Algebraic Synchronization Trees and Processes

Luca Aceto
ICE-TCS, School of Computer Science, Reykjavik University

IMT Lucca, 30 May 2012

Joint work with Arnaud Carayol (University Paris-Est), Zoltán Ésik (University of Szeged) and Anna Ingólfsdóttir (Reykjavik University)

What is this talk about?

The research question

What is the expressive power of 'algebraic' recursion schemes as a means for defining synchronization trees up to isomorphism, bisimilarity and language equivalence?

Some of the questions to be answered before we start

1. What is an 'algebraic' recursion scheme?
2. What are synchronization trees?
3. What do you mean by 'define'?

What is this talk about?

The research question

What is the expressive power of 'algebraic' recursion schemes as a means for defining synchronization trees up to isomorphism, bisimilarity and language equivalence?

Some of the questions to be answered before we start

1. What is an 'algebraic' recursion scheme?
2. What are synchronization trees?
3. What do you mean by 'define'?

Recursion schemes (by example)

An algebraic scheme

$$F(n) = \text{ifzero}(n, 1, \text{mult}(2, F(\text{pred}(n)))),$$

where the symbols ifzero, mult, pred, 1 and 2 denote given function symbols.

A regular scheme

$$\begin{aligned} X &= f(X, Y) \\ Y &= a \end{aligned}$$

What is the meaning of these recursion schemes?

Answer of initial algebra semantics

The semantics of a recursive program scheme is the infinite term tree that is the 'least fixed point' of the system of equations associated with the program scheme.

Recursion schemes (by example)

An algebraic scheme

$$F(n) = \text{ifzero}(n, 1, \text{mult}(2, F(\text{pred}(n)))),$$

where the symbols ifzero, mult, pred, 1 and 2 denote given function symbols.

A regular scheme

$$\begin{aligned} X &= f(X, Y) \\ Y &= a \end{aligned}$$

What is the meaning of these recursion schemes?

Answer of initial algebra semantics

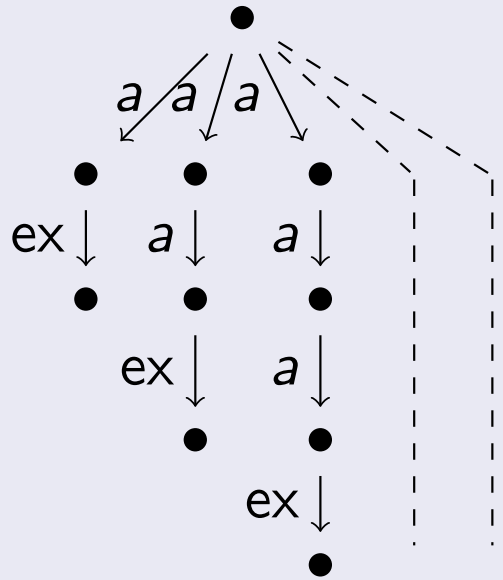
The semantics of a recursive program scheme is the infinite term tree that is the 'least fixed point' of the system of equations associated with the program scheme.

Synchronization trees: A classic model of process behaviour

Synchronization trees arise as unfoldings of labelled transition systems and have been used to give denotational semantics to process description languages.

Example of a synchronization tree

- A synchronization tree that describes a process that can perform any finite sequence of a's and terminate successfully thereafter.
- Legenda: ex = successful termination.
- We only consider countable trees.



Operations on synchronization trees

Assume an alphabet $\mathcal{A}$ of actions, with $a, b, c, d \in \mathcal{A}$.

- Constants: 0 (one node tree), 1 (the tree $r \xrightarrow{\mathsf{ex}} r'$), a (the tree $r \xrightarrow{a} r' \xrightarrow{\mathsf{ex}} r''$) for each $a \in \mathcal{A}$
- Unary operations: action prefixing $a._-$, for each $a \in \mathcal{A}$
- Binary operations:
  - sum: $t + t'$ is obtained by glueing the two trees at the root and
  - sequential composition: $t \cdot t'$ is obtained by replacing each edge of t labelled ex by a copy of t'.

Two signatures for synchronization trees: Γ and Δ

- Γ contains $+$, $0, 1$ and each letter $a \in \mathcal{A}$ as a unary symbol. (Cf. Milner's Basic CCS)
- Δ contains $+, \cdot$, $0, 1$ and each letter $a \in \mathcal{A}$ as a constant symbol. (Cf. Bergstra and Klop's Basic Process Algebra (BPA))

Bisimilarity and language equivalence over synchronization trees

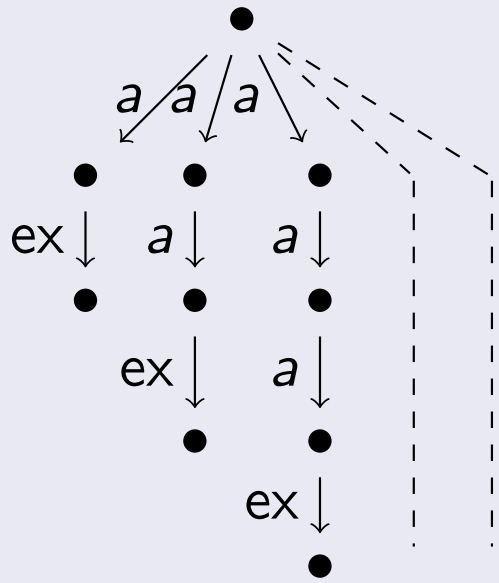
We consider synchronization trees up to isomorphism, bisimilarity and language equivalence.

