Graph Specification Languages and Applications to the Verification of Graph Transformation Systems

Barbara König

Universität Duisburg-Essen, Germany

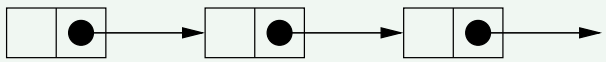
Joint work with Christoph Blume, Sander Bruggink, Raphaël Cauderlier, Mathias Hülsbusch

Overview

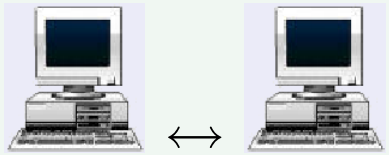
Dynamic Systems

Systems with a dynamic topology

- Heap, consisting of graph-like pointer structures



- Object-oriented systems
- Network protocols



- Mobile systems
  - Process mobility
  - Ubiquitous computing

Problems with Analysis und Verification

Problems

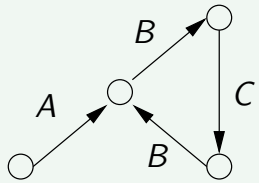
- Infinite state space
- Dynamic creation and deletion of objects
- Mobility
- Variable topology

$\rightsquigarrow$ suitable analysis and verification techniques are needed

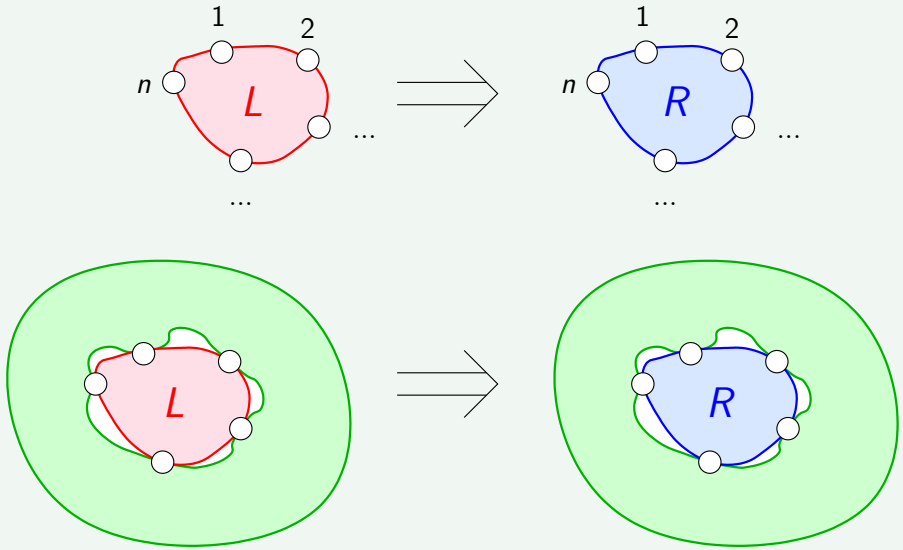
$\rightsquigarrow$ we focus on a simple, but expressive modelling language which can naturally describe dynamic systems: graph transformation systems

Graph Transformation Systems (GTS)

System state specified by edge-labelled directed graphs (in general: hypergraphs)



Dynamics: (Initial graph +) transformation rules



Graph Specification Languages

Specification languages (logics) are needed to describe sets of graphs.

There are several choices . . .

- First-order logics
- Monadic second-order logics
- Separation logic [O'Hearn]
- Various kinds of spatial logics (for instance for process calculi) [Cardelli]

Recently there has been a renewed interest in studying spatial logics, in addition to temporal logics

Logics (and related formalisms) are not only needed to specify properties, but are also used within verification algorithms.

Monadic Second-Order and First-Order Graph Logics

Monadic second-order graph logics may quantify over nodes, edges, node sets and edge sets.

$$\varphi := \varphi_1 \wedge \varphi_2 \mid \neg\varphi \mid (\exists X : V)\,\varphi \mid (\exists X : E)\,\varphi \mid (\exists x : v)\,\varphi \mid (\exists x : e)\,\varphi \mid$$
$$x = y \mid x \in X \mid edge_A(x, y_1, y_2)$$

Monadic Second-Order and First-Order Graph Logics

First-order graph logics may quantify over nodes, edges, but not over sets.

$$\varphi := \varphi_1 \wedge \varphi_2 \mid \neg\varphi \mid \cancel{(\exists X : V)\,\varphi} \mid \cancel{(\exists X : E)\,\varphi} \mid (\exists x : v)\,\varphi \mid (\exists x : e)\,\varphi \mid$$
$$x = y \mid \cancel{x \in X} \mid edge_A(x, y_1, y_2)$$

Monadic Second-Order and First-Order Graph Logics

Examples:

First-order:

- $(\exists x\colon v)\,(\exists y\colon v)\,(\exists z\colon e)\,(edge_A(z,x,y) \wedge x = y)$

  "There exists an A-labelled loop."

Second-order:

- $RC(X\colon V) = (\forall z\colon e)\,(\forall x\colon v)\,(\forall y\colon v)\,(x \in X \wedge edge_A(z,x,y) \rightarrow y \in X)$

  "The set of nodes X is closed under reachability over A-edges."
- $APath(x,y\colon v) = (\forall Z\colon V)\,(x \in Z \wedge RC(Z) \rightarrow y \in Z)$

  "There is an A-path from x to y: all sets of nodes Z that contain x and are closed under reachability contain y."

