Comparing two category-theory based logics for graphs

Raphaël Cauderlier, Barbara König, Université de Duisbourg - Essen

23 août 2010

Le contexte général

Dans le domaine de la réécriture de graphes, on a souvent besoin d'exprimer des contraintes sur des graphes et des sous-graphes d'un graphe donné. Ces contraintes sont généralement formulées par ce que l'on appelle des conditions d'application c'est-à-dire par des sous-graphes imposés ou au contraire interdits. Par exemple le model checker GROOVE [5] fonctionne de cette manière ; il modélise le système par un graphe et son l'évolution est décrite par de règles de réécriture dont l'application est limitée par des conditions d'application.

Pour augmenter leur puissance expressive, les conditions d'application ont été généralisées par Arend Rensink [6] à des arbres dont les nœuds sont étiquetés par des graphes et les arêtes par des morphismes de graphes. Ces conditions d'application généralisées (GACs) sont équivalents à la logique du premier ordre sur les graphes.

Bruno Courcelles a développé une autre logique pour les graphes : la logique monadique du second ordre sur les graphes [3] qui a été généralisée ici à une logique purement catégorique : la logique du second ordre sur les sous-objets [1]. Dans le cas des graphes, ces deux logiques du second ordre sont équivalentes.

Le problème étudié

L'équipe Informatique théorique de Duisbourg utilise à la fois des GACs sur des cospans [1] et la logique sur les sous-objets à des fins différentes. Cependant ces deux logiques n'avaient pas encore été comparées. Les formules de la logique sur les sous-objets sont faciles à lire et à écrire. Les GACs se manipulent bien (ce sont des représentations sans variables) et sont mieux connus donc on dispose de résultats théoriques [4]. L'équivalence entre les GACs et le fragment du premier ordre de la logique sur les sous-objets nous permettrait de bénéficier des avantages de ces deux représentations. Cette question n'avait pas encore été posée car la logique sur les sous-objets est relativement récente.

La contribution proposée

J'ai commencé par travailler sur l'encodage des GACs dans la logique sur les sous-objets. La première étape a consisté à simplifier les GACs en se plaçant dans le cas où tous les morphismes sont injectifs (de sorte que chaque nœud soit un sous-graphe de chacun de ses fils). Ces GACs particuliers sont faciles à transformer en formules de la logique sur les sous-objets.

J'ai ensuite réduit les GACs normaux à ces GACs simples de manière très générale. Cette étape a nécessité un résultat sophistiqué de théorie des catégories mais ça a bien fonctionné.

[1] un cospan est une paire de morphismes qui ont le même codomaine

Enfin nous avons trouvé une façon d'encoder les formules du premier ordre de la logique sur les sous-objets dans les GACs ; ça a été la partie la plus difficile et ça nous a conduit à un résultat surprenant.

Tout ce travail a été fait dans le cadre très abstrait de la théorie des catégories ; les graphes n'intervenant que dans les exemples.

Les arguments en faveur de sa validité

La solution apportée au problème est très élégante. Dans les deux directions, l'encodage a l'air naturel car il procède par une jolie construction par induction. Les preuves sont courtes sans être triviales.

Les hypothèses de travail (quelques propriétés que la catégorie doit satisfaire) sont raisonnables ; elles correspondent au cadre généralement adopté dans le domaine de la transformation de graphes et sont vérifiées pour presque toutes les structures qui ressemblent à des graphes et pour lesquelles on peut avoir envie d'appliquer ces résultats.

Le bilan et les perspectives

La théorie des catégories nous fournit non seulement des théorèmes puissants mais aussi un cadre extrêmement général qui permet de traiter certains problèmes de manière à ce que les résultats puissent être appliqués à toute catégorie vérifiant quelques axiomes.

Il serait intéressant d'encoder les GACs construits sur les cospans, strictement plus expressifs que les GACs que j'ai étudié, en des formules du second ordre et de les comparer à la logique monadique du second ordre.

Pour les applications concrètes, il faudrait s'intéresser à l'efficacité de ces différentes représentations. Comment la taille des formules augmente-t-elle avec l'encodage ? Peut-on éviter l'explosion combinatoire provoquée par la traduction en GACs ?

1 Introduction and Scientific Context

1.1 Graph Transformations and (Negative) Application Conditions

Several logics on graphs have been developed to express constraints which limit the application of rewriting rules.

