On the Recognizability of Arrow and Graph Languages

Christoph Blume Sander Bruggink **Barbara König**
Universität Duisburg-Essen, Germany

Background

Applications of finite automata and regular (word) languages are abundant in computer science.

What about regular/recognizable *graph* languages?

There exists an established notion of of recognizable graph languages by Courcelle. Our contribution:

- A categorical notion of recognizability (not just for graphs, but for arbitrary categories). It coincides with Courcelle's notion if we work in *Cospan*(**Graph**).
- A notion of recognizable graph language which works well with (double-pushout) graph transformation.
- A notion of graph automaton (= automaton functor).
- Some preliminary experiments on an implementation of such automata. Long-term goal: a graph automata tool suite.

Background

Why are we interested?

Potential applications of recognizable graph languages in verification:

- Proving termination of graph transformation systems (GTSs).
  Current termination checkers for string rewriting use regular languages. We need a comparable notion for graphs.
- Verifying invariants of GTSs.
  We need a convenient method to describe properties of graphs.
- Regular model checking for GTSs.

Plan

- *Preliminaries*: category theory and graph transformation
- *Abstract notion of recognizability*: recognizing languages of *arrows* in a category
- *Recognizing graphs*: we apply the abstract notion to the category of cospans of graphs, in order to recognize graphs with interfaces
- *Some thoughts on implementation*
- *Comparison to Courcelle's notion and to other related work*
- *Conclusion and Future Work*

Cospans

- A *cospan* is a pair of arrows with the same codomain:

$$J \rightarrow G \leftarrow K$$

- Cospan composition:

$$\begin{tikzcd} & & M \arrow[dl, "c_{\mathrm{R}}"'] \arrow[dr, "d_{\mathrm{L}}"] & & \\ J \arrow[r, "c_{\mathrm{L}}"] & G \arrow[dr, "f"'] & (\mathrm{PO}) & H \arrow[dl, "g"] & K \arrow[l, "d_{\mathrm{R}}"'] \\ & & M' & & \end{tikzcd}$$

- Cospan category $Cospan(\mathbf{C})$: the same objects as $\mathbf{C}$, (equivalence classes of) cospans as arrows, pairs of $\mathbf{C}$-identities as identities and cospan composition as composition.

Graph transformation

- *Hypergraph*: $G = \langle V, E, \mathrm{att}, \mathrm{lab}\rangle$, where
  - V is the set of nodes and E the set of edges;
  - $\mathrm{att}\colon E \to V^*$ the attachment function; and
  - $\mathrm{lab}\colon E \to \Sigma$ the labelling function.
- *Morphism* from G to H:
  structure preserving map from the nodes and edges of G to the nodes and edges of H.
- *Graph transformation with cospans*:
  Rules are pairs $\rho = \langle l, r\rangle$ with cospans $l\colon \varnothing \to L \leftarrow K$, $r\colon \varnothing \to R \leftarrow K$.
  $G \Rightarrow_{\rho,m} H$ if there is a cospan $c\colon K \to C \leftarrow \varnothing$ and

  $$[G] = l \mathbin{;} c \text{ and } [H] = r \mathbin{;} c$$

  (where $[Q] := \varnothing \to Q \leftarrow \varnothing$).
  This is equivalent to double-pushout graph rewriting!

Recognizing arrows

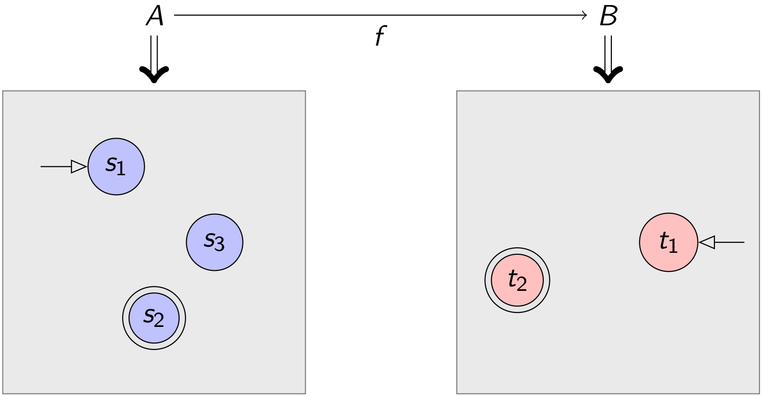
Definition (Nondeterministic Automaton Functor)