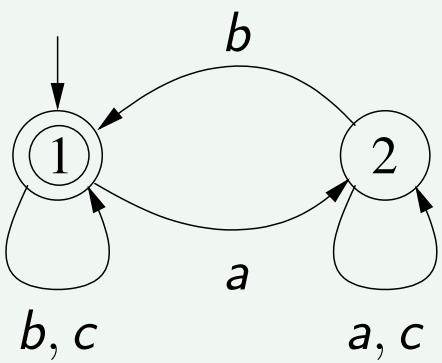
Regular Languages and Monadic Second-Order Logic

There is an intimate connection between formal languages of words and logics.

Theorem [Büchi, Elgot]

A language $L \subseteq \Sigma^*$ is regular if and only if it is expressible in monadic second-order logic on words.

Example:



$\Sigma = \{a, b, c\}$

$$\forall x(P_a(x) \rightarrow \exists y(x \leq y \wedge P_b(y)))$$

"For every position in the word with an a, there is a subsequent position in the word with a b."

Regular Languages and Monadic Second-Order Logic

Why is this interesting?

- Transforming a logical formula into an automaton transforms a specification into an algorithm (for deciding the membership problem).
- Automata are well-suited for answering the following questions:
  - Is the given regular language L empty? (unsatisfiability)
  - Is L_1 included in L_2: $L_1 \subseteq L_2$? (entailment)
  - Are L_1 and L_2 equal: $L_1 = L_2$? (equivalence)

Regular Languages and Monadic Second-Order Logic

What about graph logics and graph languages?

There is a notion of recognizable graph languages by Courcelle, which can either be defined via Myhill-Nerode style equivalences or via graph automata. More details will follow ...

Theorem (Courcelle)

Every graph language expressible in monadic second-order graph logic is recognizable.
(The other direction does not hold.)

Cospans of Graphs

Towards graph automata:

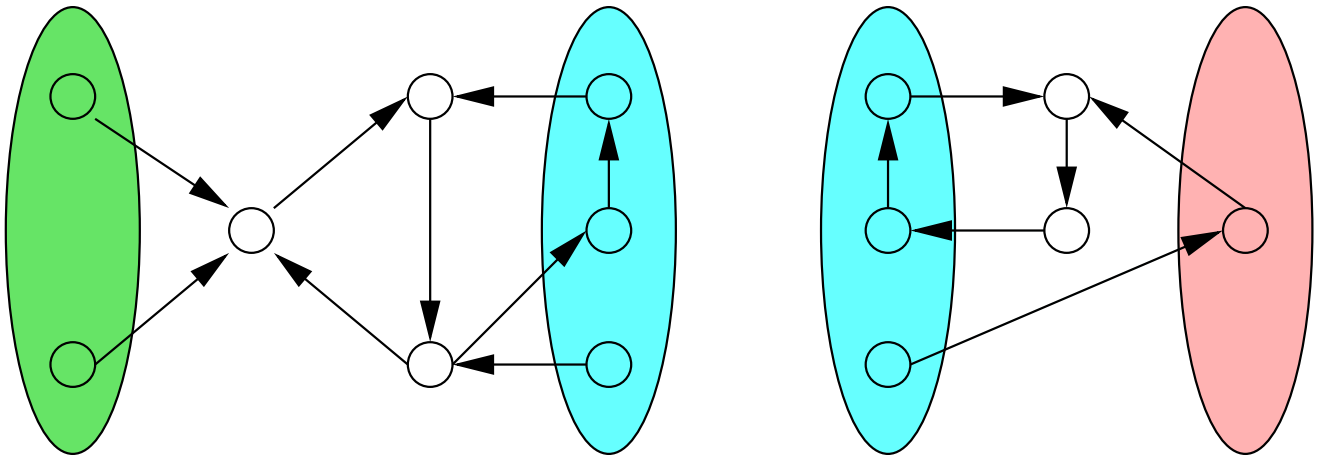
- Find appropriate decompositions of graphs in order to process them with automata.
- Split graphs into (atomic) building blocks (cospans).
- Atomic cospans play the role of alphabet symbols.
- Close relation to tree and path decompositions of graphs [Robertson, Seymour] [Bodlaender].

Cospans of Graphs

Cospans are graphs with an inner and an outer interface, composition is performed by gluing over the joint interface. (Cospan composition is denoted by ;.)

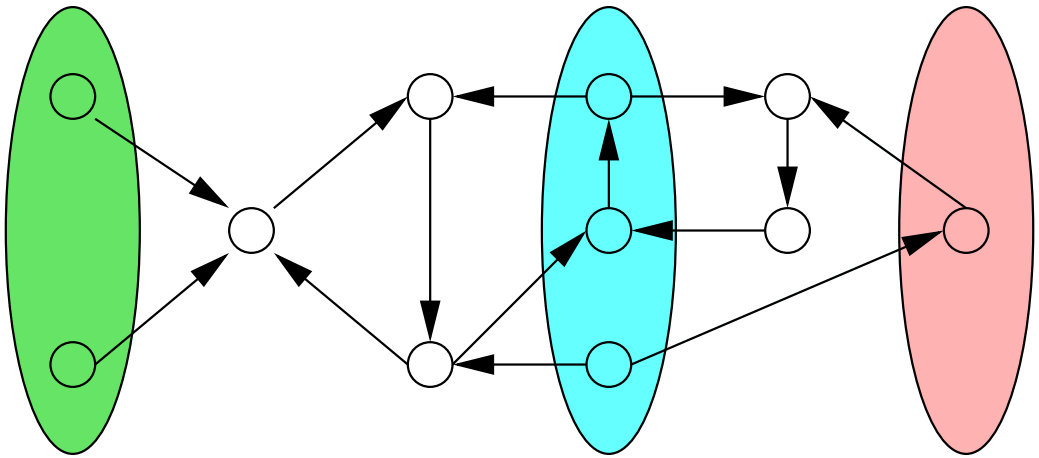
Cospans of Graphs

Cospans are graphs with an inner and an outer interface, composition is performed by gluing over the joint interface. (Cospan composition is denoted by ;.)



