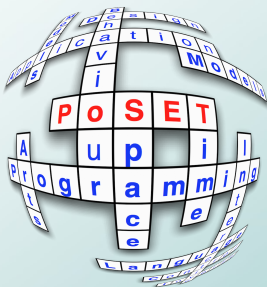


Inverse monoids of higher dimensional strings



David Janin,
research project PoSET
CNRS LaBRI & INRIA Bordeaux,
Bordeaux INP, University of Bordeaux
©ICTAC 2015, Cali, Columbia, October 2015

Design tool for music systems

My own wish list

- ▶ simple, to be used by artists, kids and myself,
 - ▶ efficient, to do live music,
 - ▶ both concurrent and parallel, just as live music,
 - ▶ modular and hierarchical, for incremental design,
 - ▶ multi time-scaled: from causal to continuous time, idem,
 - ▶ mathematically robust, to go beyond one shot prototyping,
- with many more requirements that will appear on the way.

Design tool for music systems

My own wish list

- ▶ simple, to be used by artists, kids and myself,
 - ▶ efficient, to do live music,
 - ▶ both concurrent and parallel, just as live music,
 - ▶ modular and hierarchical, for incremental design,
 - ▶ multi time-scaled: from causal to continuous time, idem,
 - ▶ mathematically robust, to go beyond one shot prototyping,
- with many more requirements that will appear on the way.

Design tool for music systems

My own wish list

- ▶ simple, to be used by artists, kids and myself,
 - ▶ efficient, to do live music,
 - ▶ both concurrent and parallel, just as live music,
 - ▶ modular and hierarchical, for incremental design,
 - ▶ multi time-scaled: from causal to continuous time, idem,
 - ▶ mathematically robust, to go beyond one shot prototyping,
- with many more requirements that will appear on the way.

Design tool for music systems

My own wish list

- ▶ simple, to be used by artists, kids and myself,
 - ▶ efficient, to do live music,
 - ▶ both concurrent and parallel, just as live music,
 - ▶ modular and hierarchical, for incremental design,
 - ▶ multi time-scaled: from causal to continuous time, idem,
 - ▶ mathematically robust, to go beyond one shot prototyping,
- with many more requirements that will appear on the way.

Design tool for music systems

My own wish list

- ▶ simple, to be used by artists, kids and myself,
 - ▶ efficient, to do live music,
 - ▶ both concurrent and parallel, just as live music,
 - ▶ modular and hierarchical, for incremental design,
 - ▶ multi time-scaled: from causal to continuous time, idem,
 - ▶ mathematically robust, to go beyond one shot prototyping,
- with many more requirements that will appear on the way.

Design tool for music systems

My own wish list

- ▶ simple, to be used by artists, kids and myself,
 - ▶ efficient, to do live music,
 - ▶ both concurrent and parallel, just as live music,
 - ▶ modular and hierarchical, for incremental design,
 - ▶ multi time-scaled: from causal to continuous time, idem,
 - ▶ mathematically robust, to go beyond one shot prototyping,
- with many more requirements that will appear on the way.

Design tool for music systems

My own wish list

- ▶ simple, to be used by artists, kids and myself,
 - ▶ efficient, to do live music,
 - ▶ both concurrent and parallel, just as live music,
 - ▶ modular and hierarchical, for incremental design,
 - ▶ multi time-scaled: from causal to continuous time, idem,
 - ▶ mathematically robust, to go beyond one shot prototyping,
- with many more requirements that will appear on the way.

Design tool for music systems

My own wish list

- ▶ simple, to be used by artists, kids and myself,
 - ▶ efficient, to do live music,
 - ▶ both concurrent and parallel, just as live music,
 - ▶ modular and hierarchical, for incremental design,
 - ▶ multi time-scaled: from causal to continuous time, idem,
 - ▶ mathematically robust, to go beyond one shot prototyping,
- with many more requirements that will appear on the way.

Design tool for music systems

My own wish list

- ▶ simple, to be used by artists, kids and myself,
 - ▶ efficient, to do live music,
 - ▶ both concurrent and parallel, just as live music,
 - ▶ modular and hierarchical, for incremental design,
 - ▶ multi time-scaled: from causal to continuous time, idem,
 - ▶ mathematically robust, to go beyond one shot prototyping,
- with many more requirements that will appear on the way.

Design tool for music systems

My own wish list

- ▶ simple, to be used by artists, kids and myself,
- ▶ efficient, to do live music,
- ▶ both concurrent and parallel, just as live music,
- ▶ modular and hierarchical, for incremental design,
- ▶ multi time-scaled: from causal to continuous time, idem,
- ▶ mathematically robust, to go beyond one shot prototyping,

with many more requirements that will appear on the way.

Design tool for music systems

My own wish list

- ▶ simple, to be used by artists, kids and myself,
- ▶ efficient, to do live music,
- ▶ both concurrent and parallel, just as live music,
- ▶ modular and hierarchical, for incremental design,
- ▶ multi time-scaled: from causal to continuous time, idem,
- ▶ mathematically robust, to go beyond one shot prototyping,

with many more requirements that will appear on the way.

Design tool for music systems

My own wish list

- ▶ simple, to be used by artists, kids and myself,
- ▶ efficient, to do live music,
- ▶ both concurrent and parallel, just as live music,
- ▶ modular and hierarchical, for incremental design,
- ▶ multi time-scaled: from causal to continuous time, idem,
- ▶ mathematically robust, to go beyond one shot prototyping,

with many more requirements that will appear on the way.

Design tool for music systems

My own wish list

- ▶ simple, to be used by artists, kids and myself,
- ▶ efficient, to do live music,
- ▶ both concurrent and parallel, just as live music,
- ▶ modular and hierarchical, for incremental design,
- ▶ multi time-scaled: from causal to continuous time, idem,
- ▶ mathematically robust, to go beyond one shot prototyping,

with many more requirements that will appear on the way.

Design tool for music systems

My own wish list

- ▶ simple, to be used by artists, kids and myself,
- ▶ efficient, to do live music,
- ▶ both concurrent and parallel, just as live music,
- ▶ modular and hierarchical, for incremental design,
- ▶ multi time-scaled: from causal to continuous time, idem,
- ▶ mathematically robust, to go beyond one shot prototyping,

with many more requirements that will appear on the way.

So let's start

From **strings** to **higher dimensional strings**

with music modeling in the background,

So let's start

From **strings** to **higher dimensional strings**

with music modeling in the background,

Sequential composition and refinement

An (abstract) musical exemple

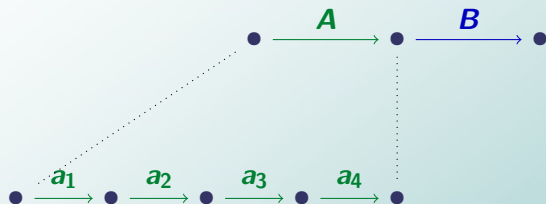


Is this satisfactory ?

