Electronic Notes in Theoretical Computer Science 50 No. 3 (2001) – Proc. GT-VMT 2001
URL: http://www.elsevier.nl/locate/entcs/volume50.html 8 pages

An AGG Application Supporting Visual Reasoning [1]

Andrea Formisano [2]

Dipartimento di Matematica e Informatica, Università di Perugia

Marta Simeoni [3]

Dipartimento di Informatica, Università 'Cà Foscari' *di Venezia*

1 Introduction

The *map calculus*, the arithmetic of dyadic relations (cf. [9]), is a ground equational formalism where one can state properties of dyadic relations over a unspecified domain of discourse. It can be seen as an evolution of the theory of relations, an algebraic approach to logic developed by, among others, A. de Morgan, C. S. Peirce and E. Schröder [8,7].

Direct algebraic manipulation of map expressions seems to be, for human beings, much less natural than developing inferences in first-order logic; it may in fact appear to be overly machine-oriented for direct hand-based exploitation. However, the situation radically changes when one resorts to a convenient representation of map expressions based on labeled graphs. Beside allowing the abstraction w.r.t inessential features of expressions, such a representation allows an easy and intuitive visual handling of map specifications. Approaches of this kind have been proposed, for instance, in [2,1,3,6].

In this work we move the first step toward the implementation of an automated tool for the mechanization of visual map-reasoning. To this end, we exploit AGG —Algebraic Graph Grammar— [4], which provides a visual programming environment for graph transformation based applications. Such applications are described by graph grammars, which consist of an initial graph and a set of graph rewriting rules. AGG supports the visual handling of both the start graph and the rewriting rules, and, once the graph grammar has been formalized, it allows the manipulation of the start graph. We use

[1] This research was partially funded by the Italian IASI-CNR (coordinated project log(SETA)); by MURST—PGR-2000; by the EC TMR Network GETGRATS (GEneral Theory of GRAph Transformation Systems); and by Esprit Working Group APPLIGRAPH.
[2] Email:formis@dipmat.unipg.it
[3] Email:simeoni@dsi.unive.it

Agg as a visual proof-assistant for map reasoning: inference mechanisms are implemented as graph rewriting rules and techniques, allowing the transformation of premises into conclusions (forward reasoning), or the reduction of theses —goals— into simpler goals and, ultimately, into known facts (backward reasoning).

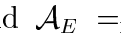
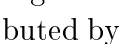
A detailed exposition of the topics of this paper, comprising a series of worked exampes, can be found in [5].

2 A graphical rendering for equational specifications

The map calculus is based on $\mathcal{L}^{\times}$, an equational language devoid of variables. Its basic ingredients are three *constants* $\emptyset$, $\mathbb{1}$, $\boldsymbol{\iota}$; infinitely many *map letters* $\mathsf{p}_1, \mathsf{p}_2, \mathsf{p}_3, \ldots$; three dyadic constructs $\cap$, $\triangle$, $;$ of map *intersection*, map *symmetric difference*, and map *composition*; and the monadic construct $^{\smile}$ of map *conversion*. A *map expression* is any term built up from this signature in the usual manner. A *map equality* is a writing of the form $Q{=}R$, where both Q and R are map expressions.

Once a nonempty domain $\mathcal{U}$ has been fixed, the map constants $\emptyset$, $\mathbb{1}$, and $\boldsymbol{\iota}$ are interpreted by putting: $\emptyset^{\Im} =_{\text{Def}} \emptyset$, $\mathbb{1}^{\Im} =_{\text{Def}} \mathcal{U}^2 =_{\text{Def}} \mathcal{U} \times \mathcal{U}$, and $\boldsymbol{\iota}^{\Im} =_{\text{Def}} \{[a,a] : a \in \mathcal{U}\}$. On the basis of the usual evaluation rules, by putting subsets $\mathsf{p}_1^{\Im}, \mathsf{p}_2^{\Im}, \mathsf{p}_3^{\Im}, \ldots$ of $\mathcal{U}^2$ in correspondence with the p_is, each map expression P comes to designate a specific map $P^{\Im}$, and each equality $Q{=}R$ turns out to be either true or false.

