A graphical representation of relational formulae with complementation

D. Cantone* A. Formisano† M. Nicolosi Asmundo* E. G. Omodeo‡

The possibility to exploit map calculus for mechanical reasoning can be grasped from [8], where Tarski and Givant show how to reformulate most axiomatic systems of Set Theory as equational theories based on a relational language devoid of quantifiers.

Map calculus [5] cannot represent *per se* an alternative to predicate calculus. As for expressive power, it corresponds in fact to a fragment of first order logic endowed with only three individual variables and with binary predicates only. As for deductive power, it is semantically incomplete: there are semantically valid equations which are not derivable within it. Moreover, predicate logic has acquired such an unquestioned status of *de facto* standard as to make one reluctant to adopt the map formalism in its stead, in spite of the greater conciseness of the latter.

Nonetheless, map calculus can be applied in synergy with predicate calculus, inside theorem provers or proof assistants, as an inferential engine to be used in the activities of proof-search and model building [4, 6, 1]. This calls for cross-translation algorithms between predicate logic and map calculus. Moreover, to increase readability by exploiting the immediate perspicuity of graphics, it is useful to design algorithms allowing one to represent formulae in a visually alluring way. In [2, 3] both problems have been addressed by presenting an algorithm for translating formulae of dyadic predicate logic into map calculus, and an algorithm for converting map expressions into a graphical representation. Both algorithms are based on suitably defined graphs; one of them is designed to treat existentially quantified conjunctions of literals, the other to treat map expressions containing the constructs of relational intersection, composition, and conversion.

In this paper the techniques introduced in [2, 3] are further extended to the treatment of formulae which involve the negation connective and of expressions involving the relational complement construct. This allows us to get a graphical representation of any map expression, and to process any formula of dyadic predicate logic with the aim of getting an equivalent map equation. In the latter case, the algorithm which we will present sometimes fails to find the sought translation even if it exists. This apparent drawback, which also affected the earlier versions of the algorithm, stems from an unsurmountable limiting result [8], namely the fact that no algorithm can establish in full generality whether a given sentence in $n+1$ variables is logically equivalent to some other sentence in n variables, for any integer number $n > 1$.

*Università di Catania, email: cantone@dmi.unict.it, nicolosi@dmi.unict.it
†Università di Perugia, email: formis@dmi.unipg.it
‡Università di Trieste, email: eomodeo@units.it

The enhanced techniques in this paper have been obtained by enriching the notion of directed multigraph (i.e., graphs allowing multiple edges and self-loops) associated with a formula or with a map expression given in [2, 3]. Here, the nodes of the multigraph are labeled with sets of variables instead of with single variables while edges are labeled with sets of formulae. Moreover, a multigraph is seen as a collection of disjoint subgraphs, called *components*. A relation $\rightsquigarrow$ is introduced between edges and components of the multigraph. Intuitively, such a relation associates each disjunctive subformula with subgraphs representing the complements of its disjuncts. This allows to "unfold" the nesting of negations and disjunctions in terms of conjunctions (as well as complementations and unions in term of intersections), so as to apply the techniques of [2, 3] to subgraphs/subformulae devoid of complementation and disjunction.

Graphical representation of $\mathcal{L}^+$-formulae

$\mathcal{L}^\times$is an equational language devoid of variables. Its basic ingredients are three *constants* $\mathbf{0}$, $\mathbf{1}$, ι, *map letters* $\mathsf{p}_1, \mathsf{p}_2, \mathsf{p}_3, \ldots$, dyadic constructs $\cap$ (map *intersection*), $\cup$ (map *union*), ";" (map *composition*), and the monadic constructs $\overline{\ }$ (map *complementation*) and $\breve{\ }$ (*conversion*). A *map expression* is any term $P, Q, R, \ldots$ 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.

The language $\mathcal{L}^+$ is a variant of a first-order dyadic predicate language: an *atomic formula* of $\mathcal{L}^+$ has either the form xQy or the form Q=R, where x, y stand for individual variables and Q, R stand for map expressions. Here propositional connectives and existential/universal quantifiers are employed as usual.

The techniques presented in [2, 3] for representing map expressions and, more generally, formulae of $\mathcal{L}^+$ are extended here to cope with the negation connective ($\neg$) and the relational complement construct ($\overline{\ }$).

