A Logic on Subobjects and Recognizability

Barbara König

Universität Duisburg-Essen, Germany

Joint work with Sander Bruggink
(published at IFIP-TCS '10)

Overview

Motivation

Our overall aim is the verification of dynamic systems, especially graph transformation systems.

There are several verification techniques which are based on regular (= recognizable) word languages.

What about recognizable graph languages?

This talk: recognizability and logics

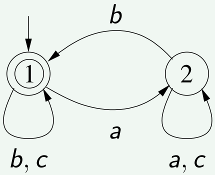
Regular Languages and Monadic Second-Order Logic

There is an intimate connection between formal languages 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 later position in the word with a b."

Regular Languages and Monadic Second-Order Logic

Why is this interesting (for our purposes)?

- Encoding a logical formula into an automaton transforms a specification into an algorithm.
- Automata are well-suited for answering the following questions:
  - Is the given regular language L empty?
  - Is L_1 included in L_2: $L_1 \subseteq L_2$?
  - Are L_1 and L_2 equal: $L_1 = L_2$?

Regular Languages and Monadic Second-Order Logic

What about graph languages?

There is a notion of recognizable graph languages by Courcelle.

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

Regular Languages and Monadic Second-Order Logic

Why stop with graphs?

- We have generalized Courcelles notion of recognizable to arbitrary categories. ($\rightsquigarrow$ IFIP-WG-Meeting in Udine)
- Contributions in this talk:
  - Present a (simple) logic on subobjects that coincides with Courcelle's logic when we instantiate it with the category of graphs.
  - Show that every language definable in this logic is recognizable (for the case of hereditary pushout categories, related to adhesive categories).
    Our encoding from logical formulas into automata is inductive on the formula (as opposed to the original proof by Courcelle).

Monadic Second-Order Graph Logics

Monadic second-order graph logics may quantify over nodes, edges, node sets and edge sets. It is defined for hypergraphs.

$$\varphi := \varphi_1 \wedge \varphi_2 \mid \neg\varphi \mid (\exists X\colon V)\,\varphi \mid (\exists X\colon E)\,\varphi \mid (\exists x\colon v)\,\varphi \mid (\exists x\colon e)\,\varphi \mid$$
$$x = y \mid x \in X \mid edge_A(x, y_1, \dots, y_{ar(A)}),$$

Example: $(\exists x\colon v)\,(\exists y\colon v)\,(\exists z\colon e)\,(edge_A(z,x,y) \wedge x = y)$
"There exists an A-labelled loop."

Monadic Second-Order Graph Logics

Courcelle also considers counting monadic second-order logic which allows to express statements such as:

> The number of nodes (or edges) in a set X is equal to k modulo m.

Here we do not consider this extension.

Graph Decompositions and Recognizability

We now look into recognizability:

- How to accept a word? $\rightsquigarrow$ Decompose it into letters and read every letter separately.
- How to accept a graph? $\rightsquigarrow$ Decompose the graph into smaller units
  $\rightsquigarrow$ path and tree decompositions

Graph Decompositions and Recognizability

Give a directed graph $G = (V, E)$ with $E \subseteq V \times V$, a tree decomposition of G consists of a tree T and sets (= bags) $X_t \subseteq V$ for every vertex t of T, satisfying the following properties:

- The union of all bags X_t equals V. (Every node lives in at least one bag.)
- For every edge $(u, v) \in E$ there exists a vertex t of T with $u, v \in X_t$. (Every edge lives in at least one bag.)
- For every node $v \in V$ the set of vertices $\{t \mid v \in X_t\}$ forms a subtree of T.

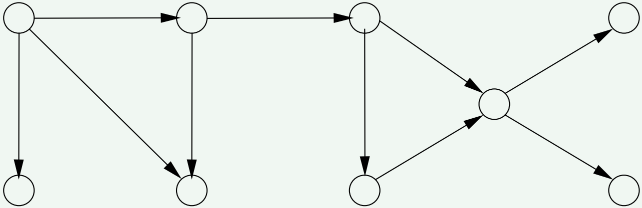
It is a path decomposition if T is a path (instead of an arbitrary tree).

