# Algorithmic Aspects of Ordered Structures

Jens Gustedt

▶ **To cite this version:**

Jens Gustedt. Algorithmic Aspects of Ordered Structures. Mathematics [math]. Technische Universität Berlin, 1992. English. tel-00549774

**HAL Id: tel-00549774**

**https://tel.archives-ouvertes.fr/tel-00549774**

Submitted on 22 Dec 2010

**HAL** is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

# ALGORITHMIC ASPECTS OF ORDERED STRUCTURES

vorgelegt von
Diplom-Mathematiker
Jens Gustedt

Vom Fachbereich 3 – Mathematik
der Technischen Universität Berlin
zur Erlangung des akademischen Grades
Doktor der Naturwissenschaften
— Dr. rer. nat. —
genehmigte Dissertation

Promotionsausschuß:

| | |
|---|---|
| Vorsitzender: | Prof. Dr. rer. nat. Günter Frank |
| Berichter: | Prof. Dr. rer. nat. Rolf H. Möhring |
| Berichter: | Prof. Dr. rer. nat. Michel Habib |

Tag der mündlichen Prüfung: 3. Juli 1992

Berlin 1992
D 83

## Zusammenfassung

In dieser Arbeit verbinden wir die Theorie der Quasi-Ordnungen mit der Theorie der Algorithmen einiger kombinatorischer Objekte. Zuerst entwickeln wir die Theorie der Wohl-Quasi-Ordnungen, WQO, im Zusammenhang zur maximalen Komplexität.

Dann geben wir ein allgemeines 0-1-Gesetz für erbliche Eigenschaften, das Auswirkungen für die mittlere Komplexität hat. Dieses Ergebnis für mittlere Komplexität wird auf die Klasse der endlichen Graphen, versehen mit der Relation "induzierter Subgraph", angewendet. Wir erhalten, daß eine große Klasse von Problemen, welche z.B. Perfektheit umfaßt, Algorithmen mit im Mittel konstanter Laufzeit haben.

Dann zeigen wir, indem wir ein Ergebnis von Damaschke für Cographen veralgemeinern, daß die Klassen der endlichen Ordnungen bzw. Graphen mit beschränktem Dekompositionsdurchmesser bzgl. der Relation "induzierte Subordnung" bzw. "induzierter Subgraph" WQO sind. Dies führt uns zu linearen Erkennungsalgorithmen für alle erblichen Eigenschaften über diesen Objekten.

Unser Hauptresultat ist dann, daß die Menge der endlichen partiellen Ordnungen eine Wohl-Quasi-Ordnung bzgl. einer gewissen Relation $\preceq$, der sogenannten Ketten-Minor-Relation, ist. Um dies zu beweisen, führen wir eine verwandte Relation auf endlichen formalen Sprachen ein, die die gleiche Eigenschaft hat. Als Folgerung erhalten wir, daß jede Eigenschaft, die erblich bzgl. $\preceq$ ist, einen Test in $O\left(|P|^c\right)$ Zeit zuläßt, wobei $c$ von der Eigenschaft abhängt. Dieser Test läßt sich leicht parallelisieren. Auf einer parallelen Maschine (CRCW PRAM) kann er so implementiert werden, daß er konstante Zeit auf $O\left(|P|^c\right)$ Prozessoren benötigt.

## Abstract

In this work we relate the theory of quasi-orders to the theory of algorithms over some combinatorial objects. First we develope the theory of well-quasi-orderings, wqo's, and relate it to the theory of worst-case complexity.

Then we give a general 0-1-law for hereditary properties that has implications for average-case complexity. This result on average-case complexity is applied to the class of finite graphs equipped with the induced subgraph relation. We obtain that a wide class of problems, including e.g. perfectness, has average constant time algorithms.

Then we show, by extending a result of Damaschke on cographs, that the classes of finite orders resp. graphs with bounded decomposition diameter form wqo's with respect to the induced suborder resp. induced subgraph relation. This leads to linear time algorithms for the recognition of any hereditary property on these objects.

Our main result is then that the set of finite posets is a wqo with respect to a certain relation $\preceq$, called chain minor relation. To prove this we introduce a similar relation on finite formal languages that also has this property. As a consequence we obtain that every property which is hereditary with respect to $\preceq$ has a test in $O\left(|P|^c\right)$ whereas $c$ depends on the property. This test has an easy parallelization with the same costs. On a parallel machine (CRCW PRAM) it may be implemented in such a way that it runs in constant time and needs $O\left(|P|^c\right)$ processors.

## Preface

As the title shows, this work tries to combine two different points of view on combinatorial objects — a structural one and an algorithmical one.

The structural approach is the theory of (well-)quasi-orders, i.e., the theory about certain relational structures. The algorithmical one is based on the concept of testing properties of objects by looking for smaller objects contained in them, i.e., by looking for forbidden substructures. The interplay between those two approaches has been very fruitful in the theory of Graph Minors developed by Robertson and Seymour in the last years, and so it seems legitimate to try to use similar concepts for other sorts of structures as well.

I hope that my attempt to bring these two different aspects — structure and algorithms — together did not fail completely and that it might give the reader a better insight into parts of the theory of combinatorial objects.

It certainly would have failed without the kind, patient, and competent support I received from Prof. R. H. Möhring who supervised this research and to whom I owe all my knowlege this work is based upon.

Also I like to thank the numerous other people that participated in discussions about the subject(s), gave hints or asked the right questions. Finally, special thanks to Karsten Weihe also for proof-reading parts of the manusscript and thus improving style and readability.

Berlin July 21, 1992

Jens Gustedt

# Contents

CHAPTER I

# Introduction

## 1. Overview

In the last years algorithmic aspects of well quasi orders (wqo) brought great progress in algorithmic graph theory. In a series of papers Robertson and Seymour ( see [RS83a], [RS86a] ... ) showed that a set of graphs together with the graph minor relation forms a wqo. This can be used to show existence of polynomial time algorithms for a wide class of problems. These problems are those, which are hereditary with respect to the graph minor relation.

A similar theory for finite posets was not known until now — i.e., that the theory of wqo's is used for such a general statement about existence of algorithms.

The starting point for this work was to investigate a relation between posets arising from application in scheduling theory, the so called chain minor relation. We show in section IV.14 that this relation leads to analogous results for finite posets as the graph minor relation does for graphs. In particular we show that the chain minor relation defines a wqo and we give existence proofs for algorithms.

But in investigating this special relation we noticed that similar concepts have been used in different contexts and different "social strata" of the scientific community, namely Discrete Mathematics, Theoretical Computer Science and Operations Research. There seem to be two main motivations for people to study wqo's, or qo's in general: an interest in structural theory (**what** are we dealing with) and an interest in algorithms and constructivity (**how** do we obtain what we claim to have). Since people are often mainly motivated by only one of these, the other half is often omitted.

We think that both sides could profit from each other, so we found it worth to collect some of the different approaches, state them systematically and unify notations. This is done in chapter II. After having given the necessary definitions and facts from a structural point of view in section I.3, we show several consequences in complexity theory in section II.6 and 7. There are two main consequences, one for worst-case complexity (sec.6), the other for average-time complexity (sec.7):

- The theory of properly encoded wqo is equivalent to (m)any theory of complexity classes.
- There is a general 0-1-law for hereditary properties.

Most of the results given in this section are not new. Nevertheless many of them have been used only implicitly by other authors, so they are perhaps stated here for the first time.

To exemplify these approaches we discuss finite graphs equipped with different order relations. For worst case complexity we try to give a short introduction to the theory of graph minors that was mentioned above (sec.6.4). Since the articles needed for this theory cover several hundreds of pages at the moment, clearly this can not be complete in any sense.

For graphs equipped with the induced suborder relation we state some results concerning average-time complexity in section 8. In particular, we show that many properties have fast average time recognition algorithms. What "fast" means, depends on the representation of the graph. It varies from constant running time if we have random access to the edges, to quadratic if they are given in one single unsorted list.

The properties that are covered by that approach are those that have a recognition algorithm that runs in time $2^{O(n)}$, $n$ being the number of vertices. This includes not only all properties with polynomial time algorithms but also some **NP**-complete problems and some for which the complexity status is not yet known, e.g. perfectness of graphs.

In Chapter III we revisit four well known quasi-orders: on antichains (sec.9), strings (sec.10), structured trees (sec.11), and special classes of posets (sec.12). We revisit them for several reasons.

The first is to explain and extend a certain proof technique, called the **minimal bad sequence** technique invented by Nash-Williams. Namely we give a technique one could call **minimal antichain** technique. This is done because from an algorithmic point of view the set of antichains of a quasi-order contains all information we need, and, on the other hand, not much effort is needed to show that all quasi-orders in question have no infinite descending chains.

Another reason is that the two main theorems in that field, Higman's String Theorem and Kruskal's Tree Theorem, also have algorithmic consequences that we found worth being stated. Indeed, this theory on strings and trees is a nice example for the tools developed in section 6 where the algorithms are well behaved and practical in the sense that they would be easy to implement.

As an application of the theory on structured trees we give an extension of an approach by Damaschke [Dam90]. He showed that the class of cographs equipped with the induced subgraph relation forms a wqo. We extend his result and proof technique to series parallel orders and more general to orders with a bounded decomposition diameter. Our result then is

- In every class of posets with bounded decomposition diameter every property that is hereditary with respect to the induced suborder relation has a linear time test.

Chapter IV is dedicated to the application of the theory developed to two special classes of combinatorial objects: formal languages and partially ordered sets.

This two kinds of structures are very closely related. The setting of formal languages is the more general one — we started studying it when we wanted to prove things about partially ordered sets. But it developed its own beauty and extended to an object of study by its own rights. We define a relation on formal languages we call string minor relation that is very similar to the chain minor relation on posets. The main result for formal languages is

- Any infinite set of formal languages that are finite and do not use any symbol in any particular string twice is a wqo.

The proof for this result is based on a finite recursion. First we show that we may assume that the *length* of all strings is bounded by a constant, and then how to find related languages of length $\leq h+1$ if we already know how to find them for length $= h$.

Finally we come back to the starting point of this research, namely partially ordered sets and the chain minor relation. This relation was introduced recently by Möhring and Müller in [MM92] to generalize certain approaches in the theory of scheduling stochastic project networks. We show:

- The chain minor relation leads to a well-quasi-ordering structure on any set of finite partially ordered sets.
- Each hereditary property with respect to this relation on partially ordered sets admits a polynomial time test.

In contrast to the situation, e.g. for graph minors, the exponent of the polynomial here strongly depends on the property.

Hoping that we will have readers from different fields, we found it necessary not only to give the notations from order theory in this chapter, but also to state some prerequisites from set theory and from complexity theory. The reader who is familiar with one of these theories may easily skip this part. In case of doubt about the notation she or he should use the index we included at the end.

## 2. Complexity Classes

In this section we will introduce the notation used from complexity theory. This is not meant be a complete introduction into that field.

For a set $A$, the **alphabet**, and an integer $k \geq 0$ a **string** of length $= k$ over $A$ is a $k$-tuple of elements in $A$. For convenience there is a unique string of length $= 0$ over any alphabet, the **empty** string denoted by $\emptyset$. An element $a \in A$ is often called a **symbol** and identified with the 1-tuple $(a)$.

For a set $A$ let $A^*$ denote the set of finite strings over $A$. An arbitrary subset $L \subseteq A^*$ is called a **formal language** over $A$. $A$ is then called the **alphabet** or **domain** of $L$, $\mathrm{dom}\,(L)$.

Let $V$ be a set, the set of objects, and $A$ be a finite set, the set of symbols. An injective mapping from $V$ into $A^*$ , the set of strings over $A$, is an **encoding** of $V$ w.r.t. $A$.

Let $c$ be an encoding of $V$. Then the **encoding length** of an element $v \in V$ is $\mathrm{length}_c\,(v) = \mathrm{length}\,(c(v))$.

Let $E$ be a function ( or **problem**) from a set $\mathfrak{I}$, the set of **instances**, to a set $\mathfrak{O}$, the **output space**.

General Assumption 2.1. *For the following we assume that the encoding for the elements in $\mathfrak{I}$ is fixed and that for all encodings the underlying alphabet is the same.*

Let then $\mathrm{length}\,(\mathfrak{i})$ denote the encoding length for $\mathfrak{i} \in \mathfrak{I}$.

To define complexity classes we need we make some assumptions about the **machine model** we are dealing with. In the sequel all machines or processors will be able to perform

- all basic logical operations such as $\wedge$ , $\vee$ or $\neg$,
- arithmetical operations such as $+$, $-$ , $*$ , mod or /
- comparisons such as $<$, $=$ or $\leq$
- control statements such as **if** or **case**
- loop statements such as **while** or **repeat** /**until**
- read and write operations from resp. on a designated input resp. output device
- read and write operations from resp. into memory

These operations and statements are called **elementary** operations or steps. The operands of these operations will be boolean values or arbitrarily large integers. The memory available will be arbitrarily large.

An **algorithm** on such a machine is a finite sequence of operations.

Definition 2.1. We say that function $E$ is **computable**, if there is an algorithm which

- reads an instance $\mathfrak{i} \in \mathfrak{I}$
- halts
- and writes $E(\mathfrak{i})$.

The **running time** of an algorithm for a specific input $\mathfrak{i}$ is the number of elementary steps executed by the algorithm when it is given $\mathfrak{i}$ as input. One of the main issues of the theory of algorithms is making estimations about the running time of an algorithm, compared to the length of the input.

We are working with the so called **unit cost model**, i.e., we make the following assumption:

General Assumption 2.2. *All elementary operations will require constant time per operation.*

This assumption, though generally used in the literature, is somewhat misleading and inconsistent. Since the minimum amount of space needed to encode an integer $n$ is $\log n$ this time is also needed to read it into memory or to perform an addition, say. So to read an input consisting of $n$ integers into memory we would need time $n \log n$.

Assumption 2.2 is often justified by saying that it is true for any concrete machine. But this is misleading since we want to make theoretical statements about all machines executing a certain algorithm.[1] Then certainly $\log n$ has to be considered as, though slowly, growing function.

On the other hand big efforts are made to avoid extra "log" factors in the running time of algorithms without making exact statements about the machine model that is assumed. We think of that as being inconsistent: on one side counting log more or less as a constant on the other side investing a lot of work into the saving of a "log" factor.

But when we want to make general statements on, for example, polynomiality we will not need to make changes to that model for sequential machines here. Just to be honest and to make clear that we are dealing with an abstract machine model we reformulate Assumption 2.2 as

General Assumption 2.3. *To determine the running time of an algorithm all elementary operations are counted as one time unit.*

Under that assumption we classify problems according to the following definition.

[1]It is easy to see that all algorithms considered on a concrete machine that halt, halt even in constant time.

Definition 2.2. We say that $E \in \mathbf{P}^{\alpha}$ if there is an algorithm which calculates $E(\mathfrak{i})$ in time $O\left(\text{length}\,(\mathfrak{i})^{\alpha}\right)$.

We say that $E$ is **polynomially solvable**, denoted by $E \in \mathbf{P}$, if there is such an $\alpha$ with $E \in \mathbf{P}^{\alpha}$, i.e $\mathbf{P} = \bigcup_{\alpha \in \mathbb{N}} \mathbf{P}^{\alpha}$.

If not stated otherwise machines will be sequential. When considering parallel machines the inconsistency mentioned above becomes more important since the running time here is usually counted in poly-logarithmic time and the cost of a parallel algorithm is then compared with the running time of a sequential one.

We suspect some of the problems arising in this field being originated by this inconsistency. For example the discussion if parallel processors may concurrently read or write into memory loses importance. If we take into account that the addressing of memory uses logarithmically many resources, simple regulations for dealing with concurrency conflicts when two processors want to write into the same memory cell can be handled at the same time. With simple regulations we mean here rules as "processor with higher id wins" or "an arbitrary processor wins". In fact most of the times conflicts occuring in our algorithms will be that several processors try to write the same information into the same memory cell.

So when considering a parallel machine model with an arbitrary number processors we will assume that all of them have parallel random access to a shared memory (PRAM) and that this access can be done concurrently for reading and writing (CRCW) instead of using the CREW (concurrent read exclusive write) model that would be more restrictive.

Definition 2.3. We say that $E \in \mathbf{NC}^{\alpha}$ if there are an algorithm on a CRCW PRAM and a constant $\beta$ depending on the algorithm such that the algorithm calculates $E(\mathfrak{i})$ in time $O\left(\left(\log \text{length}\,(\mathfrak{i})\right)^{\alpha}\right)$ and does not use more than $O\left(\text{length}\,(\mathfrak{i})^{\beta}\right)$ processors.

We say $E \in \mathbf{NC}$ if there is such an $\alpha$, i.e, $\mathbf{NC} = \bigcup_{\alpha \in \mathbb{N}} \mathbf{NC}^{\alpha}$.

If $\mathfrak{O} = \{\mathbf{true}\,, \mathbf{false}\,\}$ $E$ is a **decision problem**. $E$ is **decidable**, $E \in \mathbf{DEC}$, if it is computable.

We say that $E$ is in **NP** if there is a formal language $\mathfrak{C}$ and a property

$$\text{Test}_E\colon\ \mathfrak{I} \times \mathfrak{C} \ \rightarrow\ \{\mathbf{true}\,, \mathbf{false}\,\} \tag{2.1}$$

that fulfills:

(1) If for some $\mathfrak{i} \in \mathfrak{I}$ and $\mathfrak{c} \in \mathfrak{C}$ $\text{Test}_E\,(\mathfrak{i}, \mathfrak{c}) = \mathbf{true}$ then $E(\mathfrak{i}) = \mathbf{true}$ .
(2) There is $\alpha \in \mathbb{N}$ s.t. for all $\mathfrak{i} \in \mathfrak{I}$ with $E(\mathfrak{i}) = \mathbf{true}$ there is $\mathfrak{c}_{\mathfrak{i}} \in \mathfrak{C}$ s.t. $\text{Test}_E\,(\mathfrak{i}, \mathfrak{c}_{\mathfrak{i}}) = \mathbf{true}$ and s.t. $\text{length}\,(\mathfrak{c}_{\mathfrak{i}}) \leq \text{length}\,(\mathfrak{i})^{\alpha}$ .
(3) $\text{Test}_E \in \mathbf{P}$

Observe that 3 means in particular that $\mathrm{Test}_E$ is polynomial even if length $(\mathfrak{i})$ is considered as relevant input length. Think of $\mathfrak{E}$ being a set of valid **evidences** for $E$ and of $\mathrm{Test}_E\,(\mathfrak{i},\mathfrak{e})$ as a test if $\mathfrak{e}$ is an evidence for $E(\mathfrak{i})$. For example consider the following problem

PROBLEM 2.1. NON-REPETITIVE STRING

**Instance:** $\mathfrak{i} \in A^*$
**Question:** Are two positions in $\mathfrak{i}$ identical?

Then $\mathfrak{E} = \mathbb{N}^2$ and

$$\mathrm{Test}_E\,(\mathfrak{i},(v,w)) = \textbf{true} \text{ iff positions } v \text{ and } w \text{ of } \mathfrak{i} \text{ are identical} \tag{2.2}$$

would be a good choice.

The following definition gives a formalism to characterize problems that are at least "as hard" as any problem in **NP**. A problem $E$ is **NP**-hard if for every problem $E' \in$ **NP** there is a **polynomial transformation** to $E$, i.e., there is are algorithms $A$ and $B$ that fulfill

- $A, B \in \mathbf{P}$
- $A$ gets an instance $\mathfrak{i}'$ for $E'$ as input and outputs an instance $\mathfrak{i}$ for $E$
- $B$ gets $\mathfrak{i}'$ and an element of the output space for $E$ as input and outputs **true** or **false** .
- $B(\mathfrak{i}', E(\mathfrak{i})) \iff E'(\mathfrak{i}')$

Notice that for $E$ to be **NP**-hard it is not necessary that it is a decision problem.

$E$ is **NP**-complete if it is **NP**-hard and if $E \in$ **NP**. So these are the decision problems such that any polynomial algorithm to solve one of them would give such an algorithm for every problem in **NP**.

Unlike the other complexity classes defined above it is not clear that the negation of an **NP**-property is in **NP**, too. The reason for that is the asymmetry of statement 1. We say that $E$ is in **co-NP** if $\neg E$ is in **NP**. It is not known whether or not **NP** = **co-NP**, but $\mathbf{P} \subseteq \mathbf{NP} \cap \textbf{co-NP}$.

The name **NP** comes from the fact that this class first was considered as the class of problems which are polynomially solvable on a **nondeterministic** machine. We do not go into the details of that approach. The book [GJ79] of Garey and Johnson is the classical reference for an overview over **NP**.

REMARK 2.1. *If $\mathcal{P}$ is one of the classes **P**, **NC**, $\mathbf{P}^\alpha$, $\mathbf{NC}^\alpha$, **NP**, **co-NP**, **DEC** then $\mathcal{P}$ is countable.*

PROOF. This is clear because of Assumption 2.1 we only have countably many algorithms. □

Remark 2.2. *Let $\mathfrak{P}$ be as above and $E_1$, $E_2 \in \mathcal{P}$ then $\left(E_1 \wedge E_2\right)$ and $\left(E_1 \vee E_2\right) \in \mathcal{P}$.*

Proof. For $\mathfrak{P}$ being one of **P**, **NC**, $\mathbf{P}^\alpha$, $\mathbf{NC}^\alpha$, **DEC** we just have to execute an algorithm for each subproblem $E_1$ and $E_1$ respectively.

For **NP** the set of "evidences" to test $E_1 \wedge E_2$ is $\mathfrak{E}_1 \times \mathfrak{E}_2$ if $\mathfrak{E}_i$ is such a set for $E_i$. For $E_1 \vee E_2$ the disjoint union $\mathfrak{E}_1 \dot{\cup} \mathfrak{E}_2$ can be used. □

The analogous statement for the negation of a property in **NP** is widely suspected to be false. See the remarks on **co-NP** above.

## 3. Order Relations

**3.1. Posets and Quasi Orders.** $P = (V, <)$ is called a **partial order**, partially ordered set, **order** or **poset**, if "$<$" is a transitive ireflexive relation on the set $V$. $P$ is **finite** if $V$ is finite, write $|P| = |V|$.

$Q = \left(V, \leq\right)$ is called a **quasi-order**, **qo** for short, if "$\leq$" is transitive and reflexive.

Posets and qo's are related very closely — every poset defines a qo on its groundset in a natural way

$$v \leq w \iff \Big((v < w) \text{ or } (v = w)\Big) \tag{3.1}$$

and every qo induces a poset on the equivalence classes of the relation

$$v \cong w \iff \Big((v \leq w) \text{ and } (v \geq w)\Big). \tag{3.2}$$

The notation qo is also sometimes used in the literature for relations which are only transitive, reflexive and antisymmetric.

In the following $Q = \left(V, \leq\right)$ always denotes a qo.

A subset $C$ of $V$ is called a **chain** if every pair $v, w \in C$ is strictly related, i.e., $v < w$ or $w > v$ holds. If $Q$ itself is a chain it is also called a **total** order. Denote with $C_k$ the chain with $k$ elements.

A subset $A$ of $V$ is called an **antichain** if no pair of elements in $A$ is related, i.e for all $v, w \in A$

$$(v \leq w) \Longrightarrow (v = w). \tag{3.3}$$

Denote with $A_k$ the antichain with $k$ elements.

Chains resp. antichains are called a **maximal** if they are inclusion maximal with that property.