- Automaton functor: $\mathcal{A}\colon \mathbf{C} \to \mathbf{Rel}_{\mathrm{fin}}$.
- Each object of $\mathbf{C}$ is mapped to a finite set of states.
  Each set of states is equipped with a subset of start states and a subset of end states.
- Each arrow of $\mathbf{C}$ is mapped to a relation between states.
- The automaton functor $\mathcal{A}$ accepts an arrow from $c\colon I \to J$ if $\mathcal{A}(c)$ relates a start state of $\mathcal{A}(I)$ to an end state of $\mathcal{A}(J)$.
- An automaton functor is deterministic, if each set of start states is a singleton, and each relation is a function.

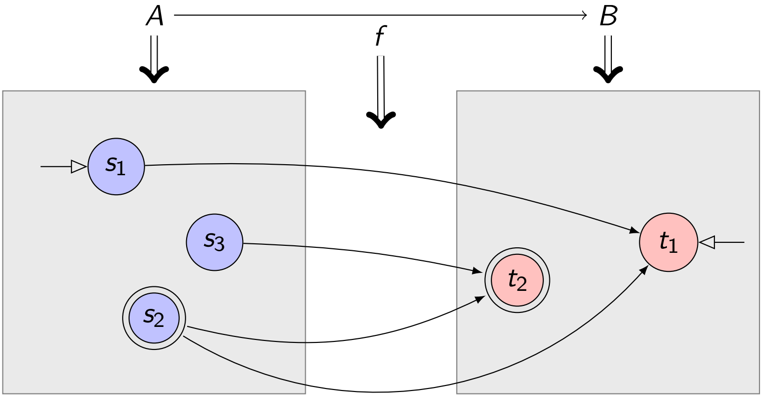
Recognizing arrows (2)

$$A \xrightarrow[f]{} B$$

Recognizing arrows (2)

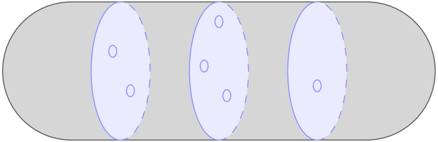


Recognizing arrows (2)