Let φ be a formula of $\mathcal{L}^+$ and let φ be constructed out of atomic formulae of the form xPy. Then φ can be represented by a multigraph $G_\varphi = (V_\varphi, E_\varphi)$ built up in such a way that:

- G_φ has at least one node in V_φ for each distinct variable in φ.
- G_φ is provided with a function $lNode$ labelling nodes with a subsets of variables of φ, and with a function $lEdge$ labelling each edge in E_φ with either a set of map expressions or a singleton containing a formula. The cardinality of the set equals the multiplicity of the edge.

A node of G_φ is *free* if its label contains no bound variables. Otherwise it is *bound*. If φ has the form xPy, we often say that G_φ represents the map expression P.

As a simple example, consider the formula $\varphi_1 = x\overline{P} \cap Qy$. Then G_{φ_1} is such that $V_{\varphi_1} = \{u_0, v_0\}$, the only edge is (u_0, v_0) with multiplicity 2, $lEdge((u_0, v_0)) = \{\overline{P}, Q\}$, $lNode(u_0) = \{x\}$ and $lNode(v_0) = \{y\}$.

A disjunctive formula φ (i.e., a formula of type $xPy \vee zQw$ or of type $x(P \cup Q)y$) is represented by a multigraph with several components. One component has a single edge labelled by $\{\varphi\}$ and is in relation $\rightsquigarrow$ with other components that, taken together, represent the dual of φ. For instance, let $\varphi_2 = xP; Qy \vee zR \cup Sw$. G_{φ_2} is the directed multigraph with components G_0, G_1, G_2 and G_3 illustrated in Fig. 1. The component G_0 is in relation $\rightsquigarrow$ with both G_1 and G_2, whereas G_1 is in relation $\rightsquigarrow$ with G_3 (where r_1 is the only bound node, implicit in $xP; Qy$).

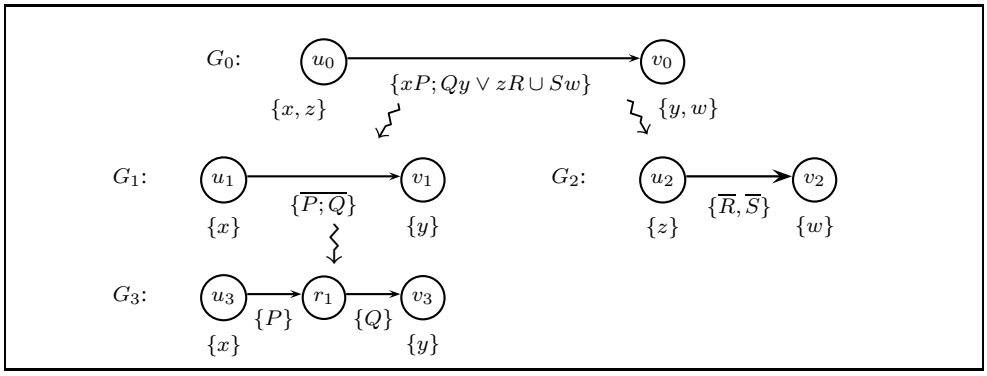

Figure 1: The multigraph G_{φ_2} associated with $\varphi_2 = xP; Qy \vee zR \cup Sw$.

The graph-fattening algorithm

The graph-fattening algorithm constructs a multigraph G associated to a map expression P. It extends the one presented in [3], which is unable to deal with map complementation and disjunction. Such a multigraph has two distinct nodes s_0 and s_1 (called source and sink) representing the two arguments of P (seen as the atomic formula xPy). The construction of G, s_0, and s_1 is carried out by starting with a graph G consisting of a single edge from s_0 to s_1, such that $lEdge((s_0, s_1)) = \{P\}$. Then, G is "expanded" by unfolding the constructs of $\mathcal{L}^{\times}$. For instance, let the disjoint multigraphs G', s'_0, s'_1 and G'', s''_0, s''_1 represent Q and R, respectively. Then, as regards composition and intersection:

- G, s_0, s_1 representing $Q; R$ is obtained from G' and G'' by 'gluing' together s'_1 and s''_0 to form a single node, and by putting $s_0 = s'_0$ and $s_1 = s''_1$;
- G, s_0, s_1 representing $Q \cap R$ is obtained from G' and G'' by gluing s''_0 to s'_0 to form s_0, and s''_1 to s'_1 to form s_1.