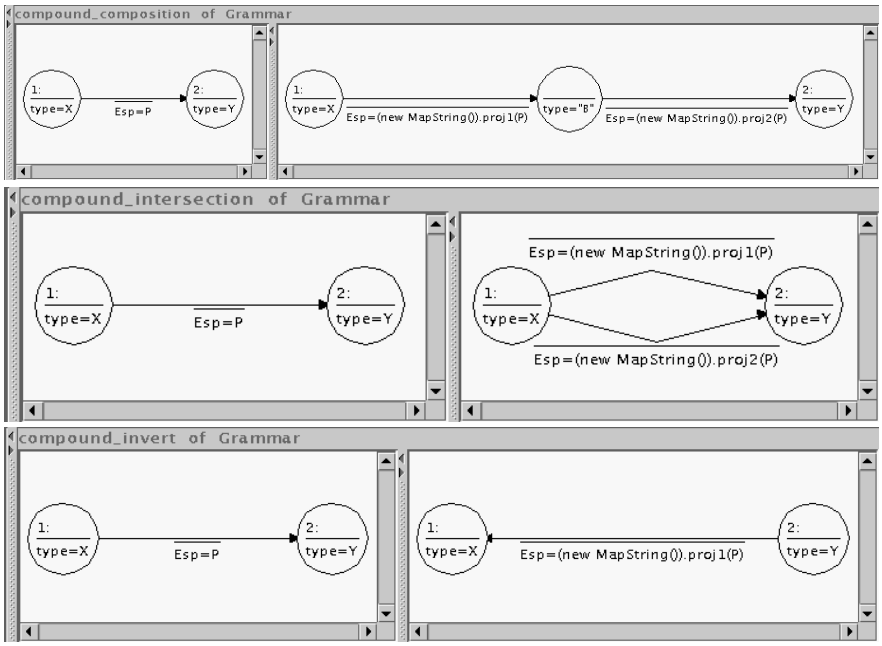
Let us now assume that φ is a conjunction $(\exists x_1)\cdots(\exists x_k)(L_1 \wedge \cdots \wedge L_n)$ of atoms, and that $vars(\varphi) \subseteq \{\mathsf{v}_1, \mathsf{v}_2, \ldots\}$ is the set of all the variables occurring in φ. Moreover, assume that $x_1, \ldots, x_k \in vars(\varphi)$ and that each L_i is of the form $x_{L_i} P_{L_i} y_{L_i}$, where $x_{L_i}, y_{L_i} \in vars(\varphi)$ are variables (assumed ranging over the domain $\mathcal{U}$ of discourse), and P_{L_i} is a map expression of $\mathcal{L}^{\times}$. Clearly, free variables may occur in φ intermixed with existentially quantified variables.

An Agg graph G_φ representing φ is so defined: given the two alphabets $\mathcal{A}_N =_{\text{Def}} \{$  $\}$ and $\mathcal{A}_E =_{\text{Def}} \{$  $\}$, for nodes and edges (graphical) labels, we have $G_\varphi = \langle G_E, G_N, G_s, G_t, G_{le}, G_{ln}\rangle$ where:
$G_E =_{\text{Def}} \{1, \ldots, n\}$ is the set of edges;
$G_N =_{\text{Def}} vars(\varphi)$ is the set of nodes;
$G_s : G_E \to G_N$ maps each $i \in G_E$ into x_{L_i} (source function);
$G_t : G_E \to G_N$ maps each $i \in G_E$ into y_{L_i} (target function);
$G_{ln} : G_N \to \mathcal{A}_N$ maps each node to the unique available label;
$G_{le} : G_E \to \mathcal{A}_E$ maps each edge to  .

