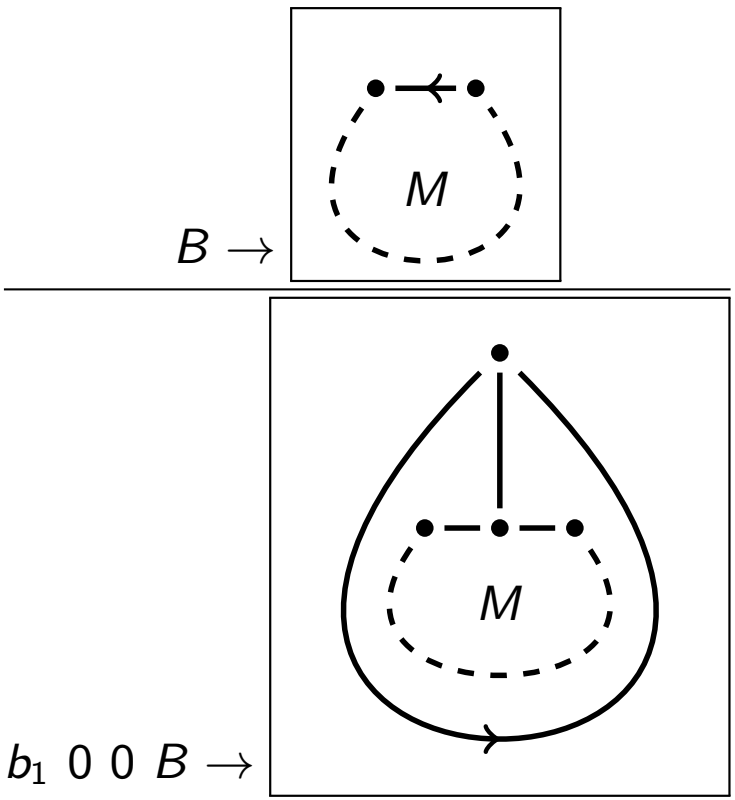
From blooms to RTM, visually (1/3)

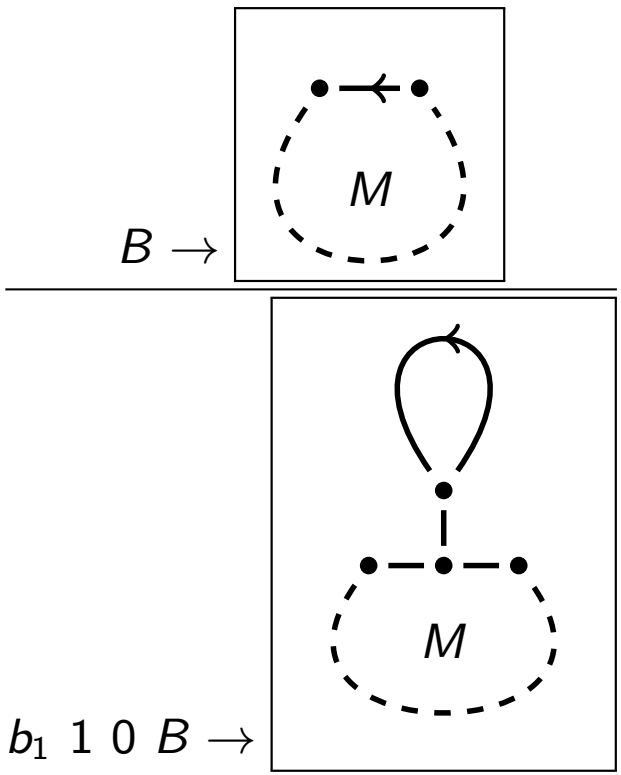


From blooms to RTM, visually (2/3)

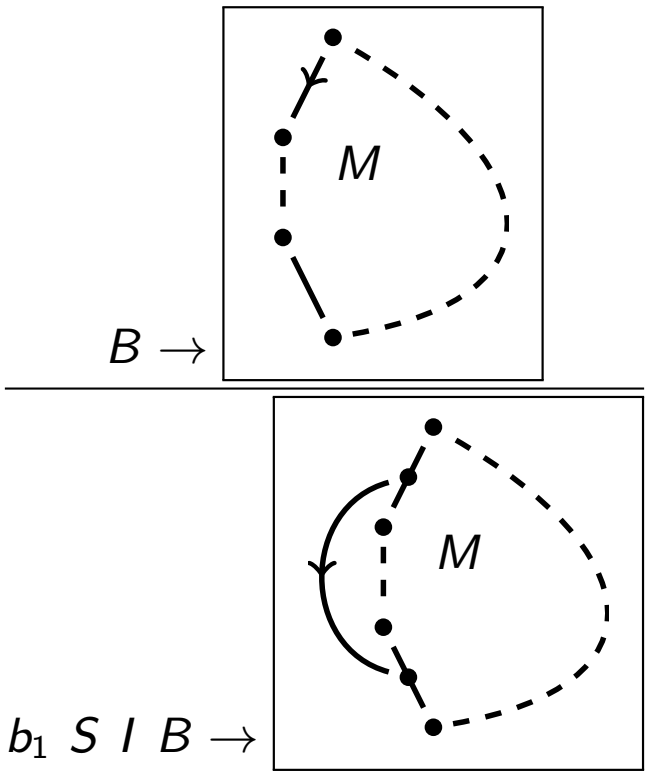


From blooms to RTM, visually (2/3)





From blooms to RTM, visually (3/3)



For $I > 0$
The splitted dart is chosen from I (rank in depth-first traversal) and S (orientation)

How to formalize maps?

- As topological objects? ☹ ☹
- As combinatorial objects (transitive couples of permutations on half-edges, modulo edge renaming)? [Dubois, _, Zeilberger, CLA'15]
- As an inductive structure constructed by edge and vertex additions? ☺
- How to discover such a structure?

Rooted map analysis and synthesis

Analysis by root edge removal

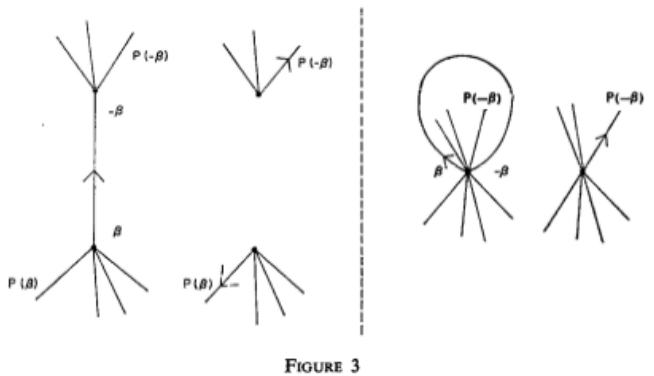


FIGURE 3

If the reduction does not disconnect the graph, it leaves an ordered graph with $(b-1)+p$ edges and $p+1$ vertices. Unless there are no longer any edges, the root-vertex has not been isolated; so shift the root to $P-(\beta)$. This reduction may be reversed in $2(b+p)-1$ ways, because the non-root end $-\beta$ of the root-edge must be the next end clockwise after the root of the reduced ordered graph, and the root β may now be inserted into one of $2[(b-1)+p]+1=2(b+p)-1$ slots. So we have

$$F_{0,0}=1, \qquad \text{and unless } b \text{ and } p \text{ are both zero}$$

$$F_{b,p}=\sum_{j=0}^{p-1}\sum_{k=0}^{b} F_{k,j}F_{b-k,p-j-1}+[2(b+p)-1]F_{b-1,p}. \tag{2}$$

T. R. S. Walsh and A. B. Lehman.

Counting rooted maps by genus I.

J. Comb. Theory, Ser. B, 1972.

Rooted map analysis and synthesis

Analysis by root edge removal

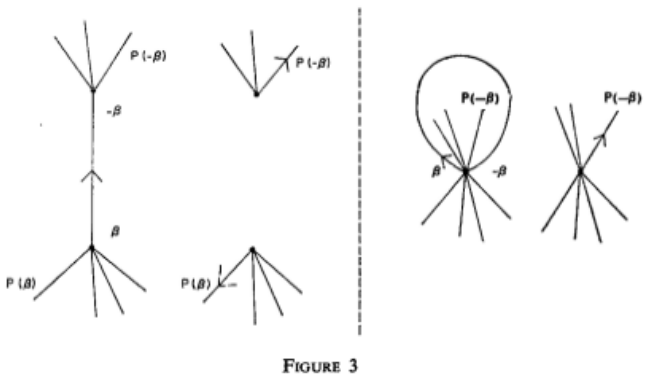


FIGURE 3

If the reduction does not disconnect the graph, it leaves an ordered graph with $(b-1)+p$ edges and $p+1$ vertices. Unless there are no longer any edges, the root-vertex has not been isolated; so shift the root to $P-(\beta)$. This reduction may be reversed in $2(b+p)-1$ ways, because the non-root end $-\beta$ of the root-edge must be the next end clockwise after the root of the reduced ordered graph, and the root β may now be inserted into one of $2[(b-1)+p]+1=2(b+p)-1$ slots. So we have

$$F_{0,0}=1, \qquad \text{and unless } b \text{ and } p \text{ are both zero}$$

$$F_{b,p}=\sum_{j=0}^{p-1}\sum_{k=0}^{b} F_{k,j}F_{b-k,p-j-1}+[2(b+p)-1]F_{b-1,p}. \tag{2}$$

Theorem (Rooted map synthesis)

Any rooted map M with $e(M)$ edges is either the vertex map (no edges) or can be uniquely obtained by adding an edge

- *either between two rooted maps M_1 and M_2 s.t. $e(M)=1+e(M_1)+e(M_2)$,*
- *or to a rooted map M_1 with $e(M_1)=e(M)-1$ edges in $2e(M_1)+1$ distinct ways.*

T. R. S. Walsh and A. B. Lehman.
Counting rooted maps by genus I.
J. Comb. Theory, Ser. B, 1972.

Rooted map analysis and synthesis

Analysis by root edge removal

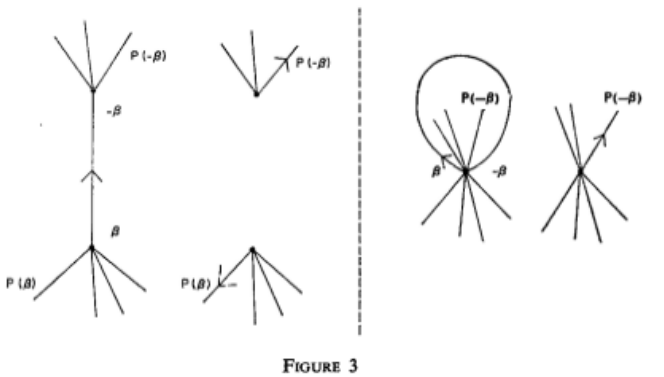


FIGURE 3

If the reduction does not disconnect the graph, it leaves an ordered graph with $(b-1)+p$ edges and $p+1$ vertices. Unless there are no longer any edges, the root-vertex has not been isolated; so shift the root to $P-(\beta)$. This reduction may be reversed in $2(b+p)-1$ ways, because the non-root end $-\beta$ of the root-edge must be the next end clockwise after the root of the reduced ordered graph, and the root β may now be inserted into one of $2[(b-1)+p]+1=2(b+p)-1$ slots. So we have

$$F_{0,0}=1, \qquad \text{and unless } b \text{ and } p \text{ are both zero}$$

$$F_{b,p}=\sum_{j=0}^{p-1}\sum_{k=0}^{b} F_{k,j}F_{b-k,p-j-1}+[2(b+p)-1]\,F_{b-1,p}. \tag{2}$$

T. R. S. Walsh and A. B. Lehman.

Counting rooted maps by genus I.
J. Comb. Theory, Ser. B, 1972.