Let us consider the following classical example ; we are specifying a file-system. Each file has an owner (a user of the system) and a name (a string). We represent all identities (files, strings and users) by labeled nodes and all relations between them by labeled edges. For instance figure 1 is a possible state of our file-system.

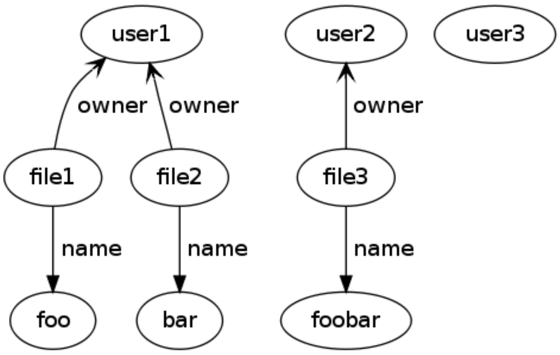


FIGURE 1 – This graph represent a possible state of the file system

We are especially interested in file renaming. The rewriting rule for renaming a file named "a" into "b" is shown in Figure 2.

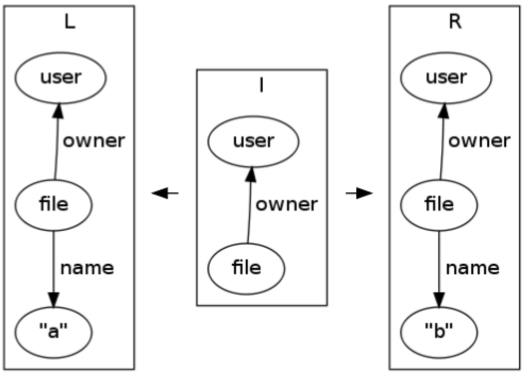


FIGURE 2 – Rewriting rule for file renaming

In this rule,

- L is called the left-hand side of the rule ; it is a graph we have to find in the representation of the file-system.
- R is called the right-hand side of the rule ; it is the graph by which we want to replace the matching of L.
- I is called the interface of the rule ; it links L and R by telling which part of R should correspond to a part of L.

This is not satisfying because this could produce several files with the same name. We want to restrict this rule by forbidding such a situation. For this purpose we add a so called Negative

Application Condition on the right-hand side of the rule (see Figure 3).

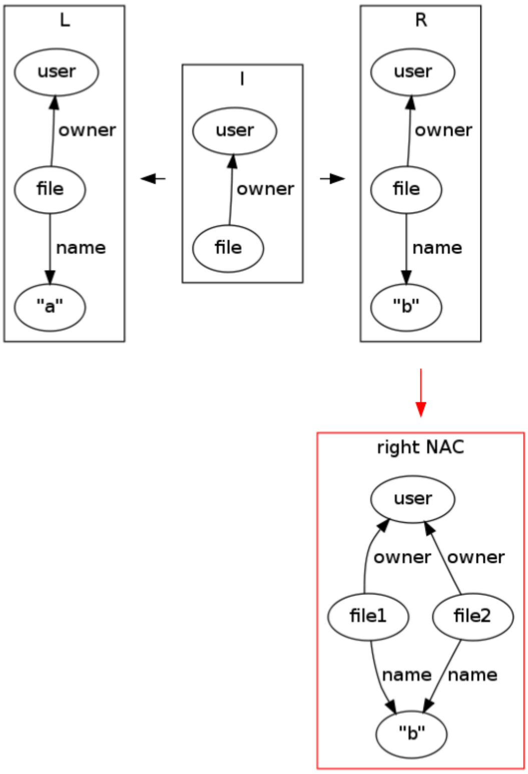


FIGURE 3 – Renaming rule with a right NAC

Now the rule can be applied only if its production is not a matching of the right NAC.

If we want to add a notion of write-protection for files, we will also need to express that protected files cannot be renamed. We add an other NAC, on the left-hand side (see Figure 4).

The application of the rule is now restricted to the matchings of L which are not matchings of the left NAC and which do not lead to a matching of the right NAC after the application of the rule.

