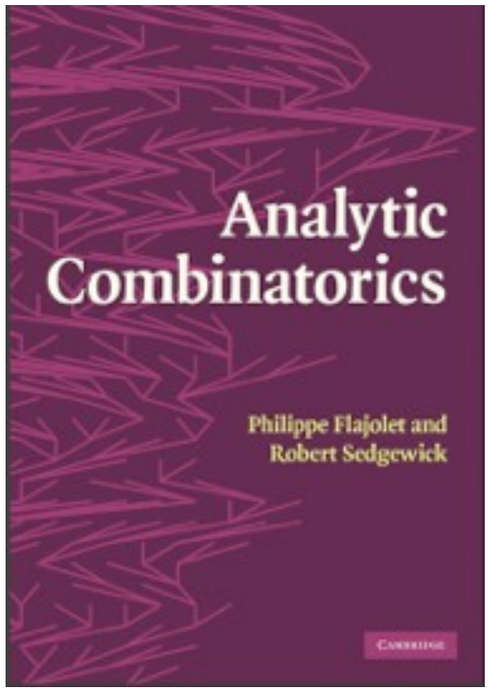
Implicit Species at the Basis of Analytic Combinatorics

Bruno Salvy
@ Inria & ENS-Lyon

Joint work with Carine Pivoteau & Michèle Soria

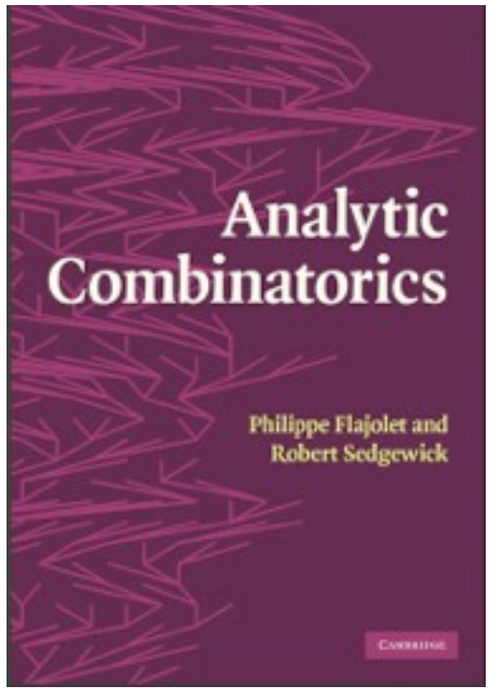
Analytic Combinatorics



Analytic Combinatorics

Approach:

1. Specify combinatorially;

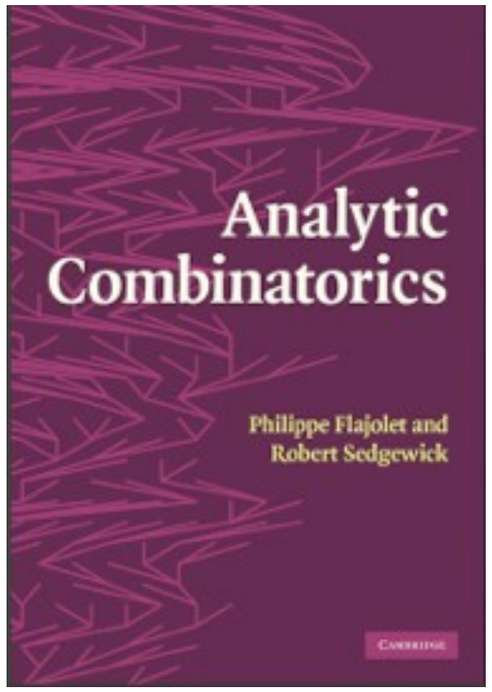


2. Everything follows

Analytic Combinatorics

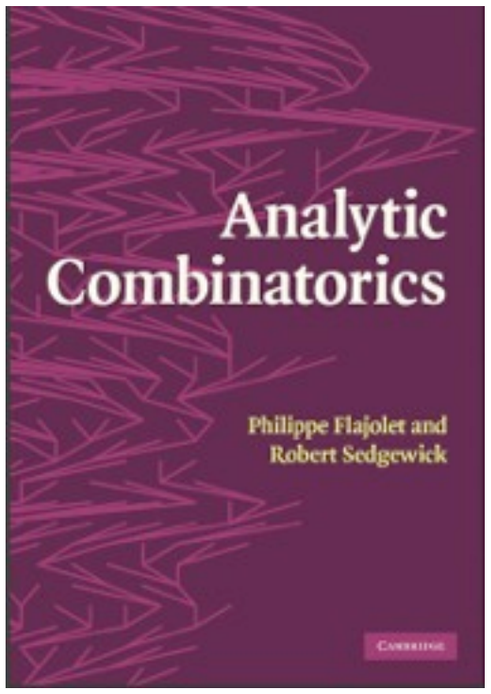
Approach:

1. Specify combinatorially;



2. Everything follows
   - generating series;
   - asymptotics;
   - limit laws.

Analytic Combinatorics

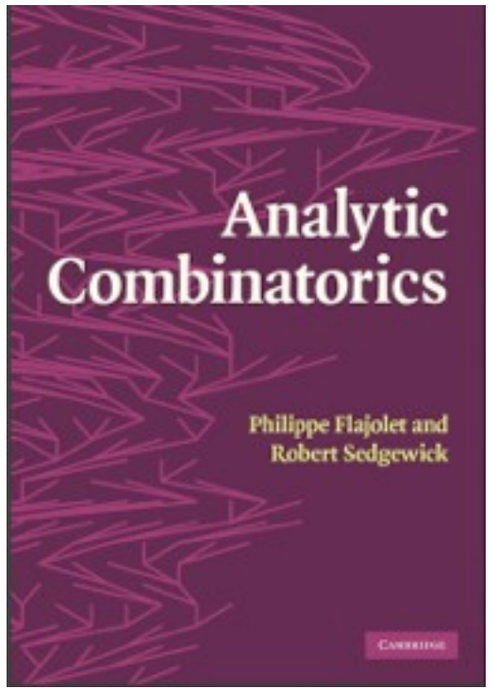


Approach:

1. Specify combinatorially;
2. Everything follows
   - generating series;
   - asymptotics;
   - limit laws.

Language: 1, $\mathcal{Z}$, +, ×, SEQ, SET, CYC and recursion.

Analytic Combinatorics



Approach:

1. Specify combinatorially;

1bis. *Check spec is well-founded;*

2. Everything follows
   - generating series;
   - asymptotics;
   - limit laws.

Language: $1, \mathcal{Z}, +, \times,$ SEQ, SET, CYC and recursion.

What is a good specification?

What is a good specification?

$\mathcal{B}=\mathcal{Z}+\mathcal{Z}\cdot\mathcal{B}\cdot\mathcal{B}$

$\mathcal{T}=\mathcal{Z}\cdot\mathrm{SET}(\mathcal{T}(\mathcal{Z}))$

$\mathcal{A}=\mathcal{A}$

$\mathcal{A}=\mathcal{Z}\cdot\mathcal{A}$

$\mathcal{A}=\mathcal{A}+\mathcal{Z}\cdot\mathcal{A}$

$\mathcal{A}=1+\mathcal{Z}\cdot\mathcal{A}, \mathcal{C}=1+\mathcal{A}\cdot\mathcal{A}$

What is a good specification?

✔ $\mathcal{B}=\mathcal{Z}+\mathcal{Z}\cdot\mathcal{B}\cdot\mathcal{B}$

$\mathcal{A}=\mathcal{Z}\cdot\mathcal{A}$

$\mathcal{T}=\mathcal{Z}\cdot\mathrm{SET}(\mathcal{T}(\mathcal{Z}))$

$\mathcal{A}=\mathcal{A}+\mathcal{Z}\cdot\mathcal{A}$

$\mathcal{A}=\mathcal{A}$

$\mathcal{A}=1+\mathcal{Z}\cdot\mathcal{A}, \mathcal{C}=1+\mathcal{A}\cdot\mathcal{A}$

What is a good specification?

✔ $\mathcal{B}=\mathcal{Z}+\mathcal{Z}\cdot\mathcal{B}\cdot\mathcal{B}$ $\quad\quad$ $\mathcal{A}=\mathcal{Z}\cdot\mathcal{A}$

✔ $\mathcal{T}=\mathcal{Z}\cdot\mathrm{SET}(\mathcal{T}(\mathcal{Z}))$ $\quad\quad$ $\mathcal{A}=\mathcal{A}+\mathcal{Z}\cdot\mathcal{A}$

$\mathcal{A}=\mathcal{A}$ $\quad\quad$ $\mathcal{A}=1+\mathcal{Z}\cdot\mathcal{A}, \mathcal{C}=1+\mathcal{A}\cdot\mathcal{A}$

What is a good specification?

✔ $\mathcal{B}=\mathcal{Z}+\mathcal{Z}\cdot\mathcal{B}\cdot\mathcal{B}$ $\mathcal{A}=\mathcal{Z}\cdot\mathcal{A}$

✔ $\mathcal{T}=\mathcal{Z}\cdot\mathrm{SET}(\mathcal{T}(\mathcal{Z}))$ $\mathcal{A}=\mathcal{A}+\mathcal{Z}\cdot\mathcal{A}$

✘ $\mathcal{A}=\mathcal{A}$ $\mathcal{A}=1+\mathcal{Z}\cdot\mathcal{A}, \mathcal{C}=1+\mathcal{A}\cdot\mathcal{A}$

What is a good specification?

✔ $\mathcal{B}=\mathcal{Z}+\mathcal{Z}\cdot\mathcal{B}\cdot\mathcal{B}$

✔ $\mathcal{T}=\mathcal{Z}\cdot\mathrm{SET}(\mathcal{T}(\mathcal{Z}))$

✘ $\mathcal{A}=\mathcal{A}$

✘ $\mathcal{A}=\mathcal{Z}\cdot\mathcal{A}$

✘ $\mathcal{A}=\mathcal{A}+\mathcal{Z}\cdot\mathcal{A}$

✘ $\mathcal{A}=1+\mathcal{Z}\cdot\mathcal{A}, \mathcal{C}=1+\mathcal{A}\cdot\mathcal{A}$

What is a good specification?

✔ $\mathcal{B}=\mathcal{Z}+\mathcal{Z}\cdot\mathcal{B}\cdot\mathcal{B}$

✔ $\mathcal{T}=\mathcal{Z}\cdot\mathrm{SET}(\mathcal{T}(\mathcal{Z}))$

✘ $\mathcal{A}=\mathcal{A}$

✘ $\mathcal{A}=\mathcal{Z}\cdot\mathcal{A}$

✘ $\mathcal{A}=\mathcal{A}+\mathcal{Z}\cdot\mathcal{A}$

$\mathcal{A}=1+\mathcal{Z}\cdot\mathcal{A}, \mathcal{C}=1+\mathcal{A}\cdot\mathcal{A}$

What is a good specification?

