

Relation-Algebraic Derivation of Spanning Tree Algorithms

Rudolf Berghammer, Burghard von Karger, and Andreas Wolf

Institut für Informatik und Praktische Mathematik
Christian-Albrechts-Universität Kiel
Preusserstraße 1–9, D–24105 Kiel, Germany

Abstract. We use Tarski’s relational algebra to derive a series of algorithms for computing spanning trees of undirected graphs, including a variant of Prim’s minimum spanning tree algorithm.

1 Introduction

The relational calculus has been very successful in the derivation and proof of algorithms for directed graphs. We claim that it is equally suitable for reasoning about undirected and even about weighted graphs. To prove our point we derive a series of increasingly powerful spanning tree algorithms which culminates in a variant of Prim’s well-known algorithm for computing a spanning tree with minimal weight.

Directed graphs and relations are essentially the same, but there are (at least) two natural ways of representing undirected graphs as relations. The first possibility are symmetric relations on vertices, also known as *adjacency relations*. This representation has the advantage of simplicity; it is well suited for calculations. Alternatively we can use *incidence relations* between the set of edges and the set of vertices. Incidence relations are harder to calculate with but they are more easily generalized to multigraphs and weighted graphs. The two representations are linked by a Galois connection. Thus, we can derive a spanning tree algorithm for adjacency relations in the calculus of symmetric relations and then transform it into an algorithm for incidence relations, which can then be further refined into a minimum spanning tree algorithm.

The remainder of this paper is organized as follows. In Sec. 2 we derive a spanning tree algorithm for adjacency relations. Then in Sec. 3 we show how a Galois connection may be used for data reification. This procedure is instantiated in Sec. 4 to produce a spanning tree algorithm for incidence relations which in Sec. 5 is refined to a minimum spanning tree algorithm. Sec. 6 gives the proofs of various facts used in the derivations. In the appendix we explain how our programs can be implemented in the relational toolbox RELVIEW.

2 Adjacency Relations

Given a connected (undirected) graph G , we are asked to compute a subgraph T of G with the following properties:

1. T is connected and cycle-free (T is a tree).
2. Every vertex of G is also a vertex of T (T spans G).

Let us assume that G is given as an *adjacency relation*, that is an irreflexive and symmetric relation on a set V (the set of vertices), and that the output T is required in the same format. Representing graphs as relations facilitates the *implementation* of algorithms, because relations can be encoded efficiently as Boolean matrices, linked lists, or binary decision diagrams. It also facilitates the *design* of algorithms, because relations are the objects of a concise algebraic calculus (which was formalized in 1941 by Tarski, see also [11, 4]). To advantage ourselves of these twin benefits let us translate the problem specification into the language of relations. We start with a couple of definitions that fix the notation and introduce some basic properties.

Definition 1 (Relational Operators and Constants) *We consider binary relations on the fixed set V . The largest and the smallest such relation are written as $\mathbf{1}$ and $\mathbf{0}$, respectively. Since we work in the fixed universe $V \times V$, every relation R has a complement \bar{R} . The composition of two relations is denoted RS , and R^{\cup} is the converse of R . The diagonal relation, which is the unit of composition, is written as \mathbf{I} . And, finally, R^* is the transitive and reflexive closure of R .*

Definition 2 (Properties of Relations) *A relation R is called symmetric if $R^{\cup} = R$, asymmetric if $R \cap R^{\cup} = \mathbf{0}$, antisymmetric if $R \cap R^{\cup} = \mathbf{I}$, irreflexive if $\mathbf{I} \cap R = \mathbf{0}$, connected if $R \neq \mathbf{0}$ and $RLR \subseteq R^*$, one-step-connected if $R \neq \mathbf{0}$ and $RLR \subseteq \mathbf{I} \cup R$.*

Definition 3 (Symmetric Relations) *For any $R \subseteq V \times V$ let $Symm(R)$ denote the set of all irreflexive and symmetric subrelations of R ; this is a complete atomistic Boolean algebra. If $R, S \in Symm(\mathbf{L})$ we say that R spans S if $R \subseteq S$ and $RL = SL$.*

Definition 4 (Atoms and Edges) *If \mathcal{L} is a lattice and $R \in \mathcal{L}$ then $atoms_{\mathcal{L}}(R)$ denotes the set of atoms (minimal non-zero elements) that are below R . In particular, for $R \in Symm(\mathbf{L})$ the elements of $edges(R) =_{df} atoms_{Symm(\mathbf{L})}(R)$ are called its edges.*

In other words, edges are relations of the specific form $\{(v, w), (w, v)\}$ for distinct vertices v and w . Note that every non-empty element of $Symm(\mathbf{L})$ contains at least one edge.

Definition 5 (Bridges and Cycles) *If $R \in Symm(\mathbf{L})$ and e is an edge then R bridges e if $e \subseteq R^*$. A relation $R \in Symm(\mathbf{L})$ is said to be cycle-free if no $e \in edges(R)$ is bridged by $R \cap \bar{e}$.*

Now assume that $G \in Symm(\mathbf{L})$ is connected. Our task is to discover a program π that establishes the postcondition

$$T \text{ is connected and cycle-free} \wedge T \in Symm(G) \wedge T \text{ spans } G. \quad (1)$$

If (1) holds then T is called a *spanning tree* of G . It is, of course, understood that G is the input of π so that π may not assign to G . The obvious candidate for an invariant property is

$$T \text{ is connected and cycle-free } \wedge T \in \text{Symm}(G). \quad (2)$$

The following lemma shows that we can establish invariant (2) by assigning an arbitrary element of $\text{edges}(G)$ to T (using the generalized assignment of the refinement calculus [9]).

Lemma 6 *Edges are cycle-free and one-step-connected.*

We can now guess the following program outline:

```

 $T := \text{edges}(G);$ 
 $\{ T \text{ is connected and cycle-free } \wedge T \in \text{Symm}(G) \}$ 
while ... do
   $e := \text{edges}(G \cap \dots);$ 
   $T := T \cup e$ 
   $\{ T \text{ is connected and cycle-free } \wedge T \in \text{Symm}(G) \}$ 
od
 $\{ T \text{ is a spanning tree of } G \}$ 

```

An annotated program, such as the above, is defined to be correct if every possible run through it satisfies all of the assertions. We assume that the reader is familiar with the proof rules for establishing correctness and also with the most elementary correctness-preserving transformations [5, 6]. The following lemma suggests a choice for the edge to be added in each iteration of the loop. Its proof is given in Sec. 6 (Lemmata 16 and 19).

Lemma 7 *Let $T \in \text{Symm}(L)$. Then we have:*

1. *If T is connected and $x \in \text{edges}(TL \cup LT)$ then $T \cup x$ is connected.*
2. *If T is cycle-free and $x \in \text{edges}(\overline{TL} \cup \overline{LT})$ then $T \cup x$ is cycle-free.*

Thus, connectedness and cycle-freeness of T can be preserved simultaneously by adding an edge in $(TL \cup LT) \cap (\overline{TL} \cup \overline{LT})$. This is precisely the *symmetric difference* of TL and LT and we will denote it $TL \text{ xor } LT$. In order to pick an edge from $G \cap (TL \text{ xor } LT)$, we must check that this relation is irreflexive, symmetric and non-empty. The first two properties follow immediately from the corresponding properties of G and T . To ensure non-emptiness, we take it as the guard of the loop. Thus, we have proved correctness of the following program:

```

 $T := \text{edge}(G);$ 
while  $G \cap (TL \text{ xor } LT) \neq \mathbf{O}$  do
   $e := \text{edges}(G \cap (TL \text{ xor } LT));$ 
   $T := T \cup e$ 
od
 $\{ T \text{ is a spanning tree of } G \}$ 

```

Note that $TL \text{ xor } LT \subseteq \overline{TL} \cup \overline{LT} \subseteq \overline{T}$, so that T increases strictly. Since T is bounded by G , termination is ensured (provided, of course, that G is finite). The following lemma shows that invariant (2) and the exit condition of the previous program imply the required postcondition. For a proof see Lemma 20 in Sec. 6.