Theorem (Rooted map synthesis)

Any rooted map M with $e(M)$ edges is either the vertex map (no edges) or can be uniquely obtained by adding an edge

- *either between two rooted maps M_1 and M_2 s.t. $e(M)=1+e(M_1)+e(M_2)$,*
- *or to a rooted map M_1 with $e(M_1)=e(M)-1$ edges in $2e(M_1)+1$ distinct ways.*

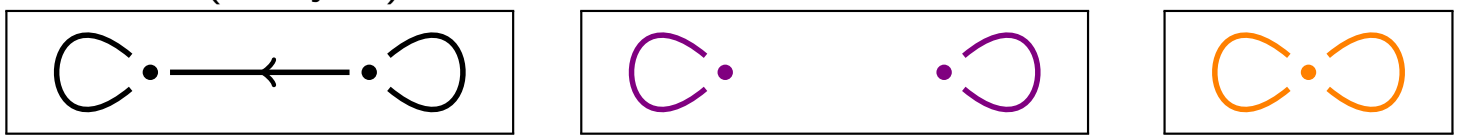
C. Dubois, A. Giorgetti, and R. Genestier.

Tests and proofs for enumerative combinatorics.
In *TAP'16*, volume 6792 of *LNCS*, pages 57–75. Springer, 2016.

- Formal proofs in Coq
- Definitions and conjectures checked
- Prolog-based bounded-exhaustive testing

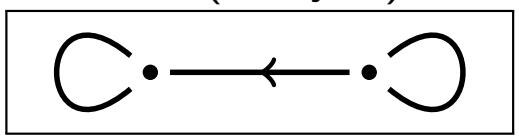
Root edge removal/addition

Removal (analysis): → deletion → contraction



Root edge removal/addition

Removal (analysis):

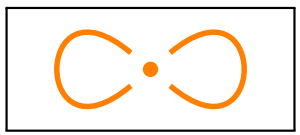


Addition (synthesis):

→ deletion

← drawing

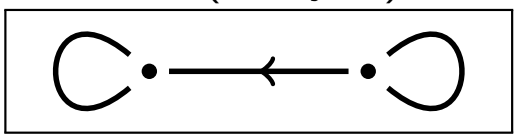
→ contraction



← stretching

Root edge removal/addition

Removal (analysis):

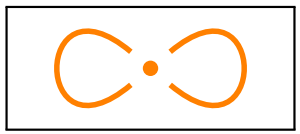


Addition (synthesis):

→ deletion

← drawing

→ contraction

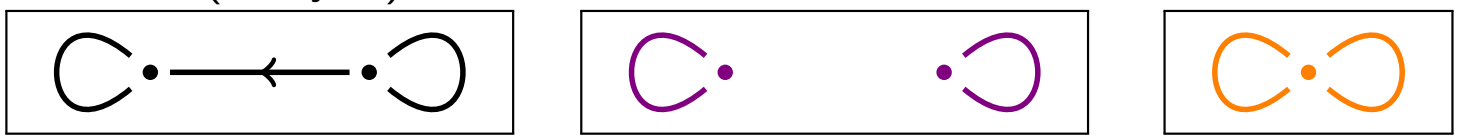


← stretching

- Works well for planar maps [Tutte, 68] and maps with any genus [Walsh & Lehman, 72]

Root edge removal/addition

Removal (analysis): → deletion → contraction

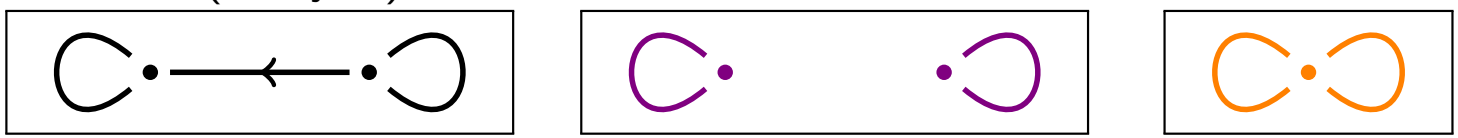


Addition (synthesis): ← drawing ← stretching

- Works well for planar maps [Tutte, 68] and maps with any genus [Walsh & Lehman, 72]
- More difficult for maps with degree constraints: deletion reduces vertex degrees, contraction increases them!
- Deletion only (for trivalent maps): Consider all the families of maps with vertex degrees 3 or less?
- With contraction: Also consider families of maps with arbitrary vertex degrees?

Root edge removal/addition

Removal (analysis): → deletion → contraction



Addition (synthesis): ← drawing ← stretching

- Works well for planar maps [Tutte, 68] and maps with any genus [Walsh & Lehman, 72]
- More difficult for maps with degree constraints: deletion reduces vertex degrees, contraction increases them!
- Deletion only (for trivalent maps): Consider all the families of maps with vertex degrees 3 or less?
- With contraction: Also consider families of maps with arbitrary vertex degrees?
- Here, a solution for trivalent maps, with only four families of maps

Method for rooted trivalent maps

1. Map decomposition/analysis by root edge removal
2. Inverse constructions: map synthesis by edge addition
3. Mutually inductive families of maps formalized as (Prolog) terms
4. Validation by counting (OEIS)
5. Interpretation of constructors as operations on combinatorial maps, represented by permutations

Method for rooted trivalent maps

1. Map decomposition/analysis by root edge removal
2. Inverse constructions: map synthesis by edge addition
3. Mutually inductive families of maps formalized as (Prolog) terms
4. Validation by counting (OEIS)
5. Interpretation of constructors as operations on combinatorial maps, represented by permutations

- Done for rooted maps [DGG16]
- Here for rooted trivalent maps
- Steps 1-3 only for one edge removal/addition

Method for rooted trivalent maps

1. Map decomposition/analysis by root edge removal
2. Inverse constructions: map synthesis by edge addition
3. Mutually inductive families of maps formalized as (Prolog) terms
4. Validation by counting (OEIS)
5. Independent inductive characterization of each map family by incremental formal composition of up to 3 constructions
6. Interpretation of constructors as operations on combinatorial maps, represented by permutations

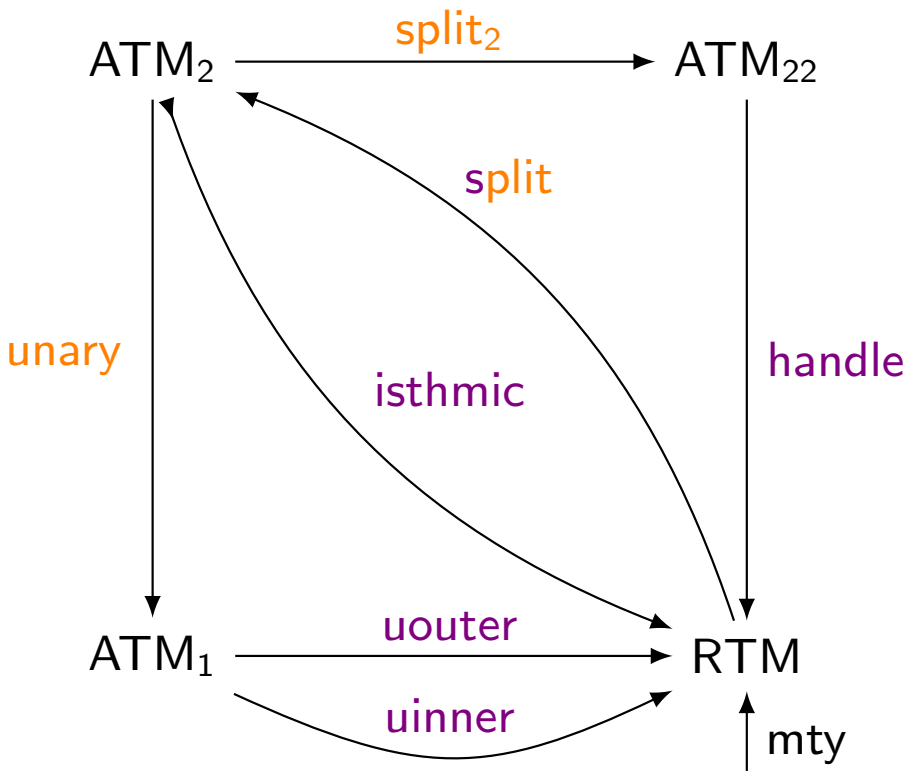
- Done for rooted maps [DGG16]
- Here for rooted trivalent maps
- Steps 1-3 only for one edge removal/addition

Method for rooted trivalent maps

1. Map decomposition/analysis by root edge removal
2. Inverse constructions: map synthesis by edge addition
3. Mutually inductive families of maps formalized as (Prolog) terms
4. Validation by counting (OEIS)
5. Independent inductive characterization of each map family by incremental formal composition of up to 3 constructions
6. Validation by counting (OEIS) at each intermediate step
7. Interpretation of constructors as operations on combinatorial maps, represented by permutations

- Done for rooted maps [DGG16]
- Here for rooted trivalent maps
- Steps 1-3 only for one edge removal/addition

Global picture

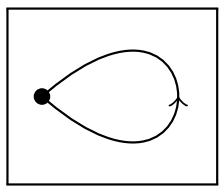
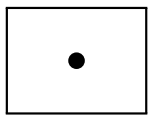