In musical refinement, temporal overlaps are common features.

Sequential composition and refinement

An (abstract) musical example

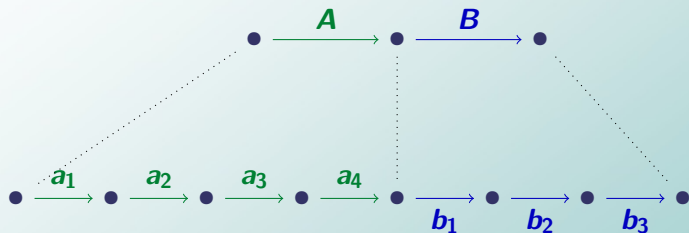


Is this satisfactory ?

In musical refinement, temporal overlaps are common features.

Sequential composition and refinement

An (abstract) musical example

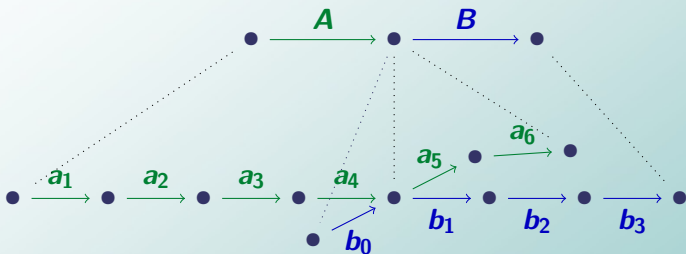


Is this satisfactory ?

In musical refinement, temporal overlaps are common features.

Sequential composition and refinement

An (abstract) musical example

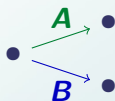


Is this satisfactory ?

In musical refinement, **temporal overlaps** are common features.

Parallel fork and refinement

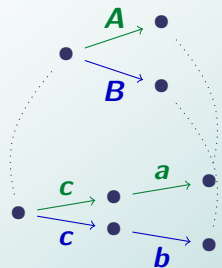
Launching two musical streams



We consider graphs that are deterministic and co-deterministic, or, more generally with hyper-edges, locally unambiguous.

Parallel fork and refinement

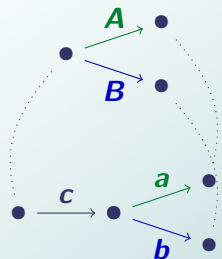
Launching two musical streams



We consider graphs that are deterministic and co-deterministic, or, more generally with hyper-edges, locally unambiguous.

Parallel fork and refinement

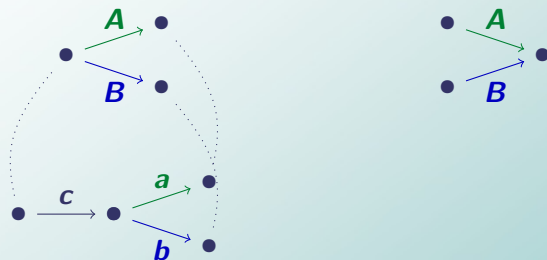
Launching two musical streams



We consider graphs that are **deterministic** and co-deterministic, or, more generally with hyper-edges, locally unambiguous.

Parallel fork/join and refinement

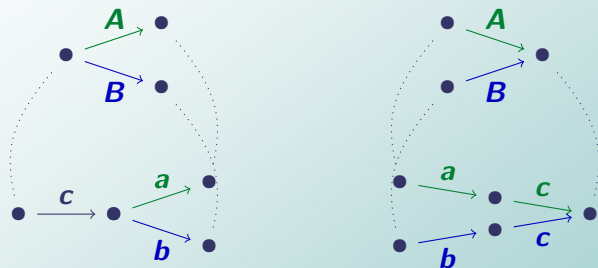
Launching/ending two musical streams



We consider graphs that are **deterministic** and co-deterministic, or, more generally with hyper-edges, locally unambiguous.

Parallel fork/join and refinement

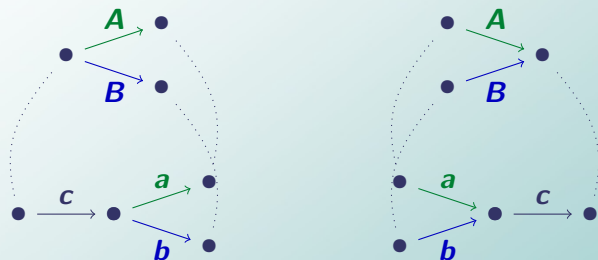
Launching/ending two musical streams



We consider graphs that are **deterministic** and co-deterministic, or, more generally with hyper-edges, locally unambiguous.

Parallel fork/join and refinement

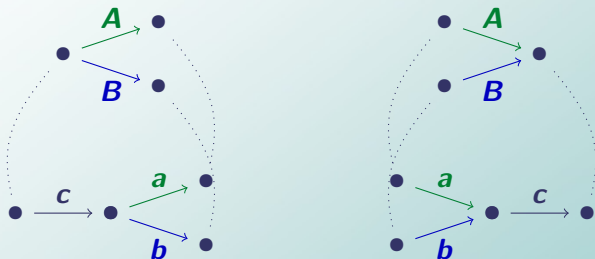
Launching/ending two musical streams



We consider graphs that are **deterministic** and **co-deterministic**, or, more generally with hyper-edges, locally unambiguous.

Parallel fork/join and refinement

Launching/ending two musical streams



We consider graphs that are **deterministic** and **co-deterministic**, or, more generally with hyper-edges, **locally unambiguous**.

So far

We have

- ▶ **locally non ambiguous** labeled (hyper-)graphs,
- ▶ with a “causal/temporal” interpretation
 - ▶ **synchronization** points : vertices,
 - ▶ **causal/temporal** flows : edges.

Question

- ▶ how to **compose** these graphs ?
- ▶ what **generators** ?
- ▶ how to **refine** them ?

Answer

- ▶ **model**: higher dimensional strings (HDS) !

So far

We have

- ▶ **locally non ambiguous** labeled (hyper-)graphs,
- ▶ with a “causal/temporal” interpretation
 - ▶ **synchronization** points : vertices,
 - ▶ **causal/temporal** flows : edges.

Question

- ▶ how to **compose** these graphs ?
- ▶ what **generators** ?
- ▶ how to **refine** them ?

Answer

- ▶ **model**: higher dimensional strings (HDS) !

So far

We have

- ▶ **locally non ambiguous** labeled (hyper-)graphs,
- ▶ with a “causal/temporal” interpretation
 - ▶ **synchronization** points : vertices,
 - ▶ **causal/temporal** flows : edges.

Question

- ▶ how to **compose** these graphs ?
- ▶ what **generators** ?
- ▶ how to **refine** them ?

Answer

- ▶ **model**: higher dimensional strings (HDS) !

So far

We have

- ▶ **locally non ambiguous** labeled (hyper-)graphs,
- ▶ with a “causal/temporal” interpretation
 - ▶ **synchronization** points : vertices,
 - ▶ **causal/temporal** flows : edges.