Lemma 8 *Let $S, T \in \text{Symm}(\mathbf{L})$ such that S is connected and $\mathbf{O} \neq T \subseteq S$. Then $(TL \text{ xor } LT) \subseteq \overline{S}$ if and only if $LS = LT$ (i.e. T spans S).*

Besides establishing the postcondition, this lemma permits us to simplify the exit condition. Thus, we have arrived at program

$$\begin{array}{l}
T := \text{edges}(G); \\
\mathbf{while } \mathbf{L}G \neq \mathbf{L}T \mathbf{ do} \\
\quad e := \text{edges}(G \cap (TL \text{ xor } LT)); \\
\quad T := T \cup e; \\
\quad \mathbf{od} \\
\{ T \text{ is a spanning tree of } G \}
\end{array} \tag{3}$$

for computing a spanning tree T of an adjacence relation G .

3 Changing the Representation

We ultimately aim at a minimum spanning tree algorithm for weighted graphs. Since a weight is a mapping on edges, it will be convenient to represent a graph as a set of edges, rather than a relation on vertices. The new representation may be isomorphic to the old one, but it does mean a change of data structure and consequently a substantial program modification. Moreover, there are many different ways of storing a set in a computer. To retain flexibility and avoid bias towards a specific implementation we will not commit ourselves to a fixed representation yet. Instead we will work in an arbitrary complete and atomistic lattice \mathcal{E} that is linked to $\text{Symm}(G)$ by two mappings

$$\Delta : \text{Symm}(G) \rightarrow \mathcal{E} \quad \Gamma : \mathcal{E} \rightarrow \text{Symm}(G).$$

We require that Δ is an *embedding* (an injective mapping that distributes over arbitrary joins and meets) and that Γ is its *lower adjoint* in the sense that $\Delta(\Gamma(p)) \supseteq p$ and $\Gamma(\Delta(R)) = R$ for every $p \in \mathcal{E}$ and $R \in \text{Symm}(G)$. The following lemma lists two simple consequences of these postulates for later use.

Lemma 9 *1. Suppose that e is an edge. Then $\Delta(e) \neq \mathbf{O}$ and $e = \Gamma(x)$ for every $x \in \text{atoms}_{\mathcal{E}}(\Delta(e))$.*
2. Suppose q is an atom of $r \in \mathcal{E}$. Then there is an edge e of $\Gamma(r)$ such that $q \in \text{atoms}_{\mathcal{E}}(\Delta(e))$.

Let $L_{\mathcal{E}}$ denote the greatest element of \mathcal{E} . Then $\Delta(G) = L_{\mathcal{E}}$ and $\Gamma(L_{\mathcal{E}}) = G$. We will say that $p \in \mathcal{E}$ is a spanning tree of \mathcal{E} if $\Gamma(p)$ is a spanning tree of $\Gamma(L_{\mathcal{E}})$ in the sense defined by postcondition (1). Assume that $\Gamma(L_{\mathcal{E}})$ is connected. Then we know from the previous section that the following program is correct:

```

G :=  $\Gamma(L_{\mathcal{E}})$ ;
T :=  $\text{edges}(G)$ ;
p :=  $\mathcal{E}$ ;
while  $L_G \neq L_T$  do
  e :=  $\text{edges}(G \cap (TL \text{ xor } LT))$ ;
  T :=  $T \cup e$ 
od
{  $\Gamma(p) = T \Rightarrow p$  is a spanning tree of  $\mathcal{E}$  }

```

We can eliminate the first assignment by replacing G with $\Gamma(L_{\mathcal{E}})$ throughout the rest of the program. To eliminate the antecedent from the postcondition we introduce the additional invariant $\Gamma(p) = T$, which we establish by refining the random assignment $p := \mathcal{E}$ and maintain by inserting an appropriate assignment to p . This is safe because p is never used and we obtain the following program:

```

T :=  $\text{edges}(\Gamma(L_{\mathcal{E}}))$ ;
p :=  $\text{atoms}_{\mathcal{E}}(\Delta(T))$ ;
{  $\Gamma(p) = T$  }
while  $L_{\Gamma(L_{\mathcal{E}})} \neq L_T$  do
  e :=  $\text{edges}(\Gamma(L_{\mathcal{E}}) \cap (TL \text{ xor } LT))$ ;
  q :=  $\text{atoms}_{\mathcal{E}}(\Delta(e))$ ;
  T, p :=  $T \cup e, p \cup q$ 
  {  $\Gamma(p) = T$  }
od
{ p is a spanning tree of  $\mathcal{E}$  }

```

The correctness of the first assertion follows from Lemma 9.1. To see that the parallel assignment to T and p preserves the new invariant we note that the assignment to q establishes $e = \Gamma(q)$ (again by Lemma 9.1), whence we have $\Gamma(p \cup q) = \Gamma(p) \cup \Gamma(q) = T \cup e$. Now we can use the invariant to eliminate T from all expressions. After that, T becomes garbage, and all assignments to T may be dropped. This results in the following program:

```

e :=  $\text{edges}(\Gamma(L_{\mathcal{E}}))$ ;
p :=  $\text{atoms}_{\mathcal{E}}(\Delta(e))$ ;
while  $L_{\Gamma(L_{\mathcal{E}})} \neq L_{\Gamma(p)}$  do
  e :=  $\text{edges}(\Gamma(L_{\mathcal{E}}) \cap (\Gamma(p)L \text{ xor } L_{\Gamma(p)}))$ ;
  q :=  $\text{atoms}_{\mathcal{E}}(\Delta(e))$ ;
  p :=  $p \cup q$ 
od
{ p is a spanning tree of  $\mathcal{E}$  }

```

By Lemma 9.2 a program piece of the form $e := \text{edges}(\Gamma(r)); q := \text{atoms}_{\mathcal{E}}(\Delta(e))$ can be refined to $e := \text{edges}(\Gamma(r)); q := \text{atoms}_{\mathcal{E}}(r)$. This transformation can be

applied directly to the initialization. In order to exploit it also for the body of the loop we assume that we are given a mapping $f : \mathcal{E} \rightarrow \mathcal{E}$ with

$$\Gamma(\mathbf{L}_{\mathcal{E}}) \cap (\Gamma(p)\mathbf{L} \text{ xor } \mathbf{L}\Gamma(p)) = \Gamma(f(p)). \quad (4)$$

After applying the refinement, both assignments to e can be scratched and the following program for computing a spanning tree p of \mathcal{E} remains:

$$\begin{aligned} & p := \text{atoms}_{\mathcal{E}}(\mathbf{L}_{\mathcal{E}}); \\ & \mathbf{while} \mathbf{L}\Gamma(\mathbf{L}_{\mathcal{E}}) \neq \mathbf{L}\Gamma(p) \mathbf{do} \\ & \quad t := \text{atoms}_{\mathcal{E}}(f(p)); \\ & \quad p := p \cup t \\ & \quad \mathbf{od} \\ & \{ p \text{ is a spanning tree of } \mathcal{E} \} \end{aligned} \quad (5)$$

4 Incidence Structures and Multigraphs

We will now instantiate program (5) derived in the previous section by choosing a particular embedding Δ , and exhibiting a mapping f that satisfies (4). We thus obtain our third program, this time for a graph that is given as an incidence structure. In the next section, a further refinement will yield a minimum spanning tree algorithm for weighted graphs.