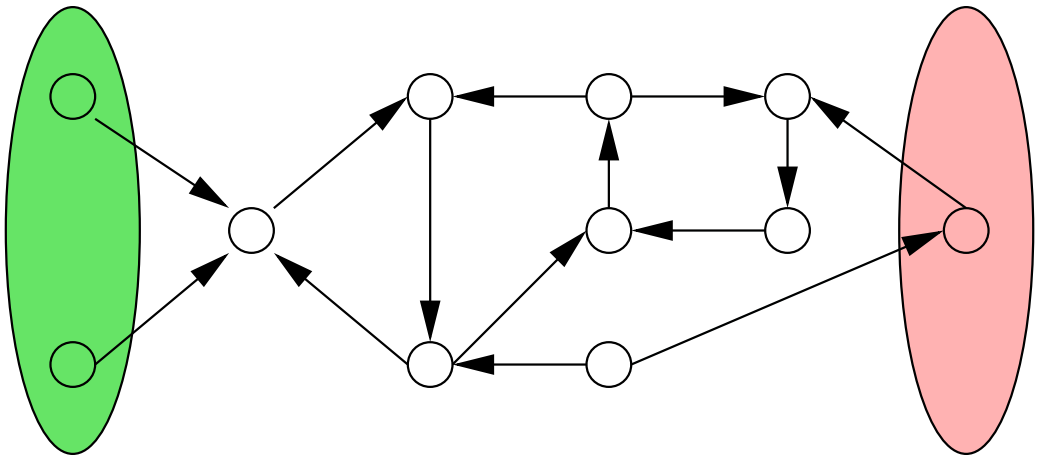
Cospans of Graphs

Cospans are graphs with an inner and an outer interface, composition is performed by gluing over the joint interface. (Cospan composition is denoted by ;.)



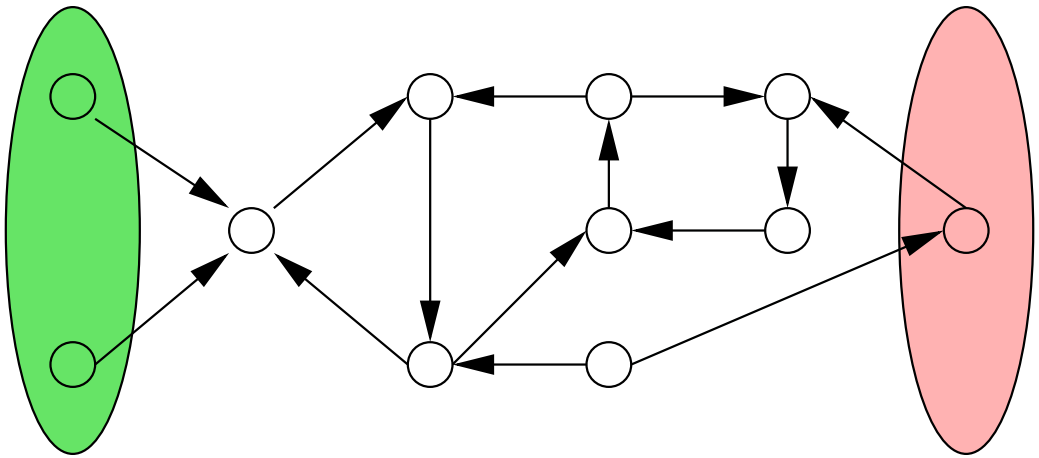
Cospans of Graphs

Cospans are graphs with an inner and an outer interface, composition is performed by gluing over the joint interface. (Cospan composition is denoted by ;.)



Cospans of Graphs

Cospans are graphs with an inner and an outer interface, composition is performed by gluing over the joint interface. (Cospan composition is denoted by ;.)



Formally: a cospan consists of three graphs (middle graph, inner and outer interface) with graph morphisms from the interface into the middle graph

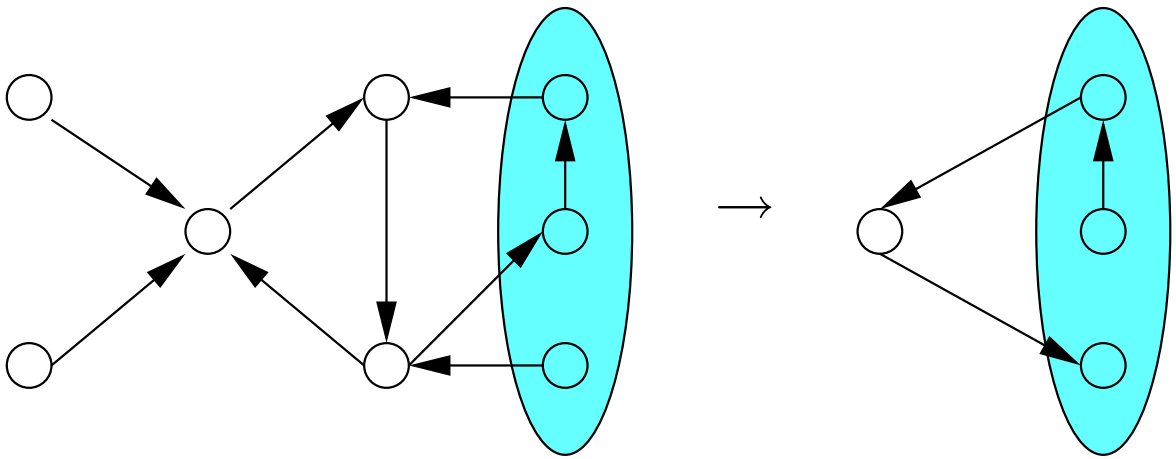
A Cospan View of Graph Transformation

Graph transformation (double-pushout rewriting [Ehrig]) can be defined via cospan compositions.