- Two synchronization trees $t = (V, v_0, E, l)$ and $t' = (V', v_0', E', l')$ are bisimilar if there is some symmetric relation $R \subseteq (V \times V') \cup (V' \times V)$ that relates their roots, and such that if $(v_1, v_2) \in R$ and there is some edge (v_1, v_1'), then there is an equally-labelled edge (v_2, v_2') with $(v_1', v_2') \in R$.
- The path language of a synchronization tree is composed of the words in $\mathcal{A}^*$ that label a path from the root to the source of an exit edge. Two trees are language equivalent if they have the same path language.

Bisimilarity and language equivalence at work

Example

- Infinitely many subtrees up to language equivalence, and therefore up to isomorphism and bisimilarity
- Path language: $\{a^n \mid n \geq 1\}$



Questions:

1. How can we use recursion schemes to define synchronization trees?
2. What type of recursion schemes do we consider?

Examples of recursion schemes defining synchronization trees

A Δ-regular recursion scheme

$$X = (X \cdot a) + a$$

A Γ-algebraic recursion scheme

$$\begin{aligned} F_1 &= F_2(a.1) \\ F_2(v) &= v + F_2(a.v). \end{aligned}$$

Both these schemes define the synchronization tree

Examples of recursion schemes defining synchronization trees

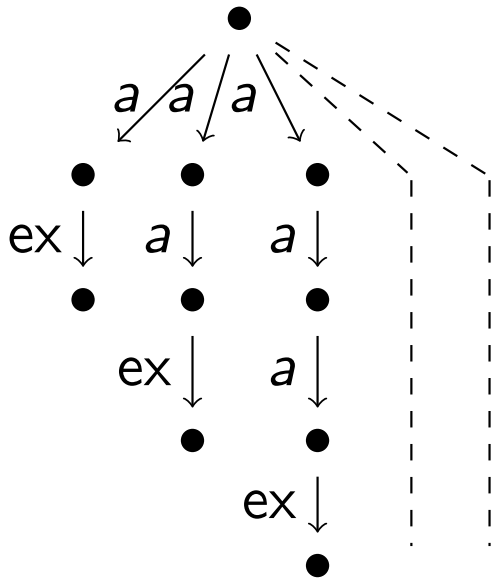
A Δ-regular recursion scheme

$$X = (X \cdot a) + a$$

A Γ-algebraic recursion scheme

$$\begin{aligned} F_1 &= F_2(a.1) \\ F_2(v) &= v + F_2(a.v). \end{aligned}$$

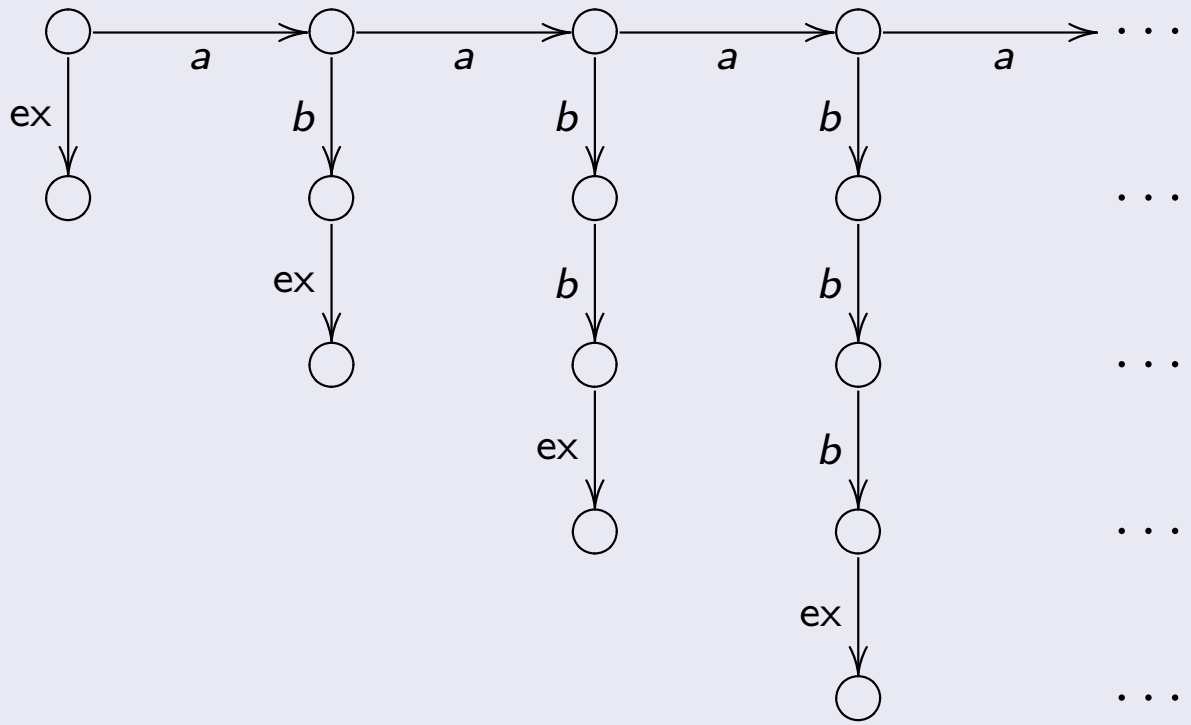
Both these schemes define the synchronization tree