- Definition 10**
1. An incidence structure (V, E, M) consists of a set V (the vertices), a set E (the edges), and an incidence relation $M \subseteq E \times V$.
 2. The incidence structure (V, E, M) is called a multigraph if we can write $M = f \cup g$ where f and g are disjoint functions in the relational sense¹ (i.e. every edge coincides with exactly two vertices).
 3. The multigraph (V, E, M) is a (proper) graph if, fg^{\cup} is antisymmetric (i.e. no pair of vertices is connected by multiple edges).

With every incidence structure (V, E, M) we can associate an *adjacence* relation G via $G =_{af} M M^{\cup} \cap \bar{\mathbf{I}}$ which is an irreflexive and symmetric relation on vertices. Now let $\mathcal{E} =_{af} \{p \subseteq E \times V \mid p\mathbf{L} = p\}$. Each $p \in \mathcal{E}$ represents a set of edges, which can be obtained by projecting p to its first component. The required link between \mathcal{E} and $\text{Sym}(G)$ is given by following mappings:

$$\begin{aligned} \Gamma(p) &= \{(x, y) \mid x \neq y \wedge \exists e \in E : (e, x), (e, y) \in M \cap p\} \\ &= (M \cap p)^{\cup} (M \cap p) \cap \bar{\mathbf{I}} \\ \Delta(R) &= \{e \mid \exists (x, y) \in R : (e, x), (e, y) \in M\} \times V \\ &= (\mathbf{I} \cap M R M^{\cup})\mathbf{L}_{\mathcal{E}}. \end{aligned}$$

The following lemma states that Γ and Δ do indeed enjoy the properties required for the derivation in the previous section. For its proof see Lemmata 26 through 29 in Sec. 6.

¹ We distinguish “meta level” mappings from the relation-algebraic (or: object level) notion of a function. A relation f is a *function* if $f^{\cup} f \subseteq \mathbf{I}$ and $\mathbf{I} \subseteq f f^{\cup}$.

Lemma 11 1. If (V, E, M) is a multigraph then Δ is an embedding and Γ is its lower adjoint.

2. If (V, E, M) is a proper graph then Δ is an isomorphism and Γ is its inverse.

Let us start with the case where (V, E, M) is a proper graph. In this case, a spanning tree of (V, E, M) is defined to be an element $p \in \mathcal{E}$ such that (1) is satisfied with $T = \Gamma(p)$ and $G = \Gamma(\mathbf{L}\mathcal{E})$. The next lemma yields a mapping $f : \mathcal{E} \rightarrow \mathcal{E}$ that satisfies the condition (4). For a proof see Lemma 30 in Sec. 6.

Lemma 12 $\Gamma(\mathbf{L}\mathcal{E}) \cap (\Gamma(p)\mathbf{L} \text{ xor } \mathbf{L}\Gamma(p)) = \Gamma(MM^{\cup}p \cap \overline{MM^{\cup}p})$.

The mapping Γ still occurs in the exit condition of (5). The following lemma allows us to eliminate Γ completely from the entire program. For the proof see Lemma 22 in Sec. 6.

Lemma 13 $\Gamma(p)\mathbf{L} = M^{\cup}p$ and $\mathbf{L}\Gamma(p) = p^{\cup}M$.

Thus, we have derived the following program for computing a spanning tree of a proper graph (V, E, M) :

$$\begin{aligned}
& p := \text{atoms}_{\mathcal{E}}(\mathbf{L}\mathcal{E}); \\
& \text{while } \mathbf{L}\mathcal{E}^{\cup}M \neq p^{\cup}M \text{ do} \\
& \quad t := \text{atoms}_{\mathcal{E}}(MM^{\cup}p \cap \overline{MM^{\cup}p}); \\
& \quad p := p \cup t \\
& \quad \text{od;} \\
& \{ p \text{ is a spanning tree of } (V, E, M) \}
\end{aligned} \tag{6}$$

This program does, in fact, also work for multigraphs. The only addition to the proof we need to make is to adapt the definition of a spanning tree: We must add the constraint that p does not contain any double edges in the sense that $\Gamma(x) = \Gamma(y)$ implies $x = y$ for all $x, y \in \text{atoms}_{\mathcal{E}}(p)$. We leave it to the reader to check that this is an invariant of the above program.

5 Weighted Graphs

Let (V, E, M) be a (proper) graph and assume that a weight $w(e) \in \mathbf{R}_{\geq 0}$ is assigned to every edge $e \in \text{atoms}_{\mathcal{E}}(\mathbf{L}\mathcal{E})$. We are asked to compute a minimum spanning tree (i.e. one for which the sum of the weights of its edges is minimal). To reuse program (6) we have to strengthen its invariant. At the very least, we must require that after each cycle the tree computed so far is *contained* in a minimum spanning tree.

There are two points in program (6) where an edge is picked non-deterministically. It seems reasonable always to select the lightest edge available. Therefore, assume that we have at our disposal a mapping $\text{least} : \mathcal{E} \rightarrow \mathcal{E}$ that satisfies, for all elements $p \in \mathcal{E}$, the following implication:

$$q \in \text{atoms}_{\mathcal{E}}(\text{least}(p)) \wedge r \in \text{atoms}_{\mathcal{E}}(p) \Rightarrow w(q) \leq w(r). \tag{7}$$

Then the following lemma shows that the choice of the lightest edge will indeed establish and then preserve the desired invariant. This is a non-trivial fact and a proof in terms of points and paths can be found in [7], Theorems 4.3.1 through 4.3.3. We will give a point-free proof; see Lemmata 32 and 33 of Sec. 6.

Lemma 14 1. *If $t \in \text{atoms}_{\mathcal{E}}(\text{least}(L_{\mathcal{E}}))$ then t is contained in a minimum spanning tree.*

2. *If p is contained in a minimum spanning tree then $p \cup t$ is contained in a minimum spanning tree for all $t \in \text{atoms}_{\mathcal{E}}(\text{least}(MM^{\cup}p \cap M\overline{M^{\cup}p}))$.*

Thus, the following program is correct:

```

p := atomsℰ(least(Lℰ));
while Lℰ∪M ≠ p∪M do
  t := atomsℰ(least(MM∪p ∩ M∪p̄));
  p := p ∪ t
od;
{ p is a minimum spanning tree of (V, E, M) wrt. to w }

```

(8)

We still need to replace the implicit definition (7) of *least* with a closed relational expression. Let W be the pre-order that w induces on E . If ρ is the order on real numbers and w is conceived as function in the relational sense then W can be defined by the formula $W =_{df} w\rho w^{\cup}$. The relational expression describing the set of all minimal elements of a set p wrt a preorder W is well-known (see [11]):

Lemma 15 *If we define $\text{least}(p) =_{df} p \cap \overline{Wp}$ then (7) holds for every $p \in \mathcal{E}$.*

This completes the development of a minimum spanning tree algorithm. In the next section we will use Tarski's relational calculus for rigorously proving the lemmata used in its derivation. Before we do that, let us make some remarks on complexity. Assume m and n to be the number of edges resp. vertices of (V, E, M) . If we use the standard Boolean matrix implementation of relations and their operations then program (8) runs in time $O(m^2 * n)$. However, if we implement relations by successor lists and compute the set of edges described by $MM^{\cup}p \cap M\overline{M^{\cup}p}$ (i.e., the “neighbour edges” of the tree constructed so far) incrementally, then a refinement with quadratic run time complexity can be obtained. This program is a variant of Prim's spanning tree algorithm [10] which relies on a similar optimization.

6 Relation-Algebraic Proofs

We believe that programs should be proved even more rigorously than most other mathematical theorems. For one thing, programming errors can be costly. Also program correctness proofs do not tend to be read by many people, so that the social process of eliminating errors works less well than in general mathematics. Ideally, program proofs are conducted in a formal calculus where each step can be checked mechanically.

