DOCTORAL THESIS IN COMPUTER SCIENCE
STOCKHOLM, SWEDEN 2014

On Difficult Topics in Theoretical Computer Science Education

EMMA ENSTRÖM

KTH ROYAL INSTITUTE OF TECHNOLOGY
COMPUTER SCIENCE AND COMMUNICATION

On Difficult Topics in Theoretical Computer Science Education

EMMA ENSTRÖM

Doctoral Thesis
Stockholm, Sweden 2014

TRITA-CSC-A 2014:15
ISSN 1653-5723 KTH School of Computer Science and Communication
ISRN-KTH/CSC/A–14/15-SE SE-100 44 Stockholm
ISBN 978-91-7595-267-3 SWEDEN

Akademisk avhandling som med tillstånd av Kungl Tekniska högskolan framläggges till offentlig granskning för avläggande av teknologie doktorsexamen i datalogi fredagen den 17 oktober 2014 klockan 14.00 i Sal F3, Kungl Tekniska högskolan, Valhallavägen 79, Stockholm.

Tryck: E-print AB

Abstract

This thesis primarily reports on an action research project that has been conducted on a course in theoretical computer science (TCS). The course is called Algorithms, data structures, and complexity (ADC) and is given at KTH Royal Institute of Technology in Stockholm, Sweden.

The ADC course is an introduction to TCS, but resembles and succeeds courses introducing programming, system development best practices, problem solving, proving, and logic. Requiring the completion of four programming projects, the course can easily be perceived as a programming course by the students. Most previous research in computer science education has been on programming and introductory courses.

The focus of the thesis work has been to understand what subject matter is particularly difficult to students. In three action research cycles, the course has been studied and improved to alleviate the discovered difficulties. We also discuss how the course design may color students' perceptions of what TCS *is*. Most of the results are descriptive.

Additionally, automated assessment has been introduced in the ADC course as well as in introductory courses for non-CS majors. Automated assessment is appreciated by the students and is directing their attention to the importance of program correctness. A drawback is that the exercises in their current form are not likely to encourage students to take responsibility for program correctness.

The most difficult tasks of the course are related to proving correctness, solving complex dynamic programming problems, and to reductions. A certain confusion regarding the epistemology, tools and discourse of the ADC course and of TCS in general can be glimpsed in the way difficulties manifest themselves. Possible consequences of viewing the highly mathematical problems and tools of ADC in more practical, programming, perspective, are discussed. It is likely that teachers could explicitly address more of the nature and discourse of TCS in order to reduce confusion among the students, for instance regarding the use of such words and constructs as "problem", "verify a solution", and "proof sketch".

One of the tools used to study difficulties was self-efficacy surveys. No correlation was found between the self-efficacy beliefs and the graded performance on the course. Further investigation of this is beyond the scope of this thesis, but may be done with tasks corresponding more closely and exclusively to each self-efficacy item.

Didactics is an additional way for a professional to understand his or her subject. Didactics is concerned with the *teaching and learning of something*, and hence sheds light on that "something" from an angle that sometimes is not reflected on by its professionals. Reflecting on didactical aspects of TCS can enrichen the understanding of the subject itself, which is one goal with this work.

Sammanfattning

I den här avhandlingen diskuteras ett aktionsforskningsprojekt som har utförts på kursen ADK (Algoritmer, datastrukturer och komplexitet) vid KTH i Stockholm. Kursen är en introduktion till teoretisk datalogi (TCS) och ges för civilingenjörsstudenterna i datateknik.

ADK-kursen är en fortsättningskurs till tidigare kurser i programmering, systemutveckling och logik. Då den bland annat omfattar fyra stora programmeringsuppgifter, kan studenterna lätt uppfatta den som ytterligare en programmeringskurs. Tidigare forskning om undervisning och lärande i datateknik/datavetenskap handlar vanligtvis om introduktionskurser i programmering.

Fokus för den här avhandlingen har varit att förstå vilka begrepp och idéer som studenterna uppfattar som extra svåra. Under tre aktionsforskningscykler har kursen studerats och förändrats för att förbättra den och avhjälpa svårigheterna. En diskussion förs också kring hur kursens diskurs och dess aktiviteter kan påverka studenternas uppfattning om vad TCS är. Resultaten är till övervägande del beskrivande.

Utöver detta har automaträttning införts, både i ADK-kursen och på andra kurser för studenter som inte läser datateknik. Automaträttning är uppskattat bland studenterna, och får dem att inse att korrekthet är en viktig aspekt av ett program eller en algoritm. En nackdel är att det inte är troligt att uppgifterna i sin nuvarande utformning signalerar att det är studenternas ansvar att säkerställa att deras program gör rätt.

De svåraste momenten i kursen är kopplade till korrekthetsbevis, till att konstruera komplicerade dynamisk programmeringsalgoritmer och till reduktioner. Man kan också i det sätt svårigheterna manifesteras ana en viss förvirring hos studenterna gällande kursens, och den teoretiska datalogins, epistemologi, medel, mål och språk. En diskussion förs kring vilka konsekvenserna blir om man betraktar det matematiska innehållet i ADK-kursen ur ett rent praktiskt programmeringsperspektiv. Lärare skulle kunna göra mycket mera för att avhjälpa den förvirringen, till exempel genom att göra fler metautläggningar om språkanvändningen och hur den skiljer sig ifrån vardagsspråket gällande ord som “problem”, “verifiera en lösning” och “bevisskiss”.

Ett av verktygen som använts för att analysera svårigheter har varit *self-efficacy*-enkäter, ungefär “självvärderingsenkäter”. Ingen korrelation kunde uppmätas mellan studenternas svar på dessa och deras betygsatta kursprestationer. För att avgöra om deras självvärderingar korrelerar med prestationer behöver andra uppgifter än kursuppgifterna användas, som har samma avgränsningar och innehåll som var och en av frågorna på enkäterna. Detta ingår inte i denna avhandling.

Didaktik utgör ett ytterligare perspektiv för forskare och lärare att förstå sitt ämne på. Eftersom didaktiken handlar om hur lärande och undervisning bör gå till, kan reflektioner kring detta göra att fler av grundantagandena inom TCS lyfts fram och används för att beskriva fältet. Denna rikare förståelse för TCS är ett av målen med avhandlingsarbetet.

Contents

Acknowledgments

I want to thank my advisors Olle Bälter and Viggo Kann for their support and encouragement during this project in the borderlands between disciplines. It is a situation sometimes most uncomfortable, to neither be wholly a computer scientist, nor identifying with the educational specialization. Theoretical computer science is a subject I immediately took to when, almost by chance, taking the ADC course myself as an undergraduate from another program at KTH. The discussions about didactics in the corridors among the staff of the TCS department have been interesting and useful, and I am happy that so many people are so concerned with providing good education for the undergraduates, and how to improve it. I strongly believe that this type of work could not be performed at a distance from where the higher education on theoretical computer science is actually conducted, as subject area knowledge from TCS itself as well as from didactics are both necessary ingredients.

I also want to thank my co-authors, and all of the students and TAs of the ADC course who have accepted to contribute to this research project with data and observations, and the teachers who responded to my web survey. For the statistical data, I especially thank Monika Lundell, who could dig up old course results and grades for comparison, on different courses which have replaced each other through the years, and Anna Björklund who provided data about admission grades for more than ten years of ADC students.

A special thank you to all PhD students and postdocs from all around the world who have been working at the TCS department of KTH during my time here, and who have provided a creative and sympathetic working and free time environment. The topics of our lunch table discussions must have been found confusing by bypassers only after a few minutes, as the associations quickly take twist and turns, and the lunches have been very enjoyable.

Also thanks to the projects and institutions funding my research studies: The Resource Center for Net-based Education, GRU, STINT and the EU project Edujudge.

Chapter 1

Introduction

This thesis is about computer science education (CSE) and didactics. It has the, for this genre, unusual target of theoretical computer science, taught to computer science majors and effectively distinguishing them from programmers. Theoretical computer science (TCS) deals with the foundations of computation, which can impose requirements on IT solutions on a more fundamental level than current state of the art of frameworks and implementations. It is a mathematical knowledge area, which may come as a surprise to some more practically oriented students, maybe especially as it comes in disguise as "something related to programming". This positions the thesis somewhere in the borderlands of mathematics didactics, and more "mainstream" computer science didactics.

When it comes to teaching, I believe that theoretical computer science is not sufficiently understood. I strive to grasp what the topics themselves contain in terms of potential difficulties – language conflicting with everyday language, perspectives that are taken for granted but need explaining, structures which can be compared and combined but are instead presented, and learned, independently.

The work presented here is, if you like, a case study. It is the result of several consecutive years of study of a course on algorithms, data structure and computational complexity, ADC, for Master of Science in Engineering program students at their undergraduate level. The first attempts at improving this course which I was involved in, happened in 2007. Most of the work, however, is from 2011-2013. It is a spiraling, ongoing course improvement project with an action research structure but involving attempts to evaluate some aspects of the course quantitatively. Some of the dependencies during these years are illustrated in Figure 1.1.

I have experienced most educational research in computer science based on constructivism. In my earlier teaching studies, the social perspective on teaching and learning was, mildly speaking, most enunciated. Here I am going to make use of tools from both constructivism and socio-cultural learning theory. When starting from myself as a teacher, looking at the subject of theoretical computer science, I will adopt a constructivist perspective and look at what it is I am presenting,

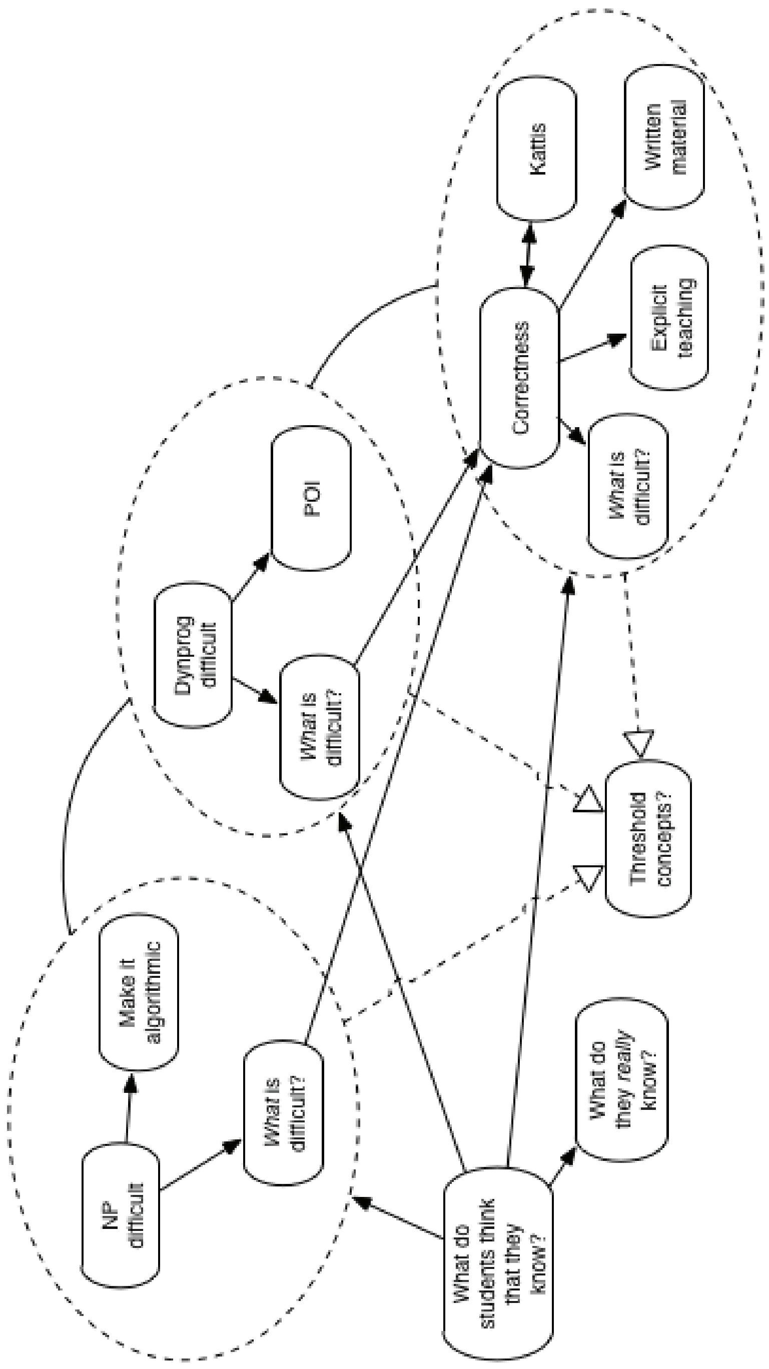

Figure 1.1: Overview of the thesis work.

what characterizes the concepts and topics I teach, what mental models I believe all computer scientists share a skeleton of, and think of such things as cognitive load and schemata in order to present a clear view of the topics. When interpreting what my students are struggling with, and why, while keeping the constructivist's tools, I believe that it is crucial to reflect on which questions belong in my course/discipline, what types of answers reside there, how do we connote the language to adapt it to a collective perspective, and – how much are the students aware of this perspective? Will they share it with me?

My main interest is: **What can be particularly difficult in TCS, and particularly, in the ADC course?** What does our teaching experiences tell us about this question? In particular, I have investigated:

1. Which conceptual difficulties are inherent in
   a) NP completeness proofs?
   b) theoretical computer science in the course ADC?
2. What tasks do students find more difficult than others, and which tasks are not considered problematic?

This is partly analyzed quantitatively, using survey responses and grades, but also discussed qualitatively based on gathered experiences of mine, and of other teachers and TAs.

A second focus is: **What do our teaching approaches and activities (among other things, automated assessment of programming exercises) convey about the nature of TCS, to the students?** This thesis treats the following aspects:

1. How do students respond to changes in course activities?
2. What does one need to master to succeed in the ADC course?
3. Are we, as teachers, explicitly unveiling the perspectives we are part of?

The first of these latter questions is evaluated quantitatively, and the two other questions are philosophical topics rather than research questions, and are discussed qualitatively in relation to the nature of the difficulties found and described.

Both of these foci are founded in the more general questions "What is theoretical computer science?" and "Which perspectives are assumed and utilized in theoretical computer science?". These are didactical questions if placed in the context of teaching, and they represent the view on didactics that I have adopted here. I am interested in the subject, in itself, and what characterizes it, compared with other experiences students are likely to have.

The results presented in this thesis will be based on previous publications, and on additional data described in the thesis. Most of the research aims to describe, as causality is troublesome to establish in a study of teaching and learning.

1.1 Preface

When I started working on my Master's thesis for Viggo Kann in 2007, it dawned on me that I had learned things from my teaching education. The differences between the strictly regulated, and nation wide streamlined, school system of "Gymnasiet" (which we loosely translate to "High School", but is a non-compulsory education for pupils aged 16-19 in Sweden) on one hand, and the Swedish university education regulations, were striking and, because of a successful indoctrination in earlier studies, puzzling to me. As a teacher in gymnasiet, my work would be to teach a couple of subjects – hopefully those I had studied – and assess students' work by means of national, regional and local specifications. From a historical perspective in Sweden, the specifications I had to adhere to were highly unspecific, but they existed and I had to relate to them. A natural, advisable and useful principle when tackling questions about didactics hence was to "go back" to these regulations and specifications, and make sure that what I wanted to do was in line with them.

I was not at the time aware that I had adopted this habit, but it became clear when Viggo explained to me that what little was actually regulated by higher authorities was not in any way relevant to choosing topics for a course, or on what level students would need to understand or deal with the chosen topics. Not even at the central level at the university was there anything that posed requirements and restrictions for most courses. It was entirely the individual teacher, or the teachers together, who decided about these matters. There were no documents, no regulations, no guiding principles that I needed to conform with when designing a new task for a course! Instead, the teacher defined the scope, depth and structure of the course.

This might seem bad. It still works, because teachers are professionals in the topics they are teaching – nothing else would be acceptable at this level. They were educated in the subjects and have both the concepts, methods, models and history of their subject to relate to. When they teach, they are acting less as officials of the society, performing tasks by stipulation from a higher authority, and more as representatives the authority of their field, than high school teacher have the freedom to. Teachers also cooperate. They discuss concepts, discuss practices, and they read information online from other universities world wide – information or experiences that they may incorporate in their teaching, or at least relate to. The most underestimated factor here is "canon". In the professions, practitioners to a large extent agree on what the core concepts and principles of their respective fields are. Courses have similar goals. Text books are sold world wide, but it is common that teachers provide supplementary information from several books to be able to present the topics they want to include, in the way they prefer.

The organization ACM (Association for Computing Machinery) also produces very detailed curricula for various engineering disciplines involving computing – curricula that compared to those for gymnasiet, at least in 2007, were extremely detailed. Swedish universities, at least KTH, try to make the computer science engineering program ACM compliant by including everything from the curriculum in their courses, but not by letting the ACM curricula define the courses. "We are ACM compliant" does not mean that the students took a course named CS1 in their first year, that was entirely governed by the ACM curriculum for the discipline, but that students were taught all topics from all courses specified by the ACM curriculum. Besides bragging about ACM compliance, universities claim that they are the best, that their students get the best job opportunities the soonest after (or, in some cases, before) graduation, and that the researchers at the university are good (partially implying that they are qualified to decide most accurately on what students need to learn). Their freedom to design courses based on what their staff believes is most relevant and useful, is related to this competition. This can be seen in comparison with "earlier" schools than universities. They are also competing, but their students are younger and still live at home. The inherent ambition is that all schools should be in some sense equally good – they are to assign grades, and the same grade for the same course should mean the same nation wide, so that it can later be used for selection. In practice, they compete for students with statements about average grades among their students. The room for them to compete with special, locally designed courses, was severely diminished lately. For universities, curriculum design is possible to use in branding and marketing.

At that time, I was surprised that such a prestigious and important part of the educational system had so few binding rules. In hindsight, I can see that I was successfully socialized into a high school teacher's professional practices. Now I am also socialized into some of the university researcher's and teacher's professional practices. During the time this second socialization process happened regulations of higher education in Sweden have also changed. The way of specifying intended learning outcomes with each course at KTH is slowly getting standardized. More discussions of how to guarantee that all goals with the education are addressed *somewhere* in the educational program are taking place, and explicit criteria for assessment are gaining land. The dominant paradigm for assessing in Sweden, in the mandatory school system and in gymnasiet, is goal-based assessment. This means that grades should be understood as absolute, and not relative, "measurements".

Canon and collegial decision making are powerful tools (and the very fundaments of organized science), and they should not be underestimated in favor of more mechanically accessible quality assurance metrics. They are not incompatible with goal-based assessment.

Changing perspectives

Here I will describe some instances of changing perspectives caused by teaching. I remember two very troublesome experiences during my own school time in particular, when I think about the sociocultural aspect of teaching and learning. The descriptions below are extremely subjective, and I cannot verify them in any way.

I was always a pupil very sensitive about what was expected of me. The mere condition that a teacher had given me a task, I interpreted as "I, your teacher, find this important" and "I, your teacher, believe that this is a suitable task for you", meaning that I should learn it because the teacher wanted me to, and could learn it because the teacher expected me to. It wouldn't make sense for a teacher to assign a task that he or she did not believe that the pupils were qualified for. It would also not make sense to require meaningless tasks. This type of student is not very useful for opening the teachers eyes to new methods and interpretations of the teaching situation, but the approach tended to be rewarding for the pupil in terms of grades. There was also some moral aspect to this – I believe that I considered it rather immoral to *not* do something that I was expected to do. The teacher expected me to perform something, and I did not want to let the teacher down.

In this context, in a small school where most teachers knew most pupils by name, there was due to a lack of trained physics teachers, a line of five different people teaching physics during my three years of studying it in this school, age 13-15. I believe it was when I was 14, the teacher of one of the semesters of that year was actually trained in physics – he was an engineering student – but not in teaching. (We also tried the opposite type of arrangement, so it is actually impressive that I learned any physics at all. . .).

The teacher assigned the class a home exam, to be done in any amount of time, and handed in by some deadline. I found this task extremely challenging! Today, I would pay quite a lot to get to see this exam, and my answers to it. All I do remember, apart from this being a very unusual experience during the first, mandatory, 9 years of school in my life, was that it was so hard. I really had to work a lot, for many hours. I remember expressing it as "first you have to find out what problem you have to solve, which is really hard, and then you have to solve it, too". I suspect that this was about forces and motion, because I remember also my struggle to come to terms with the concept of "velocity" as something completely different from what I was used to, and I think this occurred during the same semester. No one had yet described the difference between scalars and vectors to me, and also the huge generalizability of that distinction – dimensional analysis – was left for me to figure out on my own. I had really never met with the idea that words, everyday words that I thought I knew, suddenly had a very precise meaning and that that meaning *differed* from the everyday meaning. Maybe my struggle with this task was related to that?