1.2 Category Theory

Category theory is a very abstract mathematical theory which focuses on the notion of morphism. A category is given by a class of objects (in our case, objects are things which look like graphs) and for each pair (A, B) of objects, a set $\mathrm{Hom}(A, B)$ called the set of morphisms with domain A and codomain B. Morphisms behave like functions (we have an associative composition[2] and for each object A we have a morphism $\mathrm{id}_A \in \mathrm{Hom}(A, A)$ which is neutral for composition) but the definition does not require them to be functions.

We are particularly interested in the class of mono-morphisms (simply called monos) which in case of graph categories correspond to injective graph morphisms :

Definition 1. *A morphism $m \in \mathrm{Hom}(A, B)$ is called a mono if for all object C and all morphisms $f, g \in \mathrm{Hom}(C, A)$ such that $f \ ; \ m = g \ ; \ m$, we have $f = g$.*

[2] $f \ ; \ g$ denotes the composition of g and f usually written $g \circ f$

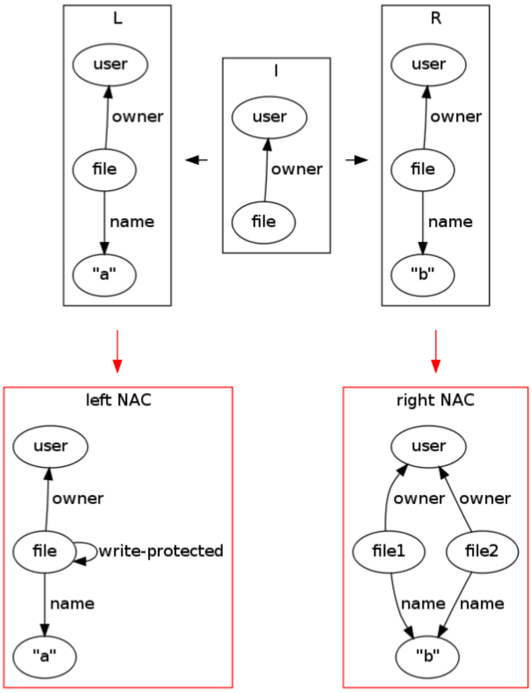


FIGURE 4 – Renaming rule with both left and right NACs

In category theory a lot of general concepts from set theory, algebra and topology are unified and generalized. For example we can build a concept of subobject which generalized the concepts of subset, subgroup, subgraph . . .

Definition 2. *A subobject of a given object A is a couple (B, m) where B is an object and m is a mono with domain B and codomain A. The subobject (B, m) of the object A will be denoted $B \overset{m}{\rightarrowtail} A$.*

Now we can define a partial order $\leqslant$ on subobjects [3] :

Definition 3. *Let $B_1 \overset{m_1}{\rightarrowtail} A$ and $B_2 \overset{m_2}{\rightarrowtail} A$ be two subobjects of the same object A, we say that $B_1 \overset{m_1}{\rightarrowtail} A$ is smaller than $B_2 \overset{m_2}{\rightarrowtail} A$ and we write $m_1 \leqslant m_2$ if there is a mono m such that the diagram*

$$\begin{tikzcd}
B_1 \arrow[rr, tail, "m"] \arrow[dr, tail, "m_1"'] & & B_2 \arrow[dl, tail, "m_2"] \\
& A &
\end{tikzcd}$$

commutes (this simply means $m_1 = m \,;\, m_2$).

Analogously [4] to the definition of monos, we can define the notion of epis which in case of graphs correspond to surjective morphisms.

Definition 4. *A morphism $e \in \mathrm{Hom}(A, B)$ is called an epi if for all object C and all morphisms $f, g \in \mathrm{Hom}(B, C)$ such that $e \,;\, f = e \,;\, g$, we have $f = g$.*

[3] In fact, $m_1 \leqslant m_2$ and $m_2 \leqslant m_1$ only implies that m_1 and m_2 are isomorphic, not necessarily equals. We can however simply work *up to isomorphism* because isomorphic objects are not distinguishable in category theory

[4] categorists name this kind of analogy ‘duality’

1.3 Monadic Second Order Logic of Graphs

Monadic second-order logic over graphs is a logic developed and studied by Bruno Courcelle [2]. 'Monadic second-order' means that we can quantify over nodes and edges (first-order quantification) and also over sets of nodes and edges (second-order quantification) but not over relations between nodes nor relations between edges nor relations between nodes and edges (this is the meaning of 'monadic'). This logic represent an interesting compromise between expressive power and complexity (it is decidable in linear time on graphs of bounded tree-width [3]).