Agg graphs may be attributed by Java objects (i.e instances of Java classes, either imported from standard libraries or user defined): we exploit the attribution mechanism in order to permit suitable manipulation of map expressions. More precisely, we introduce the node attribution function $\mathtt{type} : G_N \to \{\mathtt{B}, \mathtt{F}\}$ such that, for each node x, $\mathtt{type}(x)$ is $\mathtt{B}$ (resp. $\mathtt{F}$) if x is a bound (resp. free) variable in φ. Accordingly, a node is said *bound* if the corresponding

variable is bound in φ. We consider moreover the edge attribution function $\texttt{Esp} : G_E \to \mathcal{L}^{\times}$ which associates each edge i, with the map expression P_{L_i}.

The following Agg rules, which manifestly preserve the meaning of the involved graphs, can be combined to constitute an algorithm for associating an 'unfolded' graph G to a given (compound) map expression P. Two designated (free) nodes, named *source* and *sink*, represent the two arguments of P, and every node distinct from these two is regarded as being bound.



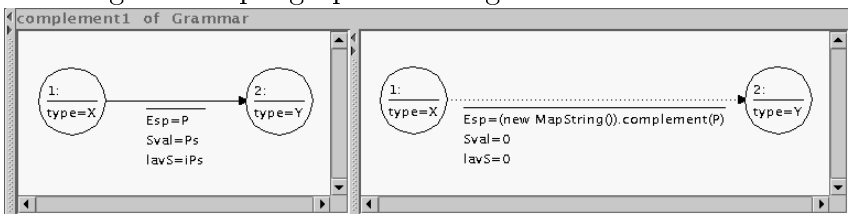
Note that X, Y, and P are variables which are instantiated with concrete attribute values when the left-hand side graphs are applied to a certain graph. The rules are applicable when P is instantiated with $P_1 \cap P_2$ (first rule), $P_1;P_2$ (second rule) and $P_1^{\smile}$ (last rule), where P_1 and P_2 stand for general map expressions. Their applicability is checked via Java methods. Moreover, the Java methods `proj1` and `proj2` return the map expressions P_1 and P_2, respectively.

Representing map equalities. Let us now extend our approach in order to suitably represent map equalities. To this end we convert the graph representation of a relation P (i.e., (xPy)) into the representation of an equality involving P by bounding the source x and sink y by universal or existential quantification. The following equalities (shown on the left-hand side), correspond to closed first-order formulas (shown on the middle), and are rendered in our graphical notation by means of particular node attribute values (shown on the right-hand side).

$P{=}\mathbb{1}$;	$(\forall x)(\forall y)(xPy)$	$\texttt{type}(x) = \texttt{A}$, $\texttt{type}(y) = \texttt{A}$
$P;\mathbb{1}{=}\mathbb{1}$, i.e., Total(P)	$(\forall x)(\exists y)(xPy)$	$\texttt{type}(x) = \texttt{A}$, $\texttt{type}(y) = \texttt{E}$
$P{\neq}\emptyset$, i.e., Total($\mathbb{1};P$);	$(\exists x)(\exists y)(xPy)$	$\texttt{type}(x) = \texttt{E}$, $\texttt{type}(y) = \texttt{E}$
$P{=}\emptyset$	$\neg(\exists x)(\exists y)(xPy)$	$\texttt{type}(x) = \texttt{NE}$, $\texttt{type}(y) = \texttt{E}$

The node attribute function has been refined to $\texttt{type} : G_N \to \{\texttt{B}, \texttt{F}, \texttt{A}, \texttt{E}, \texttt{NE}\}$ and the source x and sink y of the graph representing P are attributed according with the equality to be expressed. We refer to the four kind of graph listed above, by the writings $\forall\forall$, $\forall\exists$, $\exists\exists$, and $\neg\exists\exists$, respectively.

Dealing with complementation. In order to allow (partial) handling of complementation, we introduce the edge label: ⋯▸ , to denote that the associated attributed expression, say P, has to be intended as complemented. The following is a simple graph-rewriting rule related to this convention:



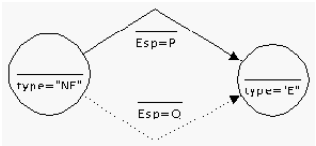
As we will see, a simple (albeit partial), treatment of complementation, allows one to model/represent map inclusions of the form $P \subseteq Q$ and, consequentely, to describe simple rewriting rules for inferring (new) map inclusions.