The width of a tree composition is

$$\max_t\{|X_t|\} - 1$$

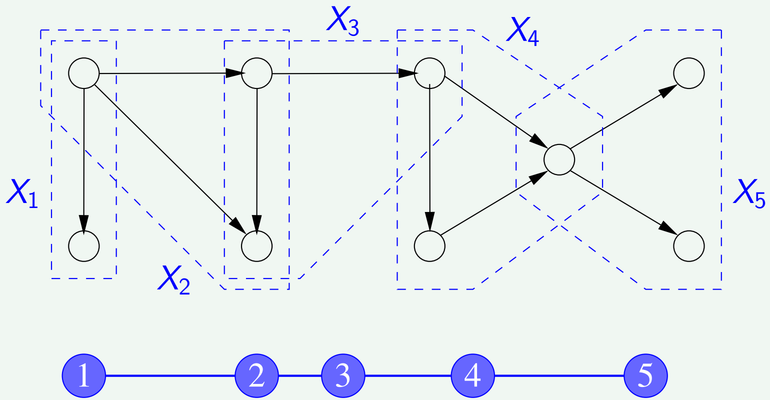
Graph Decompositions and Recognizability

Example:



Graph Decompositions and Recognizability

Example:



Path decomposition of width 2.

Graph Decompositions and Recognizability

Treewidth, pathwidth

The treewidth of graph G is the minimal width of a tree decomposition of G. Analogously for pathwidth.

Intuitively treewidth measures how similar a given graph is to a tree (path).

There is the following relation between pathwidth and treewidth of a graph G:

$$\mathrm{pwd}(G) \in O(\log n \cdot \mathrm{twd}(G))$$

where n is the number of nodes of G.

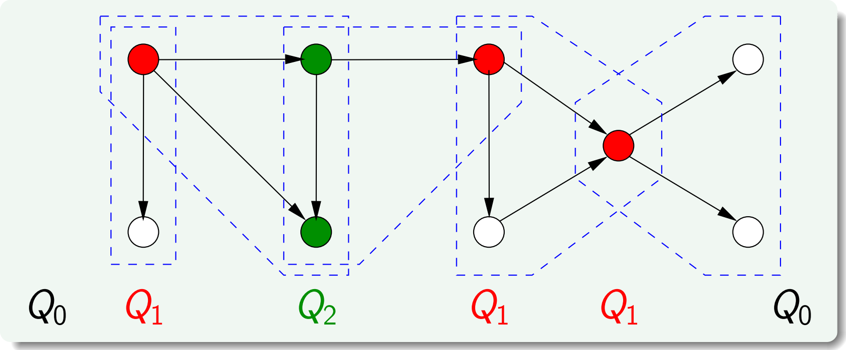
Graph Decompositions and Recognizability

Idea:

- Consider automata over graphs.
- Treat each bag as a "letter" which induces a transition between states.
- The intersection between two neighbouring bags is the interface between the bags. Associate a separate state set Q_n to each interface size n.
- Make sure that the transition function induced by a graph is independent of its decomposition.
- In order to obtain a finite automaton restrict the size of the interfaces and the size of the bags $\rightsquigarrow$ restrict to graphs of bounded pathwidth.
- For tree decompositions use tree automata.

Graph Decompositions and Recognizability

Example:



Graph Decompositions and Recognizability

The intuition is closer to our definition of recognizability that Courcelle's, but it is equivalent to Courcelle's notion:

Recognizability (Courcelle)

Define a multi-sorted algebra of graphs, where

- sorts are natural numbers (infinitely many sorts!) and the carrier of n is the set of graphs with n interface nodes.
- operations merge graphs and manipulate the interface.

Assume that we have an algebra homomorphism into an algebra $\mathcal{F}$ with the same signature but finite carrier sets.

A set of graphs is recognizable if it is the pre-image of a subset of a carrier set of $\mathcal{F}$.

Graph Decompositions and Recognizability

All this has interesting consequences for complexity theory:

Courcelle's Theorem

Let L be a recognizable graph language. Then it is decidable in linear time for graphs G of bounded treewidth whether G is contained in L.

Idea: find a tree decomposition of G (non-trivial, but possible in linear time in the size of the graph!) and check via the (tree) automaton whether G is contained in L.

Graph Decompositions and Recognizability

Corollary

Every graph property expressible in monadic second-order logic can be decided in linear time for graphs of bounded treewidth.

Example graph properties:

- subgraph isomorphism
- k-colorability
- planarity
- ...

Some of these problems (e.g., subgraph isomorphism, k-colorability) are NP-complete for graphs of unbounded treewidth.

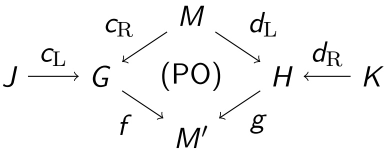
Automaton Functors

We now take a more categorical view: replace bags and their interfaces by cospans.