Question

- ▶ how to **compose** these graphs ?
- ▶ what **generators** ?
- ▶ how to **refine** them ?

Answer

- ▶ **model**: higher dimensional strings (HDS) !

Higher dimensional strings (HDS)

Birooted graphs (or cospans)



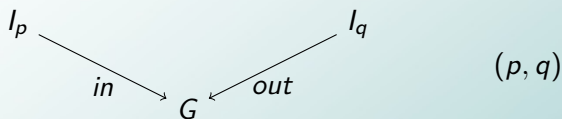
with **graph domain** G , and labeling **root morphisms** in and out from edgeless graphs I_p and I_q ... and some **connectivity** assumptions.

Birooted graph products



Higher dimensional strings (HDS)

Birooted graphs (or cospans)



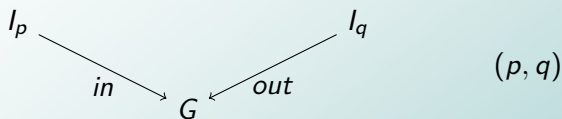
with **graph domain** G , and labeling **root morphisms** in and out from edgeless graphs I_p and I_q ... and some **connectivity assumptions**.

Birooted graph products



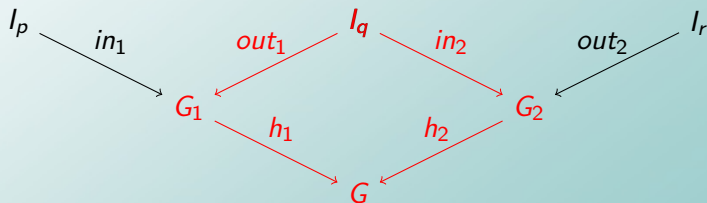
Higher dimensional strings (HDS)

Birooted graphs (or cospans)



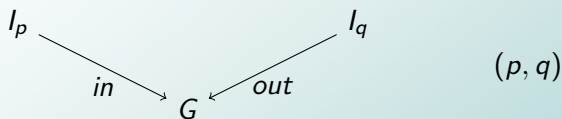
with **graph domain** G , and labeling **root morphisms** in and out from edgeless graphs I_p and I_q ... and some **connectivity assumptions**.

Birooted graph products via **pushout**



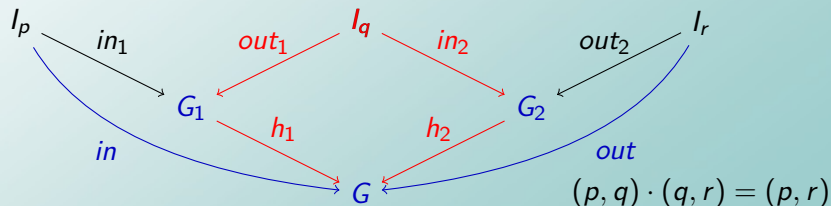
Higher dimensional strings (HDS)

Birooted graphs (or cospans)



with **graph domain** G , and labeling **root morphisms** in and out from edgeless graphs I_p and I_q ... and some **connectivity assumptions**.

Birooted graph products via pushout



A product example

Two components



Step 1: synchronization (or root gluing)

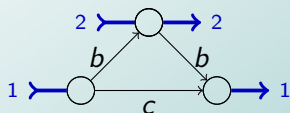
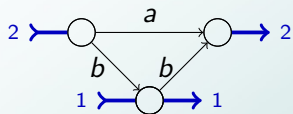


Step 2: fusion (or gluing propagation)



A product example

Two components



Step 1: synchronization (or root gluing)

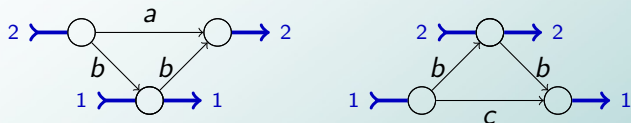


Step 2: fusion (or gluing propagation)

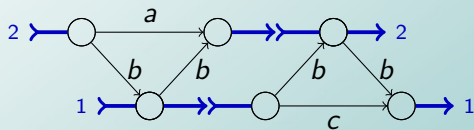


A product example

Two components



Step 1: synchronization (or root gluing)

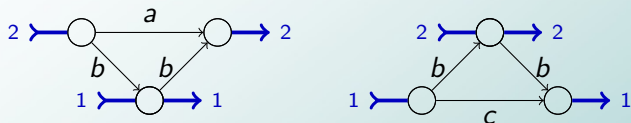


Step 2: fusion (or gluing propagation)

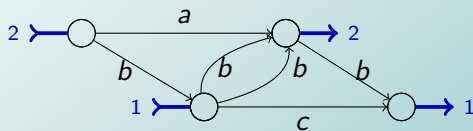


A product example

Two components



Step 1: synchronization (or root gluing)

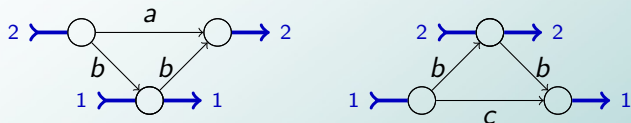


Step 2: fusion (or gluing propagation)

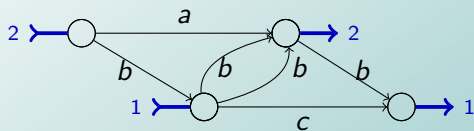


A product example

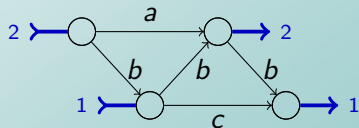
Two components



Step 1: synchronization (or root gluing)



Step 2: fusion (or gluing propagation)



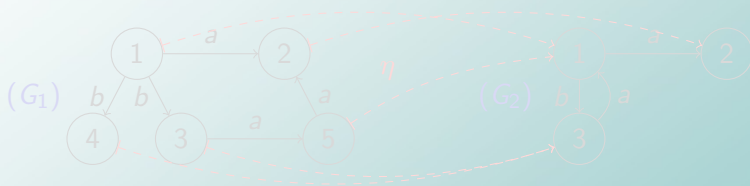
Computing product

Lemma (Bideterminization)

Every (connected) graph has a maximal locally non ambiguous image under graph (connecting) morphism.

This maximal image is computable in linear time.

An example



Computing product in two steps

- ▶ **synchronization**: glue roots (pushout in graphs),
- ▶ **fusion**: apply bideterminization (max. unambiguous image).

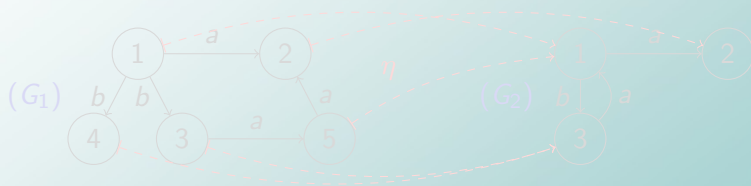
Computing product

Lemma (Bideterminization)

Every (connected) graph has a maximal locally non ambiguous image under graph (connecting) morphism.

