

Graphs and Algorithms in Constraint Satisfaction

Manuel Bodirsky

November 26, 2008

Contents

1	Introduction	2
2	Graph Homomorphisms	2
2.1	Graphs and Digraphs	2
2.2	Graph Homomorphisms	3
2.3	The H -coloring Problem	5
2.4	The Arc-consistency Procedure	6
2.5	Tree Duality and the Set Graph	8
2.6	Cores	11
2.7	Polymorphisms	14
2.8	The Path-consistency Procedure	16
2.9	Majority Operations	17
3	Constraint Satisfaction Problems	21
3.1	Relational Structures	21
3.2	The Constraint Satisfaction Problem	22
3.3	The Fundamental Lemma	25
3.4	Entailment, Finding a Solution, Finding all Solutions	26
3.5	Computational Complexity of CSPs.	26
3.6	CSPs, Logic, Databases	26
4	Constraint Propagation	29
4.1	Datalog	29
4.2	Local Consistency	30
4.3	The Existential Pebble-Game	30
5	Graph Algorithms in Constraint Satisfaction	33
5.1	The Min-Ordering Problem	33
5.2	Phylogenetic Reconstruction	34
5.3	Tree Description Constraints	38
5.4	Branching Time Constraints	40
5.5	Tree Descriptions in Computational Linguistics	42

1 Introduction

2 Graph Homomorphisms

2.1 Graphs and Digraphs

A *directed graph* (also *digraph*) G is a pair (V, E) of a set $V = V(G)$ of vertices and a binary relation $E = E(G)$ on V . Note that in general we allow that V is an infinite set. For some definitions in this section, we require that V is finite, in which case we say that G is a *finite digraph*. *Infinite* graphs will be important in later sections.

The elements (u, v) of E are called the *arcs* (or *directed edges*) of G . Note that we allow *loops*, i.e., arcs of the form (u, u) . If $(u, v) \in E(G)$ is an arc, and $w \in V(G)$ is a vertex such that $w = u$ or $w = v$, then we say that (u, v) and w are *incident*.

A digraph $G = (V, E)$ is called symmetric if $(u, v) \in E$ if and only if $(v, u) \in E$. Clearly, symmetric digraphs can be viewed as *undirected graphs*, and vice versa. Formally, an (*undirected*) *graph* is a pair (V, E) of a set $V = V(G)$ of vertices and a set $E = E(G)$ of *edges*, each of which is an unordered pair of (not necessarily distinct) elements of V .

For a digraph G , we say that G' is the *undirected graph of G* if G' is the undirected graph where $(u, v) \in E(G')$ iff $(u, v) \in E(G)$ or $(v, u) \in E(G)$. For an undirected graph G , we say that G' is an *orientation of G* if G' is a directed graph such that $V(G') = V(G)$ and $E(G')$ contains for each edge $(u, v) \in E(G)$ either the arc (u, v) or the arc (v, u) , and no other arcs.

Important examples of graphs, and corresponding notation.

- The digraph without vertices. This graph is unique.
- The *complete graph* on n vertices, denoted by K_n . This is an undirected graph on n vertices in which each vertex is joined with any other vertex.
- The *cyclic graph* on n vertices, denoted by C_n ; this is an undirected graph with the vertex set $\{v_1, \dots, v_n\}$ and edge set $\{(v_1, v_2), \dots, (v_{n-1}, v_n), (v_n, v_1)\}$.
- The *directed cycle* on n vertices, denoted by \vec{C}_n ; this is the digraph with the vertex set v_1, \dots, v_n and arc set $\{(v_1, v_2), \dots, (v_{n-1}, v_n), (v_n, v_1)\}$.
- The *path* with $n + 1$ vertices and n edges, denoted by P_n ; this is an undirected graph with the vertex set $\{v_0, \dots, v_n\}$ and edge set $\{(v_0, v_1), \dots, (v_{n-1}, v_n)\}$.
- The *directed path* with $n + 1$ vertices and n edges, denoted by \vec{P}_n ; this is a digraph with the vertex set $\{v_0, \dots, v_n\}$ and edge set $\{(v_0, v_1), \dots, (v_{n-1}, v_n)\}$.
- The *transitive tournament* on $n \geq 2$ vertices, denoted by T_n ; this is a directed graph with the vertex set $\{v_1, \dots, v_n\}$ where (v_i, v_j) is an arc if and only if $i < j$.

Let G and H be graphs (we define the following notions both for directed and for undirected graphs). Then $G + H$ denotes the *disjoint union of G and H* , which is the graph with vertex set $V(G) \cup V(H)$ (we assume that the two vertex sets are disjoint; if they are not, we take an copy of H and form the disjoint union of G with the copy of H) and edge set $E(G) \cup E(H)$. A graph G' is a *subgraph* of G if $V(G') \subseteq V(G)$ and $E(G') \subseteq E(G)$. A graph G' is an *induced subgraph* of G if $V' = V(G') \subseteq V(G)$ and $(u, v) \in E(G')$ if and only if

$(u, v) \in E(G)$ for all $u, v \in V'$. We also say that G' is *induced by V' in G* , and write $G[V']$ for G' . We write $G - u$ for $G[V(G) \setminus \{u\}]$, i.e., for the subgraph of G where the vertex u and all incident arcs are removed.

We call $|V(G)| + |E(G)|$ the *size* of a graph G ; note that the size of a graph G reflects the length of the representation of G as a bit-string (under a reasonable choice of graph representations). This quantity will be important if we analyze the efficiency of algorithms on graphs.

The following concepts are for undirected graphs only. We say that an undirected graph G contains a *cycle* if there is a sequence v_1, \dots, v_k of $k \geq 3$ pairwise distinct vertices of G such that $(v_1, v_k) \in E(G)$ and $(v_i, v_{i+1}) \in E(G)$ for all $1 \leq i \leq k - 1$. An undirected graph is called *acyclic* if it does not contain a cycle. A sequence $u_1, \dots, u_k \in V(G)$ is called a (*simple*) *path* from u_1 to u_k in G iff $(u_i, u_{i+1}) \in E(G)$ for all $1 \leq i < k$ and if all vertices u_1, \dots, u_k are pairwise distinct. Two vertices $u, v \in G$ are *at distance k* in G iff the shortest path in G from u to v has length k . We say that an undirected graph G is *connected* if for all vertices $u, v \in V(G)$ there is a path from u to v . A *forest* is an undirected acyclic graph, a *tree* is a connected forest.

We conclude with definitions that are made for directed graphs. Let G be a digraph. A sequence u_1, \dots, u_k with $(u_i, u_{i+1}) \in E(G)$ for all $1 \leq i \leq k - 1$ is called a *directed path* from u_1 to u_k . A sequence u_1, \dots, u_k is called a *directed cycle* of G if it is a directed path and additionally (u_k, u_1) is an arc of G . For some notions for digraphs G we just use the corresponding notions for undirected graphs applied to the undirected graph of G : A digraph G is called *weakly connected* if the undirected graph of G is connected. A graph is called *strongly connected* if for all vertices $x, y \in V(G)$ there is a directed path from x to y in G . Two vertices $u, v \in V(G)$ are *at distance k* in G if they are at distance k in the undirected graph of G .

Most of the definitions for graphs in this text are analogous for directed and for undirected graphs. We therefore sometimes do not explicitly mention whether we work with directed or undirected graphs, but just state certain concepts for *graphs*, for instance in Subsection 2.6 or 2.7. We want to remark that almost all definitions in this section have generalizations to *relational structures*, and these generalizations will be important for constraint satisfaction in Section 3.

Exercises.

1. Show that a digraph G is weakly connected if and only if for all digraphs H and H' such that $G = H + H'$ the digraph H or the digraph H' is the digraph without vertices.

2.2 Graph Homomorphisms

Let G and H be directed graphs. A *homomorphism* from G to H is a mapping from $V(G)$ to $V(H)$ such that $(f(u), f(v)) \in E(H)$ whenever $(u, v) \in E(G)$. If such a homomorphism exists between G and H we say that G *homomorphically maps* to H . Two directed graphs G and H are *homomorphically equivalent* if there is a homomorphism from G to H and a homomorphism from H to G .

A homomorphism from G to H is sometimes also called an *H -coloring* of G . This terminology originates from the observation that H -colorings generalize classical colorings in the sense that a graph is n -colorable if and only if it has a K_n -coloring. Graph n -colorability is

not the only natural graph property that can be described in terms of homomorphisms. A digraph is called *balanced* if it homomorphically maps to a directed path \vec{P}_n . A digraph is called *acyclic* if it homomorphically maps to a transitive tournament T_n .

The class of all finite digraphs, denoted by \mathcal{D} , can be *ordered* by the existence of homomorphisms. Formally, let \leq be a binary relation defined on \mathcal{D} by $H_1 \leq H_2$ iff there exists a homomorphism from H_1 to H_2 . If f is a homomorphism from H_1 to H_2 , and g is a homomorphism from H_2 to H_3 , then the composition $f \circ g$ of these functions is a homomorphism from H_1 to H_3 , and therefore the \leq is transitive. Since every graph H homomorphically maps to H , the order \leq is also reflexive. Thus, (\mathcal{D}, \leq) is a quasi-order. This order is called the *homomorphism order* for finite digraphs. An *antichain* in (\mathcal{D}, \leq) is a set of graphs \mathcal{S} such that for all graphs $H_1, H_2 \in \mathcal{S}$ neither $H_1 \leq H_2$ nor $H_2 \leq H_1$.

We want to illustrate the homomorphism order by presenting an infinite antichain of digraphs. For this we need the following graphs, called *zig-zags*, which are frequently used in the theory of graph homomorphisms. We may write an oriented path P as a sequence of 0's and 1's, where 0 represents a forward arc and 1 represents a backward arc. For two oriented paths P and Q with the representation $p_0, \dots, p_n \in \{0, 1\}^*$ and $Q = q_0, \dots, q_m \in \{0, 1\}^*$, respectively, the *concatenation* $P \circ Q$ of P and Q is the oriented path represented by $p_0, \dots, p_n, q_1, \dots, q_m$. The *net length* $nl(P)$ of an oriented path P represented by p_1, \dots, p_n is defined to be $|\sum_{i=1}^n (2p_i - 1)|$.

Definition 1. For $k \geq 1$, the zig-zag of order k , denoted by Z_k , is the oriented path represented by $11(01)^{k-1}1$. For $k, l \geq 2$, the k, l multi zig-zag, denoted by $Z_{k,l}$, is the oriented path represented by $1(1(01)^{k-1})^{l-1}1$.

We recommend the reader to draw pictures of $Z_{k,l}$ where forward arcs point up and backward arcs point down.

Proposition 2. The homomorphism order contains infinite antichains.

Proof. Our antichain is $\{Z_{k,k} \mid k \geq 2\}$. □

A *strong homomorphism* between two directed graphs G and H is a mapping from $V(G)$ to $V(H)$ such that $(f(u), f(v)) \in E(H)$ if and only if $(u, v) \in E(G)$ for all $u, v \in V(G)$. An *isomorphism* between two directed graphs G and H is a bijective strong homomorphism from G to H , i.e., $f(u) \neq f(v)$ for any two distinct vertices $u, v \in V(G)$.

Exercises.

2. Show that homomorphic equivalence is an equivalence relation.
3. Show that a digraph G homomorphically maps to \vec{P}_1 if and only if \vec{P}_2 does not map to G .
4. Construct an orientation of a tree that is not homomorphically equivalent to a directed path.
5. Construct a balanced orientation of a cycle that is not homomorphically equivalent to a directed path.
6. Show that a digraph G homomorphically maps to T_3 if and only if \vec{P}_3 does not map to G .

2.3 The H -coloring Problem

When does a given digraph G homomorphically map to a digraph H ? For every digraph H , this question defines a computational problem, called the H -coloring problem. The input of the H -coloring problem is a finite digraph G , and the question is whether there exists a homomorphism from G to H . Note that since graphs are a special case of digraphs, the H -coloring problem is also defined for undirected graphs H . In this case we obtain essentially the same computational problem if we allow only undirected graphs in the input; this is made precise in Exercise 9.

For every finite graph H , the H -coloring problem is obviously in NP, because for every graph G it can be verified in polynomial time whether a given mapping from $V(G)$ to $V(H)$ is a homomorphism from G to H or not. We have also seen that the H -coloring problem becomes the problem of l -colorability if H equals K_l . Therefore, we already know digraphs H where the H -coloring problem is NP-complete. However, for many digraphs H (see Exercise 10 and 11) the H -coloring problem can be solved in polynomial time. The following research question is open and probably very difficult.

Question 1. *For which digraphs H can the H -coloring problem be solved in polynomial time?*

It has been conjectured by Feder and Vardi [18] that for any finite digraph H the H -coloring problem is either NP-complete or can be solved in polynomial time. This is the so-called *dichotomy conjecture* and it is open as well. The dichotomy conjecture is remarkable since it was shown by Ladner that unless $P=NP$ there are infinitely many complexity classes between P and NP. We will come back to this conjecture in Section 3.

It is known that the dichotomy conjecture is true for finite *undirected* graphs. The H -coloring problem can be solved in polynomial time if H is homomorphically equivalent to K_2 (see Exercise 10); for all other finite undirected graphs H the H -coloring problem is NP-complete (this was proven by Hell and Nešetřil [22]).

Theorem 3 (of [22]). *If H is not homomorphically equivalent to K_2 , the H -coloring problem is NP-complete.*

Exercises.

7. Let H be a finite directed graph. Find an algorithm that decides whether there is a strong homomorphism from a given graph G to the fixed graph H . The running time of the algorithm should be polynomial in the size of G (note that we consider $|V(H)|$ to be constant).
8. Let H be a finite digraph such that the H -coloring problem can be solved in polynomial time by an algorithm A . Use algorithm A to find an algorithm B that constructs for a given finite digraph G a homomorphism to H , if such a homomorphism exists.
9. Let G and H be directed graphs, and suppose that H is symmetric. Show that $f : V(G) \rightarrow V(H)$ is a homomorphism from G to H if and only if f is a homomorphism from the undirected graph of G to the undirected graph of H .
10. Show that for any graph H that is homomorphically equivalent to K_2 the H -coloring problem can be solved in polynomial time.

<p>AC(G) Input: a finite digraph G. Data structure: a list $L(x)$ for each vertex $x \in V(G)$.</p> <p>Set $L(x) := V(H)$ for all $x \in V(G)$. Do For each $(x, y) \in E(G)$: Remove u from $L(x)$ if there is no $v \in L(y)$ with $(u, v) \in E(H)$. Remove v from $L(y)$ if there is no $u \in L(x)$ with $(u, v) \in E(H)$. If $L(x)$ is empty for some vertex $x \in V(G)$ then reject Loop until no list changes</p>
--

Figure 1: The arc-consistency procedure for H -coloring

11. Prove that for $H = T_3$ the H -coloring problem can be solved in polynomial time.
12. Prove that for $H = \vec{C}_3$ the H -coloring problem can be solved in polynomial time.

2.4 The Arc-consistency Procedure

The *arc-consistency procedure* is one of the most fundamental and well-studied algorithms that is applied for H -coloring problems. This procedure was first discovered for constraint satisfaction problems in Artificial Intelligence [33, 35]; in the graph homomorphism literature, the algorithm is sometimes called the *consistency check algorithm*.

Let H be a finite digraph, and let G be an instance of the H -coloring problem. The idea of the procedure is to maintain for each vertex in G a list of vertices of H , and each element in the list of x represents a candidate for an image of x under a homomorphism from G to H . The algorithm successively removes vertices from these lists; it only removes a vertex $u \in V(H)$ from the list for $x \in V(G)$, if there is no homomorphism from G to H that maps x to u . To detect vertices x, u such that u can be removed from the list for x , the algorithm uses two rules (in fact, one rule and a symmetric version of the same rule): if (x, y) is an edge in G , then

- remove u from $L(x)$ if there is no $v \in L(y)$ with $(u, v) \in E(H)$;
- remove v from $L(y)$ if there is no $u \in L(x)$ with $(u, v) \in E(H)$.

If eventually we can not remove any vertex from any list with these rule any more, the graph G together with the lists for each vertex is called *arc-consistent*. We will return to the notion of arc-consistency in Section 3, and place it into the more general framework of constraint satisfaction problems. Clearly, if the algorithm removes all vertices from one of the lists, then there is no homomorphism from G to H . The pseudo-code of the entire arc-consistency procedure is displayed in Figure 1.