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

$$J \rightarrow G \leftarrow K$$

- Cospan composition:



Automaton Functors

Automaton functor

Functor $\mathcal{A}\colon \mathit{Cospan}(\mathbf{C}) \to \mathbf{Rel}_{\mathrm{fin}}$, where

- 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 cospan 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 $\mathcal{A}(J)$.

It is sufficient to define automaton functors on atomic cospans (for graphs: cospans which add edges, nodes, permute nodes, restrict nodes). However, functoriality must be ensured. For graphs it is also sufficient to take as objects only discrete graphs.

There is a notion of recognizability by Griffing that is quite similar to ours.

A Logic on Subobjects

We now introduce our logics, which classifies objects of **C** via their subobjects.

Variables

- First-order variables $x\colon T$ of sort T, where T is an object of **C**. (Such variables stand for subobjects isomorphic to T.)
- Second-order variables $X\colon \Omega$ of sort Ω. (Such variables stand for arbitrary subobjects.)

Expressions

$$e := X \mid f \mathbin{;} x,$$

where $X\colon \Omega$, $x\colon T$ and $f\colon T' \rightarrowtail T$ is a mono.

Intuition: x stands for a subobject isomorphic to T, whereas $f \mathbin{;} x$ stands for a subobject of that subobject, as specified by f.

A Logic on Subobjects

Formulas

$$\tau := e_1 \sqsubseteq e_2 \mid \tau_1 \wedge \tau_2 \mid \neg\tau \mid (\exists X \colon \Omega)\,\tau \mid (\exists x \colon T)\,\tau$$

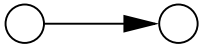
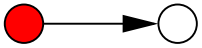
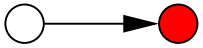
$e_1 \sqsubseteq e_2$ stands for subobject inclusion.

I will skip the definition of the semantics ("when does an object satisfy a formula?"). This is quite standard.

This is quite different from the usual categorical logics!

A Logic on Subobjects

Example formulas for **C** = **Graph**:

Let $E =$ , *src* :  $\rightarrow$ , *tgt* :  $\rightarrow$ 

- There exist two edges which have the same target node:
  $(\exists x\colon E)\,(\exists y\colon E)\,(tgt \,;\, x = tgt \,;\, y)$
- The subgraph X is closed under reachability:
  $RC(X : \Omega) := (\forall y\colon E)\,(src \,;\, y \sqsubseteq X \rightarrow tgt \,;\, y \sqsubseteq X)$
- There exists a path from node x to node y (every reachability closed subgraph containing x also contains y):
  $Path(x, y) := (\forall Z : \Omega)\,\big((id \,;\, x \sqsubseteq Z \wedge RC(Z)) \rightarrow id \,;\, y \sqsubseteq Z\big)$

A Logic on Subobjects

The logic on subobjects vs. monadic second-order graph logic

For $\mathbf{C} = \mathbf{Graph}$, we can translate every formula of our logic into monadic second-order graph logic and vice versa.
The two logics have the same expressive power.

Translation to Automaton Functors

Logics and Recognizability

Let **C** be a hereditary pushout category ($\rightsquigarrow$ Heindel, the definition is very similar to an adhesive category). Furthermore assume that **C** has an initial object 0.

Let L be a language of objects that can be characterized in the logic of subobjects. Then there is an automaton functor $\mathcal{A}$ which recognizes exactly the cospans in

$$\{0 \to A \leftarrow 0 \mid A \in L\}$$

Translation to Automaton Functors

The translation is inductive on the structure of the formula.
In order to give some intuition concerning the translation, we consider the following formula:

$$\exists(x\colon T)\mathit{true}$$

That is we ask whether there exists a subobject isomorphic to T.

Translation to Automaton Functors

State sets

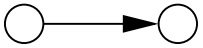
Each object (= interface) B is associated with a set of states $\mathcal{A}(B)$, where $\mathcal{A}(B)$ contains triples of arrows of the following form:

$$(v\colon V \rightarrowtail B, t_1\colon V \to \overline{V}, t_2\colon \overline{V} \to T).$$

- v specifies the intersection V of the interface with the object we are looking for.
- t_2 describes the part $\overline{V}$ of T that has already been detected.
- t_1 describes how the intersection is located inside $\overline{V}$ (and thus T).