When receiving the graded exams back from the teacher (and probably already before that), I was disappointed with my work. I was supposed to solve all of the problems, weren't I? I had 13.5 points out of 20, which was my worst result yet experienced. As it turned out, the average score on this exam was around 3. Clearly this teacher had underestimated the difficulties novices meet with in physics, clearly he had overestimated the prerequisites for 14 year old pupils to persist in solving the type of tasks he had assigned, and clearly he had no idea of what level we were on. Probably, if I had not blindly trusted teachers to know what I was able to do and what was best for me, I would never have spent so much time on this exam, and would have performed even worse. I also would have learned much less, since I really believe that this experience was beneficial for my learning. I had to work out so many new connections, realize so many things at the same time, and apply logic and probably dimensional analysis through logic reasoning, to make sense of the tasks. But imagine being a pupil with lower academic self esteem, expecting tasks to be too difficult, trying to solve some problems, giving up and and receiving a score of 1–5 out of 20! In that framing, the task was probably not beneficial at all! It probably undermined the trust to teachers, the self esteem and the ambitions in school for such a student. It is also a provoking fact that my consolation here was that I at least performed much better than the rest of the class. This parasitical way of consolidating academic self esteem is bothering to me, but for some people, it is an undeniable part of their experience.

If I could acquire my old exam, and read it with my grown up experience as a background, maybe there was nothing special with it at all. Maybe I cannot, any more, discern that dimension of utter and complete chaos among all concepts, and the hard work that was needed to make sense of it. Maybe at that very day, I acquired part of the physicist's perspective on his discipline: which questions should be posed, which methods should be used, what type of answers are valid?

The other really challenging perspective acquisition happened to me during my first year of physics courses at the university. Students at KTH are introduced to their new environment by a course in thermodynamics. This course is not an ordinary one, but some of the teaching and learning experts (those who are most focused on methods and the distinction between "old" teaching methods and "new") might fail to see this. It was a course with lectures in a great lecture hall, some worked example sessions, one or two smaller tests, and a written exam at the end. It was, however, not really teaching thermodynamics at all. It was teaching reasoning, reality check, dimension analysis; to verify answers arrived at by some calculation by some other, independent method, and to approximate whatever you did not know. That is probably an excellent way of making students realize that there has been a shift in perspective from earlier classes where physics maybe was a book that you read chapter by chapter, and where you could look up the correct answers to all examples at the end.

The teacher wanted us to become independent thinkers, and he wanted us to view physics in relation to reality, not just in relation to courses and course books. It is his way of treating the term "black" that I remember the most as a struggle with perspectives. It clearly is an everyday word, but then, in thermodynamics, it is all of a sudden not related to visible colors anymore, but to generalized characteristics of the visible "black" – and he made a point of saying it often and mixing the meanings. I was occupied trying to form the concept "thermodynamically black" for myself, and it was frustrating that he made those puns all of the time, because I had to put quite a lot of jigsaw pieces together to appreciate them. At some level, I realized that he was not lecturing about the everyday concept of "black", but it was hard to understand why he said what he did – what it was that he wanted by saying it and which characteristics of ordinary black that were transferable to this concept. Thermodynamically black is a quality some object can possess to varying degree, and "real" blackness is a somewhat idealized trait: it is not necessarily possible for an object to be completely black, in all frequencies. He forced acceptance of thermodynamically black by osmosis, by using the term in a way that a physicist would. So he did change my perspective, but I believe that I would have benefited from being taught about this, on a meta-level. My perception is that I was not. (I am probably not a very talented physics student, which is why this caused a struggle in the first place. There were domain specific "codes", that I was not seeing or understanding, but someone who had grasped more about what physics is really about probably had less trouble than I did.)

I believe strongly that more experiences of this type reside in any teaching and learning context, and that my own students would benefit from me anticipating and explicitly tackling such perspective changes. This has lead me to also reflect on indications of needs for such perspective changes during my PhD years. It is not something I have formally investigated, but still worth discussing when talking of canon and socialization processes, and worth mentioning here because we might have taken the language and the perspective for granted. Sometimes misunderstandings are totally unexpected to the teachers, like when some ADC students were angry that their proof sketches (in the shape of pictures of the theorem, at best) did not give them any points on the exam. A proof sketch should indicate *why* the theorem is true, and constitute a "skeleton" of the proof, but need not be very detailed. We had been sketching proofs the entire semester by then, and did not know that some students had failed to pick up this term.

There are more words and expressions that can mislead, if their domain specific use is not acknowledged. Another mathematics teacher mentioned the word "speciellt", which has the basic meaning "especially", but in proofs often means "in particular" with flavor of "for example". When the teacher says that the theorem is valid for real numbers, especially for 5, this will not make sense. If you understand that the teacher means that 5 is a specific example of where the theorem is valid, you don't need to waste energy on understanding why 5 is special. Yet another teacher mentioned how the class were amused as the teacher went on about "uppskatta", which in everyday language roughly means "appreciate" but also "estimate".

These are tiny details in language that cause inclusion and exclusion in a group, and while the ultimate goal is for all students to be able to use the language and be included in the group, it is the responsibility of the teacher, who knows but is perhaps not aware of both the everyday and the discipline specific meaning of words, to contrast these meanings and help students to realize that the words are terms in the particular discipline.

1.2 Contributions in shared papers

I **Computer Lab Work on Theory** This was my very first article, based on my master thesis, which means that I made 90 % of the design, planning and work but 75 % of the writing.

II **The effect of short formative web quizzes with minimal feedback** In this project I was involved in planning, design, execution and evaluation of the experiments together with my advisor Olle Bälter, and performed about 60 % of the writing of the paper.

III **Five years with Kattis – Using an Automated Assessment System in Teaching** The work described in this paper was largely conducted before I started my PhD, but would not have been published without my efforts. I contributed to some extent to the pedagogical development of the tool Kattis, by designing the new type of task that was described in my first paper. I did 70 % of the writing of this paper, and provided most of the literature references.

IV **From Theory to Practice – NP-completeness for Every CS Student** In this paper, I and the other authors collaboratively designed the research and reviewed our teaching materials. The other authors were lecturers on the courses described, and the choice of relevant "clicker questions" was mostly not my work. I did 75 % of the writing of this paper.

V **Dynamic programming – structure, difficulties and teaching** This paper is completely my own work, supported by my advisor Viggo Kann who taught the courses described.

VI **Iteratively intervening with the "most difficult" topics of an algorithms and complexity course** This paper was written as a wrap-up of the results in most previous papers, but all new results are mine. The writing of the paper was done in collaboration with the other author.

1.3 Ethics

Although primarily interested in characteristics of *the subject* of theoretical computer science, I am dealing with achievements and attitudes of my students. This requires some ethical considerations. Common developments of students' presentations of their individual homework are referenced to as part of my own teacher experience. For the data, nothing identifying about the participating students will be released, and the results are presented in aggregated form. For the many surveys, where we asked students to write their name on the page, we promised that this material was not to be dealt with during the course, and that it in no way would count towards grades. Most students accepted this and, helpfully, completed the surveys. Six students (four in 2012 and two in 2013) participated under pseudonyms. For questions that only deal with changes in self-efficacy, these surveys can still be included since these students used the same pseudonym throughout. However, for evaluating surveys by comparing self-efficacy responses with grading and assessment, of course they cannot be included. In addition to the pseudonymous answers, some students did not put any name on the paper at all, out of which some still responded to the survey. These can naturally neither be used for viewing changes between occasions, or comparisons with course achievements.

There are other ethical considerations involved in this type of work, closely related to research methodology. For some researchers, a controlled study would seem most appealing. This approach was decided against on the basis of equal treatment of students. Since we believe that our methods could alter the preconditions for learning, it would be unfair to not give the same opportunities to all students. A remaining option, provided a fully quantitative study was called for, would have been to invent *several*, different, teaching and learning activities and compare them. We would then have a record of how two new teaching methods performed, compared with one another, but still no information on how these compared with the old methods. To teach in different ways than before, because we believe that something was badly taught before, is not unethical. To omit something we believe is useful just to see whether we were right, is not fair to the students. We have used an action research perspective, and changed things during the way. This is both because we wanted to work rather applied, and because the students should be the primary concern when designing courses, not research on students.

Part I

Background

Chapter 2

Theoretical framework

The theoretical background to this work is threefold – there is the theoretical computer science background (here treated mostly as scenography), there is the learning theory background at which every educational program begins, and there are more specific types of results that are relevant to this thesis. Also some qualitative research perspectives are briefly explained. This chapter also describes the "practical" background of the work – the course and the educational setting where it took place.

2.1 Research perspectives

There is a large number of qualitative research methodologies out there. This work is largely performed within the action research perspective, and this choice was made because of the work's dual purpose: both as course development and research project. Phenomenography is not used directly, but some of the concepts are.

Phenomenography

A phenomenographic take on learning theories stems from [13] and [36]. Phenomenography is a descriptive, qualitative research method that investigates how people construct reality. Typically, a result could be "three qualitatively different ways to experience the equality symbol". A sidetrack of the research method is the *variation theory* described later in this chapter, which consists of identifying a number of necessary conditions for something to be experienced and distinguished from other experiences – a precondition for learning.

Action research

Action research is a research methodology that consists of iterative, cyclically sequenced phases. It is applied when there are changes to be made, and the distance between research and practice is minimized. It is common for teachers to do action research on their own practice, but also common that an external researcher gets involved. There are several ways of describing the action research processes, and several versions of action research methodology. They usually involve the same ingredients: Plan, act, observe and reflect, then revise your questions and methods and repeat. Some authors start the cycle with observations (O'Leary), others with planning (Kemmis & McTaggart). For instance Koshy et. al. [34] points out that the methodology usually does not contain separate phases in sequence – they often overlap with each other, and too strict methodological focus can inhibit success in the work.

2.2 Learning theories

For readers with a background in didactics, this section is only common background about learning theories, but for readers specialized in TCS, it might be helpful to have the historical context described a bit more. When dealing with matters of teaching and learning, it is necessary to relate to learning theories. These all offer some answers to the question "What is learning?", and are usually based on psychology, sociology and pedagogy. They may have various interpretations of ontology and epistemology, and are often grouped in three major categories: behaviorism, cognitive constructivism and social constructivism. The descriptions below are not describing recent findings, but instead the basic features of the learning theories.

Behaviorism

The behavioristic perspective on learning was created around the beginning of the 20th century. It was introduced by John B Watson as an attempt to make psychology into an accepted part of the natural sciences, honoring objectivity, experiments, reproducibility as do the natural sciences. The perspective dominated the early 1900s. Some names associated with this discipline are Pavlov and Skinner. Ivan Pavlov was conducting research in physiology and neurology. While investigating digestion he obsered some behaviors called "involuntary reflex actions", and further "conditioned reflexes". Maybe the most famous example is dogs salivating when something they had associated with food was present (conditioned reflex) as they would have when food was present (unconditioned, involuntary, reflex). The text book example of this is that when hearing a bell ring, that used to ring before food was administered, dogs started salivating. These experiments are described for instance in [42, lecture II]. The stimulus used in this description was a metronome, not a bell. Pavlov's results inspired the idea to think of learning as conditioned behavior. Psychologists like Skinner later experimented on animals, and sometimes on people, focusing on what could be objectively observed. Instead of speculating and utilizing inexact methods focusing on what was inside the heads of the observed people, behaviorism looks at learned behavior described in terms of *stimuli*, on positive or negative reinforcement and punishment, and *response*. Learned, conditioned, behavior soon gave rise to learning theories. Skinner was himself active in discussing teaching and learning [49]. Learning according to the behaviorist is acquiring new associations between various stimuli from the outside world, and actions, behaviour, on the part of the learner: adjusting behavior according to signals from the environment.

All subsequent learning theories oppose the behavioristic perspective in some way or another. The criticism is mainly that the generalizations from dogs learning where to eat and to not eat to humans learning in schools is not valid, and that humans are active while learning, not passively responding to stimuli from the environment (including other people). The behaviorists are further criticized for treating children as empty books, ready to be filled with content from outside, and teaching as transmitting knowledge from a sender to a recipient.

This was not the beginning of history – in an historical account of the cognitive revolution, Miller [39], describes behaviorism as a revolution, too. It was the reaction to non-scientific theories and methods dominating psychology at the time, with focus on "mind" and "consciousness", and the research methods of clinical case studies as in the case of Freud, or introspection.

It is obvious that behaviorism teaches us something about human behavior, but it is less obvious that "learning", with everything we put into that concept, is best summarized and defined in terms of behavior. Skinner himself refers to Socrates' way of "teaching" a slave boy mathematical proof by only asking questions as "It is one of the great frauds in the history of education." [49]. While true that Socrates did never lecture on the topic, the boy's responses were carefully anticipated and the questions chosen so that there were no other possible responses available for the boy. Skinner also mentions that Socrates was aware that the boy had learned nothing, and could not possibly repeat the proof later on his own. Later in the same paper, Skinner describes successful "programming" of behavior in a pigeon, where an extremely complex response that would not normally occur by itself can be learned by a step by step procedure of reinforcement. This type of knowledge is useful in marketing research where the aim is to manipulate customers into certain behaviors, in game design and to some extent in advice about child upbringing. Maybe it is most useful when it is actually behavior that we want to teach? While true that assessment is, unavoidably, based on what a pupil *shows* to have learned, teaching complex skills by means of reinforcing desired behavior and ignoring undesired behavior is too blunt a tool for efficient learning. It can result in the same situation as with Socrates and the slave boy: our pupils behave in the right way, may not have any idea of why they behave in that particular way. In the case of much school knowledge, did you really learn it if you are not aware of it?

Another angle of criticism against the radical behaviorism of Skinner, is that the very foundations of behaviorism might be philosophically challenging to many of us: that there is no such thing as a choice, that there is no free will, that everything happens as a consequence of earlier experiences and anticipated consequences of actions, in a deterministic (but only probabilistically so) way. Omitting free will, the concept of choice, does to me seem like omitting the concept of intentions in individuals.

Cognitive constructivism

One of the revolting schools of psychology started to direct interest to the *inside limitations* of the learner. The cognitive psychologists were interested in how memory worked and what type of individual cognitive activity a learner undertook. Jean Piaget is the most famous pedagogy founder from this school [43]. He studied children and their cognitive capacity developing with age. The system he came up with had very specific claims on various cognitive maturity levels and corresponding ages. Learning in general, according to Piaget and most cognitive constructivists, consists of creating schemata – mental connections between concepts in order to structure chaos and not needing to focus on unnecessary details. A schema allows related information to be considered simultaneously, as a chunk, which is simpler for people to deal with. Piaget utilized the concepts *assimilation*: when a learner binds a new concept, fact or experience to the already present representations of the world, and *accommodation*: the activity when a learner, due to a cognitive conflict, finds himself unable to assimilate some concept in present mental structures – it does not make sense – and is required to "refurnish" his mental representations of the world in new structures, new schemata, allowing this concept to fit in. This occurrence is sometimes compared to paradigm shifts in research: the old ways of interpreting mechanics could not explain and conflicted with observations: enter Newtonian mechanics. Learning according to the cognitive constructivism consists of building cognitive structures and representations of the world. It is the active and deliberate work with assimilating new concepts and accomodating the internalized view of the world when necessary.

Other types of results close to cognitive constructivism is for instance Miller's famous result from 1956, about "the magic number seven" [38], popularly known to claim that people seem to be able to keep no more than seven different things in working memory at the same time. More specifically, Miller tries to determine the amount of information a person can handle, and for "absolute judgements of one-dimensional stimuli" this seems to amount to 2.5 bits of information which would correspond to discriminating between 6 various categories that the stimuli might belong in. One of the examples of a one-dimensional stimuli is the pitch of a note. Miller also cites other results showing that if more dimensions of stimuli are added, for instance both pitch and volume of a tone, more categories can be discerned, but the increase is not simply the union of the independent results for pitch and volume.

Another result everyone "knows" that originates from cognitive psychology studies, is that the human *attention span* is limited to about ten minutes, and that lectures going on for longer loses the students' attention. The results behind this belief have been questioned lately by Wilson and Korn [56], who find that what has been found to "decline" is note taking, but not after 10-15 minutes; heart rate, but there were only four students in the experiment and they seemed to remember the last part of a 40-minutes lecture well; and the self-reported concentration levels every five minutes, but that concentration level varied considerably between different teachers. Also observations have shown that people start to lose attention during lectures, but some of the most cited studies are only based on personal hypotheses. Wilson and Korn are requesting controlled, experimental studies on attention span in order to be able to support the 10 minutes rule. Self-reporting ways of establishing the limit of the attention span by clickers have been tried by Bunce et al. [14]. While the ten minutes rule is not supported by research, it is true that pedagogy requiring involvement decreases the frequency and elongation of attention decline.

Cognitive constructivism is emphasizing the activity of the learner, and that learning builds on previous experience. No two people have the exact same knowledge, because knowledge is constructed actively by people with inevitably different previous experiences. Teaching need to be designed to take into account the students' experience, and start from there. When furthered to the extreme, teaching methods devised based on the ideologically "clean" (cognitive) constructivism erase the teacher's activity and the existence of active teaching, to the benefit of "learning by discovery". This is not a necessary consequence of cognitive constructivism, but a rich source of criticism of this learning theory. When taken to the extreme, students are expected to discover everything "by themselves", and the mere existence of some types of activities, like problem solving or "research" – when pupils are required to search for information about something and present it – are viewed as per definition good and enhancing learning.

A concept often mentioned in constructivist theories is *transfer*. It is known that students might only learn how to use certain concepts in very limited situations (resembling a lecture or test), and one important aim for educators is to enhance transfer of knowledge between fields and situations.

Social constructivism

Social constructivism is a learning theory emphasizing social and situated aspects of learning: it is impossible to separate the learning from the context, and learning does not take place within an individual, but in the interaction between individuals. Cognitive apprenticeship and socialization processes are part of the terminology. The part of the foremost thinker founding this school of thought is generally assigned to the Russian psychologist Lev Vygotsky. He emphasized that language and learning were closely interrelated, and that everything first happened on the language level, in interaction, and later could be mimicked within an individual. The maybe most famous concept of Vygotsky's theory is the ZPD: zone of proximal development. Every individual has some level to which he or she can perform tasks independently, but outside of that safe zone are some close tasks, things that the individual is not yet capable of performing on his or her own, but can manage in interaction with someone more skilled at that particular thing than themselves, which is the ZPD. In Sweden, Roger Säljö has become very influential with his texts about *socio-cultural* learning [47]. People learn by gradually becoming part of some social practice, and some community. Learning is not just about attaching new knowledge to your mind – it also changes *you* and the knowledge itself.

Another common term in social constructivism and sociocultural perspectives on learning, is *mediated action*. The concept goes back to Vygotsky – one of the media we have available for us is language. In the introduction to his book "Voices of the mind", Wertsch [55] acknowledges that there are "universal as well as socioculturally specific aspects of human mental functioning", and that there have been too many misunderstandings because the choice between these aspects and their respective research agendas have not been communicated explicitly. When choosing the sociocultural research agenda, focus is on the social aspects of both situations and the available media, for instance language. This means that learning is social even when it is happening due to mental action of one single individual.

Since emphasis is on learning as a social activity, teaching activities based on social constructivism may favor group assignments.

Situated learning

Instead of discussing these three directions, [24] view learning theories as prebehavioristic, behavioristic, processual, and contemporary. The prebehavioristic learning theory is the philosophy of learning before behaviorism, when experiments and measurements were not common, with for instance the view of a child as a "tabula rasa" that sometimes is attributed to behaviorism. Similarly, processual learning theories are seen as responsible for ideas that cognitive constructivism is sometimes blamed for, like viewing the mind as a machine. The purpose of the paper is bridging a gap between why situated learning occurs and the product view on learning, by describing *how* situated learning occurs. The authors define situated learning as "a change in mental models that happens through social interaction in a given context" [24, p. 218]. In a text critical to educational research in general, Langer [35] uses situated learning as an example on how educational psychology ought to be more involved in education research. Langer argues that there is a lack of agreement as to how to define situated learning. While agreeing that both physical and social settings can be alluded to, he observes definitions from "located in a particular community of practice" to "taking place in real life", where the latter makes situated learning irrelevant to school learning. He also finds discourse transforming the situatedness to a dogma rather than something that can be investigated, and that concepts from educational psychology are ignored by educators. One of his examples of such an ignored concept is transfer, which is not contradictory to some of the definitions of situated learning, while other definitions make it a non-issue. Both Goel and Langer appear to wish for more research on *how* situated learning occurs; its processes and characteristics, instead of merely arguing the situatedness of all learning.

Personal perspective on learning theories

When studying at an undergraduate level on a combined teaching and engineering program, I perceived the education about learning theories to encompass three different views on learning and a very prominent tendency of favoring the last one described, since it was more “complete” and hence “of a higher quality”. The idea that there is neither right nor wrong, just various understandings which can be organized in sets of higher quality understandings, where each larger set contained the previous ones and was larger and more complete is somehow expected to be more respectful to the learner’s experience and views. This learner disagreed! What you choose to describe something as depends on what your goal is, what type of lens you believe could help you see what you want to examine; what you are currently interested in. We were taught that silly people like Skinner and Pavlov believed that teaching and learning was all about stimuli and behavior, that carrots and whips were the teachers tool’s, and that you knew that someone had learned something once they behaved in an appropriate way. We were then taught about Piaget and constructivism, which imposed the constraint that you should use problem-based learning and never lecture, that learners were curious and should invent all knowledge by themselves – this was an improvement compared to behaviorism – and finally about the social practice perspective on learning, about Vygotsky but foremost about Säljö and about all learning as interaction in a context, in a situation, and hence all learning situated. The emphasis of this last perspective is on interaction between individuals, on learning as a social process rather than an individual, cognitive process, and of the knowledge itself as changing and residing within people rather than as a separate entity.