3 A proof-assistant based on graph-rewriting techniques

In our context, deriving map equalities from proper axioms, and from laws already known, can be viewed as a graph-rewriting activity. Here we focus our treatment on two particular classes of graphs: the existential graphs of the two types $\forall\forall$ and $\neg\exists\exists$. We call them *positive* and *negative* graphs, respectively. These graphs are exploited to represent premises and conclusions, theses, etc.

From this perspective, inference mechanisms are seen as graph-rewriting techniques: in forward reasoning, rewriting rules are used to transform premises into conclusions; in backward reasoning, to reduce theses —'goals', as they are often called— into simpler goals and, ultimately, into known and perhaps obvious facts.

As a basic principle, it is legitimate to replace a positive goal by a more demanding one, and a negative goal by one less demanding. E.g., new attributed edges can be added at will to a positive goal, whereas edges can be removed from a negative goal. Solving the new goal, although not necessarily equivalent to solving the previous goal, will in fact suffice for the purpose. Quite often a negative premise represents an inclusion $P \subseteq Q$, i.e., $\neg(\exists x)(\exists y)(x\, P \cap \overline{Q}\, y)$:



If a subgraph of a positive goal matches the part of the premise which represents Q, then it can be replaced by the part representing P; in a negative goal, on the opposite, Q may replace P.

From now on, to be more specific, let us consider negative graphs only.

The above-outlined basic rule is formulated in the Agg graphical language as displayed in Fig.1(1), where the nodes marked `4:` and `5:` constitute (together with the edges marked `10:` and `12:`) the negative premise; while the nodes `6:` and `9:` identify the edge to be replaced during the firing of the rule. Notice that the negative premise $P \subseteq Q$ remains unchanged in the transformed graph. Moreover, to ensure the soundness of the rule, further conditions have to be imposed. In fact, during the application of the rule, the negative premise must match with a complete connected component of the graph.

The properties of inclusion (e.g., transitivity) easily determine the simple inference rules of Fig.2; further rules for negative goals are shown in Fig.3.

As mentioned, proving a particular map equality in this visual framework amounts to repeatedly reducing the corresponding graph/goal into simplier goals until known facts are derived. Some of these facts, or *axioms*, related to map-inclusion are:

$(A_1)\ \ P \subseteq \mathbb{1}, \qquad (A_2)\ \ P \subseteq P, \qquad (A_3)\ \ \mathbb{1} \subseteq P \cup \overline{P}, \qquad (A_4)\ \ \iota\,;P \subseteq P.$

An axiom corresponds to an Agg production whose left-hand side contain the obvious inclusions while the right-hand side is empty. The application of a production of this kind removes obvious facts from the graph.

We describe now a simple example of assisted reasoning. More precisely, we prove that for any map expression N the following holds: [4]

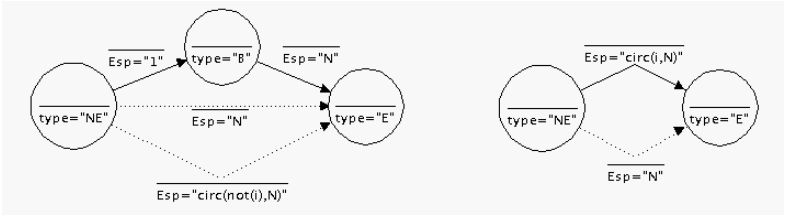
$$\mathsf{Func}(N^{\smile}) \ \rightarrow\ \mathbb{1}\,;N - N \ = \ \overline{\iota}\,;N.$$

We split the problem into two simpler theses:

(a) $\mathbb{1}\,;N - N \subseteq \overline{\iota}\,;N$; and **(b)** $\mathsf{Func}(N^{\smile}) \ \rightarrow\ \overline{\iota}\,;N \ \subseteq\ \mathbb{1}\,;N - N$,

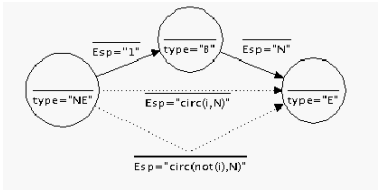
The following are the proofs of **(a)** and **(b)** as obtained by exploiting Agg.