This maximal image is computable in linear time.

An example



Computing product in two steps

- ▶ **synchronization**: glue roots (pushout in graphs),
- ▶ **fusion**: apply bideterminization (max. unambiguous image).

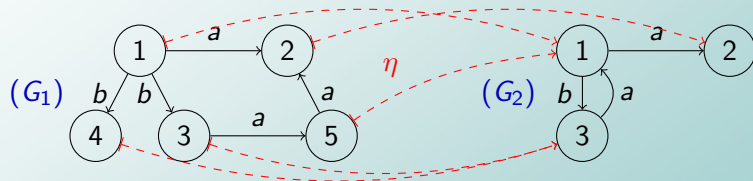
Computing product

Lemma (Bideterminization)

Every (connected) graph has a maximal locally non ambiguous image under graph (connecting) morphism.

This maximal image is computable in linear time.

An example



Computing product in two steps

- ▶ **synchronization**: glue roots (pushout in graphs),
- ▶ **fusion**: apply bideterminization (max. unambiguous image).

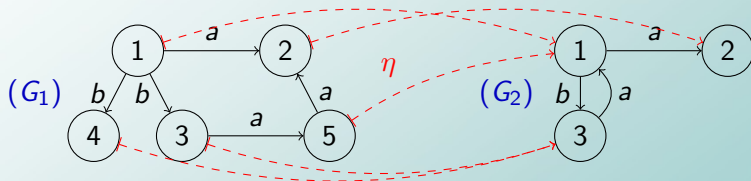
Computing product

Lemma (Bideterminization)

Every (connected) graph has a maximal locally non ambiguous image under graph (connecting) morphism.

This maximal image is computable in linear time.

An example



Computing product in two steps

- ▶ **synchronization**: glue roots (pushout in graphs),
- ▶ **fusion**: apply bideterminization (max. unambiguous image).

Refinement (by examples)

Hyper directed edges and product $HS(A)$



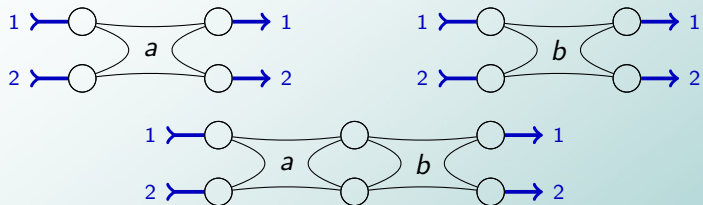
Hyper directed edges replacement via $\varphi : A \rightarrow HS(B)$



that (may ?) simply induces a morphism from $HS(A)$ to $HS(B)$.

Refinement (by examples)

Hyper directed edges and product $HS(A)$



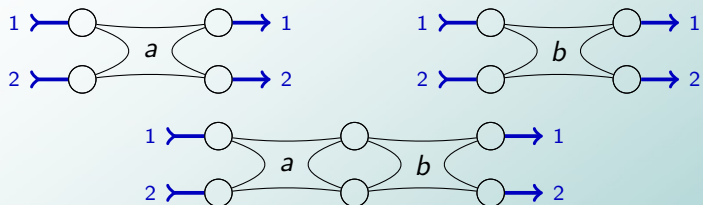
Hyper directed edges replacement via $\varphi : A \rightarrow HS(B)$



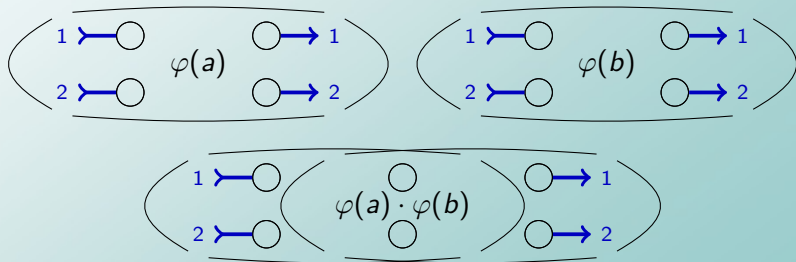
that (may ?) simply induces a morphism from $HS(A)$ to $HS(B)$.

Refinement (by examples)

Hyper directed edges and product $HS(A)$



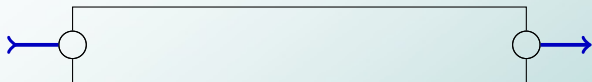
Hyper directed edges replacement via $\varphi : A \rightarrow HS(B)$



that (may ?) simply induces a morphism from $HS(A)$ to $HS(B)$. 

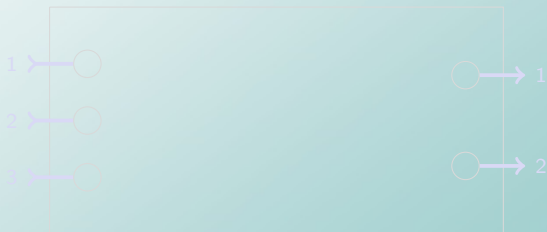
Any relevant mathematical properties ?

Strings



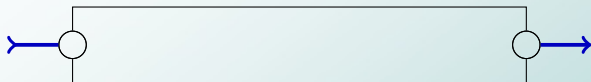
vs

Higher dimensional strings



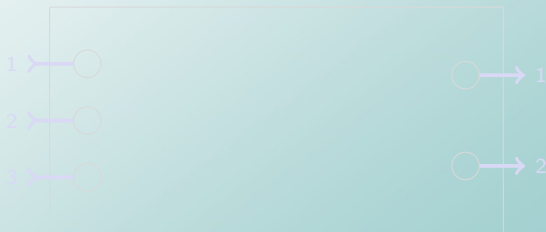
Any relevant mathematical properties ?

Strings and **monoids**



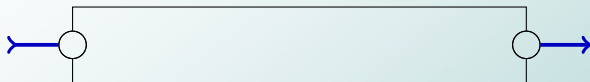
vs

Higher dimensional strings



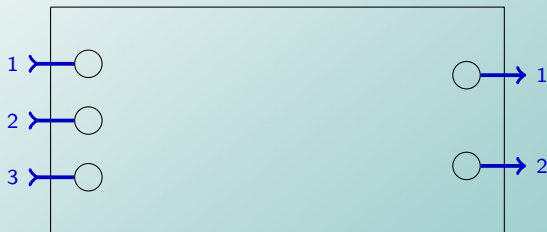
Any relevant mathematical properties ?