Let $v \leq w$ be two elements in $Q$. We say that $w$ **covers** $v$, $w$ is a **covering element** of $v$, $w$ is a **immediate successor** of $v$, $w \succ\!\!- v$, if $v \ncong w$ and for all $u$ with $v \leq u \leq v$ we have that $u \cong v$ or $u \cong w$. The set of all covering elements of $v$ is denoted with $\mathrm{ImSucc}(v)$.

A subset $I \subseteq V$ is a **lower** (resp. **upper**) **ideal** if $w \in I$ and $v \leq w$ (resp. $w \leq v$) implies $v \in I$. Observe that if $I$ is a lower ideal then $V \setminus I$ is an upper ideal. For $w \in V$ let the **initial segment** of $w$, denoted by $V^w$(read $V$ below $w$), be the lower ideal $\{v \in V \mid v \leq w\}$ and for $W \subseteq V$ define $V^W$ as $\bigcup\limits_{w \in W} V^w$. We say that $W$ generates $V^W$ and that $W$ is a **generating** set of $V^W$. The **final segment** $V_w$ is defined analogously as upper ideal. A generating set $W$ is called a **basis** of $V_W$ resp. $V^W$ if $V_{W'}$ resp. $V^{W'}$ is a proper subset of $V_W$ resp. $V^W$ for all $W' \subset W$. That means that every $w \in W$ is essentially needed to build up $V_W$.

An order $Q = \Big(V, \leq\Big)$ is a **suborder** of an order $Q' = \Big(V', \leq'\Big)$ if there is an injective mapping $\rho\colon\ V \ \to\ V'$ such that for all $v, w \in V$

$$v \leq w \Longrightarrow \rho(v) \leq' \rho(w). \tag{3.4}$$

$\rho$ is then called an **order preserving map**.

$Q$ is called an **induced** suborder of $Q'$ if "$\Longrightarrow$" can be replaced by "$\iff$" in 3.4.

If $Q$ is a suborder of $Q'$ such that $\rho$ is a bijection, then $Q'$ is called an **extension** of $Q$. $Q$ is then called a **reduction** of $Q'$. $Q'$ is called a **linear** extension if it is a total order. $Q$ and $Q'$ are **isomorphic** if there are order preserving maps $\rho\colon\ V \ \to\ V'$ and $\rho'\colon\ V' \ \to\ V$ which are inverse to each other, i.e., $\rho\Big(\rho'\Big(Q'\Big)\Big) = Q'$ and $\rho'\Big(\rho\Big(Q\Big)\Big) = Q$. In general we will not distinguish between isomorphic objects.

It is clear that the relation "is suborder" denoted by $\underset{sub}{\preceq}$ defines itself a qo on any set of orders. The same holds for the relation "is induced suborder" which we denote by $\underset{ind}{\preceq}$.

It might be a source of confusion for the reader that we deal with relations over relational structures. To help a little bit we adopt the different uses the terms "poset" and "qo" usually have: In general for our discussions the posets will be finite and the qo's will be infinite representing e.g. a set of finite posets. The relations in those qo's will be given by existence of certain morphisms between the objects. Sometimes we will speak of **ordered structures** when we recur to such qo's. This is done to emphasize the fact that the machinery we use is an abstraction of several properties of the objects.

An important example for an order relation is the subset relation between sets. If we have a set $S$ the set of subsets of $S$ equipped with $\subseteq$ is called the **Boolean Lattice** of $S$, denoted with $\mathcal{B}_S$. If $S = \{1, \dots, k\}$ we write $\mathcal{B}_k = \mathcal{B}_S$.

Another example for an order relation occurs for sets of intervals on the real line. If $\left[a, b\right]$ and $\left[c, d\right]$ are two such intervals we say $\left[a, b\right] < \left[c, d\right]$ if $b < c$. An order $P = (V, <)$ for that we can find a set of intervals $\left\{\left[l_v, r_v\right]_{v \in V}\right\}$ of intervals such that the order relation in $P$ and the one of the intervals coincide is called an **interval order**.

**3.2. Well Quasi Orders.** A sequence of elements $(v_i)$ in $Q$ is a called a **descending chain** if $v_i \geq v_j$ for all $i \leq j$. Such a chain is called **stationary** if there is $N$ such that $v_i \cong v_j$ for all $i, j \geq N$. $Q$ is called **well founded** if every descending chain is stationary.

A sequence of elements $(v_i)$ in $Q$ is called **good** if there are $i < j$ such that $v_i \leq v_j$. It is called **bad** if it is not good. It is called **perfect** if $v_i \leq v_j$ for all $i \leq j$.

The reader may easily verify the following theorem. It forms one of the foundations of our discussion — mostly we will not mention it explicitly.

THEOREM 3.1. *Let $P = (V, \leq)$ be a qo. Then the following statements are equivalent:*

(1) *$P$ is well founded and every antichain is finite.*
(2) *Every sequence in $P$ is good.*
(3) *Every sequence has a perfect subsequence.*
(4) *Every upper ideal has a finite basis.*
(5) *Every suborder $P'$ of $P$ has a finite non-empty set of absolute minima and every non-maximal element has a finite, non-empty set of covers.*

DEFINITION 3.1. A qo which fulfills one and thus all equivalent statements in Theorem 3.1 is called a **well-quasi-order**, wqo for short.

wqo's have first been considered implicitly as those qo's having property 4, which is often called the **finite basis** property. For an overview and bibliography on wqo's we refer to the articles of Milner [Mil85] and Pouzet [Pou85] in [GO85], for an historical overview see e.g. [Kru72].

We give some basic examples.

(1) All finite qo's are wqo's.
(2) $\omega = \left(\mathbb{N}, \leq\right)$, the natural order on natural numbers is a wqo.
(3) $\mathbb{N}^2$ with the componentwise ordering is a wqo.
(4) Every order that is a chain and well founded is a wqo.
(5) $\left(\mathbb{Z}, \leq\right)$, the integers, are not a wqo since they don't have an absolute minimum.

(6) $\big(\mathbb{R}, \leq\big)$, the real numbers, are not a wqo since no element has a cover.

### 3.3. Substitution Composition.

DEFINITION 3.2. Let $Q_0 = \big(X, \leq\big)$ be a qo and $\big\{Q_x = \big(V_x, \leq_x\big)\big\}_{x\in X}$ be a family of non-empty qo's with $V_x \cap V_y = \emptyset$ if $x \neq y$. The **substitution composition** $Q_0\left[\{Q_x\}_{x\in X}\right]$ is a qo on the groundset $V_\infty = \bigcup_{x\in X} V_x$ defined by

$$v \preceq w \iff \begin{cases} v \leq_x w & \text{for some } x \in X \\ & \text{or} \\ v \in V_x, w \in V_y & \text{for } x < y \end{cases} \tag{3.5}$$

It is easy to see that this indeed leads to a qo. We obtain

THEOREM 3.2. *$Q_0\left[\{Q_x\}_{x\in X}\right]$ is a wqo iff $Q_0$ and all $Q_x$ are wqo's.*

PROOF. If $Q_0$ or one of the $Q_x$ is not a wqo then clearly $Q_0\left[\{Q_x\}_{x\in X}\right]$ is not since they are induced suborders. In the reverse direction we have to show that any sequence $(v_1, v_2, \dots)$ in $Q_0\left[\{Q_x\}_{x\in X}\right]$ is good. By definition there are $(x_1, x_2, \dots)$ such that $v_i \in V_{x_i}$ for all $i$.

First we assume that there is a subsequence $\big(v_{\rho(i)}\big)$ such that $x_{\rho(i)} = x_{\rho(j)} = x$ for all $i$ and $j$. But then $\big(v_{\rho(i)}\big)$ is a sequence in $Q_x$ and good since $Q_x$ is a wqo.

If there is no such subsequence then there is one such that all $x_{\rho(i)}$ are pairwise distinct. But then $\big(x_{\rho(i)}\big)$ must be good so there are $i$ and $j$ with $x_{\rho(i)} < x_{\rho(j)}$ and by definition we also have $v_{\rho(i)} < v_{\rho(j)}$. So our sequence is good. $\square$

Three special cases of the substitution composition are covered by the following definition.

DEFINITION 3.3. $Q_0\left[\{Q_x\}_{x\in X}\right]$ is called the **parallel** composition of the $Q_x$ if $Q_0$ is an antichain. It is called the **series** composition of the $Q_x$ if $Q_0$ is a chain. It is called a **weak order** if it is the series composition of antichains.

With these definitions we are able to generate some more examples. For this let $A_i$ resp. $C_i$ denote the antichain resp. the chain of order $i$.

(1) $A_2\left[\{\omega, \omega\}\right]$ the parallel composition of $\omega$ with $\omega$ is a wqo.
(2) $\omega\left[\{A_i\}_{i\in\omega}\right]$ is a wqo. This shows that in a wqo the cardinality of the antichains may be unbounded, see Figure 3.1. This is the general situation the reader should have in mind when we will discuss special wqo relations. There is no hope for these relations to have only antichains of bounded size.

(3) $C_2\,[\{\omega, C_1\}]$ is a wqo. This shows that in general there may be infinitely many points below another one, see Figure 3.2. Most of the special relations we will study later will not have that property.

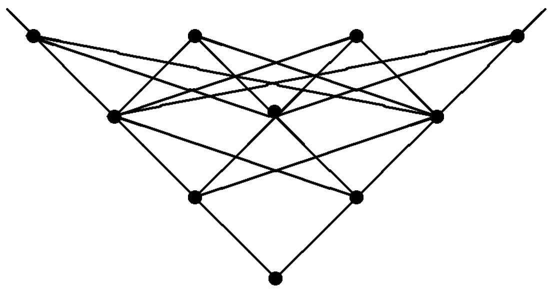

FIGURE 3.1. Arbitrary Large Antichains

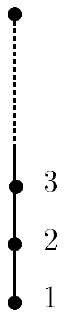

FIGURE 3.2. An infinite ascending chain with limit

## 4. The Axiom of Choice and Its Equivalents

The wellfounded chains in Example (4), p.10, are called **well-orders**, **wo** for short. wo's are a possible way to generalize "counting". One wo is smaller than another one if the first is a suborder of the latter. With that definition every wo has an immediate successor with respect to $\underset{sub}{\prec}$ that is unique up to isomorphism. A construction for this would be an easy generalization of Example (3), p.12.

For that reason wo's are also called **ordinals**. The chains $C_i$ for $i \in \mathbb{N}$ represent all classes of finite ordinals, $\omega$ is then the least infinite ordinal.

In set theory, see e.g. [Ebb79], the following theorem is shown:

THEOREM 4.1. *On the basis of **ZF** the following three statements are equivalent:*

**Axiom of Choice:** *For every set $S$ and every family $\{S_i\}_{i\in I}$ of non-empty subsets of $S$ there is a* **choice function** $\varphi\colon\ I \to S$ *such that $\varphi(i) \in S_i$ for all $i \in I$.*

**Zorn's Lemma:** *Let $Q = (V, \leq)$ be a qo. If every ascending (descending) chain in $Q$ is bounded then there is a maximal (minimal) element in $Q$.*

**Well-Ordering Theorem:** *For every set $S$ there is an order relation $\leq$ such that $(S, \leq)$ is a wo.*

Here **ZF** is the Zermelo-Fraenkel axiom system of set theory including the axiom of foundation. We can not go into the details of that system. We simply remark that this axiom system is one of the equivalent axiomatic formulations of the foundations of modern set theory. We will use the Axiom of Choice resp. Zorn's Lemma frequently.

**4.1. Cartesian Products and Minimal Bad Sequences.** For every family $\{Q_i\}_{i\in I}$ of qo's there is the **componentwise ordering** on the Cartesian product $\times_{i\in I}Q_i$ given by $v \leq w$ if $v_i \leq_{Q_i} w_i$ for all $v = \times_{i\in I}v_i$, $w = \times_{i\in I}w_i$ and $i \in I$.

LEMMA 4.1. *Let $Q = \big(V, \leq_Q\big)$ and $R = \big(W, \leq_R\big)$ be a qo. Then $Q \times R$ is a wqo iff $Q$ and $R$ are wqo's.*

PROOF. "$\Longrightarrow$" Let $v_1, v_2, \ldots$ be a sequence in $Q$. Then $(v_1, w), (v_2, w), \ldots$ for some $w \in W$ is a sequence in $Q \times R$, so it is good. So $v_1, v_2, \ldots$ is good, too.

"$\Longleftarrow$" Let $(v_1, w_1), (v_2, w_2), \ldots$ be a sequence in $Q \times R$. $v_1, v_2, \ldots$ has a perfect subsequence $v_{\rho(1)}, v_{\rho(2)}, \ldots$, say. $w_{\rho(1)}, w_{\rho(2)}, \ldots$ is good so there are $i < j$ such that $w_{\rho(i)} \leq w_{\rho(j)}$. But this shows the claim. □

The following remark is an easy observation

REMARK 4.1. *Let $a, b \in \mathbb{N}$ and $A$ be an antichain in $\omega\times\omega$ such that $\big(a, b\big) \in A$. Then $|A| \leq a + b$.*

A generalization of this fact to $\omega^3$, say, does not hold

REMARK 4.2. *Let $k \in \mathbb{N}$. Then there is an antichain $A$ in $\omega^3$ such that $\big(1, 1, 2\big) \in A$ and $|A| = k$*

Set $A = \Big\{\big(i, k - i, 1\big) \ \big|\ 1 \leq i \leq k - 1\Big\} \cup \Big\{\big(1, 1, 2\big)\Big\}$.

This means that given one element $v \in V^k$ we will not be able to give a bound on the maximal length of a bad sequence where $v$ is the first element.

A special case of Cartesian products are sequences where $I = \omega$ and $Q_i = Q_0$ for all $i \in \omega$. Observe that the infinite Cartesian product $Q^\omega$ with the usual componentwise ordering does only lead to a wqo if $Q$ is the order on one point. This is because $\left(C_2\right)^\omega$ is not well founded and $\left(A_2\right)^\omega$ consists of exactly one infinite antichain. But we obtain another important property of $Q^\omega$.

LEMMA 4.2. *Let $Q$ be a well founded but not a wqo. Then there is a minimal bad sequence in $Q^\omega$.*

PROOF. Let $\left(\left(v_i^j\right)_{i\in\omega}\right)_{j\in\omega}$ be an arbitrary descending chain in $Q^\omega$ of bad sequences in $Q$ i.e

(1) for each $j \in \omega$ we have that $\left(v_1^j, v_2^j \dots\right)$ is a bad sequence in $Q$ and
(2) for each $i \in \omega$ we have that $v_i^1 \geq v_i^2 \geq \dots$

Since $Q$ is well founded $\left(v_i^j\right)_{j\in\omega}$ is stationary for every $i \in \omega$. Let $v_i^\infty$ be the minimum say. The sequence $\left(v_i^\infty\right)_{i\in\omega}$ is obviously a lower bound for

$$\left(\left(v_i^j\right)_{i\in\omega}\right)_{j\in\omega} \tag{4.1}$$

It is bad since for each pair $i_1 < i_2$ there is $j \in \omega$ such that $v_{i_1}^\infty = v_{i_1}^j$ and $v_{i_2}^\infty = v_{i_2}^j$. So $v_{i_1}^\infty \leq v_{i_2}^\infty$ cannot hold since $\left(v_i^j\right)_{i\in\omega}$ is bad.

So we have shown that every descending chain has a lower bound. Zorn's Lemma gives the claim. □

## 5. Hereditary Properties

A property $E$ of the elements of a qo $Q$ is **hereditary** if the subset of elements with that property forms a lower ideal of $Q$ i.e. if

$$\left(E(w) \wedge (v \leq w) \Longrightarrow E(v)\right) . \tag{5.1}$$

Every antichain $A$ defines a hereditary property by

$$\mathrm{Prop}_A(v) = \neg \bigvee_{v_0\in A} (v_o \leq v) \tag{5.2}$$

Indeed if $v \leq w$ and there is $v_0$ with $v_0 \leq v$ then $v_0 \leq w$ holds too. If $A = \{v\}$ we simply write $\mathrm{Prop}_v$.

If $Q$ is also well founded we may also assign to each hereditary property $E$ an antichain in $Q/_\cong$

$$\mathrm{Obstr}_E = Min\,\{[v] \in V/_\cong \mid E(v) = \textbf{false}\,\}, \tag{5.3}$$

the set of (minimal) **obstruction**s of $E$.

Prop and Obstr are inverse:

LEMMA 5.1. *Let $Q$ be well founded. Then $Prop_{Obstr_E} = E$ and $Obstr_{Prop_A} = A$, i.e., Prop and Obstr define a 1-1 correspondence between the set of hereditary properties in $Q$ and the set of antichains of $Q/_{\cong}$.*

We omit the proof which is just a straightforward calculation.

LEMMA 5.2. *Let $Q$ be well founded and let the set of its antichains be countable. Then $Q$ is a wqo.*

PROOF. If $Q$ is not a wqo it has an infinite antichain $A$. All subsets of $A$ are antichains too, so the set of antichains can not be countable. $\square$

CHAPTER II

# Ordered Structures and Complexity Classes

## 6. Quasi Orders and Worst-Case Complexity

**6.1. Proper Encodings.** To relate qo's and complexity classes we have to say something about encodings and the related encoding length for the objects of a given qo.

Observe that if $V$ has an encoding then $|V|$ is at most countable.

Mostly the corresponding encoding of our objects will not be important by itself, so we will simply speak of length$\,(v)$. The property which is important for us is given in the following definition.

DEFINITION 6.1. An encoding of the qo $P = (V, \leq)$ is called **proper** (with resp. to $\leq$) if length is hereditary , i.e., if

$$v < w \Longrightarrow \text{length}\,(v) < \text{length}\,(w) \tag{6.1}$$

All encodings which are commonly used for graphs or posets are proper with respect to all order relations introduced on these objects. If e.g. we encode a graph $G$ by a list of edges clearly every subgraph or induced subgraph of $G$ has a smaler encoding since it has fewer edges.

This justifies

GENERAL ASSUMPTION 6.1. *All qo's in this section will be properly encoded.*

LEMMA 6.1. *Every properly encoded qo is well founded.*

PROOF. Let $c$ be the encoding and $v_1 \geq v_2 \geq \ldots$ be a descending chain of the qo. length$_c\,(v_i)$ is a decreasing sequence so there is $N$ such that for all $i \geq N$ follows length$_c\,(v_i)$ = length$_c\,(v_N) = l_\infty$. $c$ is injective so there are at most $|A|^{l_\infty}$ different $v_i$ with $i \geq N$. So there is an element $v_0$ in the sequence which appears infinitely many times. $\square$

This again justifies

General Assumption 6.2. *Every qo in this section will be properly encoded and thus countable and well founded.*

Remark 6.1. *Let* $Q = \left(V, \leq\right)$ *be a properly encoded qo,* $n \in \mathbb{N}$ *and* $V_n = \{v \in V \mid length(v) = n\}$ *then* $V_n/_{\cong}$ *is an antichain in* $Q/_{\cong}$*.*

In general this implies also that the cardinality of antichains in our qo's will not be bounded. This is because for any "reasonable" encoding the number of elements with an encoding length less than a given number should be a superlinear function.

## 6.2. Well Quasi Orders and Tests for Hereditary Properties.

Theorem 6.1. *Let* $\mathcal{P}$ *be one of the complexity classes* ***P***, ***NC***, $\boldsymbol{P}^{\alpha}$, $\boldsymbol{NC}^{\alpha}$, ***NP***, ***co-NP***, ***DEC*** *and let* $P = (V, \leq)$ *be a properly encoded qo then the following two statements are equivalent:*

(1) *$P$ is a wqo and for every $v \in V$ $Prop_v \in \mathcal{P}$*
(2) *Every hereditary property is in $\mathcal{P}$.*

Proof. First we show $1 \Longrightarrow 2$.

Let $E$ be a hereditary property in $P$. $E$ is characterized by its finite obstruction set $\mathrm{Obstr}_E$, see section 3. Let $Obstr_E = \{v_1, \ldots, v_k\}$. For $v \in V$ we know that

$$E(v) \iff \neg \bigvee_{i=1}^{k} (v_i \leq v) \iff \bigwedge_{i=1}^{k} \neg (v_i \leq v) \iff \bigwedge_{i=1}^{k} \mathrm{Prop}_{v_i}(v) \tag{6.2}$$

But with Remark 2.2 the right hand side is in $\mathcal{P}$ and so is $E$.

Now we show $2 \Longrightarrow 1$. The second part of 1 follows easily since $\mathrm{Prop}_v$ is a hereditary property. So we have to show that $P$ is a wqo.

Assume that this were not the case. $P$ is properly encoded so it is well founded.

So with Lemmas 5.1 and 5.2 there would be more than countably many hereditary properties a contradiction to Remark 2.1. □

Theorem 6.1 should be interpreted very carefully. If we have an interesting property it gives only evidence for a given input **not** having that property. Namely it points to a forbidden substructure which inhibits the property. This substructure belongs to a finite set and there is an algorithm which tests presence of this substructure. So it can be easily used to show that the property is in **co-NP**.

In a sense there is no natural evidence **why** a certain property holds. The fact that the obstruction set for the property is finite is not constructive in general. In most cases it relies on the Axiom of Choice. We will return to that aspect later.

**6.3. Algorithmically Solvable Problems and Well Quasi Orders.** Now we want to show that algorithms on wqo's can be taken as a model for many algorithmical questions. The reason for that is very simple, we easily can define an appropriate qo for any encoded set.

THEOREM 6.2. *Let $E_0 \in \mathcal{P}$ be a problem defined on the encoded set of instances $V$. Then there is an order relation $\leq$ on $V$ such that*

(1) *the encoding is proper,*
(2) *$(V, \leq)$ is a wqo*
(3) *$E_0$ is hereditary with respect to $\leq$*
(4) *every hereditary property $E$ in $(V, \leq)$ is in $\mathcal{P}$.*

PROOF. Define $\leq$ by

$$(v \leq w) \iff \Big(\big(E_0(v) = E_0(w)\big) \wedge \big(\text{length}\,(v) \leq \text{length}\,(w)\big)\Big) \tag{6.3}$$

With that 1 and 3 clearly are fulfilled.

For 2 it remains to show finiteness of the antichains, since the relation is clearly well founded. Let $A$ be an antichain. It consists of two parts

$$A_t = \{v \in A \mid E(v) = \textbf{true}\ \}$$

and

$$A_f = \{v \in A \mid E(v) = \textbf{false}\ \}\,.$$

We show that $A_t$ is finite say.

There is a value $l$ such that for all $v \in A_t$ length$\,(v) = l$, since $v, w \in A$ with distinct encoding length would be related by the definition of $\leq$.

But since the encoding is injective $|A_t|$ must be finite. A similar argument holds for $A_f$ so we have 2.

For 4 observe that $\text{Prop}_v$ is in $\mathcal{P}$. So Theorem 6.1 together with 2 gives 4. □

**6.4. Graph Minors.** The theory of graph minors is **the** example for the use of the machinery given in this section. It was mainly developed by Robertson and Seymour in series of articles called "Graph Minors". We can not go into details of that approach, but we will state the main definitions and results that are relevant for our purposes.

DEFINITION 6.2. Let $G$ and $G'$ be graphs. $G$ is a **graph minor** of $G'$, $G \underset{min}{\preceq} G'$, if it can be obtained from $G'$ by the following three operations:

(1) Delete a vertex.
(2) Delete an edge.
(3) Contract an edge.

The main structural result for graph minors is the following theorem that was previously known as Wagner's Conjecture.