We won't describe this logic in details but only give an example : let us build a formula representing the property "there is a path from x to y".

- first, we define what it means to be a subgraph :
  $SG(N : \mathrm{Nodes}, E : \mathrm{Edges}) :=$
  $(\forall\, a, b : \mathrm{node})\ (\forall\, e : \mathrm{edge})\ e \in E \wedge \mathrm{Edge}(e, a, b) \rightarrow a \in N \wedge b \in N$
- then, we build a formula meaning "the set of nodes N and the set of edges E represent a subgraph closed under reachability" :
  $RC(N : \mathrm{Nodes}, E : \mathrm{Edges}) :=$
  $SG(N, E) \wedge (\forall\, e : \mathrm{edge})\ (\forall\, a, b : \mathrm{node})\ \mathrm{Edge}(e, a, b) \wedge a \in N \wedge e \in P \rightarrow b \in N$
- and finally the existence of a path :
  $Path(x : \mathrm{node}, y : \mathrm{node}) :=$
  $(\forall\, N : \mathrm{Nodes})\ (\forall\, E : \mathrm{Edges})\ x \in N \wedge RC(N, E) \rightarrow y \in N$

2 The Two Logics

2.1 First Order Logic on Subobjects

Logic on subobjects is a transposition of Courcelle's monadic second order logic of graphs to the world of categories. Nodes and edges are replaced by subobjects of fixed structure, sets of nodes and sets of edges are replaced by subobjects of arbitrary structure. Here is the first-order fragment of the logic on subobjects.

2.1.1 Syntax

We fix a category $\mathcal{C}$. The following grammar defines the syntax of first-order logic on subobjects :

- expressions :
  $$e ::= f \fatsemi x$$
  where x is a variable typed by an object A and f is a mono with codomain A. Expressions represent subobjects restricted by a mono.
- formulae :
  $$\varphi ::= e \sqsubseteq e \mid \bigwedge_{i \in I} \varphi_i \mid \neg\varphi \mid (\forall\, x : A)\ \varphi$$
  where I is a finite set, x is a variable and A is an object. We also use $e_1 = e_2$ as an abbreviation for $(e_1 \sqsubseteq e_2) \wedge (e_2 \sqsubseteq e_1)$. The notations $\bigvee_{i \in I} \varphi_i$, $\varphi_1 \rightarrow \varphi_2$ and $(\exists\, x : A)\ \varphi$ are defined in the usual way.

2.1.2 Semantics

Definition 5. *For all object C and all mapping η of variables to monos with codomain C, we define a modeling relation as follow :*

$$C, \eta \models f_1 \fatsemi x_1 \sqsubseteq f_2 \fatsemi x_2 \textit{ iff } f_1 \,;\, \eta(x_1) \leqslant f_2 \,;\, \eta(x_2). \tag{1}$$

$$C, \eta \models \varphi_1 \wedge \varphi_2 \textit{ iff } C, \eta \models \varphi_1 \textit{ and } C, \eta \models \varphi_2. \tag{2}$$

$$C, \eta \models \neg\varphi \textit{ iff } C, \eta \not\models \varphi. \tag{3}$$

$$C, \eta \models (\exists\, x : T)\ \varphi \textit{ iff for all mono } T \overset{m}{\rightarrowtail} C \textit{ we have } C, \eta[x \mapsto m] \models \varphi \tag{4}$$

2.1.3 Examples

Let us reuse the example of file renaming.

A formula for checking that all files have different names is :
$DN := \left(\forall\, x, y : \bigcirc \xrightarrow{\text{name}} \bigcirc\right)\ tgt \fatsemi x = tgt \fatsemi y \rightarrow x = y$ where tgt is the mono which maps a single node to the tail of the arrow labelled by 'name' i.e. the target, seen as a subobject, of this edge.

A formula meaning that the given file f is write-protected :
$WP(f : \bigcirc \xleftarrow{\text{owner}} \bigcirc \xrightarrow{\text{name}} \bigcirc) := (\exists\, p : \bigcirc \circlearrowleft \text{write-protected})\ file \fatsemi f \sqsubseteq p$ where $file$ is the middle node seen as a subobject.