Strings



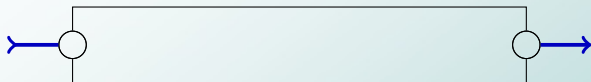
vs

Higher dimensional strings



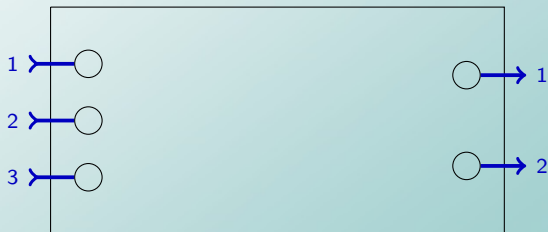
Any relevant mathematical properties ?

Strings



vs

Higher dimensional strings and **inverse monoids**



Monoid of higher dimensional strings

Theorem

Birooted graphs with product (extended by zero) form a monoid.

Theorem

Submonoids of finite birooted graphs with bounded root size are finitely generated.

Monoid of higher dimensional strings

Theorem

Birooted graphs with product (extended by zero) form a monoid.

Theorem

Submonoids of finite birooted graphs with bounded root size are finitely generated.

Monoid of higher dimensional strings

Theorem

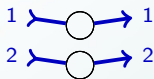
Birooted graphs with product (extended by zero) form a monoid.

Theorem

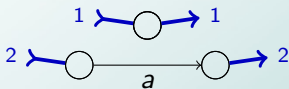
Submonoids of finite birooted graphs with bounded root size are finitely generated.

Generator examples

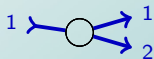
$(\mathbf{1}_2)$



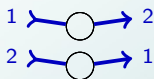
$(T_{2,a})$



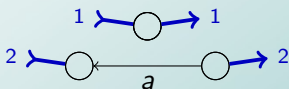
(F_2)



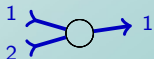
$(P_{2,1,2})$



$(T_{2,\bar{a}})$



(J_2)



Some products of generators

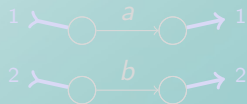
$(T_{2,a} \cdot J_2 \cdot F_2 \cdot T_{2,\bar{a}})$



$(F_2 \cdot T_{2,a} \cdot T_{2,\bar{a}} \cdot T_{2,c} \cdot J_2)$

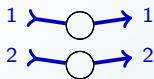


$(T_{2,b} \cdot P_{2,1,2} \cdot T_{2,a} \cdot P_{2,1,2})$

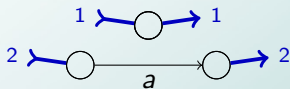


Generator examples

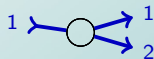
$(\mathbf{1}_2)$



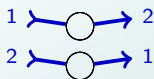
$(T_{2,a})$



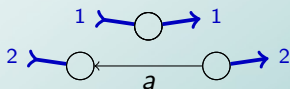
(F_2)



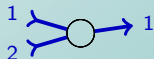
$(P_{2,1,2})$



$(T_{2,\bar{a}})$

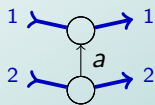


(J_2)

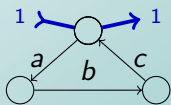


Some products of generators

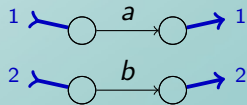
$(T_{2,a} \cdot J_2 \cdot F_2 \cdot T_{2,\bar{a}})$



$(F_2 \cdot T_{2,a} \cdot T_{2,b} \cdot T_{2,c} \cdot J_2)$



$(T_{2,b} \cdot P_{2,1,2} \cdot T_{2,a} \cdot P_{2,1,2})$



The particular case of birooted trees

Simplest generators



with $a^R = a \cdot a^{-1}$ and $a^L = a^{-1} \cdot a$

Induced monoid

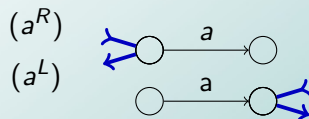
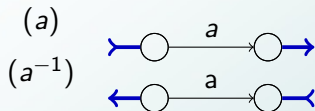
Generated elements are finite **birooted trees**.



They form the **free inverse monoid** generated by A .

The particular case of birooted trees

Simplest generators and left and right **projections**



with $a^R = a \cdot a^{-1}$ and $a^L = a^{-1} \cdot a$

Induced monoid

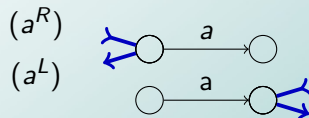
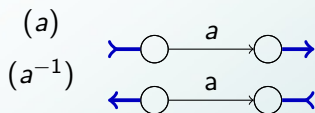
Generated elements are finite **birooted trees**.



They form the free inverse monoid generated by A .

The particular case of birooted trees

Simplest generators and left and right **projections**



with $a^R = a \cdot a^{-1}$ and $a^L = a^{-1} \cdot a$

Induced monoid

Generated elements are finite **birooted trees**.



They form the free inverse monoid generated by A .

The particular case of birooted trees

Simplest generators and left and right **projections**



with $a^R = a \cdot a^{-1}$ and $a^L = a^{-1} \cdot a$

Induced monoid

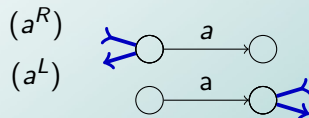
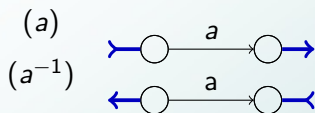
Generated elements are finite **birooted trees**.



They form the free inverse monoid generated by A .

The particular case of birooted trees

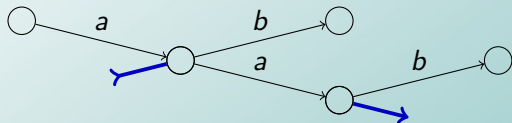
Simplest generators and left and right **projections**



with $a^R = a \cdot a^{-1}$ and $a^L = a^{-1} \cdot a$

Induced monoid

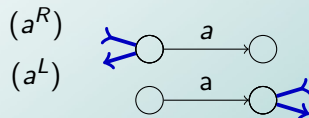
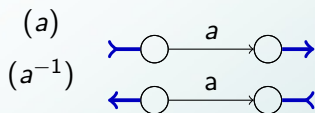
Generated elements are finite **birooted trees**.



They form the free inverse monoid generated by A .

The particular case of birooted trees

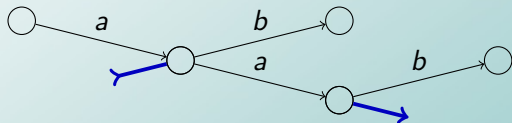
Simplest generators and left and right **projections**



with $a^R = a \cdot a^{-1}$ and $a^L = a^{-1} \cdot a$

Induced monoid

Generated elements are finite **birooted trees**.



They form the free **inverse monoid** generated by A .

Inverse monoid properties: idempotents

Lemma (Characterization)

A birooted graph $B = \langle in : I_p \rightarrow G, out : I_q \rightarrow G \rangle$ is idempotent if and only if $in = out$ (hence $p = q$).