THEOREM 6.3. *The set of finite graphs equipped with* $\underset{min}{\preceq}$ *is a wqo.*

The main step for an algorithmic result for our context is a solution for the following class of problems:

PROBLEM 6.1. $H$-MINOR

**Instance:** Graph $G = (V, E)$
**Question:** Does $G$ contain a minor isomorphic to $H$?

THEOREM 6.4. *For every graph $H$ there is an algorithm to test $H$-*MINOR *in time* $O\left(|V|^3\right)$.

The main result is then an easy consequence of the things said above:

THEOREM 6.5. *For every property $E$ on graphs that is hereditary with respect to* $\underset{min}{\preceq}$ *there is an algorithm to test it in time* $O\left(|V|^3\right)$.

Theorem 6.4 can be improved if $H$ is planar:

- For every planar graph $H$ there is an algorithm to test $H$-MINOR in time $O\left(|V|^2\right)$.
- For every property $E$ on graphs that is hereditary with respect to $\underset{min}{\preceq}$ and such that there is a planar graph $H$ with $\neg E(H)$ there is an algorithm to test it in time $O\left(|V|^2\right)$.

The articles of the Graph Minors Series published in journals until now are Graph Minors I to X. These are [RS83a], [RS86a], [RS83b], [RS90a], [RS86b], [RS86c], [RS88a], [RS90c], [RS90b], [RS91].

Until now XI to XVI circulate as manuscripts, these are [RS85b], [RS86d], [RS86e], [RS87], [RS88b], [RS89].

This theory can be used to solve several problems algorithmically. Besides giving a unified approach to many problems that have been solved before it gives also qualitative improvements on the running time for some of the problems and solves problems where the complexity status was not known. Among the problems with improved running time is

PROBLEM 6.2. $k$-PATHWIDTH

**Instance:** Graph $G$
**Question:** Is there an interval graph $G'$ such that $G$ is isomorphic to a subgraph of $G'$ and the clique size $\omega(G') < k$?

One of the problems where the complexity status was not known is the following — not even membership in **NP** has been proven before.

PROBLEM 6.3. LINKLESS EMBEDDING

**Instance:** Graph $G$

**Question:** Is there an embedding of $G$ into 3-space such that no pair of circles in $G$ forms a link?

Up to now no proof for an obstruction set characterization for this property has been given. So we only have an existence proof of an algorithm. This means if we *knew* that set we *had* and algorithm.

Overviews over Graph Minors are rare and probably not up-to-date, see [RS85a], [RS90d] and [Fel89].

Various authors participated with improvements of algorithms and application of the theory to particular problems. Among them are e.g. [BK91], [Lag90], [Ree91], [FL88c], [FL88b], [FL85], [FKL88], [FL92], [FL88a] and [FL89].

## 7. Average Time Complexity of Hereditary Properties

We want to give a general method to speed up average time complexity of algorithms. By "speeding up" an algorithm we mean the following. Assume we have an algorithm $T_E$ to test a certain property $E$. This algorithm might be expensive. Our aim is to avoid a call to $T_E$ by putting a cheaper algorithm $S_E$ in front.

$S_E$ should give one of two possible answers:

**false** The input does not have property $E$.

**maybe** The input might or might not have property $E$.

As long as the answer "**maybe** " is rare and the running time of $S_E$ is fast, we will gain something by executing $S_E$ first and then $T_E$ only if necessary. We make this more precise by the following definitions.

Let $Q = (V, \leq)$ be a properly encoded qo and

$$V_n = \{v \in V \mid \text{length}\,(v) = n\}\,.$$

We assume for each $n$ that all $v \in V_n$ are equally likely, i.e., there is a uniform distribution $P$ on $V_n$. So $\mathrm{P}\,(\{v\}) = 1/|V_n|$ for all $v \in V_n$.

We want to formalize a choice of several independent and small substructures of of a large object in $V$. For example we want to chose several independent subgraphs of a certain size from a graph. A **sample** $X$ for $Q$ is a family of random variables $X_{n,\mu,i}\colon\ V_n \;\rightarrow\; V_\mu$ with the following properties for all $\mu \in \mathbb{N}$:

(1) $X_{n,\mu,i}(v) \leq v$

(2) There is a non-decreasing function $l_\mu(.)$ such that for each $n \in \mathbb{N}$ the set of random variables $\left\{X_{n,\mu,1}, \ldots, X_{n,\mu,l_\mu(n)}\right\}$ is independent .

(3) There is a probability $0 < p_\mu < 1$ such that for all $n \in \mathbb{N}$, $v_0 \in V_\mu$ and all $0 < i \leq l_\mu(n)$ the random variables $X_{n,\mu,i}$ fulfill

$$P\left(X_{n,\mu,i} = v_0\right) \geq p_\mu$$

We call $l_\mu$ the **sample length** and $p_\mu$ the **sample probability** of $X$. With $q_\mu$ we denote $1 - p_\mu$. $q_\mu$ is an upper bound for $P\left(X_{n,\mu,i} \neq v_0\right)$.

The following lemma estimates the probability that a certain obstruction $v_0$ appears as substructure of an element $v$.

LEMMA 7.1. *Let $X$ be a sample on $Q$, $v_0 \in V_\mu$ and $v \in V_n$. Then* $P\left(v_0 \not\leq v\right) \leq (q_\mu)^{l_\mu(n)}$

PROOF. This follows directly from the independence of the set of random variables $\left\{X_{n,\mu,1}, \ldots, X_{n,\mu,l_\mu(n)}\right\}$ since

(7.1)
$$\mathrm{P}\left(v_0 \not\leq v\right) \leq \mathrm{P}\left(v_0 \neq X_{n,\mu,i}(v) \mid i = 1, \ldots, l_\mu(n)\right) \leq \prod_{i=1}^{l_\mu(n)} \mathrm{P}\left(v_0 \neq X_{n,\mu,i}(v)\right)$$

□

A property $E$ on $Q$ is called **sparse** if

$$\lim_{n\to\infty} \frac{|\{v \in V_n \mid E(v) = \textbf{true}\}|}{|V_n|} = 0. \tag{7.2}$$

It is **dense** if this fraction tends to 1.

The following theorem shows that hereditary properties are sparse in a very general setting.

THEOREM 7.1. *Let $Q = (V, \leq)$ be properly encoded with sample $X$ such that the sample length $l_\mu(.)$ is unbounded for every $\mu$. Then every non-trivial hereditary property in $Q$ is sparse.*

PROOF. Let $E$ be a hereditary property. Since it is non-trivial there is some $v_0 \in V_\mu$ for some $\mu$ such that $\neg E(v_0)$ holds. Then for all $v \in V_n$ with $E(v)$ we have that $v_0$ is not below $v$, $v_0 \not\leq v$. So

$$P\left(E(v) = \textbf{true}\right) \leq P\left(v_0 \not\leq v\right) \leq (q_\mu)^{l_\mu(n)} \tag{7.3}$$

Since $l_\mu(.)$ is unbounded this shows the claim. □

A **sample algorithm** $A_X$ is an algorithm that incrementally produces a sample $X$. We assume that such an algorithm is implemented as two distinct subroutines. The first one performs some initialization and the other one is given in such a way that for all $0 < i \leq l_\mu(n)$ the $i$-th call of this routine outputs $X_{n,\mu,i}(v)$.

We denote with $t^{init}_{\mu,A}$ and $t^{inc}_{\mu,A}$ the time such an algorithm needs for an initial phase and for each incremental step respectively.

Now let $E$ be a hereditary property and $v_0 \in V_\mu$ be an obstruction for $E$, i.e., $E$ is false on $v_0$. In addition assume we are given an algorithm $T_E$ that outputs $E(v)$ with running time $t_{T_E}(n)$ if $n = \text{length}\,(v)$.

Consider the known test routine $T_E$ as being expensive; $t_{T_E}(n)$ grows faster than we want. Here "growing fast" can mean different things:

- super-polynomial,
- linear (or low polynomial) with enormous constants of proportionality or
- super-polylogarithmic,

depending on the setting we want to deal with. The following algorithm implements a strategy to avoid the call to this routine. It simply puts the test whether or not $v_0 = X_{n,\mu,i}(v)$ for some $i$ in front of $T_E$.

ALGORITHM 7.1. $\texttt{average}_{A_X,T_E,v_0}$

```
Input:  $v \in V$
Output: $E\,(v)$
 (1)  $n :=$ length $(v)$
 (2)  Initialize $A_X$ with $v$ and $n$
 (3)  for $i := 1$ to $l_\mu\,(n)$ do begin
 (4)      $X_i := A_X$
 (5)      if $(X_i = v_0)$ then begin
 (6)          $Output :=$ false
 (7)          stop
 (8)          end
 (9)      end
(10)  $Output := T_E(v)$
(11)  stop
```

LEMMA 7.2. *The average complexity of* $\texttt{average}_{A_X,T_E,v_0}$ *is in*

$$O\left(t^{init}_{\mu,A} + t^{inc}_{\mu,A} + q_\mu^{l_\mu(n)} \cdot t_{T_E}(n)\right)$$

PROOF. The first term is obvious. The third term is just the probability that $T_E$ is executed multiplied with its running time. For the second term observe that the probability for the $i$-th execution of the for loop is bounded by $q_\mu^{i-1}$.

So the average time this loop needs is

$$O\left(\sum_{i=1}^{l_\mu(n)} q_\mu^{i-1} \cdot t_{\mu,A}^{inc}(n)\right) = O\left(t_{\mu,A}^{inc}(n) \cdot \sum_{i=1}^{l_\mu(n)} q_\mu^{i-1}\right) = O\left(t_{\mu,A}^{inc}(n) \cdot \frac{1}{p_\mu}\right). \tag{7.4}$$

But $p_\mu$ is a constant depending only on $v_0$ and not on $v$. □

We conclude with the following theorem:

THEOREM 7.2. *Let $E$ be a hereditary property on $Q = (V, \leq)$ and $A_X$ be a sample algorithm for $Q$. Suppose there is an algorithm $T_E$ to test $E$ that has worst case running time $t_T(n) = O\left(\left(1/q_\mu\right)^{l_\mu(n)}\right)$ for all $\mu$. Then $E$ can be tested in average time $O\left(t_{\mu,A}^{init} + t_{\mu,A}^{inc}\right)$.*

PROOF. We may assume that $E$ is non-trivial, i.e., it has an obstruction $v_0 \in V_\mu$ for some $\mu$. Consider the running time of $\texttt{average}_{A_X,T_E,v_0}$.

The third term of the complexity given in Lemma 7.2 is

$$O\left(q_\mu^{l_\mu(n)} \cdot t_T(n)\right) \leq O\left(q_\mu^{l_\mu(n)} \cdot \left(1/q_\mu\right)^{l_\mu(n)}\right) \leq O(1). \tag{7.5}$$

That shows the claim. □

## 8. Average Time Complexity of Graph Properties

We will exemplify this approach for average time complexity with the set $\mathfrak{G}_{ind}$ of finite graphs ordered by the induced subgraph relation $\underset{ind}{\preceq}$.

All algorithms for hereditary properties in $\mathfrak{G}_{ind}$, for which we found good average time algorithms in the literature, rely on investigations of the properties themselves. They usually test the property on some induced subgraphs — estimations of the average running time then are made by estimations about the number of yes-instances for the property. See e.g. [Wil84] and [PS92b].

### 8.1. Representations of Graphs.

We will use the following notation.

The vertices of a graph $G = (V, E)$ are denoted with $v_0, \ldots, v_{n-1}$. For an index set $\{i_1, \ldots, i_k\}$ we denote with $G_{\{i_1,\ldots,i_k\}}$ the subgraph induced by the vertices $\{v_{i_1}, \ldots, v_{i_k}\}$ and with $G_{[i,j]}$ the subgraph of $G$ induced by the vertices $v_i, v_{i+1}, \ldots, v_j$.

There are several different datastructures that are commonly used for graphs. The main differences among them are the way we obtain the information whether or not an edge is present in the graph and the space needed to store the graph.

We will denote with $\operatorname{retr}(G,i,j)$, **retrieval time**, the time to retrieve the edge $\{i,j\}$.

(1) We may have **random access** to the information, i.e., each query costs $O(1)$. This is commonly implemented by using a matrix, the **adjacency matrix**, to store the particular information. The space needed is then $O(n^2)$.
(2) We may have access via lists.
    (a) For each vertex $v_i$ there is a list of its outgoing edges. These types need $O\left(n+|E|\right)$ space. The lists may be
        (i) sorted according to $j$. Then $\operatorname{retr}(G,i,j) = O(j)$.
        (ii) unsorted. Then $\operatorname{retr}(G,i,j) = O(n)$.
    (b) There is a global list for all edges. These types need $O\left(|E|\right)$ space. Here we also may distinguish two types. The list may be
        (i) sorted according to $i$ and $j$. Then $\operatorname{retr}(G,i,j) = O(i\cdot n + j)$.
        (ii) unsorted. Then $\operatorname{retr}(G,i,j) = O\left(|E|\right)$.
(3) The graph may be given by an oracle, i.e., a routine that gives the information required. Here we may distinguish analogous types as for lists with the same retrieval costs, counted in the number of times we need to call our oracle. Clearly we can not say anything about space needed in that model.

The worst case retrieval cost of a graph is denoted with $\operatorname{retr}(G) = \max\limits_{i,j} \operatorname{retr}(G,i,j)$.

We also need some more definitions when we want to deal with average time complexity of algorithms. We chose the simplest probability model that is in use for graphs: We consider only graphs $G = (V,E)$ over a **fixed** set of vertices $\{v_i \mid 0 \le i\}$. We then have that $V = \{v_0, \dots v_{n-1}\}$ for some $n$.

In particular we will **distinguish** isomorphic graphs the isomorphism between them does not induce the identity on the vertices.

An induced subgraph $G_S$ of a graph $G$ is obtained in the following way: If $S = \{i_0, \dots, i_{\mu-1}\}$ such that $i_0 < i_1 < \dots < i_{\mu-1}$, then $G_S$ is the graph with vertices $v_0, \dots, v_{\mu-1}$ and an edge between $v_j$ and $v_k$ iff $\left\{v_{i_j}, v_{i_k}\right\} \in E(G)$. See Figure 8.1 for an example.

Let $edge_{i,j}(G)$ denote the random variable that is true if in $G$ there is an edge between $v_i$ and $v_j$. Then $P\left(edge_{i,j}(G) = \textbf{true}\right) = \frac{1}{2}$. In addition we will assume that the set of random variables $\{edge_{i,j}(G) \mid i<j\}$ is independent.

We will denote with the **access time**, $\operatorname{acc}(G)$ as

$$\operatorname{acc}(G) = \max_{i,j} \frac{\operatorname{retr}(G,i,j)}{\min\{i,j\}+1}. \tag{8.1}$$

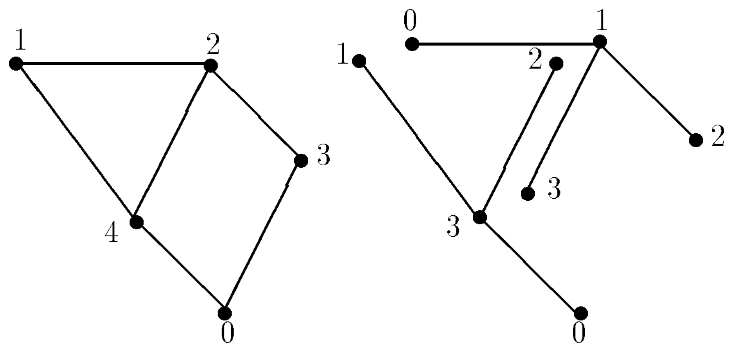

FIGURE 8.1. Two distinct induced subgraphs

This function is a "normalized" cost function, it measures the access to "small" edges. It depends on the datastructure by which the graph is given. According to our different representations of graphs we obtain.

**1:** $G$ is given as adjacency matrix. Then $\mathrm{acc}(G) = O(1)$.

**2(a)i:** $G$ is given as sorted lists of edges. Then $\mathrm{acc}(G) = O(1)$, too.

**2(a)ii:** $G$ is given as unsorted lists of edges. Then $\mathrm{acc}(G) = O(n)$ since then we have to check the whole list to know if there is an edge adjacent to vertex $v_1$.

**2(b)i:** $G$ is given as one sorted list of edges. Then $\mathrm{acc}(G) = O(n)$, too.

**2(b)ii:** $G$ is given as one unsorted list of edges or, equivalently, as an oracle that successively outputs the next edge. Then $\mathrm{acc}(G) = O(n^2)$.

**8.2. Average Time Complexity of Induced Graph Properties.** According to section 7 we want to give a sample algorithm for the set $\mathfrak{G}_{ind}$ of finite graphs ordered by the induced subgraph relation $\underset{ind}{\preceq}$.

With that sample algorithm we will show

THEOREM 8.1. *Let $E$ be a hereditary property on $\mathfrak{G}_{ind}$ and assume there is an algorithm $T_E$ to test $E(G)$ for a graph $G = (V, E)$ with $n = |V|$ in time $2^{O(n)}$.*

*Then there is an algorithm to test $E$ in average time $O(acc(G))$*

The arguments we will give also show the following corollary that gives a $O(1)$ average complexity if we have fast access to "small" edges of the graph. Clearly this only makes sense if we do not have to read the graph any more, for example if we want to use our algorithm as subroutine for other problems.

COROLLARY 8.1. *Let $E$ be a hereditary property on $\mathfrak{G}_{ind}$ and assume there is an algorithm $T_E$ to test $E(G)$ for a graph $G = (V, E)$ with $n = |V|$ in time $2^{O(n)}$. Provided that the input graph $G$ is present either as sorted lists of edges or as adjacency matrix there is an algorithm that needs $O(1)$ time in average.*

From this we will obtain fast average time algorithms for many problems.

COROLLARY 8.2. *Let $k$ be a fixed constant. The following properties of a graph $G$ can be calculated in $O(acc(G))$ average time*

(1) $pw(G) \leq k$
(2) $tw(G) \leq k$
(3) $\chi(G) \leq k$
(4) $\omega(G) \leq k$
(5) $\alpha(G) \leq k$
(6) $k(G) \leq k$
(7) *perfectness of $G$*

Here

| | |
|---|---|
| $\mathrm{pw}(G)$ | is the **pathwidth** of $G$, the minimal clique size of an interval graph $G'$ such that $G$ is isomorphic to a subgraph of $G'$ minus 1, see also $k$-PATHWIDTH on p. 20, |
| $\mathrm{tw}(G)$ | is the **treewidth** of $G$, the minimal clique size of a chordal graph $G'$ such that $G$ is isomorphic to a subgraph of $G'$ minus 1, |
| $\chi(G)$ | is the **chromatic number** of $G$, the minimum number of colors needed to color $G$, |
| $\omega(G)$ | is the **clique size** of $G$, the maximal size of a clique in $G$, |
| $\alpha(G)$ | is the **stability number** of $G$, the maximal size of an independent (stable) set of $G$, |
| $\mathrm{k}(G)$ | is is the **clique cover number**, the minimal number of cliques neeeded to cover $G$. |

The proof of this corollary will occupy a whole section, see section 8.4 below.

**8.3. A Sample Algorithm for Induced Subgraphs.** An easy sample algorithm for this kind of properties would be to take all subgraphs induced by the vertex sets $v_{i\cdot\mu+1}, \dots, v_{(i+1)\cdot\mu}$. But this would only give $\left\lfloor \frac{n}{\mu} \right\rfloor$ induced subgraphs and would thus not be sufficient to prove Theorem 8.1 with help of Theorem 7.2.

For that purpose we have to give a sample algorithm such that the corresponding sample length $l_\mu$ dominates $c \cdot n$ for every constant $c$.

ALGORITHM 8.1. `sample`$^{\mathtt{rec}}{}_{\mu}$, recursive version

**Input:** A graph $G$ with vertex set $\{v_0, \dots, v_{n-1}\}$, $n = \mu^k$ for some $k$.
**Output:** A sequence $\left(H_1, \dots, H_{l_\mu(n)}\right)$ of induced subgraphs of $G$ all having $\mu$ vertices.

```
(1)   $m := n/\mu$
(2)   if $(k > 1)$ then begin
```

```
(3)         for $i := 0$ to $\mu - 1$ do begin
(4)             $\mathtt{sample^{rec}}_{\mu}\left(G_{[i\cdot m,(i+1)\cdot m-1]}\right)$
(5)             end
(6)         end
(7)     for $i := 0$ to $m - 1$ do begin
(8)         Output $:= G_{\{i,i+m,i+2m,\dots,i+(\mu-1)m\}}$
(9)         end
```

We assume that we throw away superficial vertices if our input graph has a number of vertices that is not a power of $\mu$.

LEMMA 8.1. *The output of $\mathtt{sample^{rec}}_{\mu}(G)$ defines a sample $X$ with sample length $l_\mu(n) = k \cdot \mu^{k-1}$ if $k = \left\lfloor \log_\mu n \right\rfloor$.*

PROOF. We show that the graphs $\left(H_1, \dots H_{l_\mu(n)}\right)$ are independent choices. This is clear if we restrict ourselves to the set given in loop (7). To see independence of the whole set observe that the subgraphs that are given as input to the recursive call have no edge in common.

To see that the sample length $l_\mu(n) = k \cdot \mu^{k-1}$ apply induction on $k$. □

It is clear that this algorithm can be implemented in such a way that it has a total running time of $O\left(\mu^2 \cdot k \cdot \mu^{k-1}\right)$ if the adjacency matrix of $G$ is given. This means in particular that we used $O(n \log n)$ edges for the subgraphs out of $O(n^2)$ that were possible.

We need an analysis that is a little more detailed. For that we give an iterative variant of our algorithm.

ALGORITHM 8.2. $\mathtt{sample^{iter}}_{\mu}$, iterative version

**Input:** A graph $G$ with vertex set $\{v_0, \dots, v_{n-1}\}$, $n = \mu^k$ for some $k$.

**Output:** A sequence $\left(H_1, \dots H_{l_\mu(n)}\right)$ of induced subgraphs of $G$ all having $\mu$ vertices.

```
(1)     $r := 1$
(2)     while $(r < n)$ do begin
(3)         $i_0 := 0$
(4)         while $(i_0 < n)$ do begin
(5)             for $i := i_0$ to $i_0 + r - 1$ do begin
(6)                 Output $:= G_{\{i,i+r,i+2r,\dots,i+(\mu-1)r\}}$
(7)                 end
(8)             $i_0 := i_0 + \mu \cdot r$
```

```
(9)          end
(10)     $r := \mu \cdot r$
(11)     end
```

The following is an easy observation

REMARK 8.1. *Both versions of* `sample`$^{rec}{}_{\mu}$ *take the same set of subsets of* $[n]$ *to produce the same set of induced subgraphs as output.*

The reason why we gave the iterative version is the following lemma that would not be true for the recursive one.

LEMMA 8.2. *The iterative version of* `sample`$^{iter}{}_{\mu}$ *can be implemented in such a way that*

(1) *it needs constant time for initialization*
(2) *it needs time* $O\left(\mu \cdot \left(i+\mu\right) \cdot acc(G)\right)$ *to generate the* $i$*-th sample subgraph if* $0 \le i < n/\mu - 1$.
(3) *it needs time* $O(\mu^2)$ *to generate each other subgraphs.*

PROOF. Statement 1 is clear.

To see 2 observe that the subgraphs in question are those induced by the subsets $\{v_{i\cdot\mu}, \dots, v_{i\cdot\mu+\mu-1}\}$. We can access all edges adjacent to vertex $v_i$ that are needed in time $O\left(\left(i+\mu\right) \cdot \mathrm{acc}(G)\right)$. This shows 2.