While agreeing that these are three different perspectives, and while agreeing that you probably can not build a very good school (in the sense of preparing children for being responsible, mature decision-makers in the future) on behaviorism, I internally and sometimes intellectually rebelled against these descriptions. My criticism was that the teaching was similar to religious teaching: these are the wrong beliefs, these are the right ones – now describe your “Eureka moment” you just had when realizing this to the rest of the group, please! Others have criticized how pedagogy often is irresistant to various fads, for example Anderson et al. [1] who analyze the relationship between cognitive psychology and constructivism, and points out some misunderstandings. The religious similarities seem not to have been only my perception of the teaching in Stockholm – a couple of teacher students in Gothenburg who studied at approximately the same time were so annoyed by the evangelization that they wrote a very indignant final thesis on it [2].

It is however true that the three perspectives behaviorism, constructivism and social practice perspective on learning have dominated the 20th century’s education paradigms, and that they appeared in this order. I have in recent years found it increasingly difficult to discriminate between them in some cases, which I actually label as “reassuring”, since a more nuanced description of learning suits me better, as well as it better describes actual views held by different people. If it is something

I feel was left out of my teaching education, it is to have deeper and more intellectual discussions about the learning theories and what they were for, to compare them in various settings and with various purposes, to see what they revealed and what they hid from sight. Since this is what I was missing, and what I felt all descriptions lacked, I will write about it here.

The three theories are *not* examples of smaller or wider definitions of learning, with behaviorism as the most primitive understanding and the social constructivism/social practices perspective as the most complete, most advanced one. Instead, they represent various ways of directing ones interest. I would rather associate them with the Heisenberg principle of uncertainty than with hierarchical Venn diagrams. They contain their advocate's whole spectrum of interests, epistemology, ontology and *enemies*, and it reminds me of political debates when someone in favor of one perspective describes the others. This makes it a hard life for someone interested both in socialization processes and, for instance, cognitive load. Each perspective allows and favors some questions and some types of answers, but is either completely blind for, or disinclined to admit any advantages of seeing, the questions and answers emphasized in the other theories. Advocates for socio-cultural perspectives might be easier to find in "pure" pedagogy than in didactics, since didactics to some extent presupposes a field of knowledge as a separate entity to either harvest, take on to cultivate, become part of or in any other way relate to.

A feature of human relations, that can be utilized in the design of learning environments, is the power of expectations and the urge to not let someone who trusts in you down. This is something that could be seen using the socio-cultural perspective, but might also have been interesting for the behaviorists to study. They would, however, strive towards different goals. The behaviourist would want to establish a causal relationship, answering the question "what is the response if we tell people that we have certain expectations on them". The socio-cultural perspective would strive to describe the context and the experiences that could lead to this effct, mostly answering *what* happens in the relation between the student and the teacher, but also *why* it happens, although not with the claim to have established a rule. The cognitive constructivist perspective does not primarily deal with these social aspects on learning and motivation.

On the other hand, I do, for instance, believe that such a thing as cognitive load exists. It matches my own experience as well as all narrative around me, but if I want to discuss it I am at loss with Säljö's theory, since it is not aiming at describing such phenomena and is also, to some extent, denying the existence of such phenomena since they are intrinsic processes and such are not really part of the theory. The behaviorists would definately ignore internal processes, as we cannot study them directly, but also all social aspects on situations that cannot be expressed in terms of stimuli and responses. In many situations, the behaviorist perspective really is less sophisticated, but there are legitimate questions to be asked about behavior as well as about interaction or internal representation of the world.

The above statement of my position is my defense for using concepts from any of these theories, when it makes it easier to describe or understand something. I would like this to be no more controversial than performing a calculation in whatever space it is easiest to perform it, and transform between the spaces whenever needed.

2.3 Teaching

Teaching is an art. The researchers of those disciplines which are sometimes used as motivation for teaching methods do not always agree that their theories imply any particular teaching method. Important when designing a course or curriculum is, apart from a conscious view on learning and knowledge, some insight into assessment, feedback, cognitive requirements of certain tasks, and best practices as reported by other teachers and researchers. What are the goals? How can I reach the goals? *Constructive alignment* is a principle for designing course activities and assessment methods relevant for the intended learning outcomes. The method was devised by John Biggs and is described for instance in [11]. It is based on constructivism. The responsibility of the teacher, or curriculum designer, is to make sure that goals are communicated and appropriate tasks are set for the students for them to be able to reach the goals. The assessment is criteria based and like the course activities, aligned with the course goals in order to provide good learning and feedback opportunities. When setting out the criteria for various grades, teachers often rely on taxonomies like the SOLO taxonomy, also described in [11], or Bloom's taxonomy [12], to discriminate between qualitatively different levels of understanding of the subject. The idea is that this is important, to avoid arbitrariness in grades, which may otherwise be the case if the final grade is based solely on the quantity of correct answers to posed questions which are selected in a less systematic way.

2.4 Cognition

Pattern oriented instruction (POI) and *variation theory* (further described in section 2.7) both deal with the actual concepts of the topic of a course, and the way these are presented. POI [40] is based on *cognitive load theory*, a cognitive psychology-related theory about human memory, learning, and limitations. Cognitive load is when our working memory needs to keep many separate concepts, facts, ideas and impressions available at the same time, and with too high cognitive load, learning is supposedly impeded. If we have organized the things we need to direct attention to in a certain situation in a *schema*, we can reduce cognitive load since the connections between various parts of a schema are not separate issues to remember. See Sweller, Merrienboer and Paas [52] for a fuller description. POI is an attempt to decrease cognitive load by helping students construct schemata that correspond close enough to the agreed-upon structure of the discipline concepts. If we can help students relieve themselves of some cognitive load by acting on whole schemata instead of separate concepts and connections, this should enhance learning. Hence one way of reducing the cognitive complexity of learning a task, is to structure the ingredients well and explicitly discuss and compare structures.

Instead of just showing the various concepts, as I know them, I need to make explicit what I believe are important aspects of these concepts, and how these connect to each other. Instead of just trying to teach NP-reductions "as I know them", I need to know exactly what structure I see in NP-reductions, and how I connect them to other topics. Sometimes this will only be my personal representation, but there are also aspects that most practitioners of a discipline agree on – and take for granted.

2.5 Formative assessment and feedback

In a recent study, Jessop, El Hakim and Gibbs [28] found that there was a significant correlation between both quality and quantity of feedback on one side, and students awareness of course goals and standards on the other side, confirming earlier findings on assessment. Assessment can, according to them, be seen as having at least three main purposes: measuring achievement, improve learning during the course, and develop the learners' skills in assessing quality of the learned field in the future. The first purpose can be served by a once-and-for-all assessment of the "outcome" of a learner participating in a course which is called *summative* assessment – a typical example is a final exam. That type of assessment can also contribute, together with similar experiences and in the long run, to establishing a mental model of the standards and quality criteria to be used in the future, but it is not particularly useful for improving learning *during* the course. The focus of *formative* assessment is on improving achievement by describing how the assessed work conformed and failed to conform with course goals, that is, making clear what the gap is between desired outcomes and de facto outcomes so far. Feedback is central for this type of assessment to work, and there may be concerns about both the production of good feedback, and the reception of feedback in general.

In [23], an overview on what was known about assessment practice ten years ago is given. On courses with only final exams, students perform statistically worse than on courses with coursework assignments as well. Students have to prioritize among tasks, and are often very strategic in their learning efforts – only what is likely to be assessed is studied. Coursework, as opposed to only a final exam, is preferred by students, results in higher average marks, are perceived as more fair and predicts long term learning better. Also the quality of the learning has been shown to be higher with coursework. Feedback has, according to this overview, been shown to be the most influential factor when evaluating quality of Australian courses, but feedback is often not read by students if provided in written form, at least not if it is accompanied by a grade. Several criteria for useful feedback in higher education are given:

1. Sufficiency, both in quantity and in detail level.

2. Focus on performance and actions rather than individual characteristics. Grades can negatively affect students self-efficacy, if perceived as placing the student in a specific category. The opposite is feedback "which tells students exactly where they have gone wrong and what they can do about it".

3. Timeliness. Feedback is of no use after it is to late to improve achievement.

4. Appropriate in relation to purpose of assignment and its assessment criteria. The feedback must help clarify the standards; what counts as good and less satisfactory achievements in for that particular task. This will help students determine what is required from them in the future.

5. Appropriate in relation to students' understanding of what the task is set for. The artificial contexts of essays, lab reports and other academic tasks make little sense to students, and if the feedback and what the students expects of the task are too far apart, the feedback will only confuse. The same goes for the students' conception of learning, knowledge and the discourse of the particular discipline.

6. Feedback is received. There is no guarantee that students will read written feedback, no matter its quality. To address this, teachers are suggested to provide feedback without marks, use two-stage assignments with feedback in the middle, require self-assessment or at least requiring the students to specify what they feel that they want feedback on – and abide by those requests.

7. Feedback is acted upon. It is not if it is too late, context-specific, backward looking, discouraging or unspecific, and maybe there needs to be some follow-up on what the students did with the feedback.

The findings of [28] also imply that feedback works better if implemented coherently over an entire educational program, not only in a course.

Lab exercises, criticism and radical constructivism

One common type of formatively assessed coursework is lab exercises. They are often utilized to provide hands-on experience, combine theory and practice, enhance learning, and so on. In a Swedish text, Hult [27] criticizes that naïve belief in lab assignments, and argues that in order to have any of the anticipated positive effects, exercises need to be framed by preparing discussions, feedback, and follow-up discussions. Otherwise students get a fragmented image of the subject area, and do not connect the activity of the lab with the theoretical parts of the course. In an article on the relationship between cognitive psychology and education, Anderson et. al. [1] propose similar remedies to what they believe is a severe lack of scientific basis for trends in education, which the authors describe as struggling between two prescientific philosophies – either the associationist perspective to just show students what they need to see to form the correct associations, or the rationalist perspective of allowing them to discover what they need to learn. The article goes on to mainly criticize radical constructivism with both its social and cognitive branches. They describe successful teaching interventions based on both behaviorism ("the standard whipping boy for new reform movements in education") [1, p. 228], and radical constructivism, and interpret this as an example of how difficult it is to establish a causal correlation between the learning theory of the educational approach, and the resulting learning outcomes. They still stress that their cited experiments were actually evaluated. This, they mean, is not as common as it should be with teaching interventions. Also these authors are worried about the future of education, should science in related fields not become more integrated in educators' mindset. They strongly object to the conclusion that since learning is not passive information recording, all students must discover their knowledge without instruction.

Returning to lab assignments, they can therefore be used correctly to great benefit for the learners, or out of a religious belief about their superiority, with or without benefit for the learners, depending on whether feedback and other course activities are related to them or not.

Self-efficacy

Self-efficacy, from a constructivist perspective, is the image you create yourself, about your own ability to do something. From a social learning perspective, self-efficacy is what you come to think about yourself and your abilities to do something, after seeing your interactions with others and with the subject. The term was introduced and has been further described by Bandura [10]. The self-efficacy beliefs an individual holds about some task are specific to that field, and not necessary to self-efficacy beliefs about some other type of task. They can be measured either because increased self-efficacy is an intended learning outcome [41], or as an indicator that some teaching intervention was "working", i.e. increased learning. In that case, the self-efficacy instrument needs to be compared with actual performance and the self-efficacy beliefs must be correlated with performance. In many settings, self-efficacy beliefs have been shown to correlate with performance and motivation, among other things.

2.6 Proof's educational functions

In a theoretical paper on the role of the concept of proof in education, Hanna [25] reasons about the many functions proofs can have within mathematics (verification, explanation, systematization, discovery, communication, construction of an empirical theory, exploration of for instance definitions, and incorporation of well-known facts into new frameworks). Hanna also views the explanatory function of a proof the most valuable in classroom settings, and describes to have found this aspect of proofs surprisingly important to professional mathematicians. A proof is "better", more convincing, if it also explains *why* a statement is true. Proofs by contradiction, and proofs by induction, are not examples of explaining proofs [25, p. 9]. For my classroom setting, if Hanna's evaluation of the various functions of proof is common, it means that students are used to proofs for explaining, which is a somewhat pedagogical purpose.

Hanna also mentions *epistemological confusion* as a possible reason for apparently unmotivated beliefs and actions by mathematics students. One example of this is that in experimental sciences, verification follows proof with more ease than in mathematics. When dealing with facts about the world, you can prove theorems but they are only valid in the mathematical model you have chosen, and you might also want to verify that they conform with reality before accepting the proven statement as a fact about the world. This epistemology might influence how students perceive proof also in mathematics education. In conclusion, both "proof as in educational settings" and "proof as in the real world" might cause students to display behavior that is perceived as problematic by mathematics teachers and mathematicians.

There is also some discussion about the visualization and experimentation software that begun to appear when computers got sufficiently sophisticated and commonplace. Such software have, even by educators, been seen as something that could *replace* proofs. Since Hanna's article is the introduction to four other papers, about successful use of visualization software for understanding mathematics and deductive techniques, there is also the conclusion that with carefully designed tasks, providing the necessary opportunities for students to meet with the many cognitive activities involved in constructing a correct proof, and with teachers providing relevant input, students' understanding of the concept of proof could improve after using this type of software – and that without these factors, the software will be less effective.

A more recent contribution to that field is the work of Prusak, Herschkowitz and Schwarz [44], where students on their way to becoming mathematics teachers are provided with software for geometry and a task containing the following elements: a cognitive conflict, a collaborative situation, and the software as a device for checking conjectures with.

An understanding of the social aspects of proof can be built on the three ingredients described by [50], where classroom "proving" activities for younger children are investigated. Stylianides defines a proof as a mathematical argument involving

1. a set of accepted statements (what the community agrees on being true)

2. modes of argumentation that are valid and known or accessible to the community

3. modes of argumentation representation – ways of presenting the arguments, that are accessible to the community.

The article describes research on younger children, and the reference community in the analysis is the classroom community. However, there is a need of aligning the sets of accepted statements, modes of argumentation, and of argumentation representation, with the mathematical community if we want to educate people to make them part of that community, or at least not entirely lost within that community. Otherwise, the set of accepted statements might involve non-mathematical "truths", and false theorems, and the mode of argumentation could include empirical evaluation or testing.

Proof and proving skills

Balacheff [8] devised a taxonomy for various approaches students can have to proof. The approaches are all main types of either *pragmatic* or *conceptual* proofs, where pragmatic proofs depend on showing something, and to various extent requiring the audience to reconstruct the reasons for the actions, whereas conceptual proofs requires a decontextualization, a depersonalization and a detemporalization of the actions and the objects on which the actions were made. Later Varghese [53] made complementary notes and constructed examples of the taxonomy, suggesting that teachers would easier find out what approach each student chose, and address the concept of proof accordingly.

The approaches of the taxonomy are *naïve empiricism*, *crucial experiment*, *generic example*, and *thought experiment*. The approaches are considered hierarchal, but students can move back and forth between them. All the three first levels are various proofs by example, but the generic example approach uses the example in a way that would make it relatively easy to make the proof valid for all cases. It differs from thought experiment in that it begins with, and refers to, an example, but it is the characteristics of the example that are interesting, not the example itself. The taxonomy is concerned with approaches and not outcomes, so it is not required that the thought experiment approach results in a valid proof. In the examples by Varghese, it seems like the generic example approach will always be missing something to make it general enough, but Balacheff states that the two first approaches do not generate valid proofs, while the two second *can* do it. They differ in how much they depend on explicit actions; in how much of the actions required are internalized. The two first levels differ in that the naïve empiricist approach uses randomly selected examples, whereas the crucial experiment approach selects some examples for their possible status as more difficult, although the examples provided in the literature suggest that the size of the problem instance is the most common characteristic to vary.

The taxonomy can be useful as is. Balacheff also argues in a Piagetian way that there are cognitive developmental stages corresponding to each level, although he describes the approaches rather as states in which an individual can be, than as characteristics of the individual.

In a study on Taiwanese undergraduates, Ko and Knuth [33] use classifications of student proofs according to categories by Harel and Sowder: inductive proof scheme, non-referential symbolic proof scheme, and syntactic proof scheme. The first category corresponds to Balacheff's pragmatic approaches, the second is when students use symbols and formulas without demonstrating any understanding of what they mean, and the third category is when students use relevant theorems and definitions in their proofs. The authors, however, classify their subjects' answers as *no response*, *restatement*, *(invalid) counterexample*, *empirical*, *non-referential symbolic*, *structural* and *completeness*, and find too many answers ending up in the two first categories. No answer fell into the completeness category. For counterexamples, up to a quarter of the respondents managed to produce valid counterexamples, so that task seemed marginally easier. The authors are not certain why some students believed that restating the proposition to be proved, maybe in more technical terms, could prove it to be true. It may be the case, that the students merely wanted to show that they had understood the question, and did not at all believe that their rephrasing it would prove it, but that is impossible to say without further information. The course in this study was Advanced Calculus. The authors also stress that students do not only need proper knowledge on involved concepts, definitions, axioms and theorems to produce proofs – in addition they need some cultural understanding of what is expected of proofs and counterexamples. They also raise the question whether the teaching assistants understand proofs well enough. Jones [29] also lists several previous papers where students are found to perform unsatisfactorily when producing proofs, and investigate how prospective UK teachers, who have finished undergraduate mathematics degrees, perceive proofs. Jones finds that there are many students who have good grades in mathematics, but little explicit, reflected knowledge *about* proofs. Technical excellence is not always followed by a rich conceptual understanding.

Stylianides and Al-Murani [51] found that the method of inquiry could affect the result, when looking for a particular misconception about proofs: that a proof and a counterexample can coexist, for the same assertion, among secondary students in the UK. When surveying, it seemed like students could have this misconception, while when interviewing, there was no evidence of anyone having it. The hypothesized misconception stemmed from two earlier findings, in different studies: in 1988, Balacheff (as cited by [51]) had found that some students believed that counterexamples were exceptions and did not affect the truth of a statement, and in 1982 Fischbein [22] had found that students were not convinced that proofs were the "final word" about the truth of statement, so that the statement no longer needed any further checks. If these two misconceptions about proofs and counterexamples would appear at the same time, it would be possible to believe in the coexistence of a proof and a refutation of some given statement. Similar results were in 1993 obtained by Chazan [16]: when asked about empirical "proofs" and deductive proofs in geometry, many of the interviewed subjects either believed that "evidence is proof", that "proof is evidence" or some combination of these attitudes. For instance, many people responded that "there could always be a counterexample". Despite this, there was still a preference for deductive proofs also among the people who believed that they were not universal and among those who believed that measuring a lot of triangles constituted a proof.

That students can consider all ways of getting convinced equal, and not bother with what makes for a valid *mathematical* proof, suggests to me that students/pupils might view the proving activities in the classroom in another context than intended, and that this can happen with students at all levels. Instead of seeing that proof is a central method of mathematics, it becomes a usual method of getting convinced about something, on par with all other methods of convincing oneself about facts in real life, or learning. If students perceive the intended goal of the lecture to teach them about a *particular problem*, then the proof is only one of the things the teacher might use in order to explain the problem and its solution, and the proof approach is less important. This is however contrary to findings in a study among secondary school teachers, who viewed proofs as content of the mathematics course, but not as tools for communicating and learning mathematics [31, 32], so it would be very interesting to interview those who hand in proof-by-example solutions about their view on the classroom activities, and the likely goals of the teacher with these activities. My hypothesis is, however, supported by the reasoning of Fischbein [22] and Balacheff [8]. Fischbein claims that intuition comes more naturally than deduction in many situations, and that the urge to further test an assertion verified deductively stems from a need to understand intuitively, not only to prove formally. Balacheff remarks that the crucial experiment approach (except for when used to construct a counterexample) has another meaning in everyday social interaction than in mathematics – it is used as a defense against opposition. If the mathematical context is not noted by the participants in the interaction, crucial experiment comes naturally to many. If my students have not realized that it is mathematical proofs we are doing, and that this is a necessary ingredient in the discipline of theoretical computer science, they might not care whether they reason formally or intuitively, as long as the reasoning gives some support for the statement involved.

In a paper from 2011, Zerr and Zerr [57] describe how the students of their study are far better at correctly classifying correct proofs as correct, than incorrect ones as incorrect (96 % success rate vs. 62 % when only some errors had to be spotted, and 35 % if we require students to find all errors). They argue that this is likely if all practice students have regarding proofs is to understand those in the text book and those shown by the teacher. In the study, peer-assessment of proofs and grading of both initial proof, assessment, and revised proof are performed.

It is clear that proofs have been acknowledged as troublesome for students in mathematics education world wide for decades.

2.7 Variation theory

The term *variation theory* is the label of one of the components of a phenomenographic learning theory presented in [36] and in [13], early described by [46]. Phenomenographic research in itself is a descriptive, qualitative method.

Variation theory emphasizes the available and experienced dimensions of variation in relation to anything learned, for instance that the number of legs should always be four and an appropriate size range for the concept of "cat", whereas some variations are not allowed and changes the category of the observed object, or the variation presented in a mathematics textbook for young children, where the equal sign gets to represent an operation, a conclusion; it is always there, while the numbers before it change, the mathematical operation between the numbers change, the non-varying phenomenon is that there is an equal sign and a space to put the answer. The impact of this presentation when it comes to understanding equations is discussed for instance in [30].