✔ $\mathcal{B}=\mathcal{Z}+\mathcal{Z}\cdot\mathcal{B}\cdot\mathcal{B}$

✔ $\mathcal{T}=\mathcal{Z}\cdot\mathrm{SET}(\mathcal{T}(\mathcal{Z}))$

✘ $\mathcal{A}=\mathcal{A}$

✘ $\mathcal{A}=\mathcal{Z}\cdot\mathcal{A}$

✘ $\mathcal{A}=\mathcal{A}+\mathcal{Z}\cdot\mathcal{A}$

✔ $\mathcal{A}=1+\mathcal{Z}\cdot\mathcal{A}, \mathcal{C}=1+\mathcal{A}\cdot\mathcal{A}$

What is a good specification?

✔ $\mathcal{B}=\mathcal{Z}+\mathcal{Z}\cdot\mathcal{B}\cdot\mathcal{B}$

✔ $\mathcal{T}=\mathcal{Z}\cdot\mathrm{SET}(\mathcal{T}(\mathcal{Z}))$

✘ $\mathcal{A}=\mathcal{A}$

✘ $\mathcal{A}=\mathcal{Z}\cdot\mathcal{A}$

✘ $\mathcal{A}=\mathcal{A}+\mathcal{Z}\cdot\mathcal{A}$

✔ $\mathcal{A}=1+\mathcal{Z}\cdot\mathcal{A}, \mathcal{C}=1+\mathcal{A}\cdot\mathcal{A}$

$\mathcal{G}=\mathcal{Z}+\mathcal{S}+\mathcal{P}, \mathcal{S}=\mathrm{SEQ}(\mathcal{Z}+\mathcal{P}),\mathcal{P}=\mathrm{SET}_{>0}(\mathcal{Z}+\mathcal{S})$

$\mathcal{G}=\mathcal{Z}+\mathcal{S}+\mathcal{P}, \mathcal{S}=\mathrm{SEQ}_{>0}(\mathcal{Z}+\mathcal{P}),\mathcal{P}=\mathrm{SET}_{>0}(\mathcal{Z}+\mathcal{S})$

What is a good specification?

✔ $\mathcal{B}=\mathcal{Z}+\mathcal{Z}\cdot\mathcal{B}\cdot\mathcal{B}$

✔ $\mathcal{T}=\mathcal{Z}\cdot\mathrm{SET}(\mathcal{T}(\mathcal{Z}))$

✘ $\mathcal{A}=\mathcal{A}$

✘ $\mathcal{A}=\mathcal{Z}\cdot\mathcal{A}$

✘ $\mathcal{A}=\mathcal{A}+\mathcal{Z}\cdot\mathcal{A}$

✔ $\mathcal{A}=1+\mathcal{Z}\cdot\mathcal{A}, \mathcal{C}=1+\mathcal{A}\cdot\mathcal{A}$

✘ $\mathcal{G}=\mathcal{Z}+\mathcal{S}+\mathcal{P}, \mathcal{S}=\mathrm{SEQ}(\mathcal{Z}+\mathcal{P}),\mathcal{P}=\mathrm{SET}_{>0}(\mathcal{Z}+\mathcal{S})$

✔ $\mathcal{G}=\mathcal{Z}+\mathcal{S}+\mathcal{P}, \mathcal{S}=\mathrm{SEQ}_{>0}(\mathcal{Z}+\mathcal{P}),\mathcal{P}=\mathrm{SET}_{>0}(\mathcal{Z}+\mathcal{S})$

What is a good specification?

✔

✔

✘

```
> N1 = Cycle(Union(N16,Prod(N16,Prod(Prod(Z,Cycle(N16)),N7))))
  N2 = Prod(N8,N8)
  N3 = Union(Cycle(Prod(Z,Prod(N2,Prod(Set(Z),Union(N14,Z))))),Sequence(Prod(Z,Cycle(N14))))
  N4 = Prod(Set(Prod(Z,Sequence(Prod(N8,Z)))),N8)
  N5 = Set(Union(Prod(Z,Sequence(Prod(N11,Sequence(Union(Z,Sequence(N11)))))),Prod(N11,Sequence(N11))))
  N6 = Union(Cycle(Prod(N11,N4)),Cycle(Union(N11,Union(N11,Z))))
  N7 = Cycle(Prod(N15,Prod(Prod(Sequence(N15),Z),N15)))
  N8 = Union(N8,Prod(Sequence(Prod(Prod(Prod(Z,Z),Union(N8,N8)),Union(N8,N8))),N8))
  N9 = Set(Prod(Prod(Union(N5,Cycle(Z)),Z),Z))
  N10 = Prod(Prod(Cycle(Union(Z,Z)),N10),Union(Cycle(N13),Cycle(Prod(Prod(N13,Cycle(N13)),N10))))
  N11 = Prod(Union(Set(Prod(Sequence(Z),Prod(N12,N5))),N5),Z)
  N12 = Prod(N19,Cycle(Prod(N19,Prod(Z,Prod(Prod(Prod(Sequence(Z),Z),N1),Prod(N19,N19))))))
  N13 = Prod(Sequence(N17),Union(N17,Union(Prod(N17,Sequence(Prod(Prod(N17,N17),N17))),Z)))
  N14 = Prod(Prod(Prod(Prod(Z,N13),Z),Z),Cycle(Union(Z,Prod(Z,Cycle(N13)))))
  N15 = Prod(Prod(N2,Union(Z,Z)),Z)
  N16 = Prod(Prod(Prod(Prod(Cycle(Z),Sequence(Prod(Set(Z),Union(N15,Z)))),Z),Z),Set(Prod(Z,N10)))
  N17 = Prod(Prod(Sequence(Z),Prod(N9,Union(N17,Z))),Z)
  N18 = Union(Union(Z,Prod(Cycle(N20),Z)),Prod(Prod(N20,Union(N20,Union(Union(N20,Z),Z))),Set(N20)))
  N19 = Union(Prod(N11,Prod(Sequence(N11),Sequence(Z))),Prod(N4,Prod(Set(Prod(Z,N11)),Z)))
  N20 = Prod(Union(N19,Prod(N19,Cycle(N19))),Z)
```

What is a good specification?

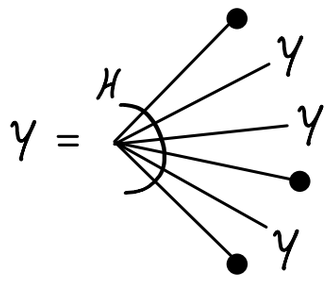
✔

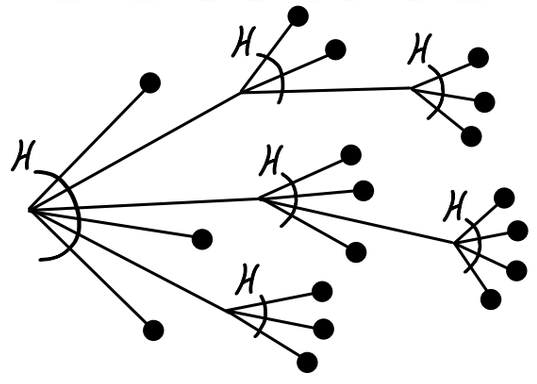
✔

✘

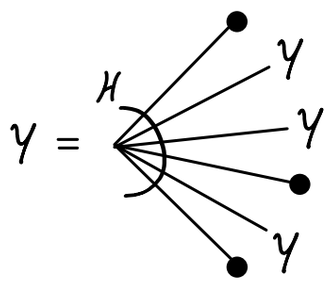
```
> N1 = Cycle(Union(N16,Prod(N16,Prod(Prod(Z,Cycle(N16)),N7))))
  N2 = Prod(N8,N8)
  N3 = Union(Cycle(Prod(Z,Prod(N2,Prod(Set(Z),Union(N14,Z))))),Sequence(Prod(Z,Cycle(N14))))
  N4 = Prod(Set(Prod(Z,Sequence(Prod(N8,Z)))),N8)
  N5 = Set(Union(Prod(Z,Sequence(Prod(N11,Sequence(Union(Z,Sequence(N11)))))),Prod(N11,Sequence(N11))))
  N6 = Union(Cycle(Prod(N11,N4)),Cycle(Union(N11,Union(N11,Z))))
  N7 = Cycle(Prod(N15,Prod(Prod(Sequence(N15),Z),N15)))
  N8 = Union(N8,Prod(Sequence(Prod(Prod(Prod(Z,Z),Union(N8,N8)),Union(N8,N8))),N8))
  N9 = Set(Prod(Prod(Union(N5,Cycle(Z)),Z),Z))
  N10 = Prod(Prod(Cycle(Union(Z,Z)),N10),Union(Cycle(N1?),Cycle(Prod(Prod(N13,Cycle(N13)),N10))))
  N11 = Prod(Union(Set(Prod(Sequence(Z),Prod(N12,N5))),N5),Z)
  N12 = Prod(N19,Cycle(Prod(N19,Prod(Z,Prod(Prod(Prod(Sequence(Z),Z),N1),Prod(N19,N19))))))
  N13 = Prod(Sequence(N17),Union(N17,Union(Prod(N17,Sequence(Prod(Prod(N17,N17),N17))),Z)))
  N14 = Prod(Prod(Prod(Prod(Z,N13),Z),Z),Cycle(Union(Z,Prod(Z,Cycle(N13)))))
  N15 = Prod(Prod(N2,Union(Z,Z)),Z)
  N16 = Prod(Prod(Prod(Prod(Cycle(Z),Sequence(Prod(Set(Z),Union(N15,Z)))),Z),Z),Set(Prod(Z,N10)))
  N17 = Prod(Prod(Sequence(Z),Prod(N9,Union(N17,Z))),Z)
  N18 = Union(Union(Z,Prod(Cycle(N20),Z)),Prod(Prod(N20,Union(N20,Union(Union(N20,Z),Z))),Set(N20)))
  N19 = Union(Prod(N11,Prod(Sequence(N11),Sequence(Z))),Prod(N4,Prod(Set(Prod(Z,N11)),Z)))
  N20 = Prod(Union(N19,Prod(N19,Cycle(N19))),Z)
```