The tree defined by a scheme (defined)

Motto: A Γ- or Δ-algebraic recursion scheme defines the synchronization tree that is the initial solution of the scheme.

Example: $F = 1 + a \cdot F \cdot b$

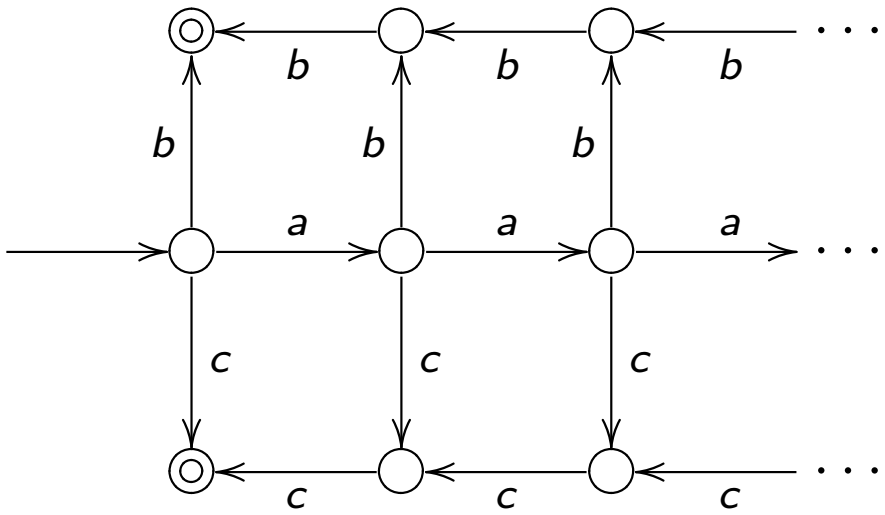


The tree defined by a scheme (example)

The Γ-algebraic recursion scheme

$$\begin{aligned} F_1 &= b + c + a.F_2(b^2, c^2) \\ F_2(v_1, v_2) &= v_1 + v_2 + a.F_2(b.v_1, c.v_2) \end{aligned}$$

defines the unfolding of



This LTS is not expressible in BPA modulo bisimilarity.

Pictorial summary of our results

Question: What is the expressive power of Γ- or Δ-regular and of Γ- or Δ-algebraic recursion schemes as a means of defining synchronization trees?

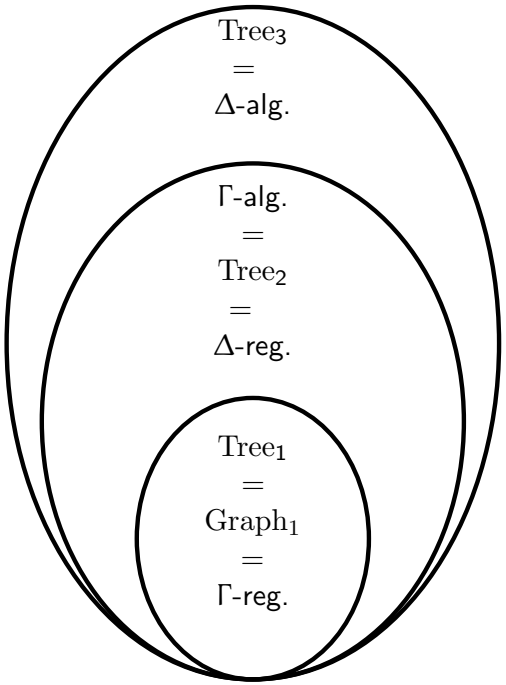
(Language equivalence)

(Bisimilarity)

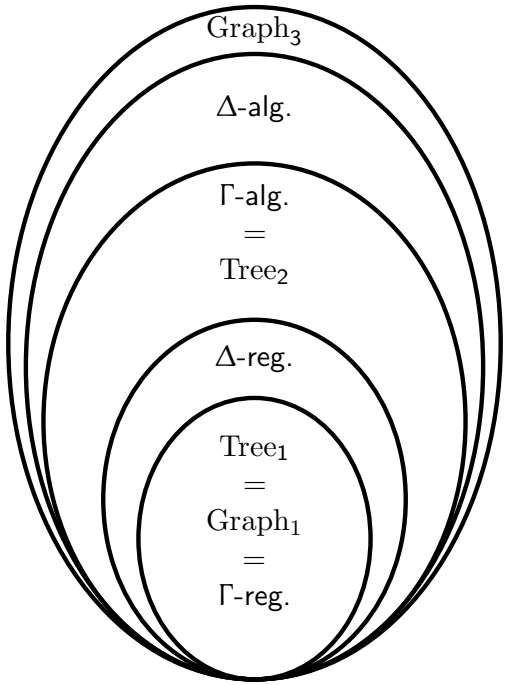
(Isomorphism)