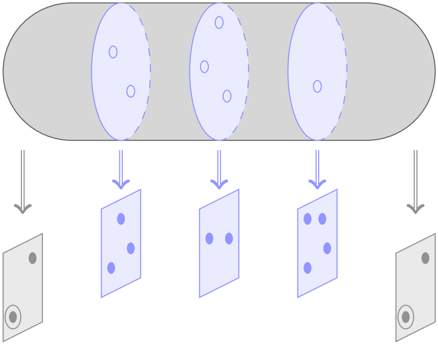


Abstract intuition

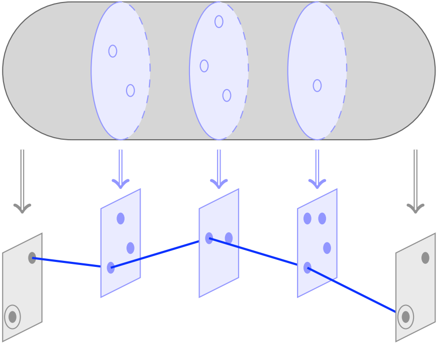
Abstract intuition



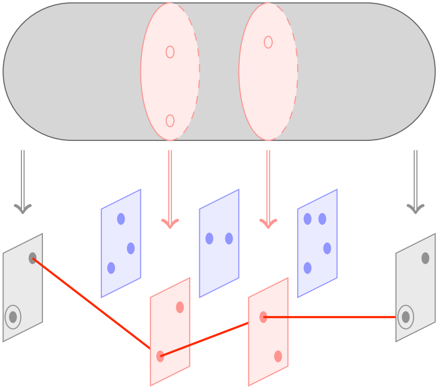
Abstract intuition



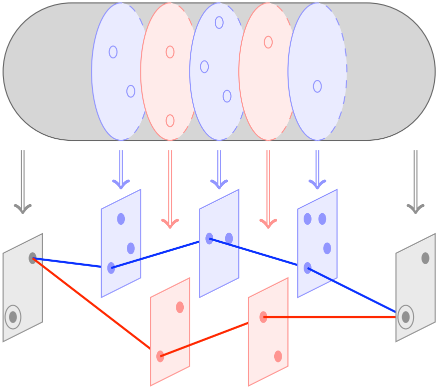
Abstract intuition



Abstract intuition



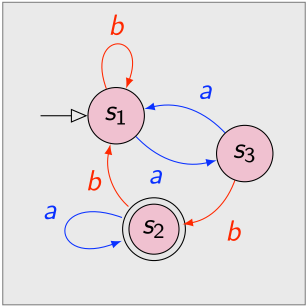
Abstract intuition



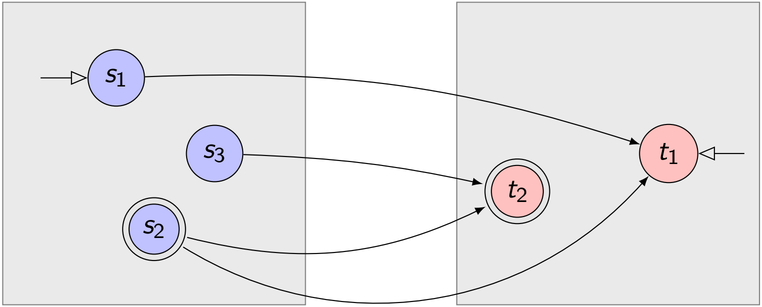
Example: Finite automata

Category $\mathbf{C}$: a single object, arrows are all words from Σ^* (where Σ is a fixed alphabet)

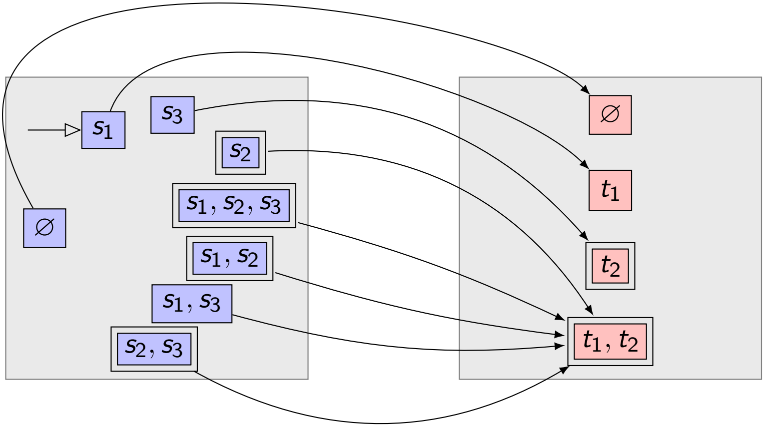
Functor corresponds to the transition function of the automaton:
$\hat{\delta}(z, vw) = \hat{\delta}(\hat{\delta}(z, v), w)$



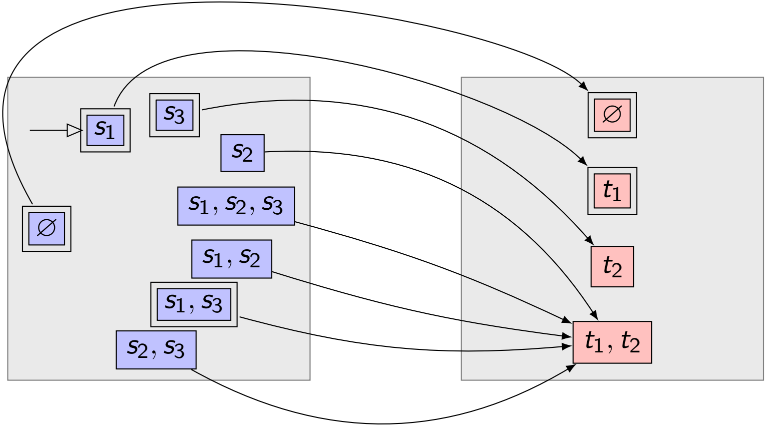
Determinization, closure, minimization



Determinization, closure, minimization



Determinization, closure, minimization



Determinization, closure, minimization

Theorem

For each automaton functor, there exists a deterministic automaton functor which recognizes the same language.