The running time of the arc-consistency procedure is for any fixed digraph H polynomial in the size of G . In a naive implementation of the procedure, the inner loop of the algorithm would go over all edges of the graph, in which case the running time of the algorithm is quadratic in the size of G . In the following we describe an implementation of the arc-consistency procedure, called AC-3, which is due to Mackworth [33], and has a worst-case running time that is linear in the size of G . Several other implementations of the

```

AC-3( $G$ )
Input: a finite digraph  $G$ .
Data structure: a list  $L(x)$  of vertices of  $H$  for each  $x \in V(G)$ .
           the worklist  $W$ : a list of arcs of  $G$ .

Subroutine Revise( $(x_0, x_1), i$ )
Input: an arc  $(x_0, x_1) \in E(G)$ , an index  $i \in \{0, 1\}$ .
change = false
for each  $u_i$  in  $L(x_i)$ 
    If there is no  $u_{1-i} \in L(x_{1-i})$  such that  $(u_0, u_1) \in E(H)$  then
        remove  $u_i$  from  $L(x_i)$ 
        change = true
    end if
end for
If change = true then
    If  $L(x_i) = \emptyset$  then reject
else
    For all arcs  $(z_0, z_1) \in E(G)$  with  $z_0 = x_i$  or  $z_1 = x_i$  add  $(z_0, z_1)$  to  $W$ 
end if

 $W := E(G)$ 
Do
    remove an arc  $(x_0, x_1)$  from  $W$ 
    Revise( $(x_0, x_1), 0$ )
    Revise( $(x_0, x_1), 1$ )
while  $W \neq \emptyset$ 

```

Figure 2: The AC-3 implementation of the arc-consistency procedure for H -coloring

arc-consistency algorithm have been proposed in the Artificial Intelligence literature, aiming at reducing the costs of the algorithm in terms of the number of vertices of H . But here we consider the size of H to be fixed, and therefore we do not follow this line of research. With AC-3, we rather present one of the simplest implementations of the arc-consistency procedure with a linear running time.

The idea of AC-3 is to maintain a *worklist*, which contains a list of arcs (x_0, x_1) of G that might help to remove a value from $L(x_0)$ or $L(x_1)$. Whenever we remove a value from a list $L(x)$, we add all arcs that are in G incident to x . Note that then any arc in G might be added at most $2|V(H)|$ many times to the worklist, which is a constant in the size of G . Hence, the while loop of the implementation is iterated for at most a linear number of times. Altogether, the running time is linear in the size of G as well.

Note that for any finite digraph H , if the arc-consistency procedure rejects an instance of the H -coloring problem, it clearly has no solution. The converse implication does not hold in general. For instance, let H be K_2 , and let G be K_3 . In this case, the arc-consistency procedure does not remove any vertex from any list, but obviously there is no homomorphism from K_3 to K_2 .

However, there are digraphs H where the arc-consistency procedure is a complete decision procedure for the H -coloring problem in the sense that it rejects an instance G of the H -coloring problem if and only if G does not homomorphically map to H . In this case we say

that the arc-consistency procedure *solves* the H -coloring problem.

Arc-consistency for pruning search. Even if we are interested in an H -coloring problem which is not solved by the arc-consistency procedure (unless $P=NP$, we know that this is the case for all digraphs H with an NP-complete H -coloring problem) the arc-consistency procedure might be useful for *pruning the search space* in exhaustive approaches to solve the H -coloring problem.

In such an approach we might use the arc-consistency procedure as a subroutine as follows. Initially, we run the arc-consistency procedure on the input instance G . If it computes an empty list, we reject. Otherwise, we select some vertex $x \in V(G)$, and set $L(x)$ to $\{u\}$ for some $u \in L(x)$. Then we proceed recursively with the resulting lists. If the arc-consistency now detects an empty list, we backtrack, but remove u from $L(x)$. Finally, if the algorithm does not detect an empty list at the first level of the recursion, we end up with singleton lists for each vertex $x \in V(G)$, which defines a homomorphism from G to H .

2.5 Tree Duality and the Set Graph

For which H does the arc-consistency procedure solve the H -coloring problem? We give two answers to this question, and present two characterizations of those finite graphs H where the arc-consistency procedure solves the H -coloring problem.

The first relies on the notion of *tree duality*. The idea of this concept is that when a digraph H has tree duality, then the question whether there is a homomorphism from a digraph G to H can be answered by the existence of a *tree obstruction* in G . This is formalized in the following definition.

Definition 4. *A digraph H has tree duality iff there exists a (not necessarily finite) set \mathcal{N} of orientations of finite trees such that for all digraphs G there is a homomorphism from G to H if and only if no digraph in \mathcal{N} homomorphically maps to G .*

We think of the set \mathcal{N} in Definition 4 as an *obstruction set* for the H -coloring problem. The pair (\mathcal{N}, H) is called a *duality pair*. We have already encountered such an obstruction set in Exercise 11, where $H = T_2$, and $\mathcal{N} = \{\bar{P}_2\}$. In other words, $(\{P_2\}, T_2)$ is a duality pair.

For the second characterization we introduce the notion of *set graphs*.

Definition 5. *For a digraph H , the (power-) set graph $P(H)$ is the digraph whose vertices are non-empty subsets of $V(H)$ and where two subsets U and V are joined by an arc if the following holds:*

- for every vertex $u \in U$, there exists a vertex $v \in V$ such that $(u, v) \in E(H)$, and
- for every vertex $v \in V$, there exists a vertex $u \in U$ such that $(u, v) \in E(H)$.

Theorem 6 is a surprising link between the completeness of the arc-consistency procedure, tree duality, and the set graph, and was discovered by Feder and Vardi [17] in the more general context of constraint satisfaction problems.

Theorem 6. *Let H be a finite digraph. Then the following are equivalent.*

1. H has tree duality;

2. If every orientation of a tree that homomorphically maps to G also maps to H , then G homomorphically maps to H ;
3. $P(H)$ homomorphically maps to H ;
4. The arc-consistency procedure solves the H -coloring problem.

Proof. We prove implications between these statements in cyclic order.

1 \rightarrow 2: Suppose H has tree duality, and let \mathcal{N} be the tree obstructions from Definition 4. Let G be a digraph, and suppose that every tree that homomorphically maps to G also maps to H . We have to show that G homomorphically maps to H . No member of \mathcal{N} homomorphically maps to H : otherwise by the definition of tree duality H would not be homomorphic to H , a contradiction. So, none of the orientations of trees in \mathcal{N} homomorphically maps to G , and by tree duality, G homomorphically maps to H .

2 \rightarrow 3: To show that $P(H)$ homomorphically maps to H , it suffices to prove that every orientation T of a tree that homomorphically maps to $P(H)$ also maps to H . Let f be a homomorphism from T to $P(H)$, and let x be any vertex of T . We construct a sequence f_0, \dots, f_n , for $n = |V(T)|$, where f_i is a homomorphism from the subgraph of T induced by the vertices at distance at most i to x in T , and f_{i+1} is an extension of f_i for all $1 \leq i \leq n$. The mapping f_0 maps x to some vertex from $f(x)$. Suppose inductively that we have already defined f_i . Let y be a vertex at distance $i + 1$ from x in T . Since T is an orientation of a tree, there is a unique $y' \in V(T)$ of distance i from x in T such that $(y, y') \in E(T)$ or $(y', y) \in E(T)$. Note that $u = f_i(y')$ is already defined. In case that $(y', y) \in E(T)$, there must be a vertex v in $f(y)$ such that $(u, v) \in E(H)$, since $(f(y'), f(y))$ must be an arc in $P(H)$, and by definition of $P(H)$. We then set $f_{i+1}(y) = v$. In case that $(y, y') \in E(T)$ we can proceed analogously. By construction, the mapping f_n is a homomorphism from T to H .

3 \rightarrow 4: Now, suppose that there is a homomorphism f from $P(H)$ to H , and let G be an instance of the H -coloring problem. We have to show that if the arc-consistency procedure does not derive an empty list for all vertices $u \in V(G)$, then there is a homomorphism from G to H . We claim that the mapping g that assigns $x \in V(G)$ the list $L(x)$ of the final stage of the arc-consistency algorithm defines a homomorphism from G to $P(H)$. Suppose otherwise that there is an arc $(x, y) \in E(G)$ such that $(g(x), g(y)) \notin E(P(H))$. Then there is a $u \in g(x)$ such that there is no $v \in g(y)$ with $(u, v) \in E(H)$, or a $v \in g(y)$ such that there is no $u \in g(x)$ with $(u, v) \in E(H)$. In the first case, the arc-consistency procedure would have removed u from $L(x)$, in the second case it would have removed v from $L(y)$, a contradiction in both cases. Composing the homomorphism g from G to $P(H)$ and the homomorphism f from $P(H)$ to H , we find a homomorphism from G to H .

4 \rightarrow 1: Suppose that the arc-consistency procedure solves the H -coloring problem. We have to show that H has tree duality. Let \mathcal{N} be the set of all orientations of trees that does not homomorphically map to H . We claim that if a digraph G does not homomorphically map to H , then there is $T \in \mathcal{N}$ that homomorphically maps to G .

By assumption, the arc-consistency procedure applied to G eventually derives the empty list for some vertex of G . We use the computation of the procedure to construct an orientation T of a tree, following the exposition in [31]. When deleting a vertex $u \in V(H)$ from the list of a vertex $x \in V(G)$, we define an orientation of a rooted tree $T_{x,u}$ with root $r_{x,u}$ such that

1. There is a homomorphism from $T_{x,u}$ to G mapping $r_{x,u}$ to x .
2. There is no homomorphism from $T_{x,u}$ to H mapping $r_{x,u}$ to u .

The vertex u is deleted from the list of x because we found an arc $(x_0, x_1) \in E(G)$ with $x_i = x$ for some $i \in \{0, 1\}$ such that there is no arc $(u_0, u_1) \in E(H)$ with $u_i = u$ and u_{1-i} in the list of x_{1-i} . We describe how to construct the tree T for the case that $i = 0$, i.e., for the case that there is an arc $(x, y) \in E(G)$ such that there is no arc $(u, v) \in E(H)$ where $v \in L(y)$; the construction for $i = 1$ is analogous.

Let p, q be vertices and (p, q) an edge of $T_{x,u}$, and select the root $r_{x,u} = p$. If $E(H)$ does not contain an arc (u, v) , then we are done, since $T_{x,u}$ clearly satisfies property (1) and (2). Otherwise, for every arc $(u, v) \in E(H)$ the vertex v has already been removed from the list $L(y)$, and hence by induction $T_{y,v}$ having properties (1) and (2) is already defined. We then add a copy of $T_{y,v}$ to $T_{x,u}$ and identify the vertex $r_{y,v}$ with q .

We verify that the resulting orientation of a tree $T_{x,u}$ satisfies (1) and (2). Let f be the homomorphism from $T_{y,v}$ mapping $r_{y,v}$ to v , which exists due to (1). The extension of f to $V(T_{x,u})$ that maps p to x is a homomorphism from $T_{x,u}$ to G , and this shows that (1) holds for $T_{x,u}$. But any homomorphism from $T_{x,u}$ to H that maps $r_{x,u}$ to u would also map the root of $T_{y,v}$ to v , which is impossible, and this shows that (2) holds for $T_{x,u}$. When the list $L(x)$ of some vertex $x \in V(G)$ becomes empty, we can construct an orientation of a tree T by identifying the roots of all $T_{x,u}$ to a vertex r . We then find a homomorphism from T to G by mapping r to x and extending the homomorphism independently on each $T_{x,u}$. But any homomorphism from T to H must map r to some element $u \in V(H)$, and hence there is a homomorphism from $T_{x,u}$ to H that maps x to u , a contradiction.

Therefore, for every graph G that does not homomorphically map to H we find an orientation of a tree from \mathcal{N} that homomorphically maps to G , and hence H has tree-duality. \square

Note that the last criterion of Theorem 6 can be used to decide algorithmically whether a given digraph H has tree duality, because we simply have to decide whether $P(H)$ homomorphically maps to H . However, the naive algorithm for this approach has a very bad worst-case running time, because the graph $P(H)$ has exponential size in H , and moreover we only know a *non-deterministic* polynomial time algorithm to test whether $P(H)$ homomorphically maps to H .

Question 2. *What is the computational complexity to decide for a given digraph H whether H has tree duality?*

Despite the potentially bad worst-case complexity of deciding whether $P(H)$ homomorphically maps to H , this condition can well be verified for several families of graphs H . We will mention one of them, namely the family of oriented paths.

Corollary 7. *The H -coloring problem can be solved by the arc-consistency procedure if H is an oriented path.*

Proof. Suppose that $1, \dots, n$ are the vertices of H such that either $(i, i+1)$ or $(i+1, i)$ is an arc in $E(H)$ for all $i < n$. We claim that the mapping $f : V(P(H)) \rightarrow V(H)$ defined by $f(S) = \min(S)$ is a homomorphism from $P(H)$ to H .

Let (U, W) be an arc in $P(H)$, and suppose for contradiction that $(\min(U), \min(W))$ is not an arc in H . By the property of $P(H)$ there must be an element w of W such that $(\min(U), w) \in E(H)$, and an element u of U such that $(u, \min(W)) \in E(H)$. Note that due to the way in which the vertices are numbered, if (k, l) is an edge of $E(H)$ then $k-1 \leq l \leq k+1$. Hence,

$$\min(U) \leq w + 1 \leq \min(W) \leq u + 1 \leq \min(U) ,$$

and therefore $(\min(U), w)$ and $(w, \min(U)) = (u, \min(W))$ are edges in opposite directions in H , which is impossible in an oriented path. The corollary now follows from Theorem 6. \square

We want to remark that there are orientations of trees H with an NP-complete H -coloring problem (the smallest known graph has 45 vertices [24]). So, unless $P=NP$, this shows us that there are orientations of trees H that do not have tree-duality.

Exercises.

13. Let H be a finite digraph. Show that $P(H)$ contains a loop if and only if H contains a directed cycle.
14. Show that the previous statement is false for infinite digraphs H .
15. Recall that a digraph is called *balanced* if it homomorphically maps to a directed path. Let H be a digraph.
 - Prove: if H is balanced, then $P(H)$ is balanced;
 - Disprove: if H is an orientation of a tree, then $P(H)$ is an orientation of a forest.
16. Show that an orientation of a tree homomorphically maps to H if and only if it maps to $P(H)$ (Hint: use parts of the proof of Theorem 6).
17. Let \mathcal{N} be the set of all zig-zags, $\{Z_1, Z_2, Z_3, \dots\}$. Show that a digraph G homomorphically maps to \vec{F}_2 if and only if no digraph in \mathcal{N} homomorphically maps to G .

2.6 Cores

An *endomorphism* of a graph H is a homomorphism from H to H . An *automorphism* of a graph H is an isomorphism from H to H . A finite graph H is called a *core* if every endomorphism of H is an automorphism. A subgraph G of H is called a *core of H* if H is homomorphically equivalent to G and G is a core.

Proposition 8. *Every finite graph H has a core, which is unique up to isomorphism.*

Proof. Any finite graph H has a core, since we can select an endomorphism e of H such that the image of e has smallest cardinality; the subgraph of H induced by $e(V(H))$ is a core of H . Let G_1 and G_2 be cores of H , and $f_1 : H \rightarrow G_1$ and $f_2 : H \rightarrow G_2$ be homomorphisms. Then $f_1 \circ f_2$ restricted to $V(G_1)$ is an endomorphism of G_1 . Since G_1 is a core, the mapping $f_1 \circ f_2$ is an automorphism of G_1 . For finite structure this implies that f_1 restricted to $V(G_2)$ is an isomorphism between G_2 and G_1 . \square

Since a core G of a finite digraph H is unique up to isomorphism, we call G *the* core of H . Cores can be characterized in many different ways; for some of them, see Exercise 19. There are examples of infinite digraphs that do not have a core in the sense defined above; see Exercise 21. Since a digraph H and its core have the same H -coloring problem, it suffices to study the H -coloring problem for core graphs H only. As we will see below, this has several advantages.

The *pre-colored H -coloring problem* for a digraph H is the following computational problem. Given is a finite digraph G and a partial mapping $c : V(G) \rightarrow V(H)$. The question is

whether c can be extended to a homomorphism from G to H . In informal words, we want to find a homomorphism from G to H where some vertices have a pre-described image.

Proposition 9. *Let H be a core. Then the H -coloring problem and the pre-colored H -coloring problem are linear-time equivalent.*

Proof. The reduction from the H -coloring problem to the pre-colored H -coloring problem is trivial, because an instance G of the H -coloring problem is equivalent to the instance (G, c) of the pre-colored H -coloring problem where c is everywhere undefined. The reduction of the pre-colored to the standard H -coloring problem we show by induction on the size of the image of partial mapping c in instances of the pre-colored H -coloring problem. Let (G, c) be an instance of the pre-colored H -coloring where c has an image of size $k \geq 1$. We show how to reduce the problem to one where the partial mapping has an image of size $k - 1$. If we compose all these reductions (note that the size of the image is bounded by $|V(H)|$), we finally obtain a reduction to the H -coloring problem.