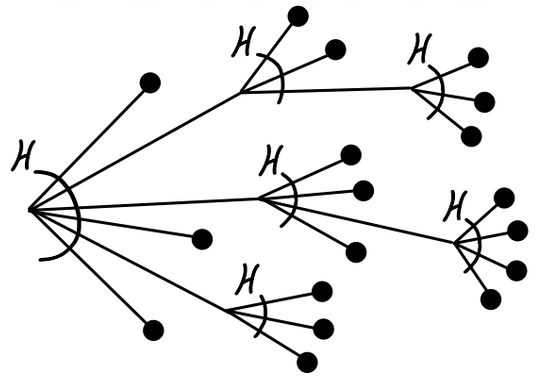
Previous work & contribution

From 

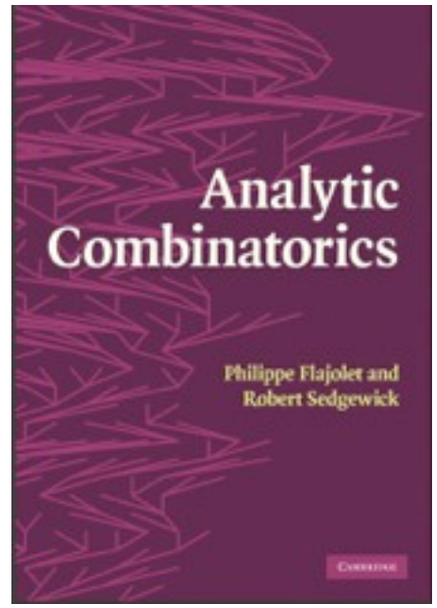
to 

Previous work & contribution

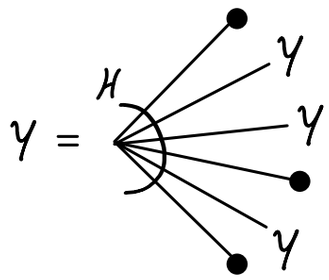
From 

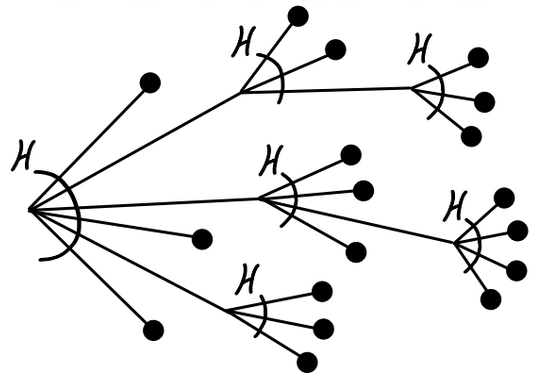
to 

- AC: sufficient condition for the Drmota-Lalley-Woods theorem (H is a contraction over power series);



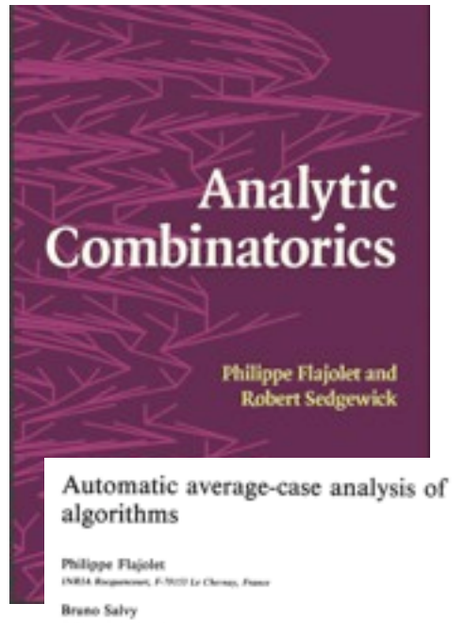
Previous work & contribution

From 

to 

- AC:
  sufficient condition for the Drmota-Lalley-Woods theorem (H is a contraction over power series);

- Flajolet-S-Zimmermann 1991:
  a defn and an algo (computation of valuations), pointer to P. Zimmermann's thesis for full algorithm;




Automatic average-case analysis of algorithms

Philippe Flajolet
INRIA Rocquencourt, F-78153 Le Chesnay, France

Bruno Salvy
INRIA and LIX, Ecole Polytechnique, F-91128 Palaiseau, France

Paul Zimmermann
INRIA Rocquencourt, F-78153 Le Chesnay, France

Abstract

Flajolet, P., Salvy, B. and Zimmermann, P., Automatic average-case analysis of algorithms, Theoretical Computer Science 79 (1991) 37–109.

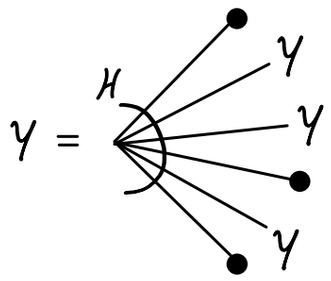
Many probabilistic properties of elementary discrete combinatorial structures of interest for the average-case analysis of algorithms prove to be decidable. This paper presents a general framework in which such decision procedures can be developed. It is based on a combination of generating function techniques for counting, and complex analysis techniques for asymptotic estimations.

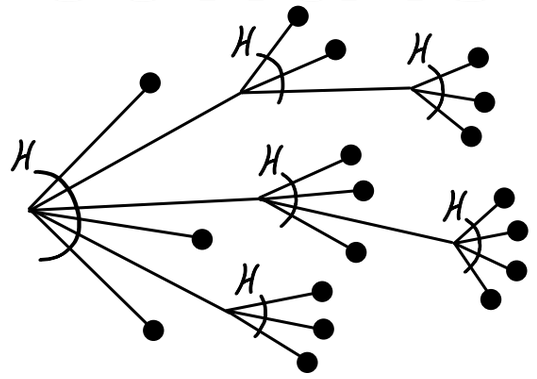
We express here the theory of exact analysis in terms of generating functions for four different domains: the iterative/recursive and unlabelled/labelled data type domains. We then present some major components of the associated asymptotic theory and exhibit a class of naturally arising functions that can be automatically analyzed.

A Sub-Fragment of this theory is also incorporated into a system called Lambda-Upsilon-Omega. In this way, using computer algebra, one can produce automatically non-trivial average-case analyses of algorithms operating over a variety of "decomposable" combinatorial structures.

At a fundamental level, this paper is part of a global attempt at understanding why so many elementary combinatorial problems tend to have elementary asymptotic solutions. In several cases, it proves possible to relate entire classes of elementary combinatorial problems whose structure is well defined with classes of elementary "special" functions and classes of asymptotic forms relative to counting, probabilities, or average-case complexity.

Previous work & contribution

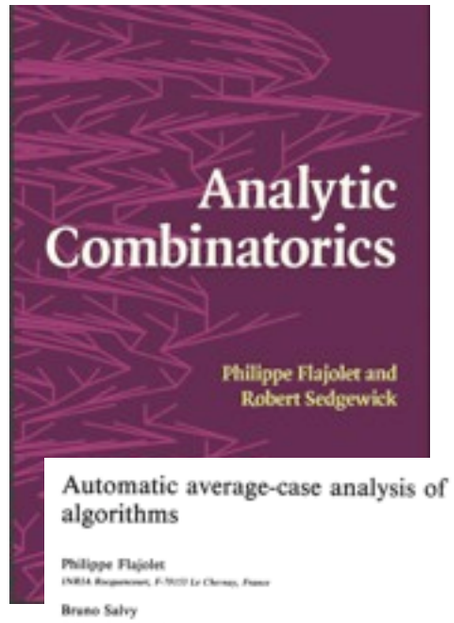
From 

to 

- AC:
  sufficient condition for the Drmota-Lalley-Woods theorem (H is a contraction over power series);

- Flajolet-S-Zimmermann 1991:
  a defn and an algo (computation of valuations), pointer to P. Zimmermann's thesis for full algorithm;

- Joyal 1981:
  a sufficient condition within species theory:
  $H(0,0)=0$ and $\partial H/\partial \mathscr{Y}(0,0)$ nilpotent,
  easily turned into a simple algorithm;




Automatic average-case analysis of algorithms

Philippe Flajolet

Bruno Salvy

Paul Zimmermann

Abstract

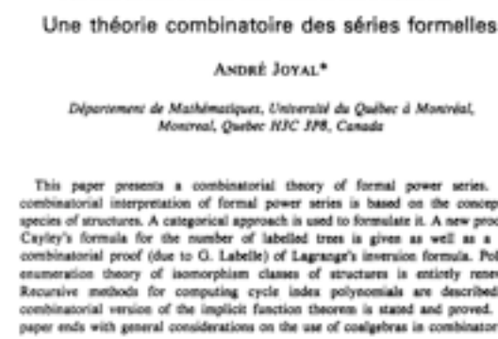
Flajolet, P., Salvy, B. and Zimmermann, P., Automatic average-case analysis of algorithms, Theoretical Computer Science 79 (1991) 37–109.

Many probabilistic properties of elementary discrete combinatorial structures of interest for the average-case analysis of algorithms prove to be decidable. This paper presents a general framework in which such decision procedures can be developed. It is based on a combination of generating function techniques for counting, and complex analysis techniques for asymptotic estimations.

We express here the theory of exact analysis in terms of generating functions for four different domains: the iterative/recursive and unlabelled/labelled data type domains. We then present some major components of the associated asymptotic theory and exhibit a class of naturally arising functions that can be automatically analyzed.

A Sub Fragment of this theory is also incorporated into a system called Lambda-Upsilon-Omega. In this way, using computer algebra, one can produce automatically non-trivial average-case analyses of algorithms operating over a variety of "decomposable" combinatorial structures.

At a fundamental level, this paper is part of a global attempt at understanding why so many elementary combinatorial problems tend to have elementary asymptotic solutions. In several cases, it proves possible to relate entire classes of elementary combinatorial problems whose structure

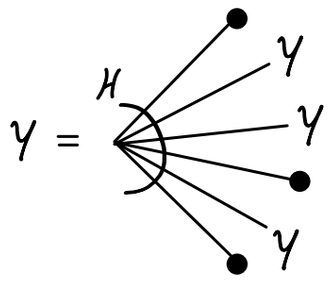
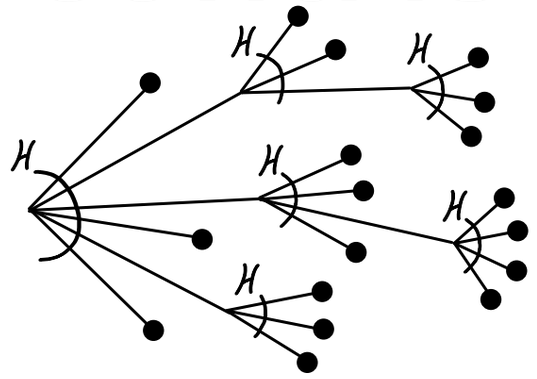

Une théorie combinatoire des séries formelles

ANDRÉ JOYAL*

Département de Mathématiques, Université du Québec à Montréal, Montréal, Québec H3C 3P8, Canada

This paper presents a combinatorial theory of formal power series. combinatorial interpretation of formal power series is based on the concep species of structures. A categorical approach is used to formulate it. A new proc Cayley's formula for the number of labelled trees is given as well as a combinatorial proof (due to G. Labelle) of Lagrange's inversion formula. Pol enumeration theory of isomorphism classes of structures is entirely rene Recursive methods for computing cycle index polynomials are described combinatorial version of the implicit function theorem is stated and proved. paper ends with general considerations on the use of coalgebras in combinator

Previous work & contribution

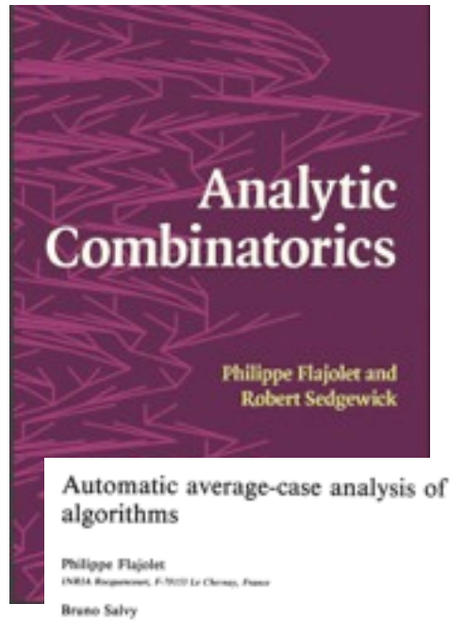
From  to 

- AC:
  sufficient condition for the Drmota-Lalley-Woods theorem (H is a contraction over power series);

- Flajolet-S-Zimmermann 1991:
  a defn and an algo (computation of valuations), pointer to P. Zimmermann's thesis for full algorithm;

- Joyal 1981:
  a sufficient condition within species theory:
  $H(0,0)=0$ and $\partial H/\partial \mathscr{Y}(0,0)$ nilpotent,
  easily turned into a simple algorithm;

- Pivoteau-S-Soria 2012 (this talk):
  necessary and sufficient condition, simple algorithms.




