Ralph-Johan Back

Structured Derivations as a Unified Proof Style for Teaching Mathematics

Turku Centre *for* Computer Science

TUCS Technical Report
No 949, July 2009

Structured Derivations as a Unified Proof Style for Teaching Mathematics

Ralph-Johan Back
Abo Akademi University, Department of Information Technologies

TUCS Technical Report
No 949, July 2009

Abstract

Structured derivations were introduced by Back and von Wright as an extension of the calculational proof style originally proposed by E.W. Dijkstra and his colleagues. Structured derivations added nested subderivations and inherited assumptions to the original calculational style. This paper introduces a further extension of the structured derivation format, and gives a precise syntax and semantics for the extended proof style. The extension provides a unification of the tree main proof styles used in mathematics today: Hilbert-style forward chaining proofs, Gentzen-style backward chaining proofs and algebraic derivations and calculations (in particular, Dijkstra's calculational proof style). Each of these proof styles can be directly modelled as an extended structured derivation. Even more importantly, the three proof styles can be freely intermixed in a single structured derivation, allowing different proof styles to be used in different parts of the derivation, each time choosing the proof style that is most suitable for the (sub)problem at hand.

We describe here (extended) structured derivations, feature by feature, and illustrate each feature with examples. We show how to model the three main proof styles as structured derivations. We give an exact syntax for this proof style and define the semantics of structured derivations. The latter is done by showing how each structured derivation can be automatically translated into an equivalent Gentzen-style derivation.

Structured derivations have been primarily developed for teaching mathematics on secondary and tertiary education level. The syntax of structured derivations determines the general structure of the proof, but does not impose any restrictions on how the basic constructs of the underlying mathematical domain are expressed. Hence, the style can be used for any kind of mathematical proofs, calculations, derivations, and general problem solving found in mathematics education at these levels. The exact syntax makes it easy to provide computer support for structured derivations.

Keywords: Teaching mathematics, logical reasoning, proof styles, Gentzen, Hilbert, Dijkstra, calculational proofs, structured derivations, high school mathematics

TUCS Laboratory
Learning and Reasoning Laboratory

1 Introduction

Mathematics is based on proofs. The proof shows the logical reasoning behind a theorem, allows us to understand the meaning of it, its limitations and its consequences. Without a proof, a theorem is like magic, with a proof it is (sometimes more, sometimes less) self evident. But proofs are considered difficult in mathematics education at the secondary level and are therefore usually avoided. When proofs are given, they are often informal and the underlying logic is not explicated. Logical notation is used to some extent in high school mathematics, but reasoning with logical expressions is treated very cursorily, or not at all. When logic is taught, it is often seen as a separate object of study (*mathematical logic)*, rather than as a useful tool for solving mathematical problems (*logical mathematics)*.

We describe here an alternative approach to teaching mathematics that is based on a systematic format for writing proofs and derivations and the explicit use of logical notation and logical inference rules in these proofs. Our starting point is the *calculational proof style* originally developed by E.W. Dijkstra and his collegues (Wim Feijen, Nettie van Gasteren, and Carel Scholten) [7, 23] (see [8] for a nice overview of the approach). They present a proof in a fixed format, as a sequence of calculation steps, with an explicit justification for each step, and they emhasize the use of formal logical notation and reasoning in the calculations. A mathematical assertion is proved by calculating its truth value, step by step.

Back and von Wright extended the calculational proof style to *structured derivations,* by adding nested derivations and a mechanism for handling and propagating assumptions in derivations [4, 1]. Structured derivations were originally developed in order to present proofs in programming logic (the refinement calculus). Later, this proof style was found to be quite useful also for teaching mathematics in school, particularly at the secondary and tertiary education level [5, 2, 17].

The syntax for structured derivations has been gradually modified by our experiences of teaching mathematics using the structured derivation style. The aim has been to make the syntax as intuitive and transparent as possible. We describe here an extension of the original structured derivations style which provides a standard format for different kinds of mathematical arguments, from numerical calculations and algebraic derivations to mathematical proofs and general problem solving. The extension provides a unification of the tree main proof styles used in mathematics today: Hilbert-style forward chaining proofs, Gentzen-style backward chaining proofs and algebraic derivations and calculations (in particular, Dijkstra's calculational proof style). This means that a proof in any of these three proof styles can be directly written as a structured derivations. But even more importantly, these three proof styles can be freely intermixed within a single structured derivation. Different parts of a derivation can be carried out in different proof styles, choosing the proof style that is most suitable for the task at hand.