Let $x \in V(G)$ and $u \in V(H)$ be such that $c(x) = u$. We first identify all vertices y of G such that $c(y) = u$ with x . Then we create a copy of H , and attach the copy to G by identifying $x \in V(G)$ with $u \in V(H)$. Let G' be the resulting graph (note that the size of G' and the size of G only differ by a constant), and let c' be the restriction of c that is undefined on x .

We claim that (G', c') has a solution for the pre-colored H -coloring problem if and only if (G, c) has a solution for the pre-colored H -coloring problem. If f is a homomorphism from G to H that extends c , we further extend f to the copy of H that is attached in G' by setting $f(u')$ to u if vertex u' is a copy of a vertex $u \in V(H)$. This extension of f clearly is a homomorphism from G' to H and extends c' .

Now, suppose that f' is a homomorphism from G' to H that extends c' . The restriction of f' to the vertices from the copy of H that is attached to x in G' is an endomorphism of H , and because H is a core, is an automorphism α of H . Let β be the inverse of α , i.e., let β be the automorphism of H such that $\beta(\alpha(v)) = v$ for all $v \in V(H)$. Let f be the mapping from $V(G)$ to $V(H)$ that maps vertices that were identified with x to $\beta(f'(x))$, and all other vertices $y \in V(G)$ to $\beta(f'(y))$. Clearly, f is a homomorphism from G to H . Since f' , α , and therefore also β map vertices $y \in V(G)$, $y \neq x$, where c is defined to $c(y)$. Moreover, because x in G' is identified to u in the copy of H , we have that $f(x) = \beta(f'(x)) = \beta(f'(u)) = u$, and therefore f is an extension of c . \square

We have already seen in Exercise 8 that the computational problem to construct a homomorphism from G to H , for fixed H and given G , can be reduced in polynomial-time to the problem of deciding whether there exists a homomorphism from G to H . The intended solution of Exercise 8 requires in the worst-case $|V(G)|^2$ many executions of the decision procedure for the H -coloring problem. Using the concept of cores and the pre-colored H -coloring problem (and its equivalence to the H -coloring problem) we can give a faster method to construct homomorphisms.

Proposition 10. *If there is an algorithm that decides the H -coloring problem in time T , then there is an algorithm that constructs a homomorphism from a given graph G to H (if such a homomorphism exists). The algorithm runs in time $O(|V(G)|T)$.*

Proof. We can assume without loss of generality that H is a core (since H and its core have the same H -coloring problem). By Proposition 9, there is an algorithm B for the pre-colored

H -coloring problem with a running time in $O(T)$. For given G , we first apply B to (G, c) for the everywhere undefined function c to decide whether there exists a homomorphism from G to H . If no, there is nothing to show. If yes, we select some $x \in V(G)$, and extend c by defining $c(x) = u$ for some $u \in V(H)$. Then we use algorithm B to decide whether there is a homomorphism from G to H that extends c . If no, we try another vertex $u \in V(H)$. Clearly, for some u the algorithm must give the answer “yes”. We proceed with the extension c where $c(x) = u$, and repeat the procedure with another vertex x from $V(G)$. At the end, c is defined for all vertices u of G , and c is a homomorphism from G to H . Clearly, since H is fixed, algorithm B is executed at most $O(|V(G)|)$ many times. \square

It turns out that *almost all graphs*, as well as almost all digraphs, are cores. It is even true that almost all graphs have only one endomorphism, namely the identity mapping $u \mapsto u$.

Definition 11. A digraph H is called rigid if the identity mapping is the only endomorphism of H .

The property of being rigid is stronger than the property of being a core, because rigid graphs are *asymmetric*, i.e., the identity is their only automorphism, whereas cores need not be asymmetric.

To formalize our statements about almost all digraphs, we say that a random digraph has a property A with high probability (whp) iff the probability that a digraph with n vertices drawn from the uniform distribution of all digraphs on n vertices has property A tends to 1 when n goes to infinity. The following theorem is stated without proof.

Theorem 12 (see [23]). A random digraph is with high probability rigid.

Another fact we want to mention without proof is that it is NP-complete to decide whether a given digraph H is a core [21].

Exercises.

18. Show that $Z_{k,l}$ is a core for all $k, l \geq 2$.
19. Prove that for every finite directed graph G the following is equivalent:
 - G is a core
 - Every endomorphism of G is injective
 - Every endomorphism of G is surjective
20. Prove that the core of a strongly connected graph is strongly connected.
21. Show that the infinite digraph $(\mathbb{Q}, <)$ has endomorphisms that are not automorphisms. Show that every digraph that is homomorphically equivalent to $(\mathbb{Q}, <)$ also has endomorphisms that are not automorphisms.
22. Adapt the arc-consistency procedure to the pre-colored H -coloring problem. Prove that for core digraphs H , if the arc-consistency procedure solves the H -coloring problem, then the arc-consistency procedure also solves the pre-colored H -coloring problem.
23. There is only one unbalanced cycle H on four vertices that is a core and not the directed cycle. Show that for this graph H the H -coloring problem can not be solved by the arc-consistency procedure.

2.7 Polymorphisms

Polymorphisms are a versatile tool for analyzing the computational complexity of H -coloring problems; as we will see, they are useful both for NP-hardness proofs and for proving the correctness of polynomial-time algorithms for H -coloring problems.

Polymorphisms can be seen as multi-dimensional variants of endomorphisms. To define them, we need the notion of *direct products* of graphs. This notion of graph product¹ can be seen as a special case of the notion of direct product as it is used in model theory [27]. There is also a connection of the direct product to category theory [23], which is why this product is sometimes also called the *categorical* graph product.

Let H_1 and H_2 be two graphs. Then the (*direct-, cross-, categorical-*) *product* $H_1 \times H_2$ of H_1 and H_2 is the graph with vertex set $V(H_1) \times V(H_2)$; the pair $((u_1, u_2), (v_1, v_2))$ is in $E(H_1 \times H_2)$ if $(u_1, v_1) \in E(H_1)$ and $(u_2, v_2) \in E(H_2)$. Note that the product is associative and commutative, and we therefore do not have to specify the order of multiplication when multiplying more than two graphs. The n -th *power* H^n of a graph H is inductively defined as follows. H^1 is by definition H . If H^i is already defined, then H^{i+1} is $H^i \times H$.

Definition 13. *A homomorphism from H^k to H , for $k \geq 1$, is called an (k -ary) polymorphism of H .*

In other words, a mapping from $V(H)^k$ to $V(H)$ is a polymorphism of H iff $(f(u_1, \dots, u_k), f(v_1, \dots, v_k)) \in E(H)$ whenever $(u_1, v_1), \dots, (u_k, v_k)$ are arcs in $E(H)$. Note that any graph H has all *projections* as polymorphisms, i.e., all mappings $p : V(H)^k \rightarrow V(H)$ that satisfy for some i the equation $p(x_1, \dots, x_k) = x_i$ for all $x_1, \dots, x_k \in V(H)$. An operation $f : V(H)^k \rightarrow V(H)$ is called *idempotent* if $f(x, \dots, x) = x$ for all $x \in V(H)$.

We now show that for almost all digraphs H the H -coloring problem is NP-complete. Essentially, this follows from the fact that for a random digraph H is *projective*.

Definition 14. *A digraph H is called projective if every idempotent polymorphism is a projection.*

This was shown for undirected graphs H in [32], and the proof for directed graphs H is analogous.

Theorem 15 (of [32]). *A random digraph is with high probability projective.*

Corollary 16. *With high probability all polymorphisms of a random digraph are projections.*

Proof. Let f be an arbitrary polymorphism of H , and consider the endomorphism $f(x, \dots, x)$ of H . By Theorem 12, whp. $g(x) = x$ for all $x \in V(H)$, and hence f is whp. idempotent. Theorem 15 then implies the claim. \square

Theorem 17. *A random digraph H has with high probability an NP-complete H -coloring problem.*

Proof. Let $n = |V(H)|$. We will show that whp. the following reduction of the K_n -coloring problem to the H -coloring problem is correct. Let G be an instance of the K_n -coloring problem. Let m be $\binom{n}{2}$, and let $\{p_1, q_1\}, \dots, \{p_m, q_m\}$ be some enumeration of all two-element subsets of $V(H)$. Replace each edge $(u, v) \in E(G)$ by a copy of H^m as follows: we remove the

¹We want to warn the reader that there are several other notions of graph products that have been studied.

edge (u, v) , create a copy of H^m , and identify the vertex (p_1, \dots, p_m) of this copy with u and (q_1, \dots, q_m) with v . The size of the resulting graph G' is clearly linear in the size of G (since H is fixed). We claim that G' homomorphically maps to H if and only if G homomorphically maps to K_n .

Clearly, if G is n -colorable, then the coloring can be extended to all copies of H^m independently as follows. Let c be the homomorphism from G to K_n , and let (x, y) be an edge of G . Since $c(x) \neq c(y)$, there exists an index i such that $c(x) = p_i$ and $c(y) = q_i$. We can then extend c to the copy of H^m that replaced the edge (x, y) by defining $c(x_1, \dots, x_m) = x_i$. Doing this for all edges (x, y) of G clearly yields a homomorphism from G' to H .

Now, assume that there is a homomorphism f from G' to H . We claim that for any edge (x, y) of G we have $f(x) \neq f(y)$. Suppose for contradiction that there is an edge (x, y) of G such that $f(x) = f(y)$, and consider the restriction f' of f to the copy of H^m that replaced the edge (x, y) in G' . The mapping f' is not a projection, because the pair $((p_1, \dots, p_m), (q_1, \dots, q_m))$ is not mapped to (p_i, q_i) , for any $i \leq m$ (since $p_i \neq q_i$). But since f' is an m -ary polymorphism of H , Corollary 16 states that whp. f is a projection, in which case we have found a contradiction. Therefore, the presented reduction is whp. correct, and we have shown the theorem. \square

Exercises.

24. Show that $(\vec{C}_3)^2 = \vec{C}_3 + \vec{C}_3 + \vec{C}_3$.
25. Let G_1, G_2, H be digraphs. Then the following holds.
 - $G_1 \rightarrow H$ and $G_2 \rightarrow H$ implies that $G_1 + G_2 \rightarrow H$.
 - $H \rightarrow G_1$ and $H \rightarrow G_2$ implies that $H \rightarrow G_1 \times G_2$.
26. Let G and H be digraphs. Prove that $G \times H$ has a directed cycle if and only if both G and H have a directed cycle.
27. Show that the homomorphism order on the set of all digraphs forms a lattice; i.e., show that for any two digraphs G and G'
 - there exist a digraph U such that G and G' homomorphically map to U and such that U homomorphically maps to every graph H with the property that both G and G' homomorphically map to H .
 - there exists a digraph L that homomorphically maps to G and to G' such that any graph H that homomorphically maps both to G and to G' also maps to L .
28. Let H be a digraph with m arcs. Show that $P(H)$ homomorphically maps to H if and only if H has an m -ary polymorphism f satisfying $f(x_1, \dots, x_m) = f(y_1, \dots, y_m)$ whenever $\{x_1, \dots, x_m\} = \{y_1, \dots, y_m\}$.

Solution: Suppose first that g is a homomorphism from $P(H)$ to H . Let f be defined by $f(x_1, \dots, x_m) = g(\{x_1, \dots, x_m\})$. If $(x_1, y_1), \dots, (x_m, y_m) \in E(H)$, then $\{x_1, \dots, x_m\}$ is adjacent to $\{y_1, \dots, y_m\}$ in $P(H)$, and hence $(f(x_1, \dots, x_m), f(y_1, \dots, y_m)) \in E(H)$.

Conversely, suppose that f is a polymorphism as described. Let $g : 2^{V(H)} \rightarrow V(H)$ be $g(\{x_1, \dots, x_m\}) = f(x_1, \dots, x_m)$, which is well-defined by the properties of f . Let $(U, V) \in E(P(H))$, and let x_1, \dots, x_p be an enumeration of the elements of U , and

y_1, \dots, y_q be an enumeration of the elements of V . The properties of $P(H)$ imply that there are $y'_1, \dots, y'_p \in V$ and $x'_1, \dots, x'_q \in U$ such that $(x_1, y'_1), \dots, (x_p, \dots, y'_p) \in E(H)$ and $(x'_1, y_1), \dots, (x'_p, \dots, y_q) \in E(H)$. Therefore,

$$g(U) = g(\{x_1, \dots, x_p\}) = f(x_1, \dots, x_p, x'_1, \dots, x'_q, x_1, \dots, x_1)$$

is adjacent to

$$g(V) = g(\{y_1, \dots, y_q\}) = f(y'_1, \dots, y'_p, y_1, \dots, y_q, y'_1, \dots, y'_1).$$

29. Suppose the G -coloring and the H -coloring problem, for two digraphs G and H , can be solved in polynomial time. Show that the $(G \times H)$ -coloring and the $(G + H)$ -coloring problems can be solved in polynomial time as well.

2.8 The Path-consistency Procedure

The path-consistency procedure is a well-studied generalization of the arc-consistency procedure. Again, it has been discovered first more generally for constraint satisfaction problems; we come back to this algorithm in Section 3, where we give a unified treatment of various forms of consistency and the corresponding algorithms. Applied to the special case of the H -coloring problem, the path-consistency procedure is also known as the pair-consistency check algorithm in the graph theory literature.

Many H -coloring problems that can not be solved by the arc-consistency procedure can still be solved in polynomial time by the path-consistency procedure. The simplest examples are $H = K_2$ (see Exercise 10) and $H = \vec{C}_3$ (see Exercise 12). The idea is to maintain a list of pairs from $V(H)^2$ for each pair in $V(G)^2$ (similarly to the arc-consistency procedure, where we maintained a list of vertices from $V(H)$ for each vertex in $V(G)$). We successively remove pairs from these lists when the pairs can be excluded *locally*. The full procedure can be found in Figure 3. The greater power of the path-consistency procedure comes at the price of a larger worst-case running time: while we have seen a linear-time implementation of the path-consistency procedure, the best known implementations of the path-consistency procedure require cubic time in the size of the input (see Exercise ??).

If we modify the path-consistency procedure as presented in Figure 3 by maintaining lists $L(x, x)$, and also process triples x, y, z that are not pairwise distinct, we obtain the so-called *strong path-consistency procedure*. This procedure is at least as strong as the arc-consistency procedure, because the lists $L(x, x)$ and the rules of the strong path-consistency procedure for $x = y$ simulate the rules of the arc-consistency procedure. In Subsection 2.9 we will see many examples of digraphs H where the path-consistency algorithm solves the H -coloring problem, but the arc-consistency algorithm does not.

The k -consistency procedure. The path-consistency procedure can be generalized further to the k -consistency procedure. In fact, arc- and path-consistency procedure are just the special case for $k = 1$ and $k = 2$, respectively. In other words, the path-consistency procedure is the 3-consistency procedure and the arc-consistency procedure is the 2-consistency procedure. Here, the idea is to maintain sets of k -tuples from $V(H)^k$ for each k -tuple from $V(G)^k$, and to successively remove tuples by local inference. It is straightforward to generalize also the details of the path-consistency procedure.

<pre> PC(G) Input: a finite digraph G. Data structure: a list $L(x, y)$ of pairs from $V(H)^2$ for each pair of distinct vertices $(x, y) \in V(G)^2$. For all $(x, y) \in V(G)^2$ If $(x, y) \in E(G)$ then $L(x, y) := E(H)$ else $L(x, y) := V(H)^2$ Do For all distinct vertices $x, y, z \in V(G)$: For each $(u, v) \in L(x, y)$: If there is no $w \in V(H)$ such that $(u, w) \in L(x, z)$ and $(w, v) \in L(z, y)$ then Remove (u, v) from $L(x, y)$ If $L(x, y)$ is empty then reject Loop until no list changes </pre>

Figure 3: The path-consistency procedure for H -coloring

For fixed H and fixed k , the running time of the k -consistency procedure is still polynomial in the size of G . But the dependency of the running time on k is clearly exponential. It turns out that the k -consistency procedure requires too much time for being useful in practical applications.

The case $k = 3$ (i.e., the path-consistency procedure) is of particular theoretical importance, as demonstrated by the following research question, which is open.

Question 3. *Is there an H -coloring problem that can be solved with the k -consistency procedure, but not with the path-consistency procedure?*

In other words, we do not know whether the k -consistency procedure is more powerful than the path-consistency procedure concerning the question whether they can solve the H -coloring problem for some H . In general, the question whether the path-consistency procedure solves the H -coloring problem can be very difficult.

Question 4. *Is it algorithmically decidable whether the H -coloring problem for a given digraph H can be solved with the path-consistency procedure?*

Exercises

30. Show that the path-consistency procedure for the H -coloring problem can (for fixed H) be implemented such that the worst-case running time is cubic in the size of the input graph. (Hint: use a worklist as in AC-3.)