Let us add a superuser represented by the node root.
Then a formula meaning that the given user u is allowed to rename the file f can be written as :
$RA(u : \bigcirc, f : \bigcirc \xleftarrow{\text{owner}} \bigcirc \xrightarrow{\text{name}} \bigcirc) :=$
$\left(\left(\exists\, su : \text{root}\right)\ u = su\right) \vee (u = user \fatsemi f \wedge \neg WP(f))$

2.2 GACs

GACs are a generalization of application conditions (positive and negative). Instead of giving only one morphism

2.2.1 Syntax

Definition 6 (**GAC**)**.** *A Generalized Application Condition (**GAC** for short) is a triple $\mathcal{A} = (A, \mathcal{Q}, S)$ where*

- *A is an object (called the root object of $\mathcal{A}$ or $\mathrm{RO}(\mathcal{A})$),*
- *$\mathcal{Q}$ is a quantifier (either $\forall$ or $\exists$) and*
- *S is a* **finite** *set of pairs $(\mathcal{A}', f)$ such that $\mathcal{A}'$ is a **GAC** and f is a morphism with domain A and codomain $\mathrm{RO}(\mathcal{A}')$.*

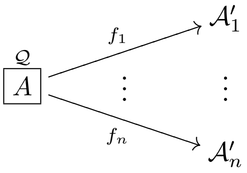


FIGURE 5 – a **GAC** is a tree of objects and morphisms

The pair $(\mathcal{A}', f)$ will be denoted by $A \xrightarrow{f} \mathcal{A}'$.

2.2.2 Semantics

Definition 7. *For all* $\mathbf{GAC}$ $\mathcal{A}$*, all object* C *and all morphism* $\mathrm{RO}(\mathcal{A}) \xrightarrow{c} C$ *we define a modeling relation as follow :*

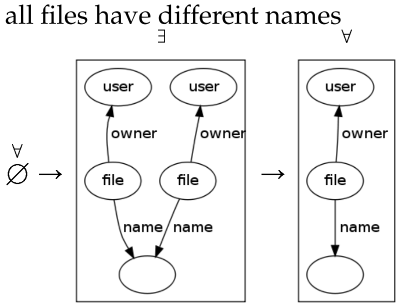
$$C, c \models (A, \forall, S) \text{ iff for every } A \xrightarrow{f} \mathcal{A}' \in S \text{ and every morphism } \mathrm{RO}(\mathcal{A}') \xrightarrow{\alpha} C$$
$$\text{such that } f' \ ; \ \alpha = c \text{ we have } C, \alpha \models \mathcal{A}' \qquad (5)$$

$$C, c \models (A, \exists, S) \text{ iff for some } A \xrightarrow{f} \mathcal{A}' \in S \text{ there is a morphism } \mathrm{RO}(\mathcal{A}') \xrightarrow{\alpha} C$$
$$\text{with } f' \ ; \ \alpha = c \text{ such that } C, \alpha \models \mathcal{A}' \qquad (6)$$

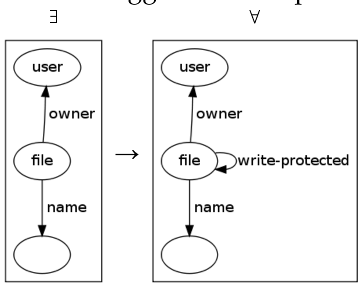
2.2.3 Examples

Here are some $\mathbf{GAC}$s corresponding to the example formulae of logic on subobjects.