Theorem

The class of recognizable languages is closed under complement, union and intersection.

Theorem

For each automaton functor, there exists a unique minimal automaton functor which recognizes the same language.

Recognizability by congruences

Let an equivalence relation $\equiv_R$ be given, defined only on arrows with the same domain and codomain.

The relation $\equiv_R$ is called *locally finite*, if for each pair of objects, there are finitely many equivalence classes of arrows between them.

It is called a *congruence* if the following holds:

$$a \equiv_R a' \text{ implies } (a\,;\,b) \equiv_R (a'\,;\,b) \quad (\text{for all } b)$$

Theorem

A language $L_{J,K}$ of arrows from J to K is recognizable iff $L_{J,K}$ is the union of some equivalence classes of a locally finite congruence.

Recognizing graphs

Definition

Let $[G] := \varnothing \to G \leftarrow \varnothing$.
A language L of graphs is recognizable whenever

$$L' := \{[G] \mid G \in L\}$$

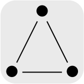
is a recognizable language in $\mathit{Cospan}(\mathbf{Graph})$.

Example: k-colorability

The class of k-colorable graphs is recognizable.

$$J \longrightarrow G \longleftarrow K$$

Example:



colorings of J colorings of G colorings of K

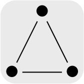
relate colorings of J to *compatible* colorings of K

Example: k-colorability

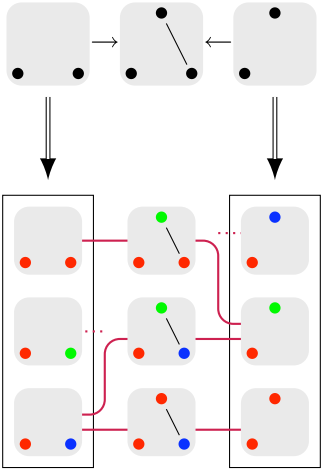
The class of k-colorable graphs is recognizable.

$$\begin{tikzcd} J \arrow[r] \arrow[d, Rrightarrow] & G \arrow[d, Rrightarrow] & K \arrow[l] \arrow[d, Rrightarrow] \\ \text{colorings of } J \arrow[rr, dash, bend right, "\text{relate colorings of } J \text{ to \textit{compatible} colorings of } K"'] & \text{colorings of } G & \text{colorings of } K \end{tikzcd}$$

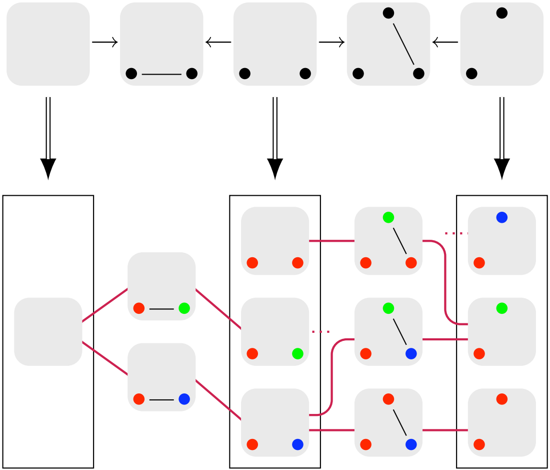
Example:



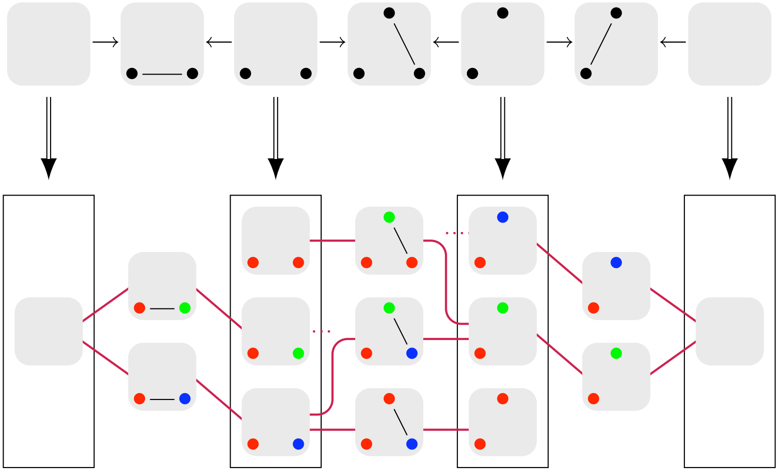
Example: k-colorability (2)



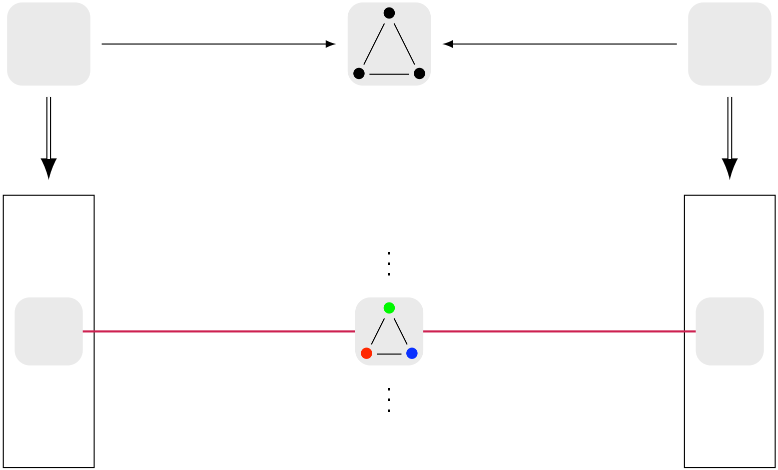
Example: k-colorability (2)



Example: k-colorability (2)



Example: k-colorability (2)



Robustness result

Let $\mathbf{CG} = \mathit{Cospan}(\mathbf{Graph})$ and let $\mathbf{CG}_{\mathsf{dis}}$ be its subcategory consisting of cospans with discrete interfaces only.

Theorem

Let a class $L_{J,K}$ of graphs with discrete interfaces J, K be be called discretely recognizable when $L_{J,K}$ is recognizable in $\mathbf{CG}_{dis}$.

Then $L_{J,K}$ is recognizable in $\mathbf{CG}$ if and only if it is discretely recognizable.

Implementation of automaton functors

Is it realistic to use automaton functors in practice?

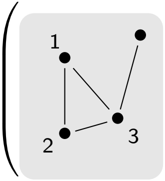
- Use only discrete interfaces (0 nodes, 1 node, 2 nodes, etc.)
- Restrict the interface size, this means that only graphs up to a certain pathwidth can be recognized (extension to treewidth)

Still: the size of state sets for the discrete interfaces usually grows exponentially.
After all, k-colorability is an NP-complete problem. (Its complexity is however linear if we restrict to graphs of bounded treewidth – Courcelle's theorem.)

Our plan: fight state space explosion with the favourite weapon of model-checkers – represent automaton functors by binary decision diagrams (BDDs). (Our recent results are very encouraging, but there's still a lot to do.)

Courcelle's algebra of graphs

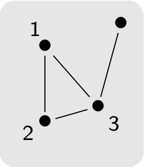
- *Carrier set*: Graphs with a number of *external* nodes (called *n-ary graphs*).
- *Operations*:
  - *Redefinition*. The external nodes are given new names.
  - *Fusion*. Given an equivalence relation on the external nodes, fuse the nodes which are equivalent to each other.
  - *Disjoint union*.