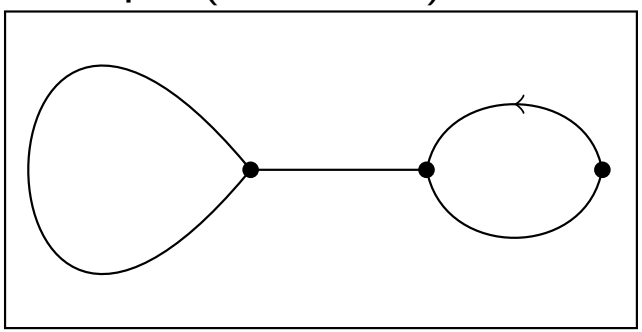
- ATM_1: univalent root vertex (degree 1), other vertices of degree 3
- ATM_{22}: ordered pair of roots, two distinct bivalent root vertices

Construction of ATM_2

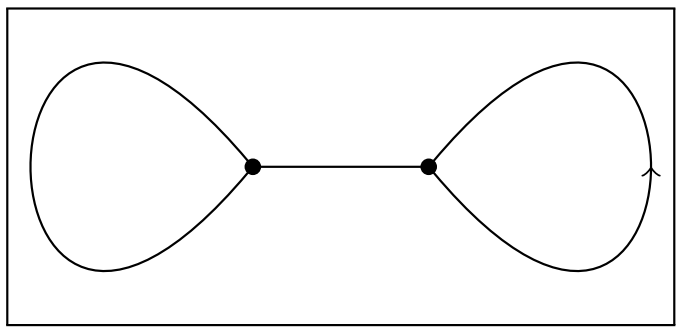
Construction of ATM_2

$$ATM_2 \xleftarrow{\text{split}} RTM$$

- For `E = 0`,  from the vertex map `mty` = 
- Example (for `E = 4`)

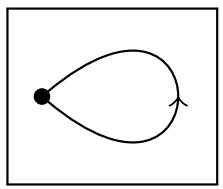
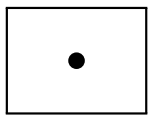


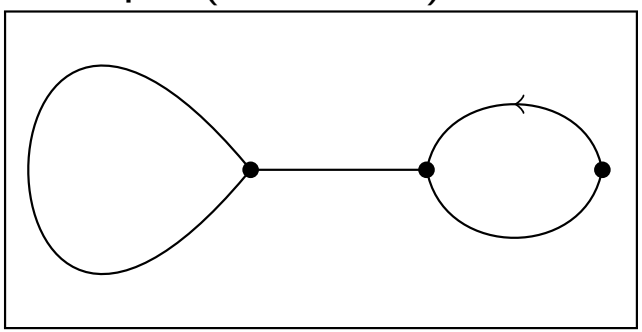
from



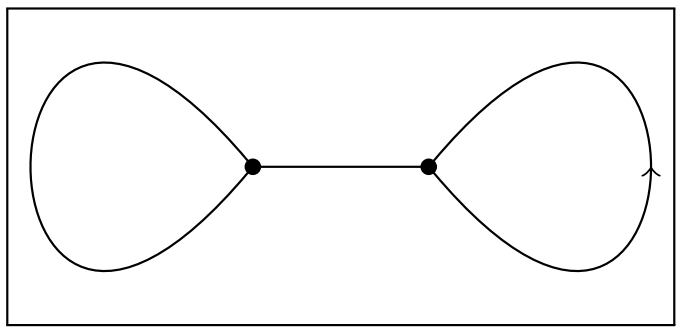
Construction of ATM_2

$$ATM_2 \xleftarrow{\text{split}} RTM$$

- For `E = 0`,  from the vertex map `mty` = 
- Example (for `E = 4`)



from



```
atm2(split(M),E) :- E > 0, D is E-1, rtm(M,D).
```

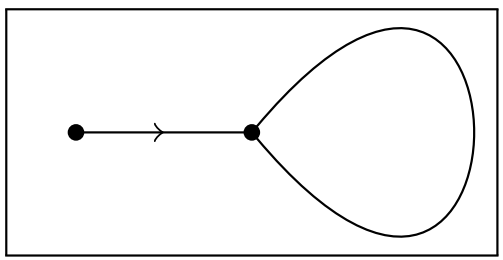
Construction of ATM_1

ATM_2

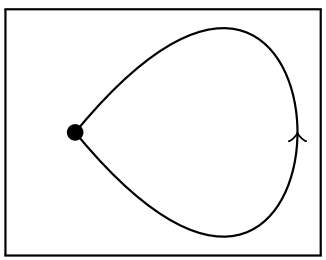
↓ unary

ATM_1

Example



from



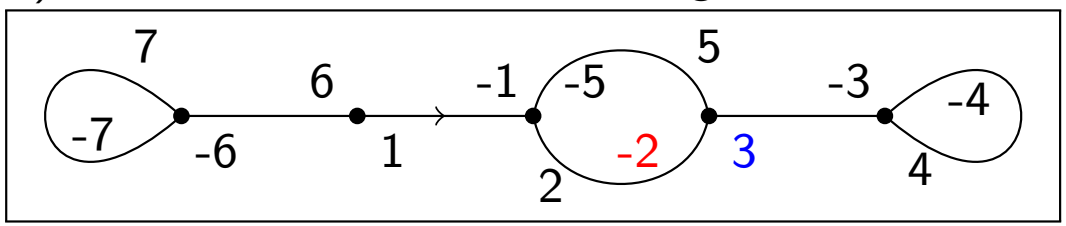
```
atm1(unary(M),E) :- E > 1, D is E-1, atm2(M,D).
```

Construction of ATM_{22}

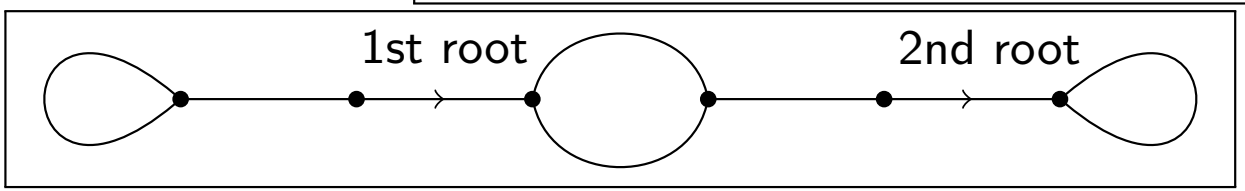
$$ATM_2 \xrightarrow{\text{split}_2} ATM_{22}$$

Add a (thus bivalent) vertex at the middle of an edge:

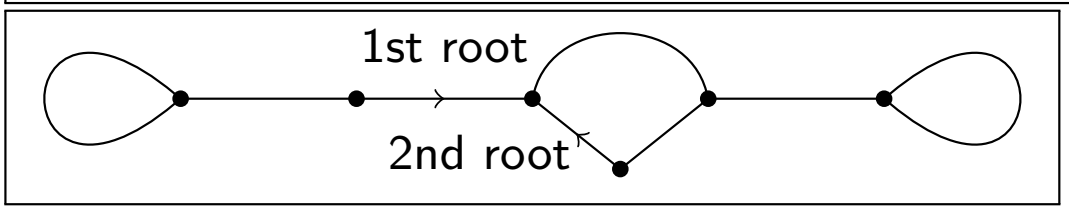
From the ATM_2



get



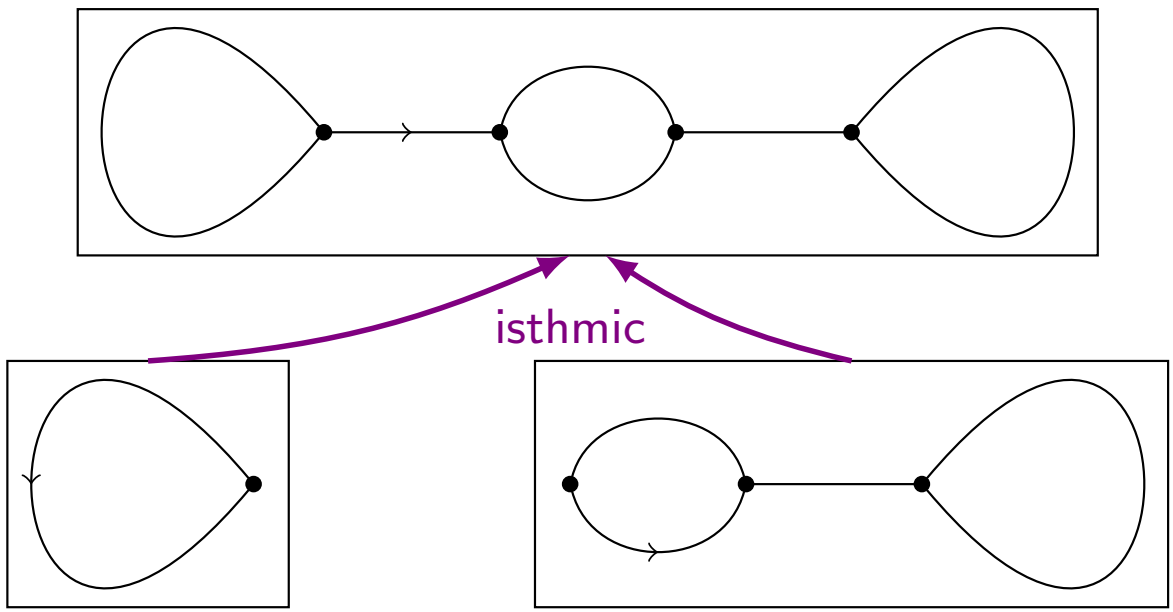
for I = 3 and



for I = -2

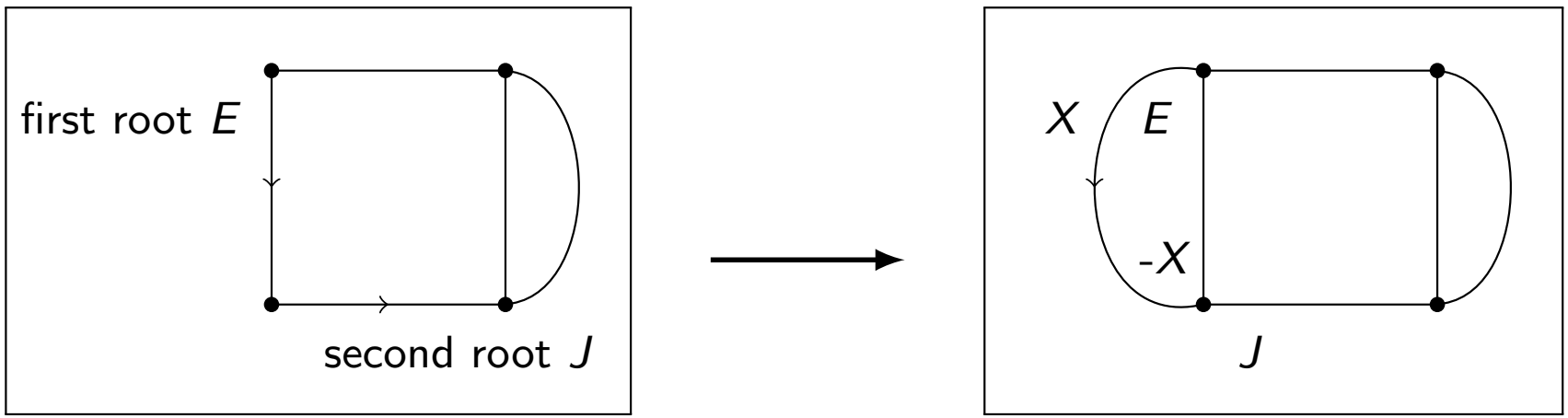
```
atm22(split2(I,M),E) :- E > 0, D is E-1, atm2(M,D),
 Dopp is -D, in(I,Dopp,D), I =\= 0.
```