Pictorial summary of our results

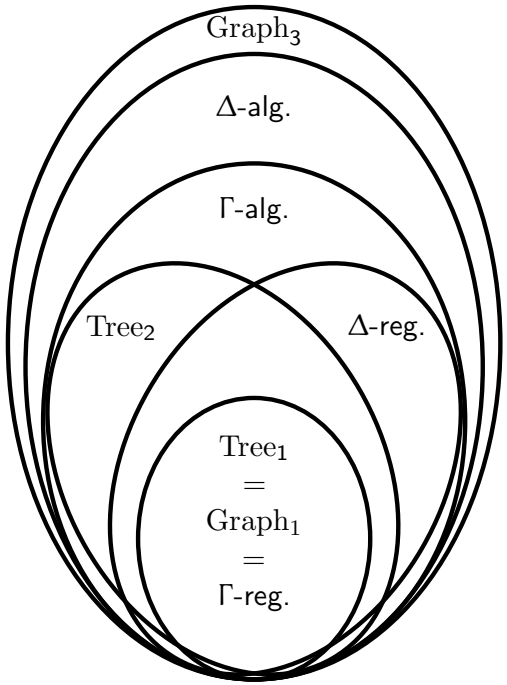
Question: What is the expressive power of Γ- or Δ-regular and of Γ- or Δ-algebraic recursion schemes as a means of defining synchronization trees?



(Language equivalence)



(Bisimilarity)



(Isomorphism)

Δ-regular $\subset$ Γ-algebraic (part 1)

Theorem

Every Δ-regular tree is Γ-algebraic. Hence every synchronization tree that is the unfolding of a BPA process is Γ-algebraic.

Proof: We translate each regular Δ-recursion scheme into a Γ-algebraic scheme with recursion variables of arity one that defines the same tree up to isomorphism.

Example

$$F = 1 + a \cdot F \cdot b \quad \mapsto \quad \begin{aligned} G_0 &= G(1) \\ G(v) &= v + a.(G(b.v)) \end{aligned}$$

Δ-regular $\subset$ Γ-algebraic (part 2)

Theorem

There exists a Γ-algebraic synchronization tree that is not bisimilar to any Δ-regular tree.

Proof: Let $\mathcal{A} = \{a, b\}$, and consider the synchronization tree T, defined by the Γ-algebraic recursion scheme:

$$\begin{aligned} S &= F(1 + b.1) \\ F(v_1) &= v_1 + a.(F(1 + b.v_1)) \ . \end{aligned}$$

We show that every Δ-regular scheme E defining a synchronization tree bisimilar to T is equivalent to a right-linear one, modulo bisimilarity. This implies that T is regular, which yields a contradiction.

Γ-algebraic $\subset$ Δ-algebraic (part 1)

Goal: To show that Γ-algebraic $\subset$ Δ-algebraic.

Approach:

1. Following Courcelle, we offer a language-theoretic characterization of the expressive power of Γ-algebraic recursion schemes.
2. We provide an example of a Δ-algebraic synchronization tree that does not have the property on which the characterization relies.

Inspiration for step 1:

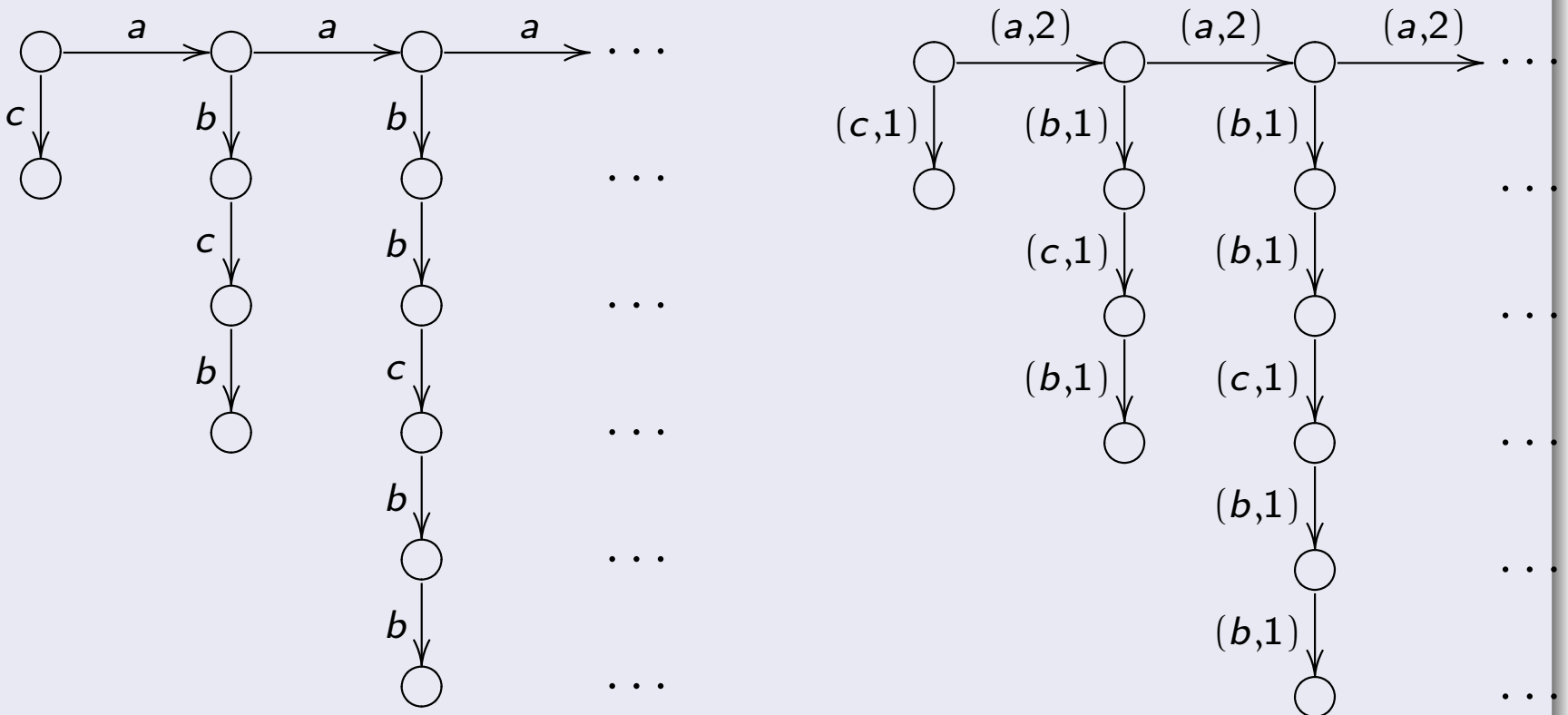
Theorem (Courcelle 1978)

A term tree is algebraic iff the set of its branches is a deterministic context-free language.

Γ-algebraic trees: Language-theoretic characterization I

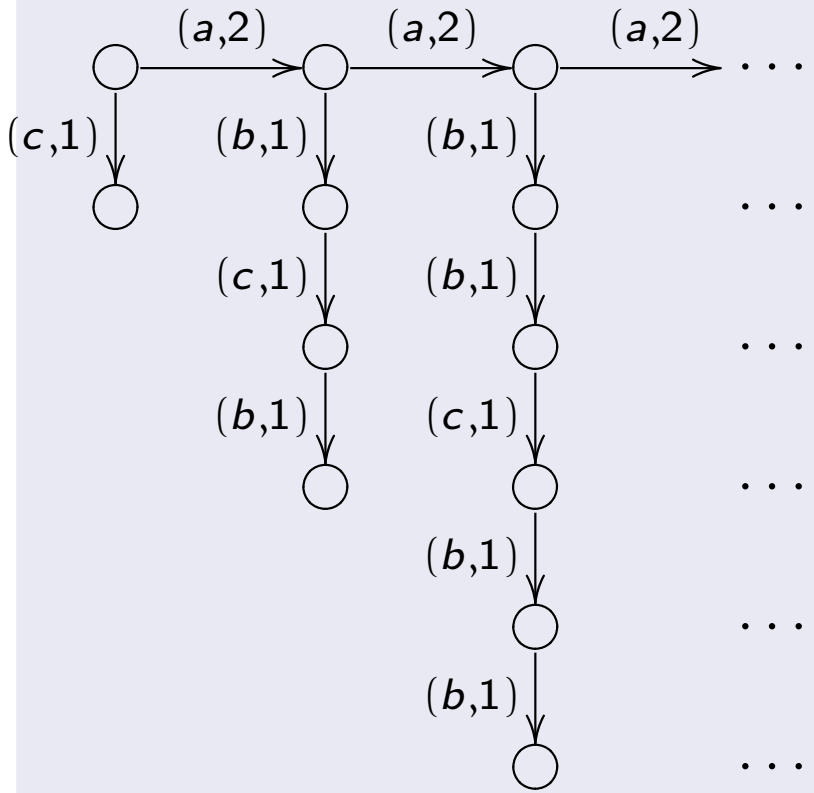
A synchronization tree is bounded if there is a constant k such that the outdegree of each vertex is at most k.