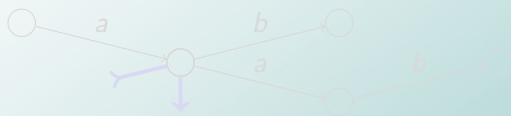
Lemma (Semi-lattice property)

Idempotents commute hence form a semi-lattice with $B_1 \wedge B_2 = B_1 \cdot B_2$.

Inverse monoid properties: idempotents

Lemma (Characterization)

A birooted graph $B = \langle in : I_p \rightarrow G, out : I_q \rightarrow G \rangle$ is idempotent if and only if $in = out$ (hence $p = q$).



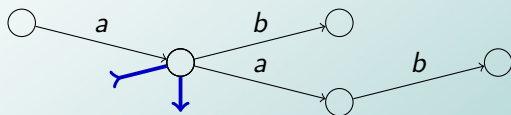
Lemma (Semi-lattice property)

Idempotents commute hence form a semi-lattice with $B_1 \wedge B_2 = B_1 \cdot B_2$.

Inverse monoid properties: idempotents

Lemma (Characterization)

A birooted graph $B = \langle in : I_p \rightarrow G, out : I_q \rightarrow G \rangle$ is idempotent if and only if $in = out$ (hence $p = q$).



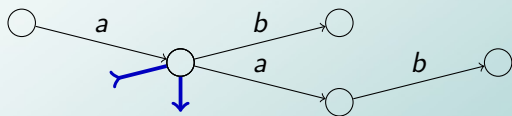
Lemma (Semi-lattice property)

Idempotents commute hence form a semi-lattice with $B_1 \wedge B_2 = B_1 \cdot B_2$.

Inverse monoid properties: idempotents

Lemma (Characterization)

A birooted graph $B = \langle in : I_p \rightarrow G, out : I_q \rightarrow G \rangle$ is idempotent if and only if $in = out$ (hence $p = q$).



Lemma (Semi-lattice property)

Idempotents commute hence form a semi-lattice with $B_1 \wedge B_2 = B_1 \cdot B_2$.

Inverse monoid properties: semigroup inverses

Lemma (Semigroup Inverse)

Take $B = \langle in : I_p \rightarrow G, out : I_q \rightarrow G \rangle$. Take $C = \langle out : I_q \rightarrow G, in : I_p \rightarrow G \rangle$. Then $B = B \cdot C \cdot B$ and $C = C \cdot B \cdot C$.



That is, C is a **semigroup inverse** of B .

And, since idempotents commute:

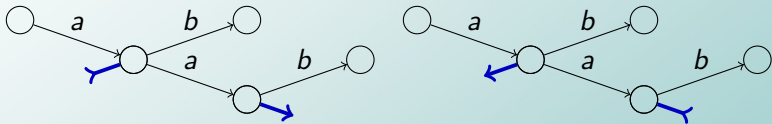
Corollary

The monoid of higher dimensional strings is an **red inverse monoid**, i.e. every element B has a unique inverse B^{-1} .

Inverse monoid properties: semigroup inverses

Lemma (Semigroup Inverse)

Take $B = \langle in : I_p \rightarrow G, out : I_q \rightarrow G \rangle$. Take $C = \langle out : I_q \rightarrow G, in : I_p \rightarrow G \rangle$. Then $B = B \cdot C \cdot B$ and $C = C \cdot B \cdot C$.



That is, C is a **semigroup inverse** of B .

And, since idempotents commute:

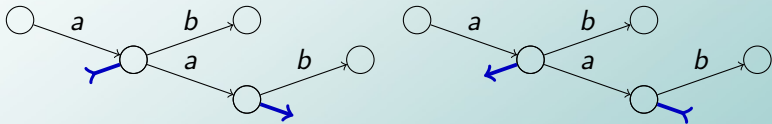
Corollary

The monoid of higher dimensional strings is an red inverse monoid, i.e. every element B has a unique inverse B^{-1} .

Inverse monoid properties: semigroup inverses

Lemma (Semigroup Inverse)

Take $B = \langle in : I_p \rightarrow G, out : I_q \rightarrow G \rangle$. Take $C = \langle out : I_q \rightarrow G, in : I_p \rightarrow G \rangle$. Then $B = B \cdot C \cdot B$ and $C = C \cdot B \cdot C$.



That is, C is a **semigroup inverse** of B .

And, since idempotents commute:

Corollary

The monoid of higher dimensional strings is an **red inverse monoid**, i.e. every element B has a **unique inverse** B^{-1} .

Inverse monoid properties: natural order and morphisms

Definition (Natural order)

$B_1 \leq B_2$ when $B_1 = B_1 \cdot B_1^{-1} \cdot B_2$.



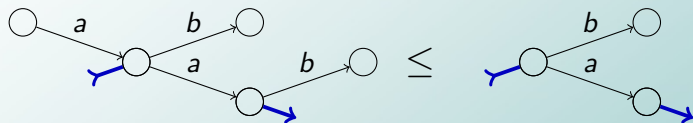
Lemma

Then $B_1 \leq B_2$ if and only if
there is a root preserving morphism $\varphi : B_2 \rightarrow B_1$.

Inverse monoid properties: natural order and morphisms

Definition (Natural order)

$B_1 \leq B_2$ when $B_1 = B_1 \cdot B_1^{-1} \cdot B_2$.



Lemma

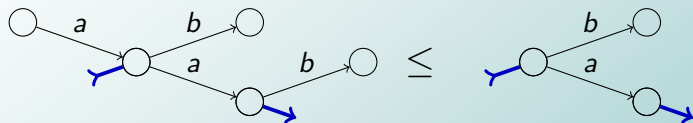
Then $B_1 \leq B_2$ if and only if

there is a root preserving morphism $\varphi : B_2 \rightarrow B_1$.

Inverse monoid properties: natural order and morphisms

Definition (Natural order)

$B_1 \leq B_2$ when $B_1 = B_1 \cdot B_1^{-1} \cdot B_2$.



Lemma

Then $B_1 \leq B_2$ if and only if

there is a root preserving morphism $\varphi : B_2 \rightarrow B_1$.

What about loops ?

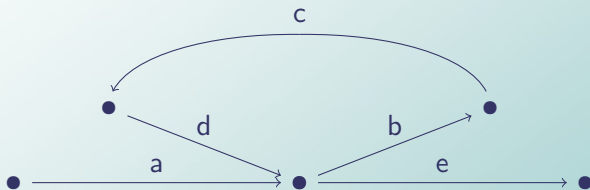
Directed cycles in graphs



could be forbidden with causal/temporal semantics of arrows.

What about loops ?

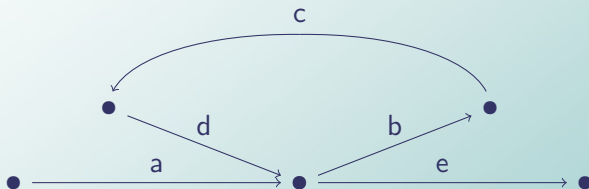
Directed cycles in graphs



could be forbidden with causal/temporal semantics of arrows.