The authors also warn that actual and perceived variation is not the same – we can benefit from directing students attention to the variation, and to give them multiple opportunities to experience variation.

Variation theory as both a theory of learning, and an educational research framework is well described by [15], where the aims of traditional phenomenography – to describe – are seen as insufficient, whereas the aims of variation theory are seen as explaining *why* the described phenomena occur. They explain the essence of variation theory as *discernment* – to actually discern critical aspects of some phenomenon, *awareness* – to be able to direct attention to critical aspects, and simultaneity – to be able to keep several critical aspects in focus at the same time, and understand that these are connected. Critical aspects of a concept might be things like size, shape, color etc., but are more abstract for more abstract concepts. There are also aspects that are not critical, and the learner needs to realize this as well. Bussey et al. write "Variation theory suggests that students' awareness can be focused on these critical features when they are allowed to experience variation in those features" [15, p. 14]. Significant patterns of variation are *contrast, generalization, separation* and *fusion*. Contrast is to compare a critical aspect of a concept with something that is not a critical aspect, generalization is to deduce critical aspects by comparing similar instances of the concept and see the common characteristics, separation is to be able to separate various critical aspects from each other, and fusion is to be able to discern several varying features of the concept at the same time.

According to variation theory, in a learning situation, the concept or phenomenon of interest is an *object of learning*. It has three sides to it, or there can be three different objects of learning at work at the same time: the *intended*, *enacted*, and *lived* objects of learning. The intentions reside in the universe of the teacher, educational software designer, or text book author. The enacted object of learning is defined by the context, for instance a classroom situation, and the lived object of learning is the way the students perceive and understand the concept of interest. Educators are attempting to influence the lived objects of learning, but these are only accessible through student's descriptions.

Bussey et al. also claim to extend variation theory by including prior experience and instructional material design in the picture, but according to my interpretation of the concept of "the teacher", it already includes all people trying to influence the learner in any way, not just the classroom teacher, and “the student” already includes the prior knowledge of the student. It is, however, worth mentioning that “the teacher” might be a split personality if the text book was written with one intention and is used with another intention. Both affect the enacted object of learning.

I find this theory useful for me when dealing with my topics and structuring material for lectures. One of the necessary steps in a variation theory approach is to understand a concept and its dimensions of variation in my own experience, and then to try make all these dimensions both visible, and simultaneously in focus. The other necessary ingredient in a variation theory learning study is to gather authentic descriptions of the concept from students, map them to hierarchically richer understandings, and ponder on the differences between the not-so-complete understandings and the canon of the discipline, to direct my attention in the future. In a real, institutional teaching/learning situation, there is likely not a session of deep interviews with all students, but in some way or other, information of this same type is delivered to the teacher to use as feedback and inspiration for next iteration. Provided, of course, that the teacher actually asks questions that require students to practice discerning important features of a situation and to focus on them simultaneously and not just get all the data as well as directions on what method to use, and then prove to be able to follow these directions. Jones [29] has shown that the ability to achieve good results when solving mathematical tasks, including proving, is not necessarily linked to a complete and rich understanding of the topic of proof, the methods and their limitations.

Possibly related to variation theory, at least to previous experienced variation, is *Lack of closure*, a mathematics didactics term for the difficulty of accepting expressions as “answers”, for instance this example by [26] where 11–14 year old students are facing the question: “If you know that $e+f = 8$, what can you then say about $e+f+g = ...$?” The children are often deeply unhappy about the expression $8+g$. Some prefer $8g$ because it doesn’t look like something you have to calculate, but others believe that they are required to come up with a numerical answer, and suggest 12 “because it is the simplest answer”. The shift from arithmetics to algebra requires that students start seeing expressions as objects, but their previous mathematical experience interprets anything with a mathematical operator as a process. This lack of closure-effect resembles how some ADC students seemingly feel about reductions in proofs – shouldn’t they result in a solution to the new problem? What is the purpose of answering with some “intermediate step”?

2.8 Threshold concepts

Threshold concepts (and transformational learning) is a theory about how learning happens, emphasizing irreversible changes in the learner’s epistemological and ontological notions [37]. There is also a notion of threshold concepts as the actual concepts, learning of which have eight characteristics:

Transformation There is a shift in epistemology which gives the learner access to new ways of thinking about the concept and about the world.

Integration Previously separate concepts, known to the learner, are connected by the understanding of the threshold concept.

Irreversibility A learned threshold concept changed the learner, and the learner cannot go back to the previous state of understanding it.

Troublesomeness The concept is troublesome in some sense – counter-intuitive or difficult.

Boundedness A threshold concept does not explain a whole discipline.

Discourse Once learned, the learner gains access to perspectives, tools and language of a discipline.

Liminality Learning a threshold concept is similar to a “rite of passage” within and beyond some liminal space consisting of partial understanding, possibly involving cognitive conflicts and the requirement to participate in a game not yet learned (for instance chemistry, mathematics, or theoretical computer science).

Reconstruction The learner needs to experience his or her identity differently, as well as the world.

The idea of connecting threshold concepts with variation theory has been explored by [7], and in [54] they are discussed in relation to schema theory and cognitive psychology. Both of these papers focus on how to design instructional experiences in order to enhance learning.

In [45], Rountree and Rountree critically analyze threshold concepts of computer science education, and their likely implications and usability for teachers.

Chapter 3

Theoretical computer science topics

This chapter does not present new findings, and consists of basic information about TCS for those already familiar with it, but the presentation will be needed for readers who are from outside the TCS community. Some of the theory taught in the course that has been of particular interest for this study will be described in this chapter.

3.1 Algorithms for problems

One of the fundamental concepts dealt with in the Algorithms, data structures and complexity (ADC) course is *a problem*. This word is connotated with numerous associations from other fields, and from everyday language. Even in the following, sometimes we will have to refer to a problem as an issue, a complication, that has to be dealt with. In earlier stages of school, in science and math classes, a problem has a meaning close to *a task, an assignment, an example question*. When discussing algorithmic problems, the problem is a generalized version of that – while a page in a math text book can be full of various "problems" on multiplication, for TCS purposes, these are all problem *instances* of one single problem called multiplication. The characteristics of a problem is hence that there is some question to be asked, and in each instance, some parameters (input) can vary. Since *problem* then can have two meanings: problem (instance) on an exam, homework or text book, and a (generalized) problem, there are also different things that can be referenced by the name *solution*; either the answer to a specific problem instance, or an algorithm that solves the generalized problem for *any* valid input instance.

Hence, "solving" a problem in the context of the ADC course typically involves presenting such an algorithm, together with convincing arguments on why the algorithm will solve the proposed problem (instances) and how time and/or memory consuming it will be depending on the input size.

When the course mentions a *solution*, it generally means the constructive solution to some particular problem instance. Problems can usually be phrased either as *decision problems*, *optimization problems* or *construction problems.* They have roughly the same input data, but are asking different questions. Take for instance the three questions: *Is the edit distance between the two words A and B as large as X?* with a yes/no answer, *How large is the edit distance between the two words A and B?*, with a numeric answer, and *How can I change word B into word A with as few operations as possible?*, with a sequence of operations on word A that turns it into word B, as many as the edit distance, as an answer. Out of these, the third way of posing the question may seem most natural for any application (what is the use of knowing that something is possible if we don't know *how* it is possible?), but the first way of posing the problem is the simplest and decision problems are used for classification of problems. However, the *solutions* discussed are usually constructive solutions, since we often want to verify characteristics of these solutions, and that is not possible for a "solution" containing only the answer "yes" or "no".

Algorithms are usually presented in *pseudo code* – step by step descriptions of how to solve a problem described using normal programming constructs as conditions (*if–else*), loops (*for, while, etc.*) and some elements from mathematics that make for compact but easily read descriptions of procedures, for instance set descriptors. At the same time, there is no fixed syntax for pseudo code (as there is for programming languages), which means that there is no formal definition of right or wrong. The genre is defined by its purpose – conveying meaning in a compact, structured and sufficiently but not too detailed way for someone on a sufficiently sophisticated skill level to easily grasp the important steps involved in the algorithm, and the principles for the solution.

3.2 Computational complexity

In theoretical computer science, the *computational complexity* is a central concept. It captures "how difficult" it is to perform an algorithm, or for a problem: "how difficult" it has to be as a theoretical minimum. For problems, a typical concern is whether any algorithm currently known matches the theoretical lower limit, which if so means that there is no gap between what is theoretically and "practically" possible. (In quites, because the model is of a certain nature, and a good algorithm might still be too bad for actual use in computer programs.)

It is not evident what "difficulty" means here, or how it should be measured, and in fact, we use several models. The most common "types" of difficulty is that something takes long time or requires a lot of memory space. When dealing with *time complexity*, we want to compare how much time is needed for performing certain tasks. A common unit for measuring time in everyday life is seconds, but for tasks that will be performed on a computer this will depend on the computer, and how fast the fastest computer in the world is will change over time. We want comparable results over time, so we have to use something else. Also, obviously, multiplying 3 by 5 and multiplying 30121105622310001 by 100000134298131 might consume very different amounts of time. This means that the time complexity depends on the size of the input, and that it depends on how many "unit operations" that need to be performed in the algorithm. If you have to perform one operation per digit in the involved numbers, for instance, then the time complexity for the algorithm is *linear* in the number of digits in the input numbers. However, if we have to perform 100 operations for each digit in the input numbers, the time complexity is *still* linear. We do not care how many operations you have to do for each digit, as long as it is a *constant* number of times. In this model, all we care about is how the function of the input size behaves *as the input size increases without bound*. We say that we measure complexity asymptotically. If an algorithm has time complexity $O(n \log n)$, *Big O* of $n \log n$, it means that if the input size is n, the algorithm will need at most some constant c times $n \log n$. What constant is not really relevant, since for all constants the function of $cn \log n$ will have the same asymptote, and approach it when n grows sufficiently large. Also note that we do not bother to specify what base we have used for this logarithm. The reason is that we can convert any number from one base to another, and that the difference is always a constant factor since any base is fixed, and hence considered a constant. We consider constants as tame animals, with a fixed value, while any variable involved is looked upon as potentially infinite in value/size.

When we have to take into account the *bit size* of the input, we call the model bit cost. There is also a simplified model, where we know that we can ignore the number of bits of the input values because of limitations in their size (then they are considered constant) or because there is something else that will provably dominate the running time. This model is called *unit cost.* If an algorithm is analyzed with *both* unit cost and bit cost (a somewhat artificial situation, since you should always use the appropriate model), the difference in time complexity is often only a factor $\log n$, $\log n$ being the number of bits required to write n, but the difference can be significant – solving a problem by means of algorithm A can go from intractable to tractable by failing to take into account the number of bit operations that will be needed. Of course, if this is the case, the unit cost model is *not* appropriate for analyzing the algorithm.

To be able to compare running time of algorithms, or lower bounds on problems, a student needs to have a sense of the relative growth rate of various functions. n grows faster than $\log n$ but slower than n^2, and so on. Really bad algorithms are *exponential*, for instance 2^n. There are also problems which cannot be solved *at all* by any computer, no matter how fast it is, for logical reasons. To understand the arguments for this, a student needs to be able to deal with and draw conclusions based on proofs by contradiction.

3.3 Algorithm construction methods: dynamic programming

When constructing algorithms, there are some common principles to choose from. *Greedy* algorithms always make the best available choice at any time, and the greedy condition is local at each time. For some problems, this approach yields a globally optimal algorithm. For other problems, it might not even result in a correct solution.

Another principle is *divide-and-conquer*, which is based on iteratively (recursively) constructing smaller problem instances until you get a size where some condition is easy to verify, and then putting together the solutions of the subproblems to a solution to the original problem. These are common methods and the ADC students are used to seeing them, but assignments can of course vary in difficulty.

A third method that the ADC students need to deal with is called *dynamic programming* (DP). There are two ways of using it. The first is called top-down and is equivalent to using the recursion and storing all computed values in a table, also known as *memoization*, and the second is called bottom-up and can seem is a bit backwards and more difficult to grasp. It is the bottom-up version that is considered in the included papers.

There is a similarity between divide-and-conquer and DP: both problem types can be solved recursively. However, for divide-and-conquer, the subproblems do not overlap and each subproblem only needs to be solved once. In DP, there can be considerable overlap between subproblems. The textbook example is the Fibonacci numbers, where a recursive solution will have difficulties with calculating the 15th number in the sequence. Dynamic programming is concerned with only calculating each subproblem (instance) once. By starting at the base cases, and calculating the sequence until we reach the sought index, we can make sure that we perform no unnecessary calculations. This order of evaluating the subproblems is the bottom-up approach. For other problems than numeric sequences, it can be tricky to come up with an evaluation order that deals with each subproblem needed exactly once, and always has calculated all “direct” subproblems once it evaluates any larger subproblem. To solve a problem with a DP algorithm, a student needs to present the algorithm, and to argue that it is quick enough and that it performs its task correctly, that is, that it always finds a solution and that there is never a better solution than the solution the algorithm returns. There might be *other* solutions just as good, but the algorithm needs to return only one. Typically, when arguing that the algorithm is correct, the student argues that the evaluation order reaches the same result as the recursive algorithm would reach, and that the recurrence relation correctly models a solution. Possibly because both actually constructing the algorithm and proving that it works can be difficult tasks for the students, the correctness arguments are, in my experience, often omitted by them in their written solutions to hand-in-assignments.

3.4 Complexity classes: NP

Once we have established how to measure complexity, we can start classifying problems according to their complexities. The main dividing lines used for this is *polynomial*, *exponential*, *finite* and *undecidable*. Problems solvable in polynomial time are problems that computers are well suited for solving. We can define algorithms and write programs that step by step calculate the results without trouble. For exponential problems, only very small instances can be solved in reasonable time. It is easy to make a computer hang on calculating larger instances, and if it would be left undisturbed, it could go on for as long as the age of the universe. And even that it is not as bad as it gets. The worst thing is problems that can *never* be solved by any computer. Impossibility results are possible to prove mathematically, without testing or enumerating all possibilities, for instance by proof by contradiction.

The problem classes we deal with here are mainly the ones called P and NP. They are classes of *decision problems*, which means that they pose a question that has answer “yes” or “no”, and nothing else. The “same” problem can be posed in slightly different ways, as described earlier, and one of them is often a decision problem.

For some problems, we know that they are solvable in exponential time but we do not know if there is some clever algorithm that could solve them efficiently. By “efficiently”, we mean in polynomial time. This does not imply that there is, at any time, any *practically* feasible way of solving large instances of the problem, since the powers of n and the constants of a polynomial are allowed to be as large as we want, as long as they are fixed numbers.

However, decision problems solvable in polynomial time are members of the complexity class P, and problems solvable in “non-deterministic polynomial time” would be solvable in polynomial time if we at any time when a choice had to be made, could be certain that we made the correct choice. Hence, there cannot be more than polynomially many choices involved in the process, and given a (constructive) solution to the problem instance we could in polynomial time *verify* the solution. The class of such problems is called NP. All problems in P are also in NP, but whether the reverse is true has not been established despite intensive research activity all over the world for decades. It is widely believed, but not known, that the classes are different. This means, that it is widely believed that *there exists no efficient algorithm for solving some problems in NP*. Actually *proving* that this is the case is one of the open problems in TCS. The most difficult problems in NP are called NP-complete which means that if we could solve them efficiently, then we could also solve all problems in NP efficiently.

Proving membership of NP

There are two common ways, that the course material uses, to prove membership of NP for a problem. One simple way of determining whether a problem belongs in NP is to show that given any *positive* (i.e. where the answer to its yes/no question is yes) problem instance and a solution to it, we can devise an algorithm that runs in polynomial time and verifies all conditions required to establish that the solution really *is* a solution to that problem instance.

Another way which, while more similar in style to many other ideas conveyed in computational complexity, seems to cause more trouble is to show that the problem can be solved in nondeterministic polynomial time. This means that while solving the problem, at every point we need to make a choice, we ask an oracle which will direct us to the right choice. If an algorithm to *solve* the problem, using such an oracle, runs in polynomial time (not counting whatever activity the oracle would have to perform), then the problem is solvable in non-deterministic polynomial time. One common way to express this in sample solutions is “guessing” a solution and verify that it matches the criteria of the problem. This non-deterministic guessing is often only referred to as “guessing” only, and I have heard many confused students try to explain the connection between being able to guess the solution, and still not being able to solve the problem efficiently.

Proving NP-completeness

Finally, after discussing the complexity class NP and its members, we might be interested in how hard a problem in NP can be. The hardest problems in NP are called *NP-complete*. They all share the property *NP-hardness*: if an NP-hard problem can be solved by an algorithm in a certain complexity class, then so can all problems in NP. Not all NP-hard problems are members of NP.

For one problem, the *satisfiability problem* (SAT), there exists a proof based on definitions of computation, that any problem in NP *can* be expressed as a SAT formula in polynomial time. Such a formula consists of *clauses* of *literals*, which are either boolean variables or negated boolean variables, and conjunctions and disjunctions. An example of a SAT formula with two clauses and four variables could be $(x_1 \wedge x_3) \vee (\neg x_2 \vee x_4)$. This formula is satisfiable since we only need one of the variables x_1, x_3, x_4 to be true, or the variable x_2 to be false, to have the entire expression evaluate to true. If all possible truth value assignments fail to give the entire formula the value *true*, the formula is unsatisfiable.

That all other problems in NP can be expressed as SAT formulae means that SAT is NP-hard. This way of showing that all problems in NP can be expressed as SAT formulae, in polynomial time, and in general showing that a problem can be solved using an algorithm for another problem is called *reduction*. The idea behind the term reduction is to make an unknown or difficult task simpler, making it into something we know how to do, but the meaning is mathematically not used in such an everyday sense. A reduction is a transformation of the input of a problem to another problem, never mind the respective difficulties of the problems. It is in a way equally possible to reduce something simple to something hard, as it is to reduce something hard to something simple. Sometimes, like in this type of proof, we reduce something *known* and *difficult* into something *unknown*, in order to show that the unknown must be difficult too.

For all problems other than SAT, say for instance problem A, we do not need to find this explicit proof that any other problem can be expressed in the parameters of A. We only need to know that any problem in NP can be reduced to SAT, and then show that SAT can be reduced to A in polynomial time. Now all actions we have taken takes polynomial time into account. This procedure is said to show that A is *at least as hard as* SAT, which means that A also is NP-hard. As soon as some problem has been shown to be NP-hard, it can be used in reductions to prove that some other problem is NP-hard, in the same way as SAT in the example with problem A. We assume *transitivity* for our *at least as hard as*-judgements.

If a problem is a member of NP, and is NP-hard, then it is said to be *NP-complete*. This classification is an important part of the ADC course, and students need to know how to prove NP-completeness. They need to know what they have to prove: that a problem is in NP and NP-hard, and they need to be able to perform the proof and know what every part of it actually proves.

Chapter 4

The teaching and learning situation

4.1 Swedish students, the ADC course and its grading

Most of the results in this thesis are based on studies of the course ADC (Algorithms, data structures and complexity). For readers who are not familiar with the Swedish education system, or who are interested in how the ADC course works, this chapter provides an explanation of what the course is about, how it is arranged and assessed, and where it belongs in the education system.

The Swedish education system

Swedish children are by law required to go to school for nine years, between ages 7 and 15 (inclusive). Most children go to a school-like pre-school at age 6 and to other types of pre-school between ages 1–3 and 5. No differences or specializations which have any formal bearing occur during the mandatory school years, but there might be space in the schedule for choosing one subject freely out of a set of available choices, among them foreign languages (English excluded, since it is mandatory).

The next step is *gymnasiet*, a non-compulsory school form for students aged 16-19, three years. In practice, almost all youth have to study there, since most employers require a *gymnasieexamen*, the degree. There are several programs or specializations, but the same programs should be available throughout the country. Some are designed to lead directly into a profession, and some are only preparing pupils for further studies. The more technical specializations meet with the entry requirements for engineering programs at university level. There are also bridging programs for students who did not choose science or technical specializations in gymnasiet, but decide later on that they want to pursue a university degree in that area.

Tutoring is free on all levels, and students receive government subsidies. Additionally, university students can borrow money from the government to finance living during studies.

Programming is not mandatory in any of the specializations in gymnasiet, and is not part of the entry requirements for the engineering and master program in computer science at KTH. Discrete mathematics is not a base requirement either, but is more common to have studied and there have been differences between universities in whether the course containing discrete mathematics is counted as an entry requirement.

If Swedish university students fail a course, they are generally entitled to re-take the entire course next time it is given, rather than losing their right to study. Additionally, students at KTH and some other technical universities are allowed to re-take exams to improve their grades, within certain limitations. Bonus points are not valid for ever, since they are for distributing the work load over the semester, and old activities will be forgotten after a few years.

ADC – Algorithms, data structures and complexity

The course ADC is taught to third year students on the Master of Computer Science and Engineering program at KTH Royal Institute of Technology in Stockholm, Sweden. In this course, there are large hall lectures, held by a professor, smaller tuition sessions in classrooms with worked problems, held by teaching assistants (TAs): PhD students or master students who have already taken the course, and computer lab sessions with TAs available to help and to assess finished assignments.