The calculus of annotated programs is one such framework. We have used it quite informally here, because this is a scientific paper and not a program documentation. In real life, the development should be performed within a proof assistant which relieves the programmer from the burden of copying the entire program at each transformation step while generating the proof obligations that must be discharged for its justification. These proof obligations are the various lemmata scattered throughout the preceding sections. Their nature varies greatly with the problem domain and it is not likely that a general-purpose theory will always be adequate for their treatment.

The data in our programs are relations and graphs. These structures are the objects of another formal calculus, known as the algebra of relations [11, 4]. Unlike the calculus of annotated programs, relational algebra is well-suited to rigorous pen-and-paper reasoning². Therefore, it can (and should) be used for giving precise proofs of graph-theoretical statements, even in a scientific paper.

6.1 Relational Algebra

We have already described the operators and constants of relational algebra in Sec. 2. In the following, we collect some basic facts on relations. Their proofs are straight-forward set-theoretic arguments. Alternatively, they may be derived from Tarski's axiomatization of relations, but then some of them require, in addition, the so-called Point Axiom. This can't be avoided, because our programs rely on being able to pick an element from any non-empty set.

- Composition is associative, monotonic, has identity $\mathbb{1}$ and is \cup -distributive.
- Transposition distributes over all Boolean operators. Moreover, we have $R^{\cup\cup} = R$ and $(RS)^{\cup} = S^{\cup}R^{\cup}$.
- Schröder equivalences: $PQ \subseteq \overline{R} \Leftrightarrow P^{\cup}R \subseteq \overline{Q} \Leftrightarrow RQ^{\cup} \subseteq \overline{P}$.
- Dedekind laws: $P \cap QR \subseteq Q(Q^{\cup}P \cap R)$ and $P \cap QR \subseteq (PR^{\cup} \cap Q)R$.
- Tarski's rule: If $R \neq \mathbf{0}$ then $\mathbb{1}R\mathbb{1} = \mathbb{1}$.
- Distributivity of functions: If f is a function then $f(R \cap S) = fR \cap fS$ and $(R \cap S)f^{\cup} = Rf^{\cup} \cap Sf^{\cup}$.
- Dedekind laws for functions: If f is a function then $f^{\cup}(R \cap fS) = f^{\cup}R \cap S$ and $(R \cap Sf^{\cup})f = Rf \cap S$.
- Identity transposition rule: $\mathbb{1} \cap R = \mathbb{1} \cap R^{\cup}$.
- Tail recursion rule: If $T \subseteq X$ and $XS \subseteq X$ then $TS^* \subseteq X$.
- If the relation $R \in \text{Symm}(L)$ is cycle-free then $A^* \cap B^* = (A \cap B)^*$ for all $A, B \in \text{Symm}(R)$.
- Star decomposition rule: If R is one-step-connected then $(S \cup R)^* = S^* \cup S^*RS^*$.
- Singling out rows: If $a\mathbb{1} = a$ then $(a \cap R)S = a \cap RS$.
- Rectangle rule: $a \cap b^{\cup} = ab^{\cup}$ provided $a\mathbb{1} = a$ and $b\mathbb{1} = b$.

² This does not mean that relation-algebraic proofs cannot or should not be developed or checked with computer assistance. In fact, there exists an excellent tool for doing just that [3].

- Vector³ identity rules: If $a\mathbf{L} = a$ then $(R \cap a^\cup) = R(\mathbf{L} \cap a)$. Conversely, if $b \subseteq \mathbf{L}$ then $Rb = R \cap (b\mathbf{L})^\cup$.
- Vector negation rule: If $a\mathbf{L} = a$ then also $\bar{a}\mathbf{L} = \bar{a}$.
- Vector associativity rule: If $a\mathbf{L} = a$ then $(R \cap a^\cup)S = R(S \cap a)$.

6.2 Adjacence Relations

The purpose of this section is to present the proofs of Sec. 2.

Lemma 16 *If $T \in \text{Symm}(\mathbf{L})$ is connected and $x \in \text{edges}(T\mathbf{L} \cup \mathbf{L}T)$ then $T \cup x$ is connected.*

Proof. $T \cup x$ is connected

$$\begin{aligned}
&\Leftrightarrow \{\text{Definition 2}\} \\
&(T \cup x)\mathbf{L}(T \cup x) \subseteq (T \cup x)^* \\
&\Leftrightarrow \{\text{Distributivity, connectedness of } T \text{ and } x\} \\
&T\mathbf{L}x \cup x\mathbf{L}T \subseteq (T \cup x)^* \\
&\Leftrightarrow \{x, T, \text{ and } \mathbf{L} \text{ are symmetric}\} \\
&T\mathbf{L}x \subseteq (T \cup x)^* \\
&\Leftrightarrow \{x = (x \cap T\mathbf{L}) \cup (x \cap \mathbf{L}T), \text{ distributivity}\} \\
&T\mathbf{L}(x \cap \mathbf{L}T) \cup T\mathbf{L}(x \cap T\mathbf{L}) \subseteq (T \cup x)^* \\
&\Leftrightarrow \{T\mathbf{L}(x \cap \mathbf{L}T) \subseteq T\mathbf{L}T \subseteq T^* \text{ since } T \text{ is connected}\} \\
&T\mathbf{L}(x \cap T\mathbf{L}) \subseteq (T \cup x)^* \\
&\Leftarrow \{\text{Vector associativity rule, } (T\mathbf{L})^\cup = \mathbf{L}T\} \\
&T\mathbf{L}T\mathbf{L}x \subseteq (T \cup x)^* \\
&\Leftrightarrow \{T \text{ is connected}\} \\
&\text{true.} \quad \blacksquare
\end{aligned}$$

Lemma 17 (Bridge Exchange) *Assume that $S \in \text{Symm}(\mathbf{L})$ bridges neither of the edges x and y . Then $S \cup x$ bridges y if and only if $S \cup y$ bridges x .*

Proof. $S \cup x$ bridges y

$$\begin{aligned}
&\Leftrightarrow \{\text{Definition 5}\} \\
&y \subseteq (S \cup x)^* \\
&\Leftrightarrow \{y \text{ is an edge and } (S \cup x)^* \text{ is symmetric}\} \\
&y \cap (S \cup x)^* \neq \mathbf{O} \\
&\Leftrightarrow \{\text{Star decomposition, edges are one-step-connected}\} \\
&y \cap (S^* \cup S^*xS^*) \neq \mathbf{O} \\
&\Leftrightarrow \{S \text{ does not bridge } y, \text{ so } y \cap S^* = \mathbf{O}\} \\
&y \cap S^*xS^* \neq \mathbf{O}
\end{aligned}$$

³ A relation R is called a *vector* if $R\mathbf{L} = R$.

$$\begin{aligned}
&\Leftrightarrow \{ \text{Schröder equivalences (twice)} \} \\
&S^{*\cup}yS^{*\cup} \cap x \neq \mathbf{O} \\
&\Leftrightarrow \{ S \text{ is symmetric} \} \\
&S^*yS^* \cap x \neq \mathbf{O} \\
&\Leftrightarrow \{ \text{Reverse first four steps} \} \\
&S \cup y \text{ bridges } x. \quad \blacksquare
\end{aligned}$$

Lemma 18 Assume that $T \in \text{Symm}(\mathbf{L})$ is cycle-free and that x is an edge which is not bridged by T . Then $T \cup x$ is cycle-free.

Proof. Let $y \in \text{edges}(T \cup x)$; we have to show that $(T \cup x) \cap \bar{y}$ does not bridge y . We may assume that $y \neq x$. Since x and y are edges, it follows that $x \cap y = \mathbf{O}$ whence $y \subseteq T$. Let $S =_{df} T \cap \bar{y}$. Since x is not bridged by T and $T = S \cup y$ the bridge exchange lemma implies that y is not bridged by $S \cup x$. \blacksquare

Lemma 19 Assume that $T \in \text{Symm}(\mathbf{L})$ is cycle-free and $x \in \text{edges}(\overline{TL} \cup \overline{LT})$. Then $T \cup x$ is cycle-free.