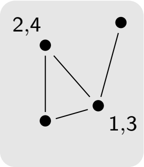
Courcelle's algebra of graphs

- *Carrier set*: Graphs with a number of *external* nodes (called *n-ary graphs*).
- *Operations*:
  - *Redefinition*. The external nodes are given new names.
  - *Fusion*. Given an equivalence relation on the external nodes, fuse the nodes which are equivalent to each other.
  - *Disjoint union*.

redef (



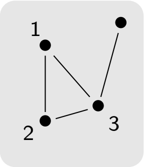
$$, \begin{pmatrix} 1 \mapsto 3, & 3 \mapsto 3 \\ 2 \mapsto 1, & 4 \mapsto 1 \end{pmatrix} \Bigg) =$$



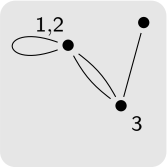
Courcelle's algebra of graphs

- *Carrier set*: Graphs with a number of *external* nodes (called *n-ary graphs*).
- *Operations*:
  - *Redefinition*. The external nodes are given new names.
  - *Fusion*. Given an equivalence relation on the external nodes, fuse the nodes which are equivalent to each other.
  - *Disjoint union*.

fuse (

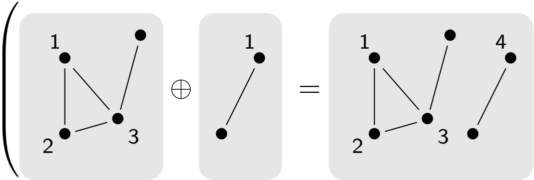


, $\left\{ \begin{matrix} \{1,2\} \\ \{3\} \end{matrix} \right\}$) =



Courcelle's algebra of graphs

- *Carrier set*: Graphs with a number of *external* nodes (called *n-ary graphs*).
- *Operations*:
  - *Redefinition*. The external nodes are given new names.
  - *Fusion*. Given an equivalence relation on the external nodes, fuse the nodes which are equivalent to each other.
  - *Disjoint union*.



Recognizability *á la* Courcelle

Definition

Let $\equiv_C$ be an equivalence relation defined on graphs with the same arity. It is *locally finite* if for each arity there are finitely many equivalence classes, and it is called a *congruence* if it respects the operations mentioned above.

A language L of n-ary graphs is Courcelle-recognizable if it is the union of (finitely many) equivalence classes of a locally finite congruence.

This notion of recognizability is equivalent to ours!

Comparing cospans and Courcelle-graphs

Differences between cospans and Courcelle graphs:

1. Cospans can have arbitrary graphs as interfaces, Courcelle's interfaces consist only of nodes.
2. Cospans have two interfaces, Courcelle's graphs only one.

Ad 2: we introduce a function bend$(\cdot)$ which "bends the interfaces together":

Comparing cospans and Courcelle-graphs

Differences between cospans and Courcelle graphs:

1. Cospans can have arbitrary graphs as interfaces, Courcelle's interfaces consist only of nodes.
2. Cospans have two interfaces, Courcelle's graphs only one.

Ad 1: solved by robustness result.

Ad 2: we introduce a function bend$(\cdot)$ which "bends the interfaces together":

Comparing cospans and Courcelle-graphs

Differences between cospans and Courcelle graphs:

1. Cospans can have arbitrary graphs as interfaces, Courcelle's interfaces consist only of nodes.
2. Cospans have two interfaces, Courcelle's graphs only one.

Ad 2: we introduce a function bend$(\cdot)$ which "bends the interfaces together":

$$J \longrightarrow G \longleftarrow K$$

Comparing cospans and Courcelle-graphs

Differences between cospans and Courcelle graphs:

1. Cospans can have arbitrary graphs as interfaces, Courcelle's interfaces consist only of nodes.
2. Cospans have two interfaces, Courcelle's graphs only one.