A Cospan View of Graph Transformation

Graph transformation (double-pushout rewriting [Ehrig]) can be defined via cospan compositions.

Rule $\ell \to r$ (ℓ: left-hand side, r: right-hand side)

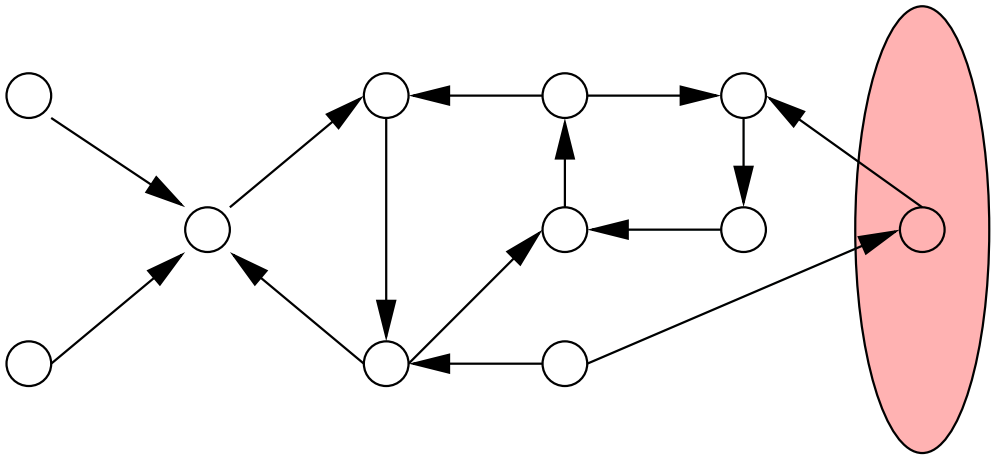


Left-hand side and right-hand side are cospans with empty inner interface

A Cospan View of Graph Transformation

Graph transformation (double-pushout rewriting [Ehrig]) can be defined via cospan compositions.

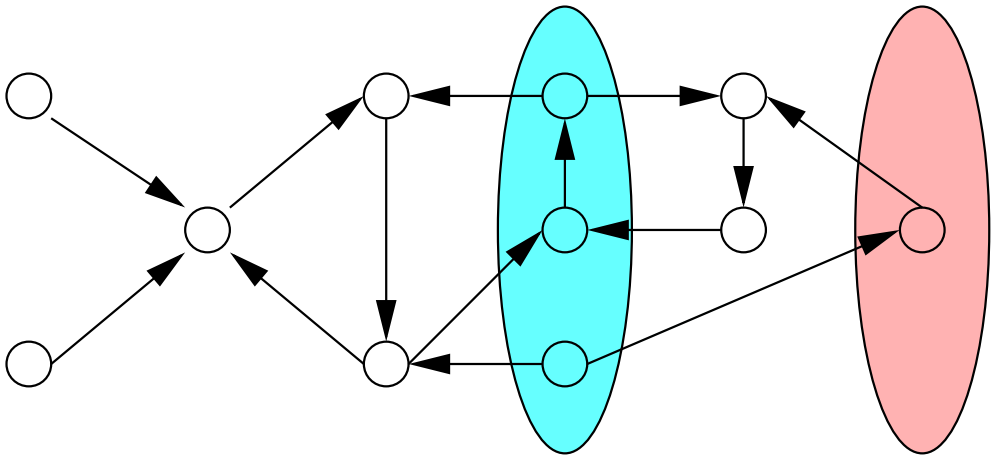
Rewriting: graph with an outer interface (to be rewritten) r



A Cospan View of Graph Transformation

Graph transformation (double-pushout rewriting [Ehrig]) can be defined via cospan compositions.

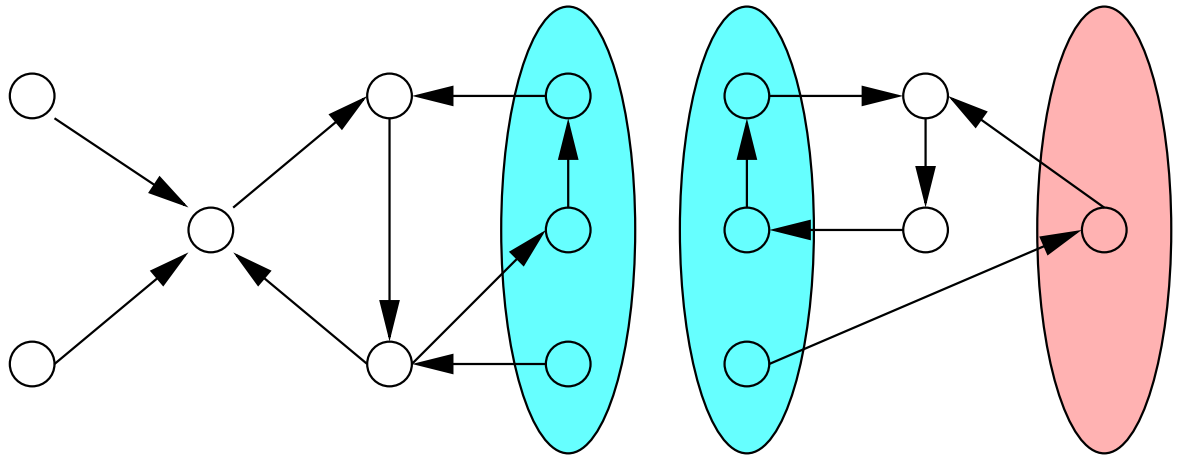
Rewriting: graph decomposed into ℓ and a context



A Cospan View of Graph Transformation

Graph transformation (double-pushout rewriting [Ehrig]) can be defined via cospan compositions.