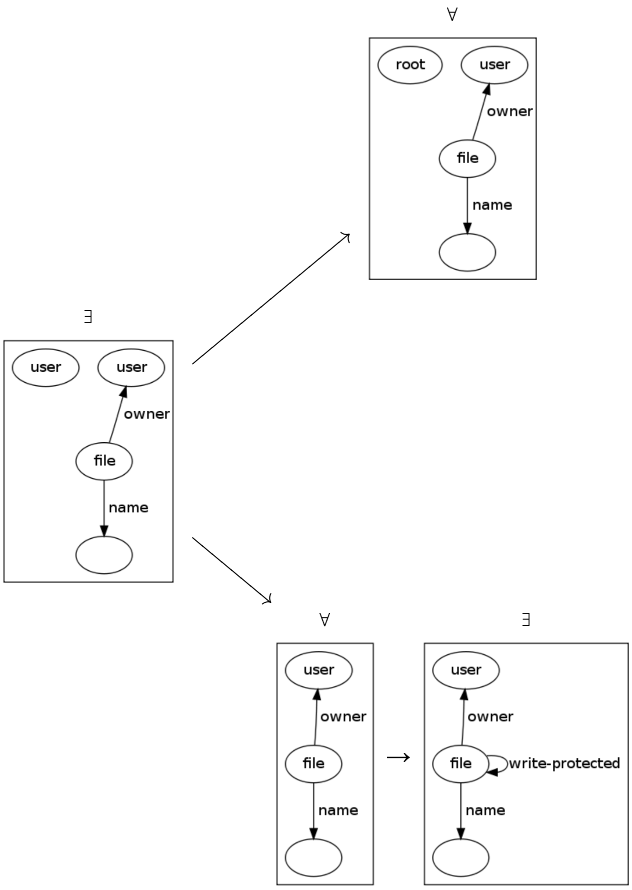
– all files have different names



– the file is tagged as write-protected



– the file can be renamed by the user



2.2.4 Negation

Definition 8 (Negation of a $\mathbf{GAC}$)**.** *Let us define the negation of a* $\mathbf{GAC}$ *as the result of switching every quantifier :*

$$\neg\,(A,\forall,S) := \left(A,\exists,\left\{A \xrightarrow{f} \neg\mathcal{A}' \mid A \xrightarrow{f} \mathcal{A}' \in S\right\}\right)$$
$$\neg\,(A,\exists,S) := \left(A,\forall,\left\{A \xrightarrow{f} \neg\mathcal{A}' \mid A \xrightarrow{f} \mathcal{A}' \in S\right\}\right)$$

It is clear that this function is interpreted as a logical negation ; for all $\mathbf{GAC}$ $\mathcal{A}$, all object C and all morphism $\mathrm{RO}(\mathcal{A}) \xrightarrow{c} C$, we have $C, c \models \neg\mathcal{A}$ if and only if $C, c \not\models \mathcal{A}$.

3 Equivalence

3.1 From GAC to Logic on Subobjects

3.1.1 Case of monos

Let us first consider the following simple case : every arrow in the category is a mono.

In this context, encoding $\mathbf{GAC}$s into formulae of logic on subobjects is quite easy :

Definition 9. *We define a translation from* $\mathbf{GAC}$*s to formulae as follow :*

$$[\![A, \forall, S]\!]\,(x) := \bigwedge_{(f,\mathcal{A}')\in S} \big(\forall\, x' : \mathrm{RO}(\mathcal{A}')\big)\ f \fatsemi x' = x \rightarrow [\![\mathcal{A}']\!]\,(x') \tag{7}$$

$$[\![A, \exists, S]\!]\,(x) := \bigvee_{(f,\mathcal{A}')\in S} \big(\exists\, x' : \mathrm{RO}(\mathcal{A}')\big)\ f \fatsemi x' = x \wedge [\![\mathcal{A}']\!]\,(x') \tag{8}$$

A $\mathbf{GAC}$ and its translation are equivalent in the following meaning :

Theorem 1. *If $\mathcal{A}$ is a $\mathbf{GAC}$ then for all object C and all mono $\mathrm{RO}(\mathcal{A}) \overset{c}{\rightarrowtail} C$, $C, c \models \mathcal{A}$ iff $C, (x \mapsto c) \models [\![\mathcal{A}]\!]\,(x)$*

3.2 General case

In this subsection, we don't assume any more that all morphisms are monos and provide a way to translate any $\mathbf{GAC}$.

Definition 10. *For all object A, we denote by Epi_A the set of all epimorphisms with domain A (up to isomorphism).*

The translation of a $\mathbf{GAC}$ $\mathcal{A}$ by an epi $\mathrm{RO}(\mathcal{A}) \overset{e}{\twoheadrightarrow} B \in \mathrm{Epi}_A$ is defined as follow :

$$(A, \mathcal{Q}, S) \uparrow e := \left(B, \mathcal{Q}, \bigcup_{(f,\mathcal{A}')\in S}\ \bigcup_{e\in \mathrm{Epi}_{\mathrm{RO}(\mathcal{A}')}} \left\{ B \overset{m}{\rightarrowtail} (\mathcal{A}' \uparrow e') \mid f \,;\, e' = e \,;\, m \right\} \right)$$