Ad 2: we introduce a function bend$(\cdot)$ which "bends the interfaces together":

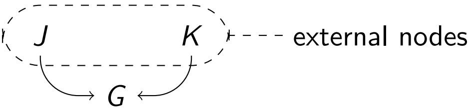
$$\begin{tikzcd} J \arrow[dr, bend right, hook] & & K \arrow[dl, bend left, hook'] \\ & G & \end{tikzcd}$$

Comparing cospans and Courcelle-graphs

Differences between cospans and Courcelle graphs:

1. Cospans can have arbitrary graphs as interfaces, Courcelle's interfaces consist only of nodes.
2. Cospans have two interfaces, Courcelle's graphs only one.

Ad 2: we introduce a function bend$(\cdot)$ which "bends the interfaces together":



Comparing cospans and Courcelle-graphs

Differences between cospans and Courcelle graphs:

1. Cospans can have arbitrary graphs as interfaces, Courcelle's interfaces consist only of nodes.
2. Cospans have two interfaces, Courcelle's graphs only one.

Ad 2: we introduce a function bend($\cdot$) which "bends the interfaces together":

$$
\begin{tikzcd}
J \arrow[dr, bend right, hook] & & K \arrow[dl, bend left, hook'] \\
& G &
\end{tikzcd}
$$

external nodes

(compare with compact-closed categories)

Equivalence

Theorem

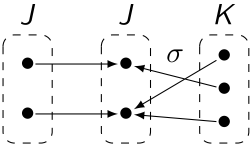
Let J be a discrete graph. A set of cospans of graphs L is the $(\varnothing, J)$-language of some automaton functor $\mathcal{A}$ if and only if $\mathrm{bend}(L)$ is Courcelle-recognizable.

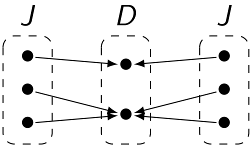
Proof.

$(\Rightarrow)$: Simulate Courcelle's operations by cospan compositions.

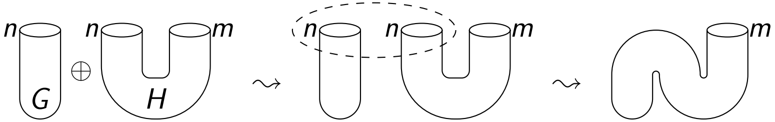
$(\Leftarrow)$: Simulate cospan compositions by Courcelle's operations.

Cospans $\Rightarrow$ Courcelle

$redef_{\sigma} =$ 

$fuse_{\theta} =$ 

Courcelle $\Rightarrow$ Cospans



Comparison to Related Work

- Courcelle (90's)
  Our work is a different view on Courcelle's notion of recognizability.
  Our notion is category-theory-based (as opposed to the algebraic notion of Courcelle) and focusses more on automata.
- Bozapalidis, Kalampakas: magmoids/graphoids (2006/2008)
  Another way to define recognizability and a notion of graph automata (weaker, but more efficient, than ours).
- Griffing: Composition-representative subsets (2003)
  Related approach that also uses functors (into a category with finite homsets)
- Arbib/Manes/Adámek/Trnková/Ehrig/...(70's): Regular Languages in a Category
  Generalizes the state set ($\rightsquigarrow$ object) and the transition function ($\rightsquigarrow$ arrow)

Comparison to Related Work

- Mezei/Wright/Eilenberg (60's): An algebraic notion of recognizability (algebra homomorphisms into multi-sorted algebras with finite carrier sets)
  Predecessor to Courcelle's work
- Habel/Kreowski/Lautemann (1993): A comparison of compatible, finite and inductive graph properties
- Context-free graph grammars (hyperedge replacement grammars): there are recognizable languages which are not context-free and vice versa $\rightsquigarrow$ no proper Chomsky hierarchy

Further research

- A coalgebraic view on automaton functors?
- Pathwidth vs. Treewidth: Allow tree decompositions of graphs
  $\leadsto$ use monoidal categories and functors?
- Implementation: we have a fairly naive prototype implementation which explicitly represents state sets and relations
  $\leadsto$ we currently switch to a BDD representation
- Generalization to Adhesive Categories: Courcelle's result that

  > *Every MSOL-definable graph language is recognizable*

  should work in that setting.
  (MSOL = monadic second-order logic)
- Verification: invariants ($\leadsto$ Master thesis of Christoph Blume), termination analysis, regular model-checking