2.9 Majority Operations

In this section, we present a powerful criterion that shows that for certain graphs H the path-consistency procedure solves the H -coloring problem. Again, this condition was first discovered in more general form by Feder and Vardi [18]; it subsumes many criteria that were studied in Artificial Intelligence and in graph theory before.

Definition 18. Let D be a set. A function f from D^3 to D is called a majority function if f satisfies the following equations, for all $x, y \in D$:

$$f(x, x, y) = f(x, y, x) = f(y, x, x) = x$$

As an example, let D be $\{1, \dots, n\}$, and consider the ternary *median* operation, which is defined as follows. Let x, y, z be three elements from D . Suppose that $\{x, y, z\} = \{a, b, c\}$, where $a \leq b \leq c$. Then $\text{median}(x, y, z)$ is defined to be b .

Theorem 19 (of [18]). *Let H be a finite digraph. If H has a polymorphism that is a majority function, then the H -coloring problem can be solved in polynomial time (by the path-consistency procedure).*

For the proof of Theorem 19 we need the following lemma.

Lemma 20. *Let f be a k -ary polymorphism of a digraph H . Let G be an instance of the H -coloring problem, and $L(x, y)$ be the list computed by the path-consistency procedure for $x, y \in V(G)$. Then f preserves the lists, i.e., if $(u_1, v_1), \dots, (u_k, v_k) \in L(x, y)$, then $(f(u_1, \dots, u_k), f(v_1, \dots, v_k)) \in L(x, y)$.*

Proof. The proof is by induction of the number of times the inner loop of the path-consistency procedure is executed. At the beginning of the procedure, the lists are preserved by f , because f is a polymorphism of H . Now, suppose that we are at some stage during the execution of the path-consistency algorithm where the lists L are preserved by f . We show that after executing the innermost loop once again, the lists L' are still preserved by f . Let $x, y, z \in V(G)$, and $(u_1, v_1), \dots, (u_k, v_k) \in L'(x, y)$; since these pairs are then also in $L(x, y)$, and by inductive assumption, there are $w_1, \dots, w_k \in V(H)$ such that $(u_1, w_1), \dots, (u_k, w_k) \in L(x, z)$ and $(w_1, v_1), \dots, (w_k, v_k) \in L(z, y)$. Thus, $(f(u_1, \dots, u_k), f(w_1, \dots, w_k)) \in L(x, z)$ and $(f(w_1, \dots, w_k), f(v_1, \dots, v_k)) \in L(z, y)$. This implies that the path-consistency procedure does not remove $(f(u_1, \dots, u_k), f(v_1, \dots, v_k))$ from $L(x, y)$, hence this pair is in $L'(x, y)$. \square

of Theorem 19. Suppose that H has a majority polymorphism $f : V(H)^3 \rightarrow V(H)$, and let G be an instance of the H -coloring problem. Clearly, if the path-consistency procedure derives the empty list for some pair (u, v) from $V(G)^2$, then there is no homomorphism from G to H .

Now suppose that after running the path-consistency procedure on G the list $L(x, y)$ is non-empty for all pairs (x, y) from $V(G)^2$. We have to show that there exists a homomorphism from G to H . We say that a homomorphism h from an induced subgraph G' of G to H preserves the lists iff $(h(x), h(y)) \in L(x, y)$ for all $x, y \in V(G')$. The proof shows by induction on i that any homomorphism from a subgraph of G on i vertices that preserves the lists can be extended to any other vertex in G such that the resulting mapping is a homomorphism to H that preserves the lists.

For the base case of the induction, observe that for all vertices $x_1, x_2, x_3 \in V(G)$ every mapping h from $\{x_1, x_2\}$ to $V(H)$ such that $(h(x_1), h(x_2)) \in L(x_1, x_2)$ can be extended to x_3 such that $(h(x_1), h(x_3)) \in L(x_1, x_3)$ and $(h(x_3), h(x_2)) \in L(x_3, x_2)$ (and hence preserves the lists), because otherwise the path-consistency procedure would have removed $(h(x_1), h(x_2))$ from $L(x_1, x_2)$.

For the inductive step, let h' be any homomorphism from a subgraph G' of G on $i \geq 3$ vertices to H that preserves the lists, and let x be any vertex of G not in G' . Let $x_1, x_2,$

and x_3 be some vertices of G' , and h'_j be the restriction of h' to $G' \setminus x_j$, for $1 \leq j \leq 3$. By inductive assumption, h'_j can be extended to x such that the resulting mapping h_j is a homomorphism to H that preserves the lists. We claim that the extension h of h' that maps x to $f(h_1(x), h_2(x), h_3(x))$ is a homomorphism to H that preserves the lists.

For all $y \in V(G')$, we have to show that $(h(x), h(y)) \in L(x, y)$ (and that $(h(y), h(x)) \in L(y, x)$, which can be shown analogously). If $y \notin \{x_1, x_2, x_3\}$, then $h(y) = h'(y) = f(h'(y), h'(y), h'(y)) = f(h_1(y), h_2(y), h_3(y))$, by the properties of f . Since $(h_1(x), h_1(y)), (h_2(x), h_2(y)), (h_3(x), h_3(y))$ are in $L(x, y)$, and since f preserves $L(x, y)$ by Lemma 20, we have $(h(x), h(y)) \in L(x, y)$, and are done in this case.

Clearly, y can be equal to at most one of $\{x_1, x_2, x_3\}$. Suppose that $y = x_1$ (the other two cases are analogous). There must be a vertex $v \in V(H)$ such that $(h_1(x), v) \in L(x, y)$ (otherwise the path-consistency procedure would have removed $(h_1(x), h_1(x_2))$ from $L(x, x_2)$). By the properties of f , we have $h(y) = h'(y) = f(v, h'(y), h'(y)) = f(v, h_2(y), h_3(y))$. Because $(h_1(x), v), (h_2(x), h_2(y)), (h_3(x), h_3(y))$ are in $L(x, y)$, Lemma 20 implies that $(h(x), h(y)) = (f(h_1(x), h_2(x), h_3(x)), f(v, h_2(y), h_3(y)))$ is in $L(x, y)$, and we are done.

Therefore, for $i = |V(G)|$, we obtain a homomorphism from G to H . \square

As an example, let H be the transitive tournament on n vertices, T_n . Suppose the vertices of T_n are the first natural numbers, $\{1, \dots, n\}$, and $(u, v) \in E(T_n)$ iff $u < v$. Then the median operation is a polymorphism of T_n , because if $u_1 < v_1, u_2 < v_2$, and $u_3 < v_3$, then clearly $\text{median}(u_1, u_2, u_3) < \text{median}(v_1, v_2, v_3)$. This yields a new proof that the H -coloring problem for $H = T_n$ is tractable.

Corollary 21. *The path-consistency procedure solves the H -coloring problem for $H = T_n$.*

Another class of examples of graphs having a majority polymorphism are *unbalanced cycles*, i.e., orientations of C_n that do not homomorphically map to a directed path [16]. We only prove a weaker result here.

Proposition 22. *Directed cycles have a majority polymorphism.*

Proof. Let f be the ternary operation that maps u, v, w to the u if u, v, w are pairwise distinct, and otherwise acts as a majority operation. We claim that f is a polymorphism of C_n . Let $(u, u'), (v, v'), (w, w') \in E(C_n)$ be arcs. If u, v, w are all distinct, then u', v', w' are clearly all distinct as well, and hence $(f(u, v, w), f(u', v', w')) = (u, u') \in E(C_n)$. Otherwise, if two elements of u, v, w are equal, say $u = v$, then u' and v' must be equal as well, and hence $(f(u, v, w), f(u', v', w')) = (u, u') \in E(C_n)$. \square

Near-unanimity functions. Majority functions are a special case of so-called *near-unanimity functions*. A function f from D^k to D is called a (k -ary) *near unanimity function* (short, an *nuf*) if f satisfies the following equations, for all $x, y \in D$:

$$f(x, \dots, x, y) = f(x, \dots, y, x) = \dots = f(y, x, \dots, x) = x$$

Obviously, a majority is a ternary near-unanimity function. Similarly as in Theorem 19 it can be shown that the existence of a k -ary nuf polymorphism of H implies that the k -consistency procedure (which was informally introduced in Subsection 2.8) solves the H -coloring problem.

Exercises.

31. A *quasi majority function* is a function from D^3 to D satisfying $f(x, x, y) = f(x, y, x) = f(y, x, x) = f(x, x, x)$ for all $x, y \in D$. Use Theorem 3 to show that an undirected graph H has an H -coloring problem that can be solved in polynomial time if H has a quasi majority polymorphism, and is NP-complete otherwise.
32. Show that every oriented path has a majority operation.
33. There is only one unbalanced cycle H on four vertices that is a core and not the directed cycle (we have seen this graph already in Exercise 23). Show that for this graph H the H -coloring problem can be solved by the path-consistency procedure.
34. Modify the path-consistency procedure such that it can deal with instances of the *pre-colored* H -coloring problem. Show that if H has a majority operation, then the modified path-consistency procedure solves the pre-colored H -coloring problem.
35. An *interval graph* H is an (undirected) graph $H = (V, E)$ such that there is an interval I_x of the real numbers for each $x \in V$, and $(x, y) \in E$ iff I_x and I_y have a non-empty intersection. Note that with this definition interval graphs are necessarily *reflexive*, i.e., $(x, x) \in E$. Show that the pre-colored H -coloring problem for interval graphs H can be solved in polynomial time. Hint: use the modified path-consistency procedure in Exercise 34.

3 Constraint Satisfaction Problems

Informally, a constraint satisfaction problem is a computational problem where the input consists of a set of variables and a set of constraints imposed on these variables, and where the task is to decide whether there is assignment of values to the variables such that all the constraints are satisfied. Such problems occur naturally in many areas in computer science, for example in Artificial Intelligence, computational linguistics, computational biology, type inference for programming languages, and scheduling. Before we further restrict and formalize the notion of constraint satisfaction problems, we present well-known examples that fit to the above informal description of constraint satisfaction problems.

Example 1: Sudokus. A Sudoku is a number placement puzzle. The objective is to fill a 9×9 grid so that each column, each row, and each of the nine 3×3 subgrids contains all the digits from 1 to 9. The puzzle setter provides a partially completed grid, such that there is exactly one way to fill the remaining entries.

Sudokus are CSPs: the fields of the grid are the variables, the digits are the values, and there are two types of constraints: constraints that pre-describe that certain fields have to be filled with certain digits, and the constraints imposed on rows, columns, and subgrids which say that out of nine fields each digit must appear exactly once.

The strategies employed by humans to solve Sudokus are typical examples of constraint propagation techniques as introduced in Section 4.

Example 2: H -coloring problems. H -coloring problems are constraint satisfaction problems, where the variables are the vertices of the input graph G , and the constraints are the arcs of G . The values that the variables may take are the vertices of H , and an assignment of values to variables satisfies all constraints, if the assignment preserves all the edges, i.e., is a graph homomorphism.

Example 3: Linear Inequalities. Linear program feasibility can be modelled as a constraint satisfaction problem: here the constraints are linear inequalities of the form $a_1x_1 + \dots + a_nx_k \leq b$, where $a_1, \dots, a_k, b \in \mathbb{R}$ and x_1, \dots, x_k are variables, and we have to assign real values to the variables such that all the inequalities are satisfied. In other words, we are looking for a point in n -dimensional space (n is the total number of variables) that lies inside a polyhedron given as the intersection of half-spaces in n -dimensional space.

3.1 Relational Structures

Our first formalization of the constraint satisfaction problem generalizes the H -coloring problem to *relational structures*. Another formulation of the CSP, which is essentially equivalent but provides another perspective and uses different terminology, can be found in Subsection 3.6.

A *relational signature* τ is a set of *relation symbols* R_i , each associated with an *arity* k_i . In this text, all relational signatures will be finite, and we therefore sometimes assume that they are finite without mentioning this. A (*relational*) *structure* Γ over relational signature τ (also called τ -*structure*) is a set D_Γ (the *domain*) together with a relation $R_i \subseteq D_\Gamma^{k_i}$ for each relation symbol of arity k_i . If necessary, we write R_i^Γ to indicate that we are talking about the relation R_i belonging to the structure Γ . For simplicity, we denote both a relation symbol

and its corresponding relation with the same symbol. For a τ -structure Γ and $R \in \tau$ it will also be convenient to say that $R(u_1, \dots, u_k)$ holds in Γ iff $(u_1, \dots, u_k) \in R$.

Let Γ and Δ be τ -structures. A *homomorphism* from Γ to Δ is a function f from D_Γ to D_Δ such that for each n -ary relation symbol in τ and each n -tuple \bar{a} , if $\bar{a} \in R^\Gamma$, then $(f(a_1), \dots, f(a_n)) \in R^\Delta$. In this case we say that the map f *preserves* the relation R . If f also preserves the complements of all relations, and is injective, we say that f is an *embedding*. Similarly as the notion of homomorphisms between relational structures, most of the other concepts introduced in Section 2 generalize from digraphs to relational structures; see Exercise 52.

If we add relations to a given τ -structure Γ , then the resulting structure Γ' with a larger signature $\tau' \supset \tau$ is called a τ' -*expansion* of Γ , and Γ is called a τ -*reduct* of Γ' . The concept of *induced substructures* generalizes the concept of induced subgraphs: If Γ and Γ' are structures of the same signature, with $D(\Gamma) \subseteq D(\Gamma')$, and the inclusion map is an embedding of Γ into Γ' , then we say that Γ is an *induced substructure* of Γ' , and we write $\Gamma = \Gamma'[D(\Gamma)]$. We also write $\Gamma - x$ for $\Gamma[D(\Gamma) \setminus \{x\}]$.

Exercises.

36. Generalize the concept of the set graph, cores, products, polymorphisms, and quasi near-unanimity operations to relational structures.

3.2 The Constraint Satisfaction Problem

Many constraint satisfaction problems can be modeled as homomorphism problems. Let Γ be a (finite or infinite) structure with a (finite) relational signature τ . Then the *constraint satisfaction problem (CSP)* for Γ is the following computational problem.

Problem 1. **CSP(Γ)**

INSTANCE: A finite τ -structure S .

QUESTION: Is there a homomorphism from S to Γ ?

The structure Γ is called the *template* of the constraint satisfaction problem CSP(Γ). Sometimes, Γ is also called the *constraint language* of the constraint satisfaction problem CSP(Γ). Note that we did not assume that the domain of Γ is finite; in fact, most examples and most applications mentioned in this text can only be formulated with infinite domain Γ .

If S homomorphically maps to Γ , we also say that S is *consistent* or *satisfiable* (there are different names for the same concept because of the different fields of applications where CSPs occur, each field using a different name). The homomorphism f from S to Γ is also called a *solution*. Note that even though it is in general not clear how solutions to a given instance of the CSP are represented, since the domain of Γ might be infinite. However, also for infinite-domain Γ the problem CSP(Γ) is a well-defined computational problem.

Also note that we did assume that the signature of Γ is finite. For finite signatures, we can fix any representation of the relations in the finite τ -structure S , and, from a computational complexity perspective, yield the same CSP. For infinite signatures, the complexity of CSP(Γ) might depend on the representation of the relations in S . So, unless stated otherwise, the computational problem CSP(Γ) is only defined for finite relational signatures. Also see the discussion of some examples of CSPs later in this Subsection.

Examples. Clearly, all H -coloring problems are constraint satisfaction problems in this sense. We will now see many additional computational problems that can be cast as $\text{CSP}(\Gamma)$ for appropriate Γ .

There are many problems in Boolean satisfiability that can be cast as CSPs as well. Two well-known problems for formulas in 3SAT that remain NP-complete even if all clauses in the input only contain positive literals are 1-in-3-3SAT and NOT-ALL-EQUAL-3SAT; both problems can be found in the classical list of NP-complete problems by Garey and Johnson [19].

Problem 2. Positive-1-in-3-3SAT

INSTANCE: A propositional 3SAT formula with only positive literals

QUESTION: Is there a Boolean assignment for the variables such that in each clause exactly one literal is true?

Problem 3. Positive-Not-All-Equal-3SAT

INSTANCE: A propositional 3SAT formula with only positive literals

QUESTION: Is there a Boolean assignment for the variables such that in each clause neither all three literals are true nor all three are false

These problems can be formulated as $\text{CSP}(\Gamma)$ for the template

$$\Gamma = (\{0, 1\}, \{(0, 0, 1), (0, 1, 0), (1, 0, 0)\}) ,$$

or respectively the template

$$\Gamma = (\{0, 1\}, \{0, 1\}^3 \setminus \{(0, 0, 0), (1, 1, 1)\}) .$$

These problems can also be formulated as CSPs if we do not impose the restriction that all literals are positive. The idea is to use a different ternary relation for each of the eight ways how the variables in a clause with three literals might be negated. In this way, we can also model the classical problem of 3SAT as a CSP. E.g., for clauses of the type $x \vee y \vee \neg z$ in the 3SAT problem we use the ternary relation $R^{++-} = \{0, 1\}^3 \setminus \{(0, 0, 1)\}$.