Construction of RTM: Isthmic case



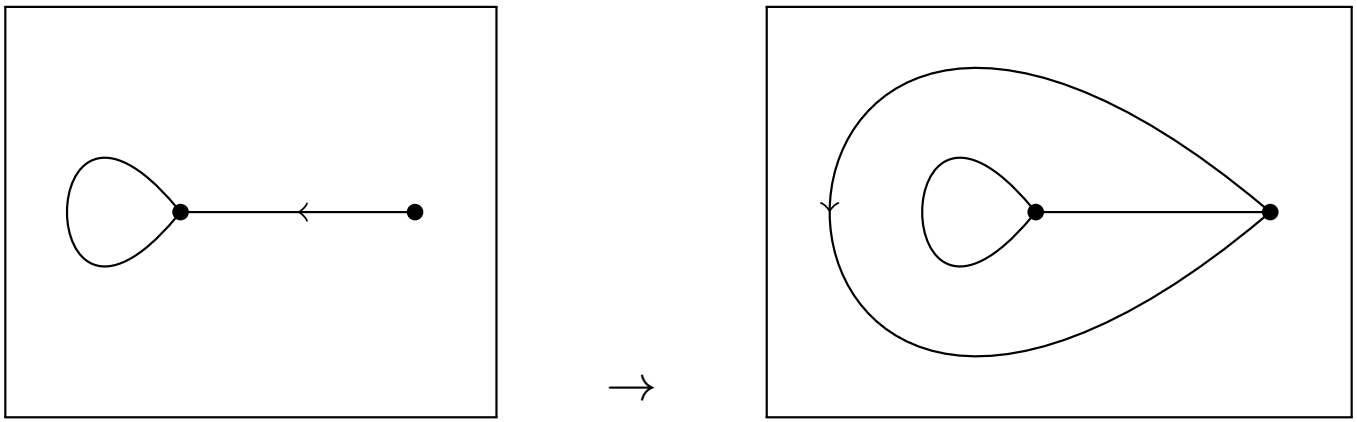
- Binary isthmic construction of one RTM from two ATM_2

Construction of RTM: case 2



ATM_{22} handle RTM

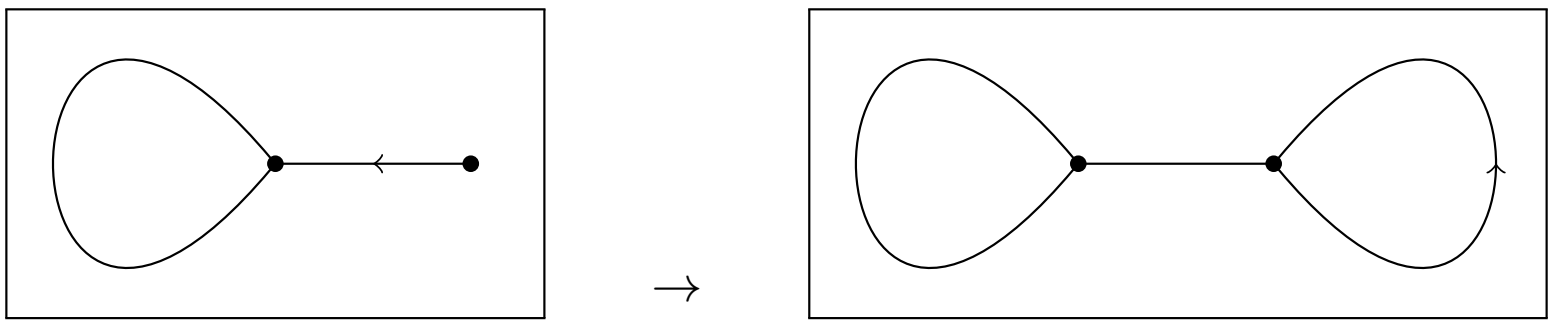
Construction of RTM: case 3



ATM_1 uouter RTM

The root face is unary

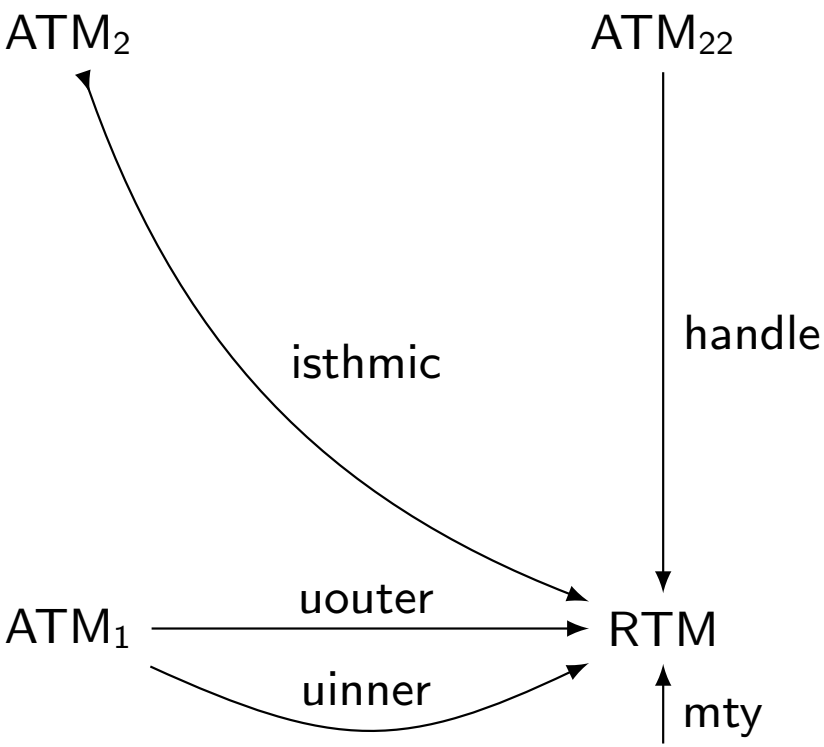
Construction of RTM: case 4



ATM_1 uinner RTM

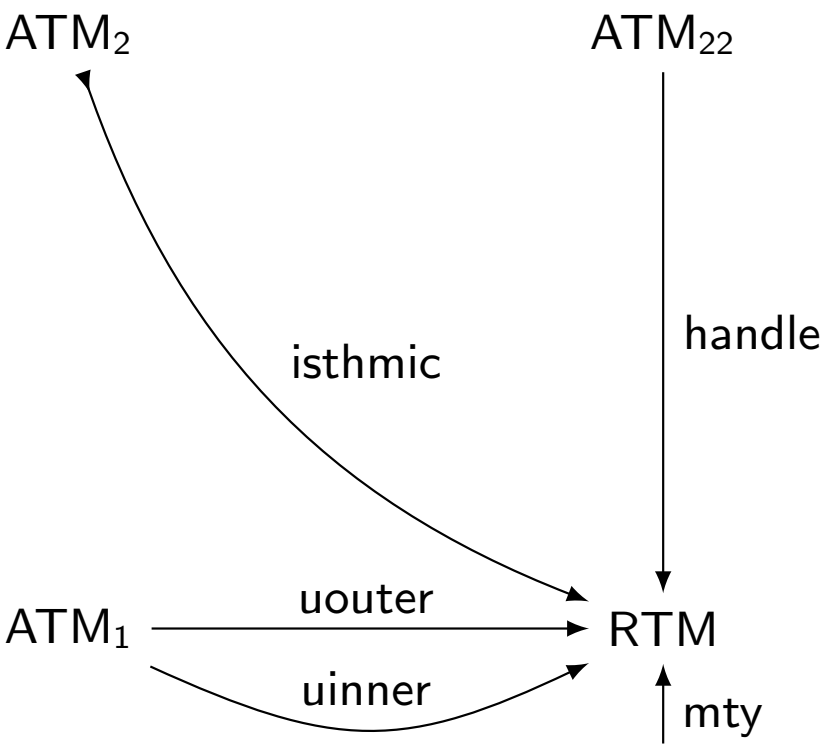
The inner face (incident to the opposite of the root) is unary