During the generation of these subgraphs we may build up the adjacency matrix of $G$ since we access all pairs $v_i$, $v_j$ with $i < j$ and know whether or not they share an edge. This shows 3. □

PROOF OF THEOREM 8.1. We have to revisit the proof of Lemma 7.2, i.e., the estimation of the average time complexity of the algorithm `average`.

The second term of the running time now reads

$$
\begin{aligned}
&O\left(\sum_{i=1}^{l_\mu(n)} q_\mu^{i-1}\cdot\mu\cdot\left(i+\mu\right)\cdot\operatorname{acc}(G)\right)\\
&=O\left(\sum_{i=1}^{l_\mu(n)} i\cdot q_\mu^{i-1}\cdot\operatorname{acc}(G)\right)\\
&=O\left(\frac{\operatorname{acc}(G)}{\left(1-q_\mu\right)^2}\right)\\
&=O\left(p_\mu^{-2}\cdot\operatorname{acc}(G)\right)
\end{aligned}\tag{8.2}
$$

This is $O(\operatorname{acc}(G))$ since $p_\mu$ is a constant. □

The previous proof also gives an indication how the constants of proportionality look like. According to the proof of Theorem 7.2 the expensive algorithm $T_E$ contributes only an additive constant to the average time of $\texttt{average}_{\texttt{sample}_\mu,T_E,v_0}$. By making the constant of proportionality in Theorem 8.1 large enough, we can ensure that the running time of $T_E$ is properly bounded by $2^{c\cdot n}$, say. So the average contribution of $T_E$ to the running time can be universally bounded — the bound depending only on our specific machine model.

Corollary 8.3. *There is a universal constant $C$ such that for every hereditary property $E$ in $\mathfrak{G}_{ind}$ that has a test algorithm $T_E$ as considered in Theorem* 8.1 *and that has an obstruction consisting of $\mu$ vertices can be tested in average time $t(G)$ with*

$$
t(G)\le\begin{cases} C\cdot 2^{\binom{\mu}{2}}\cdot acc(G) & \text{if } acc(G)=O\left(retr(G)\right).\\ C\cdot 2^{2\binom{\mu}{2}}\cdot acc(G) & \text{if } acc(G)=O\left(retr(G)/n\right)\end{cases}\tag{8.3}
$$

This shows that if we have a fast datastructure to access edges with small endpoints, we may gain an average time speedup. This speedup implies a growth of the constant of proportionality, which seems to be acceptable when $\mu$ is small.

The proof of this corollary follows directly from what is said above, so we omit it.

**8.4. Special Properties — Proof of Corollary 8.2.** Now we come to Corollary 8.2, i.e., we want to show that several graph properties fit into our setting. The properties we investigate are just a small subset of what is possible — the subset chosen is more or less arbitrary and mainly motivated to give the reader a better insight.

Observe that the properties given have very different worst-case complexity. It varies from well behaved polynomial, i.e. $k$ not in the exponent, via **NP**-complete, to problems for which the complexity status is not known.

The "easiest" cases are those of pathwidth and treewidth. They are both **NP**-complete if $k$ is part of the input as was shown by Arnborg, Corneil and Proskurowski, see [ACP87], and remain so even if the input graph is very restricted, see e.g. [Gus89].

But if $k$ is a fixed constant we have fast algorithms to test whether or not $\mathrm{tw}(G) \leq k$. The fastest realistic algorithm for $\mathrm{tw}(G) \leq k$ was given in [Ree91]. It has a worst case complexity of $O(n \log n)$. So it fits well into our setting if the graph is given as sorted list of edges, say. We then obtain an average complexity of $O(1)$.

For pathwidth, the fastest algorithm to test $\mathrm{pw}(G) \leq k$ is given by the theory of Robertson and Seymour and runs in $O(n^2)$ This is so since there are planar graphs (e.g. trees) with $\mathrm{pw}(G) > k$ for every $k$. A more practical algorithm that we could use is one given by a dynamic programming technique in [ACP87]. But according to Theorem 8.1 any such algorithm has $O(1)$ average complexity if we embed it into our setting.

Now we consider $\omega(G) \leq k$. This problem is again **NP**-complete if $k$ is part of the input. But in contrast to the two previous problems there is no (even theoretical) algorithm known to solve the problem on fixed $k$ in time $O(n^\alpha)$, $\alpha$ not depending on $k$.

In the following we denote the set $\{0, \ldots, n-1\}$ by $[n]$.

ALGORITHM 8.3. $T_{\omega,k}$

```
 Input:   Graph $G$ with $n$ vertices and positive integer $k$
Output:   true if $\omega(G) \leq k$, false otherwise.
   (1)    for all $S\colon \left[k+1\right] \rightarrow [n]$ do begin
   (2)        if $G_S = K_{k+1}$ then begin
   (3)            Output := false
   (4)            stop
   (5)            end
   (6)        end
   (7)    Output := true
```

This algorithm can be implemented such that the running time is $O\left(n^{k+1}\right)$. So Theorem 8.1 can be applied easily — $\texttt{average}_{\texttt{sample}_k, T_{\omega,k}, K_{k+1}}$ has the right average time complexity, if $k$ is fixed.

Since this algorithm itself is just looking for the only minimal obstruction $K_{k+1}$, we even don't have to apply our sample algorithm. We simply have to warrant that the enumeration in line (1) is done in the right order.

Though $\chi(G)$ seems to be very similar to $\omega(G)$, it is not. As we have seen $\omega(G) \leq k$ can be solved in polynomial time if $k$ is a fixed constant. In contrast to that the problem $\chi(G) \leq k$ is **NP**-complete even is $k \geq 3$ is a fixed constant, see [NR87]. It was first solved in average time $O(1)$ by Wilf [Wil84], see also [BW85] and [Wil86].

ALGORITHM 8.4. $T_{\chi,k}$

```
Input:   Graph $G$ with $n$ vertices and positive integer $k$
Output:  **true** if $\chi(G) \leq k$, **false** otherwise.
   (1)   **for** all $\phi\colon [n] \rightarrow [k]$ **do begin**
   (2)       **if** $\phi$ is an admissible coloring of $G$ **then begin**
   (3)           Output := **true**
   (4)           **stop**
   (5)           **end**
   (6)       **end**
   (7)   Output := **false**
```

This brute force algorithm has a running time that is $O(n^2 \cdot k^n)$, so it fits into our setting if $k$ is fixed. Clearly nobody would try to attack the problem like this, if she/he would be really interested in the chromatic number of a specific graph. It only makes sense, if we investigate many graphs — graphs that we presume as all being randomly chosen, independent of each other.

The minimal obstruction we would use is clearly $K_{k+1}$ again.

The reader may easily give analogous algorithms for $\alpha(G)$ and $\mathrm{k}(G)$. The first thing that comes in mind is to apply the algorithms above to $\bar{G}$, the complementary graph of $G$. But this is not a good idea, since it would destroy the average case complexity in general.

Now we give an algorithm to test whether or not a graph is perfect. A graph is called **perfect** if $\chi(G') = \omega(G')$ for all $G' \underset{ind}{\preceq} G$. It seems that for this problem no good average time algorithm has been published before, but that Steger, [Ste92], independently found such an algorithm that relies on the estimation of the number of perfect graphs that was given in [PS92a], and that we do not need for our approach.

This problem is chosen as example, for which we need a much more sophisticated algorithm. The basic idea is to use Lovász's Perfect Graph Theorem, namely that

$G$ being perfect is equivalent to

$$\omega(G') \cdot \alpha(G') \geq \left|V\left(G'\right)\right| \tag{8.4}$$

for all $G' \underset{ind}{\preceq} G$, see the book of Golumbic [Gol80] for more details and references.

ALGORITHM 8.5. $T_{perf}$

```
Input:    Graph $G$ with $n$ vertices
Output:   **true** if $G$ is perfect, **false** otherwise.
(1)       for all $G' \underset{ind}{\preceq} G$ with $\leq 2$ vertices do initialize $\omega[G']$ and $\alpha[G']$
(2)       for $i := 3$ to $n$ do begin
(3)           for all $G' \underset{ind}{\preceq} G$ with $i$ vertices do begin
(4)               $\omega[G'] := 0$ ; $\alpha[G'] := 0$
(5)               $allclique :=$ true ; $allstable :=$ true
(6)               for all $H \underset{ind}{\preceq} G'$ with $i-1$ vertices do begin
(7)                   if $\omega[H] \neq i-1$ then $allclique :=$ false
(8)                   if $\alpha[H] \neq i-1$ then $allstable :=$ false
(9)                   $\omega[G'] := \max\{\omega[H], \omega[G']\}$
(10)                  $\alpha[G'] := \max\{\alpha[H], \alpha[G']\}$
(11)                  end
(12)              if $allclique$ then $\omega[G'] := i$
(13)              if $allstable$ then $\alpha[G'] := i$
(14)              if $\left(\omega[G'] \cdot \alpha[G'] < i\right)$ then begin
(15)                  Output := false
(16)                  stop
(17)                  end
(18)              end
(19)          end
(20)      Output := true
```

LEMMA 8.3. *$T_{perf}(G)$ is correct and has running time $O(n \cdot 2^n)$.*

PROOF. First we show correctness. It is clear that any induced subgraph $G'$ of $G$ is only visited if all its induced subgraphs $H$ have been visited before. Lines (4) to (13) calculate $\omega(G')$ and $\alpha(G')$. If $G'$ is a clique or independent set all its induced subgraphs $H$ are so. This lets us detect whether or not $G'$ has such a structure.

If it is not a clique or independent set then one of its induced subgraphs $H$ contains a maximal clique or a maximal independent set respectively. So we just have to calculate the maximum over all such subgraphs.

Thus we calculate $\omega(G')$ and $\alpha(G')$ correctly. But if we know both values we can easily apply Lovász's Theorem, and so we have correctness.

For the running time observe that the two outer loops together give the factor $2^n$. The inner loop gives $O(n)$, so we have in total $O(n \cdot 2^n) = O\left(2^{\log n} \cdot 2^n\right) = O\left(2^{n+\log n}\right) = 2^{O(n)}$. □

CHAPTER III

# Some Well Known QO's revisited

Now we revisit four well-known qo's, given by relations on antichains, strings, trees and special classes of posets. We investigate these ordered structures for different reasons. The relation on antichains will be useful to show that certain relations lead to wqo's. The others then are given merely to exemplify some of the tools we developed until then. We will use the relation on antichains to show that they are wqo's and then Theorem 6.1 to show existence of algorithms for hereditary properties.

Then we use these results on trees to show that any class of posets with bounded decomposition diameter forms a wqo and admits linear time tests for hereditary properties.

Our approach differs from other general approaches known for "tree-like" structures, e.g. those of Arnborg et al. [ALS91] or Courcelle et al. [CM92]. These authors constructively give algorithms for problems that can be formulated with certain logics. This constructiveness is on one side an advantage since, in principle, it is possible to build a "compiler" that gets a logic formula as input and outputs a recognition algorithm for the corresponding property. The advantage of our approach is that we can show existence of algorithms for problems where constructive proofs might not exists.

## 9. A QO on Antichains

We want to relate the obstruction sets of two properties $E_1$ and $E_2$ where one implies the other. The following definition is then motivated by the observation that if $a_2$ is a minimal obstruction for $E_2$ then there is $a_1 \in \operatorname{Obstr}(E_1)$ with $a_1 \leq a_2$. It will give us a comfortable mechanism to prove that some qo are indeed wqo.

Definition 9.1. Let $A_1$ and $A_2$ be antichains in the qo $Q = \Big(V, \leq\Big)$. We write

$$\Big(A_1 \underset{anti}{\preceq} A_2\Big) \iff \bigwedge_{a_2 \in A_2} \bigvee_{a_1 \in A_1} (a_1 \leq a_2) \tag{9.1}$$

Notice that $\underset{anti}{\preceq}$ is a little bit counter-intuitive since $A \subseteq A'$ implies that $A' \underset{anti}{\preceq} A$.

It seems that a related relation between antichains was first used by Dilworth in [Dil58], cf. also [Beh88] or [Reu91]. Indeed it is used for the maximal antichains of a finite poset. On these objects the relation given here and the one where we would exchange the quantifiers in 9.1 coincide. We will see below that this relation on antichains is equivalent to a relation on lower ideals that was considered before by various authors, see e.g. [DPR81] or [LMP85].

Lemma 9.1. *$\underset{anti}{\preceq}$ is a qo relation on the set of antichains.*

Proof. Reflexivity is clear.

$\underset{anti}{\preceq}$ is transitive: Let $A_1 \underset{anti}{\preceq} A_2 \underset{anti}{\preceq} A_3$. For all $a_3 \in A_3$ there is $a_2 \in A_2$ and a $a_1 \in A_1$ with $a_1 \leq a_2 \leq a_3$. So we have $a_1 \leq a_3$, too. This gives transitivity.

$\underset{anti}{\preceq}$ is antisymmetric. Suppose we have $A_1 \underset{anti}{\preceq} A_2$ and $A_2 \underset{anti}{\preceq} A_1$. We show that $A_2 \subseteq A_1$.

For all $a_2 \in A_2$ there is $a_1 \in A_1$ with $a_1 \leq a_2$. There is also $a_2' \in A_2$ with $a_2' \leq a_1$. So $a_2' \leq a_1 \leq a_2$. But $A_2$ is an antichain so $a_2' = a_2 = a_1$. This shows $A_2 \subseteq A_1$. $A_1 \subseteq A_2$ follows by symmetry. $\square$

Lemma 9.2. *Let $E_1$ and $E_2$ be hereditary properties in $Q = \Big(V, \leq\Big)$, $A_i = Obstr_{E_i}$, $V_i = \{v \in V \mid E_i(v) = \textbf{true}\}$ and $\bar{V}_i = V \setminus V_i$ the corresponding lower and upper ideals. Then the following statements are equivalent.*

(1) $E_1 \Longrightarrow E_2$
(2) $V_1 \subseteq V_2$
(3) $\bar{V}_2 \subseteq \bar{V}_1$
(4) $A_1 \underset{anti}{\preceq} A_2$

Proof. It is clear that 1, 2 and 3 are just reformulations of one another.

First we show "1 $\Longrightarrow$ 4 ". Assume we have $\neg \Big(A_1 \underset{anti}{\preceq} A_2\Big)$ so there is $a_2 \in A_2$ such that for all $a_1 \in A_2$ $\neg\big(a_1 \leq a_2\big)$. Since these are the obstructions for $E_1$ this means that $\Big(E_1(a_2) = \textbf{true}\Big) \Longrightarrow \Big(E_2(a_2) = \textbf{true}\Big)$, a contradiction to $a_2$ being an obstruction for $E_2$.

Now we show "4 $\Longrightarrow$ 1". Assume $A_1 \underset{anti}{\preceq} A_2$. Let for $v \in V$ $E_2(v)$ be false. Then there is $a_2 \in A_2$ with $a_2 \leq v$. By assumption there is $a_1 \in A_1$ with $a_1 \leq a_2$ thus $a_1 \leq v$ holds too. So $E_1(v) = \textbf{false}$ . So $\neg E_1 \Longrightarrow \neg E_2$ and thus $E_1 \Longrightarrow E_2$. $\square$

Another property we will need is given in the following lemma.

LEMMA 9.3. *Let $Q = \big(V, \leq\big)$ be a qo and $A_1$ and $A_2$ be antichains in $Q$. Then there is a unique antichain $B$ which fulfills $B \underset{anti}{\preceq} A_1$ and $B \underset{anti}{\preceq} A_2$ and is maximal with that property.*

PROOF. Clearly $B = Min\{A_1 \cup A_2\}$ the set of minima of the union of $A_1$ and $A_2$ fulfills all properties desired. $\square$

An order with a unique lower bound for any arbitrary pair of elements is often called a **semi-lattice**.

We denote the unique maximal lower bound with $\inf\{A_1, A_2\}$. Clearly this construction leads also to a unique lower bound of any finite set of antichains. For finite posets this, and an analogous observation for an upper limit, shows that the set of maximal antichains forms a lattice, see e.g. [Dil58, Reu91].

The same construction gives also a unique maximal lower bound for arbitrary collections of antichains if $Q$ is well founded.

THEOREM 9.1. *Let $\mathfrak{A}$ be an arbitrary set of antichains of the well founded qo $Q = \big(V, \leq\big)$. Then there is a unique antichain $B$ which fulfills*

$$B \underset{anti}{\preceq} A \text{ for all } A \in \mathfrak{A} \tag{9.2}$$

*and is maximal with that property.*

PROOF. Denote with $\inf_{a \in \mathfrak{A}} A$, or inf for short, the set of minima of the union of $A \in \mathfrak{A}$. Because $Q$ is well founded this is well defined and fulfills property 9.2.

Let $B$ have property 9.2. We show that $B \underset{anti}{\preceq} \inf$.

Let $a \in \inf$. Then there is $A \in \mathfrak{A}$ with $a \in A$ and so there is $b \in B$ such that $b \leq a$. $\square$

LEMMA 9.4. *Let $Q = \big(V, \leq\big)$ be a well founded qo but not a wqo. Then there is a minimal infinite antichain with respect to $\underset{anti}{\preceq}$.*

PROOF. Let $A_1 \underset{anti}{\succeq} A_2 \underset{anti}{\succeq} \cdots$ be a descending chain of infinite antichains. Let

$$A_\infty = \{v \in V \mid \text{there is } N \in \mathbb{N} \text{ s.t. } v \in A_i \text{ for all } i \geq N\} = \inf_{i \in \mathbb{N}} A_i \tag{9.3}$$

We show that $A_\infty$ is infinite. Assume the contrary. Then there is $N_0$ such that $A_\infty \subset A_i$ for all $i \geq N_0$. $A_{N_0}$ is infinite so there is an $a_0 \in A_{N_0} \setminus A_\infty$. By inductive choices there are $a_i \in A_{N_0+i}$ such that $a_0 \geq a_1 \geq \cdots$. All these $a_i$ are not in $A_\infty$ since otherwise $a_0$ would be related to an element of $A_\infty$ which is a subset of $A_{N_0}$.

This descending chain in $Q$ is stationary. So there are $a_\infty$ and $N_1$ such that $a_i = a_\infty$ for all $i \geq N_1$. But then $a_\infty \in A_\infty$ a contradiction.

So any descending chain of infinite antichains has a lower bound that is infinite, too. Zorn's Lemma gives the claim. □

Lemma 9.4 does not mean that the set of antichains of $Q$ is well founded with respect to $\underset{anti}{\preceq}$ if $Q$ is not a wqo. For that let $\{a_1, a_2, \ldots\}$ be a countable antichain of $Q$. Then $A_i = \{a_1, \ldots, a_i\}$ for $i \in \mathbb{N}$ defines an infinite descending chain that is not stationary.

THEOREM 9.2. *Let $Q = \left(V, \leq\right)$ be well founded but not a wqo, $A \subseteq V$ be a minimal infinite antichain and let $V^{<A} = \{v \in V \mid$ there is $a \in A$ with $v < a\}$. Then $Q^{<A} = \left(V^{<A}, \leq\right)$ is a wqo.*

PROOF. Clearly $Q^{<A}$ is also well founded.

Now let $B \subseteq V^{<A}$ be an arbitrary antichain and $\inf\{A, B\}$ as given above.

$\inf\{A, B\}$ is an antichain. By definition we have also $\inf\{A, B\} \underset{anti}{\preceq} A$ so $\inf\{A, B\}$ must be finite.

But since $B \subseteq V^{<A}$ we have that $B \subseteq \inf\{A, B\}$. So $B$ is finite, too. □

COROLLARY 9.1. *Let $Q$, $A$ and $v \in V^{<A}$ be as above. Then the set*

$$A^{\| v} = \{a \in A \mid a \parallel v\},$$

*is finite.*

## 10. Strings

Now we revisit a well-known qo given by a relation on strings. We investigate this ordered structure to exemplify some of the tools we developed until now. We will use Theorem 9.2 to show that this is a wqo and then Theorem 6.1 to show existence of algorithms for hereditary properties.

## 10.1. Higman's Theorem.

Definition 10.1. Let $P = (V, \leq)$ be a qo and $\alpha$ , $\beta \in V^*$, $\alpha = (a_1, \dots, a_r)$, $\beta = \big(b_1, \dots, b_s\big)$. Then we say $\alpha \underset{\star}{\preceq} \beta$ if there is a mapping $\rho$ such that $a_i \leq b_{\rho(i)}$ and that is strictly monotonous. See Figure 10.1.

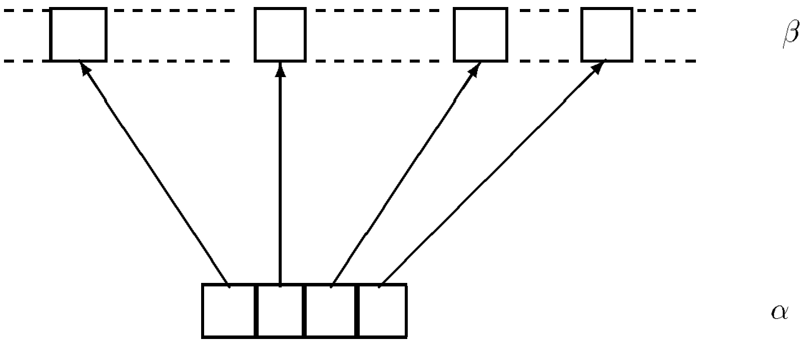

Figure 10.1.

With $P^*$ we denote $(V^*, \underset{\star}{\preceq})$ which clearly is a qo. It is also clear that "$\underset{\star}{\preceq}$" is admissible for the concatenation of strings. If we have $\alpha \underset{\star}{\preceq} \beta$ and $\gamma \underset{\star}{\preceq} \delta$ then $\alpha \cdot \gamma \underset{\star}{\preceq} \beta \cdot \delta$ holds, too. The following theorem is due to Higman [Hig52].

Theorem 10.1 (Higman's Theorem). *$P^* = (V^*, \underset{\star}{\preceq})$ is a wqo iff $P = \big(V, \leq\big)$ is a wqo.*

We give a variation of the proof of Nash-Williams [NW63], see also [Pou85].

Proof. "$\Longrightarrow$" holds since $P$ may be seen as the suborder of $P^*$ consisting of the one-element strings.

"$\Longleftarrow$" It is easy to see that $P^*$ is well founded since in a descending chain $\alpha_1 \underset{\star}{\succeq} \alpha_2 \underset{\star}{\succeq} \cdots$ all $\alpha_i$ have bounded length.

Suppose now $P^*$ is not a wqo. With Lemma 9.4 it would have a minimal infinite antichain $A$, say.

Let $Q = \big(P^*\big)^{<A}$ be the wqo of elements in $P^*$ which are strictly below that antichain, see Theorem 9.2. The set

$$A' = \{\alpha \in P^* \mid \text{there is } v \in V \text{ such that } v\alpha \in A\} \tag{10.1}$$

is a subset of $Q$. It must be infinite so it has a perfect subsequence of pairwise disjoint elements, $(\alpha_i)_{i \in \omega}$ say.

Now consider the corresponding sequence $(v_i)_{i\in\omega}$ in $P$ such that $v_i\alpha_i \in A$. It is good, so there are $i < j$ with $v_i \leq v_j$. Thus we would also have

$$v_i\alpha_i \underset{\star}{\preceq} v_j\alpha_j \tag{10.2}$$

a contradiction. □

**10.2. Fast Tests for Strings.** For the following discussion on algorithms for strings we will assume that all strings are given by an array. Since we want to handle strings over arbitrary wqo, we naturally can not say anything about an encoding of the elements in these. Therefore we assume that the elements in such a wqo are given as pointers and that we have an oracle to test the relation. In total, if we want to consider $Q^*$ for a wqo $Q = (V, \leq)$

- $\text{size}(\alpha) = c \cdot \sum_{i=1}^{l} \text{size}(a_i)$, where $\alpha = a_1 \cdots a_l$ and $c$ is a universal constant not depending on $\alpha$ and
- an algorithm $\texttt{lesseq}_Q(a, b)$ to test whether or not $a \leq b$ is given.