(a) The representation of $\mathbb{1}\,;N - N \ \subseteq\ \overline{\iota}\,;N$ as a $\neg\exists\exists$-graph is:

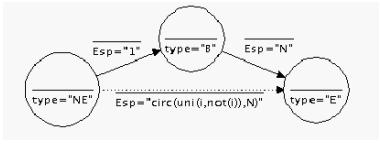


On the right we have added an instance of the axiom (A_4) since the first step will consist in applying rule (2) of Fig.1. As a result we obtain the graph (i) below, which can be rewritten into graph (ii) by applying a rule implementing a generalization of the 'De Morgan' law $\overline{P\,;T} \cap \overline{Q\,;T} = \overline{(P \cup Q)\,;T}$.

(i)



(ii)



By (1) of Fig.3 Agg reduces the goal (ii) into the two goals:

[4] Here $\mathsf{Func}(P)$ stands for the map equality $P \cap \overline{P\,;\overline{\iota}} = P$.

(1)

(2)

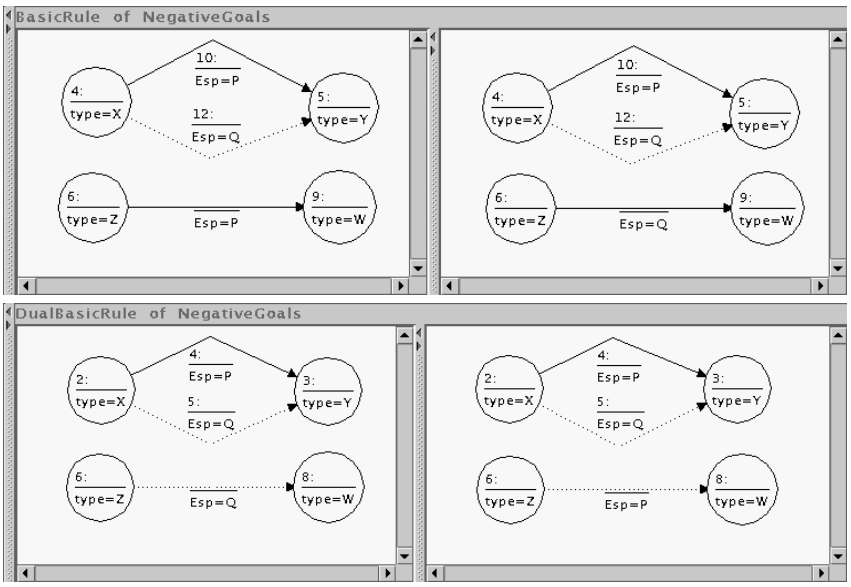


Fig. 1. Basic inference rules for $\neg\exists\exists$-goals

(1)

(2)