Automatic average-case analysis of algorithms

Philippe Flajolet

Bruno Salvy

Paul Zimmermann

Une théorie combinatoire des séries formelles

ANDRÉ JOYAL*

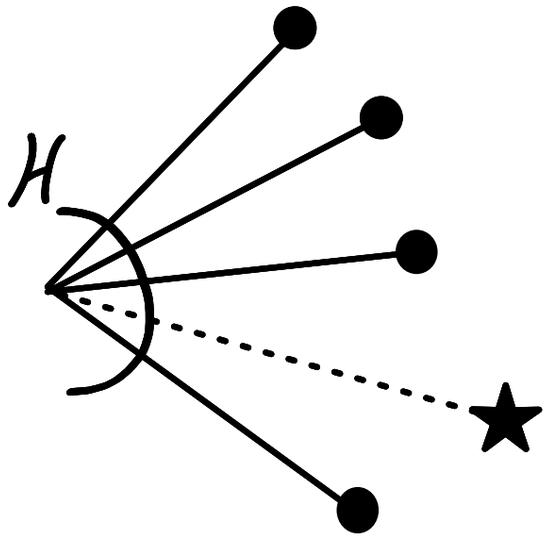
Département de Mathématiques, Université du Québec à Montréal, Montréal, Québec H3C 3P8, Canada

Journal of Combinatorial Theory, Series A

Algorithms for combinatorial structures: Well-founded systems and Newton iterations

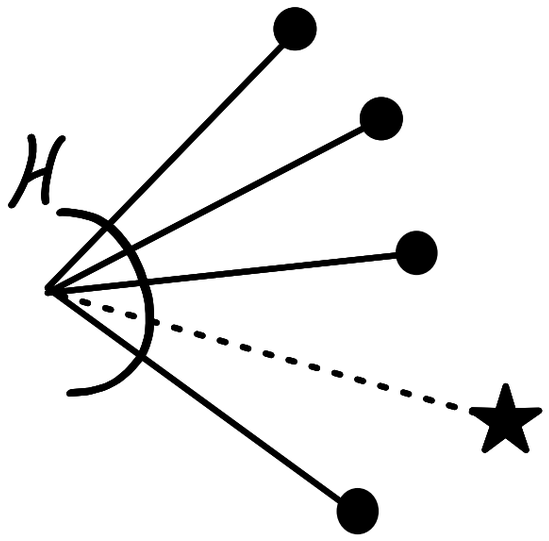
Carine Pivoteau, Bruno Salvy, Michèle Soria

Derivative



$$\mathcal{H}'[U] = \mathcal{H}[U + \{\star\}]$$

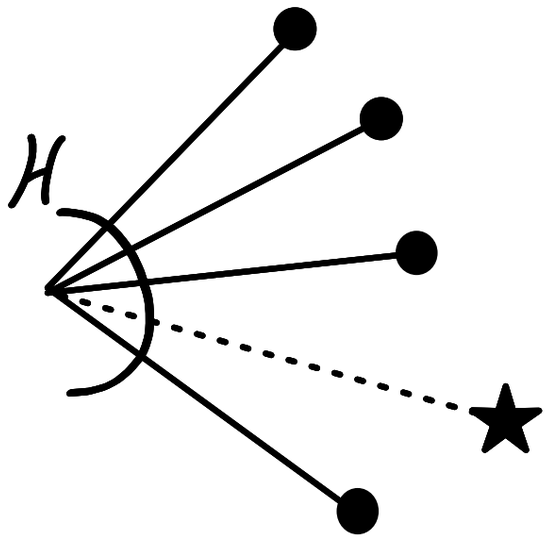
Derivative



$$\mathcal{H}'[U] = \mathcal{H}[U + \{\star\}]$$

- 0’=1’=0; Z’=1; SET’=SET; CYC’=SEQ; SEQ’=SEQ·SEQ;

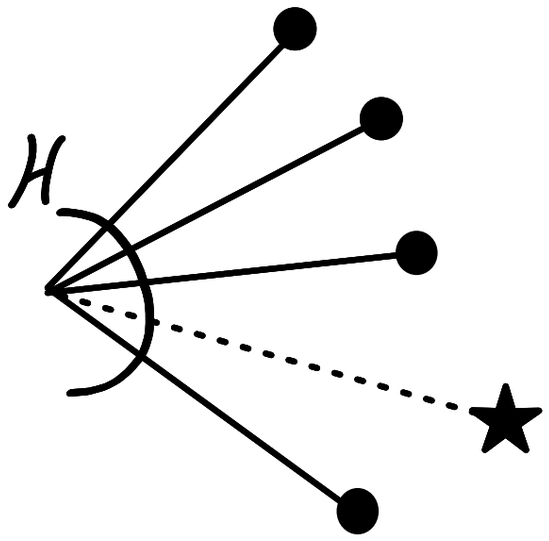
Derivative



$$\mathcal{H}'[U] = \mathcal{H}[U + \{\star\}]$$

- 0’=1’=0; Z’=1; SET’=SET; CYC’=SEQ; SEQ’=SEQ·SEQ;
- $(\mathscr{F}+\mathscr{G})' = \mathscr{F}' + \mathscr{G}'$; $(\mathscr{F}\cdot\mathscr{G})' = \mathscr{F}'\cdot\mathscr{G} + \mathscr{F}\cdot\mathscr{G}'$;

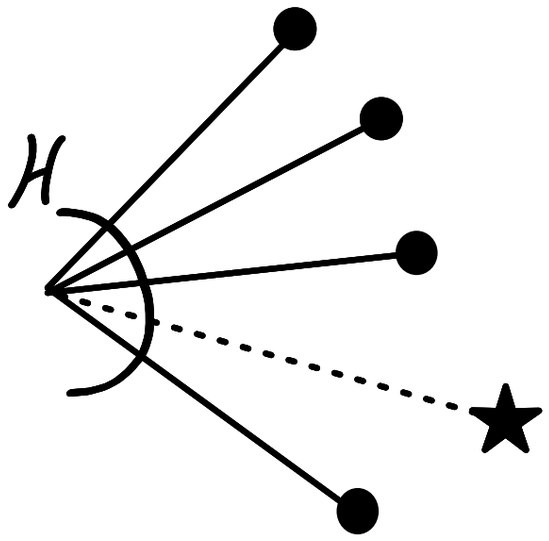
Derivative



$$\mathcal{H}'[U] = \mathcal{H}[U + \{\star\}]$$

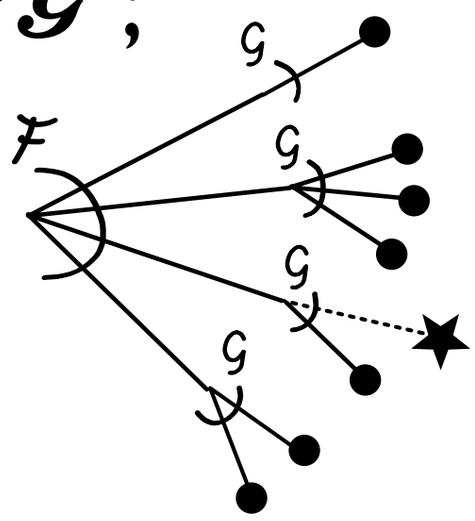
- 0’=1’=0; Z’=1; SET’=SET; CYC’=SEQ; SEQ’=SEQ·SEQ;
- $(\mathscr{F}+\mathscr{G})' = \mathscr{F}' + \mathscr{G}'$; $(\mathscr{F}\cdot\mathscr{G})' = \mathscr{F}'\cdot\mathscr{G} + \mathscr{F}\cdot\mathscr{G}'$;
- $\mathscr{F}(\mathscr{G})' = \mathscr{F}'(\mathscr{G})\cdot\mathscr{G}'$.

Derivative



$$\mathcal{H}'[U] = \mathcal{H}[U + \{\star\}]$$

- 0’=1’=0; Z’=1; SET’=SET; CYC’=SEQ; SEQ’=SEQ·SEQ;
- $(\mathscr{F}+\mathscr{G})' = \mathscr{F}' + \mathscr{G}'$; $(\mathscr{F}\cdot\mathscr{G})' = \mathscr{F}'\cdot\mathscr{G} + \mathscr{F}\cdot\mathscr{G}'$;
- $\mathscr{F}(\mathscr{G})' = \mathscr{F}'(\mathscr{G})\cdot\mathscr{G}'$.



Dependency Graphs

$$\boldsymbol{\mathcal{Y}} = \boldsymbol{\Phi}(\mathcal{Z}, \boldsymbol{\mathcal{Y}}) \quad : \quad \begin{cases} \mathcal{Y}_1 & = \Phi_1(\mathcal{Z}, \mathcal{Y}_1, \ldots, \mathcal{Y}_p) \\ & \vdots \\ \mathcal{Y}_p & = \Phi_p(\mathcal{Z}, \mathcal{Y}_1, \ldots, \mathcal{Y}_p) \end{cases}$$

- dep. graph of the grammar:
  - vertices: $\mathcal{Y}_1, \ldots, \mathcal{Y}_p$
  - edge $\mathcal{Y}_i \to \mathcal{Y}_j$ when $\partial \Phi_i / \partial \mathcal{Y}_j \neq 0$

Dependency Graphs

$$\mathcal{Y} = \Phi(\mathcal{Z}, \mathcal{Y}) \quad : \quad \begin{cases} \mathcal{Y}_1 & = \Phi_1(\mathcal{Z}, \mathcal{Y}_1, \ldots, \mathcal{Y}_p) \\ & \vdots \\ \mathcal{Y}_p & = \Phi_p(\mathcal{Z}, \mathcal{Y}_1, \ldots, \mathcal{Y}_p) \end{cases}$$

- dep. graph of the grammar:
  - vertices: $\mathcal{Y}_1, \ldots, \mathcal{Y}_p$
    - edge $\mathcal{Y}_i \to \mathcal{Y}_j$ when $\partial \Phi_i / \partial \mathcal{Y}_j \neq 0$