We will show

Theorem 10.2. *Let $Q$ be a qo and $\mathcal{P} \in$ $\boldsymbol{P}, \boldsymbol{P}^\tau, \boldsymbol{NC}, \boldsymbol{NC}^\tau, \boldsymbol{NP}, \boldsymbol{DEC}$, $\tau \geq 1$. If every hereditary property in $Q$ is in $\mathcal{P}$ then every hereditary property in $Q^*$ is in $\mathcal{P}$, too.*

First we give an easy algorithm for sequential machines.

Algorithm 10.1. $\texttt{scan}^{\texttt{seq}}{}_Q(\alpha, \beta)$

```
 Input:  $\alpha = a_1 \cdots a_k$ and $\beta = b_1 \cdots b_l$
Output:  $\rho_1 < \cdots < \rho_k$ such that $a_i \leq b_{\rho_i}$ if it exists or false if not.
   (1)   if $k = 0$ then return true
   (2)   if $k > l$ then return false
   (3)   for $i := 1$ to $l$ do $c_i := b_i$
   (4)   for $i := 1$ to $k$ do $c_{l+i} := a_i$
   (5)   $j := 1$
   (6)   for $i := 1$ to $k$ do begin
   (7)       while $\neg\texttt{lesseq}_Q(a_i, c_j)$ do $j := j + 1$
   (8)       $\rho_i := j$
   (9)       $j := j + 1$
  (10)       end
  (11)   if $\rho_k > l$ then return false
  (12)   else return $\rho$
```

The following remark is easy, so we omit the proof.

REMARK 10.1. *$\mathtt{scan}^{seq}$ is correct and has a running time of $O(k+l)$ plus $O(l)$ queries to the oracle* `lesseq`.

LEMMA 10.1. *Let $Q = (V, \leq)$ be a qo such that $Prop_a$ for fixed $a \in V$ is in $\boldsymbol{P}^{\tau}$ for $\tau \geq 1$. Then for every fixed $\alpha \in Q^*$ $Prop_\alpha$ is in $\boldsymbol{P}^{\tau}$, too.*

PROOF. Let $\alpha = a_1 \cdots a_k$ be fixed. The running time of $\mathtt{scan}^{\mathtt{seq}}$ without the oracle queries is $O(k+l) = O(l)$ since $k$ is fixed.

Let $c_{max}$ be the maximum of the constants of proportionality for the tests for properties $\mathrm{Prop}_{a_1}, \ldots \mathrm{Prop}_{a_k}$. The running time for the queries is the bounded by

$$c_{max} \cdot \sum_{i=1}^{l} \mathrm{size}(b_i)^{\tau} \leq c_{max} \cdot \left( \sum_{i=1}^{l} \mathrm{size}(b_i) \right)^{\tau} \leq c_{max} \cdot \mathrm{size}(\beta)^{\tau} \tag{10.3}$$

□

Now we give a parallel version or our algorithm. It is based on the observation that the embedding we find with $\mathtt{scan}^{\mathtt{seq}}$ is a very special one. The numbers $\rho_i$ are as small as possible.

ALGORITHM 10.2. $\mathtt{scan}^{\mathtt{par}}{}_Q(\alpha, \beta)$

```
 Input:  $\alpha = a_1 \cdots a_k$ and $\beta = b_1 \cdots b_l$
Output:  $\rho_1 < \cdots < \rho_k$ such that $a_i \leq b_{\rho(i)}$ if it exists or false if not.
   (1)   if $k = 0$ then return true
   (2)   if $k > l$ then return false
   (3)   $i_0 := 0$
   (4)   for $j := l+1$ to $l+k$ do $m_j := j$
   (5)   for $j := 1$ to $k$ do begin
   (6)       for $i := 1$ to $l$ do parallel begin
   (7)           if lesseq$(a_j, b_i)$ then $m_i := i$
   (8)           else $m_i := l+i$
   (9)           end
  (10)       $\rho_j := \min_{i>i_0} m_i$
  (11)       $i_0 := \rho_j + 1$
  (12)       end
  (13)   if $\rho_k > l$ then return false
  (14)   else return $\rho$
```

It is clear that $\mathtt{scan}^{\mathtt{par}}$ gives the same output as $\mathtt{scan}^{\mathtt{seq}}$.

LEMMA 10.2. *Let $Q = (V, \leq)$ be a qo such that $Prop_a$ for fixed $a \in V$ is in $\boldsymbol{NC}^\tau$ for $\tau \geq 1$. Then for every fixed $\alpha \in Q^*$ $Prop_\alpha$ is in $\boldsymbol{NC}^\tau$, too.*

PROOF. It is clear that the running time for the loop (6) is dominated by the largest time for the `lesseq`. The number of processors can be estimated in the same way as the running time in the sequential case.

The calculation of the minimum can be done efficiently with $O(l/\log l)$ processors in time $O(\log l)$ with **re-scheduling**. Re-scheduling is based on the idea that, if we have to perform a task $\log l$ times such that we need $l/2^m$ processors in step $m$, we can distribute the work to be done on $l/\log l$ processors such that the running time increases only by a constant factor. So it is efficient, too. □

## 11. Trees

### 11.1. Structured Trees.

DEFINITION 11.1. A **rooted tree** is a triple $\left(V, r, S\right)$ where $V$ is a finite set, $r \in V$, the **root**, and $S$ is a string of rooted trees such that either $\left(V, r, S\right) = \left(\{r\}, r, \emptyset\right)$ or if $S = T_1 \dots T_k$ with $T_i = \left(V_i, r_i, S_i\right)$ then

(1) $V \setminus \{r\} = \bigcup_i V_i$ and
(2) $V_i \cap V_j = \emptyset$ for $i \neq j$.

For $T = \left(V, r, S\right)$ and $v \in V$ define $T_v$ as the unique subtree of $T$ rooted at $v$.

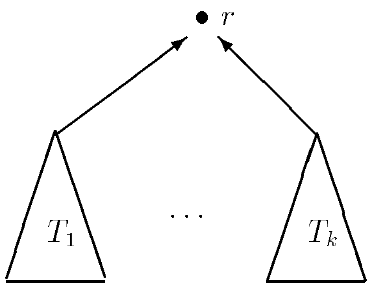

FIGURE 11.1. A rooted tree

An element $v \in V$ is called a **leaf** if $T_v = \left(\{v\}, v, \emptyset\right)$ and it is called a **inner node** if it is not a leaf.

Rooted trees can be easily seen as special partial orders. For every such tree $T\left(V, r, S\right)$ define $r > v$ for all $v \in V \setminus \{r\}$. $>$ is transitive by the recursive definition of $T$.

For two elements $v, w \in V$ define the **least common ancestor** $\mathrm{LCA}_T(v,w)$ as the smallest $x \in V$ such that $x \geq v$ and $x \geq w$. Such an element always exists. The **postorder** on $T$ is the unique linear extension $\underset{post}{<}$ of $<$ which fulfills

(1) if $r_0 = \mathrm{LCA}_T(v,w) \notin \{v,w\}$ and the subtree of $v$ in $T_{r_0}$ is left of the subtree of $w$ then $v \underset{post}{<} w$
(2) if $v$ is a node in $T_{r_0}$ then $v \underset{post}{<} r_0$

Let $T_1 = \left(V_1, r_1, S_1\right)$ and $T_2 = \left(V_2, r_2, S_2\right)$ be rooted trees. $T_1$ is **homeomorphically** embeddable into $T_2$, $T_1 \underset{\circ}{\preceq} T_2$, if there is an injection $\rho\colon\ V_1 \to V_2$ which respects $\underset{post}{<}$ and LCA. i.e., for all $v, w \in V_1$ the following conditions are satisfied.

$$\rho\left(\mathrm{LCA}_{T_1}(v,w)\right) = \mathrm{LCA}_{T_2}(\rho(v), \rho(w)) \tag{11.1}$$

$$v \underset{post}{<} w \iff \rho(v) \underset{post}{<} \rho(w) \tag{11.2}$$

Now let $Q = \left(\mathfrak{W}, \leq_Q\right)$ be a qo. A **structured** or **weighted** tree with weights in $Q$ is a quadruple $T^{\mathfrak{w}} = \left(V, r, S, \mathfrak{w}\right)$ where $T = \left(V, r, S\right)$ is a rooted tree and $\mathfrak{w}\colon\ V \to \mathfrak{W}$ is an arbitrary function.

Denote the set of structured trees over $Q$ with $\mathfrak{T}^Q$.

$T_1^{\mathfrak{w}_1}$ is **homeomorphically** embeddable into $T_2^{\mathfrak{w}_2}$, $T_1^{\mathfrak{w}_1} \underset{\circ}{\preceq} T_2^{\mathfrak{w}_2}$, if there is a homeomorphic embedding $\rho$ of $T_1$ into $T_2$ which respects the weight functions. i.e., for all $v \in V_1$

$$\mathfrak{w}_1(v) \underset{\circ}{\preceq} \mathfrak{w}_1(\rho(v)). \tag{11.3}$$

Denote $\left(\mathfrak{T}^Q, \underset{\circ}{\preceq}\right)$ with $Q^\circ$. It is clear that this is a qo.

Observe that the postorder for each structured tree defines a string over $Q$ in a natural way. Two such strings are related by $\underset{\star}{\preceq}$ if the corresponding trees are related by $\underset{\circ}{\preceq}$. So the following famous theorem of Kruskal, [Kru60], can be seen as an extension of Higman's Theorem 10.1.

THEOREM 11.1 (KRUSKAL'S TREE THEOREM). *$Q^\circ$ is a wqo iff $Q$ is a wqo.*

PROOF. For the proof we follow basically the same ideas as described for Higman's Theorem.

First it is easy to see that $Q^\circ$ is well founded if $Q$ is so. This is because the cardinalities of the groundsets of a descending chain of structured trees are bounded.

Assume now that $Q^\circ$ is not a wqo. Then there is a minimal infinite antichain $A$, say. Let $T^{\mathfrak{w}} = \left(V, r, S, \mathfrak{w}\right) \in A$ be with $S = \left(T_1^{\mathfrak{w}}, \dots, T_k^{\mathfrak{w}}\right)$. For all $i$ we have then that $T_i^{\mathfrak{w}} \underset{\circ\, Q}{\prec} T^{\mathfrak{w}}$.

With Theorem 9.2 we have that $\mathfrak{T}$ the set of all these subtrees is a wqo. So Higman's Theorem 10.1 gives that the strings over this wqo $\mathfrak{T}^*$ form a wqo, too.

If we denote with $\mathfrak{S}$ the set of strings $S$ that occur in the definition of some $T \in A$ we have that $\mathfrak{S} \subseteq \mathfrak{T}^*$. So there is a sequence $S_1 \underset{\star}{\prec} S_2 \underset{\star}{\prec} \cdots$ such that the corresponding trees $T_i = \left(V_i, r_i, S_i, \mathfrak{w}_i\right) \in A$ are pairwise distinct.

Now consider the corresponding subsequence of the weights of the roots

$$\left(\mathfrak{w}_i\left(r_i\right)\right)_{i\in\omega}. \tag{11.4}$$

It is good, so there are $i < j$ such that

$$\mathfrak{w}_i\left(r_i\right) \leq_Q \mathfrak{w}_j\left(r_j\right) \tag{11.5}$$

But then we may extend the embedding of $S_i$ into $S_j$ such that we achieve

$$T_i^{\mathfrak{w}_i} \underset{\circ}{\preceq} T_j^{\mathfrak{w}_j}, \tag{11.6}$$

a contradiction. □

**11.2. Algorithms for Structured Trees.** We will give test algorithms for structured trees. For that purpose we proceed analogously as we did for strings and give a sequential one first, and a parallel one afterwards. As we did there, we will also assume that the weights are given by pointers and we have an oracle `lesseq`.

The rooted trees will be given in such a way we have direct access from a vertex $v$ to the string $S^v$ of the subtree $T_v$.

THEOREM 11.2. *Let $Q$ be a qo and $\mathcal{P} \in$ $\boldsymbol{P}, \boldsymbol{P}^\tau, \boldsymbol{NC}, \boldsymbol{NC}^\tau, \boldsymbol{NP}, \boldsymbol{DEC}$, $\tau \geq 1$. If every hereditary property in $Q$ is in $\mathcal{P}$ then every hereditary property in $Q^\circ$ is in $\mathcal{P}$, too.*

We split the proof of this theorem over several lemmas. By the definition it would be easy to formulate a recursive algorithm to calculate $T_1^{\mathfrak{w}_1} \underset{\circ}{\preceq} T_2^{\mathfrak{w}_2}$. We chose an iterative approach that allows a parallelization afterwards.

ALGORITHM 11.1. $\texttt{embed}^{\texttt{seq}}{}_Q\left(T_1^{\mathfrak{w}_1}, T_2^{\mathfrak{w}_2}\right)$

**Input:** $T_1^{\mathfrak{w}_1}$ and $T_2^{\mathfrak{w}_2}$. The vertices of both trees are given in postorder. According to that order they are identified with the numbers $1, \ldots, k$ and $1, \ldots, l$ respectively.

**Output:** $T_1^{\mathfrak{w}_1} \underset{\circ}{\preceq} T_2^{\mathfrak{w}_2}$

```
 (1)  **if** $k = 0$ **then return true**
 (2)  **if** $k > l$ **then return false**
 (3)  **for** $j := 1$ **to** $l$ **do begin**
 (4)      $weight_j := \{0 < i \leq k \mid \mathfrak{w}_1(i) \leq \mathfrak{w}_2(j)\}$
 (5)      $val_j := \emptyset$
 (6)      **end**
 (7)  **for** $i := 1$ **to** $k$ **do begin**
 (8)      **for** $j := 1$ **to** $l$ **do begin**
 (9)          **if** $\left(S_1^i \underset{\star}{\preceq} S_2^j\right) \wedge (i \in weight_j)$ **then** $val_j := val_j \cup \{i\}$
(10)          **end**
(11)      **for** $j := 1$ **to** $l$ **do begin**
(12)          $val_j := \bigcup_{s \in S_2^j} val_s$
(13)          **end**
(14)      **end**
(15)  **return** $(k \in val_l)$
```

It is easy to see that this algorithm is correct, since we access the vertices in the right order.

LEMMA 11.1. *If $T_1$ is fixed* `embed`$^{seq}$ *can be implemented such that it runs in time $O(l)$ plus the time needed for $O(l)$ queries to the oracle.*

PROOF. All sets $val_.$ have a cardinality bounded by $k$. We may assume that we are given a representation of $\mathcal{B}_k$, the Boolean Lattice on $k$ points. This representation can be chosen such that each of the set operations mentioned needs $O(1)$ time.

The critical calculations to consider are the test for $\underset{\star}{\preceq}$ and the union in line (12). When we access $j$ the necessary information for all $s \in S_2^j$ is already present at vertex $s$. To test $\underset{\star}{\preceq}$ build two strings in $\left(\mathcal{B}_k\right)^*$; $\alpha = \{s_1^1\} \cdots \{s_1^{m_i}\}$ and $\beta = val_{s_2^1} \cdots val_{s_2^{m_j}}$, where $S_i = s_1^1, \ldots, s_1^{m_i}$ and $S_j = s_2^1, \ldots, s_2^{m_j}$. Now clearly $\alpha \underset{\star}{\preceq} \beta$ iff $S_1^i \underset{\star}{\preceq} S_2^j$. So $\texttt{scan}^{\texttt{seq}}{}_{\mathcal{B}_k}(\alpha, \beta)$ does the job. Since every vertex appears in at most one string $S_2^j$ the time calls to $\texttt{scan}^{\texttt{seq}}{}_{\mathcal{B}_k}$ need in total is $O(l)$.

An analogous argumentation holds for the union in line (12). □

The following lemma is an immediate consequence, so we omit the proof.

LEMMA 11.2. *Let $Q = (V, \leq)$ be a qo such that $Prop_v$ for fixed $v \in V$ is in $\boldsymbol{P}^\tau$ for $\tau \geq 1$. Then $Prop_{T_1^{\mathfrak{w}_1}}$ for every fixed $T_1^{\mathfrak{w}_1} \in Q^\circ$ is in $\boldsymbol{P}^\tau$, too.*

Now we are going to parallelize the algorithm $\texttt{embed}^{\texttt{seq}}$. We use an approach similar to the one chosen by Miller and Reif [MR85], see also Abrahamson et al. [ADKP89]. It can be seen as a generalization of the so called **list ranking**. List ranking is based on the observation that if we have a linked list, such that each element is connected to its successor and to the successor of the successor, information may be propagated along this list in logarithmically many steps.

For this algorithm we will assume that we have values $par_2^j$, giving the parent vertex of $j$ in $T_2$ if it exists, and $gpar_2^j = par_2^{par_2^j}$, the "grand parent" of $j$. Set $par_2^l = l$ and $gpar_2^j = l$ if $par_2^j = l$.

ALGORITHM 11.2. $\texttt{embed}^{\texttt{par}}{}_Q\,(T_1^{\mathfrak{w}_1}, T_2^{\mathfrak{w}_2})$

**Input:** $T_1^{\mathfrak{w}_1}$ and $T_2^{\mathfrak{w}_2}$. The vertices of both trees are given in postorder. According to that order they are identified with the numbers $1, \dots, k$ and $1, \dots, l$ respectively.

**Output:** $T_1^{\mathfrak{w}_1} \underset{\circ}{\preceq} T_2^{\mathfrak{w}_2}$

```
(1)  if $k = 0$ then return true
(2)  if $k > l$ then return false
(3)  for $j := 1$ to $l$ do parallel begin
(4)      $weight_j := \{0 < i \leq k \mid \mathfrak{w}_1(i) \leq \mathfrak{w}_2(j)\}$
(5)      $val_j := \emptyset$
(6)      end
(7)  for $i := 1$ to $k$ do begin
(8)      for $j := 1$ to $l$ do parallel begin
(9)          $ok_j := \left(S_1^i \underset{\star}{\preceq} S_2^j\right) \wedge (i \in weight_j)$
(10)         end
(11)     for $m := 0$ to $\lfloor \log l \rfloor$ do begin
(12)         for $j := 1$ to $l$ do parallel begin
(13)             if $ok_j$ then begin
(14)                 $ok_{par_j}$ := true
(15)                 $ok_{gpar_j}$ := true
(16)                 end
(17)             end
(18)         end
(19)     if $\neg ok_l$ then return false
```

```
(20)        for $j := 1$ to $l$ do parallel begin
(21)            if $ok_j$ then $val_j := val_j \cup \{i\}$
(22)            end
(23)        end
(24)    return true
```

LEMMA 11.3. *$\texttt{embed}^{par}$ is correct. Besides the calls to $\texttt{lesseq}_Q$ it can be implemented in such a way that it has a running time of $O(k \log l)$ and needs $O(l)$ processors.*

PROOF. Correctness: We have to show that $val$ always contains the right information, i.e., $i \in val_j$ if the subtree $T_1^j$ of $T_1$ rooted at $i$ can be embedded into the subtree $T_2^j$ of $T_2$ rooted at $j$. But this is true since if we had $j_0$ such that $T_1^i$ embeds "directly" then all its parent nodes have the information after $O(\log l)$ propagation steps (12).

The estimation of the running time and amount of processors needed is straightforward and thus omitted. □

Observe that the running time strongly relies on the fact that we are using a CRCW PRAM. The only write conflict that can occur is that two processors want to write the same value **true** into the same place for their common parent resp. grandparent.

The algorithm given here is not totally optimal since the product of time and amount of processors needed is $O(l \cdot \log l)$ and not $O(l)$. This could be improved by re-scheduling of loop (12). But the technique would be much more complicated as for the calculation of a minimum, say.

To conclude the proof of Theorem 11.2 we give, without proof,

LEMMA 11.4. *Let $Q = (V, \leq)$ be a qo such that $Prop_v$ for fixed $v \in V$ is in $\boldsymbol{NC}^\tau$ for $\tau \geq 1$. Then $Prop_{T_1^{\mathfrak{w}_1}}$ for every fixed $T_1^{\mathfrak{w}_1} \in Q^\circ$ is in $\boldsymbol{NC}^\tau$, too.*

## 12. Special Classes of Posets

Because of their treelike composition rules some classes of posets that are recursively composed from small ones give a good example for the theory of structured trees. This approach was first used by Damaschke in [Dam90] for a certain class of graphs, the cographs.

**12.1. Series Parallel Orders and Cographs.** We give an application of Theorem 11.2 to a special class of orders called series parallel orders and to the associated class of comparability graphs called cographs. See e.g. [Möh89] for references for these objects.

DEFINITION 12.1. A finite order is a **series parallel order** if

(1) it is the order on 1 point,
(2) it is obtained from series parallel orders by a series composition,
(3) it is obtained from series parallel orders by a parallel composition.

With $\mathfrak{O}^{sp}_{ind}$ we denote the set of finite series parallel orders equipped with $\underset{ind}{\preceq}$.

This recursive definition lets us easily define associated structured trees to series parallel orders:

DEFINITION 12.2. Let $P$ be series parallel order. The **cotree** $T^{\mathfrak{w}} = \left(V, r, S, \mathfrak{w}\right)$ of $P$ is either the tree on one node that is labeled $l$ iff $P$ is the order on 1 point, or if $P = P_0\left[\{P_i\}_{1,\ldots,k}\right]$ then

(1) $S = \left(T_1, \ldots, T_k\right)$ and $T_i^{\mathfrak{w}_i}$ are the cotrees of the $P_i$,
(2) $\mathfrak{w}(v) = \mathfrak{w}_i(v)$ for $v \in P_i$
(3) $\mathfrak{w}(r) = s$ if $P_0 = C_k$ and
(4) $\mathfrak{w}(r) = p$ if $P_0 = A_k$.

So a cotree is a rooted tree weighted over the trivial order $Q_{ind} = \left(\{l, s, p\}, \emptyset\right)$ where $l$, $s$ and $p$ stand for "leaf", "series" and "parallel" respectively. As we defined it here the cotree of a series parallel order is not unique.

The following is an easy observation, see e.g. [Möh89]:

LEMMA 12.1. *Let $P = (V, <)$ be a series parallel order and $T^{\mathfrak{w}} = \left(V, r, S, \mathfrak{w}\right)$ a corresponding cotree. Then $v < w$ iff $\mathfrak{w}\left(LCA_T(v, w)\right) = s$ and the subtree of $v$ is left of the subtree of $w$.*

An easy corollary out of that is

COROLLARY 12.1. *Let $P_1 = (V_1, <_1)$ and $P_2 = (V_2, <_2)$ be series parallel orders and $T_i^{\mathfrak{w}_i} = \left(V_i, r_i, S_i, \mathfrak{w}_i\right)$ the corresponding cotrees such that $T_1^{\mathfrak{w}_1} \underset{\circ}{\preceq} T_2^{\mathfrak{w}_2}$. Then $P_1 \underset{ind}{\preceq} P_2$.*

The following is then a slight extension of a result of Damaschke, [Dam90]

THEOREM 12.1. *$\mathfrak{O}^{sp}_{ind}$ is a wqo and every hereditary property $E$ in $\mathfrak{O}^{sp}_{ind}$ has a test that runs in $O(n)$ sequential time or $O(\log n)$ on $O(n)$ processors provided the cotree of the input is given.*

Indeed Damaschke has proven an analogous result for cographs that are the comparability graphs of series parallel orders.