Step 1: From a bounded tree t (left) to a determinization t' (right)



Γ-algebraic trees: Language-theoretic characterization II

Step 2: From a a determinization t' (left) to its branch language (right)

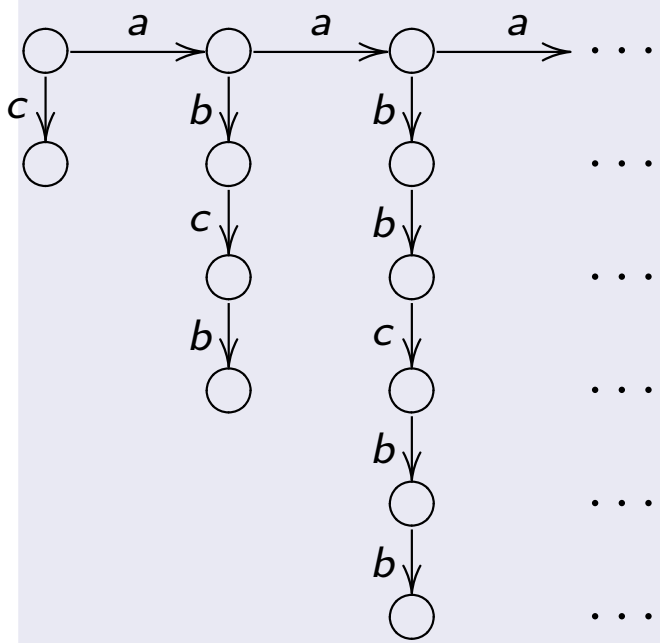


- The branch language $L(t')$ contains words such as 2, $2(c,1)0$, $2(a,2)2$, $2(a,2)2(b,1)1$, ...
- The family $\mathcal{L}(t)$ of branch languages corresponding to t consists of all the $L(t')$ with t' a determinization of t.

Γ-algebraic trees: Language-theoretic characterization III

Theorem (Courcelle-type characterization): A bounded synchronization tree t is Γ-algebraic iff $\mathcal{L}(t)$ contains a deterministic context-free language.

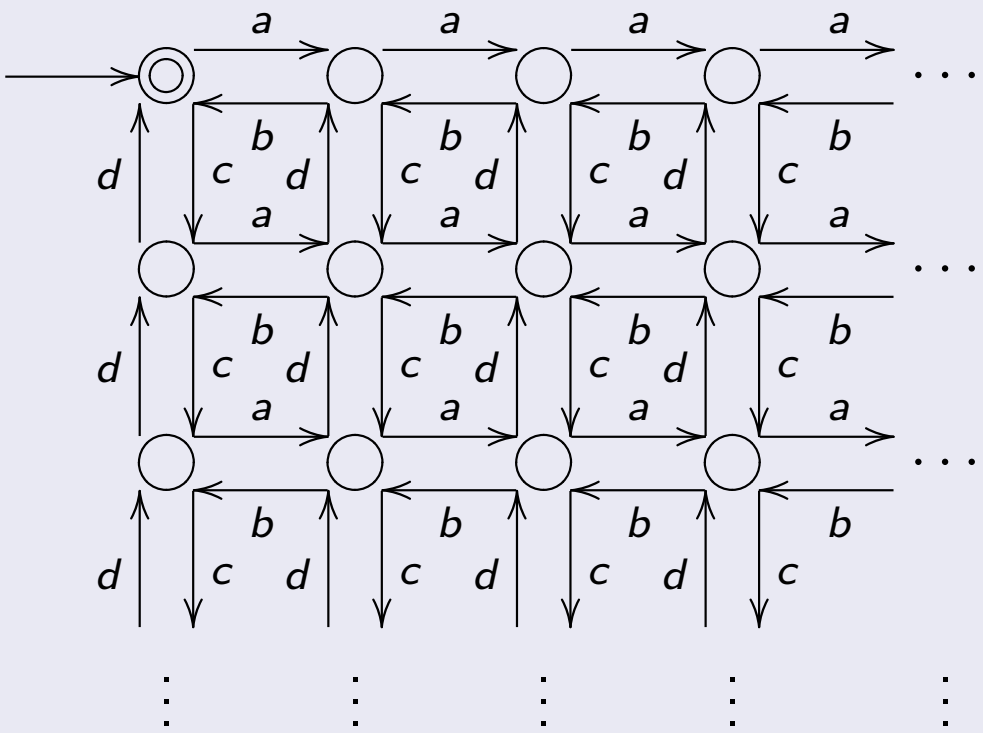
Corollary: The following tree is Δ-algebraic, but not Γ-algebraic.



$$F_0 = F(1)$$
$$F(v) = a \cdot F(b \cdot v) + v \cdot c \cdot v \cdot 0$$

Results on the bag over a binary alphabet (part 1)

(Bergstra and Klop, 1984) The bag over a binary alphabet is not definable in BPA



Theorem: The synchronization tree t_{bag} that is the unfolding of the LTS above is not Γ-algebraic, even up to language equivalence.

Results on the bag over a binary alphabet (part 2)

Theorem

The synchronization tree t_{bag} is not Δ-algebraic.

Proof: By the power of logic!