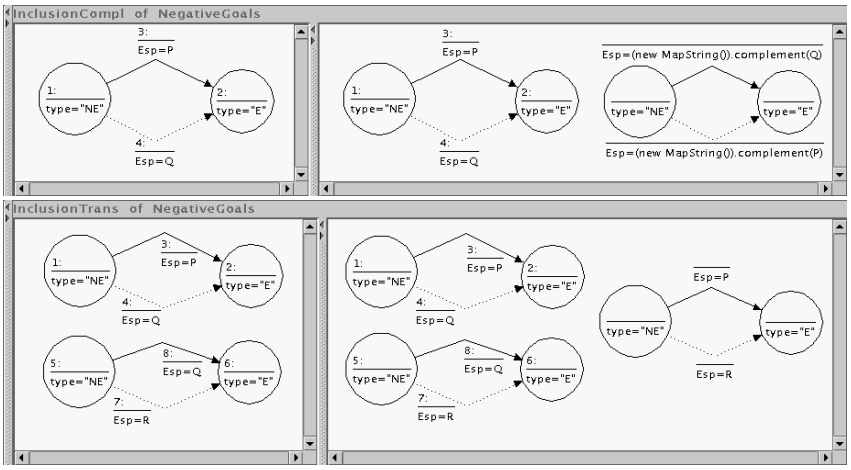
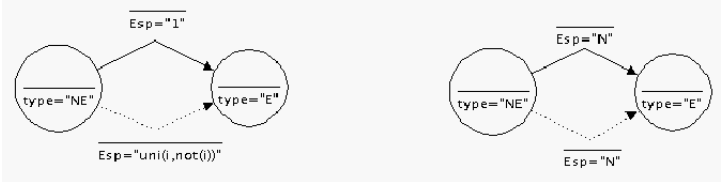


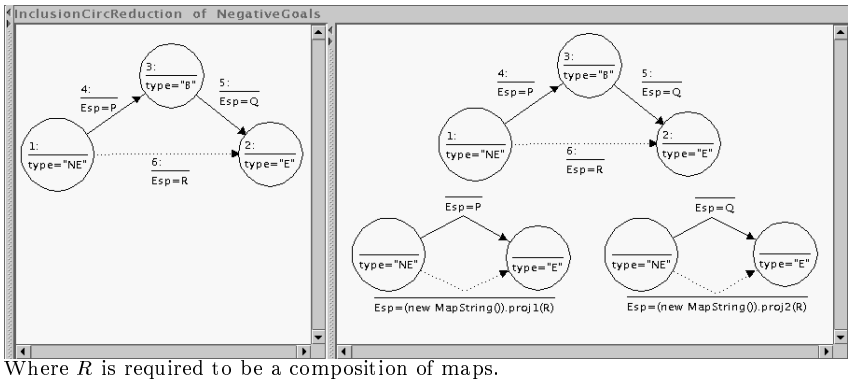
Fig. 2. Simple inference rules for map inclusion



which are then immediately proved by using (A_3) and (A_2), respectively. This completes the first part of the proof.

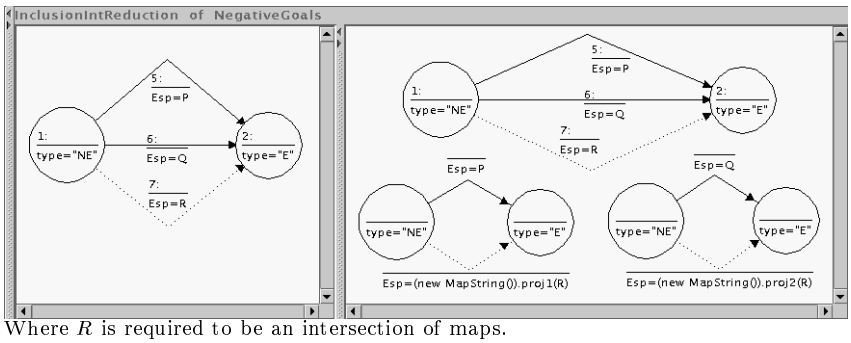
(b) The assumption $\mathsf{Func}(\,N^{\smile}\,)$ can be rewriten as $\overline{\boldsymbol{\tau}};N\subseteq\overline{N}$, and represented by the graph (i) below; while our thesis originates the starting graph (ii)

(1)



Where R is required to be a composition of maps.

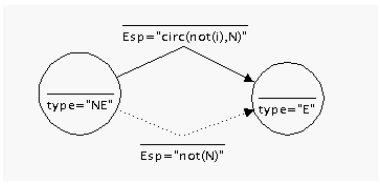
(2)



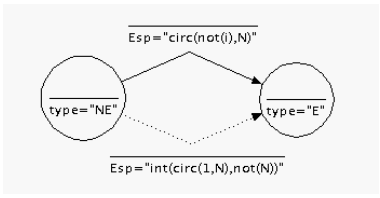
Where R is required to be an intersection of maps.