What about loops ?

Directed cycles in graphs



could be forbidden with **causal/temporal semantics** of arrows.

Causal higher dimensional strings

Remark

The property “there is a cycle” is invariant under product.
These birooted graphs form a semigroup ideal \perp .

Quotient by ideal

By quotient (by ideal) all non causal element of \perp collapse to zero:
the forbidden model.

Definition

Causal $HDS(A) = HDS(A)/\perp$

Causal higher dimensional strings

Remark

The property “there is a cycle” is invariant under product.
These birooted graphs form a semigroup ideal \perp .

Quotient by ideal

By quotient (by ideal) all non causal element of \perp collapse to zero:
the forbidden model.

Definition

Causal $HDS(A) = HDS(A)/\perp$

Causal higher dimensional strings

Remark

The property “there is a cycle” is invariant under product.
These birooted graphs form a semigroup ideal \perp .

Quotient by ideal

By quotient (by ideal) all non causal element of \perp collapse to zero:
the forbidden model.

Definition

Causal $HDS(A) = HDS(A)/\perp$

What about language theory of HDS ?

We shall see that:

- ▶ grids are finitely generated !
- ▶ hence emptiness of MSO definable languages of HDS is undecidable,
- ▶ but disjoint products allows to define large classes of languages with decidable cases.

What about language theory of HDS ?

We shall see that:

- ▶ grids are finitely generated !
- ▶ hence emptiness of MSO definable languages of HDS is undecidable,
- ▶ but disjoint products allows to define large classes of languages with decidable cases.

What about language theory of HDS ?

We shall see that:

- ▶ grids are finitely generated !
- ▶ hence emptiness of MSO definable languages of HDS is undecidable,
- ▶ but disjoint products allows to define large classes of languages with decidable cases.

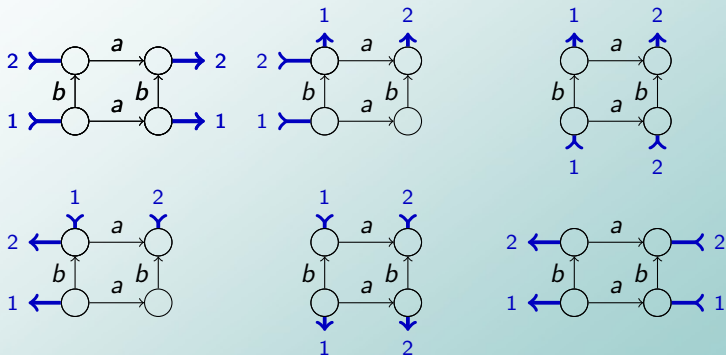
What about language theory of HDS ?

We shall see that:

- ▶ grids are finitely generated !
- ▶ hence emptiness of MSO definable languages of HDS is undecidable,
- ▶ but disjoint products allows to define large classes of languages with decidable cases.

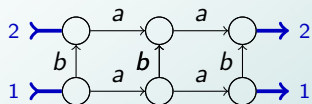
Higher-dimensionality and related undecidability

From squares to grids



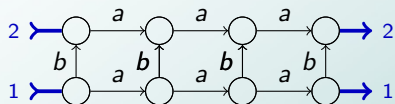
Higher-dimensionality and related undecidability

From squares to grids



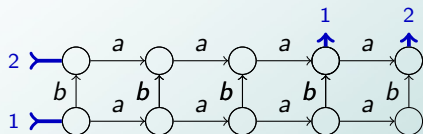
Higher-dimensionality and related undecidability

From squares to grids



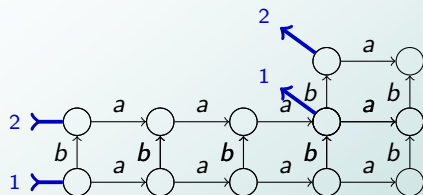
Higher-dimensionality and related undecidability

From squares to grids



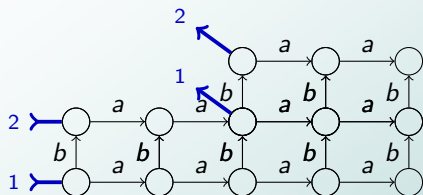
Higher-dimensionality and related undecidability

From squares to grids



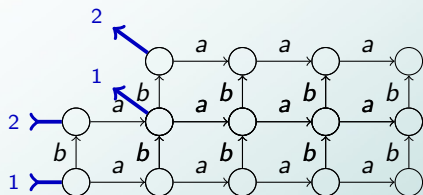
Higher-dimensionality and related undecidability

From squares to grids



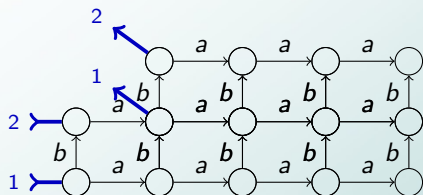
Higher-dimensionality and related undecidability

From squares to grids



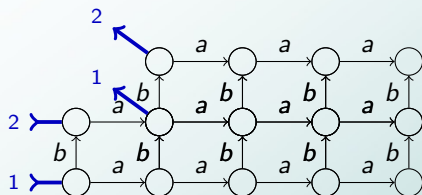
Higher-dimensionality and related undecidability

From squares to grids



Higher-dimensionality and related undecidability

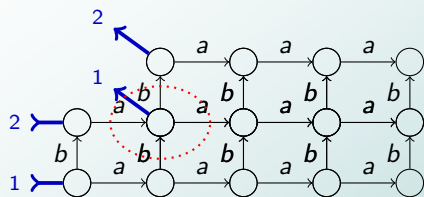
From squares to grids



As a consequence, MSO languages of HDS have undecidable emptiness.

Higher-dimensionality and related undecidability

From squares to grids



Such a node “clashes” in the gluing !

Undecidability comes from complex gluing !

Controlled glueing: disjoint product

Definition

A product is a **disjoint product** when it is limited to **roots glueing**, i.e. trivial fusion phase.

Viewing a disjoint product



Controlled glueing: disjoint product

Definition

A product is a **disjoint product** when it is limited to roots glueing, i.e. trivial fusion phase.

Viewing a disjoint product

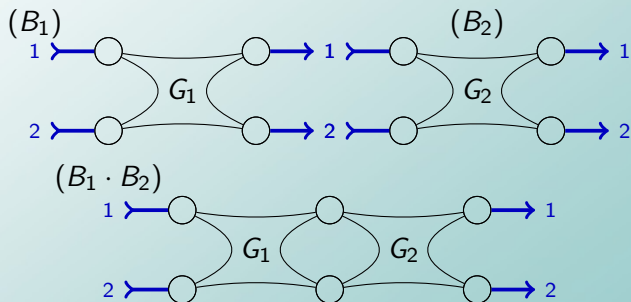


Controlled glueing: disjoint product

Definition

A product is a **disjoint product** when it is limited to roots glueing, i.e. trivial fusion phase.