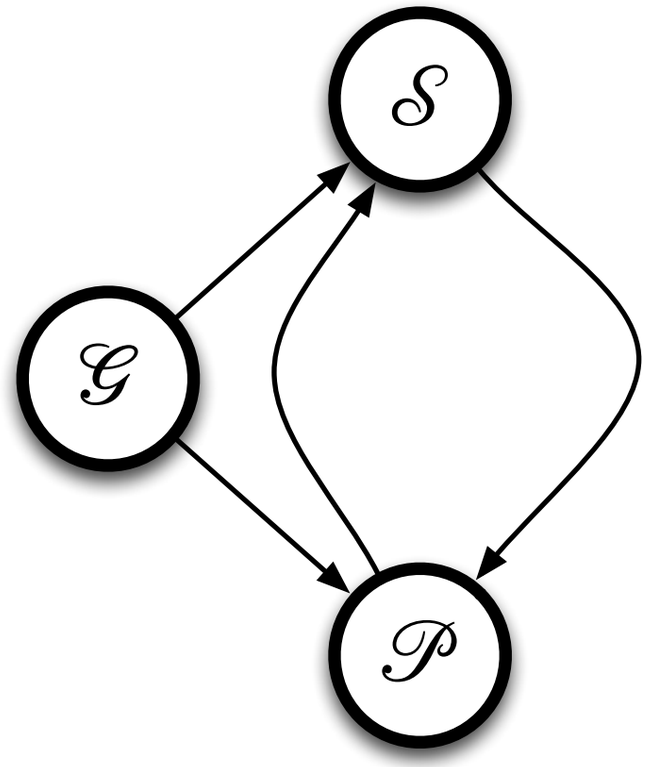
$$\begin{cases} \mathcal{G} = \mathcal{Z} + \mathcal{S} + \mathcal{P}, \\ \mathcal{S} = \textsc{Seq}_{>0}(\mathcal{Z} + \mathcal{P}), \\ \mathcal{P} = \textsc{Set}_{>1}(\mathcal{Z} + \mathcal{S}), \end{cases}$$

Dependency Graphs

$$\mathcal{Y} = \Phi(\mathcal{Z}, \mathcal{Y}) \quad : \quad \begin{cases} \mathcal{Y}_1 &= \Phi_1(\mathcal{Z}, \mathcal{Y}_1, \ldots, \mathcal{Y}_p) \\ &\vdots \\ \mathcal{Y}_p &= \Phi_p(\mathcal{Z}, \mathcal{Y}_1, \ldots, \mathcal{Y}_p) \end{cases}$$

- dep. graph of the grammar:
  - vertices: $\mathcal{Y}_1, \ldots, \mathcal{Y}_p$
  - **edge** $\mathcal{Y}_i \to \mathcal{Y}_j$ **when** $\partial \Phi_i / \partial \mathcal{Y}_j \neq 0$

$$\begin{cases} \mathcal{G} = \mathcal{Z} + \mathcal{S} + \mathcal{P}, \\ \mathcal{S} = \textsc{Seq}_{>0}(\mathcal{Z} + \mathcal{P}), \\ \mathcal{P} = \textsc{Set}_{>1}(\mathcal{Z} + \mathcal{S}), \end{cases}$$



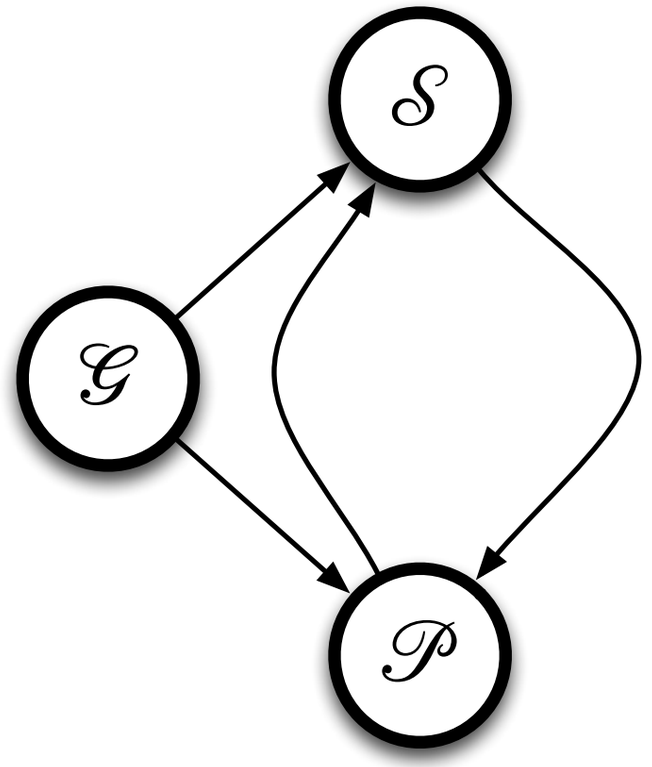
Dependency Graphs

$$\mathcal{Y} = \Phi(\mathcal{Z}, \mathcal{Y}) \quad : \quad \begin{cases} \mathcal{Y}_1 &= \Phi_1(\mathcal{Z}, \mathcal{Y}_1, \ldots, \mathcal{Y}_p) \\ &\vdots \\ \mathcal{Y}_p &= \Phi_p(\mathcal{Z}, \mathcal{Y}_1, \ldots, \mathcal{Y}_p) \end{cases}$$

- dep. graph of the grammar:
  - vertices: $\mathcal{Y}_1, \ldots, \mathcal{Y}_p$
  - **edge** $\mathcal{Y}_i \to \mathcal{Y}_j$ **when** $\partial \Phi_i / \partial \mathcal{Y}_j \neq 0$

$$\begin{cases} \mathcal{G} = \mathcal{Z} + \mathcal{S} + \mathcal{P}, \\ \mathcal{S} = \textsc{Seq}_{>0}(\mathcal{Z} + \mathcal{P}), \\ \mathcal{P} = \textsc{Set}_{>1}(\mathcal{Z} + \mathcal{S}), \end{cases}$$

$$\frac{\partial \mathbf{\Phi}}{\partial \mathcal{Y}} = \begin{pmatrix} 0 & 1 & 1 \\ 0 & 0 & \textsc{Seq}(\mathcal{Z} + \mathcal{P})^2 \\ 0 & \textsc{Set}_{>0}(\mathcal{Z} + \mathcal{S}) & 0 \end{pmatrix}$$



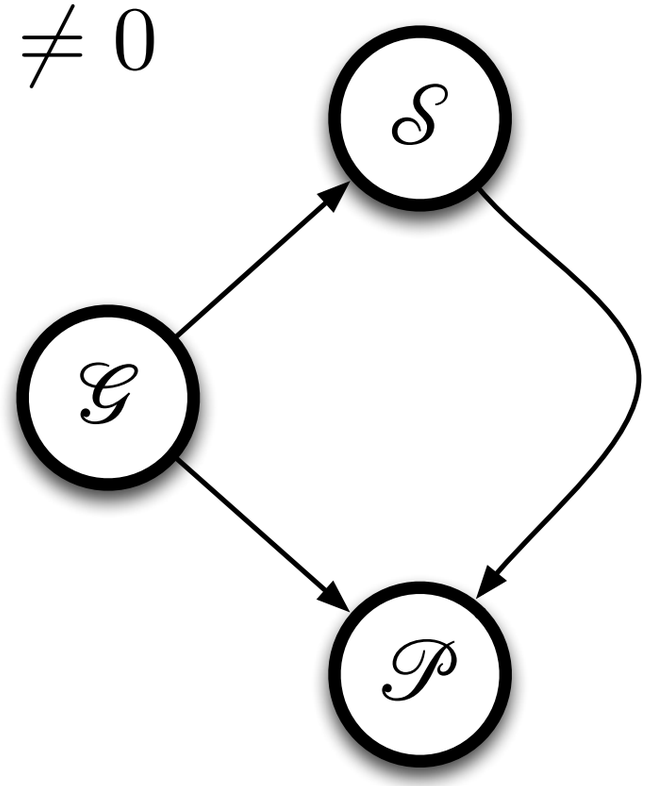
Dependency Graphs

$$\mathcal{Y} = \Phi(\mathcal{Z}, \mathcal{Y}) \quad : \quad \begin{cases} \mathcal{Y}_1 &= \Phi_1(\mathcal{Z}, \mathcal{Y}_1, \ldots, \mathcal{Y}_p) \\ &\vdots \\ \mathcal{Y}_p &= \Phi_p(\mathcal{Z}, \mathcal{Y}_1, \ldots, \mathcal{Y}_p) \end{cases}$$

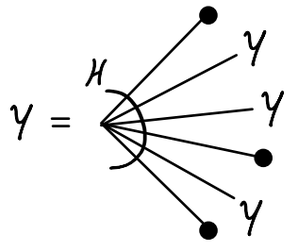
- dep. graph of the grammar at 0:
  - vertices: $\mathcal{Y}_1, \ldots, \mathcal{Y}_p$
  - edge $\mathcal{Y}_i \to \mathcal{Y}_j$ when $\partial \Phi_i / \partial \mathcal{Y}_j (0, \mathbf{0}) \neq 0$

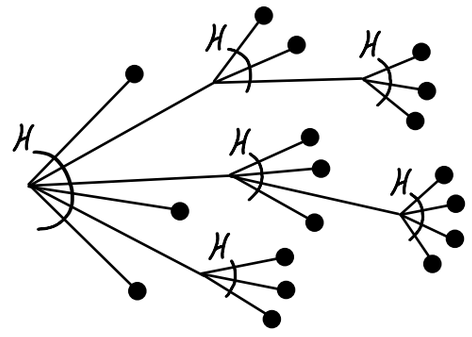
$$\begin{cases} \mathcal{G} = \mathcal{Z} + \mathcal{S} + \mathcal{P}, \\ \mathcal{S} = \textsc{Seq}_{>0}(\mathcal{Z} + \mathcal{P}), \\ \mathcal{P} = \textsc{Set}_{>1}(\mathcal{Z} + \mathcal{S}), \end{cases}$$

$$\frac{\partial \mathbf{\Phi}}{\partial \mathcal{Y}} = \begin{pmatrix} 0 & 1 & 1 \\ 0 & 0 & \textsc{Seq}(\mathcal{Z} + \mathcal{P})^2 \\ 0 & \textsc{Set}_{>0}(\mathcal{Z} + \mathcal{S}) & 0 \end{pmatrix}$$



A non-constructive definition





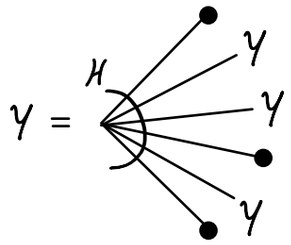
Def. $\mathcal{Y}=\mathcal{H}(\mathcal{X},\mathcal{Y})$ is *well-founded* when

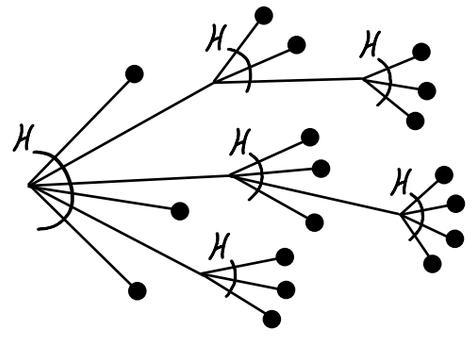
(a) the following iteration is convergent:

$$\mathcal{Y}^{[0]}=0,\quad \mathcal{Y}^{[n+1]}=\mathcal{H}(\mathcal{X},\mathcal{Y}^{[n]})\quad (n\geq 0)$$

(b) the limit does not have 0-coordinates.