The extended structured derivations style described here is more or less com-

patible with the original proof style, but there are some differences, in particular on how assumptions are treated. We will refer to the original structured derivation proof style as SDS-1 (*Structured Derivation Style 1*) and to the extension described in this article as SDS-2 when we need to distinguish between these two versions.

The primary application for structured derivations is in teaching mathematics at different educational levels, from junior high school and high school to freshman courses at the university. This means that the derivation style should be easy to read, easy to understand, easy to write and easy to check for errors. The main tool for achieving this is to insist on explicit motivations for each derivation step, in the tradition of Dijkstra's calculational proofs. Writing down explicit motivations for each step in the derivation makes the proof much easier to follow at the time that it is presented, as well as afterwards, when working on assignments or preparing for an exam.

Structured derivations are intended to be used for all kinds of mathematics courses, and at different level of mathematical rigour. Structured derivations fix a specific format (syntax) for how to present mathematical arguments, and gives an exact meaning (semantics) to any proof constructed in this format, by describing how that proof can be translated into an equivalent Gentzen-style proof. But it does not fix the underlying mathematical domain, which can be chosen freely. A derivation uses the standard notation and definitions of the underlying domain, together with the available theorems. The level of rigour at which the argumentation is carried out can vary from informal argumentation to precise and exact mathematical proofs. In the extreme, we can use structured derivations to express completely axiomatic, machine checkable proofs.

Another guiding principle for structured derivations, also inspired by the calculational proof style, is the strong emphasis on using logical notation and explicit logical inference rules in high school mathematics. Logic is everywhere in school mathematics, but it is usually hidden behind informal verbal expressions. Logical notation is used to some extent, but the students are not taught the rules for how to manipulate logical expressions. Hence, the use of logic remains at the level of informal understanding, and the opportunity to speed up reasoning by manipulating logical expressions is missed. This contrasts sharply with the strong emphasis on doing arithmetic and algebraic calculations, where a similar speed up is gained by having a precise notation for arithmetic and algebraic expressions, and routinely applying algebraic rules in derivations.

Structured derivations have been designed with computer support in mind. The syntax is defined precisely, so that it can be parsed by a computer. This makes it possible to check that a proof is syntactically correct. The computer can translate a structured derivation automatically into a standard Gentzen-style proof. This opens up the possibility to use computers to automatically check that a derivation is correct. If a structured derivation is carried out over a completely mechanized domain (say integers and reals in HOL, Isabelle or PVS), then the

Gentzen-style proof that results from the translation can be directly checked by an interactive or automatic proof checker. If the underlying domain is less formal, then some natural language understanding processing may be needed.

Structured derivations are intended to also support the construction of larger proofs, as well as reading, understanding and checking the correctness of such proofs. The format is based on a hierarchical notation of a derivation, where a main derivation can be split up into a number of more detailed subderivations, which in turn can be split up into even more detailed subderivations, and so on. The ideal tool for working with structured derivations is an outlining editor, where the user can selectively show and hide subproofs at different levels of detail.

Structured derivations should be seen as a new for describing mathematical arguments, but it is not a new logical framework. The formalism is ultimately based on Gentzen-style proofs, and axioms and inference rules in Gentzen-style proof frameworks can be used freely in structured derivations. We are also not trying to change the way professional mathematicians carry out and present their mathematical derivations; they have a strong intuitive understanding of the logic behind mathematical derivation, and can quickly grasp the ideas behind a mathematical argument in a traditional form. The structured derivations style may provide an alternative for some professional mathematicians, in the same way as Dijkstra's calculational style has been readily adapted by a number of researchers within the Formal Methods community, but the traditional way of doing mathematics is a fast and efficient way of communication between professionals. We are instead targeting the novices in mathematics, the students who need more detailed and explicit explanations of proofs and derivations, to help them comprehend the material they are being taught with less guesswork and false starts.

Related work The calculational proof style has been adopted quite widely in articles and text books on programming methods, in particular in the context of formal (or logical, mathematical) methods for constructing correct programs, see e.g. the textbook by David Gries and Fred Schneider [12] and by Jan van de Snepscheut [22]. Gries and Schneider have also studied the use of calculational proofs in high school teaching [11, 10, 13], making a strong case for the advantages of this method in practical mathematics education.