Proof. We note that T does not bridge x for otherwise we get $x \subseteq \bar{1} \cap T^* \subseteq TT^* \subseteq TL \cap LT$, contrary to assumption. Now Lemma 18 applies. \blacksquare

Lemma 20 Let $S, T \in \text{Symm}(\mathbf{L})$ such that S is connected and $\mathbf{O} \neq T \subseteq S$. Then $(\overline{TL} \text{ xor } \overline{LT}) \subseteq \bar{S}$ if and only if $LS = LT$.

Proof. The implication from right to left is very easy, and we leave it to the reader. For the other implication, assume that $(\overline{TL} \text{ xor } \overline{LT}) \subseteq \bar{S}$. We start with

$$\begin{aligned}
<S \\
&\subseteq \{ TS = TS \cap L, \text{ Dedekind law} \} \\
<(S \cap T^{\cup}L) \\
&\subseteq \{ \text{Symmetry of } T \text{ and } S \subseteq \overline{TL \text{ xor } LT} \subseteq \overline{TL} \cup \overline{LT} \} \\
<LT \\
&\subseteq \\
<.
\end{aligned}$$

as auxiliary result. Since $T \subseteq S$ the following calculation completes the proof:

$$\begin{aligned}
&LS \\
&= \{ \text{Tarski's Rule, } TT \neq \mathbf{O} \text{ as } T \neq \mathbf{O} \text{ symmetric} \} \\
<TLS \\
&\subseteq \{ T \subseteq S \} \\
<SLS \\
&\subseteq \{ S \text{ is connected} \} \\
<S^* \\
&\subseteq \{ TS^* \subseteq LT \text{ by auxiliary result and tail recursion} \} \\
<. \quad \blacksquare
\end{aligned}$$

6.3 Incidence Structures and Multigraphs

In the following, we assume that (V, E, M) is a multigraph. Then $M = f \cup g$ where f and g are disjoint functions from E to V . We let the variables p and q range over $\mathcal{E} = \{r \subseteq E \times V \mid r\mathbf{L} = r\}$. They can be thought of as denoting sets of edges.

Lemma 21 $\Gamma(p) = (f \cap p)^\cup g \cup (g \cap p)^\cup f = f^\cup (f \cap p)g \cup g^\cup (f \cap p)f$.

$$\begin{aligned}
\text{Proof. } \Gamma(p) &= \\
&= \quad \{\text{Definition of } \Gamma, \text{ vector associativity}\} \\
&\quad \bar{1} \cap (M \cap p)^\cup M \\
&= \quad \{M = f \cup g, \text{ distributivity}\} \\
&\quad \bar{1} \cap ((f \cap p)^\cup f \cup (f \cap p)^\cup g \cup (g \cap p)^\cup f \cup (g \cap p)^\cup g) \\
&= \quad \{f^\cup f \subseteq \bar{1} \text{ and } g^\cup g \subseteq \bar{1} \text{ since } f \text{ and } g \text{ are a functions}\} \\
&\quad \bar{1} \cap ((f \cap p)^\cup g \cup (g \cap p)^\cup f) \\
&= \quad \{f^\cup g \subseteq \bar{1} \text{ and } g^\cup f \subseteq \bar{1} \text{ since } f, g \text{ are disjoint (Schröder)}\} \\
&\quad (f \cap p)^\cup g \cup (g \cap p)^\cup f \\
&= \quad \{\text{Vector identity rule}\} \\
&\quad f^\cup (f \cap p)g \cup g^\cup (f \cap p)f. \quad \blacksquare
\end{aligned}$$

Lemma 22 $M^\cup p = \Gamma(p)\mathbf{L}$ and $p^\cup M = \mathbf{L}\Gamma(p)$.

Proof. Since $\Gamma(p)$ is symmetric, these equations are equivalent and we only show the first one.

$$\begin{aligned}
M^\cup p &= \\
&= \quad \{M = f \cup g, \text{ distributivity}\} \\
&\quad f^\cup p \cup g^\cup p \\
&= \quad \{\text{Vector associativity}\} \\
&\quad (f \cap p)^\cup \mathbf{L} \cup (g \cap p)^\cup \mathbf{L} \\
&= \quad \{f\mathbf{L} = \mathbf{L} \text{ and } g\mathbf{L} = \mathbf{L} \text{ since } f \text{ and } g \text{ are total}\} \\
&\quad (f \cap p)^\cup g\mathbf{L} \cup (g \cap p)^\cup f\mathbf{L} \\
&= \quad \{\text{Distributivity, Lemma 21}\} \\
&\quad \Gamma(p)\mathbf{L}. \quad \blacksquare
\end{aligned}$$

Lemma 23 Let $a \subseteq V \times V$ with $a\mathbf{L} = a$. Then $\Gamma(Ma) = (a \cup a^\cup) \cap \Gamma(\mathbf{L}\mathcal{E})$.

$$\begin{aligned}
\text{Proof. } \Gamma(Ma) &= \\
&= \quad \{\text{Lemma 21}\} \\
&\quad (f \cap Ma)^\cup g \cup (g \cap Ma)^\cup f \\
&= \quad \{M = f \cup g, \text{ distributivity}\} \\
&\quad (f \cap fa)^\cup g \cup (g \cap fa)^\cup f \cup (f \cap ga)^\cup g \cup (g \cap ga)^\cup f
\end{aligned}$$

$$\begin{aligned}
&= \{ \text{Distributivity of functions} \} \\
&\quad (f(l \cap a))^{\cup} g \cup (g \cap f a)^{\cup} f \cup (f \cap g a)^{\cup} g \cup (g(l \cap a))^{\cup} f \\
&= \{ \text{Distributivity of transposition} \} \\
&\quad (l \cap a) f^{\cup} g \cup (g^{\cup} \cap a^{\cup} f^{\cup}) f \cup (f^{\cup} \cap a^{\cup} g^{\cup}) g \cup (l \cap a) g^{\cup} f \\
&= \{ \text{Singling out rows, Dedekind law of functions} \} \\
&\quad (a \cap f^{\cup} g) \cup (g^{\cup} f \cap a^{\cup}) \cup (f^{\cup} g \cap a^{\cup}) \cup (a \cap g^{\cup} f) \\
&= \{ \text{Boolean algebra} \} \\
&\quad (a \cup a^{\cup}) \cap (f^{\cup} g \cup g^{\cup} f) \\
&= \{ \text{Lemma 21} \} \\
&\quad (a \cup a^{\cup}) \cap \Gamma^{\cup}(\mathbb{L}_{\mathcal{E}}). \quad \blacksquare
\end{aligned}$$

Lemma 24 *Let $a, b \subseteq V \times V$ with $a\mathbb{L} = a$ and $b\mathbb{L} = b$. Then we have the distributivity property $\Gamma(Ma) \cap \Gamma(Mb) = \Gamma(Ma \cap Mb)$.*

Proof. The inclusion from right to left follows from the fact that Γ is monotonic. By Lemma 23 the other inclusion equivaless

$$\Gamma(\mathbb{L}_{\mathcal{E}}) \cap (a \cup a^{\cup}) \cap (b \cup b^{\cup}) \subseteq \Gamma(Ma \cap Mb).$$

Since every relation in the image of Γ is symmetric it is sufficient to prove

$$\Gamma(\mathbb{L}_{\mathcal{E}}) \cap a \cap b \subseteq \Gamma(Ma \cap Mb)$$

and

$$\Gamma(\mathbb{L}_{\mathcal{E}}) \cap a \cap b^{\cup} \subseteq \Gamma(Ma \cap Mb).$$