We now discuss problems that require infinite domain templates.

Problem 4. Digraph Acyclicity

INSTANCE: A digraph G .

QUESTION: Is G acyclic?

To formulate this problem as a CSP we use the dense linear order $(\mathbb{Q}, <)$ on the rational numbers as a template. Since $<$ is a binary relation, we can view this relation as the set of arcs of an infinite digraph. Clearly, if G contains a directed cycle, there is no mapping $h : V(G) \rightarrow \mathbb{Q}$ such that for every arc (u, v) in G we have $f(u) < f(v)$. Conversely, if G is acyclic, it is not hard to see that then there exists a homomorphism from G to $(\mathbb{Q}, <)$.

An extension of this problem is the CSP for the so-called *point algebra* in temporal reasoning; the template here is (\mathbb{Q}, \leq, \neq) . Note that for any pair (u, v) from \mathbb{Q}^2 , exactly one out of the following three possibilities applies: $u < v$, $u > v$, and $u = v$. Hence, there are eight binary relations that can be obtained by boolean combinations of these three relations:

$$u \leq v, u \geq v, u \neq v, u < v, u > v, u = v, \text{false}, \text{true} .$$

All of these relations can be defined by appropriately intersecting and flipping arguments of the two relations \leq and \neq that we have included in our template.

The *Betweenness problem* is an important NP-complete problem, and again listed in the book of Garey and Johnson [19]. It is again a constraint satisfaction problem over the domain \mathbb{Q} .

Problem 5. Betweenness

INSTANCE: A finite relational structure $(V, Betw)$ where $Betw$ is a ternary relation

QUESTION: Is there a mapping $\alpha : V \rightarrow \mathbb{Q}$ such that for each triple $(x, y, z) \in Betw$ we have $\alpha(x) < \alpha(y) < \alpha(z)$ or $\alpha(z) < \alpha(y) < \alpha(x)$?

The following problem will be an important illustration for results presented in Section 4 and Section 5.

Problem 6. Min-Ordering

INSTANCE: A finite relational structure (V, R^{min}) where R^{min} is a ternary relation

QUESTION: Is there a mapping $\alpha : V \rightarrow \mathbb{Q}$ such that for each triple $(x, y, z) \in R^{min}$ we have $\alpha(x) > \alpha(y)$ or $\alpha(x) > \alpha(z)$?

This problem can be formulated as $CSP(\mathbb{Q}, R^{min})$ where $R^{min} = \{(u, v, w) \in \mathbb{Q}^3 \mid u > v \vee u > w\}$.

The list of natural computational problems that have natural formalizations as a CSP is rather long, and our selection of the following problem is quite arbitrary (the author simply likes it!).

Problem 7. Circular-Embedding

INSTANCE: A digraph G

QUESTION: Is there a mapping $\alpha : V(G) \rightarrow \mathbb{R}^2$ such that for every arc $(u, v) \in E(G)$ the point $(0, 0)$ lies to the left of the oriented line passing through $\alpha(u), \alpha(v)$?

The next examples demonstrate the limits of the notion of constraint satisfaction problems studied in this section.

Sudokus. Sudokus are CSPs that were mentioned already in Subsection 3.2. Every Sudoku can be posed as an instance of $CSP(\Gamma)$ for $\Gamma = \{1, \dots, 9\}, R, P_1, \dots, P_9$, where $R = \{(t_1, \dots, t_9) \mid |\{t_1, \dots, t_9\}| = 9\}$, and $P_i = \{i\}$. However, $CSP(\Gamma)$ has many instances that do not correspond to Sudokus, simply because every Sudoku has 81 fields, whereas instances of $CSP(\Gamma)$ might have arbitrarily many variables. Moreover, in Sudokus the constraints concerning the relation R are placed in a very special pre-described way. Hence, $CSP(\Gamma)$ is a more general problem, and does not precisely capture the problem of solving Sudokus. However, since there is only a finite number of Sudokus, solving Sudokus is trivial from a theoretical perspective anyway.

Linear Inequalities. Linear Inequalities, i.e., the problem to decide the feasibility of a linear program have been mentioned above as an example of a constraint satisfaction problem in Subsection 3.2. If we want to formulate this problem as $CSP(\Gamma)$ for an appropriate relational structure Γ , we have to use relational structures with an *infinite signature*. For each n and each $a_1, \dots, a_n, b \in \mathbb{R}$ we need an n -ary relation $\{(x_1, \dots, x_n) \mid a_1x_1 + \dots + a_nx_n \leq b\}$.

Horn Satisfiability. There are also finite-domain CSPs where infinite signatures are necessary to precisely formulate the CSP as $\text{CSP}(\Gamma)$. Consider the problem of Horn-SAT. The instance here consists of a set of Horn clauses (there is no restriction on the size of the Horn clauses in the input), and the task is to find a truth assignment of the variables that satisfies all the clauses. To formulate this problem as $\text{CSP}(\Gamma)$, we use for Γ the two-element structure that contains a k -ary relation for each Horn clause with k variables.

In this text we mostly work with finite signatures, and we assume finite signatures throughout unless stated otherwise.

Exercises.

37. Show that $\text{CSP}(\mathbb{Q}, \leq, \neq)$ can not be formulated as $\text{CSP}(\Gamma)$ for a relational structure Γ with a finite domain.

3.3 The Fundamental Lemma

$\text{CSP}(\Gamma)$ can also be considered to be a set – the set of all finite structures S that homomorphically map to Γ .

Definition 23. *We say that a set \mathcal{C} of relational structures is closed under disjoint unions iff whenever $A, B \in \mathcal{C}$ then $A + B \in \mathcal{C}$. We say that \mathcal{C} is closed under inverse homomorphisms if $A \notin \mathcal{C}$ and A homomorphically maps to B , then $B \notin \mathcal{C}$.*

The following is a fundamental lemma for constraint satisfaction problems.

Lemma 24. *Let \mathcal{C} be a class of finite relational structures. Then $\mathcal{C} = \text{CSP}(\Gamma)$ for some relational structure Γ over an infinite domain if and only if \mathcal{C} is closed under disjoint unions and inverse homomorphisms.*

Proof. Clearly, the disjoint union of two satisfiable instance of $\text{CSP}(\Gamma)$ is again satisfiable. Moreover, if A is not satisfiable, and there is a homomorphism from A to B , then B is not satisfiable as well.

For the other direction, suppose that \mathcal{C} is a class of relational structures that is closed under disjoint unions and inverse homomorphisms. Let Γ be the (infinite) disjoint union of all structures in \mathcal{C} . Clearly, for every structure in \mathcal{C} homomorphically maps to Γ . Now suppose for contradiction that there is a structure that homomorphically maps to Γ but is not in \mathcal{C} . Let S be such a structure with a minimal number of vertices. The structure I induced by $V(S)$ in Γ is by assumption on \mathcal{C} not in \mathcal{C} as well, and therefore I must be a disjoint union of several components of Γ . Each component has less vertices than S and hence homomorphically maps to Γ . Since \mathcal{C} is closed under disjoint unions, I must be in \mathcal{C} , a contradiction. \square

Examples. For the following computational problems it can be verified that the positive instances are closed under disjoint union and inverse homomorphisms (see Exercise 38). Hence, Lemma 24 shows that they can be formulated as $\text{CSP}(\Gamma)$ for some relational structure Γ .

Problem 8. **Triangle-Freeness**

INSTANCE: An undirected graph G

QUESTION: Is G triangle-free?

Problem 9. Acyclic-Colorings

INSTANCE: A digraph G

QUESTION: Is there a partition $V = V_1 \uplus V_2$ of the vertices V of G such that both $G[V_1]$ and $G[V_2]$ is acyclic?

Problem 10. No-Mono-Tri

INSTANCE: An undirected graph G

QUESTION: Is there a partition $V = V_1 \uplus V_2$ of the vertices V of G such that both $G[V_1]$ and $G[V_2]$ are triangle-free?

In all three cases it can be verified that the problem can be formulated as $\text{CSP}(\Gamma)$ for a relational structure Γ with a finite domain (see Exercise 39).

Exercises.

- 38. Show that the problems *Triangle-freeness*, *Acyclic-Colorings*, and *No-Mono-Tri* are closed under disjoint unions and inverse homomorphisms.
- 39. Show that the problems *Triangle-freeness*, *Acyclic-Colorings*, and *No-Mono-Tri* can not be formulated as $\text{CSP}(\Gamma)$ for a relational structure Γ with a finite domain.

3.4 Entailment, Finding a Solution, Finding all Solutions

In several applications that can be modeled using constraints, the constraint satisfaction problem is not the most interesting computational problem. However, it turns out that most computational questions related to constraints can be reduced, under fairly weak assumptions, to the constraint satisfaction problem.

To be completed.

3.5 Computational Complexity of CSPs.

To be completed. INCLUDE THAT EVERY PROBLEM IS EQUIV TO A CSP.

3.6 CSPs, Logic, Databases

A first-order formula ϕ is called *primitive positive* if it is of the form

$$\exists z_1, \dots, z_m. \psi_1 \wedge \dots \wedge \psi_l$$

where ψ_1, \dots, ψ_l are *atomic* formulas, i.e., formulas of the form $R(y_1, \dots, y_s)$.

Let Γ be a structure with the relational signature τ , and suppose that ϕ is a τ -formula, that is, all relation symbols that appear in ϕ are from τ . Then for every ordering x_1, \dots, x_k of the free variables the formula ϕ gives rise to a k -ary relation $R = R(\phi(x_1, \dots, x_k))$, defined as

$$R = \{(t_1, \dots, t_k) \in D(\Gamma) \mid \phi(t_1, \dots, t_k) \text{ is true in } \Gamma\} .$$

In this case, we say that R has the *primitive positive definition* $\phi(x_1, \dots, x_k)$, and that R is *primitive positive definable*.

As usual, a formula without free variables is called a sentence. Note that the $\text{CSP}(\Gamma)$ has the following equivalent formulation in logic. The input is a primitive positive sentence Φ , and the question is whether Φ is true in Γ . The conjuncts of Φ are also called the *constraints* of the input. It is obvious that this is essentially the same problem as $\text{CSP}(\Gamma)$ as it was defined in Subsection 3.2.

Lemma 25. *Let $\Gamma = (\mathbb{Q}, R_1, \dots, R_k)$ be a relational structure, and let R be a relation that has a primitive positive definition in Γ . Then $\text{CSP}(\Gamma)$ and $\text{CSP}((\mathbb{Q}, R, R_1, R_2, \dots))$ are linear-time equivalent.*

Proof. It is clear that $\text{CSP}(\Gamma)$ reduces to the new problem. So suppose that Φ is an instance of $\text{CSP}(\Gamma)$ (Φ is no considered as a primitive positive sentence). Replace each conjunct $R(x_1, \dots, x_l)$ of Φ by its primitive positive definition $\psi(x_1, \dots, x_l)$. Finally, move all quantifiers to the front, such that the resulting formula Φ' is in prenex normal form. It is straightforward to verify that Γ satisfies Φ if and only if $(\mathbb{Q}, R, R_1, R_2, \dots)$ satisfies Φ' , and it is also clear that Φ' can be constructed in linear time in the representation size of Φ . \square

We illustrate how to use this lemma and prove that C_n -coloring is NP-hard for $n \geq 3$. We know that K_n -coloring is NP-complete for $n \geq 3$, so by Lemma 25 (and because H -coloring problems are CSPs) it suffices to find a primitive positive definition of the binary edge relation E' of K_n in C_n . Let E be the edge-relation of C_n . Then the primitive positive definition of $E'(x, y)$ is as follows.

$$\begin{aligned} \exists p_1, p_2, p_3, q_1, q_2. & E(x, p_1) \wedge E(p_1, p_2) \wedge E(p_2, p_3) \wedge E(p_3, y) \\ & \wedge E(x, p_1) \wedge E(p_1, p_2) \wedge E(p_2, y) \end{aligned}$$

This formula simply says that there is an even path of length ≤ 4 and an odd path of length ≤ 3 from x to y . This is the case for two vertices u, v in C_5 iff $x \neq y$.

CSPs and Conjunctive Queries. There is a strong connection between the constraint satisfaction problem and database theory. In one of the classical models in database theory, a *database* is simply a relational structure. Figure 4 shows an example of a database with one ternary relation.

Student	Course	Lecturer
Peter Muskl	Graphs and Algorithms 1	Gerhard Langsam
Sibylle Meier	Graphs and Algorithms 2	Ludwig Schnell
...

Figure 4: A table in a (relational) database.

A conjunctive query is a primitive positive formula, together with an enumeration x_1, \dots, x_k of the free variables. Here are examples of questions to the database from Figure 4 that can be formulated with conjunctive queries:

- List all pairs of persons (A, B) s.t. A was a student of B in a course and B was a student of A in a course.

- List all pairs of students that have attended the same course.

An important problem in database theory is the problem of *conjunctive query containment*. If we have two queries Q_1 and Q_2 with k free variables x_1, \dots, x_k , it might be that they define the same k -ary relation in all relational structures Γ ; in this case we say that the two queries are *equivalent*. In practice, we would like to decide whether this is the case, because it might be the query Q_1 is simpler to evaluate in large databases. This motivates the question whether there is an algorithm that efficiently decides whether $R(Q_1(x_1, \dots, x_k)) \subseteq R(Q_2(x_1, \dots, x_k))$ for all relational structures Γ . In this case we say that first query is *contained* in the second query, and write $Q_1 \subseteq Q_2$. Clearly, if we can decide query containment, we can also decide query equivalence.

To formalize the connection of this problem with constraint satisfaction, we have to define the *canonical database* S_Q of a query Q , which is a relational structure defined as follows. The vertices of S_Q are the variables (both the free variables and the existentially quantified variables) that occur in Q , and the signature of S_Q is the signature of Q . There is a tuple (v_1, \dots, v_k) in a relation R of S_Q iff Q contains a conjunct $R(v_1, \dots, v_k)$.

Theorem 26 (Chandra,Merlin'77). *For two conjunctive queries Q_1 and Q_2 with free variables x_1, \dots, x_k , we have $Q_1 \subseteq Q_2$ if and only if there is a homomorphism from S_{Q_1} to S_{Q_2} that fixes x_1, \dots, x_k .*

Proof. Suppose first that $Q_1 \subseteq Q_2$. The query Q_1 defines on S_{Q_1} the relation $R_1 = R(Q_1(x_1, \dots, x_k))$. Clearly, R_1 contains the tuple (x_1, \dots, x_k) . By assumption R_1 is contained in the relation $R_2 = R(Q_2(x_1, \dots, x_k))$ defined by Q_2 on the same relational structure S_{Q_1} . In particular, there are values v_1, \dots, v_l for the existentially quantified variables y_1, \dots, y_l in Q_2 that show that $(x_1, \dots, x_k) \in R_2$. It is straightforward to verify that the mapping that assigns v_i to y_i and that fixes x_1, \dots, x_k is a homomorphism from S_{Q_2} to S_{Q_1} . \square

4 Constraint Propagation

Constraint propagation is a very general algorithmic tool in constraint satisfaction. The power of constraint propagation is intensively studied, both in theory, where the main research questions are still open, and in the literature that is more directed towards applications, where efficient constraint propagation algorithms are of central interest.

In Section 2, we have already seen two examples of constraint propagation algorithms, the arc-consistency procedure and the path-consistency procedure for H -coloring. In this section, we will have a broader perspective on these procedures, which allows us to also formulate procedures for constraint satisfaction problems for templates with infinite domains.

The general idea of constraint propagation is to infer new constraints that are implied by given constraints or by constraints that have already been inferred before. If we are able to derive 'false' in that way, we know that the constraint satisfaction problem has no solution. Frequently, it is necessary to formulate the inferred constraints in a different constraint language than the language of the CSP we started with. Formally, to solve $\text{CSP}(\Gamma)$ we might formulate algorithms for $\text{CSP}(\Gamma')$ where Γ' is an expansion of Γ .

4.1 Datalog

NEED AN EXAMPLE. Datalog is the language of logic programs without function symbols, see e.g. [15,29]. Equivalently, Datalog can be seen as conjunctive queries with a recursion mechanism. The definition of Datalog given here will be purely operational; for the standard semantical approach to the evaluation of Datalog programs see [15,29]. A Datalog program is a finite set of Horn clauses, i.e., clauses of the form $\psi :- \phi_1, \dots, \phi_l$, where $l \geq 0$ and where $\psi, \phi_1, \dots, \phi_l$ are atomic formulas. The formula ψ is called the *head* of the rule, and $\phi_1 \wedge \dots \wedge \phi_l$ is called the *body*. For technical reasons, we assume that all variables in the head also occur in the body.