A non-constructive definition





Def. $\mathcal{Y}=\mathcal{H}(\mathcal{X},\mathcal{Y})$ is *well-founded* when

(a) the following iteration is convergent:

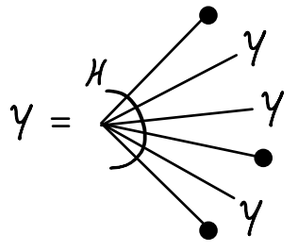
$$\mathcal{Y}^{[0]}=0,\quad \mathcal{Y}^{[n+1]}=\mathcal{H}(\mathcal{X},\mathcal{Y}^{[n]})\quad (n\geq 0)$$

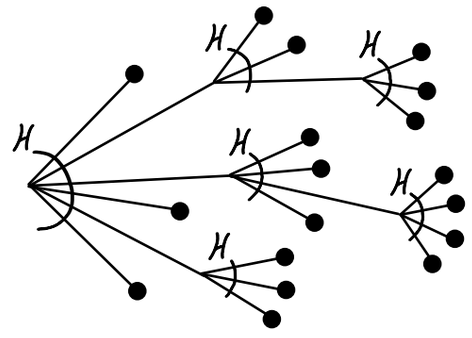
(b) the limit does not have 0-coordinates.

Joyal's Implicit Species Thm. When $\mathcal{H}(0,0)=0$,

$\partial\mathcal{H}/\partial\mathcal{Y}(0,0)$ nilpotent $\Longrightarrow$ (a).

A non-constructive definition





Def. $\mathcal{Y}=\mathcal{H}(\mathcal{X},\mathcal{Y})$ is *well-founded* when

(a) the following iteration is convergent:

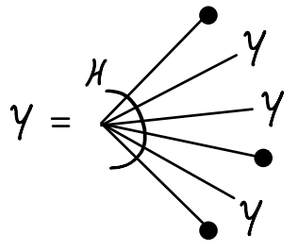
$$\mathcal{Y}^{[0]}=0,\quad \mathcal{Y}^{[n+1]}=\mathcal{H}(\mathcal{X},\mathcal{Y}^{[n]})\quad (n\geq 0)$$

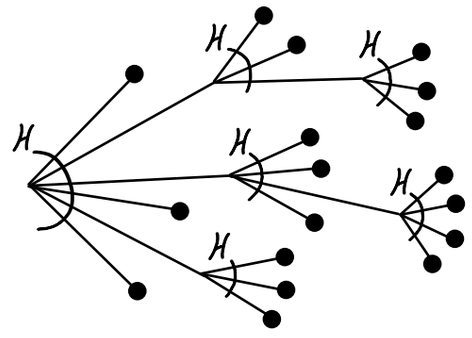
(b) the limit does not have 0-coordinates.

Joyal's Implicit Species Thm. When $\mathcal{H}(0,0)=0$,

$\partial\mathcal{H}/\partial\mathcal{Y}(0,0)$ nilpotent $\Longrightarrow$ (a).

A non-constructive definition





Def. $\mathcal{Y}=\mathcal{H}(\mathcal{Z},\mathcal{Y})$ is *well-founded* when

(a) the following iteration is convergent:

$$\mathcal{Y}^{[0]}=0,\quad \mathcal{Y}^{[n+1]}=\mathcal{H}(\mathcal{Z},\mathcal{Y}^{[n]})\quad (n\geq 0)$$

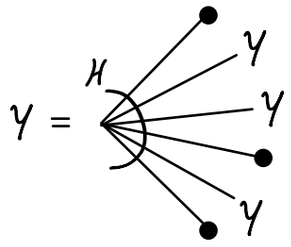
(b) the limit does not have 0-coordinates.

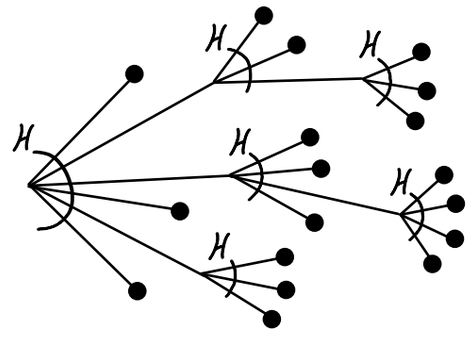
Joyal's Implicit Species Thm. When $\mathcal{H}(0,0)=0$,

$\partial\mathcal{H}/\partial\mathcal{Y}(0,0)$ nilpotent $\Longrightarrow$ (a).

The dependency graph at 0 has no cycle.

A non-constructive definition





Def. $\mathcal{Y}=\mathcal{H}(\mathcal{X},\mathcal{Y})$ is *well-founded* when

(a) the following iteration is convergent:

$$\mathcal{Y}^{[0]}=0,\quad \mathcal{Y}^{[n+1]}=\mathcal{H}(\mathcal{X},\mathcal{Y}^{[n]})\quad (n\geq 0)$$

(b) the limit does not have 0-coordinates.

Joyal's Implicit Species Thm. When $\mathcal{H}(0,0)=0$,

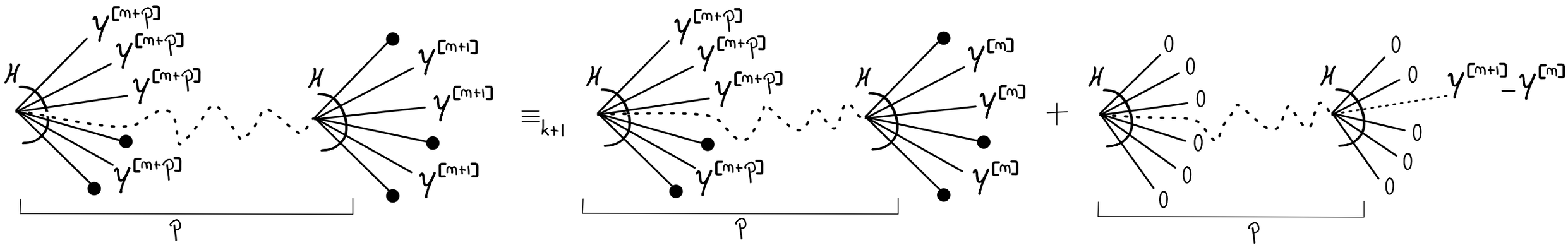
(a)+(b) $\overset{\text{new}}{\Longrightarrow}$ $\partial\mathcal{H}/\partial\mathcal{Y}(0,0)$ nilpotent $\Longrightarrow$ (a).

The dependency graph at 0 has no cycle.

Proof of Joyal's IST

Def. $\mathscr{A} =_k \mathscr{B}$ if they coincide up to size k (contact).

Key Lemma.
If $\mathscr{Y}^{[n+1]} =_k \mathscr{Y}^{[n]}$, then $\mathscr{Y}^{[n+p+1]} =_{k+1} \mathscr{Y}^{[n+p]}$ (p=dim).



0-coordinates

Cor. $\mathscr{Y}$ has a 0-coordinate $\iff \mathscr{Y}^{[p]}$ has a 0-coordinate.

0-coordinates

Cor. $\mathcal{Y}$ has a 0-coordinate $\iff$ $\mathcal{Y}^{[p]}$ has a 0-coordinate.

```
  Algo Well-founded at 0
1. Check cycles in dep.
graph at 0;
2. Check 0-coords in Y[p].
```

0-coordinates

Cor. $\mathcal{Y}$ has a 0-coordinate $\Longleftrightarrow \mathcal{Y}^{[p]}$ has a 0-coordinate.

```
  Algo Well-founded at 0
1. Check cycles in dep.
graph at 0;
2. Check 0-coords in Y[p].
```

Ex. $\mathcal{A}=\mathcal{Z}+\mathcal{B}+\mathrm{SEQ}(\mathcal{C})$, $\mathcal{B}=\mathcal{Z}\cdot\mathcal{D}+\mathcal{Z}\cdot\mathcal{B}\cdot\mathcal{B}$, $\mathcal{C}=\mathcal{Z}\cdot\mathrm{SEQ}(\mathcal{B})$, $\mathcal{D}=\mathcal{B}+\mathcal{Z}\cdot\mathcal{D}$.

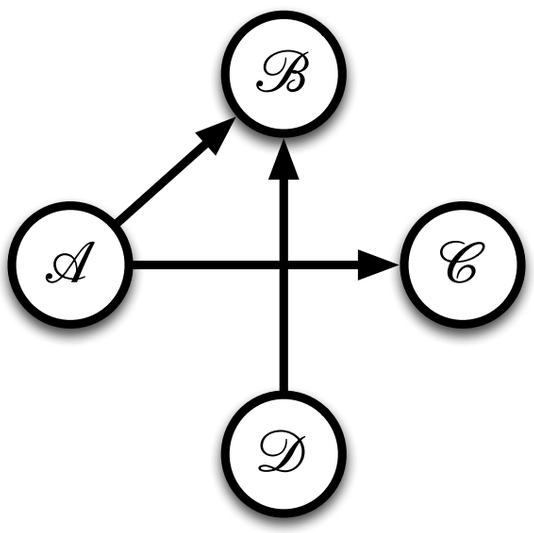
0-coordinates

Cor. $\mathcal{Y}$ has a 0-coordinate $\Longleftrightarrow \mathcal{Y}^{[p]}$ has a 0-coordinate.

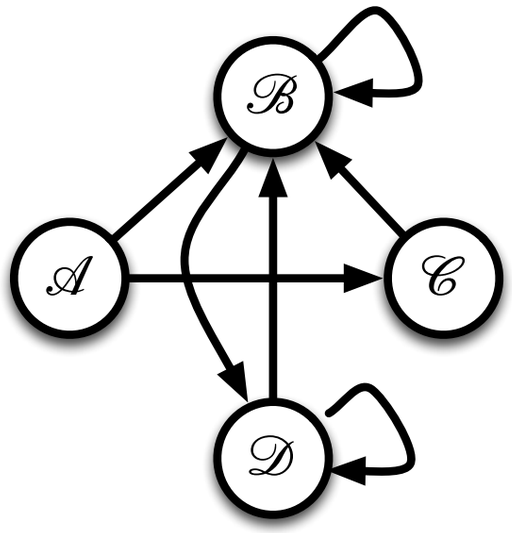
```
  Algo Well-founded at 0
1. Check cycles in dep.
graph at 0;
2. Check 0-coords in Y[p].
```

Ex. $\mathcal{A}=\mathcal{Z}+\mathcal{B}+\mathrm{SEQ}(\mathcal{C})$, $\mathcal{B}=\mathcal{Z}\cdot\mathcal{D}+\mathcal{Z}\cdot\mathcal{B}\cdot\mathcal{B}$, $\mathcal{C}=\mathcal{Z}\cdot\mathrm{SEQ}(\mathcal{B})$, $\mathcal{D}=\mathcal{B}+\mathcal{Z}\cdot\mathcal{D}$.

dep. graph at 0



dep. graph



0-coordinates

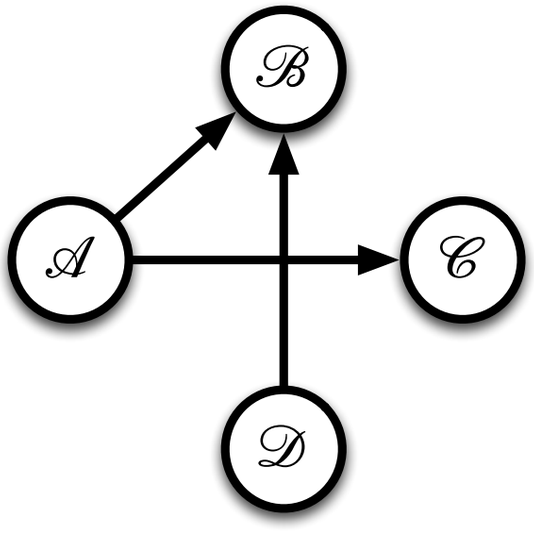
Cor. $\mathcal{Y}$ has a 0-coordinate $\Longleftrightarrow \mathcal{Y}^{[p]}$ has a 0-coordinate.