The first of these two inclusions is immediate from Lemma 23 (with $a \cap b$ in place of a) and monotonicity; the following calculation yields the second one:

$$\begin{aligned}
&a \cap b^{\cup} \cap \Gamma(\mathbb{L}_{\mathcal{E}}) \\
&= \{ \text{Rectangle rule, definition of } \Gamma \} \\
&\quad ab^{\cup} \cap M^{\cup} M \cap \bar{1} \\
&\subseteq \{ \text{Dedekind laws (twice)} \} \\
&\quad M^{\cup} (Mab^{\cup} M^{\cup} \cap l) M \cap \bar{1} \\
&= \{ \text{Rectangle rule} \} \\
&\quad M^{\cup} (Ma \cap (Mb)^{\cup} \cap l) M \cap \bar{1} \\
&= \{ \text{Identity transposition rule} \} \\
&\quad M^{\cup} (Ma \cap Mb \cap l) M \cap \bar{1} \\
&= \{ \text{Vector identity rule} \} \\
&\quad (M \cap Ma \cap Mb)^{\cup} M \cap \bar{1} \\
&= \{ \text{Definition of } \Gamma \} \\
&\quad \Gamma(Ma \cap Mb). \quad \blacksquare
\end{aligned}$$

Lemma 25 *If $R \in \text{Symm}(G)$ then $\Delta(R) = (I \cap fRg^\cup)\mathcal{L}_\mathcal{E}$.*

Proof. By definition, we have $\Delta(R) = (I \cap MRM^\cup)\mathcal{L}_\mathcal{E}$. Next we substitute $f \cup g$ for the relation M and multiply out. Then two of the four resulting terms vanish, e.g. $I \cap fRf^\cup \subseteq f(f^\cup f \cap R)f^\cup \subseteq f(I \cap R)f^\cup = \mathbf{O}$. The remaining two terms are equal, because of the identity transposition rule. ■

Lemma 26 *The mapping $\Delta : \text{Symm}(G) \rightarrow \mathcal{E}$ distributes over all meets and joins.*

Proof. By the previous lemma we have

$$\Delta = (x \mapsto (I \cap x)\mathcal{L}_\mathcal{E}) \circ (x \mapsto fx) \circ (x \mapsto xg^\cup)$$

and each of the three mappings on the right hand side distributes over all meets and joins (recall that f and g are functions). ■

Lemma 27 *$\Delta(\Gamma(p)) \supseteq p$ for every $p \in \mathcal{E}$.*

Proof.

$$\begin{aligned} \Delta(\Gamma(p)) &= \{\text{Lemma 25}\} \\ & (I \cap f\Gamma(p)g^\cup)\mathcal{L}_\mathcal{E} \\ &\supseteq \{\text{Lemma 21}\} \\ & (I \cap ff^\cup(I \cap p)gg^\cup)\mathcal{L}_\mathcal{E} \\ &\supseteq \{ff^\cup \supseteq I \text{ and } gg^\cup \supseteq I \text{ because } f \text{ and } g \text{ are functions}\} \\ & (I \cap p)\mathcal{L}_\mathcal{E} \\ &= \{\text{Vector identity rule}\} \\ & p. \end{aligned}$$

■

Lemma 28 *$\Gamma(\Delta(R)) = R$ for every $R \in \text{Symm}(\Gamma(\mathcal{L}_\mathcal{E}))$.*

Proof.

$$\begin{aligned} \Gamma(\Delta(R)) &= \{\text{Lemma 21}\} \\ & f^\cup(I \cap \Delta(R))g \cup g^\cup(I \cap \Delta(R))f \\ &= \{\text{Lemma 25}\} \\ & f^\cup(I \cap fRg^\cup)g \cup g^\cup(I \cap gRf^\cup)f \\ &= \{\text{Dedekind law for functions}\} \\ & f^\cup(g \cap fR) \cup g^\cup(f \cap gR) \\ &= \{\text{Dedekind law for functions}\} \\ & (f^\cup g \cap R) \cup (g^\cup f \cap R) \\ &= \{\text{Lemma 21, distributivity}\} \\ & R \cap \Gamma(\mathcal{L}_\mathcal{E}) \\ &= \{R \in \text{Symm}(\Gamma(\mathcal{L}_\mathcal{E}))\} \\ & R. \end{aligned}$$

■

Lemma 29 *If (V, E, M) is a graph then $\Delta(\Gamma(p)) \subseteq p$.*

Proof. Before we start with the main calculation we show that $R(p \cup \perp)R \cap \perp \subseteq p$ for every antisymmetric relation R :

$$\begin{aligned}
& R(p \cup \perp)R \cap \perp \\
\subseteq & \quad \{\text{Use Dedekind to factor out } (p \cup \perp)R\} \\
& (R \cap ((p \cap \perp)R)^\cup) (p \cup \perp) R \\
\subseteq & \quad \{\text{Monotonicity}\} \\
& (R \cap R^\cup)p\mathbf{L} \\
\subseteq & \quad \{R \text{ is antisymmetric, } p\mathbf{L} = p\} \\
& p.
\end{aligned}$$

In the following calculation the above result is used with $R = fg^\cup$ (which is antisymmetric by Def. 10).

$$\begin{aligned}
& \Delta(\Gamma(p)) \\
= & \quad \{\text{Lemma 25}\} \\
& (\perp \cap f\Gamma(p)g^\cup)\mathbf{L}_\mathcal{E} \\
= & \quad \{\text{Lemma 21, distributivity}\} \\
& (\perp \cap (ff^\cup(p \cap \perp)gg^\cup)\mathbf{L}_\mathcal{E} \cup (\perp \cap (fg^\cup(p \cap \perp)fg^\cup))\mathbf{L}_\mathcal{E} \\
\subseteq & \quad \{ff^\cup \subseteq \perp \text{ and } gg^\cup \subseteq \perp \text{ and auxiliary result}\} \\
& (\perp \cap p)\mathbf{L}_\mathcal{E} \\
\subseteq & \quad \{\text{Vector identity law}\} \\
& p. \quad \blacksquare
\end{aligned}$$

Lemma 30 *Let $a =_{df} \Gamma(p)\mathbf{L}$. Then $\Gamma(\mathbf{L}_\mathcal{E}) \cap (a \text{ xor } a^\cup) = \Gamma(MM^\cup p \cap M\overline{M^\cup p})$.*

$$\begin{aligned}
\textit{Proof.} \quad & \Gamma(\mathbf{L}_\mathcal{E}) \cap (a \text{ xor } a^\cup) \\
= & \quad \{\text{Definition of } \text{ xor}\} \\
& \Gamma(\mathbf{L}_\mathcal{E}) \cap (a \cup a^\cup) \cap (\overline{a} \cup \overline{a^\cup}) \\
= & \quad \{\text{Lemma 23 (note that } \overline{a\mathbf{L}} = \overline{a} \text{ by vector negation)}\} \\
& \Gamma(Ma) \cap \Gamma(M\overline{a}) \\
= & \quad \{\text{Lemma 24}\} \\
& \Gamma(Ma \cap M\overline{a}) \\
= & \quad \{\text{Lemma 22}\} \\
& \Gamma(MM^\cup p \cap M\overline{M^\cup p}). \quad \blacksquare
\end{aligned}$$

6.4 Weighted Graphs

Assume that $\mathbf{L} = V \times V$ is finite and that $T \in \text{Symm}(\mathbf{L})$ is cycle-free. Then the transitive-reflexive closure operator is a universally conjunctive mapping from $\text{Symm}(T)$ to the lattice of all subrelations of T^* . It follows that we can define a

lower adjoint, i.e. a mapping that maps each $S \subseteq T^*$ to some $S^{[T]} \in \text{Symm}(T)$ such that the following Galois connection holds:

$$S \subseteq R^* \quad \Leftrightarrow \quad S^{[T]} \subseteq R \quad \text{for all } S \subseteq T^* \text{ and } R \in \text{Symm}(T). \quad (9)$$