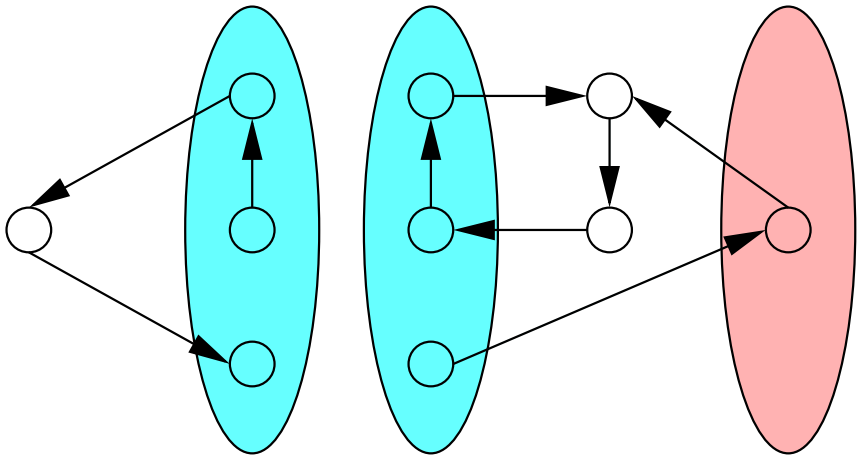
Rewriting: graph decomposed into ℓ and a context



A Cospan View of Graph Transformation

Graph transformation (double-pushout rewriting [Ehrig]) can be defined via cospan compositions.

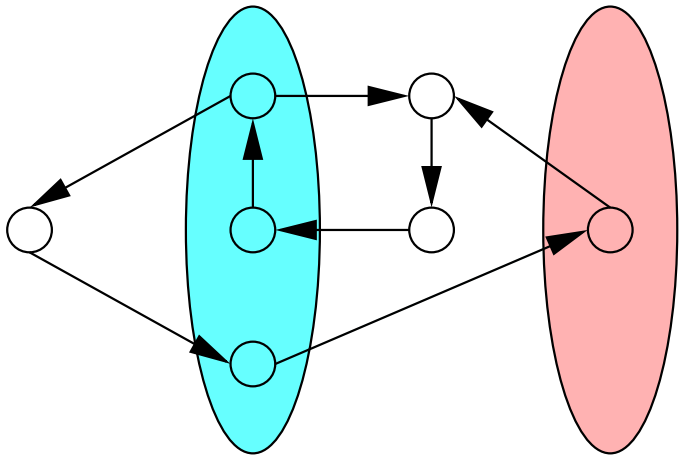
Rewriting: replace ℓ by r



A Cospan View of Graph Transformation

Graph transformation (double-pushout rewriting [Ehrig]) can be defined via cospan compositions.

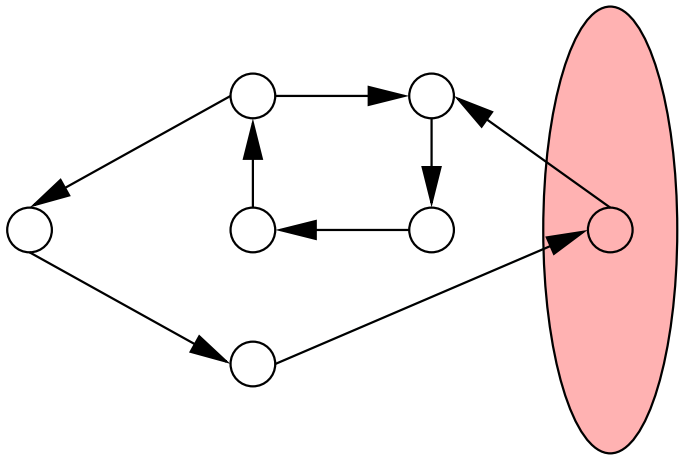
Rewriting: graph decomposed into r and the context



A Cospan View of Graph Transformation

Graph transformation (double-pushout rewriting [Ehrig]) can be defined via cospan compositions.

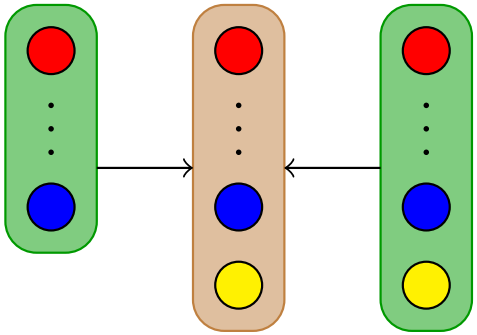
Rewriting: resulting graph



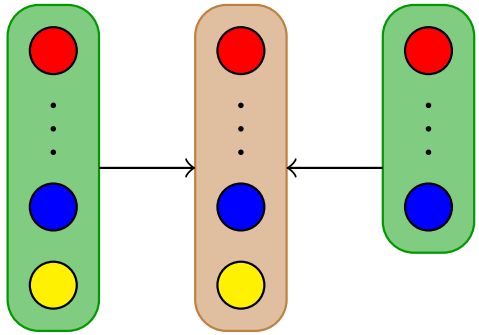
Graph Automata Based on Cospans

Atomic cospans:

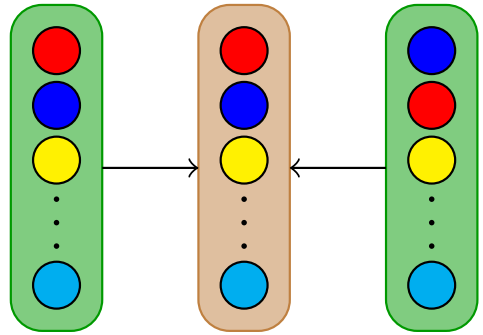
$vertex^k =$



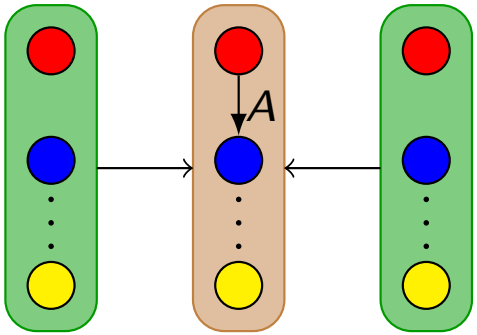
$res^k =$



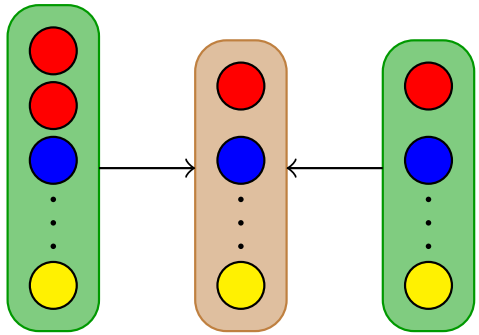
$trans^k =$



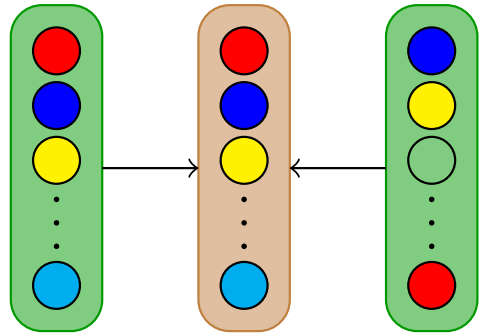
$connect^k_A =$



$fuse^k =$



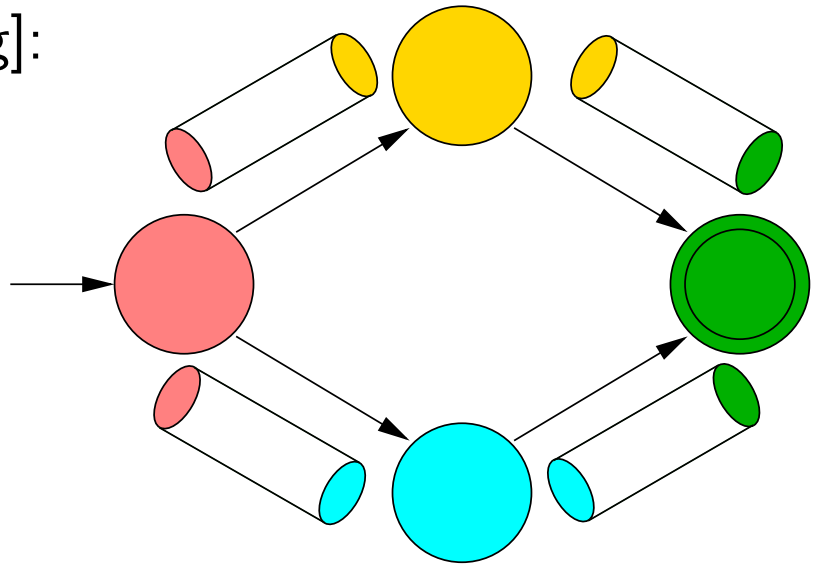
$perm^k =$



Graph Automata Based on Cospans

Graph automata [Bruggink, König]:

- states are typed by interface size (nodes only),
- transitions are labelled with atomic cospans (playing the role of alphabet symbols).



Different decompositions of the same cospan must induce the same transition function. Only finitely many states for each interface.

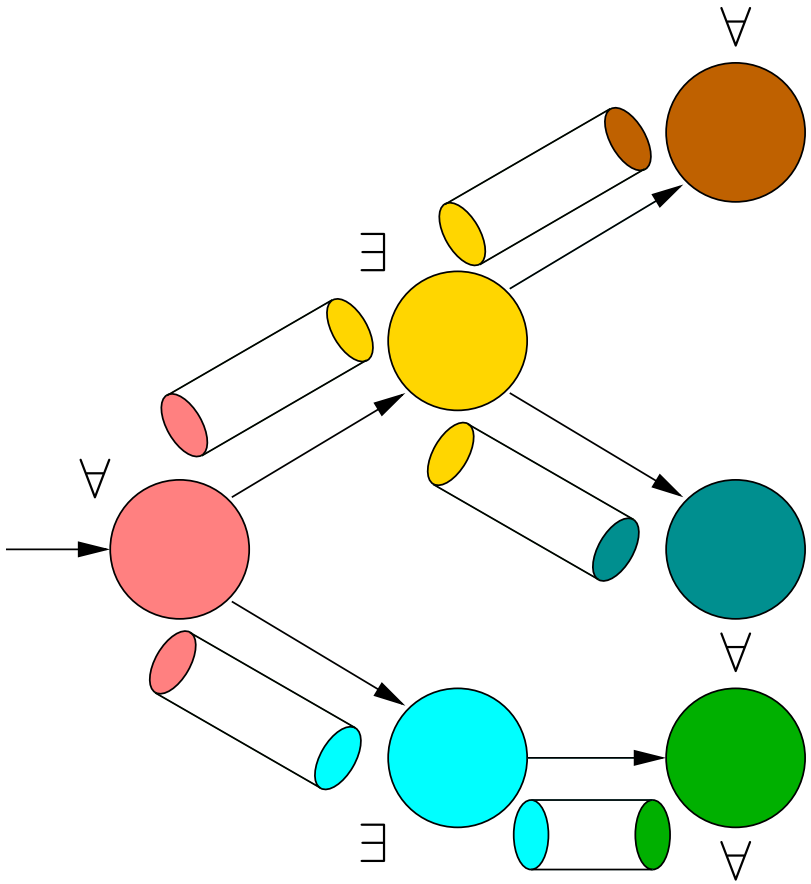
Graph automata yield a notion of recognizability equivalent to Courcelle's (more expressive than monadic second-order logics).