PROOF. The wqo-property is an immediate consequence of Kruskal's Tree Theorem together with Corollary 12.1.

For the running time observe that the set of vertices of an induced suborder $P_0$ of a series parallel order $P$ induces a "subtree" $T_0$ of the cotree $T$ for $P$ that is a cotree for $P_0$.

But every obstruction $P_0$ for $E$ admits only a finite number of cotrees. So to test our property we have to test all trees for all obstructions, in total a finite number of trees, the number only depending on $E$, not on the input.

So Theorem 11.2 proves the claim. $\square$

A cotree can be found in $O(n+m)$ sequential time, see [CPS85], resp. in $O(\log n)$ on $O\left(\frac{n^2+mn}{\log n}\right)$ processors, see [LO92], where $n$ is the number of points and $m$ is the number of $m$ related pairs. So we need these running times if we have to construct the cotree.

**12.2. Bounded Decomposition Diameter.** We extend what we said about $\mathfrak{O}^{sp}_{ind}$ to other classes of orders. Therefore let in the following $S = \left(Q_1, \ldots, Q_k\right)$ be a finite set of finite orders that is assumed to be fixed in the sequel, and let $Q_i = (X_i, <_i)$ for every $i$.

DEFINITION 12.3. $\mathfrak{O}^{S}_{ind}$ is the set of orders given by the following recursive definition equipped with $\underset{ind}{\preceq}$:

(1) The order on one point is in $\mathfrak{O}^{S}_{ind}$.
(2) For all $i$ and $x \in X_i$ if $P_x \in \mathfrak{O}^{S}_{ind}$ then $Q_i\left[\{P_x\}_{x\in X_i}\right] \in \mathfrak{O}^{S}_{ind}$.

This classes of orders have been introduced by Habib and Möhring, [HM87], where a slightly different definition is given.

The value $\max_i\left\{|X_i|\right\}$ is called the **decomposition diameter** of the class $\mathfrak{O}^{S}_{ind}$.

If $S = \{A_2, C_2\}$ we again obtain the class of series parallel orders.

We define a **generalized cotree** in an analogous way we defined the cotree for series parallel orders. For that purpose we assume that the elements $x \in X_i$ of the orders $Q_i \in S$ are given in a fixed ordering $x_i^1, x_i^2, \ldots$. Then we obtain a structured tree for each $P \in \mathfrak{O}^{S}_{ind}$ with weights chosen form $S \cup \{l\}$. Again we easily obtain the following lemma:

LEMMA 12.2. *Let $P = (V,<) \in \mathfrak{O}^{S}_{ind}$, $T^{\mathfrak{w}} = \left(V, r, s, \mathfrak{w}\right)$ a corresponding cotree, $v, w \in V$ with $\mathfrak{w}\left(LCA_T(v,w)\right) = Q_0 = (X_0, <_0)$ and $x_v, x_w \in X_0$ such that $v$ resp. $w$ is the subtree of $x_v$ resp. $x_w$. Then $v < w$ iff $x_v <_0 x_w$.*

Again an easy corollary out of that is

COROLLARY 12.2. *Let* $P_1 = (V_1, <_1)$ *and* $P_2 = (V_2, <_2)$ *be in* $\mathfrak{O}^S_{ind}$ *and* $T_i^{\mathfrak{w}_i} = \left(V_i, r_i, S_i, \mathfrak{w}_i\right)$ *the corresponding cotrees such that* $T_1^{\mathfrak{w}_1} \underset{\circ}{\preceq} T_2^{\mathfrak{w}_2}$. *Then* $P_1 \underset{ind}{\preceq} P_2$.

This is so since only such pairs of nodes of the cotrees are mapped for which the corresponding elements of $S$ are identical. In particular all such pairs of nodes have the same degree.

Now we obtain an analogous theorem as before, but for much wider classes of orders.

THEOREM 12.2. $\mathfrak{O}^S_{ind}$ *is a wqo and every hereditary property* $E$ *in* $\mathfrak{O}^S_{ind}$ *has a test that runs in* $O(n)$ *sequential time or* $O(\log n)$ *on* $O(n)$ *processors provided the cotree of the input is given.*

PROOF. With what is said above and Kruskal's Theorem $\mathfrak{O}^S_{ind}$ clearly is a wqo. We have to show that it admits linear time algorithms for the properties $\mathrm{Prop}_{P_0}$, $P_0$ any fixed order in $\mathfrak{O}^S_{ind}$.

We cannot argue the same way as we did for Theorem 12.1 since we have no equivalence in Corollary 12.2. There are to ways to circumvent this problem. Either we may modify our relation on the cotrees to get equivalence or we may give an alternative algorithm that computes $\mathrm{Prop}_{P_0}$. We chose the later one.

For that let $P_0 \in \mathfrak{O}^S_{ind}$ be arbitrary but fixed and $P_1, \ldots, P_k$ be an arbitrary enumeration of the induced suborders of $P_0$. Denote the set $\{P_i \mid i = 0, \ldots, k\} \cup \{(\emptyset, \emptyset)\}$ with $R$ and the set of subsets of $R$ with $2^R$.

For every $Q = (V, <) \in S$ with $V = \{v_1, \ldots, v_\ell\}$ and every $\rho = (r_1, \ldots, r_\ell) \in (2^R)^\ell$ we calculate the following set in advance

(12.1)
$$Indu(Q, \rho) = \left\{P \in R \mid \exists\ s_i \in r_i \text{ s.t. } P \text{ is induced suborder of } Q\left[\{s_i\}_{i=1,\ldots,\ell}\right]\right\}$$

that is the set of all suborders of $P_0$ that can be constructed from $Q$ by substituting the vertices of $Q$ with orders chosen from particular sets of orders $r_i$.

All these sets can be calculated in advance in constant time since $S$ and $P_0$, and thus $R$, are fixed.

But with this information at hand it is easy to modify `embed` such that for every cotree $T^{\mathfrak{w}}$ of an order $P \in \mathfrak{O}^S_{ind}$ that is given as input the whole set $\{P' \mid P' \in R,\ P' \text{ is induced suborder of } P\}$ is computed from the corresponding sets that are calculated for the children of the root of $T^{\mathfrak{w}}$. $\square$

CHAPTER IV

# Special Order Relations for Combinatorial Structures

## 13. Formal Languages

We introduce now a qo relation on formal languages. We will then apply the machinery of Chapter II, i.e., we will show a wqo-theorem and give algorithms for hereditary properties.

The main motivation fo us to study these objects is that in Section 14 we will reduce the problem of posets being related by the so called chain minor relation to a similar problem on languages.

**13.1. The String Minor Relation for Formal Languages.** For $\alpha = a_1 \cdots a_k$ and $\beta = b_1 \cdots b_l$ we say $\beta \underset{\leftrightarrow}{\prec} \alpha$ if there is a mapping $\pi\colon \ \{1, \ldots, l\} \ \to \ \{1, \ldots, k\}$ which is strictly monotonous and such that $a_{\pi(1)} \cdots a_{\pi(l)} = b_1 \cdots b_l$ or equivalently if $\alpha$ and $\beta$ are related by $\underset{\star}{\preceq}$ and if their alphabet forms an antichain.

For two formal languages $L', L \subseteq A^*$ we say that $L' \underset{\leftrightarrow}{\prec} L$ if there is some $\alpha \in L$ with $\beta \underset{\leftrightarrow}{\prec} \alpha$ for all $\beta \in L'$.

Observe that for two languages $L'$ and $L$ to be related by $\underset{\leftrightarrow}{\prec}$ it is necessary that $\mathrm{dom}\,(L') \subseteq \mathrm{dom}\,(L)$. This restriction is relaxed in the following.

A labeling $\rho$ from $A$ to $B$ is a mapping $\rho\colon \ A \ \to \ B^{\leq 1}$ or equivalently a partial mapping from $A$ to $B$. For all $a \in A$ with $\rho(a) = \emptyset$ we will say that $\rho$ is **undeclared** for $a$. $\rho$ is the **trivial** labeling if it is undeclared for all $a \in A$.

For a labeling $\rho$ from $A$ to $B$ and $\alpha = a_1 \cdots a_k \in A^*$ $\rho(\alpha)$ is the concatenation $\rho(a_1) \cdots \rho(a_k)$ of strings in $B^*$.

For a language $L \subseteq A^*$ set $\rho(L) = \bigcup_{\alpha \in L} \rho(\alpha)$. We say $L' \underset{lang}{\preceq} L$ if there is a labeling $\rho$ such that $L' \underset{\leftrightarrow}{\prec} \rho(L)$. $\rho$ is then called a **string morphism**.

**13.2. Well Quasi Ordering Finite Languages.** One of our applications for our relation on languages we have in mind are the sets of maximal chains of posets.

There we will see each maximal chain as string of its elements — the elements appearing in the string in the same order they appear in the chain.

To show a suitable wqo-theorem on languages we do not allow symbols to appear several times in a particular string. This fits well to our application on posets. So we will consider languages $L$ that fulfill:

Definition 13.1. A language $L$ such that $\text{length}(\alpha) = |\text{dom}(\alpha)|$ for all $\alpha \in L$ is called **non-repetitive** or **simple**.

We then are able to show

Theorem 13.1. *Every set of finite non-repetitive languages ordered by $\underset{lang}{\preceq}$ is a wqo.*

The proof of this theorem depends strongly on Definition 13.1. We think that it should hold in the general case, too, but a proof probably will need some new ideas.

We will be able to prove two other theorems for two other restricted classes of languages. The hardest to prove will be

Theorem 13.2. *Every set of finite languages $S$ such that there is a constant $l_S$ with $\text{length}(L) \leq l_S$ for all $L \in S$ is a wqo with respect to $\underset{lang}{\preceq}$.*

With the following lemma Theorem 13.1 will be an immediate consequence of Theorem 13.2.

Lemma 13.1. *Every infinite sequence $(L_i)$ of finite non-repetitive languages such that $\text{length}(L_i)$ is unbounded is good.*

Proof. We show in particular that there is some $j$ such that $L_1 \underset{lang}{\preceq} L_j$ . Set $l = \text{size}(L_1)$ . Construct a string (with repetition of elements) $\beta \in \text{dom}(L_1)$ by concatenating all $\alpha \in L_1$ in an arbitrary order. We know that $\text{length}(\beta) = l$. Let $\beta = b_1 \cdots b_l$ .

Since $\text{length}(L_i)$ is unbounded there is $j$ such that $\text{length}(L_j) \geq l$ and thus there is $\beta' = b'_1 \cdots b'_m \in L_j$ with $m = |\text{dom}(\beta')| = \text{length}(\beta') \geq l$ .

Now define a labeling $\rho$ from $\text{dom}(L_j)$ to $\text{dom}(L_1)$ by

$$\rho(b') = \begin{cases} b_i & \text{iff } b' = b'_i \text{ for } 1 \leq i \leq l \\ \emptyset & \text{otherwise .} \end{cases} \tag{13.1}$$

It is clear that $\rho$ has all properties desired. □

The reader may verify that Definition 13.1 is not needed in its full strength for that proof. It would be sufficient to assume that $|\text{dom}(L_i)|$ is increasing.

Proof of Theorem 13.1. Let $S$ be a set of non-repetitive languages and let $\big(L_i\big)_{i\in\omega}$ be an arbitrary sequence of elements of $S$. If $\big(\text{length}\,(L_i)\big)_{i\in\omega}$ is unbounded apply Lemma 13.1. If it is bounded apply Theorem 13.2. $\square$

Much easier than the one for Theorem 13.2 will be the proof of the following theorem that gives the wqo-property for the relation $\underset{\leftrightarrow}{\preceq}$ if we consider only languages over a fixed finite alphabet. Remember that this relation is much more restrictive than $\underset{lang}{\preceq}$. The proof is easier since we may apply the machinery of section 9.

Theorem 13.3. *Let $A$ be finite. Then every set of finite formal languages $S \subseteq A^*$ is a wqo with resp. to $\underset{\leftrightarrow}{\preceq}$.*

Proof. We proceed analogously to the theorems of Higman and Kruskal.

Again it is easy to see that we have a well founded relation. Suppose now that our assumption is false. Then there would be a minimal infinite antichain $\mathfrak{A}$ of languages with respect to $\underset{anti}{\preceq}$.

For every $L \in \mathfrak{A}$ let $\alpha_L \in L$ be an arbitrary string $L' = L \setminus \{\alpha_L\}$ and $\mathfrak{A}' = \{L' \mid L \in \mathfrak{A}\}$. Because of Theorem 9.2 $\mathfrak{A}'$ is a wqo.

Let $\big(L_i\big)_{i\in\omega}$ be an infinite sequence such that $L'_1 \underset{\leftrightarrow}{\preceq} L'_2 \underset{\leftrightarrow}{\preceq} \ldots$ is perfect. Then Higman's Theorem shows that $(\alpha_i)_{i\in\omega}$ is good so there are $i < j$ with $\alpha_i \underset{\star}{\preceq} \alpha_j$. But then $L_i \underset{\leftrightarrow}{\preceq} L_j$, a contradiction. $\square$

Observe that the statement of this theorem would not hold for infinite formal languages since the relation $\underset{\leftrightarrow}{\preceq}$ then is not well founded.

**13.3. Bounded Length.** First we will consider the special case that all strings in all languages have the same *length*. For that purpose we will need some further technical definitions to characterize common behavior of strings. In particular we want to classify strings which have certain positions identical and certain others not.

A type over $A$ will denote a string over $A$ with several positions **undefined** or **blanc** i.e. Type$\,(A) = \big(A \cup \{\sqcup\}\big)^*$. Here '$\sqcup$' is an additional symbol not in $A$. $\sqcup k = \underbrace{\sqcup\cdots\sqcup}_{k}$ for $k \geq 0$ are the **trivial** types.

It is clear that $A^* \subset$ Type$\,(A)$.

We extend the set operations "$\cap$", "$\setminus$" and "$\subseteq$" to types of strings of same *length* in a natural way. For $\alpha = a_1\cdots a_k$, $\beta = b_1\cdots b_k$ we set $\alpha\cap\beta = c_1\cdots c_k$

where

$$c_i = \begin{cases} a_i & \text{iff } a_i = b_i \\ \sqcup & \text{otherwise.} \end{cases} \tag{13.2}$$

We say $\alpha \subseteq \beta$ if $\alpha \cap \beta = \alpha$. If $\alpha \subseteq \beta$ then $\beta \setminus \alpha = c_1 \cdots c_k$ where

$$c_i = \begin{cases} \sqcup & \text{if } a_i \neq \sqcup \\ b_i & \text{otherwise.} \end{cases} \tag{13.3}$$

These operations have all properties we would expect — $\cap$ commutes and $\subseteq$ is transitive and reflexive. So Type$(A)$ forms a qo with resp. to $\subseteq$.

In that qo we have initial and final segments as usual. For $X, Y, T \subseteq$ Type$(A)$ we may define $T_X^Y$, $T$ **between** $X$ and $Y$, by

$$T_X^Y = T_X \cap T^X = \{\tau \in T \mid \exists x \in X, y \in Y \;\; x \subseteq \tau \subseteq y\}. \tag{13.4}$$

$\alpha$ and $\beta$ are $\tau$-independent if $\alpha \cap \beta \subseteq \tau$.

A language $L$ is $\tau$-independent if all pairs $\alpha \neq \beta$ are so. For a language $L$ we denote by ind$(\tau, L)$ the maximum cardinality of a $\tau$-independent subset

$$\operatorname{ind}(\tau, L) = \max\Big\{|L'| \mid L' \subseteq L \;,\; L' \;\tau\text{-independent}\Big\} \tag{13.5}$$

Looking at sequences of languages the following definition handles a situation which is ideal for our purposes — there is an isomorphic "sublanguage" in every language in the sequence s.t. the amount of "other" strings gets arbitrary large.

Definition 13.2. For a sequence $L_1, L_2 \ldots$ of languages a **bottleneck** is a sequence $T_1, T_2 \ldots$ with $T_i \subseteq$ Type$(\text{dom}(L_i))$ and which fulfills the following properties:

(1) For all $i \in \mathbb{N}$ and $\alpha \in L_i$ there is $\tau_\alpha \in T_i$ with $\tau_\alpha \subseteq \alpha$.
(2) There is $T_0$ s.t. $T_i$ is isomorphic to $T_0$ for all $i \in \mathbb{N}$ .
(3) For all $\tau \in T_0$ either $\tau \in L_i$ for all $i \in \mathbb{N}$ or the sequence ind$(\tau, L_i)$ is unbounded and monotonous.

We will always assume that $t_\alpha$ as required in 1 is maximal with respect to $\subseteq$ having that property. For a simple example of a bottleneck see fig. 13.1. Here $T_0$ is indicated by the two boxes in every $L_i$. Every string "passes" this boxes.

Theorem 13.4. *For every sequence $\Big(L_i\Big)$ of finite languages with bounded length there is a choice $\rho$ of a subsequence $\Big(L_{\rho(i)}\Big)$ that has a bottleneck.*

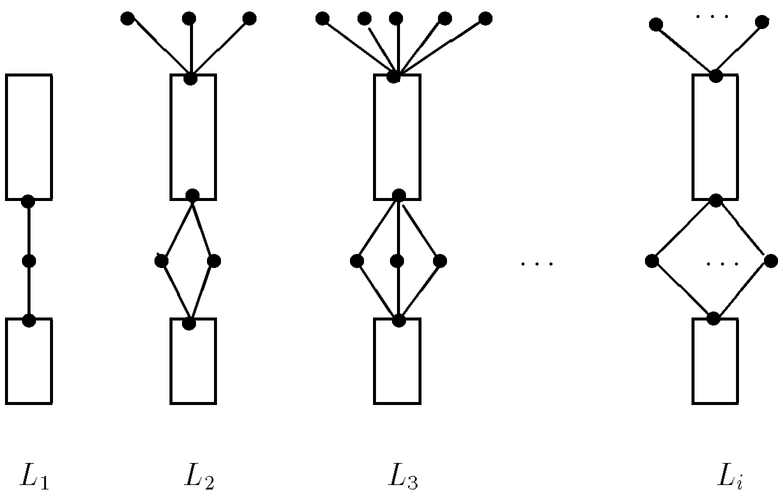

FIGURE 13.1. An Example of a bottleneck

To show this theorem we will present "algorithms" where the word algorithm is extended in a certain sense. We will consider choices of subsequences as one step. We do this in order to have a comfortable mechanism for recursive choices.

First we consider an algorithm `freeze` which fulfills the following specifications.

ALGORITHM 13.1. $\texttt{freeze}\left(\left(L_i\right), \left(T_i\right), \rho, T_0\right)$

**Input:** sequences $\left(L_i\right)$ and $\left(T_i\right)$, $T_i \subseteq \mathrm{Type}(L_i)$ s. t. $|T_i|$ and $|\mathrm{dom}\,(T_i)|$ are globally bounded.

**Output:** Choice $\rho$ of subsequence $\left(L_{\rho(i)}\right)$ and $T_0$ s.t. all $T_{\rho(i)}$ are isomorphic to $T_0$.

It is clear that such a choice is possible, since there are only finitely many isomorphism types for the $T_i$ .

The second algorithm `freeze&thaw` contains the core of our argument. It consists of three phases:

- Steps (1) to (5) ("Bottom") are executed at the lowest recursion level. They handle two easy cases. One is that $|L_i|$ is bounded, the other is that $L_i$ is itself its own bottleneck.
- Steps (9) to (11) ("Init") initialize on higher recursion levels.
- The loop starting at (13) ("Recursion") generates the recursive calls. For each possible type of string one such call is executed. Observe that for those calls we have $\tau_0 \subset \tau$. Hence the deepest recursion level is bounded by the maximal *length*.

ALGORITHM 13.2. `freeze&thaw` $\left(\tau_0, \left(L_i\right), \rho, \left(T_i\right)\right)$

```
     Input:  Sequence $\left(L_i\right)$ of languages, type $\tau_0$ s.t. $\tau_0 \subseteq \alpha$ for all $i \in \mathbb{N}$ and $\alpha \in L_i$.
    Output:  Choice $\rho$ of subsequence $\left(L_{\rho(i)}\right)$ with bottleneck $T_{\rho(i)}$
    Bottom:
       (1)   if $|L_i|$ is bounded then begin
       (2)       freeze $\left(\left(L_i\right), \left(L_i\right), \rho, L_0\right)$
       (3)       return $\rho$ and $L_{\rho(i)}$
       (4)       end
       (5)   if ind$(\tau_0, L_i)$ is unbounded then begin
       (6)       chose $\rho$ s.t. ind $\left(\tau_0, L_{\rho(i)}\right)$ is monotonous
       (7)       return $\rho$ and $(\tau_0)_{i \in \mathbb{N}}$
       (8)       end
      Init:
       (9)   Let $L_i' \subseteq L_i$ be maximum $\tau_0$-independent
      (10)   freeze $\left(\left(L_i\right), \left(L_i'\right), \rho, L_0\right)$ ;
      (11)   for all $i \in \mathbb{N}$ do $T_i := \emptyset$
      (12)   $S :=$ Type(dom$(L_0)$)
 Recursion:
      (13)   for all $\tau \in S_{\tau_0}^{L_0}$ with $\tau \neq \tau_0$ do begin
      (14)       for all i do $L_i' := \left(L_i\right)_\tau$
      (15)       freeze&thaw $\left(\tau, \left(L'_{\rho(i)}\right), \rho', \left(T'_{\rho(i)}\right)\right)$
      (16)       $\rho := \rho' \circ \rho$ ; $T_i := T_i \cup T_i'$
      (17)       freeze $\left(\left(L_{\rho(i)}\right), \left(T_{\rho(i)}\right), \rho', L_0'\right)$
      (18)       $\rho := \rho' \circ \rho$
      (19)       end ;
      (20)   return $\rho$ and $\left(T_i\right)$.
```

We give some explanations of this algorithm. Suppose we run our algorithm with $\tau_0 = \sqcup k$. If ind$(\sqcup k, .)$ is unbounded we will find a large language with many independent strings of length $= k$. This language then can be used to cover a small language $L_0$ with length$(L_0) \leq k$. See figure 13.2 for an example.

If *ind* is bounded (see fig. 13.3) the groundset of a maximum independent set of strings is bounded and one application of `freeze&thaw` without the recursive calls gives a subsequence where all these sublanguages are isomorphic. But if we look carefully at the rest of the languages there may be parts for which we do not have full control of what happens (indicated by "?"). For those parts we have to

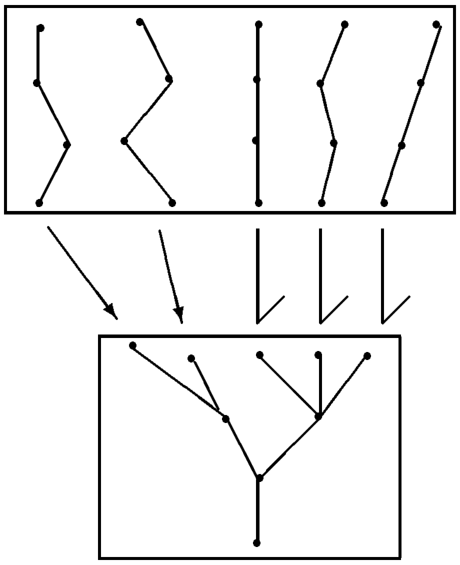

FIGURE 13.2. Many independent strings

apply `freeze&thaw` recursively. This leeds only to a finite recursion depth since the parts for which we apply recursion are "essentially" shorter.