1. t_{bag} has an undecidable Monadic Second Order theory. (We encode the halting program for a two-counter machine P in Monadic Second Order logic by constructing a formula φ_P such that the machine does not halt on input $(0,0)$ iff t_{bag} is a model of φ_P.)
2. Every Δ-algebraic synchronization tree has a decidable Monadic Second Order theory, because the Δ-algebraic synchronization trees are in the class Graph_3 in the Caucal hierarchy.

The Caucal Hierarchy of Edge-labelled Graphs

Definition of the Caucal hierarchy

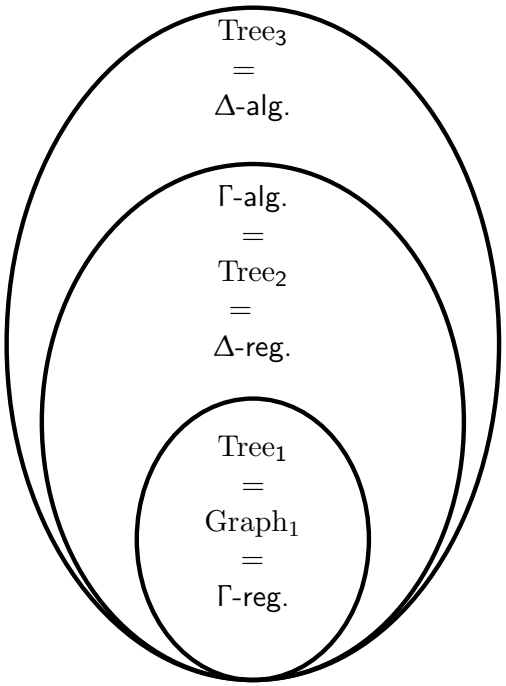
$$\begin{array}{ccc} \mathrm{Tree}_0 \text{ (finite trees)} & & \mathrm{Graph}_0 \text{ (finite graphs)} \\ \mathrm{Tree}_{n+1} & \xleftarrow{\text{unfold}} & \mathrm{Graph}_n \\ \mathrm{Tree}_{n+1} & \xrightarrow{\text{MSO transduction}} & \mathrm{Graph}_{n+1} \end{array}$$

- Tree_1 is the collection of regular trees.
- Graph_1 is the set of all prefix-recognizable graphs.

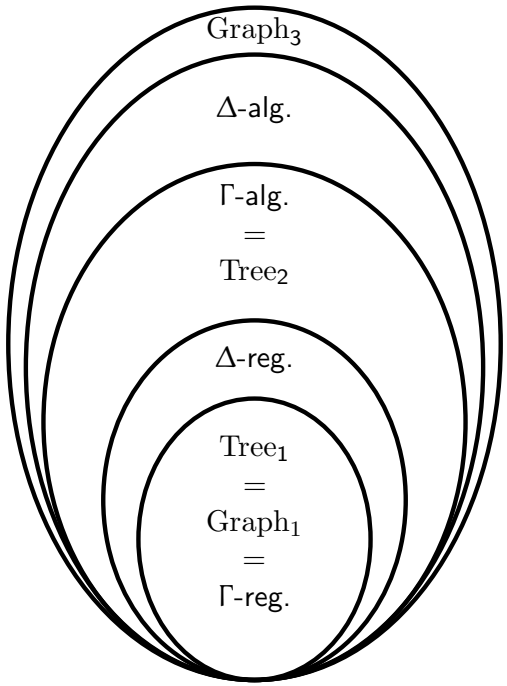
Fundamental Theorem (Caucal): All graphs in the above-defined hierarchy have a decidable Monadic Second Order theory.

Recursion Schemes vs. the Caucal Hierarchy

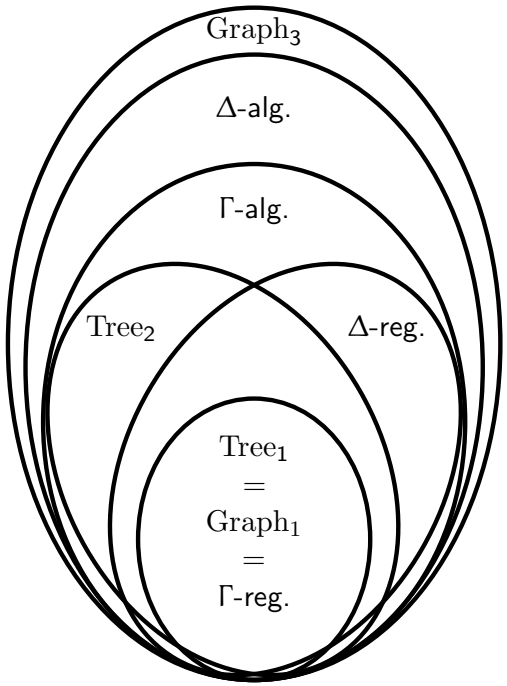
The return of the pictorial summary!