For practical applications the interface size has to be bounded (= languages of bounded pathwidth).

Graph Conditions Based on Cospans

Graph condition:

- Acyclic finite automaton
- Transition labels are arbitrary cospans (not necessarily atomic cospans)
- Quantifiers on the states (as in alternating automata [Chandra, Kozen, Stockmeyer])
- States with universal quantification and no outgoing transitions accept everything (they represent *true*)



Graph Conditions Based on Cospans

Expressiveness of Graph Conditions

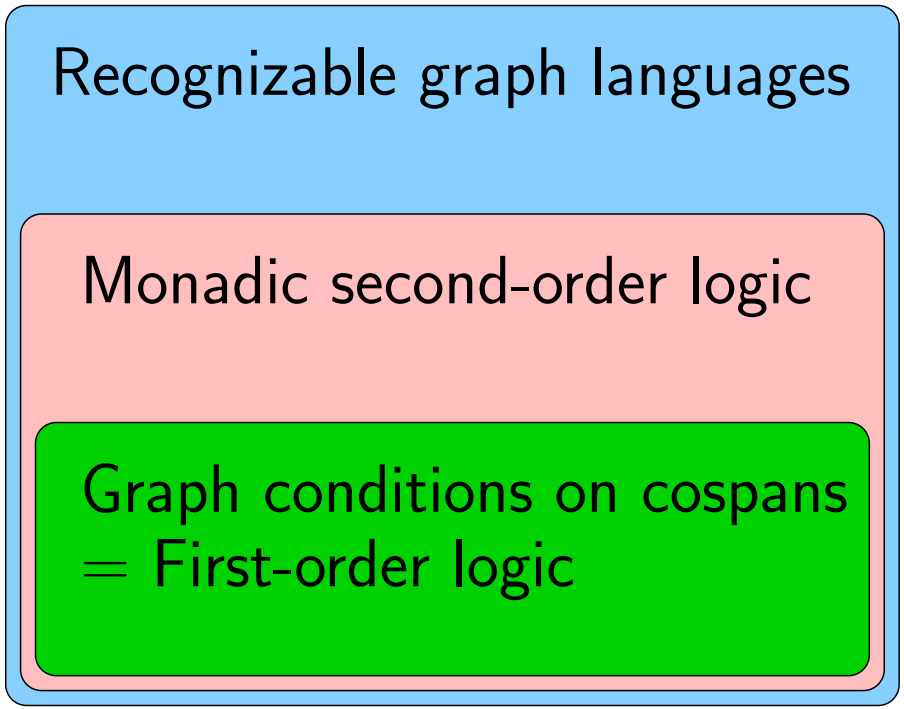
Under certain conditions (finite set of edge labels) graph conditions are ...

- equivalent to nested application conditions [Habel, Pennemann] ...
- and also equivalent to first-order logic (cf. [Rensink])

But: can be more expressive when we generalize to more general graph-like structures [Bruggink, Cauderlier, Hülsbusch, König]

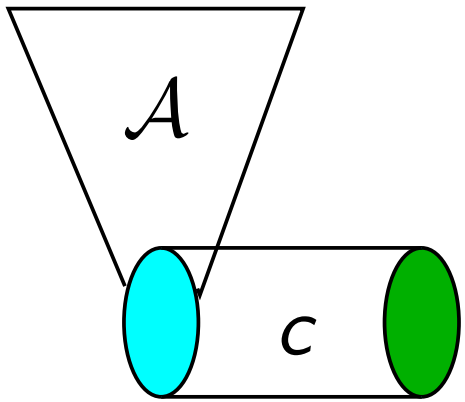
Comparison of Specification Formalisms

Summary: hierarchy of specification formalisms



Graph Conditions Based on Cospans

Graphic notation:

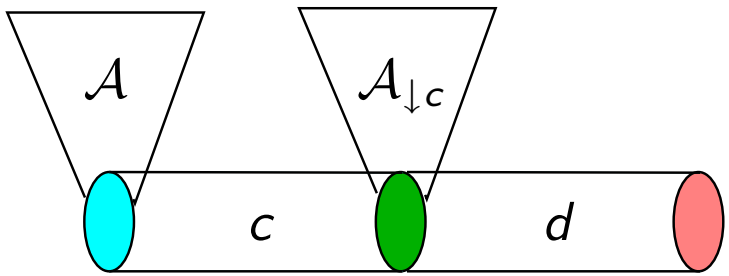


- $\mathcal{A}$ represents a property (graph condition, graph language) for cospans with a fixed inner interface (= type of the initial state of the automaton)
- $c \models \mathcal{A}$ says that cospan c satisfies property $\mathcal{A}$

Algorithmic Constructions: Shift, Partial Evaluation

Define: a construction (shift, partial evaluation) on graph properties that satisfies

$$c; d \models \mathcal{A} \iff d \models \mathcal{A}_{\downarrow c}$$



- On graph automata: replace initial states by the states reachable via c (from the initial state)
- On graph conditions: more complex, basically determine all overlaps between $\mathcal{A}$ and c

Such a shift operation is much more difficult to define directly on the logics.