A more detailed explanation and a proof of correctness of this algorithm is given in the discussion of the following two lemmas. Theorem 13.4 will then be an immediate consequence.

LEMMA 13.2. *Given as input a sequence of languages as required,* ***freeze&thaw*** *results in finitely many choices of subsequences and thus has a well defined output.*

PROOF. Let the input $\tau_0 = t_1 \cdots t_l$ and $k = |\{i \mid t_i = \sqcup\}|$, the number of undefined positions in $\tau_0$. We proceed by induction on $k$.

For $k = 0$ or $k = 1$ the statement is obvious. Let us suppose we have shown it for all $k' < k$.

If `freeze&thaw` returns before step (9) we are done. If not, we know that $|\mathrm{dom}\,(L_i')|$ and $|L_i'|$ are globally bounded, so step (10). runs correctly.

But now $|\mathrm{dom}\,(L_0)|$ is finite, too. So there are only finitely many $\tau$ for which the loop (13) is executed.

All these $\tau$ are nontrivial and fulfill $\tau_0 \subset \tau$. The number of undefined positions in $\tau$ is strictly less than $k$ and we may apply induction on each call in (15).

This proves the statement. □

LEMMA 13.3. *Given as input a sequence of languages as required, the output of* ***freeze&thaw*** *is a bottleneck.*

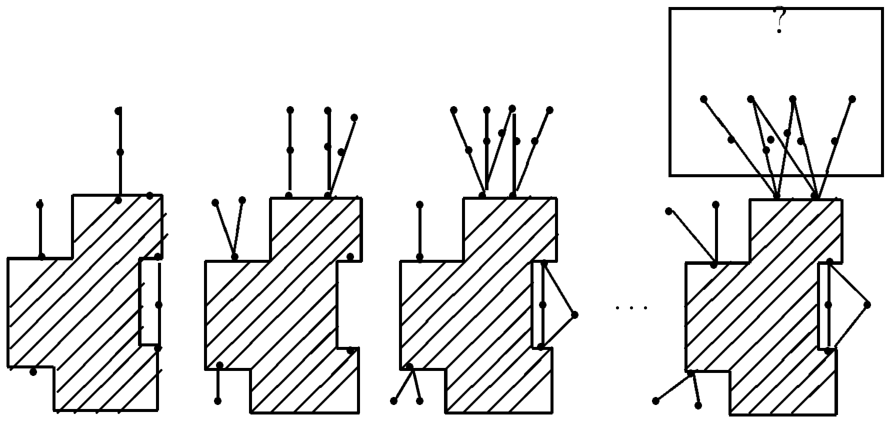

FIGURE 13.3. Subsequence with isomorphic restrictions

PROOF. To show the bottleneck properties we have to assign a $\tau_\alpha$ to all $\alpha \in L_{\rho(i)}$. If `freeze&thaw` stops before step (9) our choice is unique. Otherwise if there is $\tau \in T_{\rho(i)}$ s.t. $\alpha = \tau$ we choose this $\tau$.

If there is no such $\tau$ we choose $\tau_\alpha \in T_{\rho(i)}$ that is maximal with resp. to "$\subseteq$" .

Such a $\tau_\alpha$ always exists, since $L'_i$ was chosen to be maximum $\tau_0$-independent. So there must be $\beta \in L'_i$ with $\alpha \cap \beta \supset \tau_0$ . Such a $\tau_0$ has the appropriate *length*, since all elements of our languages have the same *length*. That shows property 1.

Property 2. follows from the last execution of step (17).

Property 3 holds since $\tau$ is only put into $T_i$ via step (3) or (5) on the deepest recursion level. If it is put in by step (3) it was unique, and in step (5) ind $\left(\tau, L_{\rho(i)}\right)$ is unbounded. □

Now we give an algorithm `bottleneck` that gives a bottleneck in the case that there are strings of different *length* and that this value is globally bounded by a constant, $\ell_0$ say.

ALGORITHM 13.3. $\texttt{bottleneck}_{\ell_0}\left(\left(L_i\right), \rho, \left(T_i\right)\right)$

**Input:** Sequence $\left(L_i\right)$ of languages such that $\text{length}\left(L_i\right) \leq \ell_0$ for all $i \in \mathbb{N}$.

**Output:** Choice $\rho$ of subsequence $\left(L_{\rho(i)}\right)$ with bottleneck $T_{\rho(i)}$

```
(1)  $\rho := id$
(2)  for all $i \in \mathbb{N}$ do $T_i := \emptyset$
(3)  for $\ell := 1$ to $\ell_0$ do begin
(4)      for all $i \in \mathbb{N}$ do $L_i^\ell := \{\alpha \in L_i \mid \text{length}(\alpha) = \ell\}$
(5)      freeze&thaw$\left(\sqcup\ell, L_{\rho(i)}^\ell, \rho', T_{\rho(i)}^\ell\right)$
(6)      $\rho := \rho' \circ \rho$
```

```
(7)           for all $i \in \mathbb{N}$ do $T_i := T_i \cup T_i^\ell$
(8)           end ;
(9)      freeze$\left(\left(L_{\rho(i)}\right), \left(T_{\rho(i)}\right), \rho', T_0\right)$
(10)     $\rho := \rho' \circ \rho$
(11)     return $\rho$ and $\left(T_i\right)$.
```

LEMMA 13.4. *Given as input a sequence of languages as required, the output of* `bottleneck` *is a bottleneck.*

PROOF. Clearly every call of `freeze&thaw` guaranties that for every $\alpha \in L_{\rho(i)}$ there is $\tau$ in $T_{\rho(i)}$ with $\tau \subseteq \alpha$. Since $\left|T_{\rho(i)}\right|$ is globally bounded `freeze` runs correctly and so $T_0$ has property 2 in Definition 13.2.

For property 3 observe that for $\tau \in T_0$ with length$(\tau) = l$ and such that ind$\left(\tau, L^l_{\rho(i)}\right)$ is unbounded, ind$\left(\tau, L_{\rho(i)}\right)$ is unbounded, too. □

## 13.4. Finding Related Languages.

LEMMA 13.5. *Every sequence $\left(L_i\right)$ of finite languages such that length$(L_i)$ is bounded is good.*

PROOF. After application of `bottleneck` with input $\left(L_i\right)$ we may assume w.l.o.g. that it has a bottleneck $T_0$. We have to show that there is $j$ s.t. $L_1 \preceq L_j$.

Choose $j$ s.t. ind$(\tau, L_j) >$ size$(L_1)$ for all $\tau \in T_0$ with $\tau \notin L_1$. Such a $j$ exists because of property 3 of the bottleneck.

We now have to find a partial mapping $\rho$ from dom$(L_j)$ to dom$(L_1)$.

For all $v \in$ dom$(T_0)$ we set $\rho(v) = v$.

For all $\tau \in T_0$ and all $\alpha \in L_1$ with $\tau_\alpha = \tau$ we choose inductively $\beta \in L_j$ with $\tau \subseteq \beta$ and such that $\rho$ is still undeclared for all $v \in$ dom$(\beta \setminus \tau) \cap$ dom$(L_j)$. Then we extend $\rho$ s.t. $\rho(\beta) = \alpha$.

This choice can always be done since all $\beta$ in question have the same *length* as $\alpha$ and $\tau$, and since ind$(\tau, L_j)$ is sufficiently large:

If $k$ the number of undefined positions in $\tau$ and $V_\tau =$ dom$(\{\beta \setminus \tau \mid \beta \in L_j\})$ we have that $|V_\tau| = k \cdot$ ind$(\tau, L_j)$. By construction $\rho$ cannot use more than size$(L_1)$ elements of dom$(L_j)$. So there is always some $\beta$ left over that we can use.

But now $\rho$ is a partial mapping from dom$(L_j)$ to dom$(L_1)$ s.t for every $\alpha \in L_1$ there is $\beta \in L_j$ with $\rho(\beta) = \alpha$ and thus $L_1 \preceq L_j$. □

**13.5. Finding a Fixed Language as String Minor.** We will show the following theorem:

THEOREM 13.5. *Every property of finite non-repetitive languages which is hereditary with resp. to $\preceq$ has a decision algorithm which runs in polynomial time.*

With Theorems 6.1 and 13.1 it will follow immediately from

LEMMA 13.6. *Let $L_1$ and $L_2$ be finite languages and $k = size(L_1) \geq 3$. Then there is a constant $c_1$ depending only on $L_1$ and an algorithm to decide whether or not $L_1 \preceq L_2$ holds that runs in*

- *$c_1 \cdot size(L_2) \cdot |dom(L_2)|^k + c_1$ sequential time*
- *constant time with $O\left(size(L_2) \cdot |dom(L_2)|^k\right)$ processors.*

To prove this lemma we show that there is always a "small" language $L_0$ that is between $L_1$ and $L_2$ if $L_1 \underset{lang}{\preceq} L_2$, and that fulfills $L_0 \underset{\leftrightarrow}{\prec} L_2$.

LEMMA 13.7. *Let $L_1 \preceq L_2$ be finite formal languages and $k = size(L_1) \geq 3$. Then there is a language $L_0$ which fulfills:*

(1) $L_1 \preceq L_0$
(2) $L_0 \underset{\leftrightarrow}{\prec} L_2$
(3) $|L_0| \leq k$
(4) $dom(L_0) \leq k$
(5) $size(L_0) \leq k^2$.

PROOF. Let $\rho$ be a string morphism which gives $L_1 \underset{lang}{\preceq} L_2$ and let $\alpha_1^1, \ldots, \alpha_1^k$ be the elements of $L_1$ . There are strings $\alpha_2^1, \ldots, \alpha_2^k$ in $L_2$ such that $\alpha_1^i \underset{\leftrightarrow}{\prec} \rho(\alpha_2^i)$. Set $L_0 = \{\alpha_2^i\}$ and $\rho_0 = \rho\Big|_{L_0}$ . $L_0$ and $\rho_0$ obviously have all properties desired. □

PROOF OF LEMMA 13.6. First we give an algorithm and then we shortly describe what is does.

ALGORITHM 13.4. Test$_{L_1}$

```
 Input:   Language $L_2$.
Output:   true if $L_1 \underset{lang}{\preceq} L_2$, false otherwise.
Prepro:
   (1)    Find all possible $L_0$ according to points 1, 3, and 4 and 5 of Lemma 13.7.
        Denote the corresponding set of languages with $S$.
   (2)    $ok$ := false
```

**Find:**

```
 (3)  for all $L_0 \overset{\prec}{\leftrightarrow} L_2$ with 4 do parallel begin
 (4)      for all $K_0 \in S$ do parallel begin
 (5)          lang-ok:= true
 (6)          for all $\alpha \in K_0$ do parallel begin
 (7)              string-ok:= false
 (8)              for all $\beta \in L_0$ do parallel begin
 (9)                  if $\mathtt{scan}^{\mathtt{par}}{}_{A_k}(\alpha, \beta)$ then string-ok:= true
(10)                  end
(11)              if ¬ string-ok then lang-ok:= false
(12)              end
(13)          if lang-ok then ok:= true
(14)          end
(15)      end
(16)  return ok
```

$L_1 \preceq L_2$ holds iff there is $L_0$ as specified in Lemma 13.7. There are only finitely many isomorphism types of such languages, so there are also finitely many with chosen permutation of the elements. For each such $L_0$ we test whether or not $L_1 \preceq L_0$ holds.

This preprocessing, **Prepro**, depends only on $L_1$ .

Then we have to test all $\binom{|\mathrm{dom}(L_2)|}{k} = O\left(|\mathrm{dom}\,(L_2)|^k\right)$ induced sublanguages of $L_2$ with at most $k$ symbols. Given a subset of $\mathrm{dom}\,(L_2)$ of that size we may calculate the induced sublanguage $L_0$ of $L_2$ in time $O\,(\mathrm{size}\,(L_2))$. We may also assume that we rename the symbols in $\mathrm{dom}\,(L_0)$ to $1, \ldots, |domL_0|$. Now we test for each language $K_0$ in $S$ if $K_0 \overset{\prec}{\leftrightarrow} L_0$. This can be done in time $O\left(|K_0| \cdot \mathrm{size}\,(L_0)\right)$. In total we obtain a running time of

$$(13.6)\quad O\left(|S| \cdot |K_0| \cdot \binom{|\mathrm{dom}\,(L_2)|}{k} \cdot \mathrm{size}\,(L_0)\right) \leq c_1 \cdot |\mathrm{dom}\,(L_2)|^k \cdot \mathrm{size}\,(L_2)$$

All this can be done efficiently in parallel. □

Lemma 13.6 also indicates that the complexity strongly depends on the size of the underlying alphabet — the size of the language $L_2$ itself only contributes a linear term.

Finally all this gives us

THEOREM 13.6. *Every property of finite languages over a fixed finite alphabet which is hereditary with resp. to $\preceq$ has a decision algorithm which runs in linear time.*

## 14. Posets

Now we turn to finite posets as objects of an ordered structure.

**14.1. The Chain Minor Relation.** Let $P = (V, <)$ and $P' = (V', <')$ be posets. We say $P$ is **chain minor** of $P'$, $P \preceq P'$, if there is a partial mapping $\rho\colon\ V' \to V$ that has the following property:

> *For every chain $C$ in $P$ there is a chain $C'$ in $P'$ such that $\rho\big|_{C'}$ is an isomorphism of chains.*

$C'$ is then called a **lift** of $C$ and $\rho$ is called a **chain morphism**.

Here $\rho\big|_{C'}$ denotes the partial mapping induced on $P'\big|_{C'}$, the order restricted to the groundset of $C'$.

Observe that every chain morphism is onto and that $\preceq$ defines a qo on any set of posets.

Figure 14.1 gives a non-trivial example for this relation.

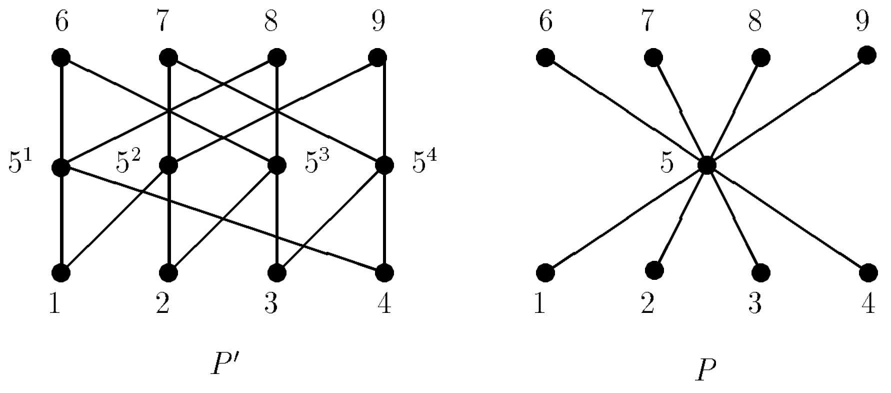

FIGURE 14.1. A Chain Minor that is No Suborder

Observe that in this example $P$ is not a suborder of $P'$, and that there is no other poset $P''$ between them. So $P'$ covers $P$ with respect to $\preceq$. So in general we can not have nice descending chains with small intermediate steps from the larger posets to the smaller ones as we had for graph minors for example.

**14.2. Motivation from Scheduling.** The chain minor relation has been introduced by Möhring and Müller in [MM92], where it is used to generalize certain approaches in the theory of scheduling stochastic project networks.

We will restrict ourselves to precedence constrain scheduling problems, i.e., for us such a problem is given by a poset $P = (V, <)$ where $V$ is a set of jobs and $v < w$ means $v$ must be scheduled before $w$. A schedule of $P$ is then an assignment of time intervals $\left[l_v, r_w\right]$ to the jobs that is consistent with $<$ or, equivalently, an interval extension $Q$ of $P$. In addition to $P$ side constraints — such as processing times, due dates, resource requirements for individual jobs or groups of jobs — might be given, but we will not go into the details of such specialized problems. See e.g. [MR89] for an overview.

Usually there are several distinct parameters of a certain schedule that are considered. We will restrict ourselves to the maximum completion time and the number of processors or machines. The maximum **completion time** is the largest interval endpoint needed (provided all are integers and the smallest one is 0); the **number of machines** can be defined as the width of the corresponding interval order.

Our relation is useful for scheduling problems because, loosely spoken, the chains are those objects that cause restrictions for the jobs to be scheduled: the processing times of a chain leading to a certain job sum up to a lower bound for the beginning of that job.

THEOREM 14.1. *Let $P$ be an arbitrary poset and $Q$ be an interval order. If $P$ is a chain minor of $Q$ then it is suborder, too.*

COROLLARY 14.1. *Let $P \preceq P'$ and $Q$ be a schedule of $P'$. Then $Q$ is a schedule of $P$, too.*

PROOF. We have $P \preceq P' \preceq Q$ and so $P \preceq Q$ by transitivity of $\preceq$. But then Theorem 14.1 immediately gives the claim. □

To prove Theorem 14.1 we need a lemma.

LEMMA 14.1. *Let $\rho_0$ be a chain morphism from an interval order $Q$ to a poset $P = (V, <)$, and let $v_0 \in V$ be minimal with $\left|\rho_0^{-1}(v_0)\right| > 1$. Then $\rho_0$ can be modified to a chain morphism $\rho$ such that $\rho^{-1}(v) \subseteq \rho_0^{-1}(v)$ for all $v \in V$ and such that $\left|\rho_0^{-1}(v_0)\right| = 1$.*

PROOF. Let $Q$ be given by an interval representation $\left[l_w, r_w\right]$ for all $w \in W$. We give an algorithm to modify $\rho_0$.

ALGORITHM 14.1.

```
(1)  choose $w_0 \in \rho_0^{-1}(v_0)$ s.t. $r_{w_0}$ is minimal.
(2)  $\rho^{-1}(v_0) := \{w_0\}$
(3)  **for all** $v \in V \setminus \{v_0\}$ **do begin**
(4)      **if** $v \succ\!\!- v_0$ **then begin**
(5)          **for all** $w \in \rho_0^{-1}(v)$ **do begin**
(6)              **if** $l_w < r_{w_0}$ **then** $\rho_0^{-1}(v) := \rho_0^{-1}(v) \setminus \{w\}$
(7)              **end**
```

```
(8)         end
(9)      $\rho^{-1}(v) := \rho_0^{-1}(v)$
(10)     end
```

We have to show that for each maximal chain $C$ in $P$ there is still some chain $C'$ in $Q$ that is a lift of $C$ with respect to $\rho$. Let $C''$ denote the lift of $C$ with respect to $\rho_0$. If none of the vertices in $C''$ is touched by our algorithm there is nothing to show.

$C''$ is only involved if it either contains $w \in \rho_0^{-1}(v_0)$ with

(1) $r_w \geq r_{w_0}$, or
(2) $l_w < r_{w_0}$ for $w \in \rho_0^{-1}(v)$ with $v_0 \prec\!\!\cdot\; v$.

For 1 we simply replace $w$ by $w_0$ in $C''$ to obtain a new chain $C'$ since $r_{w_1} \leq l_{w_0}$ for all $w_1 \in \rho_0^{-1}(v_1)$ with $v_1 < v_0$.

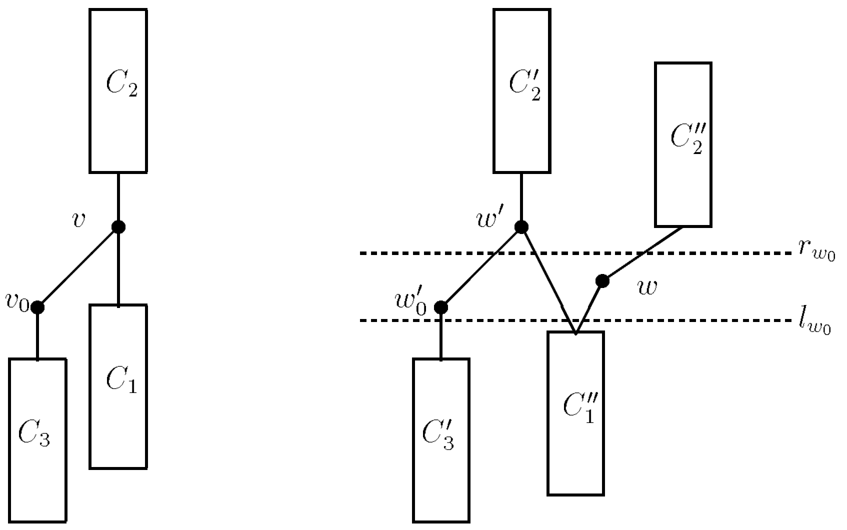

FIGURE 14.2. Pasting Chains

For 2 we have $C_1$, $C_2$, $C_1''$, and $C_2''$ such that $C = C_1 v C_2$ and $C'' = C_1'' w C_2''$, see figure 14.2.

Now let $C_3$ be a chain in $P$ such that $C_3 v_0 v C_2$ is maximal. Such a chain always exist, since $v$ covers $v_0$. This chain has a lift $C_3' w_0' w' C_2'$ such that $w_0' \in \rho_0^{-1}(v_0)$ and $w' \in \rho_0^{-1}(v)$. Now $w'$ fulfills $r_{w_0} \leq l_{w'}$ and so it is not eliminated in this iteration. But then $C_1'' w' C_2'$ is still a valid lift for $C$. $\square$

Now we give an algorithm to solve the problem as a whole.

ALGORITHM 14.2.

```
Input:   Poset $P = (V, <)$, interval order $Q = (W, <)$ with given interval repre-
         sentation $\left[l_w, r_w\right]$ for $w \in W$, and $\rho^{-1}(v) \subseteq W$ for all $v \in V$ that defines a
         chain morphism $\rho$ from $Q$ to $P$.
Output:  Embedding $\varphi$ of $P$ into $Q$.
  (1)    for all $v_0 \in V$ in a linear extension of $P$ do begin
  (2)        choose $w_0 \in \rho^{-1}(v_0)$ s.t. $r_{w_0}$ is minimal.
  (3)        $\rho^{-1}(v_0) := \{w_0\}$
  (4)        for all $v \in V$ s.t. $v \succ v_0$ do begin
  (5)            for all $w \in \rho^{-1}(v)$ do begin
  (6)                if $l_w \leq r_{w_0}$ then $\rho^{-1}(v) := \rho^{-1}(v) \setminus \{w\}$
  (7)                end
  (8)            end
  (9)        end
 (10)    for all $v \in V$ do $\varphi(v) := \rho^{-1}(v)$
```

The result of this algorithm when it is applied to the example of Figure 14.1 is shown in Figure 14.3. Here the boxes symbolize time slots for the jobs.

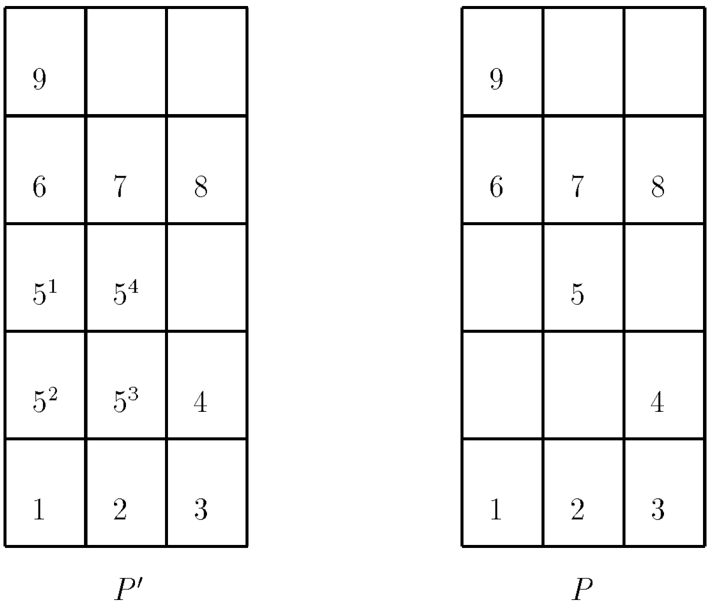

FIGURE 14.3. Two Schedules

The proof of Theorem 14.1 is now an immediate consequence of the following lemma.