Theorem 2. *If the category $\mathcal{C}$ is (Epi, Mono)-structured then $\mathcal{A}$ and its translations are equivalent : for all $\mathbf{GAC}$ $\mathcal{A} = (A, \mathcal{Q}, S)$, all object C, all epi $A \overset{e}{\twoheadrightarrow} B \in \mathrm{Epi}_A$ and all mono $B \overset{m}{\rightarrowtail} C$, we have $C, (e \,;\, m) \models \mathcal{A}$ iff $C, m \models_m \mathcal{A} \uparrow e$ where $\models_m$ is the $\mathbf{GAC}$ modeling relation defined in category $\mathcal{C}_m$ which objects are $\mathcal{C}$-objects and arrows are $\mathcal{C}$-monos.*

3.3 From Subobject Logic to GACs

Formulae of first-order logic on subobjects can be encoded into mono-*GAC*s by the following inductive construction :

$$\llbracket f \fatsemi x \sqsubseteq g \fatsemi y \rrbracket^{\eta}_{B} := \begin{cases} (B, \forall, \varnothing) & \text{if } f \; ; \; \eta(x) \leqslant g \; ; \; \eta(y) \\ \\ (B, \exists, \varnothing) & \text{if } f \; ; \; \eta(x) \nleqslant g \; ; \; \eta(y) \end{cases}$$

$$\left\llbracket \bigwedge_{i \in I} \varphi_i \right\rrbracket^{\eta}_{B} := \left(B, \forall, \left\{ B \overset{\mathrm{id}}{\rightarrowtail} \llbracket \varphi_i \rrbracket^{\eta}_{B} \mid i \in I \right\}\right)$$

$$\llbracket \neg \varphi \rrbracket^{\eta}_{B} := \neg \llbracket \varphi \rrbracket^{\eta}_{B}$$

$$\llbracket (\forall\, x : A) \; \varphi \rrbracket^{\eta}_{B} := \Bigg(B, \forall, \Bigg\{ B \overset{m}{\rightarrowtail} \llbracket \varphi \rrbracket^{\eta'}_{B'} \mid \exists e \in \mathrm{Epi}_{A \uplus B}, \mu, m \text{ st.}$$

$$\begin{tikzcd} A \arrow[d, tail, "i_A"'] \arrow[dr, tail, "\mu"] & \\ A \uplus B \arrow[r, two heads, "e"] & B' \\ B \arrow[u, tail, "i_B"] \arrow[ur, tail, "m"'] & \end{tikzcd} \qquad \text{commutes}$$

$$\text{where } \eta' : \begin{cases} x \mapsto \mu \\ y \mapsto \eta(y) \; ; \; m & \text{if } y \neq x \end{cases}$$

$$i_A \text{ and } i_B \text{ are the canonical injections.} \Bigg\} \Bigg)$$

Theorem 3. *In an adhesive category $\mathcal{C}$, for any formula φ of the logic on subobjects, any object C and any C-valuation σ, the following assertions are equivalent :*

- *$C, \sigma \models \varphi$*
- *for all object B, all mono $B \overset{c}{\rightarrowtail} C$ and all B-valuation η such that $\sigma = \eta \; ; \; c$ we have $C, c \models_m \llbracket \varphi \rrbracket^{\eta}_{B}$*
- *there are an object B, a mono $B \overset{c}{\rightarrowtail} C$ and a B-valuation η such that $\sigma = \eta \; ; \; c$ and $C, c \models_m \llbracket \varphi \rrbracket^{\eta}_{B}$*

This result really surprised us ; here is the closed version :

Corollary 1. *In an adhesive category $\mathcal{C}$, for any closed formula φ of the logic on subobjects and any object C we have $C \models \varphi$ iff $C \models_m \llbracket \varphi \rrbracket^{\varnothing}_{\varnothing}$.*

4 Conclusion and Further Research

The very abstract setting of category theory, was really useful because all results are available in almost all graph-like structures (the so called adhesive categories). This abstract level is not too hard to understand because a lot of notions in category theory are generalizations of notions in usual mathematical structure like sets.