Invariant Checking

We now look into applications in the area of verification (of graph transformation systems):

Problem

Check that a rule $R = (\ell \to r)$ preserves property $\mathcal{A}$. That is given a graph (seen as a cospan with empty interface) that satisfies $\mathcal{A}$, we can ensure that it satisfies $\mathcal{A}$ after application of rule R.

Solution

Check

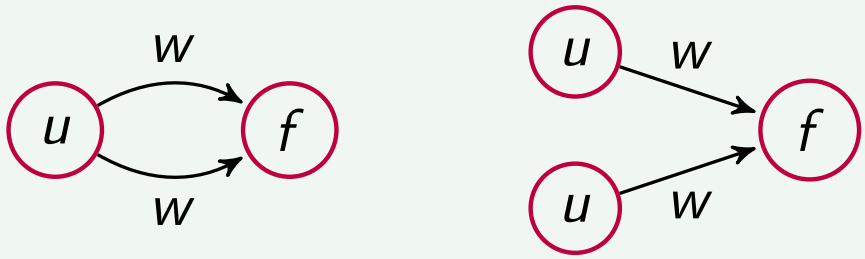
$$\mathcal{A}_{\downarrow \ell} \models \mathcal{A}_{\downarrow r}$$

Read: the contexts in which the left-hand side ℓ satisfies $\mathcal{A}$ are contained in the contexts in which the right-hand side r satisfies $\mathcal{A}$.

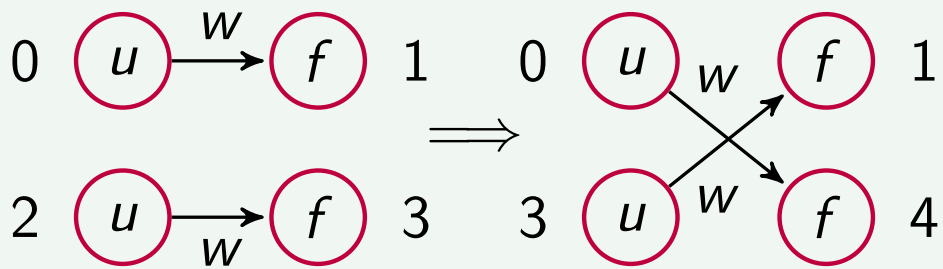
Invariant Checking

Example: Multi-user file system

Forbidden Subgraphs (multiple write access)



Show that certain reconfiguration rules preserve the absence of forbidden subgraphs



Strongest Postconditions

Problem

Let $R = (\ell \to r)$ be a transformation rule.
Compute the strongest postcondition $sp(\mathcal{A}, R)$ for $\mathcal{A}$ and R (characterize the set of successors of the graphs specified by $\mathcal{A}$ after application of rule R)

Solution

$$sp(\mathcal{A}, R) = \exists r.(\mathcal{A}_{\downarrow \ell})$$

Read: there exists a right-hand side r and a context which – together with the left-hand ℓ – satisfies $\mathcal{A}$

Needed: operator $\exists r$ ("there exists a cospans r such that ...")

Strongest Postconditions

Operator $\exists r$

- On graph automata: fairly complex operation
- On graph conditions: just introduce a new initial state (existentially quantified) and add an r-transition to the previous initial state

Similarly: define weakest precondition using universal quantification

Abstract Interpretation

Problem

Obtain an abstract transition system that simulates the original transition system. The abstraction can be used to verify safety properties (such as non-reachability queries).

Solution (initial ideas, work in progress)

- Describe abstract states via graph properties
- Abstract transitions are obtained by computing strongest postconditions (cf. materializations in abstract graph transformation [Rensink] [Sagiv, Reps, Wilhelm])
- Abstraction/widening techniques to keep number of abstract states finite
- Counterexample-guided abstraction refinement (CEGAR)

Conclusion

Implementation

Graph automata are huge! (State space explosion when the permissible interface size is increased)

$\rightsquigarrow$ use reduction techniques for large state spaces (binary decision diagrams)

Tool RAVEN (developed by Christoph Blume at the university of Duisburg-Essen)

Conclusion

Undecidability issues

The satisfiability (and entailment) problem is undecidable, already for first-order logic!

Workaround:

- Both problems become decidable for monadic second-order logic if we restrict to graphs of bounded pathwidth (bound the interface size of atomic cospans) or of bounded treewidth [Courcelle]
- Heuristics for first-order logic [Pennemann]

Conclusion

Categorical foundations

All definitions and constructions can be extended to arbitrary graph-like structures (adhesive categories).

Related work

- Recognizable graph languages [Courcelle]
- Nested application conditions in graph rewriting (and rewriting in adhesive categories) [Habel/Pennemann, Ehrig et al.]
- Weakest preconditions and strongest postconditions in high-level transformation systems [Habel/Pennemann '09]
- First-order graph logic vs. conditions [Rensink '04]


Blume, C., Bruggink, H. S., Friedrich, M., and König, B. (2013).
Treewidth, pathwidth and cospan decompositions with applications to graph-accepting tree automata.
Journal of Visual Languages & Computing, 24(3):192–206.


Blume, C., H. J. S. B., Engelke, D., and König, B. (2012).
Efficient symbolic implementation of graph automata with applications to invariant checking.
In *Proc. of ICGT '12 (International Conference on Graph Transformation)*, pages 264–278. Springer.
LNCS 7562.


Bruggink, H. S., Cauderlier, R., König, B., and Hülsbusch, M. (2011).
Conditional reactive systems.
In *Proc. of FSTTCS '11*, volume 13 of *LIPIcs*. Schloss Dagstuhl – Leibniz Center for Informatics.


Bruggink, H. S. and König, B. (to appear).
Recognizable languages of arrows.
Mathematical Structures in Computer Science.