An element x of a basis of a vector space may be replaced with a different vector y provided x is needed for expressing y as a linear combination of basis vectors. Similarly, an element x of a spanning tree may be replaced with y if y is needed for bridging p :

Lemma 31 (Edge Replacement) *Suppose that Q is a spanning tree of G and let $x \in \text{edges}(G)$. If $y \in \text{edges}(x^{[Q]})$ then $(Q \cap \overline{y}) \cup x$ is a spanning tree of G .*

Proof. For convenience, let $S = Q \cap \overline{y}$. We have assumed that $y \subseteq x^{[Q]}$, so $x^{[Q]} \not\subseteq Q \cap \overline{y}$ and the Galois connection (9) yields $x \not\subseteq S^*$. In other words, S does not bridge x , and Lemma 18 implies that $S \cup x$ is cycle-free. Moreover, the bridge exchange lemma tells us that $S \cup x$ bridges y . It follows that $Q^* = (S \cup x)^*$ and we conclude that $S \cup x$ is, indeed, a spanning tree. ■

For the rest of the section we assume that (V, E, M) is a proper graph with adjacency relation G . Then $\Gamma : \mathcal{E} \rightarrow \text{Symm}(G)$ is an isomorphism and Δ is its inverse (see the definitions in Sec. 4 and Lemma 11). Moreover, a weight mapping w and a mapping *least* satisfying (7) are assumed to be given.

Lemma 32 *If $t \in \text{atoms}_{\mathcal{E}}(\text{least}(\mathbb{L}_{\mathcal{E}}))$ then t is contained in a minimum spanning tree.*

Proof. Let q be a spanning tree of \mathcal{E} and let $G =_{df} \Gamma(\mathbb{L}_{\mathcal{E}})$, $Q =_{df} \Gamma(q)$ and $x =_{df} \Gamma(t)$. Since Γ is an isomorphism, x is an edge. Moreover Q is a spanning tree of G and therefore bridges x . Thus, we may pick $y \in \text{edges}(x^{[Q]})$. Since Γ is an isomorphism, we have $y = \Gamma(s)$ for some $s \in \text{atoms}_{\mathcal{E}}(q)$. By the edge replacement lemma, $(Q \cap \overline{y}) \cup x$ is another spanning tree of G and it follows that $(q \cap \overline{s}) \cup t$ is a spanning tree of \mathcal{E} . By (7) we have $w(t) \leq w(s)$ so this new spanning tree is also minimal. ■

Lemma 33 *Assume that p is contained in a minimum spanning tree q and that $t \in \text{atoms}_{\mathcal{E}}(\text{least}(MM^{\cup}p \cap M\overline{M^{\cup}p}))$. Then also $p \cup t$ is contained in a minimum spanning tree.*

Proof. Let $P =_{df} \Gamma(p)$, $G =_{df} \Gamma(\mathbb{L}_{\mathcal{E}})$, $Q =_{df} \Gamma(q)$ and $x =_{df} \Gamma(t)$. Since Γ maps atoms to edges, Lemma 30 implies $x \in \text{edges}(G \cap (PL \text{ xor } \mathbb{L}P))$. Let $R =_{df} x^{[Q]}$, so that $x \subseteq R^*$ by the Galois connection (9). Define

$$R_1 =_{df} R \cap (PL \cup \mathbb{L}P) \quad R_2 =_{df} R \cap (\overline{P\mathbb{L}} \cup \overline{\mathbb{L}P}).$$

Obviously, $R = R_1 \cup R_2$. We claim that $R_1 \cap R_2 \neq \mathbf{O}$. Assume false. Then

$$R_1 R_2 \subseteq (R \cap \overline{R_2})(R \cap \overline{R_1}) \subseteq \overline{LP\mathbb{L}} = \mathbf{O}$$

and, similarly, $R_2R_1 = \mathbf{O}$, whence

$$x \subseteq R^* = (R_1 \cup R_2)^* = R_1^* \cup R_2^*.$$

But x is an edge and R_1 and R_2 are symmetric, so $x \subseteq R_1^*$ or $x \subseteq R_2^*$. Now the Galois connection (9) implies $x^{[Q]} \subseteq R_1$ or $x^{[Q]} \subseteq R_2$. Since $R = x^{[Q]}$ and $R_1 \cap R_2 = \mathbf{O}$ it follows that $R_1 = \mathbf{O}$ or $R_2 = \mathbf{O}$. But if, say, $R_1 = \mathbf{O}$ then

$$x \subseteq (R^* \cap \bar{1}) \cap (PL \cup LP) = \mathbf{O},$$

and a similar contradiction follows from $R_2 = \mathbf{O}$.

Thus, we have established $R_1 \cap R_2 \neq \mathbf{O}$. Now let $y \in \text{edges}(R_1 \cap R_2)$. Then we have that $y \neq x$ and $y \in \text{edges}(G \cap (PL \text{ xor } LP))$. By the edge replacement lemma, $(Q \cap \bar{y}) \cup x$ is a spanning tree of G which contains $P \cup x$. Just like in the previous proof we take $s \in \text{atoms}_{\mathcal{E}}(q)$ with $\Gamma(s) = y$ and it follows that $(q \cap \bar{s}) \cup t$ is a minimum spanning tree of \mathcal{E} which contains $p \cup t$. ■

7 Discussion

The length of our proofs may seem somewhat out of proportion with the difficulty of our results, and it is legitimate to question the adequacy of the relational method for deriving graph-theoretic algorithms. For one thing, the higher level of rigorousness imposed by the relational calculus has required us to explicitly prove a number of statements which the graphical intuition of the mathematician would have dismissed as obvious. But if you look through the proofs you will also find that many of the calculations deal with the properties of the Galois correspondence between incidence and adjacency relations. These results are not specific to spanning trees and should therefore be reusable. The present article is a first attempt at using relational algebra to develop algorithms on undirected and weighted graphs and we expect subsequent work to be more efficient.

There is another approach to the computation of spanning trees, which uses matroids rather than relations. It is based on the observation that the set of all cycle-free subgraphs of a graph forms a matroid (a family of sets that is closed under formation of subsets and satisfies a condition analogous to the well-known Steinitz exchange theorem for linearly independent subsets of a vector space). A spanning tree is a maximal element of this matroid. To find a spanning tree, start with T being the empty set of edges and repeatedly add edges to T while maintaining the invariant that T is an element of the matroid (i.e. T is a forest). Like ours, this scheme generalizes to weighted graphs; it leads to Kruskal's well-known algorithm for computing a minimum spanning tree [8].

It is worth noting that both Prim's and Kruskal's algorithm are "greedy" in the sense that each iteration of the loop adds exactly an edge to the tree resp. forest that will preserve the invariant. Unlike Prim's algorithm, Kruskal's algorithm does not maintain connectedness. When T is represented as a relation the lack of connectedness makes it harder to find an edge whose addition will preserve the invariant. Kruskal solved this problem by introducing a special data structure for representing forests, but that is a different story.

Acknowledgements We thank J. Ravelo for discussions on the proof of the bridge exchange lemma and also acknowledge the contribution of the anonymous referees.