Construction of trivalent maps: 5 cases

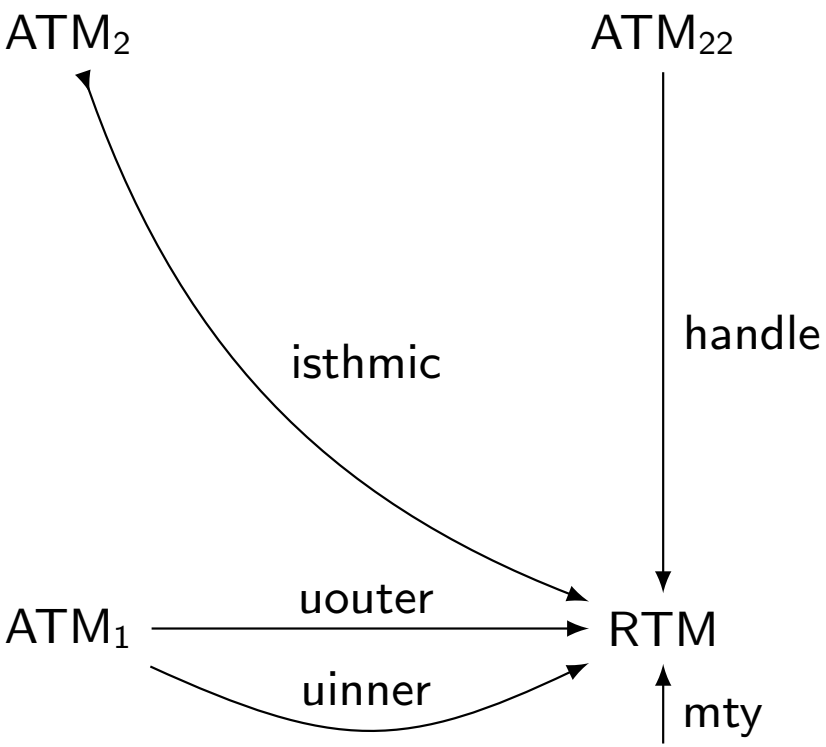


Construction of trivalent maps: 5 cases

```
rtm(mty,0).
```

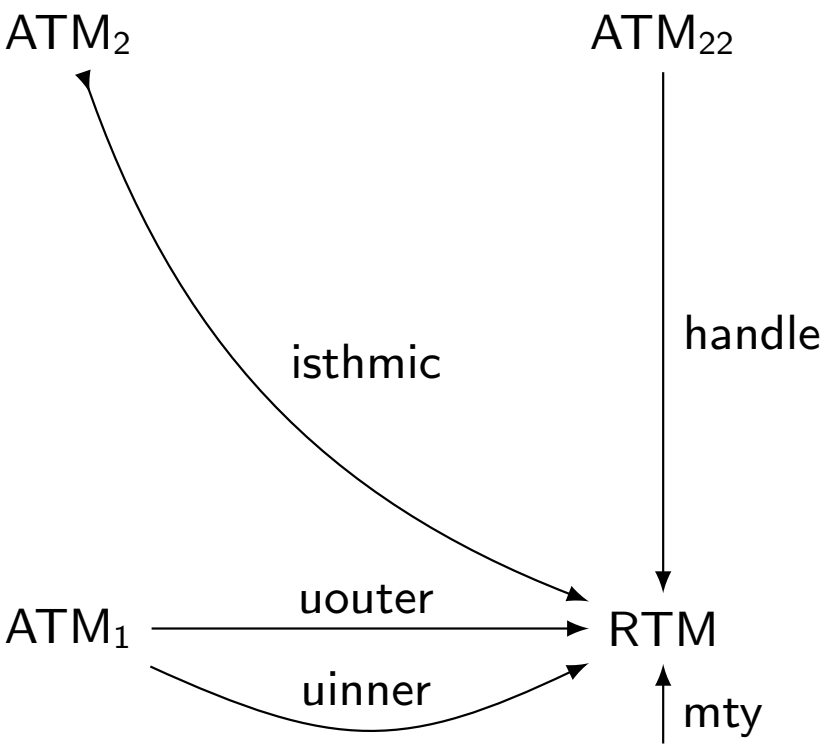


Construction of trivalent maps: 5 cases



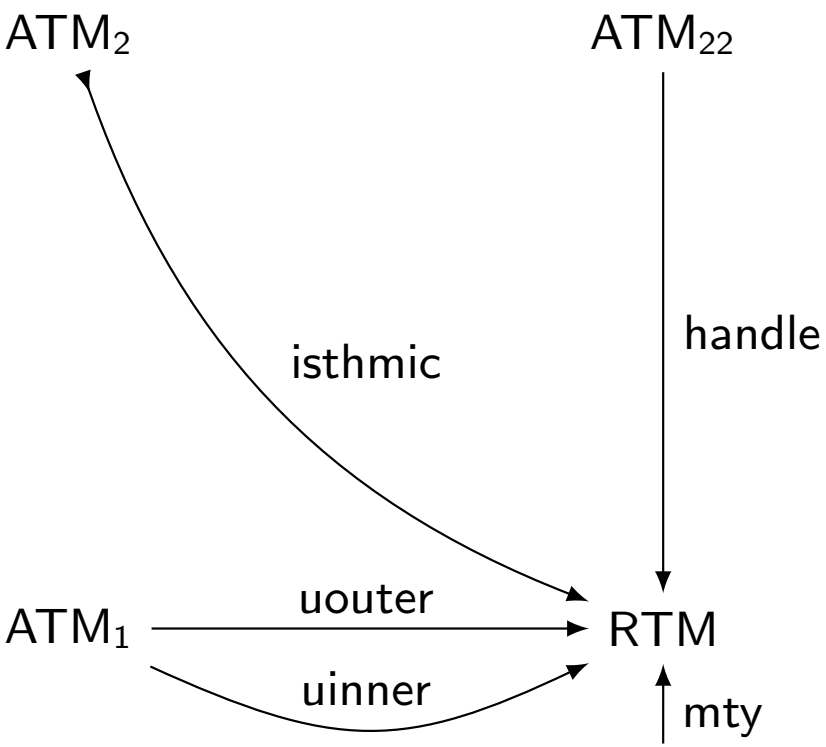

```
rtm(mty,0).

rtm(isthmic(M2,U2),E) :-
 E > 2, D is E-1, in(E1,1,D),
 atm2(M2,E1),
 E2 is D-E1, E2 > 0,
 atm2(U2,E2).
```

Construction of trivalent maps: 5 cases



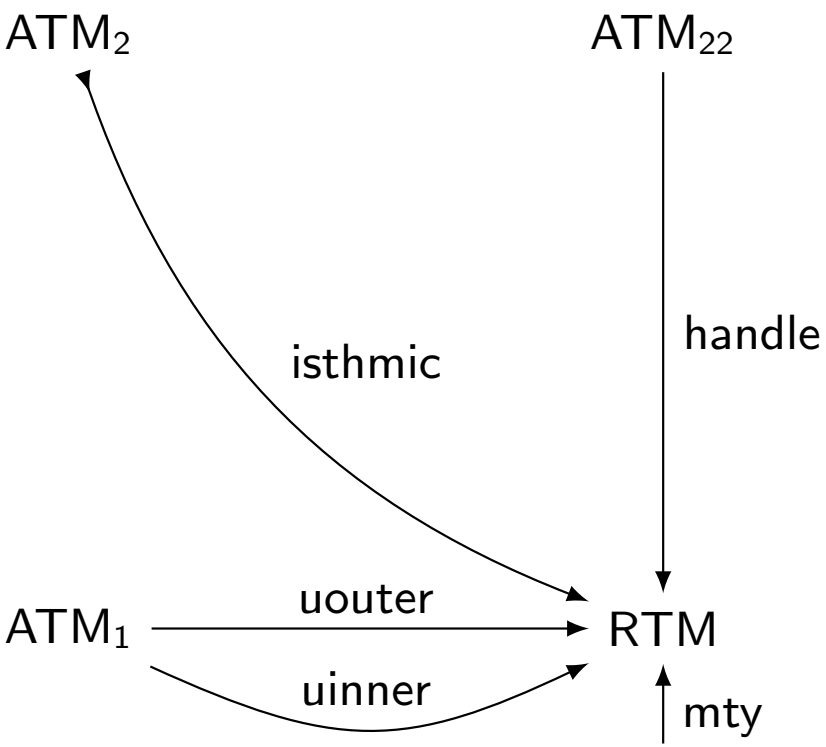

```
rtm(mty,0).

rtm(isthmic(M2,U2),E) :-
 E > 2, D is E-1, in(E1,1,D),
 atm2(M2,E1),
 E2 is D-E1, E2 > 0,
 atm2(U2,E2).

rtm(uouter(M),E) :- E > 0,
 D is E-1, atm1(M,D).

rtm(uinner(M),E) :- E > 0,
 D is E-1, atm1(M,D).
```

Construction of trivalent maps: 5 cases




```
rtm(mty,0).

rtm(isthmic(M2,U2),E) :-
 E > 2, D is E-1, in(E1,1,D),
 atm2(M2,E1),
 E2 is D-E1, E2 > 0,
 atm2(U2,E2).

rtm(uouter(M),E) :- E > 0,
 D is E-1, atm1(M,D).

rtm(uinner(M),E) :- E > 0,
 D is E-1, atm1(M,D).

rtm(handle(M),E) :-
 E > 0, D is E-1, atm22(M,D).
```

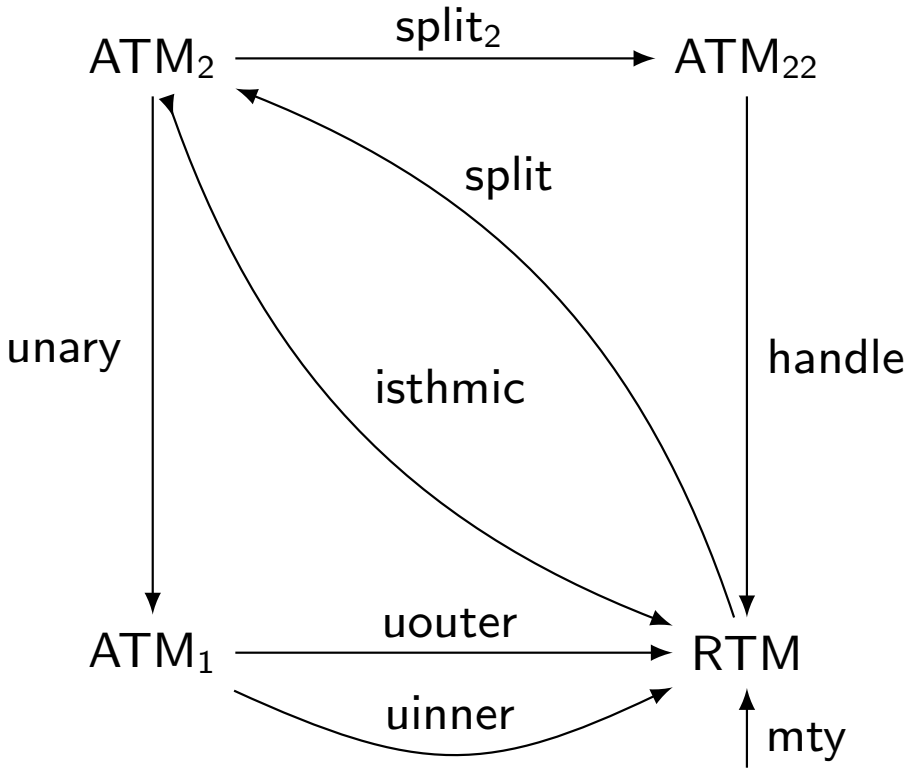
Construction of trivalent maps: 5 cases




```
rtm(mty,0).

rtm(isthmic(M2,U2),E) :-
 E > 2, D is E-1, in(E1,1,D),
 atm2(M2,E1),
 E2 is D-E1, E2 > 0,
 atm2(U2,E2).

rtm(uouter(M),E) :- E > 0,
 D is E-1, atm1(M,D).

rtm(uinner(M),E) :- E > 0,
 D is E-1, atm1(M,D).

rtm(handle(M),E) :-
 E > 0, D is E-1, atm22(M,D).
```