Grading is performed in relation to fixed criteria on an A–F scale. The Swedish university grading systems were changed to conform with ECTS (European Credit Transfer and Accumulation System, an attempt to make grades in all of the EU comparable) in 2007, but the grades do not, at least not everywhere, mean that the lowest 10 % of the passing students received an E. Since the ADC course encompasses many topics, and the expected proficiency level for students at each grade has to be described for each topic, there is a matrix describing what is required for each grade. Examination is performed continuously and in various ways. E is the lowest *passing* grade. This means that students with an E on this course actually have managed to demonstrate knowledge of the basics of all of the topics involved, and it is not to be considered a bad grade. The same goes for every grade: to earn it, the student needs to demonstrate that he or she knows enough on *all* areas of the course – not that he or she on average meets with, for instance, C criteria. Hence, “percent of the grade” is not a meaningful descriptor of any examination task.

This system of grading is transparent and fair, but complicated in comparison to “old” systems of weighing all activities together and *calculating* the grade. Because of this, and since KTH courses are undergoing changes with respect to examination, each year some students comment on the difference and argue that this system is too strict. Recently (in 2011), the grading and assessment in gymnasiet has been redesigned to something more similar to the ADC course than before.

There are three major pieces involved in the grading: two individual homework assignments and a written exam. The homework is to be handed in in writing, but no grade is assigned until the student has presented the solution to a teacher/TA. If the written solution is too poorly written, or lack important parts, a failing grade is given. Otherwise, the student presents the solutions and answers questions about it, so that the teacher can get a grasp of whether this student knew enough for the grade he or she aspired on. There are usually three tasks, and solving one satisfactorily is required for an E. Solving two gives a C, and solving all three of them gives an A. The tasks are designed to meet with different criteria which means that the problem statements convey different amounts of information, and the assessment of the first task requires less perfection than the assessment of the third task. What criteria each task is testing, is written on the problem statement. One smaller error is accepted, but lowers the grade one step.

The written exam only tests E-C level tasks. It is rather short, and followed by a peer assessment session. To prove knowledge of more difficult tasks, if their grade on the exam is at least E, students can solve a programming task and present it to a teacher (no written report in this case).

If a student has a passing grade on all three examination assignments and at least C on two, he or she is allowed to try to improve grades at an oral exam. Then the student chooses what examination assignment the oral exam should substitute (provided that the grade will be sufficiently high), and is assigned a task that corresponds to the examination part and the grade that the student chose. The student gets one hour to think about it. Afterwards, the student can present the general ideas of his or her solution to a teacher during 30 minutes. Students are allowed to try improving their grade on more than one examination part, but no more time is allotted.

Mandatory computer lab assignments (only graded pass/fail) are also part of the course examination. Students solve these in groups of two, and then present and describe their solution to a TA. The role of the TA is to try verifying that the students understand their solution (that is, that they probably have written the code by themselves), and to look at the coding style and algorithmic approach, choice of data structures, etc., and discuss these with the students. Three of the lab assignments are automatically assessed for correctness, but on one of the assignments the TA also needs to verify that the program works and solves the correct task.

To help students get started with the lab assignments, there are optional, peer-assessed theory questions for each lab. These are handed in in written form, and discussed in class. If solved, in time and to a satisfactory degree, they render bonus points for the written exam; the same goes for the lab assignments themselves.

The format of instruction in the course is one hour lectures with the course responsible three times a week, two hour worked example sessions with a TA once a week, and computer lab help available and scheduled in a booked computer lab two hours a week.

4.2 Long-term goals for the ADC students

What does it mean to be a theoretical computer scientist, and should the students become theoretical computer scientists? The students of the ADC course are generally not continuing an academic career within theoretical computer science. They *are* going to become familiar with the type of questions and the type of answers that characterizes theoretical computer science. The course contains programming exercises as well as theoretical assignments and homework assignments. The programming tasks take a lot of time to finish for many students, and some actually think that these are the main content of the course. When working on a written assignment about proofs and algorithms, entirely different perspectives need to be taken, and it is not clear that the students do this. This leads them to seeing important ingredients in a written proof as "petty details".

The things I, as a teacher computer scientist, want students to know and remember from this course is not necessarily that Strassen's algorithm is based on decomposition and which steps are included in Ford-Fulkerson's method or what complexity it has. I want them to know that computational complexity is a foundational concept, and I want them to be aware that we want to compare algorithms by their inherent complexity, regardless of current level of technology, and express for instance "speed" in something that does not inflate as computers get faster. I would like them to know that there are different models for calculating "speed" of an algorithm, for instance unit cost or cost per bit. These are generalizations, very blunt instruments, and the analysis may at times lack application in real life, where in contrast you want to know something more specific, including constants and sometimes actual seconds... Time is only one efficiency measure. There are always resources limiting performance of computations, whether these are time, memory, bandwidth or something else. There are several models for measuring complexity, with respect to various resources. The main one we focus on here is time.

I want them to know that theoreticians are worried about super-polynomial complexity, whereas practitioners may be worried about anything above linear, depending on area of application. I want them to know that problems are generalized descriptions of a situation with a certain specification of input and of output, and that everything matching these specifications are problem instances of that problem. I want them to know that algorithms are for specific problems, and need to be able to deal with any instance. Also, I want them to be aware that the algorithms need to be proven to be both correct and efficient, and that "mostly correct" is a concept that has very little space in theoretical computer science. To the extent that it exists, it deals with one-sided errors and probabilistic algorithms with either correctness or efficiency only with high probability.

I want them to remember that there were such things as problem classes, with problems sharing some complexity characteristic. I also want them to remember that to prove that a problem is as hard as possible within the NP class, you need to be able to reduce something at least that hard to your problem. Therefore the reductions in proof by contradiction is a standard tool in theoretical computer science. I want everyone to remember that there are problems with intrinsic complexity so high that they cannot be solved by any computer at all, ever, and not just because no one has been clever enough yet. (But that of course *some* of the instances can be solved.)

In conclusion, if they remember that the course was about computational complexity, that you were typically worried about how efficient your algorithm was, that the most common type of complexity dealt with in the course was time complexity, measured in numbers of operations or in numbers of bit operations, and that there was an important border between problems efficiently solvable and harder problems, and another between at all solvable problems and intractable ones that would be great. If they also remember that you need to prove correctness, termination and efficiency for any algorithm you design, and that some common techniques for designing algorithms are divide-and-conquer (decomposition), dynamic programming or greedy choices and that also data structures need to be analyzed with respect to complexity in the same way as algorithms; that the choice of data structure is as vital as the design technique for an algorithm, then they have grasped the main themes of the course.

However, *during* the course I want them to be aware of more things. They need to remember some of the sample algorithms we have been discussing, in order to be able to compare new problems with old problems and algorithms for old problems, and then decide whether the old algorithm will work as is, could be modified to work, or will not work at all for the new problem, due to some characteristics of the problems. I want them to be able to express what they generally intuitively know is true – that an algorithm works – using arguments that are solid and specific. Sometimes their intuition is wrong, and I would like them to notice this when working on a proof, seeing that proofs are not for teaching situations, but for researchers and programmers to know for themselves that their thoughts about some problem are sound.

4.3 Kattis background

There is a hint of discrepancy in the way students and teachers view algorithmic tasks regarding *correctness*. Teachers want it to be an integral part of the problem solving process to check that the algorithm is (provably) correct and that the implementation, in the case of programming tasks, does not fail on certain test cases. Students see that algorithms can be incorrect, and that implementations can be, but they may try arguing that their solution works for *most* cases. Implicitly, they then want a reward for trying and possibly an honorary mention for actually succeeding, while the teacher is likely to not assign a passing grade to someone who has not tested an implementation or who cannot argue correctness of an algorithm. To a teacher, a solution is always correct (or has a probabilistic guarantee of some kind). To a student, there are solutions, which should be rewarded, and also a dimension of correctness that can be applied on solutions, so that we can have incorrect, correct or maybe also mostly correct solutions. While it is possible to argue that someone who can design an almost correct algorithm might have understood the problem at hand better than someone who can only design an algorithm that is completely misbehaving, small and rare errors in implementations are harder to catch and might be disastrous, given a test procedure that cannot test all possible input.

Ultimately, our goal is that students develop the ability to critically analyze their algorithms and implementations, identifying corner cases and identifying arguments for correctness that are at least proof sketches. For programming tasks in the ADC course, students work in pairs and are passed or failed by a human teacher/TA. However, their tasks are often to write sufficiently complicated programs that the teacher might not be able to spot potentially interesting corner cases during a ten minutes discussion with the students. If the teacher's task is to judge whether the implementation actually works, this will easily steal focus from everything else, and still many students might pass while their programs have errors.

To support students and teachers in this respect, and to enforce efficiency by running all programs on the same machine with the same resource limits, we use automated assessment for the lab assignments. Before presenting their program (orally) to a teacher, the students submit their code to the system Kattis and get a response directly when Kattis has run their code on all the pre-designed test cases. Students then have to show that Kattis accepted their code, and the teacher can discuss other things with them.

Every year since 2006, when Kattis was first used in this course, some questions on the course evaluation have been about Kattis. There is no direct indication that students distrust the verdicts from Kattis.

There are, however, signs that some students do not consider program correctness as a criterium for satisfiable performance on a programming assignment. There are also indications that the use of Kattis, while relieving the teacher of the error-prone and tedious task of testing all programs very thoroughly and while enforcing the teachers' view on the relationship between correctness and solutions, discourage some students from testing or writing "proper", easily human-readable, well-structured and commented code, simply because this is not tested by Kattis. Among the answers to the course evaluation, there are always some students claiming that they got an entirely new perspective on correctness after a course that was as "picky" about it as this one, and other students who say that they test their programs less than they otherwise would have had, since Kattis takes care of that for them. There are also always some complaints about not getting to see the exact inputs on which the program failed – crashed or returned an incorrect output.

Part II

The research

Chapter 5

Method

The main work presented in this thesis concerns one particular course at KTH Royal Institute of Technology. It is a mandatory, intermediate course on algorithms, data structures and computational complexity, abbreviated ADC. The ADC course is renowned among the students as difficult, somewhat different and (among a considerable fraction of them) really interesting and fun. On the other hand, there are students graduating from KTH hoping that they will never, ever again encounter the topics of the course. This course is the arena for most investigations and improvements in my thesis because I belong in the category who enjoyed the course, but also because the teacher of this course is determined to make it accessible and interesting to as many students as possible.

The papers included in this thesis are loosely held together by an action research approach. This has several reasons: we want the course to improve as much as possible, and as soon as possible; we do not want to experiment with applying likely improvements to benefit only half of the students (experiment group and control group); and we want input from the previous initiatives to influence every next step we take.

The included articles address which topics are difficult, and how students respond to attempts at making the topics less difficult. Together with the previously published results, some additional, qualitative data has been gathered from surveys and group interviews with teachers and from the actual student work presented during the courses.

5.1 Action research approach to ADC

While continuously trying to improve the course, the action research cycles eventually united with the academic year, so that there was one cycle per year.

First cycle – computational complexity and reductions

The first cycle lasted the longest, encompassing several years and some sub-cycles. It started with the observation that many students complained that computational complexity, complexity classes and reductions were much more difficult than algorithms and data structures. Those parts of the course are different from one another not only in content, but also in context. While the first part of the course seems like a natural continuation of earlier data structure and algorithms courses (like CS1), the complexity part is mathematical to its nature, and even though computers and coding still come into the picture, the "culture", by which I mean the type of tasks, the type of (model) answers sought for and possibly also the language, changes. Hence, the first attempt was to try connecting the type of course work required in both parts of the course. There was already one written homework to be presented orally, and questions on the theory exam, for both parts of ADC, but there were lectures, worked example sessions and computer lab hours scheduled for the algorithms and data structures part, and only lectures and worked example sessions for complexity. Less teaching occasions could imply "less important", and the difference in culture was possibly more strongly perceived when one kind of activity was omitted.

The first change to the course was to *add* a task – a computer lab exercise on complexity. This task is special in comparison with the other computer lab exercises of ADC. It requires only half a page of code, whereas the other lab assignments are much larger. The task was automatically checked for correctness in the system Kattis, so that students could work when- and wherever they liked. This seems to have remedied one of the problems – that complexity was not connected to other coursework and hence not practiced enough.

Another characteristic of this task is that it is in effect mostly on theory, and not about efficiency of the program. By design, it was under-specified when it came to input and output – there were descriptions of three problems, their respective in- and output formats, and the task to code a reduction for the sake of proving one of them being NP hard (the two others were already known to be NP hard.) We have of course absolutely no interest in teaching students on this level to transform input of one format to input of another format. Instead, we want students to create a mental model on when you can do this, and for what purposes. To pass the examination of this task, you need to know what your program does and *why it does what it does.* You need to be able to argue why you can know this, i.e., produce some correctness arguments of the method. A teacher assistant will, after you have shown the Kattis feedback, assess your shown ability to do this.

The lab exercise was initially voluntary but later became mandatory. It seemed to affect the average grade on the home assignment on complexity, adjusted for the average grade on the earlier assignment on algorithms. Several years passed with this setup.

The "real" first cycle began when we attacked the weaker performance on the complexity part of ADC on several fronts, simultaneously. We introduced visualizations developed by another teacher teaching similar material in an exchange where his school tried out our lab exercise. The lectures were changed to a shorter format, but more interactive, and “clicker questions” were prepared. These are short questions with several answer options available, and among the reasons to use them is to promote reflection and student activity, and to help the teacher plan the remainder of the lecture based on how the class answers. Also, previous students were contacted and asked about their work experience with computational complexity, and this resulted in a motivational lecture being introduced, explaining that such content actually does show up in work situations, how it can look there, and how you tackle intractable problems. Students’ results were still monitored as usual, but we also added a cover page to the homework, where students could fill in *where* they had learned a list of core competencies for solving the homework, and a final survey aside the usual course survey, asking specifically about the new activities. The course survey and the final survey were anonymous, while the homework cover page required students to write their name on it.

Second cycle – dynamic programming

After the appreciated changes of the complexity teaching, some students of the previous year realized that although they were familiar with algorithms in general, dynamic programming was an algorithm construction method that they were not used to. To them, the previous setting scaffolded and helped them more with complexity than with algorithms. Since the extra lab on complexity was on the same topic as the homework, you could practice on the lab, but for algorithms there were no preparation exercises on dynamic programming that were assessed *and* mandatory. As dynamic programming was perceived difficult, the next cycle was dedicated to teaching it better, and evaluating knowledge and, to some extent, attitude, better. We used the same methods that were earlier utilized for complexity, but here we also tried structuring the material according to the principles of Pattern Oriented Instruction, since one of our concerns was that there might be too much details and involved steps for the students to cope with.

In addition to the evaluations similar to the first cycle, students’ attitudes in form of self-efficacy beliefs were also investigated, before and after the teaching of dynamic programming. As the self-efficacy surveys were to be compared between the two occasions, students were asked to give out their names on both. We promised not to use them for any course-related purpose other than this research, but wanted to be able to see self-efficacy beliefs among student who varied in achievement level.

A group discussion with all teacher assistants who graded the homework was also conducted and recorded. This was done with the purpose of providing the TAs yet another opportunity to reflect on the teaching and to develop as teachers, and they were paid meeting wages for attending. Other purposes were to improve the course, and to get more information both for teaching purposes and for the research.

We wanted them to discuss the questions we asked, not just have individual answers from each one of them.

Third cycle – pseudocode and proofs

A task that students often claimed incompetence in is writing pseudo code. This might be connected to the lack of formal syntax for pseudo code, but also to it being taken for granted in teaching. Something that teachers (and some students) wanted to improve was the ability to write proofs for algorithms and reductions. Students appeared to be too focused on *inventing* the algorithm, that is, finding out what strategy would solve a given problem. While productive and successful for the mere construction of an algorithm, this is not sufficient to fulfill the course requirements. When asked *why* an algorithm worked, few students had prepared logically consistent arguments for their homework presentations. The proofs were maybe perceived as "add-ons" but not essential when judging whether a solution was valid or not. This discrepancy between what students and teachers believed about the role of proof in the ADC course, made us want to try improving the *teaching* about proofs. The content we wanted to address this cycle was never the main topics of lectures, and while introducing some lecturing about it, most work went into constantly reminding students that they were seeing pseudo code and proofs during lectures and example sessions on other topics. Both pseudo code and proofs are culturally important tools in theoretical computer science, and they are integrated into everything else in the course.

To get more detailed information on how teachers appreciated the students' knowledge at the oral presentation of the homework, a rubrics-based protocol was constructed and used this year. The protocol by necessity had to be constructed primarily to support grading, but for some items on the self-efficacy surveys, there are corresponding protocol entries that can be used to compare students' self-stated confidence in their proficiency level with teachers' assessments of the same students' skills.

All three cycles partly involved making silent knowledge visible and explicitly teaching *about* things that were previously learned implicitly, via constant exposure and "osmosis", and not learned by all students. Identifying which concepts and tasks have aspects of silent knowledge, what language needs to be explicitly defined and repeatedly used in a deliberate way to help forming an identity as "someone who knows some theoretical computer science" is in my opinion an important, but maybe forgotten, part of teaching this subject. The same goes, of course, for other advanced subjects as well, which students are supposed to gain some familiarity and skill with during their education.

With the data on what students felt confident doing, and what they did not know how to do, we tried to understand what parts of the involved topics are the most difficult, and discussed such aspects of the subject of theoretical computer science. We also set up a web survey for teachers teaching different courses that had something in common with ADC, to collect information on what other teachers had found to be problematic or to require extra planning and attention. Unfortunately, we did not receive so many responses.

5.2 Data

The data in these studies come from our course assessment records, students' admission grades and previous study results, the annual anonymous online course evaluation, anonymous surveys about the research projects, identifiable responses to self-efficacy surveys, self-stated proficiency levels at homework due dates, and every other communication channel between teachers and students of the course has of course been watched carefully to spot potential problems and issues needing to be addressed. This includes oral feedback at homework presentations and lab presentations, emails from students asking about some particular concept they find difficult.

I have used the package knitr by Yihui Xie and the R package xtable to transform my data to LaTeXtables. Data has been stored in spreadsheets, in a SQLite database, and in .csv files. Spreadsheets have been used to *input* data, which has later been exported to .csv files and, after some processing, into a SQLite database together with export from the grade administration systems for querying preference purposes. Python 2.7 scripts are transforming the data before it enters the database. After introducing R, this workflow may not be optimal, since R integrates with both databases and .csv files, but keeping the processing scripts intact appeared as the least error-prone solution.

The group interview data with TAs was used for input to the next iteration. There is no complete transcripti of the interview, but some quotes from that interview also appear in the discussion of difficulties students can have.

The survey for teachers was collected digitally, and has also been used as a source for new ideas and to compare our experiences with those of other teachers. The written comments of that survey are the sources of information, since the purpose and setup of this survey would not make statistical data treatment legitimate.

5.3 Validity and reliability – Using grades as evaluation

There are obvious problems with assessing the outcome of our experiments by looking at grades we have assigned our own students. On the other hand, if there had been no grades, we would have tried to establish how well people were doing by assessing it somehow. This is a characteristics of action research, but of varying degree depending on which people are involved. However, all involved parties are part of the project and none is an objective, outside observer. Since the criteria for each grade in this course are well described and discussed among teachers, grades are not arbitrarily assigned and personal bias is less influencing than without detailed criteria. Also, it is our own teaching, and our own hopes for outcomes, that we are investigating. It might, however, be worth mentioning that teachers assigning homework grades normally did not know which, or how many, teaching and learning activities each student had participated in. They could probably find out, but investigating this is not encouraged, and the work would not be paid. The self-efficacy beliefs questionnaire and the grades were never compared during assessment, as the self-efficacy questionnaire was kept in sealed envelopes until after the course. This means that the un-blind method could affect the teacher's total experience of the course, but that would have to include seeing the whole student cohort better than last year's. Other things affecting teacher's grading include rumors about how well this year's students did compared to last year's on previous courses.

Self-stated proficiency levels

The calibration of data of the type "How well can you do X?" between different people will vary, but hopefully people's individual perceptions of the available levels will not vary between occasions, so comparing the same individual's responses at different times could still reveal something about changes of some underlying knowledge. On the other hand, when something is regarded *harder* after the course than before, it is possible that the students who lowered their self-efficacy scores changed either their quality criteria, or their perception of the entire topic, and hence are answering "another" question afterwards than before the course. We wanted to see if there was any correlation between how certain students were that they could perform some course-related tasks, and how well teachers found them performing similar tasks. In order to achieve this, we listed all self-efficacy items and all assessed tasks of the course, and marked for which items it could be possible to correlate answers with some assessed task. We then did the same thing for the errors that could appear on the homework grading protocol, supposedly more fine-grained than the grade: marked which errors should correlate with lower scores if the self-efficacy correlated with these performances.

Chapter 6

Results

This chapter summarizes the included papers and their results. Paper II and Paper III are not part of the overall action research project. Paper II instead sheds light on how simple feedback affects students' attitudes, and Paper III is about a separate action research inspired project, explaining the automated programming assessment system Kattis and its underlying philosophy. Kattis is highly relevant for the teaching and learning environment of the ADC course, and the Kattis results that follow the summary of the included publications are from ADC course surveys. Paper VI is a journal version of the previous ADC results, together with the proof and pseudo code results of 2013.