Powerful results are also available and their corresponding theorems in specific categories are not always well known. For example the (Epi, Mono)-diagonalization property (see Figure 6) in the category of sets and functions is

Theorem 4. *for all functions f and g, all injective function m and all surjective function e such that $g \circ e = m \circ f$, there exists a unique function d such that $f = d \circ e$ and $g = m \circ d$.*

$$\begin{tikzcd}
{} \arrow[r, two heads, "e"] \arrow[d, "f"'] & {} \arrow[d, "g"] \arrow[dl, dashed, "\exists ! d"'] \\
{} \arrow[r, tail, "m"'] & {}
\end{tikzcd}$$

FIGURE 6 – (Epi, Mono)-diagonalization property

It would be interesting to look at the more expressive cospan-**GAC**s and second-order logic on subobjects ; that is to compare their expressive power and try to encode GACs into second-order formulae.

In case of cospan-GACs, I encountered two difficulties :

– a minor difficulty : we have to encode cospan composition i.e. pushouts (see Figure 7) in the logic of subobjects. This is not extremely hard at second-order level.

– a major difficulty : semantics have a major difference : for the logic on subobjects, we keep the same model (the object C in the definition) in subformulae but in the semantics for cospan-GAC (see Figure 8), the middle-objects of nested cospans may be different.

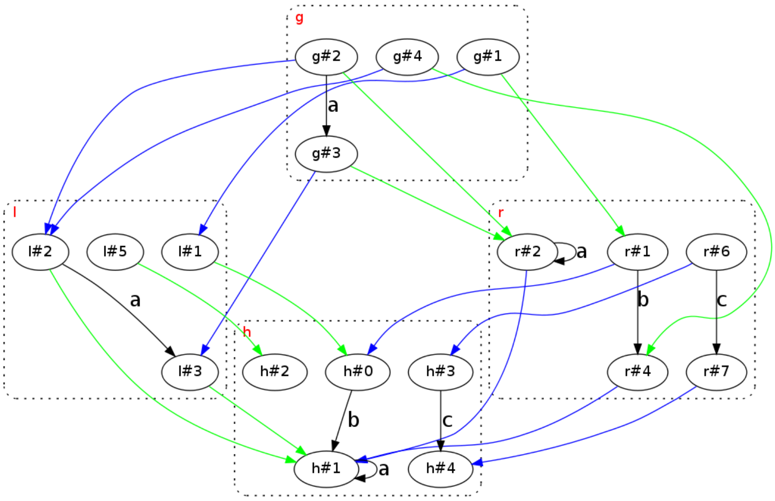


FIGURE 7 – An example of pushout for graphs and graph morphisms

For concrete applications, one would be interested in a comparison of the efficiency of the two representations. It may be possible to find a more concise encoding of formulae into GACs

$$\begin{tikzcd}
\overset{Q}{\boxed{\text{A}}} \arrow[r] \arrow[dr] & I \arrow[d] \arrow[dr, phantom, "\text{pushout}"] & \mathcal{A}' \arrow[l] \arrow[d] \\
 & C & C' \arrow[l] \\
 & & B \arrow[ul] \arrow[u]
\end{tikzcd}$$

FIGURE 8 – cospan-**GAC** semantics, C and C' may be different

Références

[1] H.J. Sander Bruggink and Barbara König. A logic on subobjects and recognizability. In *Proc. of IFIP-TCS '10*, IFIP AICT. Springer, 2010. to appear.

[2] B. Courcelle. The expression of graph properties and graph transformations in monadic second-order logic. In G. Rozenberg, editor, *Handbook of Graph Grammars and Computing by Graph Transformation, Vol.1 : Foundations*, chapter 5. World Scientific, 1997.

[3] Bruno Courcelle. The monadic second-order logic of graphs I. Recognizable sets of finite graphs. *Information and Computation*, 85 :12–75, 1990.

[4] Annegret Habel and Karl-Heinz Pennemann. Nested constraints and application conditions for high-level structures. In *Formal Methods in Software and Systems Modeling. Essays Dedicated to Hartmut Ehrig, on the Occasion of His 60th Birthday*, pages 294–308. Springer, 2005. LNCS 3393.

[5] J. H. Kuperus. Nested quantification in graph transformation rules. Master's thesis, Department of Computer Science, University of Twente, 2007.

[6] Arend Rensink. Representing first-order logic using graphs. In *Proc. of ICGT '04*, pages 319–335. Springer, 2004. LNCS 3256.