The relation symbols occurring in the head of some clause are called *intentional database predicates* (or *IDBs*), and all other relation symbols in the clauses are called *extensional database predicates* (or *EDBs*). A Datalog program has *width* (l,k) if all IDBs are at most l -ary, and if all rules have at most k distinct variables.

Let Γ be a structure with (finite) relational signature τ . We might use Datalog programs to solve $\text{CSP}(\Gamma)$ as follows. Let Π be a Datalog program whose EDBs are τ , and let σ be the set of IDBs of Π . We assume that there is one distinguished 0-ary intentional relation symbol *false*. Now, suppose we are given an instance A of $\text{CSP}(\Gamma)$. An *evaluation* of Π on A proceeds in steps $i = 0, 1, \dots$. At each step i we maintain a set of literals S^i with extensional or intentional predicates; it always holds that $S^i \subset S^{i+1}$. Each clause of Π is understood as a rule that may derive a new literal (with a relation symbol from L) from the literals in S^i . Initially, we start with the set S^0 containing all the literals for tuples in the instance A . Now suppose that $R_1(x_1^1, \dots, x_{k_1}^1), \dots, R_l(x_1^l, \dots, x_{k_l}^l)$ are literals in S^i , and $R_0(y_1^0, \dots, y_{k_0}^0) :- R_1(y_1^1, \dots, y_{k_1}^1), \dots, R_l(y_1^l, \dots, y_{k_l}^l)$ is a rule from Π , where $y_j^i = y_{j'}^{i'}$ if $x_j^i = x_{j'}^{i'}$. Then $R_0(x_1^0, \dots, x_l^0)$ is the newly derived literal in S^{i+1} , where $x_j^0 = x_{j'}^i$ if and only if $y_j^0 = y_{j'}^i$. The procedure stops if no new literal can be derived. We say that Π is *sound* for $\text{CSP}(\Gamma)$ if S does not homomorphically map to Γ whenever Π derives *false* on S . We say that Π *solves* $\text{CSP}(\Gamma)$ if Π derives *false* on S if and only if S does not homomorphically map to Γ .

Exercises.

40. Formulate the arc-consistency and the path-consistency procedures for the H -coloring problem as Datalog programs.
41. Show that $\text{CSP}((\mathbb{Q}, \leq, \neq))$ can be solved by a $(2, 3)$ -Datalog program.

4.2 Local Consistency

To be completed.

4.3 The Existential Pebble-Game

The existential (l, k) -pebble game is studied in the context of constraint satisfaction to characterize the expressive power of Datalog [12, 18, 29]. In particular, it can be used to show that certain constraint satisfaction problems can *not* be solved by Datalog programs.

The game is played by the players Spoiler and Duplicator on (possibly infinite) structures A and B of the same relational signature. Each player has k pebbles, p_1, \dots, p_k for Spoiler and q_1, \dots, q_k for Duplicator. Spoiler places his pebbles on elements of A , Duplicator her pebbles on elements of B . Initially, no pebbles are placed. In each round of the game Spoiler picks $k-l$ pebbles. If some of these pebbles are already placed on A , then Spoiler removes them from A , and Duplicator responds by removing the corresponding pebbles from B . Spoiler places the $k-l$ pebbles on elements of A , and Duplicator responds by placing the corresponding pebbles on elements of B . Let i_1, \dots, i_m be the indices of the pebbles that are placed on A (and B) after the i -th round. Let a_{i_1}, \dots, a_{i_m} (b_{i_1}, \dots, b_{i_m}) be the elements of A (B) pebbled with the pebbles p_{i_1}, \dots, p_{i_m} (q_{i_1}, \dots, q_{i_m}) after the i -th round. If the partial mapping h from A to B defined by $h(a_{i_j}) = b_{i_j}$, for $1 \leq j \leq m$, is not a homomorphism from $A[\{a_{i_1}, \dots, a_{i_m}\}]$ to B , then the game is over, and Spoiler wins. Duplicator wins if the game continues forever.

We are interested in the situations where Duplicator can always win the game, i.e., where Duplicator has a winning strategy. It turns out that in this case Duplicator can always play *memoryless* in the sense that the decisions of Duplicator are only based on the current position of the pebbles, and not the previous decisions in the game.

Definition 27. A (positional) winning strategy for Duplicator for the existential (l, k) -pebble game on A, B is a non-empty set \mathcal{H} of partial homomorphisms from A to B such that

- \mathcal{H} is closed under restrictions of its members, and
- for all functions h in \mathcal{H} with domain of size $d \leq l$ and for all $a_1, \dots, a_{k-d} \in A$ there is an extension $h' \in \mathcal{H}$ of h such that h' is also defined on a_1, \dots, a_{k-d} .

In the following, let Γ be a τ -structure, either finite, or infinite with finitely many l -ary primitive positive definable relations.

Theorem 28. $\text{CSP}(\Gamma)$ cannot be solved by an (l, k) -Datalog program if and only if there exists a finite τ -structure A that does not homomorphically map to Γ such that Duplicator wins the existential (l, k) -pebble game on (A, Γ) .

Before we prove this result, we have to introduce the notion of *canonical* Datalog programs.

Definition 29. The canonical (l, k) -Datalog program for $\text{CSP}(\Gamma)$ is a Datalog program Π that contains an IDB for every at most l -ary primitive positive definable relation in Γ (by

assumption on Γ , there are only finitely many such relations). The empty 0-ary relation is also denoted by *false*. The EDBs are precisely the relation symbols in τ . The program Π contains a rule $\psi :- \psi_1, \dots, \psi_j$ if the corresponding implication contains at most k variables, is valid in Γ , and ψ is of the form $R(y_1, \dots, y_s)$ for an IDB R and $s \leq l$.

Proof. To be completed. □

The main application of Theorem 28 is to show that certain constraint satisfaction problems can *not* be solved by Datalog. We demonstrate this with the min-ordering problem (see Subsection 3.2), which is $\text{CSP}(\mathbb{Q}, R^{\min})$ where $R^{\min} = \{(x, y, z) \mid x > y \vee x > z\}$. This CSP can be solved in polynomial time (as we will see in Section 5).

Theorem 30. *There is no Datalog program that solves $\text{CSP}((\mathbb{Q}, R^{\min}))$.*

Proof. Let k be an arbitrary number. To apply Theorem 28 we have to construct an inconsistent instance Φ of $\text{CSP}(\mathbb{Q}, R^{\min})$ such that Duplicator wins the existential k -pebble game on Φ .

For this, let G be a 4-regular graph of girth $2k + 1$, i.e., all cycles in G have more than $2k$ vertices. It is known and easy to see that such graphs exist; it is even known that there are such graphs of size exponential in k [28]. Orient the edges in G such that there are exactly two outgoing and two incoming edges for each vertex in G . Since G is 4-regular, there exists an Euler tour for G (see e.g. [13]), which shows that such an orientation exists.

Now we can define our instance Φ of $\text{CSP}(\mathbb{Q}, R^{\min})$ as follows. The variables of Φ are the vertices from G . The instance Φ contains the constraint $R^{\min}(w, u, v)$ iff uw and vw are the two incoming edges at vertex w . We claim that Φ does not have a solution: if there was a solution, some variable w must denote the minimal value. But for every variable w we find a constraint $R^{\min}(w, u, v)$ in Φ , and this constraint is violated since either u or v must be strictly smaller than w .

Consider a connected non-empty subgraph G' of G having at most $2k$ vertices where only one vertex r has no outgoing edges, and where all vertices have either two or no incoming edges. Since G has girth $2k + 1$, G' must be a binary tree with root r . We call G' *dominated*, if all leaves in G' are pebbled.

Duplicator always maintains the property that whenever the root r in a dominated tree is pebbled during the game, then the value assigned to r is strictly larger than the minimum of all the values assigned to the leaves. Clearly, this property is satisfied at the beginning of the game.

Suppose that during the game Spoiler pebbles the variable u . Let T_1, \dots, T_s be those newly created dominated trees in G that have pebbled roots r_1, \dots, r_s , for $s \geq 0$. If $s > 0$, let r_i be the root that received the minimal value a among all the roots r_1, \dots, r_s . We claim that if u is the root of a dominated tree T , then a is strictly larger than the minimum b of all the values assigned to the leaves of T . Otherwise, the graph $T \cup T_i$ was a dominated tree that violates the invariant even before the variable u has been pebbled, a contradiction. Therefore, in this case Duplicator can choose a value c between b and a for the variable u . Since c is smaller than a , in all the new dominated trees T_1, \dots, T_s in G the value assigned to r_1, \dots, r_s is strictly larger than c , and hence the invariant is preserved. In particular, if $R^{\min}(w, u, v)$ (or $R^{\min}(w, v, u)$) is a constraint in Φ where w and v have been pebbled, then this constraint is satisfied by the assignment.

Since c is larger than b , this choice also guarantees that if v, v' are pebbled variables then any constraint of the form $R^{\min}(u, v, v')$ is satisfied, because in this case the variables u, v, v' induce a dominated tree with root u in G .

If there is no dominated tree T where u is the root, then Duplicator assigns a value to u that is smaller than all values assigned to other variables. If $s = 0$, Duplicator plays a value that is larger than all values assigned to other variables. In both cases it is easy to check that Duplicator maintains the invariant, and satisfies all constraints $\phi \in \Phi$ where all variables are pebbled. By induction, we have shown that Duplicator has a winning strategy for the existential k -pebble game on Φ . \square

Exercises.

42. For all $0 < n < m$, determine the smallest value k such that Spoiler wins the existential $(k, k + 1)$ -pebble game on T_n, T_m .
43. Determine for all $n, m \geq 2$ and $k \geq 2$ whether Spoiler or Duplicator has a winning strategy in the existential $(k, k + 1)$ -pebble game on K_n, K_m .
44. Determine for all $n, m \geq 3$ the smallest value k such that Spoiler wins the existential $(k, k + 1)$ -pebble game on \vec{C}_n, \vec{C}_m .
45. Write a Datalog program that solves the H -coloring problem for $H = \vec{C}_3$.
46. Describe the digraphs A such that Spoiler wins the existential $(1, 2)$ -pebble game on A and T_3 .

5 Graph Algorithms in Constraint Satisfaction

In this section we present efficient algorithms for constraint satisfaction problems that are based on graph decompositions. Most of these problems can probably² not be solved by Datalog. They all have in common that they work on appropriately defined (directed or undirected) graphs on the variables of the constraint satisfaction problem. This graph then guides a recursive decomposition procedure that correctly decides the CSP. Moreover, it is usually easy to adapt this “divide-and-conquer” procedure such that it directly constructs a representation of a solution for the CSP. In many examples, the algorithms can be made faster if we use so-called *decremental* graph algorithms.

How these graphs are defined, and how the decomposition is performed highly depends on the CSP under consideration. However, even though we will see examples of this approach in various application areas, it turns out that the algorithms and the correctness proofs have quite some similarities.

In many examples, we need to evaluate certain (weak or strong) connectivity properties on the constraint graph. Luckily, the weakly and the strongly connected components of a graph can be computed very efficiently. For the weakly connected components, this can be done easily via depth-first-search. Also the strongly connected components can be computed by clever modifications of depth-first-search. One way to do this is by Kosaraju’s algorithm, which makes use of the fact that the transpose graph (the same graph with the direction of every edge reversed) has exactly the same strongly connected components as the original graph [3]. Another possibility is Tarjan’s algorithm [37]. Both algorithms have a running time that is linear in the size of the input graph.

5.1 The Min-Ordering Problem

As a warm-up, we present a simple polynomial-time algorithm for the min-ordering problem. Recall that such an algorithm can not be a Datalog program, as we have shown in Section 4. The algorithm that we present is not yet a graph algorithm, but it illustrates several of the ideas that are employed later for more complicated problems. Recall the min-ordering problem: we are given a set V of variables, and a set R^{\min} of triples on these variables. We want to find a mapping $\alpha : V \rightarrow \mathbb{Q}$ such that for each triple (x, y, z) either $\alpha(x) > \alpha(y)$ or $\alpha(x) > \alpha(z)$.

Let (V, R^{\min}) be an instance of the min-ordering problem that has a solution α . Clearly there must be a variable x such that $\alpha(x) \leq \alpha(y)$ for all $y \in V$.

Definition 31. *A variable in an instance (V, R^{\min}) of the min-ordering problem is called free if the instance has a solution α where $\alpha(x) \leq \alpha(y)$ for all $y \in V$.*

If x is free, then R^{\min} cannot contain a triple (x, y, z) where x appears in the first entry, because then either $\alpha(y)$ or $\alpha(z)$ are strictly smaller than $\alpha(x)$, a contradiction. We say that a variable $x \in V$ is a *min-candidate* if R^{\min} does not contain a tuple (x, y, z) where x appears in the first entry.

Lemma 32. *If an instance of the min-ordering problem does not contain min-candidates, then it is inconsistent.*

²in some cases, this question is open

Proof. Suppose the instance has a solution, then there must exist a free variable. But as we have seen, free variables cannot be min-candidates. \square

It turns out that if an instance of the min-ordering problem is satisfiable, then *every* min-candidate is free. This follows from the correctness of the following algorithm.

```

Min-Ordering( $I$ )
Input: A structure  $(V, R^{\min})$  with a ternary relation  $R^{\min}$ .

If  $V = \emptyset$  then accept
else
    Select a min-candidate  $x$ ;
    If no such node exists then reject;
    else Min-Ordering( $I - x$ )
end if

```

Figure 5: The Min-Ordering algorithm.

Proposition 33. *The Min-Ordering procedure given in Figure 5 correctly decides the min-ordering problem in linear time in the input size.*

Proof. Let I be an instance of the min-ordering problem. If at some point during the execution of the Min-Ordering on I we recursively apply the min-ordering procedure to a substructure I' of I and do not find a min-candidate, then Lemma 32 implies that I' and therefore also I does not have a solution.

Otherwise, it is clear that I has a solution if the set of variables is empty. Inductively assume that $I - x$ has a solution α . Then the extension α' of α that maps x to a value smaller than the value of α for all other variables is clearly a solution to I . Hence, the algorithm is correct. It is clear that it can be implemented such that it has a linear worst-case running time. \square

Exercises.

42. Show that if an instance of the min-ordering problem has a solution, then it also has an injective solution.

5.2 Phylogenetic Reconstruction

In computational biology, *phylogenetic analysis* is a field where we have to deal with partial information about evolutionary trees. An *evolutionary tree* for a set of species is a rooted tree where the leaves are bijectively labeled with the species from the set. Constructing evolutionary trees from biological data is a difficult problem for a variety of reasons (see [20]). Many approaches assume that the evolutionary tree is built from data based on the comparison of a single protein or a single position in aligned protein sequences, but very often the resulting tree will be different depending on which particular protein or position is used. Several trees, each from a different protein or position, must be built and be shown to be “generally consistent” before the implied evolutionary history is considered reliable.

The question whether such consistency tests can be automated motivates in computational biology the various *consensus tree problems* (also called *common supertree problems*) that are studied in this subsection [20].

We fix some standard terminology concerning rooted trees. Let T be a tree with a distinguished vertex r , the *root* of T . For $u, v \in V(T)$, we say that u *lies below* v if the path from u to r passes through v . We say that u *lies strictly below* v if u lies below v and $u \neq v$. The *youngest common ancestor (yca)* of two vertices $u, v \in V(T)$ is the node w such that both u and v lie below w and w has maximal distance from r . Note that the yca, viewed as a binary operation, is commutative and associative, and hence there is a canonical definition of the yca of a set of elements u_1, \dots, u_k .

Rooted Triple Consistency. The rooted triple problem is one of the fundamental problems in phylogenetic reconstruction. In 1981, Aho, Sagiv, Szymanski, and Ullman [2] presented a polynomial time algorithm to this problem, motivated independently from computational biology by questions in database theory.

Problem 11. Rooted-Triple-Consistency

INSTANCE: A finite relational structure (V, R^{yca}) , where R is a ternary relation

QUESTION: Is there a rooted tree T with leaves X and a mapping $\alpha : V \rightarrow X$ such that for every triple $(x, y, z) \in R^{yca}$ the yca of $\alpha(x)$ and $\alpha(y)$ lies strictly below the yca of $\alpha(x)$ and $\alpha(z)$ in T ?

It can be verified that the rooted-triple-consistency problem is indeed a constraint satisfaction problem (see Exercise 43; also see the paragraph on rooted triples and Datalog below).