(Language equivalence)



(Bisimilarity)

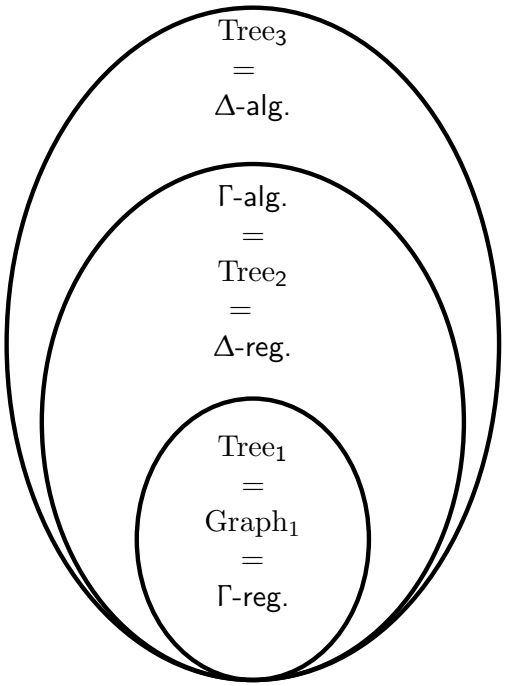


(Isomorphism)

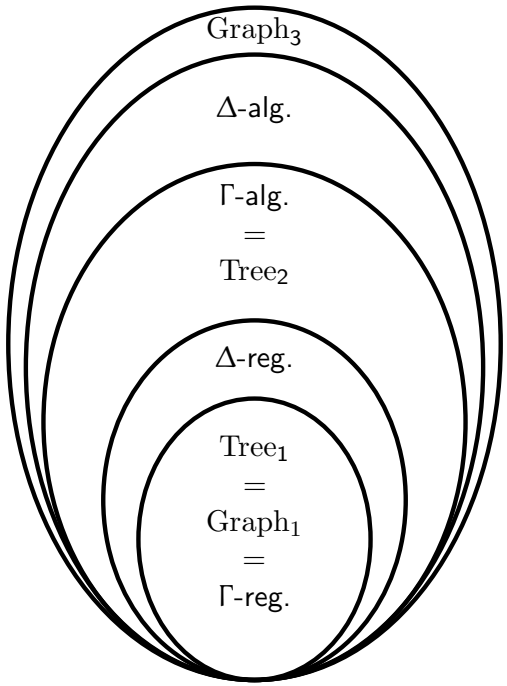
What can we use this for?

Recursion Schemes vs. the Caucal Hierarchy

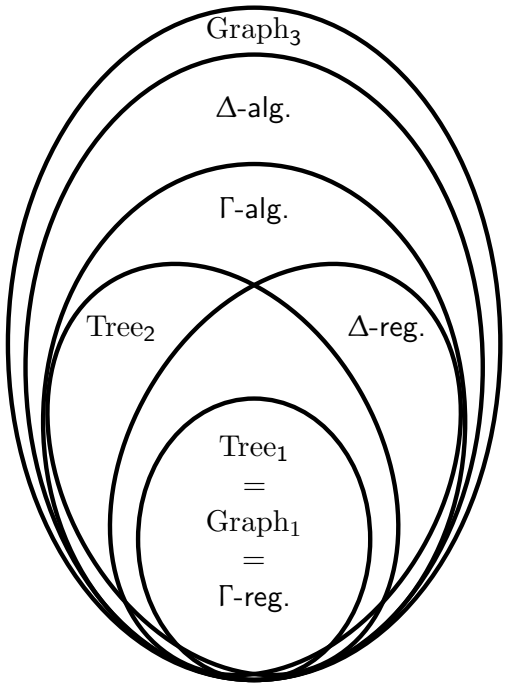
The return of the pictorial summary!



(Language equivalence)



(Bisimilarity)



(Isomorphism)

What can we use this for?

Logic Strikes Back!

Theorem

There exists a Γ-algebraic synchronization tree whose minimization with respect to bisimilarity does not have a decidable MSO-theory, and hence does not belong to the Caucal hierarchy.

Proof: Consider the Γ-algebraic recursion scheme:

$$\begin{aligned} S &= F(0,0) \\ F(x,y) &= a.F(f.x,y) + b.F(x,f.y) + c.F(0,y) + d.F(x,0) + \\ &\quad e.x + e.y. \end{aligned}$$

We show that the minimization of the tree defined by the above scheme has an undecidable MSO-theory by reduction from the halting problem for 2-counter machines with increment, reset and equality test.

Conclusions

The message (reprise)

1. We have studied the expressive power of algebraic recursion schemes as a means for defining synchronization trees up to isomorphism, bisimilarity and language equivalence.
2. We have compared the expressive power of recursion schemes with those of the low levels of the Caucal hierarchy.
3. The results rely on tools from the Caucal hierarchy, concurrency theory, initial algebra semantics, language theory, and Monadic Second Order logic, amongst others.
4. For more results and open problems see the ICALP 2012 paper and the current draft version of the full paper.

Parting Words

Thank you!

Any Questions?

You are welcome to visit us at ICE-TCS

ICE-TCS

Icelandic Centre of Excellence
in Theoretical Computer Science

(Spoiler warning: We have no money. . .)