Viewing a disjoint product



Disjoint product and partial algebra

Lemma

*Though partial, the disjoint product is **associative**.*

Corollary

Partial algebra techniques [Bur86] and MSO type theory [She75] apply leading to a decidable, expressive and efficient MSO language theory.

The case of **birooted trees**

detailed in [BJ14, Jan15] with disjoint products and projections.

Disjoint product and partial algebra

Lemma

*Though partial, the disjoint product is **associative**.*

Corollary

Partial algebra techniques [Bur86] and MSO type theory [She75] apply leading to a decidable, expressive and efficient MSO language theory.

The case of brooted trees

detailed in [BJ14, Jan15] with disjoint products and projections.

Disjoint product and partial algebra

Lemma

*Though partial, the disjoint product is **associative**.*

Corollary

Partial algebra techniques [Bur86] and MSO type theory [She75] apply leading to a decidable, expressive and efficient MSO language theory.

The case of **birooted trees**

detailed in [BJ14, Jan15] with disjoint products and projections.

Disjoint product and partial algebra

Lemma

*Though partial, the disjoint product is **associative**.*

Corollary

Partial algebra techniques [Bur86] and MSO type theory [She75] apply leading to a decidable, expressive and efficient MSO language theory.

The case of birooted trees

detailed in [BJ14, Jan15] with disjoint products and projections.

Disjoint product and graphs of bounded tree-width

Lemma

The tree-width of a disjoint product is bounded by the max. tree-width of the components (roots seen as hyper-edges).

Corollary

MSO model checking techniques over graphs of bounded tree-width also apply [CE12].

Remark

In disjoint product, the number of roots strictly controls the amount of gluing.

Disjoint product and graphs of bounded tree-width

Lemma

The tree-width of a disjoint product is bounded by the max. tree-width of the components (roots seen as hyper-edges).

Corollary

MSO model checking techniques over graphs of bounded tree-width also apply [CE12].

Remark

In disjoint product, the number of roots strictly controls the amount of gluing.

Disjoint product and graphs of bounded tree-width

Lemma

The tree-width of a disjoint product is bounded by the max. tree-width of the components (roots seen as hyper-edges).

Corollary

MSO model checking techniques over graphs of bounded tree-width also apply [CE12].

Remark

In disjoint product, the number of roots strictly controls the amount of gluing.

Disjoint product and graphs of bounded tree-width

Lemma

The tree-width of a disjoint product is bounded by the max. tree-width of the components (roots seen as hyper-edges).

Corollary

MSO model checking techniques over graphs of bounded tree-width also apply [CE12].

Remark

In disjoint product, the number of roots strictly controls the amount of gluing.

Conclusion

Done

- ▶ **simple monoids** for large classes of (birooted) graphs,
- ▶ an underlying rich and robust theory : **inverse semigroups**,
- ▶ language theory and tools via **non-det. automata** [Jan15], **partial algebra** [Bur86, BJ14], and also **walking automata** [Jan16], etc. . . .

Ongoing

- ▶ (fun) application in **music system programming tools**,
- ▶ causal and I/O **modeling experiments**,

To be scheduled

- ▶ **beyond music**: multi-scale reactive hybrid systems modeling,
- ▶ **connection with existing (and used) refinement methods**,
- ▶ **development of a visual modeling/programming tool**.

Conclusion

Done

- ▶ **simple monoids** for large classes of (birooted) graphs,
- ▶ an underlying rich and robust theory : **inverse semigroups**,
- ▶ language theory and tools via **non-det. automata** [Jan15], **partial algebra** [Bur86, BJ14], and also **walking automata** [Jan16], etc...

Ongoing

- ▶ (fun) application in **music system programming tools**,
- ▶ causal and I/O **modeling experiments**,

To be scheduled

- ▶ **beyond music**: multi-scale reactive hybrid systems modeling,
- ▶ **connection with existing (and used) refinement methods**,
- ▶ **development of a visual modeling/programming tool**

Conclusion

Done

- ▶ **simple monoids** for large classes of (birooted) graphs,
- ▶ an underlying rich and robust theory : **inverse semigroups**,
- ▶ language theory and tools via **non-det. automata** [Jan15], **partial algebra** [Bur86, BJ14], and also **walking automata** [Jan16], etc. . . . ,

Ongoing

- ▶ (fun) application in **music system programming tools**,
- ▶ causal and I/O **modeling experiments**,

To be scheduled

- ▶ **beyond music**: multi-scale reactive hybrid systems modeling,
- ▶ **connection with existing (and used) refinement methods**,
- ▶ **development of a visual modeling/programming tool**.

Conclusion

Done

- ▶ **simple monoids** for large classes of (biregular) graphs,
- ▶ an underlying rich and robust theory : **inverse semigroups**,
- ▶ language theory and tools via **non-det. automata** [Jan15], **partial algebra** [Bur86, BJ14], and also **walking automata** [Jan16], etc. . . . ,

Ongoing

- ▶ (fun) application in **music system** programming tools,
- ▶ causal and I/O **modeling experiments**,

To be scheduled

- ▶ **beyond music**: multi-scale reactive hybrid systems modeling,
- ▶ connection with existing (and used) refinement methods,
- ▶ development of a visual modeling/programming tool.

Conclusion

Done

- ▶ **simple monoids** for large classes of (biregular) graphs,
- ▶ an underlying rich and robust theory : **inverse semigroups**,
- ▶ language theory and tools via **non-det. automata** [Jan15], **partial algebra** [Bur86, BJ14], and also **walking automata** [Jan16], etc. . . . ,

Ongoing

- ▶ (fun) application in **music system** programming tools,
- ▶ causal and I/O **modeling experiments**,

To be scheduled

- ▶ beyond music: **multi-scale reactive hybrid systems** modeling,
- ▶ connection with existing (and used) **refinement methods**,
- ▶ development of a **visual modeling/programming tool**.

For a structured programming of time and space...



Thanks for your attention !

- [BJ14] A. Blumensath and D. Janin.
A syntactic congruence for languages of birooted trees.
Semigroup Forum, 2014.
- [Bur86] P. Burmeister.
A Model Theoretic Oriented Approach to Partial Algebras.
Akademie-Verlag, 1986.
- [CE12] B. Courcelle and J. Engelfriet.
Graph structure and monadic second-order logic, a language theoretic approach, volume 138 of *Encyclopedia of mathematics and its applications*.
Cambridge University Press, 2012.
- [Jan15] D. Janin.
On labeled birooted trees languages: Algebras, automata and logic.
Information and Computation, 243:222–248, 2015.
- [Jan16] D. Janin.

Walking automata in free inverse monoids.

In *Int. Conf. on Current Trends in Theo. and Prac. of Comp. Science (SOFSEM)*, 2016.

[She75] S. Shelah.

The monadic theory of order.

Annals of Mathematics, 102:379–419, 1975.