Algo Well-founded at 0
1. Check cycles in dep. graph at 0;
2. Check 0-coords in $Y^{[p]}$.

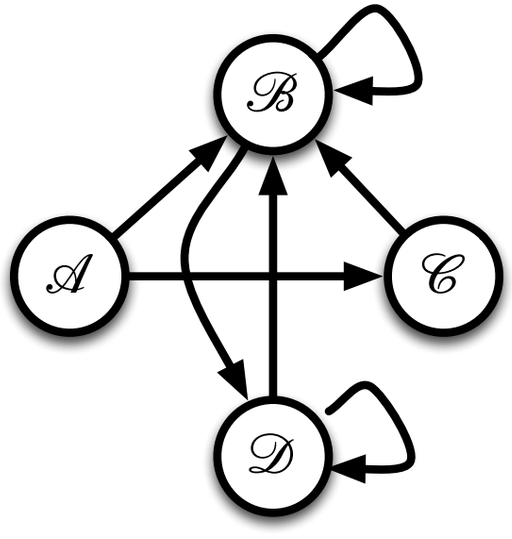
Algo Check 0-coords
1. Initialize where $H(Z,0)\neq 0$;
2. Propagate p times reversing the arrows in dep. graph;
3. Check empty nodes.

Ex. $\mathcal{A}=\mathcal{Z}+\mathcal{B}+\mathrm{SEQ}(\mathcal{C})$, $\mathcal{B}=\mathcal{Z}\cdot\mathcal{D}+\mathcal{Z}\cdot\mathcal{B}\cdot\mathcal{B}$, $\mathcal{C}=\mathcal{Z}\cdot\mathrm{SEQ}(\mathcal{B})$, $\mathcal{D}=\mathcal{B}+\mathcal{Z}\cdot\mathcal{D}$.

dep. graph at 0



dep. graph



0-coordinates

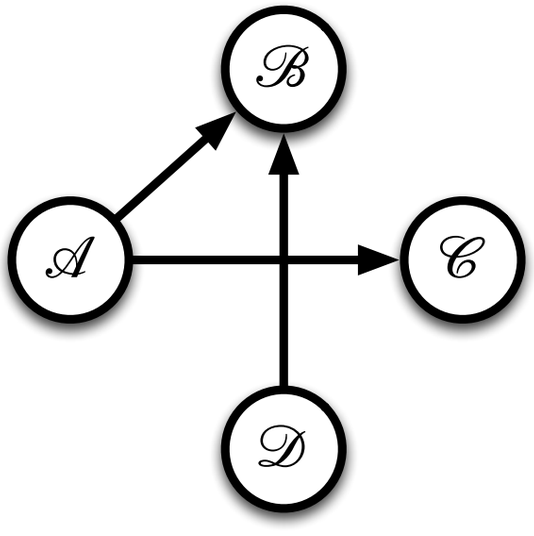
Cor. $\mathcal{Y}$ has a 0-coordinate $\Longleftrightarrow \mathcal{Y}^{[p]}$ has a 0-coordinate.

Algo Well-founded at 0
1. Check cycles in dep. graph at 0;
2. Check 0-coords in $Y^{[p]}$.

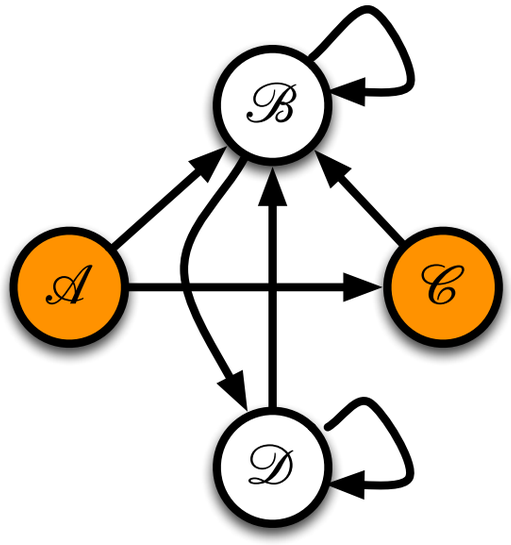
Algo Check 0-coords
1. Initialize where $H(Z,0)\neq 0$;
2. Propagate p times reversing the arrows in dep. graph;
3. Check empty nodes.

Ex. $\mathcal{A}=\mathcal{Z}+\mathcal{B}+\mathrm{SEQ}(\mathcal{C})$, $\mathcal{B}=\mathcal{Z}\cdot\mathcal{D}+\mathcal{Z}\cdot\mathcal{B}\cdot\mathcal{B}$, $\mathcal{C}=\mathcal{Z}\cdot\mathrm{SEQ}(\mathcal{B})$, $\mathcal{D}=\mathcal{B}+\mathcal{Z}\cdot\mathcal{D}$.

dep. graph at 0



dep. graph



0-coordinates

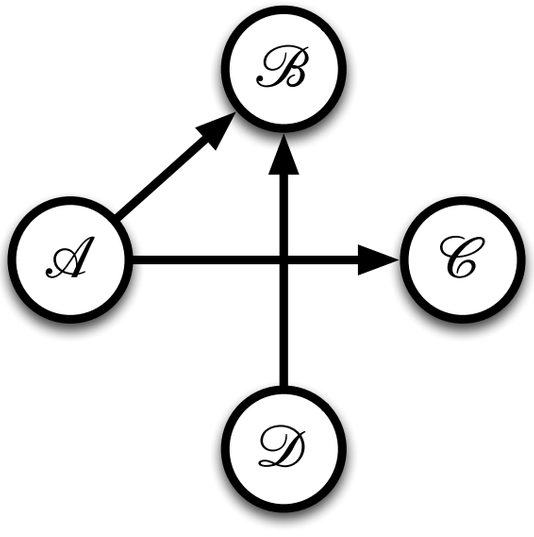
Cor. $\mathcal{Y}$ has a 0-coordinate $\Longleftrightarrow \mathcal{Y}^{[p]}$ has a 0-coordinate.

Algo Well-founded at 0
1. Check cycles in dep. graph at 0;
2. Check 0-coords in $Y^{[p]}$.

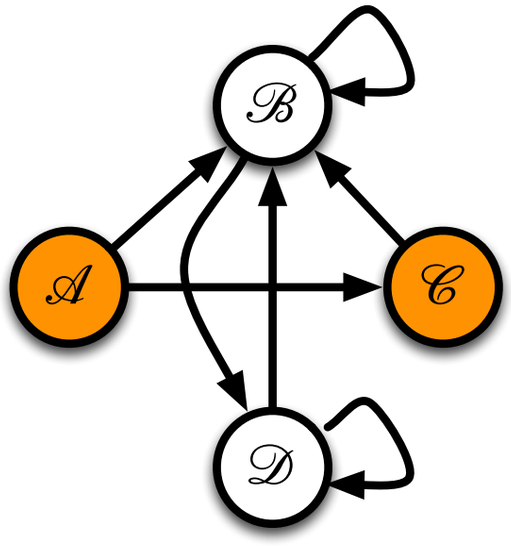
Algo Check 0-coords
1. Initialize where $H(Z,0)\neq 0$;
2. Propagate p times reversing the arrows in dep. graph;
3. Check empty nodes.

Ex. $\mathcal{A}=\mathcal{Z}+\mathcal{B}+\mathrm{SEQ}(\mathcal{C})$, $\mathcal{B}=\mathcal{Z}\cdot\mathcal{D}+\mathcal{Z}\cdot\mathcal{B}\cdot\mathcal{B}$, $\mathcal{C}=\mathcal{Z}\cdot\mathrm{SEQ}(\mathcal{B})$, $\mathcal{D}=\mathcal{B}+\mathcal{Z}\cdot\mathcal{D}$.

dep. graph at 0



dep. graph



reduced grammar

$\mathcal{A}=\mathcal{Z}+\mathrm{SEQ}(\mathcal{C})$,
$\mathcal{C}=\mathcal{Z}$.

Polynomial Species

$\mathscr{A} = \mathscr{Z} + \mathscr{Z} \cdot \mathscr{B} \cdot \mathscr{B}$;

$\mathscr{B} = \mathscr{Z} + \mathscr{Z} . \mathscr{Z} .$

Def. $\mathscr{Y}$ polynomial when its gf is.

Prop. $\mathscr{Y} = \mathscr{H}(\mathscr{Z}, \mathscr{Y})$ well-founded at 0.
Polynomial solution
$\Longleftrightarrow$ $\mathscr{H}$ polynomial and $\partial\mathscr{H}/\partial\mathscr{Y}$ nilpotent.

The dependency graph itself has no cycle

Polynomial Species

$\mathscr{A} = \mathscr{Z} + \mathscr{Z} \cdot \mathscr{B} \cdot \mathscr{B}$;

$\mathscr{B} = \mathscr{Z} + \mathscr{Z} . \mathscr{Z} .$

Def. $\mathscr{Y}$ polynomial when its gf is.

Prop. $\mathscr{Y} = \mathscr{H}(\mathscr{Z}, \mathscr{Y})$ well-founded at 0.
Polynomial solution
$\iff$ $\mathscr{H}$ polynomial and $\partial\mathscr{H}/\partial\mathscr{Y}$ nilpotent.

The dependency graph itself has no cycle

This method can be extended to *partially polynomial*
($\mathscr{Y}(\mathscr{Z}, \mathscr{U})$ polynomial wrt $\mathscr{U}$ when $[z^n]\mathbf{Y}(z,u)$ polynomial for all n)

Well-Founded: gl case

$$\mathscr{Y}=\mathscr{H}(\mathscr{X},\mathscr{Y})$$

Natural examples where $\mathscr{H}(0,0)\neq 0$:

Sequence: $\mathscr{A}=1+\mathscr{X}\cdot\mathscr{A}$;

Functional graphs: $\mathscr{T}=\mathscr{X}\cdot\mathrm{SET}(\mathscr{T})$;

$\mathscr{C}=\mathrm{CYC}(\mathscr{T})$; $\mathscr{G}=\mathrm{SET}(\mathscr{C})$;

Difficulty: avoid creating an infinite number of objects of a fixed size.