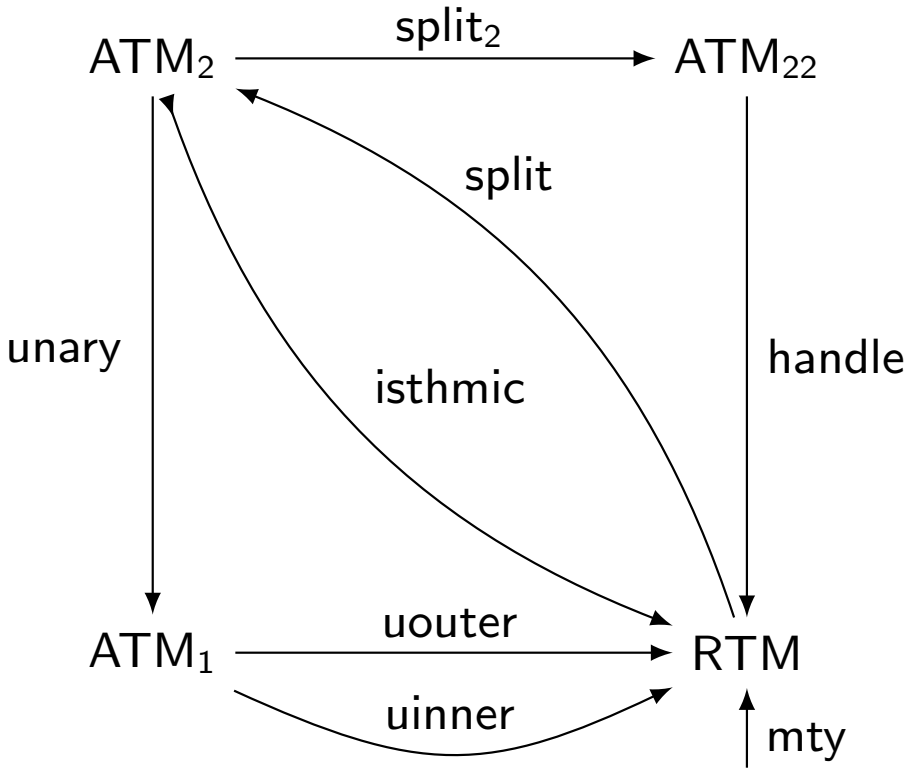
Validation by counting: 1, 5, 60, 1105, 27120, 828250, 30220800, in less than 1 minute (`http://oeis.org/A062980`)

Where formalization helps



Elimination of intermediate maps
by composition of constructors

Where formalization helps



Elimination of intermediate maps by composition of constructors

1. Composition of `split` and `unary`, detailed
2. Elimination of ATM_{22}
3. Elimination of ATM_1
4. Elimination of ATM_2
5. Put everything together

Step 1: Composition of `split` and `unary`

Since the family ATM_2 is characterized by

```
atm2(split(M),E) :- E > 0, D is E-1, rtm(M,D).
```

the clause

```
atm1(unary(M),E) :- E > 1, D is E-1, atm2(M,D).
```

can be replaced with

- `atm1(unary(split(M)),E) :- E > 1, D is E-1, atm2(split(M),D).`

  by case analysis,

- `atm1(unary(split(M)),E) :- E > 1, D is E-1, D > 0, C is D-1, rtm(M,C).`

  by unfolding, and

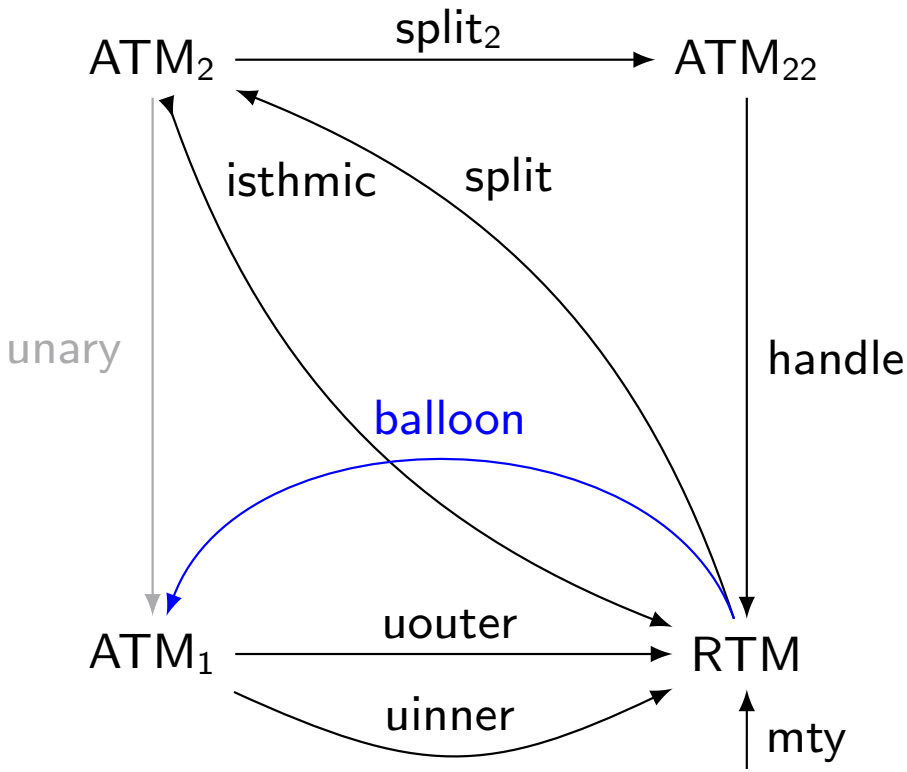
- `atm1(balloon(M),E) :- E > 1, C is E-2, rtm(M,C).`

  by folding, with `balloon = split;unary` (+ variable elimination)

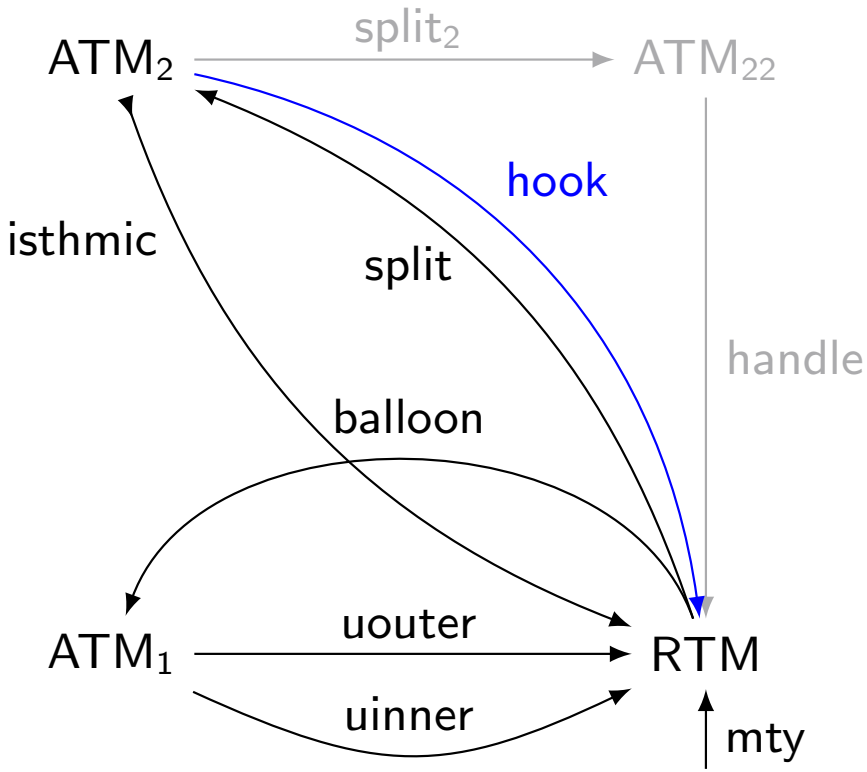
- Validation by counting: again `http://oeis.org/A062980`

New global picture

balloon(M) = unary(split(M))



Step 2: Elimination of ATM_{22}



hook = $split_2$;handle

Elimination of ATM_{22}, formally

With

```
hook(I,M) = handle(split2(I,M))
```

and

```
atm22(split2(I,M),E) :- E > 0, D is E-1, atm2(M,D), Max is 2*D,
 in(I,1,Max).
```

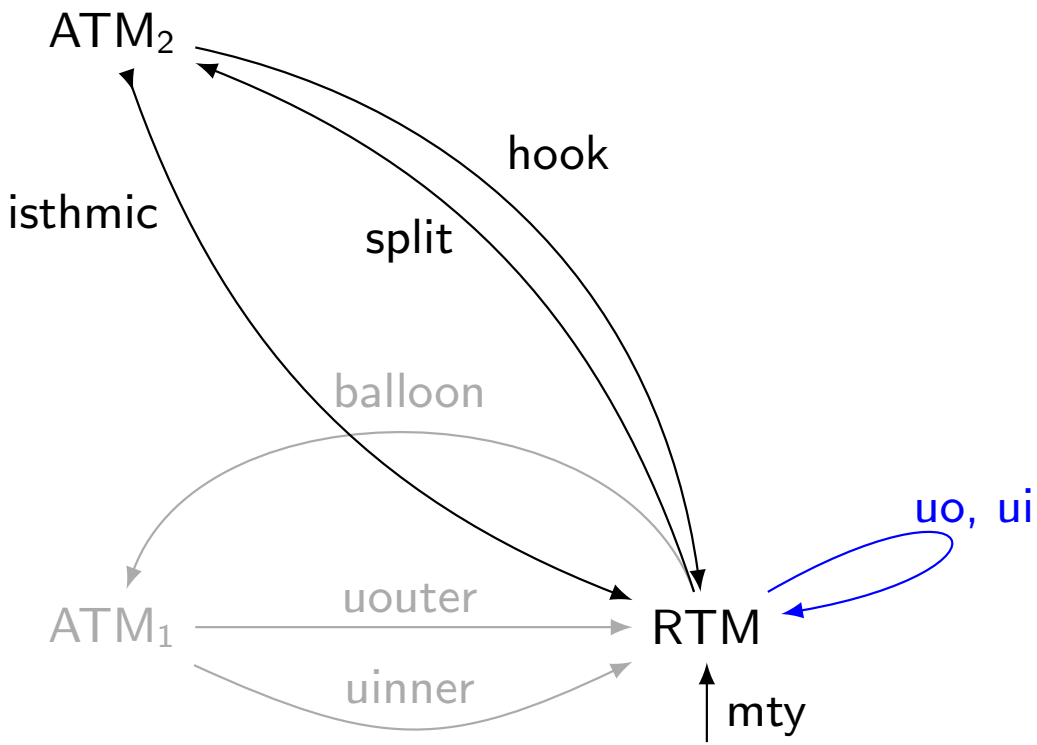
the clause

```
rtm(handle(M),E) :- E > 0, D is E-1, atm22(M,D).
```

becomes

```
rtm(hook(I,M),E) :- E > 1, C is E-2, atm2(M,C), Max is 2*C,
 in(I,1,Max).
```