The chapter also presents some data that is not discussed in the included publications. These consist of the group interview with TAs, descriptions of common errors, difficulties, misconceptions or communication issues on homework assignments, the web survey for TCS teachers, and student responses to questions about Kattis on the annual, anonymous, course evaluation survey of ADC. All these sources of information are only used qualitatively: the group interview and the descriptions of homework errors to provide some information on how the students' performance was perceived by TAs, the Kattis survey question to exemplify how Kattis defines the context of the course, and the questions to other TCS educators in order to see which topics they mentioned as problematic for a broader perspective on the topics than provided by the ADC course itself.

6.1 List of included publications

I **Computer Lab Work on Theory**
Appeared in the Proceedings of ITiCSE 2010, Ankara (June 2010)[19].

II **The effect of short formative diagnostic web quizzes with minimal feedback**
Appeared in Computers & Education No. 60 (2013), p. 234–242 [9].

III **Five Years with Kattis**
Appeared in the Proceedings of the 41st ASEE/IEEE Frontiers in Education Conference, Rapid City (Oct 2011) [21].

IV **NP completeness for every student**
Appeared in the Proceedings of ITiCSE'13, Canterbury (July 2013)[17].

V **Dynamic programming – structure, difficulties, and teaching**
Appeared in Proceedings of FIE 2013, Oklahoma City (Oct 2013) [18].

VI **Iteratively intervening with the "most difficult" topics of an algorithms and complexity course**
In submission.

6.2 Summarized results from included papers

Here follows a summary of the results of each of the included papers.

Computer lab work on theory

Some issues with the theory part of the course were probably overcome. The complexity homework was not anymore perceived as remarkably more difficult than the algorithms homework, and students started appreciating to get to practice NP-reductions before the homework assignment. On the other hand, taking tools from one paradigm and applying them within another paradigm, can cause confusion as to what the activities mean, what their purposes are. The programming exercise *did* get people to work more with the complexity part of the course, but at the same time did not make them aware that they were now doing something conceptually different from when designing algorithms to solve problems.

The effect of short formative diagnostic web quizzes with minimal feedback

This article is not about the same course as the other ones, but it is concerned with automated assessment and feedback. Already at the level of providing really basic, multiple choice questions with minimal feedback, a fraction of the students report starting working harder because they found the questions more difficult than the lectures, while others did not feel the same urge to get the teacher's blessing on everything they did as the teacher was used to. They got confirmation that they were *not* falling behind, and they needed that. Students were reaching out for all types of prepared questions and activities providing feedback. These were mostly beginner students, not used to the topics in a basic programming course, some of which may have been under the impression that programming probably was for someone else. It does show that many students, at least in beginner courses, are not good at judging their own capabilities, but also that many of them know that.

Five years with Kattis

This work is a description of how our school works with automated assessment. Kattis is the system used, and it is an ACM ICPC style programming “judge”, to which source code is sent and which sends back responses after testing the submitted source code using secret test cases.

Students are happy to use Kattis, but sometimes reproaches Kattis for being so strict. The acceptance may have risen during the years. Students sometimes want to argue over the quality criteria for a lab assignment, but it is of course impossible to argue with a server. Kattis serves as scapegoat for unpopular CPU time requirements and errors on tasks with difficult corner cases, and teachers appreciate that they do not have to play that part. Also, grading has probably become more fair after introducing Kattis, since human teachers cannot run test cases and spot errors as efficiently.

Indicating something like this is the fact that the first year ADC used Kattis, fewer students were accepted on time. We do not believe this to be due to particular difficulties with the interface of Kattis, but rather a consequence of thorough testing. Another course significantly increased students’ passing rate after introducing Kattis, since students had more opportunity to spot errors in time, supporting the interpretation that Kattis in itself does not make courses harder. Students appreciate not having to be worried about forgotten corner cases at the lab deadline.

From theory to practice: NP-completeness for every CS student

This paper is a follow up on Paper I. Students having difficulties with complexity still existed, not surprisingly, after introducing the lab assignment on complexity. Here we describe a more thorough attack on the topic and how to present it to students. The work is done in cooperation with Pierluigi Crescenzi in Florence, whose students were demonstrating similar difficulties with NP-completeness.

The approach is to treat the tasks more like algorithmic tasks, both in presentation and in homework assignments, but also to focus on motivating the usefulness of complexity and on what is meant by a reduction in a proof by contradiction. We introduced visualizations, the programming assignment, a motivational section on a lecture with old students’ statements on how complexity mattered in their work, we managed to isolate some of the important things about NP-completeness reductions in a separate task, and we made the lectures more interactive. We also handed out self-efficacy surveys before and after the teaching on complexity. Students on the ADC course improved their grades on the complexity homework, on average, if they had been participating in many of the activities we provided, compared to how the same students performed on the algorithms homework earlier in the course. Students were also given a questionnaire about the activities, and we could see that all activities were perceived as beneficial for some group of students.

Dynamic programming

Out of the algorithm construction methods of the ADC course, dynamic programming is seen by the students as the most challenging one to grasp. The method involves many possibly troublesome concepts and procedures, and we could not be certain exactly what the problems were. We made the same changes to the teaching as we had previously done to the complexity teaching, and also tried to structure the material with respect to variations, themes and common patterns, in line with the cognitive psychology related teaching method Pattern Oriented Instruction. The teaching on dynamic programming only lasts for two weeks, with four large lectures and two tutoring sessions with problem solving, but still we tried to bring more structure into the many tasks involved in constructing a dynamic programming algorithm. We also handed out self-efficacy surveys on this topic, and used the results partly to see whether students experienced themselves improving their knowledge, and partly to see which tasks had the lowest average scores and hence may have been the most difficult.

Together with other surveys on where students believed they had learned things, we could see that correctness proofs, choosing a suitable problem to reduce, and the direction of the reduction were the most difficult tasks, and that students had learned most on lectures and on working with the theory questions to the programming assignment, and the assignment itself. Some tasks many people marked that they had still not mastered. They correspond vaguely to the self-efficacy survey items with the lowest scores.

Iteratively intervening with the "most difficult" topics of an algorithms and complexity course

This paper is an extension of the work in papers 1, 3, 4 and 5. It presents complementary data from the most recent years of ADC for the results in previous papers, and adds survey items on pseudo code and proof construction, which have seemed to be difficult parts of both dynamic programming and complexity. The increased efforts to explicitly point out pseudocode examples and proofs during ADC 2013 did not seem to have an overwhelming effect on the final self-efficacy beliefs of the students that year, compared to 2012.

We also try to evaluate some of the self-efficacy items in relation to how well students are performing when trying to take on tasks resembling or including the tasks of the self-efficacy items. When plotting the self-efficacy scores for the items we saw any possibility of verifying, against the results of the selected assessed tasks related to each item, there was for most items no trace of any correlation. For supposedly negatively correlating indications – errors reported on the homework grading protocols – there was too little data. TAs were not required to put everything that was wrong in the protocol – it could at times be very difficult to decide what the assessments should be. Instead, if fatal errors were found, the TAs could mark these in the protocol and leave out any other errors. This experimental method to validate the self-efficacy items failed, in one case because of missing data and in the other case either because the items do not correlate with achievements, or because the self-efficacy items and the examination tasks are different in scope and separated in time.

When comparing the admission grades of the ADC students with their ADC course and homework grades, no correlations were found. However, there was a correlation between grades on four prerequisite courses and the ADC grades. When performing future course changes, it will then be possible to use the prerequisite grades as reference when different years are compared.

6.3 Kattis survey results

There are some themes that repeatedly occur on the final, anonymous course survey for the questions concerning Kattis. These are:

"Working programs" not accepted by Kattis Some students complain that lab assignments are harder when automatically corrected, but rarely specify what unnecessary requirements Kattis is imposing. "In lab 1 you just needed the program to work. In lab 2 you had to make it work and also needed to make kattis accept the result (which sometimes felt like 50 % of the job.)" (Student 2006). Others are complaining about Kattis not accepting the output in reverse order, or when you forget to separate all numbers with spaces. Maybe these errors, which can be seen as residing more in the programming logic than in the algorithm construction logic, could cause people to claim that Kattis does not accept all correct programs.

A great asset to have Kattis helping you In various ways, students express that Kattis helps them to spot errors and to feel less nervous for the lab presentation. "Great that you can focus on understanding the problem and the algorithm construction at the presentation, rather than whether the algorithm is correct or not."(Student 2011).

Debugging and Kattis It is probably too tempting to use Kattis *instead* of testing your program. This means that a lot of code is submitted to Kattis, and, in worst case, that even compile errors are caught by Kattis and not locally. Some students express this as a weakness in themselves, others as a feature of Kattis. "Contributes to more active debugging instead of just asking a TA for help. I have learned a lot from the lab assignments in the course!" (Student 2011), "Less focus on learning to write your own test cases." (Student 2013)

We would hope for everyone to realize that this is not an optimal workflow, and it would be desirable to be able to construct tasks that do not encourage this behavior – either by forbidding it, rewarding other behavior higher, or requiring sub tasks that need independent testing. As mentioned in [21], this can be enforced within Kattis, but as discussed in [19], design of tasks where testing is an explicit step is probably a good way to go. On the other hand, many students report testing much more carefully when having access to Kattis: “Lab assignments in other courses where Kattis is not used, then I have not tested my code much at all. At least not with so many test cases that are common in Kattis. This has made me think much more about what could be wrong with an algorithm, since Kattis does not tell you much about that.” (Student 2013)

Feedback The easiest way for a teacher to set up a programming problem in Kattis gives minimal feedback to the student: immediately Kattis responds with “Accepted” or some type of error. It is, especially after a few years of continuous improvements of Kattis, possible to set up problems so that feedback is given in other ways, including showing the secret test cases to students (although this is not a feature of the problem, but of Kattis course management). The level of understanding of why no test cases are supplied when the program fails may have risen during the years, as Kattis has become a natural part of the study environment for CS students at KTH. “But there is little feedback on what exactly is wrong. . . When it looks like it works, many hours of fumbling in the dark remains before you come up with what tests must have been used.” (Student 2007).

Risk of uglier code As with testing, some people view code quality as something that they should take responsibility for. Others are just noting the decline in this aspect. Since for some tasks the only thing that is rewarded is speed, and the test suite is fixed, students start including assembler code, or avoid object orientation since it takes more time than writing code that has other qualities than time efficiency. “Don’t know if it is due to Kattis, but I have noticed that my code has quit being so neatly commented as it were, once upon a time. . . ” (Student 2011) This can to some extent be mitigated in the design of tasks, where there should be something objectively positive to compete for, but in a course the best remedy is to let the human teachers balance the feedback from Kattis with opinions on coding style, structure and maintainability.

Students are also, reasonably, complaining about discrepancies in time measurement in 2013 and in 2011 – there were at those occasions when things did not work properly when Kattis had heavy load. They are also rightfully complaining when a test file is incorrectly formatted, which could easily happen with old problems, not packeted according to a standard. There are different flavors of “give me the test data” – from the idea that since someone is reading the code anyway, we cannot hardcode the correct answers, to the suggestion that the type of test should be hinted, or two rounds (Google code jam style), one with test data that you will receive and one with secret test data, will be applied. Over the years, the number of angry “GIVE ME the test data” comments seems to have decreased, while more reflecting comments on the same topic still remain. This may reflect some cultural heritage within the group of students, whose beginners each year are guided into their new life as KTH students by their older peers. It might also be the consequence of more teachers being accustomed to Kattis, and meeting the more "naïve" complaints during labs. Many wish for differentiated time limits between various programming languages. During the first years, Java programs had 1 s more to go than C and C++ programs, and these were the only language options. Later, the JVM was judged to start sufficiently quickly for this exception not to be needed anymore. Interpreted languages, like Python, take even more time than Java and hence require better optimizations. Some complaints are along the lines of "the asymptotical time complexity of my algorithm is OK, so I should be accepted". This suggests that these students successfully have spotted one important course goal, but try to make it the *only* quality criterium. To have to make optimizations to not over-engage the Java garbage collector, or to learn that the implementation of LinkedList in Java is *not* an efficient one, seem to not belong in this course according to some students. It is true that this is another perspective, not so much elaborated on during lectures, on efficient algorithms and programming.

The varying effect on testing and code quality needs monitoring. The TAs are supposed to give feedback on this, and make sure that the code solves the problem and that the students can explain their code. The feedback from TAs works best when there is continuity and existing relationships between the people. I have, for instance, witnessed a TA saying "Hey, this is ugly! Isn't this code far below your quality?", with the tone and effect of this TA expecting much better from this particular student, and the effect of the student not wanting to let the TA down. So, despite being eager to leave the task behind and move on to other things, without the TA really threatening to not pass the him on the assignment – that would be outrageously capricious grading with respect to the transparent grading system – the student went back to his keyboard to produce a result that was a well performed task by him, not barely enough for a less skilled programmer at the same course. Without interference like this, Kattis will contribute to bad habits for *some* students.

6.4 Verify a. . . what?

When a student is presenting solutions to the complexity homework, there are a number of points where confusion may arise. It may be possible to write something that is not incorrect, and still have no clue of what that written sentence means, and this usually shows up at the presentation. Sometimes, coming up with a reduction is seen as the main task, and other requirements of an assignment might be missed or skipped. In other cases, students are either naively interpreting words and sentences according to common sense, without reference to the TCS context, or really struggling hard with making sense out of (seemingly) conflicting statements. The latter group often have aha-moments during the presentation, when asked the right questions, and feel relieved and happy to have their cognitive conflict resolved.

The first group might also come all the way through spotting conflicting statements and resolving the conflicts, but sometimes instead end up annoyed that the task was "under-specified" and all requirements were not mentioned.

The concept we have worked most on is the reduction, and what it tells us about the involved problems. However, there are other things that bother students. What follows is a qualitative description of areas where difficulties are encountered during oral presentation of homework.

Verify a yes-instance Some students have missed the word "solution" when reading and listening to descriptions of how to prove that a problem is in NP. This means that they have to wrap their mind around the paradox of a problem being to hard to solve efficiently, yet we can take any yes instance and "verify" that it has a solution (i.e., solve it). This does not make sense, and students deal with it by either stating that they think it is weird, omitting it until being asked about what is missing, or making some interpretation of any of the words in the Swedish sentence "Verifiera en ja-instans" that in some way resolves the issue. One way is to verify that the instance is a valid problem instance. Another is to focus on the word "en", which here means "a" but when stressed instead takes the meaning of "one". So if there exists *one* yes-instance that we can verify (solve) efficiently, that should be sufficient, right?

Verify a yes-solution Well, what is a positive solution? The word "yes"? This, rather logical, approach makes the students who take it prone to attack the task in the same way as students who had missed the word "solution" entirely, and just remembered to verify something. How would we verify the answer "yes"? Well, probably by showing that yes was a possible answer to the question in the problem. Unfortunately, in theoretical computer science terms, this is equivalent to solving the problem, which is of course hard. The same ways out as for the previous item present themselves to those who have made this interpretation. The didactical problem here, is how the canon defines the tasks. The language is highly dependent on this context, and the context is difficult to derive from the language. Everyone doing theoretical computer science *knows* that NP is a class of decision problems, but nevertheless, while showing that a problem belongs in NP, the witness is a solution to the constructive version of the same problem. Maybe introducing the technical meaning of the word "witness" here could allow us to avoid some misleading associations or presumptions regarding the word "solution".

How much do we need to verify? Given that we have sorted out all uncertainties regarding *solution* and what type of solution we mean, it remains to understand what input and output the verification needs. For the problem *Set partitioning*, for instance, the problem instance is a set of numbers and the question "Can we partition this set into two sets with equal sum?". Students may verify a solution by checking the sum in both partitions, and entirely forget about checking that they really *are* partitions. They understand that the input of the verification contains the solution, but forget to check it against the problem instance it is said to solve. For instance, $\{2, 5\}$ and $\{7\}$ certainly are two sets having equal sum, but they are not a valid partitioning of $\{2, 5, 6, 7\}$. Sometimes the sample solution of a lecture problem consists of a function mapping elements to subsets, cleverly taking care of both "all elements need to exist", "no new elements may be added", "no element can be used multiple times", and the more loose concept of "which element goes into which partition". It implicitly forces us to let the verification algorithm take the problem instance into account. These multiple purposes are not discerned by all students, but when applying a pattern-matching approach they may still be able to solve another problem with this sample solution as template.

I don't verify, I guess (non-deterministically) Maybe because this method of showing NP-membership seems more formal and appears in other texts, or maybe because of the difficulties with making sense of the language around "verifying", some students prefer to guess a solution non-deterministically and show that it solves the problem instead. These students are more often confused than those who verify solutions. Here difficulties arise because non-deterministic guessing is a thought experiment rather than a valid algorithm construction method in practice. "Guessing", on the other hand, is used in everyday language and so you can be lead to assume that the problem can (sometimes) be solved by guessing, and that this is what is meant here – yet another way of trying to devise an algorithm for solving the problem instead of showing NP-membership for it.

Rephrasing as a decision problem Another issue when verifying solutions to other versions of a problem, is that it is not easy for all students to determine which decision problem corresponds to a given optimization problem, or what construction problem they need to verify a solution to. A famous NP-complete problem is *Traveling Salesperson*, with input as a number of cities and the pairwise distances between them, and the question "What is the shortest route that visits each city exactly once and then returns to the original city?". Since the answer to this question is a number – the length of this shortest route – the problem is an optimization problem. When trying to phrase a problem like this as a decision problem, students sometimes try with "Can I find a route that visits each city only once and then returns to the original city?". The issue with this is that, while certainly a decision problem, this problem has a trivial solution. Yes – we have pairwise distances between all cities, so we can construct a route that visits them all in $n!$ ways. There was something about the length of the route that was important in the optimization version, so in the decision version the length also has to play some part. Typically, we introduce a limit, assign it a variable name B, and ask for a route with length at most B. A possible distractor in this context is the satisfiability problem

SAT, which usually in its basic form asks "Is this formula satisfiable?" with no limit involved. If SAT is used as template, it will be difficult to invent a goal.

The "solution" cannot be verified efficiently If the data said to represent a solution contains too little information, verifying the solution against its problem instance might still be an NP-complete task. A decontextualized example of a problem is where we have a graph G and for all pairs of vertices (X, Y) a goal $T(X, Y)$ for how many node-disjunct paths we want between these vertices, and a limit B. Can we pick at most B of G's edges so that all the goals are fulfilled? Note that, if we let the solution to the constructive version of this problem consist of the B edges we should pick, finding all the disjoint paths is still not easy. (Actually, it is basically the same task as before.) The solution will instead need to specify all paths, and then we can check that the ones between the same endpoints are disjoint and sufficiently many, that all edges and vertices existed in the original graph, and that the number of edges in total is at most B. I remember that to me, personally, the fact that this was a valid way of defining a solution was not obvious.

Maximal vs maximum The use of the terms *maximal* and *maximum* can probably confuse also native English speakers when they turn to study mathematics or mathematics like subjects. In Swedish, there are not two similar words used interchangeably in everyday language. (Both words exist in Swedish, but maximal is an adjective and maximum is usually a noun.) In TCS, we take maximal to mean a local maximum, and maximum to mean the global maximum value. In other branches of mathematics, the distinction is defined in terms of partial orderings: if we have an element E of a set S, and there is no element of S larger than E, then E is maximal. If E is larger than all other elements, E is the maximum element. A sentence can make no sense whatsoever if the wrong interpretation is chosen.

Expecting specified cost model Sometimes, students underestimate the role that common sense *is* allowed to play when solving problems in ADC. They may expect that the teacher writes in the assignment instructions whether bit cost or unit cost is supposed to be used. This happens for easier tasks, testing for criteria for E, but never for tasks involving A criteria. The choice between models is not arbitrary – you use as lazy a method as you can get away with, and if the bit size of the input data matters, this needs to be taken into account in the analysis.

6.5 Group interview with TAs

During the cycle introducing dynprog changes, we conducted a semi-structured group interview and discussion with all TAs who had been taking presentations of the homework. They were asked about what students did well and not so well, if some of the results surprised the TAs in any way, whether the students understood their motivation for the assigned grades, and if there were differences from previous years.

This discussion made it clear that many students often have thought about their homework, and can come in and start pointing at several places where there are missing commas, missing constraints or details in error. However, it has not happened to any assistant that a student shows up and apologizes for having handed in something that really doesn't work. Students often find errors in their solutions, but as one of the TAs put it "They generally have great understanding for their solution idea, but they never appear to question by themselves whether the idea really works." He meant that students could determine that what they had written did not match what they wanted to write, and spot the differences, but not determine whether what they wanted to write really would solve the problem. The tasks of this homework assignment were by several TAs considered harder than usual. The first task was not possible to pattern-match with previous years' first tasks, the second tasks needed dynprog in four dimensions and the last one about solving something "as efficiently as possible" had the difficulty of not specifying in advance exactly how efficiently that would be. The course responsible teacher meant that this latter was not realistic, and that it would exclude several reasonable techniques, leaving open for pattern-matching instead of showing the highest ability criteria.