Well-Founded: gl case

$$\mathcal{Y}=\mathcal{H}(\mathcal{Z},\mathcal{Y})$$

Natural examples where $\mathcal{H}(0,0)\neq 0$:

Sequence: $\mathcal{A}=1+\mathcal{Z}\cdot\mathcal{A}$;

Functional graphs: $\mathcal{T}=\mathcal{Z}\cdot\mathrm{SET}(\mathcal{T})$;

$\mathcal{C}=\mathrm{CYC}(\mathcal{T})$; $\mathcal{G}=\mathrm{SET}(\mathcal{C})$;

Difficulty: avoid creating an infinite number of objects of a fixed size.

Def. Associated system:

$$\mathcal{Y}=\mathcal{K}(\mathcal{Z},\mathcal{U},\mathcal{Y})=\mathcal{H}(\mathcal{Z},\mathcal{Y})-\mathcal{H}(0,\mathbf{0})+\mathcal{U}\cdot\mathcal{H}(0,\mathbf{0}).$$

Well-Founded: gl case

$$\mathcal{Y}=\mathcal{H}(\mathcal{Z},\mathcal{Y})$$

Natural examples where $\mathcal{H}(0,0)\neq 0$:

Sequence: $\mathcal{A}=1+\mathcal{Z}\cdot\mathcal{A}$;

Functional graphs: $\mathcal{T}=\mathcal{Z}\cdot\textsc{Set}(\mathcal{T})$;

$\mathcal{C}=\textsc{Cyc}(\mathcal{T})$; $\mathcal{G}=\textsc{Set}(\mathcal{C})$;

Difficulty: avoid creating an infinite number of objects of a fixed size.

Def. Associated system:

$$\mathcal{Y}=\mathcal{K}(\mathcal{Z},\mathcal{U},\mathcal{Y})=\mathcal{H}(\mathcal{Z},\mathcal{Y})-\mathcal{H}(0,\mathbf{0})+\mathcal{U}\cdot\mathcal{H}(0,\mathbf{0}).$$

Thm. $\mathcal{Y}=\mathcal{H}(\mathcal{Z},\mathcal{Y})$ well-founded $\Longleftrightarrow$ $\mathcal{Y}=\mathcal{K}(\mathcal{Z},\mathcal{U},\mathcal{Y})$ is well-founded at 0 and its solution polynomial wrt $\mathcal{U}$.

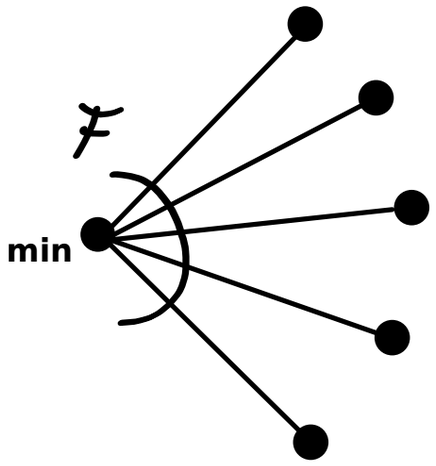
➥ computations reduced to simple manipulations of dependency graphs

Integral Equations

$$\int_0^{\mathcal{Z}} \mathcal{F}(\mathcal{T})d\mathcal{T}$$

- Increasing trees;
- alternating permutations;
- $\textsc{Set}(\mathcal{A}) = 1 + \int \textsc{Set}(\mathcal{A})\mathcal{A}'$
- $\textsc{Cyc}(\mathcal{A}) = \int \textsc{Seq}(\mathcal{A})\mathcal{A}'$

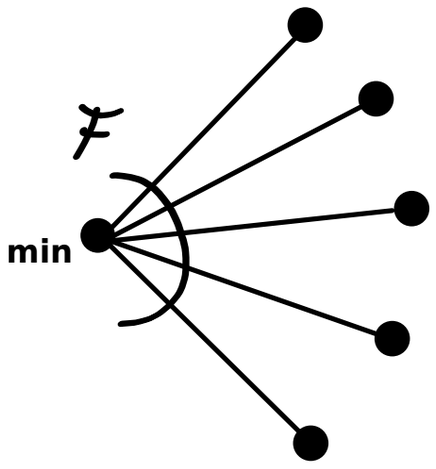
Integral Equations



$$\int_0^{\mathcal{Z}} \mathcal{F}(\mathcal{T})d\mathcal{T}$$

- Increasing trees;
- alternating permutations;
- $\textsc{Set}(\mathcal{A}) = 1 + \int \textsc{Set}(\mathcal{A})\mathcal{A}'$
- $\textsc{Cyc}(\mathcal{A}) = \int \textsc{Seq}(\mathcal{A})\mathcal{A}'$

Integral Equations

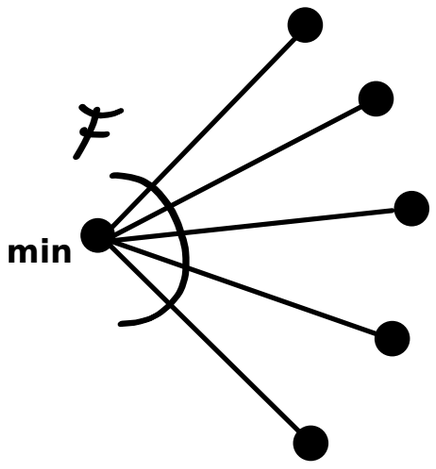


$$\int_0^{\mathcal{Z}} \mathcal{F}(\mathcal{T})d\mathcal{T}$$

- Increasing trees;
- alternating permutations;
- $\textsc{Set}(\mathcal{A}) = 1 + \int \textsc{Set}(\mathcal{A})\mathcal{A}'$
- $\textsc{Cyc}(\mathcal{A}) = \int \textsc{Seq}(\mathcal{A})\mathcal{A}'$

$$\boldsymbol{\mathcal{Y}}(\mathcal{Z}) = \boldsymbol{\mathcal{H}}(\mathcal{Z}, \boldsymbol{\mathcal{Y}}(\mathcal{Z})) + \int_0^{\mathcal{Z}} \boldsymbol{\mathcal{G}}(\mathcal{T}, \boldsymbol{\mathcal{Y}}(\mathcal{T}))\, d\mathcal{T}$$

Integral Equations



$$\int_0^{\mathcal{Z}} \mathcal{F}(\mathcal{T})d\mathcal{T}$$

- Increasing trees;
- alternating permutations;
- $\textsc{Set}(\mathcal{A}) = 1 + \int \textsc{Set}(\mathcal{A})\mathcal{A}'$
- $\textsc{Cyc}(\mathcal{A}) = \int \textsc{Seq}(\mathcal{A})\mathcal{A}'$

Prop. $\boldsymbol{\mathcal{Y}}(\mathcal{Z}) = \boldsymbol{\mathcal{H}}(\mathcal{Z}, \boldsymbol{\mathcal{Y}}(\mathcal{Z})) + \int_0^{\mathcal{Z}} \boldsymbol{\mathcal{G}}(\mathcal{T}, \boldsymbol{\mathcal{Y}}(\mathcal{T}))\, d\mathcal{T}$

well-founded when:

1. the part without $\int$ is well-founded;
2. an extra (technical) condition when $\boldsymbol{\mathcal{H}}(0, \mathbf{0}) \neq \mathbf{0}$.

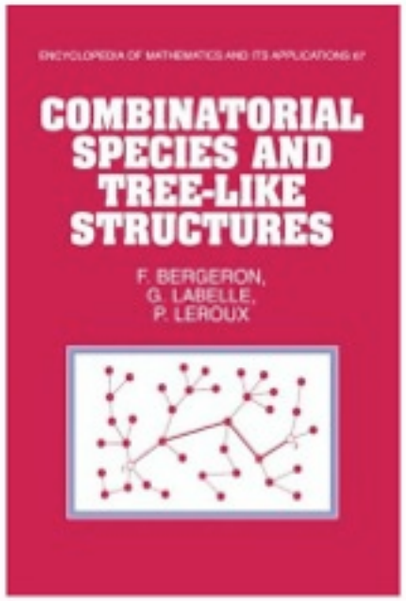
Conclusion

Conclusion

- Simple algorithms check that a specification is well-founded;

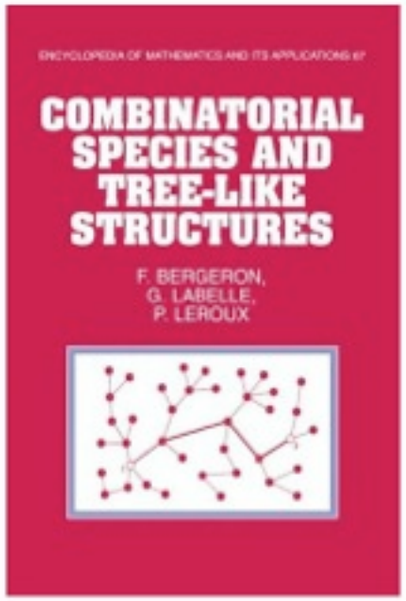
Conclusion

- Simple algorithms check that a specification is well-founded;
- Species theory provides a good framework for these computations;



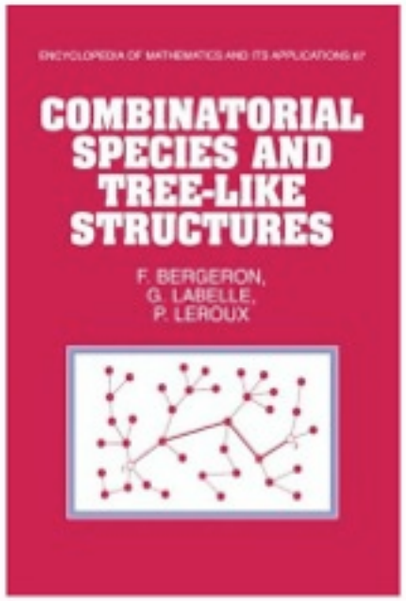
Conclusion

- Simple algorithms check that a specification is well-founded;
- Species theory provides a good framework for these computations;
- Nilpotence of the Jacobian matrix is a practical criterion in proofs;



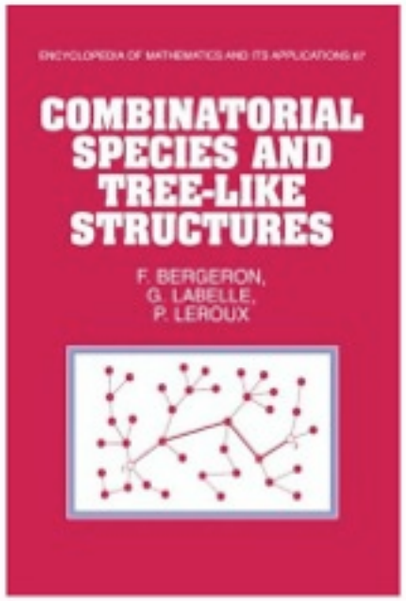
Conclusion

- Simple algorithms check that a specification is well-founded;
- Species theory provides a good framework for these computations;
- Nilpotence of the Jacobian matrix is a practical criterion in proofs;
- Much more is possible: see Michèle Soria's talk on Friday.



Conclusion

- Simple algorithms check that a specification is well-founded;
- Species theory provides a good framework for these computations;
- Nilpotence of the Jacobian matrix is a practical criterion in proofs;
- Much more is possible: see Michèle Soria's talk on Friday.



The end