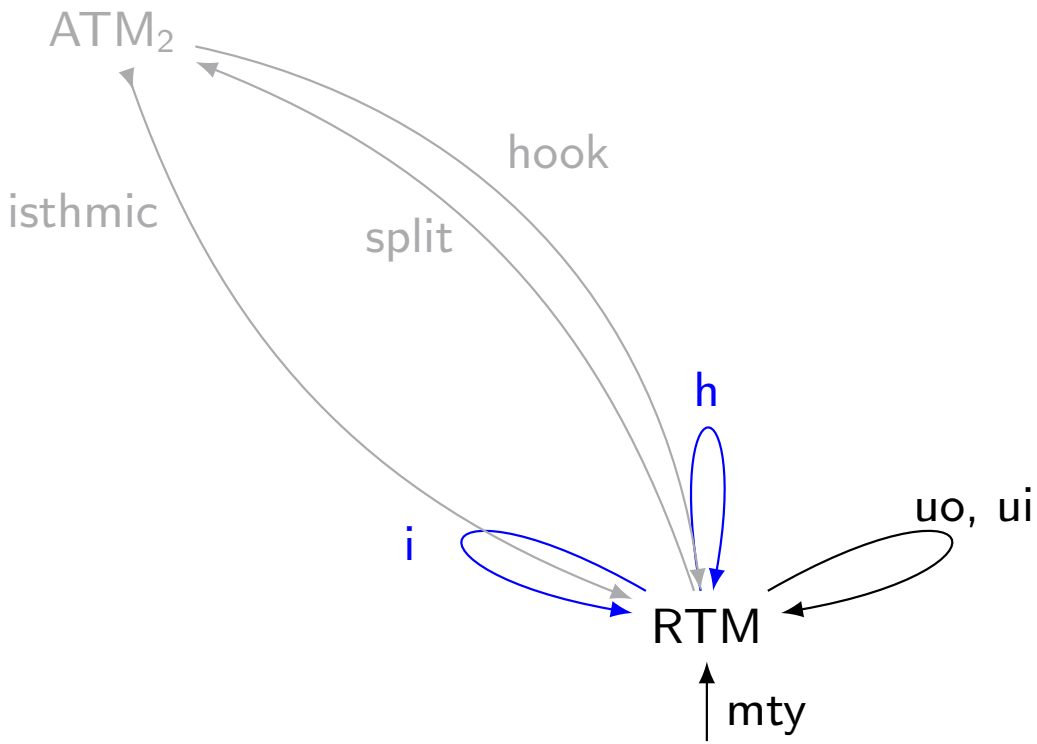
Step 3: Elimination of ATM_1



uo = balloon;uouter and ui = balloon;uinner

```
rtm(uo(M),E) :- E > 2, B is E-3, rtm(M,B).
rtm(ui(M),E) :- E > 2, B is E-3, rtm(M,B).
```

Step 4: Elimination of ATM_2



h(I,M) = hook(I,split(M))
i(M1,M2) = isthmic(split(M1),split(M2))

```
rtm(h(I,M),E) :- E > 2, B is E-3, rtm(M,B), Max is 2*B+2, in(I,1,Max).
rtm(i(M1,M2),E) :- E > 2, B is E-3, in(E1,0,B), rtm(M1,E1),
 E2 is B-E1, rtm(M2,E2).
```

Inductive definition of trivalent maps

- All together

```
rtm(mty,0).
rtm(i(M1,M2),E) :- E > 2, B is E-3, in(E1,0,B),
 rtm(M1,E1), E2 is B-E1, rtm(M2,E2).
rtm(uo(M),E) :- E > 2, B is E-3, rtm(M,B).
rtm(ui(M),E) :- E > 2, B is E-3, rtm(M,B).
rtm(h(I,M),E) :- E > 2, B is E-3, rtm(M,B), Max is 2*B+2,
 in(I,1,Max).
```

Inductive definition of trivalent maps

- All together

```
rtm(mty,0).
rtm(i(M1,M2),E) :- E > 2, B is E-3, in(E1,0,B),
 rtm(M1,E1), E2 is B-E1, rtm(M2,E2).
rtm(uo(M),E) :- E > 2, B is E-3, rtm(M,B).
rtm(ui(M),E) :- E > 2, B is E-3, rtm(M,B).
rtm(h(I,M),E) :- E > 2, B is E-3, rtm(M,B), Max is 2*B+2,
 in(I,1,Max).
```

- Unary cases merged (`K = I+2`)

```
rtm(mty,0).
rtm(b(M1,M2),E) :- E > 2, B is E-3, in(E1,0,B),
 rtm(M1,E1), E2 is B-E1, rtm(M2,E2).
rtm(u(K,M),E) :- E > 2, B is E-3, rtm(M,B), Max is 2*B+3,
 in(K,0,Max).
```

Inductive definition of trivalent maps

- All together

```
rtm(mty,0).
rtm(i(M1,M2),E) :- E > 2, B is E-3, in(E1,0,B),
 rtm(M1,E1), E2 is B-E1, rtm(M2,E2).
rtm(uo(M),E) :- E > 2, B is E-3, rtm(M,B).
rtm(ui(M),E) :- E > 2, B is E-3, rtm(M,B).
rtm(h(I,M),E) :- E > 2, B is E-3, rtm(M,B), Max is 2*B+2,
 in(I,1,Max).
```

- Unary cases merged (`K = I+2`)

```
rtm(mty,0).
rtm(b(M1,M2),E) :- E > 2, B is E-3, in(E1,0,B),
 rtm(M1,E1), E2 is B-E1, rtm(M2,E2).
rtm(u(K,M),E) :- E > 2, B is E-3, rtm(M,B), Max is 2*B+3,
 in(K,0,Max).
```

Inductive definition of trivalent maps

Let $n(M) = e(M)/3$ be the *size* of a rooted trivalent map M

```
rtm(mty,0).
rtm(b(M1,M2),N) :- N > 0, Nm1 is N-1, in(N1,0,Nm1),
 rtm(M1,N1), N2 is Nm1-N1, rtm(M2,N2).
rtm(u(K,M),N) :- N > 0, Nm1 is N-1, rtm(M,Nm1),
 Max is 6*Nm1+3, in(K,0,Max).
```

Inductive definition of trivalent maps

Let $n(M) = e(M)/3$ be the *size* of a rooted trivalent map M

```
rtm(mty,0).
rtm(b(M1,M2),N) :- N > 0, Nm1 is N-1, in(N1,0,Nm1),
 rtm(M1,N1), N2 is Nm1-N1, rtm(M2,N2).
rtm(u(K,M),N) :- N > 0, Nm1 is N-1, rtm(M,Nm1),
 Max is 6*Nm1+3, in(K,0,Max).
```

Theorem (Rooted trivalent map by size)

A rooted trivalent map M of size $n(M)$ is either the vertex map (size 0) or can be uniquely obtained by adding an edge

- *between two rooted trivalent maps M_1 and M_2 such that $n(M) = n(M_1) + n(M_2) + 1$,*
- *or to a rooted trivalent map M_1 with $n(M_1) = n(M) - 1$, in $6n(M_1) + 4$ distinct ways.*

Inductive definition of trivalent maps

Let $n(M) = e(M)/3$ be the *size* of a rooted trivalent map M

```
rtm(mty,0).
rtm(b(M1,M2),N) :- N > 0, Nm1 is N-1, in(N1,0,Nm1),
 rtm(M1,N1), N2 is Nm1-N1, rtm(M2,N2).
rtm(u(K,M),N) :- N > 0, Nm1 is N-1, rtm(M,Nm1),
 Max is 6*Nm1+3, in(K,0,Max).
```

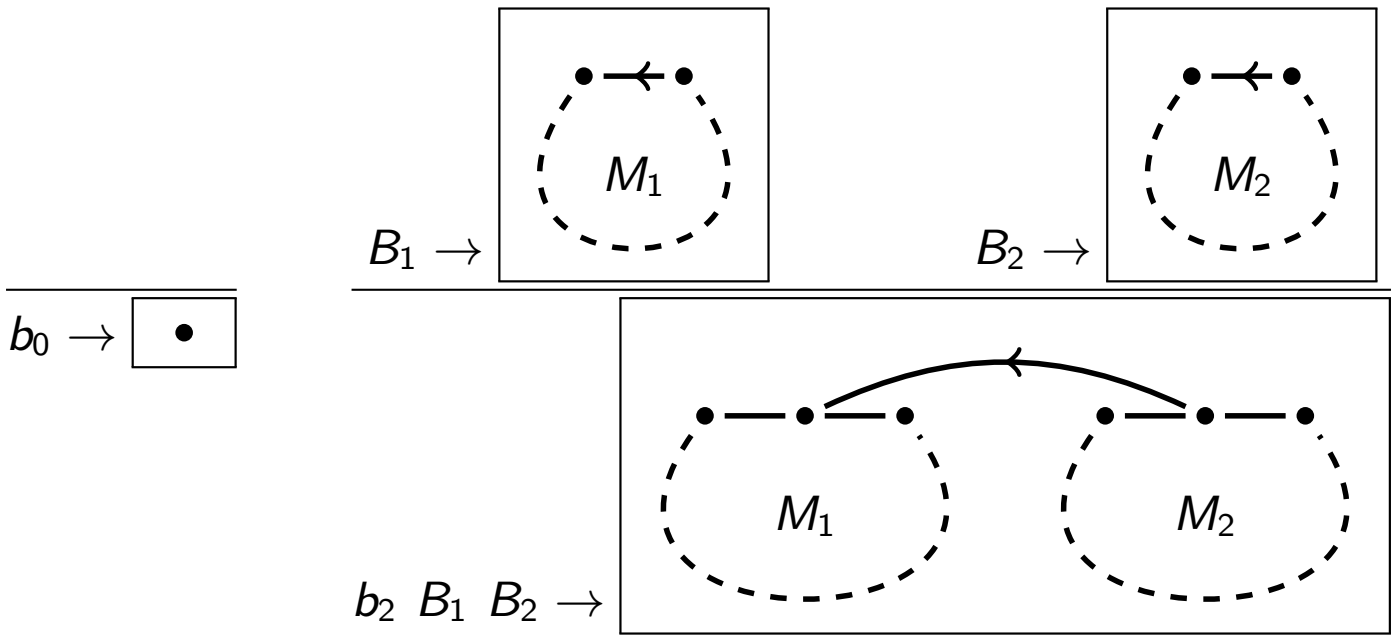
Theorem (Rooted trivalent map by size)