TAs also found that the students seemed unusually unskilled in writing algorithms in pseudo code and also in basic mathematics. As another TA put it, "Previous years, students have shown that they do not understand logarithms. This year, they grabbed the opportunity to show that they also do not know exponentiation well." Other assistants remarked that the usual problems with which logarithm "laws" really exists were often resolved by resorting to exponentiation and examples, and that this usually worked well. The argumentation from some students this year suggested that it would not have worked well this year, if tried. The issue was whether $O(2^{2n}) \in O(2^n)$, and the students meant that they could safely ignore the "constant" in the exponent. This can give some valuable insight into how students may misunderstand the asymptotical worst case complexity and which things to neglect versus to focus on in this setting. One of the TAs estimated that 30 % should have been failed based on how poorly they understood basic algorithm analysis relevant for the course.

Regarding whether students understood when their solutions were not satisfying, assistants had various experiences, involving having lengthy discussions with some students about not lowering the requirements, being impressed by the reasoning regarding dynamic programming by many students, observing that hints were helping significantly to make students see what was wrong, and finding students seemingly more focused on actually understanding the errors, than on achieving a good grade.

The most common errors for the three different tasks were, for the grade E task: inability to explain what was written, lack of understanding of the handed in solution, too vague descriptions to be assessed, or too detailed pseudo code – without touching the important steps of solving the problem. Some misunderstandings occurred of the *Master's theorem*, one method included in the course, about analyzing recursive algorithms.

For the grade C task: exhaustive search instead of dynprog (too slow), leaving some dimension of sub problems untouched, errors in determining an evaluation order, or only checking the sub problems with maximum value on one of the parameters, while searching for the best solution. (This is not guaranteed to work – some paths will not exhaust this parameter, yet reach a higher optimal value in the calculations.)

For the grade A task: too slow algorithms, or greedy algorithms that did not succeed on some simple inputs. Also here many students were unhappy when they did not receive an A because their – working – algorithms were too slow. The grade criterium for this task was to be able to solve difficult problems with the method that is best suited, that is, to compare the various algorithm construction methods and pick the one with best time efficiency.

These results were input for the last cycle, where emphasis was on more often explaining characteristics of pseudo code and what proofs were good for, and on assessing more accurately what students could and could not do, introducing assessment sheets for the homework assignments.

6.6 Web survey for teachers

In order to find out about "universally" difficult concepts, and not only things seen at our university, we constructed an open web survey and invited all teachers world wide who, according to the publisher, used the same text book, and all teachers in Sweden who gave similar courses. We also wrote to them to pass on the invitation to anyone who could have something useful to say on the topic. The results are not to be treated quantitatively, as we have no idea on how many people have seen the survey. Only eight people responded. Their answers are supposed to be used for asking quantitatively about the occurrence of several types of issues at different places in a later survey, possibly indicating threshold concepts of TCS. Those who did respond to the survey often mentioned the same things that we have studied further, reductions and dynamic programming, as difficult, along with "anything considering the p-word" (proofs). For instance, one of the respondents said that "Students often don't get the skill to write or even identify correct proofs.", and "Some students are able to prove that 1 is equal to 0, without seeing any problem with that." Also algorithm analysis, Big-O notation and invariants were mentioned.

These difficulties were spotted at homework assignments, questions around midterms, exams and by slow progress in class, e.g. by questions and by errors made by students.

We also asked about ways to tackle these issues. In Sweden, some teachers claimed that students were ill prepared for formal mathematics, but maybe worse –

that it was not widely believed that mathematics was really relevant to CS studies and professional identities. Surely, there are programming tasks that you can work with professionally without knowing much about correctness proofs, but those are not the only tasks that engineering studies are aiming for. They should also become specialists and be able to design systems that are efficient with resources and are not based on vague guesses that a method might work. So the teacher who wrote most on this, believed in informing potential students about the role of mathematics in CS to make expectations match reality better. Others mentioned many examples, programming exercise, and attempts at giving all theorems a face, a name and a history, and structuring the teaching. One said that it was also valuable to students to understand that some parts of TCS are inherently difficult, and there is no pedagogical key to this – students will have to deal with the difficulties.

Part III

Discussion

Chapter 7

Discussion

In this chapter some of the themes from the previous results are discussed further.

7.1 The difficulties encountered at homework presentations

The qualitative descriptions of difficulties related to homework presentations, nearly all concern the use of language in the course. Some students have difficulties discerning the borders between course specific language and everyday language. When viewed through the lens of variation theory, they have not experienced what the teacher specified task could be contrasted with, or they include too many options among possible interpretations.

Interpreting these difficulties in a social context seems more rewarding. The narrative of TAs, that some students appeared to not be able to explain the written solutions they handed in, indicates that they have tried to use the language of the course, but are uncertain of its meaning. There are several social ways of interpreting this phenomenon.

The inability to explain solutions sometimes stem from the solutions being copied from someone else.

Sometimes students can misinterpret the pedagogical situation, where they are to explain the solutions, for a critique towards what they have written. "If I have mentioned this, and the teacher still asks about it, I'm at loss with what else to add". For these occasion, the apparent inability to explain a solution resolves when the student expresses something like this aloud, as the TA can then rephrase the question or motivate why it is posed.

Further on, students are asked why their solutions work. Proving, and expressing solutions on a general level, seem to be difficult to students, either because of the task of proving being hard in itself, or because the context of the proving activity is not interpreted as it is by the teacher. The language used when formulating a task for students, show traces of more functions of proof: *verify*, *show*, *derive*, *argue*, *explain* and so on, as well as *prove*. If the main purpose of most proofs is, or is experienced as, explaining, these terms will all be synonymous, transferring some of their associations to the concept of proof. Additionally, if I verify something in my everyday life, I might not start with a theorem and a set of axioms and definitions. When my students present a non-proof to me, it often is a "proof by example", which *does* in some situations have explanatory properties but not qualifies as proving the proposition. I rarely get the impression that my students are unaware of why something is true, especially when this something is that the algorithm they just presented to me solves the proposed problem. Do they only lack language, words or symbols to express this, or is it the culture of proving that holds them back? Our purpose with proving activities in the ADC course, however, only strives to make the students express the *whys* in some way. Why is this true? Why can it not be in another way? We do not require strictly formal proofs, but convincing arguments that would be at the core of any proof. So the students have some knowledge of the why, and we want them to express that knowledge explicitly, refer to some causes or reasons for things being a certain way. It would seem like we are not very far apart. On an aggregated level, students were not considerably more comfortable with proofs and proving activities after the teaching was altered to more often mention these aspects of the course. If they *do* know the why:s, and *have* experienced that proving is part of the course, then it would be easier to conclude that proving in itself is the difficulty.

Explaining correctness for the solutions is not always perceived as a separate task by all students, and the idea "Wasn't what I did correct?" blocks many attempts from the TA to get the student to motivate the solution. It is, as suggested by the self-efficacy results described in [20], not necessarily in *knowing* why the algorithm is correct that the difficulties reside. It may well be hardest for a student to decide whether a description will be correct by the standards of the course. Here students almost acknowledge that they do not feel that they understand the social practice of TCS, and hence are unable to respond to the questions of the TA. Compare this with Stylianides [50], where the validity of a proof depends on it utilizing a set of accepted statements, modes of argumentation and modes of argumentation representation for a particular community. Maybe these three ingredients could be useful in lecture planning, when the teacher considers what should be explained or expressed in various activities. The teacher want the students to attain familiarity with these characteristics of the mathematical community, so instead of just *using* the modes of argumentation from mathematics, it could be worth explaining that these are accepted mathematical reasoning practices, and to point out what distinguishes them from other modes of argumentation.

Yet another interpretation of this is that the students are applying standards of other situations to establish correctness, for example stating that they have implemented and tested the algorithm and "it was fast", or that it works for a particular example. This is very similar to the "epistemological confusion" described by Hanna [25]: if the proving activity of mathematics can be applied with influences of physics, requiring the model to be verified against the real world after it has provided mathematical results, then it seems likely that the proving activity of TCS can be influenced by standards from system development, where testing is important, causing students to *test* their algorithm instead of proving correctness.

For both of these interpretations, it remains a fact that teachers have exemplified good solution practice, at least sometimes, and hopefully meta-discussed the occasions when they did not present complete solutions. However, the students may not have *experienced* these examples, nor what the alternative to an complete presentation would have been. The culture of proving is still a viable option for explaining these occasions of non-proving or apparent, and the idea behind the presentation session with the TA may look different from a student's perspective, where the written and handed in text for most purposes plays a greater part in their view than in the TA's.

Testing is another important concept in computer science education, and testing is usually not the same thing as proving. Still, it may seem so. Especially since we attribute so much dignity to the automated tests of programming assignments done by the Kattis system. This results in homework presentations where the student argues: “I have tested 2000 various random test cases, and my program works on all of them.” Well, then please explain how likely it is that any of the corner cases for your method (which are they?) would be generated randomly! Sometimes, shouldn't the probability for this be almost 0?

7.2 Reductions

Reductions in the context of NP-completeness proofs are likely to meet with most criteria for a threshold concept. Since reductions can be utilized in so many ways, some structured way of arranging possible experiences of them should, according variation theory, be helpful. In “negative” reductions, that is, when the reduction is supposed to be used for an infeasibility result, it is somehow uses backwards. Solving problems by reduction is a *fundamental idea* of computer science according to Schwill [48], and one that has been shown to appear counter-intuitive or mysterious to students. Armoni and others [3, 5, 4] have found reductions to be problematic because there are so many details of a problem that could be explicitly addressed, leading to explicit solutions instead of reductions. Some of them have also connected recursion with reductions through the concept of *reversing*, in which proof by contradiction is included [6]. In the case of using reductions backwards, maybe the most important variational aspect is what does *not* vary: the very central concept of proof by contradiction motivating the activity. This was addressed by a type of problem where only the implications of the existence of reductions was possible to focus on, and this task was appreciated. When, in previous years, only dealing with the actual reduction algorithms, ADC students may have been “blinded” by the many details of the particular algorithms, and therefore unable to experience any variation or lack thereof in the way the arguments *about* the reductions went.

Another context of reductions in ADC is related to grouping corresponding problems of other problem types (decision, optimization and construction versions, respectively). This may be something students do not get proper instructions about. It may seem arbitrary how the decision version is defined, given an optimization formulation. While true that there could possibly be options available, the main criteria should be *that it must be possible to solve the optimization version of the problem by repeatedly calling the decision version.* Here, students may expect other rules to apply in the background, making the task into a guessing game instead. (Compare with the additional requirements of the (Karp) NP-reductions that we only call the other problem once, and never post-process its answer, not even by negating it.) What we actually require from two problems to consider them versions of the same problem, could be explained more often. This may be the cause of the uncertainty regarding reductions between problem types, found in the self-efficacy survey statistics described in [20], but it is also possible that the reductions in themselves were the main problem. Elaborating more on the distinction between problems, and problem *versions*, might be advisable in future teaching of ADC.

7.3 Programming versus proving

As mentioned in the introduction, it is possible that many students saw the ADC course as yet another programming course. Anecdotally, I have met older students, especially those who aspire on being TAs, who have been enthusiastic about how useful they found the course, perhaps because it changed their perspective on the computer science universe. I have also met others who get a hurt look and ascertain that they will never, ever again deal with something like that. It would be interesting to know if this feeling is more common among the students who were most unprepared for the mathematical content of the course.

Programming and proving *are* two very different types of tasks, and to expect the first and find that you have to deal with the second could be a disappointment – or an entirely new experience, opening up a wonderful, new world. Also towards the end of the course, some students justify their homework solution by attaching source code for a program they have written, to convince themselves that their algorithm solved the problem. The "restatement" kind of answer to a proving task, similar to what was described by [33], appears in homework solutions too. Sometimes it appears as the answer to the entire homework problem and not just to the correctness proof part. In my experience, most students who have responded like that are not surprised that their homework is not accepted, so it is most likely that they know this is not a solution.

It seems to me that proofs are partly hard because students are not used to proving things, and partly hard because students are not anticipating the quality criteria to be formal. They have not wholly embraced the nature of theoretical computer science, or they only see part of it. The problems and their solutions are natural ingredients, but the proofs may be seen as pedagogical tools rather than course content. In a mathematics course, the things to prove are often theorems or statements, backed up by an implicit promise that this idea is worth proving, as it may be useful for building the next layer of mathematical knowledge on. The correctness proofs for algorithms do contribute to an intricate system of known and unknown problems and problem classes. However, the proving tasks may appear as problem solving tasks, as these activities go hand in hand in TCS. It may then be easy to miss the point that the proof actually is *included* in the task, and not accompanying it for pedagogical purposes. Typically, they have been assigned problems to solve in previous programming courses, and although these courses also prove correctness, the requirements may have been lower. Sometimes, the learning objective is to design good test cases, in which case these procedures might steal focus from proving correctness, or seem as a substitute for it.

When there are several parts to address in a proof, some students are happy with just showing one of these. This happens also with verification of a solution to an NP-hard problem: some students verify that a solution meets with the requirements of solutions to the problem in general, but fail to see that they need to verify that the suggested solution is a solution to the particular problem instance we are considering for the moment. Probably the students are capable of spotting more possible ways for a suggested solution to be false, but they are not in a skeptical mindset. It can sometimes take a lot of hints before they realize that they will have to check *all* criteria of the problem, not just the "most difficult" criterium.

By taking a tool from algorithms – write a program – to practice something inherently theoretical, we might help students to link their previous experiences to a new concept. There is however a risk that we will confuse them. It is obvious that many students view the computer lab assignment in [19] through the lens of the algorithmic context, to much a higher degree than the analogy holds. The special context of mathematics and proofs (which they are unwilling to accept in the algorithms area), is then entirely lost. The meaningfulness of the task can in that case be questioned, as there is no obvious application for coding negative reductions anywhere in professional life as a developer. When trying to make novel use of an old tool, in this case construct a lab exercise on "un-labbable" content, it is important to explain and to point out the strangeness. It is good if both students and instructors share a view on what is the goal and what are the allowed methods and reasonable strategies to use.

7.4 Automated assessment, feedback and socialization

Two types of automated assessment have been tried in the included articles. In [9], students got access to simple, parameterized quiz questions and the article [21] explains how we work with automated assessment of the programming exercises on somewhat more advanced courses. In the first example, students were not CS majors. Their response to having access to more formative assessment tasks was positive, and they wanted to keep that access after the experiments ended. The feedback was sparse, far from meeting all the criteria of good feedback posed by

[23], but immediate. The extra check point for formative assessment made some students study harder, and some students study less. These effects suggest that the simple quizzes really provided formative assessment to students.

The second example concerned more advanced CS students, who already knew that testing is considered important in system development. The level of feedback was similar, but the tasks typically assessed in more respects than correctness by human teachers. Here, automated assessment was used for what computers are better at than humans: systematically performing routine tasks and measure utilized memory and time resources. The feedback on these issues was relevant, but not pointing the students in any particular direction. This version of feedback is not "sufficient" according to the criteria by [23], but serves its purpose well if the tasks are designed carefully and the educational setting does not rely solely on automated assessment. It is a necessary criterion for a lab assignment in ADC to be solved *correctly*, so the errors found by Kattis matter to students.

Students mentioned in their course evaluation answers that Kattis affects how they think about program correctness. Some students said that they probably had never written a correct program before, in all their life. Others confess that they do not adhere to the highest standards for programming style and maintainability when programming for Kattis. Some happily declare that it is great that Kattis takes care of the testing for them (pity that they don't get to see exactly what test data the program failed on), while others note that they are lazier with their testing – or that they are thinking it over much more carefully than before. In some programming courses, students can be happy with a program that needs to be force quitted, or that always outputs error messages together with some, presumably expected, data. With Kattis, a program that does not terminate gets Time Limit Exceeded instead of Accepted, regardless of whether it calculated the correct answer or not. A program that outputs error messages will get the response Wrong Answer, because error messages are not in the output specification. This means that the teacher gets some help in having the students internalize the intended correctness criteria.

The disguised proving task

Returning to the automated assessment of the theory exercise: the complexity computer lab, feedback on it could be summarized as: You get feedback on data format if you reduce in the wrong direction, and you get corner-case-feedback in case your algorithm does not *always* perform well. The main function of the feedback might well be that you can relax before the presentation of the task. This model for feedback might be responsible for the phenomenon that students focus on formatting of output (or sometimes input). Many have chosen a very practical or format-oriented approach to the lab assignment.

Another effect of immediate feedback is that students can easier utilize a trial-and-error approach. They might submit new versions without very much consideration of the changes undertaken and their consequences. On the other hand, the iterative approach also shows that they are eager to improve their programs and that they are competitive.

Of some concern is the idea that it is difficult to "find the right reduction". The generality of the method, and the freedom of the prover, have not been internalized. Some students believe in a 1-1 relationships between instances of problems, when we perform negative reductions. This is generally not the case. That perspective is interesting, because it is rooted in the type of glasses you wear, the expectations you have on the activity, and the community and context you associate it with. It is also interesting that the endeavor for perfection can amount to unnecessary steps in the reduction, for instance attempts at solving *clique*, another NP-complete problem, in order to be able to present the problem instance for the new problem in the "aesthetically best possible way". This is likely to be associated with how the problem is phrased, and to mitigate the use of such unfruitful approaches, there is a theory question that students solve before the lab, leading the students into not assuming that the clique representation of the data is necessary.

The very detailed specifications, including sample input and output data, is a very definite way of communicating expectations. Students are comfortable with this, although some will always claim that their program was correct but Kattis complained about something extra, not included in the task. They usually do not specify what they mean by this. Maybe they mean for instance that a path from A to B is not considered the same thing as a path from B to A, or that if you forget to flush the buffer Kattis might not receive your output in time, or that if you forget space between the numbers, Kattis will not find many enough, but maybe one too large number.

At the same time, Kattis sets expectations for the rest of the course. When a task is not specified in the same way as usual, this being explained in the instructions, students still want detailed specifications for that task. It is also remarkable that students who are working hard with many tasks in a difficult course, actually *ask* to get more practice tasks for various techniques in Kattis. One motivation for using Kattis exercises in a course can be the same one as discussed for the quizzes in [9]: that the exercises might affect grade expectations and the students' perception of their own understanding towards an objectively true level, and that these are performance indicators of a course. It is probably more rewarding to work with tasks where you get feedback immediately, than preparing for tutorial sessions by working on the problems in advance.

In order to make Kattis foster a better attitude towards testing programs, the tasks mentioned in Paper I [19] probably need to get implemented – if students actually are required to submit test cases, and the quality of these test cases is assessed, the competition could be steered towards finding the best test case instead of only writing the fastest program.

7.5 Interventions in teaching and demonstrating quality criteria and perspectives

This section mainly addresses the issue of what the teaching and learning activities tell students about the nature of TCS.

When we were working on the second cycle of this project, introducing the self-efficacy surveys, a couple of students spontaneously mentioned to the lecturer that these surveys were useful learning tools. It was not a common view among the students, according to the final survey on the course activities of that year, but it might still be true. If the problem with some type of task is that there are too many steps involved, then a check list is useful. If there is some task that you did not understand was part of the actual course content, but that task appears on the survey, then you might spot it and reflect on that. To let the homework be accompanied by a list of tasks relevant to the homework, and a matrix to mark where you had learned them, might allow it to be useful as template for proof-reading your solutions and see that they are complete. These are all things that would be interesting to investigate further.

The ADC course already has grading criteria written and explained on the course web pages. On exams and homework assignments, there is a note on what criteria each task is relevant to. A grading protocol for a homework assignment is yet another way of showing students what teachers mean by quality. The oral feedback at homework presentations often help students to gain insights, but the feedback on lab assignments might vary more in quality. To explain what a “good” solution is, and show something that is not good, is necessary. The tutorial sessions after the homework, when students have tried hard to solve the problems, are useful for discussing requirements. However, nothing can be described if teachers and students are not using the same language. I believe that there is great room for improvement in exposing words and sayings which may seem to be easily understandable, but which in a mathematical context means other things than they do in everyday language.

7.6 What the ADC students learn to *be*

Here we elaborate a bit on what a computer science student needs to learn to get a grasp on TCS, and what the teaching and learning activities convey about this.

All of the discussed topics help building the practice that is theoretical computer science. Proofs take a central role, correctness becomes more salient, possibly at the expense of designing the algorithms, *efficient* means something far from what you actually measure if you have a program and some test data, and the language uses many new or unexpected terms. These are parts of the practice that the ADC students learn exists, and hopefully they become a part of. They learn to “sketch” proofs, to analyze the operations in an algorithm with regard to the number of elementary operations necessary. Many of the difficulties encountered in the course may stem from a mismatch between teachers' and students' expectations on the tasks, the classroom situations and the topics of TCS. The things students believe that the tasks are for, are probably logical with respect to how the students perceive the course and the topics, even if they are completely wrong from the teacher's perspective. If teachers manage to drag all these mismatches into the light, students will be more easily included in the TCS community.

The important thing is probably that it becomes clear early on to students that if they have apprehended the "contract" between teacher and student' according to this description, they will miss out on some important content of the course: "The teacher should tell me what to do. My job is to be good at doing it. If not told what to do, I am not required to take any action, and could not be reproached for that. The focus of every task in this course is to solve a problem. The algorithm is my job; its correctness is something entirely separate that, if not otherwise specified, is someone else's. If my algorithm is not correct, it might have other qualities that should attain to a high grade, or at least I should get something for trying! "

Since the teacher wants students to aspire on higher quality goals than "only" solving problems, they mean that the simplest type of questions in the course might follow the contract expected by the students to some extent "If you only want a passing grade, focus on performing routine tasks well and reading instructions. If you are aspiring to higher grades, you need to be able to evaluate various approaches, compare various interpretations, and choose something that is appropriate for the task at hand. These abilities are in the grading criteria and will be assessed. You will not get hints for all tasks. There are certain ingredients in solving the problems of this course that you will *always* be expected to utilize, most typically algorithm analysis and correctness arguments. A task is not solved if there is no reference to why and how an algorithm works, or how efficient it is. *You* are going to become specialists, so it is *your* responsibility to show that your proposed solutions works, as in all mathematical contexts, not your audience's."