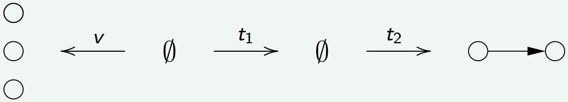
Translation to Automaton Functors

Example: category of graphs

We consider an interface of size 3 and $T =$ . The following triples of morphisms

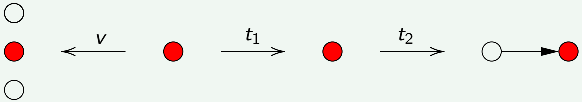
$$B \xleftarrow{v} V \xrightarrow{t_1} \overline{V} \xrightarrow{t_2} T$$

have the following meaning as states:

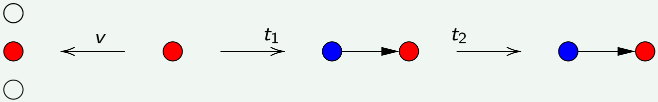


So far we have not seen any part of T.

Translation to Automaton Functors



We have only seen the red node of T so far and it is in the current interface.

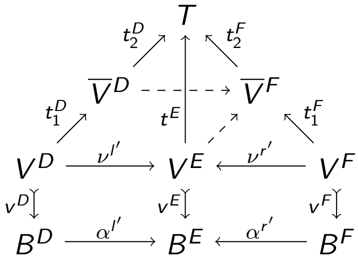


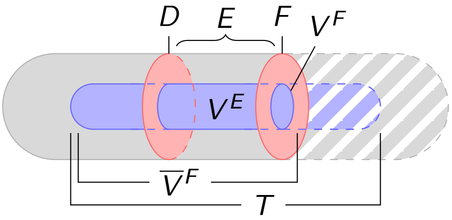
We have seen all of T so far, but the red node is the only one left in the current interface.

Translation to Automaton Functors

Transition relation

Given a cospan interfaces $B^D \to B^E \leftarrow B^F$ there is a transition from (v^D, t_1^D, t_2^D) to (v^F, t_1^F, t_2^F) whenever there is an object V^E and arrows such that the two squares below are pullbacks and the trapezoid in the middle is a pushout.



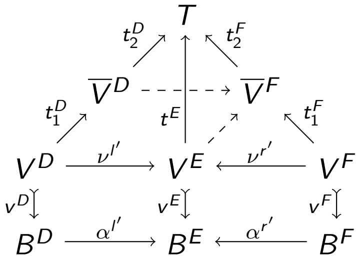


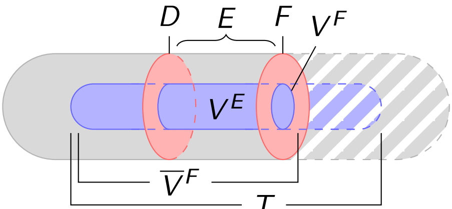
Translation to Automaton Functors

Transition relation

Intuitively: We nondeterministically guess a part V^E of the cospan that extends $\overline{V}^D$ to $\overline{V}^F$ ($\rightsquigarrow$ pushout).

If we restrict V^E to the left and right interface we must obtain V^D and V^F ($\rightsquigarrow$ pullbacks).





Translation to Automaton Functors

More details concerning the translation:

- The encoding of second-order quantifiers is easier, since we only have to non-deterministically guess possible extensions, without checking their relation to a given object T.
- Formulas with free variables are encoded by considering automaton functors from a category where objects are **C**-objects with valuations and arrows are cospans between such objects.
- The translation is done inductively on the structure of the formula. Boolean operations are handled as usual (via cartesian products of the state sets or exchange of final/non-final states). However, negation requires prior determinization and could be very costly.
- In order to obtain our results we have to show we obtain a functor.

Conclusion

- We presented a generalization of the theorems by Büchi-Elgot and Courcelle (languages expressible in monadic second-order are regular) to a general categorical setting.
- Apart from the theoretical appeal, we are also expecting practical consequences:
  - The manual definition of automaton functors is very cumbersome. Hence we now have a way to generate them automatically.
  - The inductive translation looks reasonable. (For the case of graphs such an inductive encoding has only been defined recently by Courcelle & Durand.)
  - Still, state space explosion is a problem. We are working on a BDD-based implementation, with quite encouraging results. For subgraph isomorphism we can now easily generate automata up to interface size 100.

Conclusion

Applications for a graph automaton tool suite:

- Invariant checking $\rightsquigarrow$ GT-VMT '10
  (joint work with Christoph Blume)
- Termination analysis
- Regular model-checking