The worst-case running time of the algorithm of Aho et al. is in $O(nm)$, where $n = |V|$ and $m = |T|$. We later show how the running time of the algorithm can be improved using decremental graph connectivity algorithms. The basic idea of the algorithm is to construct for a given instance $I = (V, T)$ of the rooted triple consistency problem a graph $G = (V, E)$, where $E = \{\{x, y\} \mid x, y \in V \text{ and there is } z \in V \text{ s.t. } (x, y, z) \in R^{yca}\}$.

Lemma 34. *If the graph G defined for an instance $I = (V, R^{yca})$ of the rooted triple consistency problem is connected, then I is inconsistent.*

Proof. Suppose that there is a rooted tree T with leaves X and a mapping $\alpha : V \rightarrow X$ such that for every triple $(x, y, z) \in R^{yca}$ the yca of x, y lies strictly below the yca of x, z in T . Let r be the yca of $\alpha(V)$, i.e., the set of all leaves in the image of V under α . It can not be that all vertices in $\alpha(V)$ lie below the same child of r in T (otherwise the child would have been the yca of $\alpha(V)$). Since the graph G is connected, there is an edge $\{x, y\}$ in G such that $\alpha(x)$ and $\alpha(y)$ lie below different children of r in T . Hence, there is a $z \in V$ such that $(x, y, z) \in R$. By assumption, the yca of $\alpha(x)$ and $\alpha(y)$, which is r , lies strictly below the yca of $\alpha(x)$ and $\alpha(z)$, a contradiction to the choice of r . \square

If G is disconnected, let G_1, \dots, G_k be the distinct connected components of G (they can be computed using any linear time algorithm). For each G_i , recurse on the problem $I[V(G_i)]$, i.e., the instance $(V(G_i), R_i^{yca})$, where R_i^{yca} is a ternary relation that contains all triples $(x, y, z) \in R^{yca}$ such that $x, y, z \in V(G_i)$. If any of these recursive calls reports an inconsistency, then the instance is clearly inconsistent as well. Otherwise, by inductive

assumption there is a tree T_i and a mapping $\alpha_i : V(G_i) \rightarrow V(T_i)$ showing that $I[V(G_i)]$ is consistent. Let T be the tree obtained by creating a new vertex r and adding T_1, \dots, T_k by making the roots of these trees as the children of r . Let α be the mapping that maps x to $\alpha_i(x)$ if $x \in G_i$. We claim that T and α show that the instance I is consistent. If $(x, y, z) \in R^{yca}$ lies completely inside one component, there is nothing to show, since T_i and α_i guarantee that the yca of $\alpha(x)$ and $\alpha(y)$ lies strictly below the yca of $\alpha(x)$ and $\alpha(z)$. It can not be that x and y are in distinct components, since they are connected by an edge in G . Hence, all other tuples must be of the form that $x, y \in G_i$ and $z \in G_j$ for $i \neq j$. But in this case the yca of $\alpha(x)$ and $\alpha(y)$ lies below the root of T_i and hence strictly below r , which is the yca of $\alpha(x)$ and $\alpha(z)$. This concludes the correctness proof of the algorithm shown in Figure 6.

```

ASSU( $I$ )
Input: a structure  $(V, R^{yca})$  with a ternary relation  $R^{yca}$ .

If  $R^{yca} = \emptyset$  then
    Create a new vertex  $r$ , and linke all  $v \in V$  below  $r$ .
    return the tree rooted at  $r$ 
else
    Construct the graph  $G$  for  $I$ ;
    Compute the connected components  $G_1, \dots, G_k$  of  $G$ ;
    If there is only one connected component then reject
    else
        Let  $T_i$  be the tree  $\text{ASSU}(I[V(G_i)])$ , for all  $i \leq k$ ;
        Create a new vertex  $r$ , and link  $T_i$  below  $r$  for all  $i \leq k$ ;
        return the tree rooted at  $r$ 
    end if
end if

```

Figure 6: The ASSU algorithm.

Decremental Graph Connectivity. In each recursive call of the ASSU, the connected components are computed from scratch again, even though only some vertices together with incident edges are removed from the graph. Henzinger, King, and Warnow [25] observed that one can exploit *decremental graph connectivity* algorithms to improve the overall running time of the algorithm.

In a *dynamic graph problem* the task is to decide a graph property A for a graph that is evolving over time. The graph may change since edges of vertices are added or deleted. To decide the property A , algorithms for dynamic graph problems maintain a data structure. The data structure can be updated each time that the graph changes. We do not in advance when and how the graph will be updated, and when and how often property A has to be decided.

Often, the number of vertices is fixed and only edges are added and removed. One special case are dynamic graph problems where edges are only removed; in this case, we speak of a *decremental* graph problems.

There is a series of paper about decremental graph connectivity; see Figure 7. As usual, let n be the number of vertices and m the number of rooted triples in the instance.

Proposition 35. *The ASSU algorithm can be implemented with a worst-case running time in $O(m \log^2(n))$.*

Proof. We use a deterministic decremental graph connectivity algorithm of Holm, de Lichtenberg, and Thorup [38], which has a query time in $O(\log n / \log \log n)$, and an update time in $O(\log^2 n)$.

We implement the computation of the graph G and its connected components as follows. Initially, the graph G is fully constructed, and we also compute all connected components of G . We know that all edges (x, y) in the graph can be removed where for all rooted triples (x, y, z) the vertices x and y are in a different component than z . Using the decremental graph connectivity algorithm, we remove all such edges one by one, and each time make a query whether x and y are still connected in the graph. If this stops being the case, we start a search (e.g., depth-first search) from both a and b , alternating with every discovered edge between the two searches. Eventually, the component with the fewer edges will be completely discovered, and we re-label all vertices in this component. The length of the search is proportional to the number of edges in this component. Since each edge is in the component with the fewer edges at most $\log m$ times during the execution of the algorithm, each edge is visited at most $\log m$ times. The total running time for the search is hence in $O(m \log m)$.

Answering all decremental graph connectivity queries takes $O(m \log n / \log \log n)$, and the update costs sum up to $O(m \log^2 n)$. Therefore, the running time of the entire algorithm is in $O(m \log^2 n)$. \square

Rooted Triples and Datalog. There has been work on whether or not the rooted triple consistency problem can also be solved by local consistency techniques [9]. However, these results are not strong enough to provide an answer to the following question, which is still open.

Question 5. *Is there a Datalog program that solves the rooted triple consistency problem?*

To use the characterization of (l, k) -Datalog by the existential pebble game, we have to formulate the rooted triple consistency problem as CSP(Γ) where Γ is a relational structure with finitely many l -ary primitive positive definable relations. The following structure Δ

Year	Authors	Deterministic	Randomized
85	Frederickson	$O(\sqrt{m})$	
92	Eppstein, Galil, Italiano	$O(\sqrt{n})$	
95	Henzinger, King		$O(m \log^3(n))$
96	Henzinger, Thorup		$O(\log^2(n))$
97	Henzinger, King	$O(n^{1/3} \log n)$	
97	Holm, de Lichtenberg, Thorup	$O(\log^2(n))$	

Figure 7: History of decremental graph connectivity algorithms. The table shows the amortized update time per edge deletion.

satisfies this condition. The domain of Δ is $\mathbb{N} \rightarrow \mathbb{Q}$, i.e., the set of all functions from the natural numbers to the rational numbers (hence, the domain of Δ is uncountable). For two elements f, g of Δ , let $k_{f,g}$ be the largest natural number such that $f(i) = g(i)$ for all $i < k_{f,g}$. The ternary relation R^{yca} in Δ is the relation consisting of all triples (f, g, h) such that either $k_{f,g} > k_{f,h}$ or $k_{f,g} = k_{f,h}$ and $f(k_{f,g}) < h(k_{f,h})$.

Indeed, it is true [1, 14] that for all l there is only a finite number of l -ary primitive positive definable relations in Δ . Hence, the conditions of Theorem 28 apply to Δ , and we can use the existential pebble game to prove that the rooted triple consistency problem can not be solved by (2, 3)-Datalog (see Exercise 44).

Hard Phylogenetic Reconstruction Problems. It quickly happens with phylogenetic reconstruction problems that we pass the border between tractability and NP-completeness. The following unrooted version of the rooted triple consistency problem, for instance, is NP-complete [36].

Problem 12. Quartet-Consistency

INSTANCE: A finite relational structure (V, Q) , where Q is a quaternary relation.

QUESTION: Is there a tree T and a mapping $\alpha : V \rightarrow V(T)$ such that for each quadruple $xy|uv$ in C the paths from $\alpha(x)$ to $\alpha(y)$ and from $\alpha(u)$ to $\alpha(v)$ in T do not have common vertices?

Exercises.

43. Use Lemma 24 to verify the claim of the text that the rooted-triple-consistency problem is a constraint satisfaction problem.

44. Prove that the rooted-triple-consistency problem can not be solved by a (2, 3)-Datalog program.

Hint: Use the structure Δ described in this subsection, and play the existential (2, 3)-pebble game on an appropriate inconsistent instance S of the rooted-triple-consistency problem and Γ . **Alternatively:** Construct an inconsistent instance S where any sound (2, 3)-Datalog program Π can never apply a rule that derives false. Prove this by induction on the evaluation of Π on S .

5.3 Tree Description Constraints

The tree description consistency problem has been introduced by Cornell in computational linguistics [11]. We discuss a simplified version of Cornell’s tree descriptions here, and present an efficient algorithm to decide their consistency. The presented algorithm combines the ideas from the algorithms for the min-ordering problem in Subsection 5.1 and the rooted triple consistency problem in Subsection 5.2. For the terminology concerning rooted trees, see Subsection 5.2.

Problem 13. Tree-Description-Consistency

INSTANCE: A finite structure $(V, <, ||)$ where $<$ and $||$ are binary relations.

QUESTION: Is there a rooted tree T and $\alpha : V \rightarrow V(T)$ such that if $x < y$ then $\alpha(y)$ lies strictly below $\alpha(x)$ in T , and if $x||y$ then neither $\alpha(x)$ lies below $\alpha(y)$ nor $\alpha(y)$ lies below $\alpha(x)$

in T ?

It is straightforward to verify that this problem is closed under disjoint unions and inverse homomorphisms. By Lemma 24, there exists an infinite structure Γ such that the Tree-Description-Consistency Problem can be formulated as $\text{CSP}(\Gamma)$. Note that the definition of $\|\cdot\|$ is symmetric, and therefore we do not distinguish between the constraint $x\|y$ and $y\|x$.

It will be convenient to describe the tree-description-consistency problem as the CSP for a special infinite structure Λ . The structure Λ is studied in permutation group theory and model theory, and is called the *dense proper semilinear order* [1, 10, 14]. The domain of the structure is the set Λ of all non-empty finite sequences $a = (q_0, q_1, \dots, q_{n-1})$ of rational numbers. Let $a < b$ if either

- b is a proper initial subsequence of a , i.e., $b = (q_0, \dots, q_{n-1})$ and $a = (q_0, \dots, q_{n-1}, q_n, \dots, q_m)$, or
- $b = (q_0, \dots, q_{n-1}, q_n)$ and $a = (q_0, \dots, q_{n-1}, q'_n, q_{n+1}, \dots, q_m)$, where the rational number q_n is smaller than q'_n .

Finally, we define $x\|y$ if neither $x < y$ nor $y < x$ nor $x = y$.

It is straightforward to verify that $\text{CSP}(\Lambda)$ is the tree-description-consistency problem. In particular, a solution to an instance $I = (V, <, \|\cdot\|)$ is a homomorphism $f : V \rightarrow V(\Lambda)$. Note that precisely one of the following four cases holds for any two vertices $u, v \in V(\Lambda)$:

$$u = v, u < v, v < u, u\|v$$

Moreover, it can be shown that Λ has for every l a finite number of l -ary primitive positive definable relations (see Sections 3.6 and 4.3).

Definition 36. Let $(V, <, \|\cdot\|)$ be an instance of the tree-description-consistency problem. We say that $x \in V$ is *free* if there exists a solution α such that there is no $y \in V$ with $\alpha(y) < \alpha(x)$.

Lemma 37. Let $I = (V, <, \|\cdot\|)$ be a consistent instance of the tree-description-consistency problem. If the digraph $(V, <)$ of I is weakly connected, then I must have a free variable $x \in V$.

Proof. Let α be a solution of I . Assume for contradiction that there are two distinct vertices x and x' with the property that there is no $y \in V$ with $\alpha(y) < \alpha(x)$, and no $y \in V$ with $\alpha(y) < \alpha(x')$. Note that then $\alpha(x)\|\alpha(x')$ in Λ . Since $(V, <)$ is weakly connected, there must be a sequence $x = x_0, \dots, x_k = x'$ of vertices in V such that $x_i < x_{i+1}$ or $x_i > x_{i+1}$ in I for all $0 \leq i < k$. Hence, in the sequence $\alpha(x_0), \dots, \alpha(x_k)$ there must be a vertex $\alpha(x_j)$ such that $\alpha(x_j) < \alpha(x)$ and $\alpha(x_j) < \alpha(x')$. This contradicts the choice of x and x' . \square

It is clear that if x is free then there can not be a $y \in V$ such that $x\|y$ or $y < x$ in I . Similarly as in Subsection 5.1, we therefore make the following definition.

Definition 38. A variable of an instance $I = (V, <, \|\cdot\|)$ of the tree-description-consistency problem is called *blocked*, if there is a $y \in V$ such that $x\|y$ or $y < x$ holds in I .

Corollary 39. An instance $I = (V, <, \|\cdot\|)$ where $(V, <)$ is weakly connected and all variables are blocked is inconsistent.

```

TDC( $I$ )
Input: a structure  $I = (V, <, ||)$  with a two binary relations  $<$  and  $||$ .

Compute the weakly connected components  $V_1, \dots, V_k$  of  $(V, <)$ ;
For  $i = 1$  to  $k$ 
    If there is an unblocked  $x$  in  $I[V_i]$  call  $\text{TDC}(I[V_i] - x)$ 
    else reject.
end for

```

Figure 8: The TDC algorithm.

Proposition 40. *The TDC algorithm given in Figure 9 rejects an instance of the tree description consistency problem if and only if the instance is inconsistent, in time that is quadratic in the size of the instance.*

Similarly as for the ASSU algorithm, it is also possible to speed up the TDC algorithm by employing a decremental graph connectivity algorithm.

Exercises.

45. Show that there is a polynomial-time reduction from the rooted triple consistency problem to the tree description consistency problem.
46. Verify that the set of consistent binary plane tree descriptions is closed under disjoint unions and inverse homomorphisms.
47. Show that every consistent instance of the tree-description-consistency problem has a solution that is *injective*.
48. Let $\Lambda = (\mathbb{Q}^*, <, ||)$ be the structure as described in this subsection. Let $\Lambda' = (\mathbb{Q}^*, E, F)$ where $(x, y) \in E$ iff not $u||v$ in Λ , and $(x, y) \in F$ iff not $x \leq y$ in Λ . Show that Λ' and Λ are *primitive positive inter-definable*, i.e., that every relation in Λ' has a primitive positive definition in Λ and vice versa.
49. Let f be a function that satisfies the recursion $f(1) = c$, and $f(m) = dm + \max_{0 < i < m} (f(i) + f(m - i))$ for $m > 1$. Determine the asymptotic growth of f .
50. Develop an algorithm that solves the partial-tree-description consistency problem in time $O(m \log^2 n)$.

5.4 Branching Time Constraints

Reasoning about time is a central topic in Artificial Intelligence. The most important model is *linearly ordered time*, and we have already seen the constraint satisfaction problem for the so-called *point algebra*, which has been studied in Artificial Intelligence [34]. Another important model is *branching time*, where for every time point the past is linearly ordered, but the future is only partially ordered. Even though the motivation to study corresponding constraint satisfaction problem problems are quite different, we see a close connection to the

Tree-description-consistency problem of the previous section. In fact, the problem studied here can be considered an extension of Problem 13.

Problem 14. Branching-Time-Consistency

INSTANCE: A finite relational structure $I = (V, \leq, ||=, \neq)$ where \leq , $||=$, and \neq are binary relations

QUESTION: Is there a rooted tree T and a mapping $\alpha : V \rightarrow V(T)$ such that the following is satisfied:

- a) If $x \leq y$, then $\alpha(x)$ lies above $\alpha(y)$ in T ;
- b) If $x ||= y$, then neither $\alpha(x)$ lies strictly above $\alpha(y)$ nor $\alpha(y)$ lies strictly above $\alpha(x)$;
- c) If $x \neq y$, then $\alpha(x) \neq \alpha(y)$

The first polynomial-time algorithm for this problem is due to Hirsch [26], and has a worst-case running time in $O(n^5)$. This was later improved by Jonsson and Broxvall [8], who presented an algorithm running in $O(n^{3.376})$ (this algorithm uses an $O(n^{2.376})$ algorithm for fast integer matrix multiplication). The algorithm presented here (from [7]) is simpler (e.g., it does not use fast matrix multiplication) and runs in $O(nm)$.