Here, for space limits, we avoid describing the treatment of the remaining constructs. In that cases the representation exploits the relation $\rightsquigarrow$ and graphs with multiple components.

The graph-thinning algorithm

The aim of the graph-thinning algorithm is to determine, out of a given formula φ of $\mathcal{L}^+$, an equivalent formula ψ where all the quantifiers have been eliminated. A simpler version of this problem has been analyzed in [3], by designing an algorithm that looks for a quantifier-free formula of $\mathcal{L}^+$ equivalent to a given existentially quantified conjunction of literals of the form xPy. The refined algorithm we introduce here transforms the input formula φ by operating on its subformulae, in a bottom-up manner, with the purpose of deriving, at the same time, both ψ and G_ψ. Notice that our algorithm may fail in this task. However, this does not mean that φ does not admit a quantifier-free translation in $\mathcal{L}^+$, but it simply witnesses the incompleteness of our algorithm in solving a problem which is, in fact, undecidable in full generality.

The construction process of ψ and G_ψ is carried out through a bottom-up visit of the syntax tree of φ. G_ψ is obtained by first constructing the graph components relative to the innermost subformulae of φ and then gluing them together using the rules of *collapse* (construction of

maximal conjunctions of atomic subformulae), of *normalization*, of *fusion* of multiple edges, of *bypass* and *bigamy*. At the same time, the formula ψ is derived from φ by replacing each subformula of φ with the formula represented by the corresponding component of G_ψ. Because of the mentioned way of rendering negation through the relation $\rightsquigarrow$, applications of simplification and gluing rules can be restricted to conjunctions of atomic formulae only. Disjunctions like $\beta_1 \vee \ldots \vee \beta_n$ are put in the form $\neg(\neg\beta_1 \wedge \ldots \wedge \neg\beta_n)$. In turn, the edge of G labeled by $\neg(\neg\beta_1 \wedge \ldots \wedge \neg\beta_n)$ is put in relation $\rightsquigarrow$ with the components of G representing the β_is.

Conclusion and Future Work

We have enhanced preexisting algorithms for translating dyadic first-order logic into map calculus which rely on a specific graph representation of map expressions. As an outcome, we are able to deal with map expressions and with formulae containing the relational complement construct and the negation connective. We plan to further enhance the graph-thinning algorithm by improving the bypass and bigamy rules (making them more liberal in case of nesting of quantifiers) and by incorporating into the algorithm rules that exploit semantical information, such as functionality of maps (i.e., the knowledge that an expression P is single-valued). This kind of rules could enable an otherwise unachievable translation. A first attempt in the implementation of a proof-assistant based on the graph representation of map expressions has been done in [7]. In that case the algorithms described in [3] have been implemented on top of the attributed graph-transformation system AGG. In order to validate the approach described in this paper, an implementation of the new algorithms is due (as a stand-alone tool or in integration with standard theorem provers for first-order logic). This represents a challenging topic for further research.

References

[1] Belinfante, J.G.F. (2000). Gödel's algorithm for class formation. In D.McAllester, ed, *Proc. of CADE'00*.

[2] Cantone, D., Cavarra, A., and Omodeo, E. G. (1997). On existentially quantified conjunctions of atomic formulae of $\mathcal{L}^+$. In M. P. Bonacina and U. Furbach, eds, *Proc. of FTP'97*.

[3] Cantone, D., Formisano, A., Omodeo, E. G., and Zarba, C. G. (2003) Compiling dyadic first-order specifications into map algebra. TCS 293(2):447-475.

[4] Formisano, A., Nicolosi Asmundo, M. (2006). An efficient relational deductive system for propositional non-classical logics. *J. Applied Non-Classical Logics*, 16(3-4):367-408.

[5] Formisano, A., Omodeo, E. G., and Temperini, M. (2000). Goals and benchmarks for automated map reasoning. *Journal of Symbolic Computation*, 29(2):259-297.

[6] Formisano, A., Omodeo, E. G., and Temperini, M. (2001) Instructing equational set-reasoning with Otter. In R. Goré, A. Leitsch, and T. Nipkow, eds, *Proc. of IJCAR'01*.

[7] Formisano, A., and Simeoni, M. (2001). An AGG application supporting visual reasoning. In L. Baresi and M. Pezzè, eds., *Proc. of GT-VMT'01 (ICALP 2001)*. ENTCS, 50(3).

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