The teachers believe that students need to learn not only to use the tools presented in the course, but also to learn *when* each tool is useful, and how to choose a tool that will help them. The problems presented in the course, and their solutions, are more background and props for the concepts and tools that *are* important. It is a delicate task to explain this, when most of the time the teacher will still discuss problems and their solutions and correctness arguments. This is a part of the ADC course where maybe more can be done regarding the alignment of learning goals and activities.

To be able to feel included in the TCS community, and to behave in such a way that you seem to be in the right place, requires you to know that the topics studied may *seem* like everyday tasks, and that there occurs both problem solving and programming, but that the field in itself is a mathematical world, where you pose hypotheses, prove theorems and sketch proofs. It may be perceived as more applied than pure mathematics, but the way in which it is applied is not straightforward. One major ingredient is a *problem*. This is a question (or maybe three different questions, linked in some mysterious way?) about some category of data you might have, and something you might want to accomplish. Unlike problem solving for the sake of the solution, many problems only have the answer "yes" or "no", where the natural follow-up question in real life would be "So? *How* do you achieve yes or no?". If you receive a constructive solution, you can see that it works, and you might not want to solve other problem instances with the same method. The follow-up question in TCS is instead "*Why* is yes/no the true answer?", "Why will this method always work?" One large set of activities only aim for classifying problems according to how long it would take to solve them, but the way of defining how long it takes differs significantly both from engineering practice and from everyday use of the words. In fact, many "efficient" algorithms would never be feasible to use on an actual computer. There are impossibility results to be acquainted with, and habits of mind like "try backwards!", "can we solve this if we could solve this other problem?", "Exactly *how* bad is this algorithm?" and "What if we add/remove just one constraint?". There are common ways of phrasing questions, in terms of languages as sets of data with some characteristic, graphs, formulae, verify, refute, oracles, witnesses, "a solution", and so on. Many of the terms also appear in everyday language, often with a less specified meaning and other connotations. Sometimes distinctions appear in one natural language but not in another, like maximal and maximum or the Swedish "en" as one/a. The discipline likes analogies, puns and historical references, too, and assign problems names in order to remember them. To not immediately associate any of these ingredients of TCS with its TCS interpretations, will make you confused or mislead you in any context of TCS. The very "essence" of TCS might qualify as a threshold concept, being transformative, non-reversible, and opening up new opportunities, short of the requirement of being limited and not a whole discipline [37].

This practice is not necessarily explained to students, but *shown* to them by the example of the teacher. The teacher, however, is also representing another practice – teaching. The students will need to interpret the actions of the teacher either as ramifications of the teacher teaching, or as examples of how TCS professionals work. When proofs are demonstrated, it could be as either of these – and if the things proven do not seem likely to be useful for building mathematics on, the logical conclusion might be that they are proven to explain and make the student understand.

Part IV

Conclusions

Chapter 8

Conclusions

To summarize the conclusions, we return to the two main themes described in the introduction. The first theme studied was:

What can be particularly difficult in TCS, and particularly, in the ADC course?

1. Which conceptual difficulties are inherent in
    a) NP completeness proofs?
    b) theoretical computer science in the course ADC?
2. What do students find difficult and what do they find easier?

Based upon the previous discussion, possible conceptual difficulties in NP-completeness proofs are what set of “rules” about what constitutes a good proof to apply in a given context, but also knowing what needs proving. According to some of the teachers responding to my survey, anything involving proofs will be considered difficult. The NP-completeness proofs also involve a threshold concept candidate: reductions.

Some of the more specific difficulties with reductions can be overcome by targeting them explicitly, like what the direction of a reduction tells us. This was shown when the rate of reductions in the wrong direction decreased after explicit instruction on reductions in isolation, with no problems specified. It is likely that this could be done for more concepts than reductions.

Maybe most inherently difficult is both the concept and the action of proving correctness of algorithms – students often display signs of uncertainty regarding this type of task, and of what counts as a good proof. Besides being a known difficult ingredient in mathematics courses, this can be related to their understanding of the purpose of proving correctness, in the ADC course and elsewhere, and this would be interesting to study further.

The general difficulties with dynprog is the entirety of the task: solve this problem with dynprog. It includes many steps, and students are more comfortable performing just a couple of them, than remembering all of them without cues or hints. Just one of the included sub tasks involved received as low self-efficacy scores as solving a problem from scratch, without hints: finding an appropriate evaluation order of the subproblems of a larger problem. Finding recurrence relations is not perceived as difficult in itself, but we believe it to be difficult when combined with all other tasks necessary for a dynprog solution. The POI approach may or may not have helped – it is possible that the short interval of time assigned for dynprog is not sufficient to utilize POI very well, but it would require much more detailed study to draw conclusions. The task of reducing between problem types – solving the construction version of a problem where access is provided to an algorithm solving the decision version, also in the context of dynprog displayed less certainty among students. This is not a task involved in solving a problem with dynprog per se.

Other examples of difficult course content is likely the very aim of many of the tasks – "only" classification of problems. The context switch from problem solving to problem classification may be veiled behind the many similarities between tasks in computational complexity and algorithm construction, respectively. Still one of the included papers, Paper IV, in this dissertation attempted to *increase* the similarities between these two topics. Emphasizing the similarities was tried in order to help students feel at ease with the more mathematical parts. It worked relatively well in our setting, where students seem to benefit more from participating in the activities introduced in order to enhance the learning of complexity, than in activities targeting dynprog – the algorithmic topic where difficulties have been observed. Of course, there is not proof of casual relationships between these observations.

The second theme was: **What do our teaching approaches and activities (among other things, automated assessment of programming exercises) convey about the nature of TCS, to the students?** The aspects of this that this thesis deal with are:

1. How do students respond to changes in the courses regarding activities?
2. What does anyone need to master to succeed in the ADC course?
3. Are we, as teachers, unveiling the perspectives we are part of, explicitly?

Conclusions from the included articles include that automated assessment is working well, and is appreciated by the students for its availability and immediate feedback, but is sometimes causing associations to other contexts with other quality criteria than the ADC course. It certainly makes students aware that teachers care about program correctness, but is unlikely to signal to all students that ascertaining correctness is *their* responsibility and not the teachers' or the assessment system's, at least without tasks designed with this awareness as an intended learning outcome. Using automated assessment for theory assignments, like the complexity lab, makes the students spend time on that assignment, but does not foster the attitude that this is time well spent although there are some indications that this might be the case. The likely reason for this is that the automated assessment system comes with other ideas on standards, quality and purpose than the theory exercise, and students may focus on similarities to other programming assignments and never notice the differences.

Using self-efficacy surveys to collect information on what many students were more or less comfortable with, as done in [20, 18, 17], can be used to generate hypotheses about what material to investigate or analyze further. The attempts to find correlations between the used self-efficacy items and students' performance on the ADC course were not successful, either due to the items themselves not being well phrased, or due to the assessed tasks in the course not targeting specific self-efficacy item capabilities exclusively. Time passing by between tests and surveys might also play a part here. More systematic evaluation of the self-efficacy items is required if the increased self-efficacy for these items should be taken as an indication of learning. If a correlation can be established, increased self-efficacy is a positive indication in itself and may be used for evaluating teaching experiments.

The approaches of this thesis were motivated by an interest in the subject of theoretical computer science and its possible inherent difficulties. Difficulties are, according to the previous discussion, often to be understood as a mismatch of expectations and context of the teaching and learning situation in the eyes of teachers versus in the eyes of students. *Difficult concepts* appear to be reductions in general and reductions in NP-completeness proofs in particular. There are also indications that it is difficult to solve entire tasks with no help, meeting the requirements of the TCS practice, and particularly in what the concept of a proof is in TCS, why it is there, and what constitutes a good proof. Some qualitative analysis suggests that also the language of TCS presents a difficulty in itself, that is never explicitly addressed.

Bibliography

[1] John R. Anderson, Lynne M. Reder, Herbert A. Simon, K. Anders Ericsson, and Robert Glaser. Radical constructivism and cognitive psychology. *Brookings Papers on Education Policy*, (1):pp. 227–278, 1998. ISSN 10962719. URL http://www.jstor.org/stable/20067198.

[2] Joel André and Susanna Salmijärvi. Det sociokulturella perspektivet. Master's thesis, Göteborgs universitet, 2009.

[3] Michal Armoni. Reductive thinking in a quantitative perspective: the case of the algorithm course. In *ITiCSE '08: Proceedings of the 13th annual conference on Innovation and technology in computer science education*, pages 53–57, New York, NY, USA, 2008. ACM. ISBN 978-1-60558-078-4. URL http://doi.acm.org/10.1145/1384271.1384288.

[4] Michal Armoni. Reduction in CS: A (mostly) quantitative analysis of reductive solutions to algorithmic problems. *J. Educ. Resour. Comput.*, 8(4): 11:1–11:30, January 2009. ISSN 1531-4278. URL http://doi.acm.org/10.1145/1482348.1482350.

[5] Michal Armoni, Judith Gal-Ezer, and Orit Hazzan. Reductive thinking in undergraduate CS courses. In *ITICSE '06: Proc. 11th ann. SIGCSE conf. on Innovation and technology in Comp. Sci. education*, pages 133–137, New York, NY, USA, 2006. ACM. ISBN 1-59593-055-8.

[6] Michal Armoni and David Ginat. Reversing: a fundamental idea in computer science. *Computer Science Education*, 18(3):213–230, 2008. URL http://www.tandfonline.com/doi/abs/10.1080/08993400802332670.

[7] Caroline Baillie, John A. Bowden, and Jan H. F. Meyer. Threshold capabilities: threshold concepts and knowledge capability linked through variation theory. *Higher Education*, 65(2):227–246, 2013. ISSN 0018-1560. URL http://dx.doi.org/10.1007/s10734-012-9540-5.

[8] Nicolas Balacheff. Aspects of proof in pupils' practice of school mathematics. *Mathematics, teachers and children*, pages 216–235, 1988.

[9] Olle Bälter, Emma Enström, and Bernhard Klingenberg. The effect of short formative diagnostic web quizzes with minimal feedback. *Computers & Education*, 60(1):234 – 242, 2013. ISSN 0360-1315. URL `http://www.sciencedirect.com/science/article/pii/S0360131512002047`.

[10] Albert Bandura. *Social foundations of thought and action: A social cognitive theory.* Prentice-Hall series in social learning theory. Prentice-Hall, Englewood Cliffs, New Jersey, 1986.

[11] J. Biggs and C. Tang. *Teaching for Quality Learning.* SRHE and Open University Press imprint. McGraw-Hill Companies, Incorporated, 2007. ISBN 9780335221271. URL `http://books.google.se/books?id=9m9TBX4cWooC`.

[12] Benjamin S. Bloom, Max D. Engelhart, Edward J. Furst, Walker H. Hill, and David R. Kratwohl. *Taxonomy of educational objectives Handbook 1: cognitive domain.* Longman Group Ltd., London, 1956.

[13] John Bowden and Ference Marton. *The University of Learning: Beyond Quality and Competence in Higher Education.* Kogan Page, 1998. ISBN 0 7494 2292 0. URL `http://books.google.se/books?id=nuJZxw_UoKIC`.

[14] D. M. Bunce, E. A. Flens, and K. Y. Neiles. How long can students pay attention in class? A study of student attention decline using clickers. *Journal of Chemical Education*, 87:1438–1443, December 2010.

[15] Thomas J. Bussey, MaryKay Orgill, and Kent J. Crippen. Variation theory: A theory of learning and a useful theoretical framework for chemical education research. *Chem. Educ. Res. Pract.*, 14:9–22, 2013. URL `http://dx.doi.org/10.1039/C2RP20145C`.

[16] Daniel Chazan. High school geometry students' justification for their views of empirical evidence and mathematical proof. *Educational Studies in Mathematics*, 24(4):pp. 359–387, 1993. ISSN 00131954. URL `http://www.jstor.org/stable/3482650`.

[17] Pierluigi Crescenzi, Emma Enström, and Viggo Kann. From theory to practice: NP-completeness for every CS student. In *ITiCSE '13: Proceedings of the eighteenth annual conference on Innovation and technology in computer science education*, pages 16–21, New York, NY, USA, 2013. ACM. URL `http://doi.acm.org/10.1145/2462476.2465582`.

[18] Emma Enström. Dynamic programming - structure, difficulties and teaching. In *2013 Frontiers in Education Conference (FIE 2013)*, Oklahoma City, USA, October 2013. ©2013 IEEE. Reprinted with permission.

[19] Emma Enström and Viggo Kann. Computer lab work on theory. In *ITiCSE '10: Proceedings of the fifteenth annual conference on Innovation and technology in computer science education*, pages 93–97, New York, NY, USA,

2010. ACM. ISBN 978-1-60558-729-5. URL `http://doi.acm.org/10.1145/1822090.1822118`.

[20] Emma Enström and Viggo Kann. *Iteratively intervening with the "most difficult" topics fo an algorithms and complexity course.* In submission, 2014.

[21] Emma Enström, Gunnar Kreitz, Fredrik Niemelä, Pehr Söderman, and Viggo Kann. Five years with Kattis - using an automated assessment system in teaching. In IEEE, editor, *Proceedings of the 41st ASEE/IEEE Frontiers in Education Conference, Rapid City, SD*, Frontiers in Education. IEEE, ISBN: 978-1-61284-467-1, 2011. ©2011 IEEE. Reprinted with permission.

[22] Efraim Fischbein. Intuition and proof. *For the learning of mathematics*, pages 9–24, 1982.

[23] G. Gibbs and C. Simpson. Conditions under which assessment supports students' learning. *Learning and teaching in higher education*, 1(1):3–31, 2004.

[24] Lakshmi Goel, Norman Johnson, Iris Junglas, and Blake Ives. Situated learning: Conceptualization and measurement. *Decision Sciences Journal of Innovative Education*, 8(1):215–240, 2010. ISSN 1540-4609. URL `http://dx.doi.org/10.1111/j.1540-4609.2009.00252.x`.

[25] Gila Hanna. Proof, explanation and exploration: An overview. *Educational Studies in Mathematics*, 44(1-2):5–23, 2000. ISSN 0013-1954. URL `http://dx.doi.org/10.1023/A%3A1012737223465`.

[26] Jeremy Hodgen, Dietmar Küchenmann, Margaret Brown, and Robert Coe. Childrens understandings of algebra 30 years on. In M. Joubert, editor, *Proceedings of the British Society for Research into Learning Mathematics*, volume 28, November 2008.

[27] Håkan Hult. Laborationen – myt och verklighet. CUP:s rapportserie 6, Linköpings universitet, 2000.

[28] Tansy Jessop, Yassein El Hakim, and Graham Gibbs. The whole is greater than the sum of its parts: a large-scale study of students' learning in response to different programme assessment patterns. *Assessment & Evaluation in Higher Education*, 39(1):73–88, 2014. URL `http://dx.doi.org/10.1080/02602938.2013.792108`.

[29] Keith Jones. The student experience of mathematical proof at university level. *International Journal of Mathematical Education in Science and Technology*, 31(1):53–60, 2000. URL `http://dx.doi.org/10.1080/002073900287381`.

[30] Carolyn Kieran. Concepts associated with the equality symbol. *Educational Studies in Mathematics*, (12):317–326, 1981.

[31] Eric J Knuth. Secondary school mathematics teachers' conceptions of proof. *Journal for research in mathematics education*, pages 379–405, 2002.

[32] Eric J Knuth. Teachers' conceptions of proof in the context of secondary school mathematics. *Journal of Mathematics Teacher Education*, 5(1):61–88, 2002.

[33] Yi-Yin Ko and Eric Knuth. Undergraduate mathematics majors' writing performance producing proofs and counterexamples about continuous functions. *The Journal of Mathematical Behavior*, 28(1):68 – 77, 2009. ISSN 0732-3123. URL `http://www.sciencedirect.com/science/article/pii/S0732312309000200`.

[34] Elizabeth Koshy, Valsa Koshy, and Heather Waterman. *Action research in healthcare*. Sage, 2010.

[35] Philip Langer. Situated learning: What ever happened to educational psychology? *Educational Psychology Review*, 21(2):181–192, 2009. ISSN 1040-726X. URL `http://dx.doi.org/10.1007/s10648-009-9099-6`.

[36] Ference Marton and Shirley Booth. *Learning and Awareness (Educational Psychology Series)*. Routledge, 1997. ISBN 0805824553.

[37] Jan Meyer and Ray Land. *Overcoming Barriers to Student Understanding: Threshold concepts and troublesome knowledge*. Routledge, 2006. ISBN 0415374308.

[38] George A. Miller. The magical number seven, plus or minus two: Some limits on our capacity for processing information. *The Psychological Review*, 63(2): 81–97, March 1956. URL `http://www.musanim.com/miller1956/`.

[39] George A Miller. The cognitive revolution: a historical perspective. *Trends in Cognitive Sciences*, 7(3):141 – 144, 2003. ISSN 1364-6613. URL `http://www.sciencedirect.com/science/article/pii/S1364661303000299`.

[40] Orna Muller. Pattern oriented instruction and the enhancement of analogical reasoning. In *Proceedings of the first international workshop on Computing education research*, ICER '05, pages 57–67, New York, NY, USA, 2005. ACM. ISBN 1-59593-043-4. URL `http://doi.acm.org/10.1145/1089786.1089792`.

[41] Maria Pampaka, Irene Kleanthous, Graeme D. Hutcheson, and Geoff Wake. Measuring mathematics self-efficacy as a learning outcome. *Research in Mathematics Education*, 13(2):169–190, 2011. URL `http://dx.doi.org/10.1080/14794802.2011.585828`.

[42] Ivan Pavlov. *Conditioned Reflexes: An investigation of the Physiological Activity of the Cerebral Cortex*, page 142. Oxford University Press, London, 1927.

[43] Jean Piaget. *Piaget and His School*, chapter 1: Piaget's theory. Springer, New York Heidelberg Berlin, 1976.

[44] Naomi Prusak, Rina Hershkowitz, and Baruch B. Schwarz. From visual reasoning to logical necessity through argumentative design. *Educational Studies in Mathematics*, 79(1):19–40, 2012.

[45] Janet Rountree and Nathan Rountree. Issues regarding threshold concepts in computer science. In *Proceedings of the Eleventh Australasian Conference on Computing Education-Volume 95*, pages 139–146. Australian Computer Society, Inc., 2009.

[46] Ulla Runesson. *Variationens pedagogik – skilda sätt att behandla ett matematiskt innehåll*. PhD thesis, Acta Universitatis Gothoburgensis, Göteborg, 1999. ISBN 91-7346-344-2.

[47] R. Säljö. *Lärande i praktiken: ett sociokulturellt perspektiv*. Norstedts akademiska förlag, 2005. ISBN 9789172274365.

[48] Andreas Schwill. Fundamental ideas of computer science. *Bulletin European Association for Theoretical Computer Science*, (53):274–274, 1994.

[49] B. F. Skinner. Review lecture: The technology of teaching. *Proceedings of the Royal Society of London. Series B, Biological Sciences*, 162(989):pp. 427–443, 1965. ISSN 00804649. URL `http://www.jstor.org/stable/75554`.

[50] Andreas J Stylianides. Proof and proving in school mathematics. *Journal for Research in Mathematics Education*, pages 289–321, 2007.

[51] Andreas J. Stylianides and Thabit Al-Murani. Can a proof and a counterexample coexist? Students' conceptions about the relationship between proof and refutation. *Research in Mathematics Education*, 12(1):21–36, 2010. URL `http://dx.doi.org/10.1080/14794800903569774`.

[52] John Sweller, Jeroen J.G. van Merrienboer, and Fred G.W.C. Paas. Cognitive architecture and instructional design. *Educational Psychology Review*, 10 (3):251–296, 1998. ISSN 1040-726X. URL `http://dx.doi.org/10.1023/A%3A1022193728205`.

[53] Thomas Varghese. Possible student justification of proofs. *School Science and Mathematics*, 111(8):409–415, 2011. ISSN 1949-8594. URL `http://dx.doi.org/10.1111/j.1949-8594.2011.00106.x`.

[54] Guy Walker. A cognitive approach to threshold concepts. *Higher Education*, 65(2):247–263, 2013.

[55] J.V. Wertsch. *Voices of the Mind: Sociocultural Approach to Mediated Action*. Harvard University Press, 2009. ISBN 9780674045101. URL `http://books.google.se/books?id=9EtTuaPMtjAC`.

[56] Karen Wilson and James H. Korn. Attention during lectures: Beyond ten minutes. *Teaching of Psychology*, 34(2):85–89, 2007. URL `http://dx.doi.org/10.1080/00986280701291291`.

[57] Jessica M. Zerr and Ryan J. Zerr. Learning from their mistakes: Using students' incorrect proofs as a pedagogical tool. *PRIMUS*, 21(6):530–544, 2011. URL `http://dx.doi.org/10.1080/10511970903386915`.