References

1. Berghammer R., Schmidt G.: RELVIEW – A computer system for the manipulation of relations. In: Nivat M., et al. (eds.): Proc. AMAST '93, Workshops in Computing, Springer Verlag, 405-406 (1993)
2. Berghammer R., von Karger B., Ulke C.: Relation-algebraic analysis of Petri nets with RELVIEW. In: Margaria T., Steffen B. (eds.): Proc. TACAS '96, LNCS 1055, Springer Verlag, 49-69 (1996)
3. Hattensperger, C.: Computer-aided proofs in relational algebras (in German). Dissertation, Faculty of Computer Science, University of German Forces Munich (1997)
4. Brink C., Kahl W., Schmidt G. (eds.): Relational methods in computer science. Advances in Computing Science, Springer Verlag (1997)
5. Dijkstra E.W.: A discipline of programming. Prentice-Hall (1976)
6. Gries D.: The science of computer programming. Springer Verlag (1981)
7. Jungnickel D.: Graphen, Netzwerke und Algorithmen. BI Wissenschaftsverlag, 3rd ed. (1994)
8. Kruskal J.B.: On the shortest spanning subtree of a graph and the travelling salesman problem. Proc. AMS 7, 48-50 (1956)
9. Morgan C., Vickers T. (eds.): On the refinement calculus. Springer Verlag (1994)
10. Prim R.C.: Shortest connection networks and some generalizations. Bell Syst. Tech. J. 36, 1389-1401 (1957)
11. Schmidt G., Ströhlein T.: Relations and graphs. Discrete Mathematics for Computer Scientists, EATCS Monographs on Theoretical Computer Science, Springer Verlag (1993)

Appendix: Implementation

Relational algebra has a fixed and surprisingly small set of operations which – in the case of finite carrier sets – can be implemented very efficiently using Boolean arrays, predecessor resp. successor lists, or binary decision diagrams, for example. At Kiel University we have developed a visual computer system for calculating with relations, called RELVIEW [1, 2]. It is written in the C programming language and makes full use of the X-windows graphical user interface. For more information on the RELVIEW system, see the Web page <http://www.informatik.uni-kiel.de/~progsys/relview.html>.

The main purpose of RELVIEW is the evaluation of relational terms which are constructed from the relations of its workspace using pre-defined operations and tests, user-defined relational mappings, and user-defined relational programs. A RELVIEW program is much like a function procedure in Pascal or Modula 2, except that its basic data types are only relations. For example, our final program

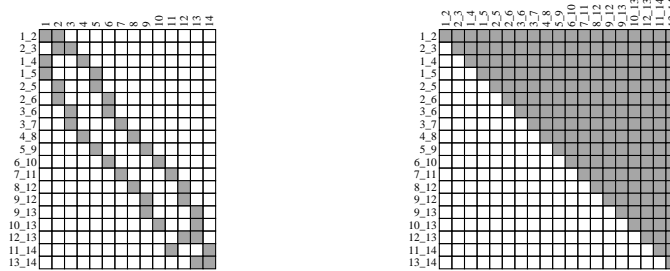
(8) for computing a minimum spanning tree in RELVIEW looks as follows:

```

Prim(M,W)
  DECL least(W,v) = v & -(-W*v);
  L, p, t
  BEG L = L(M);
  p = point(least(W,L));
  WHILE -eq(L^*M,p^*M) DO
    t = point(least(W,M*M^*p & M*(-(M^*p)));
    p = p | t OD
  RETURN p
END.

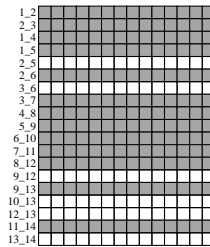
```

Let us now look at a concrete example. In RELVIEW all data are modeled as binary relations, which can be visualized as Boolean matrices. The input consists of two such matrices, say M and W , where M is the incidence relation of (V, E, M) with 14 vertices and 19 edges, and W is the pre-order $W \subseteq E \times E$ defined by the edge weights. RELVIEW displays M and W as follows:



To increase legibility, we have instructed RELVIEW to label the rows and columns of M and W . Vertices are simply numbered from 1 to 14, whereas each edge is labeled by the set of its vertices.

Next we ask RELVIEW to evaluate the relational term $\text{Prim}(M,W)$ and to put the edge labels on the rows of the result. On the relation window of the system we see the following 19×14 Boolean vector p . It represents a minimum spanning tree $p \subseteq E \times V$ of (V, E, M) wrt. the pre-order W :

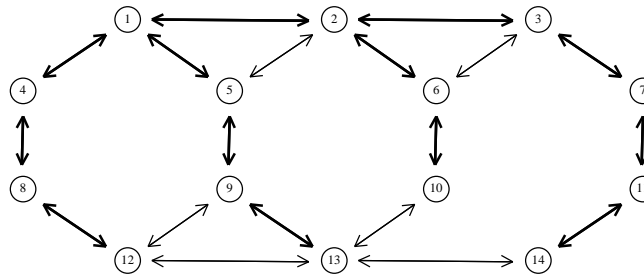


Homogeneous relations, such as $\Gamma(L_{\mathcal{E}})$, may be visualized as directed graphs and RELVIEW offers several sophisticated algorithms for drawing them nicely.

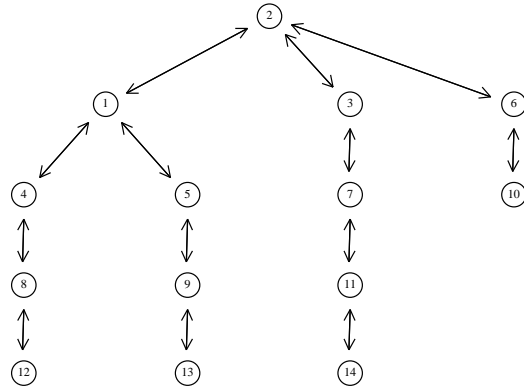
Moreover, a subrelation such as $\Gamma(p)$, may be visualized by printing its elements as bold arrows. To do this, we first ask RELVIEW to compute $G = \Gamma(L_{\mathcal{E}}) \subseteq V \times V$ and $T = \Gamma(p) \subseteq V \times V$ according to the definition in Sec. 4. After adding the vertex labels to the rows and columns we obtain the following two 14×14 Boolean matrices **Gamma_G** and **Gamma_p** on the system's relation window:



Then we ask the RELVIEW system to print the Boolean matrix **Gamma_G** as a graph while highlighting the subgraph corresponding to the matrix **Gamma_p**. The result is the following picture on the graph window of RELVIEW. It shows the graph that was given as input and a minimum spanning tree.



If we use the tree drawing algorithm of RELVIEW to draw the Boolean matrix **Gamma_p**, then we get the following picture:



The graphical representation of relations lends itself to visual editing and the most intuitive way of inputting an adjacency relation is to draw its graph on

RELVIEW's graph window. In the following we discuss how an incidence relation can be computed from its adjacency relation, so that also incidence relations can be fed into the system as graphs.

So assume we are given an irreflexive and asymmetric relation $R \subseteq V \times V$. We will now construct a relational expression for the incidence relation M of (V, E, M) such that $M \cup M \cap \bar{I} = R \cup R^{\cup}$. First, we compute from R a vector v which describes it as a subset of $V \times V$. Using the two projection functions π and ρ from $V \times V$ to the first resp. second component, we can write v as $v = (\pi R \cap \rho) \mathbb{L}$. Since R is irreflexive and asymmetric, it is isomorphic to the set $\{\{x, y\} \mid x, y \in V\}$ and can, therefore, be taken as set E of edges. With this choice, E is the subset of $V \times V$ given by the injective embedding function (see [11]) $inj(v) \subseteq E \times (V \times V)$ defined by the vector v . We get $M = inj(v)(\pi \cup \rho)$ and, thus, we arrive at the equation

$$M = inj((\pi R \cap \rho) \mathbb{L})(\pi \cup \rho).$$

RELVIEW knows about product spaces and their projection functions. The computation of the injective embedding function from a given non-empty vector is also implemented. With the aid of these features we can write

```

AssToInc(R)
  DECL Prod = PROD(R,R);
          pi, rho, M
  BEG   pi = p-1(Prod);
          rho = p-2(Prod);
          M = inj(dom(pi*R & rho)) * (pi | rho)
  RETURN M
END

```

as RELVIEW program for converting an adjacency relation R into an incidence relation M .