Fig. 3. Simple reduction rules for $\neg\exists\exists$-goals involving map inclusion

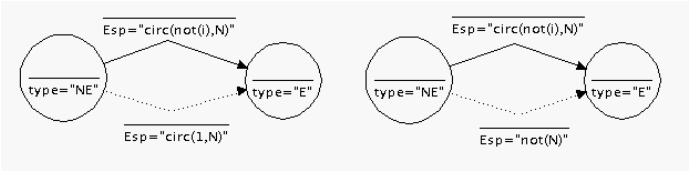
(i)



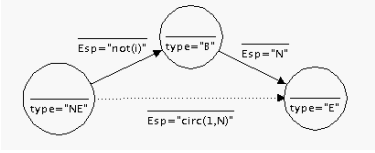
(ii)



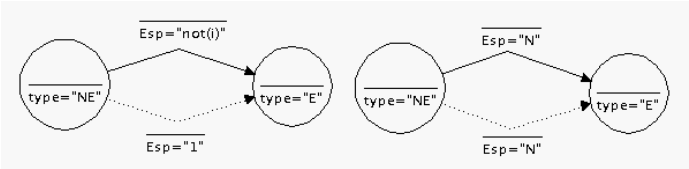
By applying an instance of rule (2) of Fig.3 —obtained identifying P and Q in (2)— Agg reduces the thesis to two subgoals:



The goal on the right corresponds to our hypothesis $\mathsf{Func}(N^{\smile})$; the other one can be further reduced as follows



by applying the 'compound_composition' rule of page 3. A further step reduces the remaining goal by means of (1) of Fig.3, yielding the two goals:



which are instances of (A_1) and (A_2), resp.. This completes the proof.

4 Concluding remarks

In this paper we reported on a first attempt in implementing graphical techniques for map representation and reasoning. An interesting further development would consist in the design and implementation of a more sophisticated proof-assistant: consider for instance the capability of performing backtracking or suggesting the user the 'better' rule to apply. The implementation of these features could be considered as a first step toward the realization of a tool for automated map-reasoning based on graph-transformation techniques.

References

[1] Cantone, D., A. Formisano, E. G. Omodeo and C. G. Zarba, *Compiling dyadic first-order specifications into map algebra*, in: *Proc. 2nd AMAST—AMILP 2000* (2000), pp. 35–54.

[2] Chiacchiaretta, A., A. Formisano and E. G. Omodeo, *Map reasoning through existential multigraphs*, Tech. Rep. 05/00, Dip. di Matematica Pura ed Applicata, Univ. di L'Aquila (2000).

[3] Curtis, S. and G. Lowe, *Proofs with graphs*, Science of Computer Programming **26** (1996), pp. 197–216, Mathematics of program construction, Kloster Irsee.

[4] Ermel, C., M. Rudolf and G. Taentzer, *The* AGG *approach: Language and environment*, in: H. Ehrig, G. Engels, H.-J. Kreowski and G. Rozenberg, editors, *Handbook of Graph Grammars and Computing by Graph Transformation, Vol. 2.*, World Scientific, Singapore (1999).

[5] Formisano, A. and M. Simeoni, *Graphs and maps: rewriting techniques at work*, Tech. Rep. 2001-01, TU-Berlin (2001).

[6] Kahl, W., *Relational matching for graphical calculi of relations*, Information Sciences **119** (1999), pp. 253–273.

[7] Maddux, R. D., *The origin of relation algebras in the development and axiomatization of the calculus of relations*, Studia Logica **50** (1991).

[8] Schröder, E., "Vorlesungen über die Algebra der Logik (exakte Logik)," B. Teubner, Leipzig, 1891–95, [Reprinted by Chelsea Pub. Co., New York, (1966)].

[9] Tarski, A. and S. Givant, "A formalization of Set Theory without variables," Colloquium Publications **41**, American Mathematical Society (1987).