The algorithm we present here extends the ideas of the algorithm in the previous section in an important aspect. Recall that every consistent instance of the tree-description-consistency problem also has an injective solution (Exercise 42). This is no longer true for the branching-time-consistency problem, since cycles with respect to the relation \leq in instances I of this problem force variables to have equal values in all solutions. We therefore arrive at the concept of *free sets of variables*, a concept that will be important in many other constraint satisfaction problems.

Definition 41. *A subset of variables F in an instance of the branching-time-consistency problem is called free if there exists a solution (T, α) of I such that $\alpha(x)$ denotes the root of T for all $x \in F$.*

The following can be shown analogously as in the proof of Lemma 37.

Lemma 42. *If $I = (V, \leq, ||=, \neq)$ is a consistent instance of the branching-time-consistency problem, and the graph (V, \leq) is weakly connected, then there exists a free set of variables.*

Definition 43. *A set of variables $F \subseteq V$ of an instance of the branching-time-consistency problem is blocked if there is a $y \in V \setminus F$ such that $y \leq x$ or $y ||= x$ or if there are $x, y \in F$ such that $x \neq y$.*

Clearly, a set of variables that is blocked can not be free.

Corollary 44. *An instance I where (V, \leq) is weakly connected and all sets of variables are blocked is inconsistent.*

To detect free sets of variables, let $G = (V, E)$ be the directed graph that contains an edge $(x, y) \in E$ if $x \leq y$ or $x ||= y$ in I . A strongly connected components of G is unblocked if it is *terminal*, i.e., if there are no arcs (u, v) where $u \in F$ and $v \in V \setminus F$ in G .

Decremental Strong Connectivity. Unfortunately, there is no decremental graph connectivity algorithm known that can help to improve the time complexity of the branching-time-consistency algorithm. There are several non-trivial dynamic algorithms known for

```

PDC( $I$ )
Input: a structure  $I = (V, \leq, ||=, \neq)$  with binary relations  $\leq$ ,  $||=$ , and  $\neq$ .

Compute the weakly connected components  $V_1, \dots, V_k$  of  $(V, \leq)$ ;
For  $i = 1$  to  $k$ 
    Compute the sccs of the graph of  $I[V_i]$ ;
    if there is an unblocked terminal scc  $F$  then call  $\text{PDC}(I[V_i \setminus F])$ 
    else reject
end for

```

Figure 9: The branching-time-consistency algorithm.

Year	Authors	Update Time	Query Time
2000	Demetrescu and Italiano	$O(n^2)$	$O(1)$
2002	Roditty, Zwick	$O(m\sqrt{n})$	$O(\sqrt{n})$
2004	Roditty, Zwick	$O(m + n \log n)$	$O(n)$

Figure 10: History of dynamic strong connectivity algorithms. All the listed algorithms are deterministic. The table shows the amortized running times.

strong connectivity of digraphs, see Figure 10. However, all of these algorithms have an update-time that is at least linear in the number of edges of the graph, and hence they can not be used to improve the quadratic running time of the algorithm that computes the strongly connected components from scratch each time the graph is modified.

A positive answer to the following question would allow us to speed up a series of algorithms for constraint satisfaction problems (also see Subsection 5.5).

Question 6. *Find an algorithm for dynamic strong connectivity where both the query and the update time is amortized sub-linear in the size of the graph.*

Exercises.

- 51. Show that there is a polynomial-time reduction from the problem of partial tree description consistency to the branching time constraint consistency.
- 52. Verify the claim that the set of consistent branching time constraints is closed under disjoint unions and inverse homomorphisms.

5.5 Tree Descriptions in Computational Linguistics

Natural language sentences can be ambiguous, both syntactically and semantically. An classical example of syntactical ambiguity is the sentence “The spy sees the man with the binocular”. A classical example of semantical ambiguity is the sentence “Every person in this room speaks a language”. The syntactical structure of the last sentence is rather clear, but there is a weaker and a stronger interpretation of this sentence (see discussion below).

In computational linguistics there are several competing formalisms for analyzing the syntactical or the semantical structure of a natural language sentence. Some of the formalisms

in computational linguistics try to cope with the ambiguities, and rather than associating a single syntactical or semantical structure to a natural language sentence, they use *implicit* representations for *sets* of possible readings of the sentence.

The syntactical structure in formal languages as well as the syntactical and semantical structure of natural language sentences is typically tree-like. This can be illustrated with context-free languages. For a given word in a context-free language, the syntactical structure is the derivation tree of the word in the context-free grammar that generates the language. But also the semantical representations of natural language sentences are typically tree-like. Formulas in first-order logic, for instance, are recursively defined, and hence have a tree-like structure.

Hence, if we want to represent *all* readings of a sentence, we have to devise a way to describe sets of trees. One way to implicitly represent sets of trees is to employ *constraints* that talk about trees. The constraints express partial knowledge that we may have about the syntactical or the semantical structure of the natural language sentence, and the *solution space*, i.e., the set of all structures satisfying all constraints represents the set of possible readings of the sentence.

We do not describe one of the formalisms for partial tree descriptions in computational linguistics in detail [4–6, 30], but rather describe one of the related computational problems. The problem has an elegant description as a constraint satisfaction problem, and it lies at the heart of the combinatorial aspect of one of the formalisms in computational linguistics. As described in Section 3.4, many other interesting tasks concerning this formalism can be reduced to solving the constraint satisfaction problem.

We say that a rooted tree T is *plane* if there is an order on the children of each vertex. In particular, if T is a binary tree we can distinguish between the *left* and the *right* child of each non-leave vertex in T .

Problem 15. Consistency of binary plane tree descriptions

INSTANCE: A relational structure (V, L, R) where L and R are binary relations.

QUESTION: Is there a rooted binary plane tree T and a mapping $\alpha : V \rightarrow V(T)$ such that for every $(x, y) \in L$ (for every $(x, y) \in R$) $\alpha(x)$ has a left (right, respectively) child which lies above $\alpha(y)$ in T ?

Again it can be verified that the positive instances of this computational problem satisfy the conditions of Lemma 24 (see Exercise 53), and therefore this problem is a constraint satisfaction problem in the sense of Section 3. However, we do not work with the infinite template in this Section, but rather work with the following notion of solution.

Definition 45. A solution for an instance $I = (V, L, R)$ of the binary plane tree description problem is a pair (T, α) where T is a binary plane tree with root r and α is a mapping from V to T that shows that I is consistent as in the definition of the binary plane tree description problem.

Definition 46. Let (V, L, R) be an instance of the binary plane tree description consistency problem. A variable $x \in V$ is called a free if there is no $y \in V$ such that $(y, x) \in L$ or $(y, x) \in R$, and if there are no z, z' such that $(x, z) \in L$ and $(x, z') \in R$.

For an instance $I = (V, L, R)$, consider the following directed graph G . The vertex set of G is $L \cup R$. Two vertices (x, y) and (x', y') are joined by an arc if at least one of the following cases applies.

1. $y = y'$;
2. $(x, y) \in L$, $(x', y') \in R$, and $x = x'$;
3. $y = x'$.

Lemma 47. *Let $I = (V, L, R)$ be an instance of the binary plane tree description consistency problem without free variables. If the graph G of I is strongly connected, then I is inconsistent.*

Proof. Suppose there is a tree T and a mapping $\alpha : V \rightarrow V(T)$ that shows that I is consistent. Then there must be a variable $x \in V$ such that there is not $y \in V$ such that $\alpha(x)$ lies strictly below $\alpha(y)$. In particular, for such a variable x there is no $y \in V$ such that $(y, x) \in L$ or $(y, x) \in R$. Since x is not free, there must be $z, z' \in V$ with $(x, z) \in L$ and $(x, z') \in R$. Let $((p, q), (p', q'))$ be an edge in G . Note that either all vertices $\{\alpha(p), \alpha(q), \alpha(p'), \alpha(q')\}$ are below $\alpha(z)$, or below $\alpha(z')$, or $p = p'$, $(p, q) \in L$, $(p', q') \in R$ (by the choice of x). Hence, we have found a non-trivial directed cut in the graph G , consisting of the vertices (p, q) where $\alpha(p)$ and $\alpha(q)$ are both below $\alpha(z)$. This contradicts strong connectivity of G . \square

We say that a subset S of the vertices of a digraph G is a (*directed*) *cut* iff there is no arc $(x, y) \in E(G)$ such that $x \in V(G) \setminus S$ and $y \in S$. A cut S is called *non-trivial* if $S \neq \emptyset$ and $S \neq V(G)$. Recall that the strongly connected components of a digraph G can be computed in linear time in the size of G . If the graph G is not strongly connected, there are at least two strongly connected components, and each of the strongly connected components in this case defines a non-trivial cut in G .

```

NDC( $I$ )
Input: a structure  $I = (V, L, R)$  with a two binary relations  $L$  and  $R$ .

If  $I$  has a free vertex  $x$  then call  $\text{NDC}(I - x)$ .
else
  Compute the graph  $G$  of  $I$ ;
  Compute the strongly connected components of  $G$ ;
  If  $G$  is strongly connected then reject
  else
    Let  $C$  be a non-trivial directed cut in  $G$ ;
    Let  $V' \subset V$  be the vertices that appear in edges of  $C$ ;
    Call  $\text{NDC}(I[V'])$ ;
    Call  $\text{NDC}(I[V \setminus V'])$ 
  end if
end if

```

Figure 11: The NDC algorithm for the binary plane tree description consistency problem.

Proposition 48. *The NDC algorithm given in Figure 11 rejects an instance of the plane binary tree description consistency problem if and only if the instance is inconsistent, in time that is quadratic in the size of the instance.*

Exercises.

53. Verify the claim that the set of consistent binary plane tree descriptions is closed under disjoint unions and inverse homomorphisms.
54. Is there a solution for the following instance of the Binary-Plane-Tree-Description-Consistency problem?

$$\begin{aligned} & (\{x_1, x_2, x_3, x_4, x_5, y_1, y_2, y_3, y_4, y_5, z_1\}, \\ & \{(x_1, y_1), (x_2, y_2), (x_3, y_3), (x_4, y_3), (x_4, y_4), (x_5, y_4)\}, \\ & \{(x_1, y_2), (x_2, y_3), (x_3, y_1), (x_4, y_5), (x_5, y_5), (x_5, z_1), (z_1, y_2)\}) \end{aligned}$$

55. Implement an algorithm that finds a non-trivial cut in a given graph G , if such a cut exists. The running time of the algorithm should be linear in the size of G .

References

- [1] S. Adeleke and P. M. Neumann. Relations related to betweenness: their structure and automorphisms. *AMS Memoir*, 131(623), 1998.
- [2] A. Aho, Y. Sagiv, T. Szymanski, and J. Ullman. Inferring a tree from lowest common ancestors with an application to the optimization of relational expressions. *SIAM Journal on Computing*, 10(3):405–421, 1981.
- [3] A. V. Aho, J. E. Hopcroft, and J. D. Ullman. *The Design and Analysis of Computer Algorithms*. Addison-Wesley, 1974.
- [4] E. Althaus, D. Duchier, A. Koller, K. Mehlhorn, J. Niehren, and S. Thiel. An efficient algorithm for the configuration problem of dominance graphs. In *Proceedings of the 12th ACM-SIAM Symposium on Discrete Algorithms (SODA'01)*, pages 815–824, Washington, DC, 2001.
- [5] E. Althaus, D. Duchier, A. Koller, K. Mehlhorn, J. Niehren, and S. Thiel. An efficient graph algorithm for dominance constraints. *Journal of Algorithms*, pages 194–219, 2003.
- [6] M. Bodirsky, D. Duchier, J. Niehren, and S. Miele. A new algorithm for normal dominance constraints. In *Proceedings of the ACM-SIAM Symposium on Discrete Algorithms (SODA'04)*, pages 59–67, New Orleans, January 2004.
- [7] M. Bodirsky and M. Kutz. Pure dominance constraints. In *Proceedings of STACS'02*, pages 287–298, 2002.
- [8] M. Broxvall and P. Jonsson. Point algebras for temporal reasoning: Algorithms and complexity. *Artif. Intell.*, 149(2):179–220, 2003.
- [9] D. Bryant and M. Steel. Extension operations on sets of leaf-labelled trees. *Advances in Applied Mathematics*, 16:425–453, 1995.
- [10] P. J. Cameron. The random graph. *R. L. Graham and J. Nešetřil, Editors, The Mathematics of Paul Erdős*, 1996.

- [11] T. Cornell. On determining the consistency of partial descriptions of trees. In *Proceedings of the ACL*, pages 163–170, 1994.
- [12] V. Dalmau, P. G. Kolaitis, and M. Y. Vardi. Constraint satisfaction, bounded treewidth, and finite-variable logics. In *Proceedings of CP'02*, pages 310–326, 2002.
- [13] R. Diestel. *Graph Theory, 3rd edition*. Springer–Verlag, New York, 2005.
- [14] M. Droste. Structure of partially ordered sets with transitive automorphism groups. *AMS Memoir*, 57(334), 1985.
- [15] H.-D. Ebbinghaus and J. Flum. *Finite Model Theory*. Springer, 1999. 2nd edition.
- [16] T. Feder. Classification of homomorphisms to oriented cycles and of k -partite satisfiability. *SIAM J. Discrete Math.*, 14(4):471–480, 2001.
- [17] T. Feder and M. Vardi. Monotone monadic SNP and constraint satisfaction. In *STOC'93*, pages 612 – 622, 1993.
- [18] T. Feder and M. Vardi. The computational structure of monotone monadic SNP and constraint satisfaction: A study through Datalog and group theory. *SIAM Journal on Computing*, 28:57–104, 1999.
- [19] M. Garey and D. Johnson. *A guide to NP-completeness*. CSLI Press, 1978.
- [20] D. Gusfield. *Algorithms on strings, trees, and sequences. Computer Science and Computational Biology*. Cambridge University Press, New York, 1997.
- [21] P. Hell and J. Nešetřil. The core of a graph. *Discrete Math.*, 109:117–126, 1992.
- [22] P. Hell and J. Nešetřil. On the complexity of H-coloring. *Journal of Combinatorial Theory, Series B*, 48:92–110, 1990.
- [23] P. Hell and J. Nešetřil. *Graphs and Homomorphisms*. Oxford University Press, 2004.
- [24] P. Hell, J. Nešetřil, and X. Zhu. Duality and polynomial testing of tree homomorphisms. *TAMS*, 348(4):1281–1297, 1996.
- [25] M. Henzinger, V. King, and T. Warnow. Constructing a tree from homeomorphic subtrees, with applications to computational evolutionary biology. In *Proceedings of the 7th Symposium on Discrete Algorithms (SODA'96)*, pages 333–340, 1996.
- [26] R. Hirsch. Expressive power and complexity in algebraic logic. *Journal of Logic and Computation*, 7(3):309 – 351, 1997.
- [27] W. Hodges. *A shorter model theory*. Cambridge University Press, 1997.
- [28] S. Janson, T. Łuczak, and A. Ruciński. *Random Graphs*. John Wiley and Sons, 2000.
- [29] P. G. Kolaitis and M. Y. Vardi. Conjunctive-query containment and constraint satisfaction. In *Proceedings of PODS'98*, pages 205–213, 1998.

- [30] A. Koller, K. Mehlhorn, and J. Niehren. A polynomial-time fragment of dominance constraints. In *Proceedings of the 38th Annual Meeting of the Association for Computational Linguistics (ACL'00)*, Hong Kong, Oct. 2000.
- [31] B. Larose, C. Loten, and C. Tardif. A characterisation of first-order constraint satisfaction problems. *Logical Methods in Computer Science*, DOI: 10.2168/LMCS-3(4:6), 2007.
- [32] T. Luczak and J. Nešetřil. When is a random graph projective? *Eur. J. Comb.*, 27(7), 2006.
- [33] A. K. Mackworth. Consistency in networks of relations. *AI*, 8:99–118, 1977.
- [34] H. K. Marc Vilain and P. van Beek. Constraint propagation algorithms for temporal reasoning: A revised report. *Reading in Qualitative Reasoning about Physical Systems*, pages 373–381, 1989.
- [35] U. Montanari. Networks of constraints: Fundamental properties and applications to picture processing. *Inf. Sci.*, 7:95–132, 1974.
- [36] M. Steel. The complexity of reconstructing trees from qualitative characters and subtrees. *Journal of Classification*, 9:91–116, 1992.
- [37] R. Tarjan. Depth-first search and linear graph algorithms. *SIAM Journal on Computing*, 1(2):146–160, 1972.
- [38] M. Thorup, J. Holm, and K. de Lichtenberg. Poly-logarithmic deterministic fully-dynamic graph algorithms I: connectivity and minimum spanning tree. Technical report, Department of Computer Science, University of Copenhagen, 1998.