A rooted trivalent map M of size $n(M)$ is either the vertex map (size 0) or can be uniquely obtained by adding an edge

- *between two rooted trivalent maps M_1 and M_2 such that $n(M) = n(M_1) + n(M_2) + 1$,*
- *or to a rooted trivalent map M_1 with $n(M_1) = n(M) - 1$, in $6n(M_1) + 4$ distinct ways.*

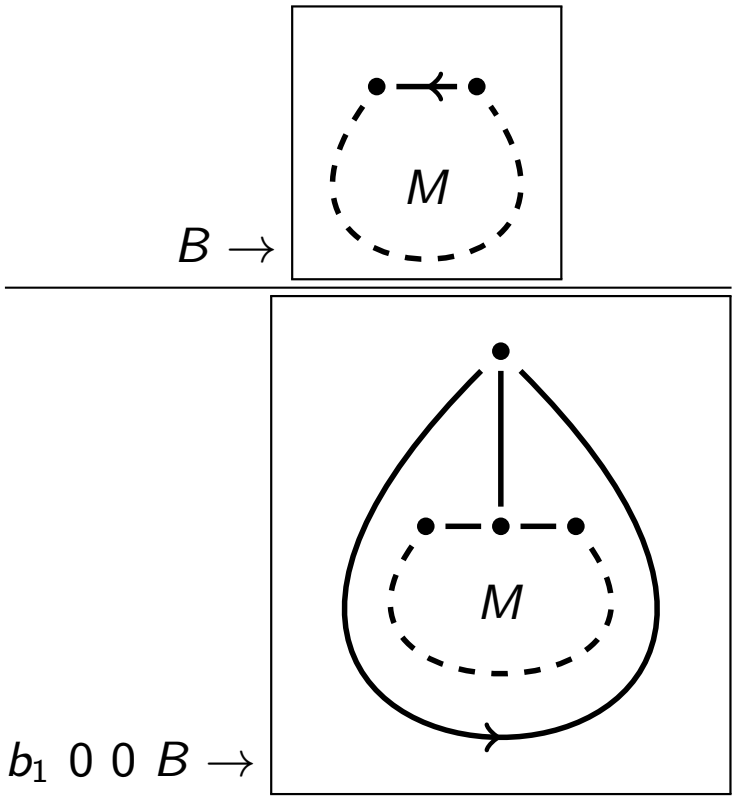
Back to blooms!

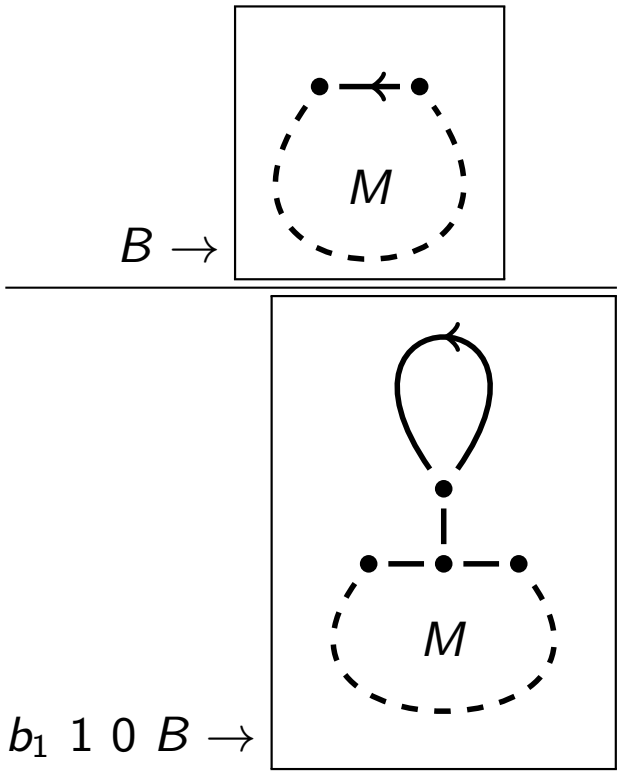
From blooms to RTM, formally (1/3)




```
bloomrtm(b0,mty).
bloomrtm(b2(B1,B2),isthmic(split(M2),split(M1))) :-
  bloomrtm(B1,M1), bloomrtm(B2,M2).
```

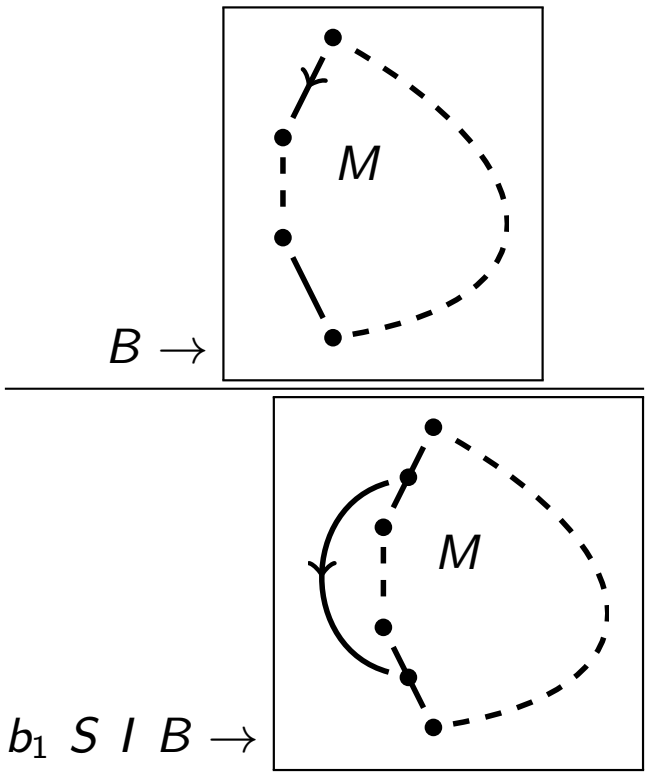
From blooms to RTM, formally (2/3)






```
bloomrtm(b1(0,0,B),uouter(unary(split(M)))) :- bloomrtm(B,M).
bloomrtm(b1(1,0,B),uinner(unary(split(M)))) :- bloomrtm(B,M).
```

From blooms to RTM, formally (3/3)




```
bloomrtm(b1(0,I,B),handle(split2(K,split(M)))) :- I > 0,
  nb01(B,N), K is 3*N-1-I, bloomrtm(B,M).
bloomrtm(b1(1,I,B),handle(split2(K,split(M)))) :- I > 0,
  nb01(B,N), K is -(3*N-1-I), bloomrtm(B,M).
```

Map semantics checked by counting

```
br([B,M],N) :- bloom(B,N), bloomrtm(B,M).

:- iterate(0,6,br).
```

$\rightsquigarrow$ 1, 5, 60, 1105, 27120, 828250, 30220800 (`http://oeis.org/A062980`)

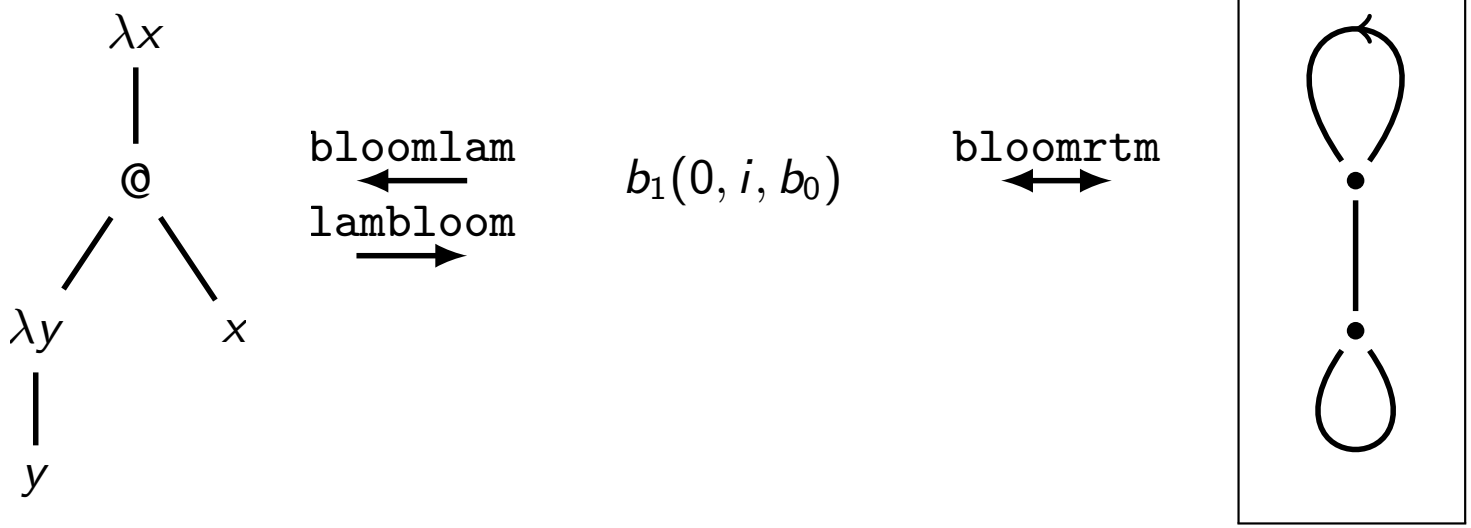
Not sufficient: several other interpretations also hold!

$\rightsquigarrow$ labeled structures [Dubois, _, Zeilberger, CLA'15]

Outline

Other point of view

λ terms | Blooms | RTM



Adapted to random and bounded-exhaustive generation, maybe also to formal proof

`bloomrtm` in Prolog with CLP(FD) is a binary relation specifying the two transformations

Work in progress

- Justification of map semantics (`bloomrtm`) with labeled structures
- Some formal proofs remain difficult, even with the dependent type for blooms
- Correspondence with SwissCheese datatype?

Pierre Lescanne.
Quantitative aspects of linear and affine closed lambda terms.
arXiv:1702.03085, CLA'2017.

- Correspondence with well-orientations of maps?

Noam Zeilberger.
A theory of linear typings as flows on 3-valent graphs.
Logical Methods in Computer Science, 2018.

- Thanks for your attention
- Questions?