LEMMA 14.2. *Algorithm* 14.2 *is correct. Provided the transitive reduction of $P$ is given it can be implemented such that it has a running time $O(m+n+p)$ where $n$ is the number of covering relations in $P$, $m = \sum_{v\in V} \left|\rho_0^{-1}(v)\right|$ and $p$ is the maximal difference between two endpoints of intervals for $Q$.*

PROOF. For the correctness observe that with Lemma 14.1 at the end of our algorithm $\rho$ is a chain morphism with $|\rho^{-1}(v)| = 1$ for all $v \in V$. So $\varphi$ is well-defined and an embedding.

To obtain the running time we have to process the information needed efficiently. For every $v \in V$ we assume that we have the values $r_w$ for $w \in \rho^{-1}(v)$ in doubly linked list that is sorted. Thus we may assume that we always have random access to the smallest among them to make the choice in (2).

In addition we assume that we have the corresponding values $l_w$ in a sorted array. Now we can implement loop (5) by incrementally looking at the minimal element in this array until we find one that is $\geq r_{w_0}$. So we access all $w$ with $l_w < r_{w_0}$ in constant time per each and can update the list of $r_w$-values accordingly in constant time per update. Since every $w$ is involved at most once in such an update the total time needed for all updates is $O(m)$.

To initialize these lists resp. arrays we have to sort all values $l_w$ and $r_w$. This can be done in time $O(m+p)$ with e.g. bucketsort. □

With Theorem 14.1 we will be able to determine the complexity status for the following problem:

PROBLEM 14.1. CHAIN MINORS

**Instance:** posets $P$ and $Q$
**Question:** Is $P$ a chain minor of $Q$?

We achieve the following proposition:

PROPOSITION 14.1. CHAIN MINORS *is* ***NP****-hard.*

PROOF. We give a reduction from the following problem, see problem no. SS9 in [GJ79].

PROBLEM 14.2. PRECEDENCE CONSTRAINED SCHEDULING

**Instance:** Poset $P = (V, <)$ and values $m$ and $l$.
**Question:** Is there a scheduling for $P$ in time $l$ on $m$ machines such that all task $v \in V$ are performed in 1 time unit?

This problem has been shown to be **NP**-complete by Ullman in [Ull75].

We may assume that for this problem $m$ and $l$ are less than $n = |V|$ since otherwise the problem is easy to solve. So we may construct the weak order $Q = C_l\left[\{A_m\}_{1\leq i\leq l}\right]$ in time $O\left(n^2\right)$. $P$ has a $l$-$m$-schedule iff $P \preceq Q$.

But now if we are able to decide $P \preceq Q$, Theorem 14.1 shows that $P \underset{sub}{\preceq} Q$, too. □

Finally we remark that it is not clear whether or not a given partial mapping is a chain morphism can be tested in polynomial time. That is because the definition makes a statement about potentially exponentially many objects — the (maximal) chains of $P$. So it is not even clear if the decision problem for $P \preceq Q$ is in **NP**.

But it is in **NP** if we restrict ourselves to the class of orders with height not exceeding 3. This is so because then the amount of maximal chains is roughly bounded by $|P|^3$. Since Precedence Constrained Scheduling has been shown by Lenstra and Rinnooy Kan, see [LR78], to be **NP**-complete if the input $l$ is restricted to $l \leq 3$ (and thus $P$ having height at most 3) we obtain the following corollary:

Corollary 14.2. Chain Minors *is **NP**-complete if the inputs $P$ and $Q$ are restricted to have height at most* 3.

**14.3. Well Quasi Ordering Finite Posets.** Our first aim is the following theorem.

Theorem 14.2. *Any set of finite posets is a wqo with respect to $\preceq$ .*

Proof. For a poset $P$ every chain defines a string in $V^*$ in a natural way. Let $L_{max}(P)$ be the language given by the maximal chains of $P$.

It is clear that

$$L_{max}(P) \preceq L_{max}(P') \Longrightarrow P \preceq P', \tag{14.1}$$

and that $L_{max}(P)$ is non-repetitive. So Theorem 13.1 proves the claim. □

**14.4. Algorithms for Posets.**

Theorem 14.3. *Every property of finite posets which is hereditary with resp. to $\preceq$ has a decision algorithm which runs*

— *in sequential polynomial time*
— *in constant time on a CRCW PRAM and uses polynomially many processors.*

In contrast to the argumentation above we can not use $L_{max}$ to prove Theorem 14.3 since it may be exponentially large compared with the poset. But we can use similar arguments as we used for formal languages.

LEMMA 14.3. *Let $P_1 \preceq P_2$ be posets. Then there is a poset $P_0$ which fulfills:*

(1) $P_1 \preceq P_0$
(2) *$P_0$ is induced suborder of $P_2$*
(3) $|P_0| \leq size(L_{max}(P_1))$

PROOF. Let $\rho$ be a chain morphism which gives $P_1 \preceq P_2$ and let $C_1^1, \ldots, C_1^k$ be the maximal chains of $P_1$ . There are chains $C_2^1, \ldots, C_2^k$ in $P_2$ such that $\rho\big|_{C_2^i}$ is an isomorphism of chains for all $i$. Set $V_0 = \bigcup C_2^i$, $P_0 = P_2\big|_{V_0}$ and $\rho_0 = \rho\big|_{V_0}$ . $P_0$ and $\rho_0$ obviously have all properties desired. $\square$

LEMMA 14.4. *Let $P_1$ and $P_2$ be finite posets and $k = size(L_{max}(P_1)) \geq 3$. Then there is a constant $l$ depending only on $P_1$ and an algorithm to decide whether or not $P_1 \preceq P_2$ holds that runs*

— *in $O\left(k^2 \cdot |P_2|^k + l\right)$ sequential time*
— *in $O(k^2 + l)$ time on a CRCW PRAM with $O\left(|P_2|^k\right)$ processors.*

PROOF. First we give an algorithm and then we shortly describe what it does.

ALGORITHM 14.3. Test$_{P_1}$

```
Input:   Poset $P_2$.
Output:  true if $P_1 \preceq P_2$, false otherwise.
Prepro:
   (1)   Find all possible $P_0$ according to Lemma 14.3.
   (2)   Encode them as 0-1-strings.
   (3)   Store them in a binary tree $T$.
 Init:   Calculate the matrix of the transitive closure of $P_2$.
   (4)   $ok$ := false
 Find:
   (5)   for all $V_0 \subseteq V_2$ with $|V_0| \leq k$ do parallel begin
   (6)       $P_0 := P_2\big|_{V_0}$
   (7)       if $P_0 \in T$ then $ok$ := true
   (8)       end
   (9)   return $ok$
```

$P_1 \preceq P_2$ holds iff there is $P_0$ as specified in Lemma 14.3. There are only finitely many isomorphism types of such posets, so there are also finitely many with chosen permutation of the elements. For each such $P_0$ we test whether or not $P_1 \preceq P_0$ holds.

We encode each of these posets as matrix of relations – put a 1 at place $(i, j)$ if $x_i < x_j$ holds. Each matrix can then be seen as string of 0's and 1's with length $k^2$.

The set of all these strings can be handled efficiently with a binary tree of height $k^2$, such that each string is represented by a path from the root to a leaf.

This preprocessing, **Prepro**, depends only on $P_1$ .

Then, in an initialization phase, **Init**, we calculate the matrix for $P_2$. This can be done in time $O\left(|P_2|^2\right)$, but since we may assume $k \geq 2$ this makes no problem.

Then we have to test all $\binom{|P_2|}{k} = O\left(|P_2|^k\right)$ induced suborders of $P_2$ with at most $k$ elements. Given a subset of $P_2$ of that size we may calculate its matrix in $O(k^2)$ time and then scan the binary tree given above if it is valid or not.

Thus each such suborder can be determined in $O(k^2)$ time.

All this can be done efficiently in parallel, the only exception is the initialization phase. To do this in constant time we need $O\left(|P_2|^3\right)$ processors. □

Notice that if we assume a CREW instead of a CRCW PRAM the only thing which makes difficulties is the communication of results. To know if any of the posets is of a valid type we need $O\left(\log |P_2|^k\right) = O\left(\log |P_2|\right)$ time.

Proof of Theorem 14.3. Let $E$ be a property of finite posets which is hereditary with resp. to $\preceq$ . By Theorem 14.2 we know that the set of minimal obstructions for $E$ is finite, $\{P_1, \ldots, P_l\}$ say.

Set $k_{max} = \max\{\text{size}(L_{max}(P_i)), 3\}$ . With Lemma 14.4 we know that the test for

$$\left(P_1 \preceq P\right) \vee \cdots \vee \left(P_l \preceq P\right) \tag{14.2}$$

can be done in $O\left(|P|^{k_{max}}\right)$ time resp. in constant time with $O\left(|P|^{k_{max}}\right)$ processors. □

# Index

# Bibliography

[ACP87] Stefan Arnborg, Derek Corneil, and Andrej Proskurowski, *Complexity of finding embeddings in and k-tree*, SIAM J. Algebraic Discrete Methods **8** (1987), no. 2, 277–284.

[ADKP89] K. Abrahamson, N. Dadoun, D. G. Kirkpatrick, and T. Przytycka, *A simple parallel tree contraction algorithm*, J. Algorithms **10** (1989), 287–302.

[ALS91] Stefan Arnborg, Jens Lagergren, and Detlef Seese, *Easy problems for tree-decomposable graphs*, J. Algorithms **12** (1991), 308–340.

[AO89] Algorithms and Order (Ivan Rival, ed.), Kluwer Acad. Publ., Dordrecht, 1989.

[APS89] Advances in Project Scheduling (R. Slowinski and J. Weglarz, eds.), Elseviers Science Pub., Amsterdam, 1989.

[Beh88] G. Behrendt, *Maximal antichains in partially ordered sets*, Ars Combin. **25** (1988), no. C, 149–157.

[BK91] Hans L. Bodlaender and Ton Kloks, *Better algorithms for the pathwidth and treewidth of graphs*, in [ICA91] (1991), 544–555.

[BW85] E. Bender and Herbert S. Wilf, *A theoretical analysis of backtracking in the graph-coloring case*, J. Algorithms **6** (1985), 275–282.

[CM92] Bruno Courcelle and M. Mosbah, *Monadic second-order evaluations on tree-decomposable graphs*, in [WG91] (1992), 13–24.

[CPS85] D. G. Corneil, Y. Perl, and L. Stewart, *A linear recognition algorithm for cographs*, SIAM J. Comput. **14** (1985), 926–934.

[Dam90] Peter Damaschke, *Induced subgraphs and well-quasi-ordering*, J. Graph Theory **14** (1990), no. 4, 427–435.

[Dil58] Robert P. Dilworth, *Some combinatorial problems on partially ordered sets*, republished in [DIL90] (1958), 13–18.

[DIL90] The Dilworth Theorems, Selectet Papers of Robert P. Dilworth (Kenneth P. Bogard, Ralph Freese, and Joseph P. S. Kung, eds.), Birkhäuser Verlag, Boston, Basel, Berlin, 1990.

[DPR81] D. Duffus, M. Pouzet, and I. Rival, *Complete ordered sets with no infinite antichains*, Discrete Math. **35** (1981), 39–52.

[Ebb79] Heinz-Dieter Ebbinghaus, Einführung in die Mengenlehre, Wissenschaftliche Buchgeselllschaft, Darmstadt, 1979.

[Fel89] Michael R. Fellows, *The Robertson-Seymour theorems: a survey of applications*, see [GA89], 1989, pp. 1–18.

[FKL88] Michael R. Fellows, Nancy G. Kinnersley, and Michael A. Langston, *Finite-basis theorems and a computation-integrated approach to obstruction set isolation*, Tech. report, Washington State University, 1988.

[FL85] Michael R. Fellows and Michael A. Langston, *Nonconstructive advances in polynomial-time complexity*, Inform. Process. Lett. (1985).

[FL88a] Michael R. Fellows and Michael A. Langston, *Layout permutation problems and well-partially-ordered sets*, Proceedings of the $5^{th}$ MIT Conference on Advanced Research in VLSI (Jonathan Allen and F. Thomson Leighton, eds.), 1988, pp. 315–327.

[FL88b] Michael R. Fellows and Michael A. Langston, *Nonconstructive tools for proving polynomial-time decidability*, J. Assoc. Comput. Mach. **35** (1988), no. 3, 727–739.

[FL88c] Michael R. Fellows and Michael A. Langston, *On search, decision and the efficiency of polomial-time algorithms*, Washington State University (1988).

[FL89] Michael R. Fellows and Michael A. Langston, *An analogue of the Myhill-Nerode theorem and its use in computing finite-basis characterizations*, see [FOC89], 1989, pp. 520–525.

[FL92] Michael R. Fellows and Michael A. Langston, *On well-partial-order theory and its application to combinatorial problems of VLSI design*, SIAM J. Disc. Math. **5** (1992), no. 1, 117–126.

[FOC85] 26th Annual Symposion On Foundations of Computer Science, IEEE, The Institute of Electrical and Electronics Engineers, IEEE Computer Society Press, 1985.

[FOC89] 30th Annual Symposion On Foundations of Computer Science, IEEE, The Institute of Electrical and Electronics Engineers, IEEE Computer Society Press, 1989.

[FOC90] 31th Annual Symposion On Foundations of Computer Science, IEEE, The Institute of Electrical and Electronics Engineers, IEEE Computer Society Press, 1990.

[GA89] Graphs and Algorithms (R. B. Richter, ed.), American Math. Soc., Contemp. Math., 89, 1989, Proceedings of the AMS-IMS-SIAM joint Summer Research Conference 1987.

[GJ79] Michael R. Garey and David S. Johnson, Computers and Intractability, W. H. Freeman and Company, New York, 1979.

[GM91] Graph Minors (Neil Robertson and Paul Seymour, eds.), American Math. Soc., Providence, RI, 1991, Proceedings of the AMS-IMS-SIAM Joint Summer Research Conference, Seattle WA, June 1991.

[GO85] Graphs and Orders (Ivan Rival, ed.), D. Reidel Publishing Company, Dordrecht, 1985.

[Gol80] Martin C. Golumbic, Algorithmic Graph Theory and Perfect Graphs, Academic Press, New York, 1980.

[Gus89] Jens Gustedt, *On the pathwidth of chordal graphs*, Discrete Appl. Math. (1989), accepted for publication.

[Gus91] Jens Gustedt, *Well-quasi-ordering finite posets and formal languages*, Tech. Report 290, Technische Universität Berlin, 1991, submitted.

[Gus92] Jens Gustedt, *Well-quasi-ordering finite posets*, accepted for publication in [GM91] (1992), extended abstract of [Gus91].

[Hig52] G. Higman, *Ordering by divisibility in abstract algebras*, Proc. London Math. Soc. **2** (1952), 326–336.

[HM87] Michel Habib and Rolf H. Möhring, *On some complexity properties of $N$-free posets and posets with bounded decomposition diameter*, Discrete Math. **63** (1987), 157–182.

[ICA91] Proceedings of the 18'th International Colloquium on Automata, Languages and Programming, Springer-Verlag, 1991, Lecture Note in Comp. Sc., vol. 510.

[Kru60] J. B. Kruskal, *Well quasi ordering, the tree theorem and Vazsonyi's conjecture*, Trans. Am. Math. Soc. **95** (1960), 210–225.

[Kru72] J. B. Kruskal, *The theory of well-quasi-ordering: a frequently discovered concept*, J. Combin. Theory Ser. A **13** (1972), 297–305.

[Lag90] Jens Lagergren, *Efficient parallel algorithms for tree-decomposition and related problems*, see [FOC90], 1990.

[LMP85] J. D. Lawson, Michael Mislove, and Hilary A. Priestley, *Infinite chains in semilattices*, Order **2** (1985), 275–290.

[LO92] R. Lin and S. Olariu, *An NC recognition algorithm for cographs*, Discrete Appl. Math. (1992), accepted for publication.

[LR78] J. K. Lenstra and A. G. H. Rinnooy Khan, *Complexity of scheduling under precedence constraints*, Operations Res. **26** (1978), 22–35.

[Mil85] E. C. Milner, *Basic wqo- and bqo-theory*, [GO85], 1985, pp. 487–502.

[MM92] Rolf H. Möhring and Rudolf Müller, *A combinatorial approach to obtain bounds for stochastic project neworks*, Tech. report, Technische Universität Berlin, 1992.

[Möh89] Rolf H. Möhring, *Computationally tractable classes of ordered sets*, [AO89], 1989, pp. 105–194.

[MR85] Gary L. Miller and John H. Reif, *Parallel tree contraction and its application*, in [FOC85] (1985), 478–489.

[MR89] Rolf H. Möhring and Franz J. Radermacher, *The order-theoretic approach to scheduling: The deterministic case*, in [APS89] (1989), 29–66.

[NR87] Jaroslav Nešetřil and Vojtěch Rödl, *Complexity of diagrams*, Order **3** (1987), 321–330.

[NW63] C. St. J. A. Nash-Williams, *On well-quasi-ordering finite trees*, Math. Proc. Cambridge Philos. Soc. **59** (1963), 833–835.

[PFV90] Paths, Flows, and VLSI-Layout (Bernhard Korte, László Lovasz, Hans Jürgen Prömel, and Alexander Schrijver, eds.), Springer-Verlag, 1990.

[Pou85] M. Pouzet, *Applications of well quasi-ordering and better quasi-ordering*, [GO85], 1985, pp. 503–519.

[PS92a] Hans Jürgen Prömel and Angelika Steger, *Allmost all Berge graphs are perfect*, Combinatorics, Probability and Computing (1992), accepted for publication.

[PS92b] Hans Jürgen Prömel and Angelika Steger, *Coloring clique-free graphs in linear expected time*, Random Structures and Algorithms (1992), accepted for publication.

[Ree91] Bruce A. Reed, *Finding approximate separators and computing tree width quickly*, private communication (1991).

[Reu91] K. Reuter, *The jump number and the lattice of maximal antichains*, Discrete Math. (1991).

[RS83a] Neil Robertson and Paul Seymour, *Graph minors I, excluding a forest*, J. Combin. Theory Ser. B **35** (1983), 39–61.

[RS83b] Neil Robertson and Paul Seymour, *Graph minors III, planar tree-width*, J. Combin. Theory Ser. B **36** (1983), 49–64.

[RS85a] Neil Robertson and Paul Seymour, *Graph minors – a survey*, Surveys in Combinatorics (Glasgow 1985) (I. Anderson, ed.), Cambridge Univ. Press, Cambridge-New York, 1985, pp. 153–171.

[RS85b] Neil Robertson and Paul Seymour, *Graph minors XI, distance on a surface*, *Manuscript* (1985).

[RS86a] Neil Robertson and Paul Seymour, *Graph minors II, algorithmic aspects of tree-width*, J. Algorithms **7** (1986), 309–322.

[RS86b] Neil Robertson and Paul Seymour, *Graph minors V, excluding a planar graph*, J. Combin. Theory Ser. B **41** (1986), 92–114.

[RS86c] Neil Robertson and Paul Seymour, *Graph minors VI, disjoint paths across a disc*, J. Combin. Theory Ser. B **41** (1986), 115–138.

[RS86d] Neil Robertson and Paul Seymour, *Graph minors XII, excluding a non-planar graph*, *Manuscript* (1986).

[RS86e] Neil Robertson and Paul Seymour, *Graph minors XIII, the disjoint paths problem*, *Manuscript* (1986).

[RS87] Neil Robertson and Paul Seymour, *Graph minors XIV, taming a vortex*, *Manuscript* (1987).

[RS88a] Neil Robertson and Paul Seymour, *Graph minors VII, disjoint paths on a surface*, J. Combin. Theory Ser. B **45** (1988), 212–254.

[RS88b] Neil Robertson and Paul Seymour, *Graph minors XV, Wagner's conjecture*, *Manuscript* (1988).

[RS89] Neil Robertson and Paul Seymour, *Graph minors XVI, well-quasi-ordering on a surface*, *Manuscript* (1989).

[RS90a] Neil Robertson and Paul Seymour, *Graph minors IV, tree-width and well-quasi-ordering*, J. Combin. Theory Ser. B **48** (1990), 227–254.

[RS90b] Neil Robertson and Paul Seymour, *Graph minors IX, disjoint crossed paths*, J. Combin. Theory Ser. B **49** (1990), 40–77.

[RS90c] Neil Robertson and Paul Seymour, *Graph minors VIII, a Kuratowski theorem for general surfaces*, J. Combin. Theory Ser. B **48** (1990), 255–288.

[RS90d] Neil Robertson and Paul Seymour, *An outline of a disjoint paths algorithm*, [PFV90] (1990), 267–292.

[RS91] Neil Robertson and Paul Seymour, *Graph minors X, obstructions to tree-decomposition*, J. Combin. Theory Ser. B **52** (1991), 153–190.

[Ste92] Angelika Steger, *An $O(1)$ average time algorithm for perfectness*, private communication, 1992.

[Ull75] J. D. Ullman, *NP-complete scheduling problems*, J. Comput. System Sci. **10** (1975), 384–393.

[WG91] Graph-Theoretic Concepts in Computer Science (G. Schmidt and R. Berghammer, eds.), Springer-Verlag, 1991, 17th International Workshop WG '91.

[Wil84] H. S. Wilf, *Backtrack: an $O(1)$ expected time algorithm for the graph coloring problem*, Inform. Process. Lett. **18** (1984), 119–121.

[Wil86] Herbert S. Wilf, Algorithms and Complexity, Prentice-Hall International, Inc., 1986.

## Lebenslauf

| | |
|---|---|
| 10.4. 1960 | geboren in Flensburg |
| 1965 – 1978 | Schulbesuch in Paris, Bonn, Wilhelmshaven und wieder in Bonn. |
| Juni 1978 | Abitur am Helmholtz-Gymnasium, Bonn-Duisdorf. |
| WS78/79 – WS79/80 | Chemie-Studium an der Universität Bonn |
| 2.1. 1980 – 30.4. 1981 | Zivildienst am Institut für Anästhesiologie der Universität Bonn |
| WS81/82 – SS87 | Mathematik-Studium an der Universität Bonn |
| WS84/85 – SS87 | Studentische Hilskraft mit Tutorentätigkeit am Institut für Mathematik der Universität Bonn |
| Juli 1987 | Diplom in Mathematik mit Nebenfach Philosophie |
| 1.10. 1987 – 31.12. 1990 | wissenschaftlicher Mitarbeiter im Fachbereich Mathematik der Technischen Universität Berlin |
| Sep. 1990 | Gastaufenthalt am CRIM, Montpellier, Frankreich |
| seit 1.1. 1991 | Wissenschaftlicher Mitarbeiter im DFG-Projekt Algorithmische Ordnungstheorie |
| Sep. 1991 | Gastaufenthalt am LIRMM, Montpellier, Frankreich |
| Nov. 1991 | Gastaufenthalt Rijksuniversiteit Utrecht |