There has been some work on making the other proof styles also more attractive in teaching mathematics. Lamport proposed a style for more human readable Gentzen-like proofs, where indentation was used as a structuring device [14]. The Hilbert-like proof style for geometry has been tried in schools using a two-column proof format (see e.g. en.wikibooks.org/wiki/Geometry/Chapter_2). Interactive theorem provers like Isabelle [16] (e.g., the Isar front end [24]), Mizar [20, 21] and PVS [19] have also been equipped with more user friendly front ends for reading and writing proofs. However, these front ends usually target power users, and are not suitable as such for teaching mathematical proofs on secondary education levels.

The original motivation von Wright and I had in developing the structured derivations style was to try to make proofs in programming logic more transparent and easier to read. Structured derivations are used throughout in our book on refinement calculus [4], where we use this style to present a large number of more or less complex proofs in algebra and logic. The method was also presented separately in a journal article [1]. Dijkstra introduced his own logic for calculational proofs, and explained his proof style in terms of Hilbert-style proofs, whereas structured derivations were seen as a variant of Gentzen-style proofs. This gives more support for working with explicit assumptions in derivations. Our basic framework was higher order logic. This framework is originally due to Alonzo Church [6], but we based our approach on a variant of this logic developed by Michael Gordon and Tom Melham [9], used as the basis for the interactive theorem prover HOL.

Contents of article The paper is structured as follows. The next section introduces structured derivations informally, as a style for presenting mathematical proofs, calculations, derivations, and argumentations, feature by feature. We illustrate each feature with a standard mathematics problem from high school. We also give an overview of our experiences from using structured derivations in practice, with pointers to articles where these experiences are explained in more detail. Section 3 discusses the relationship between structured derivations and the main proof styles considered here: calculational style proofs, Hilbert-style proofs and Gentzen-style proofs. Section 4 gives a more precise and detailed overview of the syntax of structured derivations, and discusses different dialects of the style. Section 5 presents a more formal definition of the meaning of structured derivations, by showing how a structured derivation can be translated into a standard Gentzen-style proof. We end with summarizing the main results and discussing some work ahead.

2 Structured derivations

We describe here the main features of structured derivations, one by one. We take the calculational proof style introduced by Dijkstra and his colleagues as the starting point, and show how to extend it to a general proof style that also covers Hilbert like proofs and Gentzen like proofs. We could equally well have taken any of the two other proof styles as the starting point, but we choose to start from the calculational style because this has been the historical development for us. We illustrate the features of structured derivations with examples taken from ordinary upper and lower secondary school mathematics. Our intention is to show how traditional mathematical arguments, used in standard mathematics education, can be expressed in a new way that is clearer and easier to understand .

2.1 Calculations

A calculation is essentially a relation chain of the form

$$term_0\, rel_1\, term_1\, rel_2\, term_2\ \ldots\ term_{k-1}\, rel_k\, term_k$$

where each rel_i, $i = 1, \ldots, k$, is either equality or some specific reflexive transitive relation rel. The chain stands for the proposition

$$term_0\, rel_1\, term_1\ \wedge\ term_1\, rel_2\, term_2\ \wedge \ldots\ \wedge\ term_{k-1}\, rel_k\, term_k$$

From this we conclude by transitivity that

$$term_0\, rel\, term_k$$

A simple example is the following calculation:

$$\begin{array}{lcl} (x+1)(x+2)+ & \geq & (x+1)(x+2) \\ & = & x^2 + x + 2x + 2 \\ & = & x^2 + 3x + 2 \\ & \geq & x^2 + 3x \\ & = & x(x+3) \end{array}$$

where the reflexive transitive relation is $\geq$. This shows that

$$(x+1)(x+2) + 1 \geq x(x+3)$$

Syntax of calculational proofs The general structured derivation syntax for a calculation is as follows (with $k \geq 1$).

derivation ::=	
•	*term*$_0$
rel_1	*justification*$_1$
	term$_1$
	$\vdots$
	term$_{k-1}$
rel_k	*justification*$_k$
	term$_k$
□	

justification ::=
{*motivation*}

We reserve one line for each term in the derivation, and one line for the relation between the terms together with the motivation for why the relation holds between the terms. The motivation is given in curly brackets. More than one line can be used for a term or for a motivation, if needed. This is essentially the calculational proof style originally introduced by Dijkstra. The proof is laid out in two columns, where the first column is reserved for the relation symbols, and the second column is reserved for the terms and the motivations. The start of the derivation is indicated by a bullet and the end of the derivation by an optional square.

Dijkstra combines this basic format with the explicit use of predicate calculus in mathematical arguments. This allows him to give beautiful logical derivations of sometimes quite sophisticated mathematical theorems. He was particularly fond of calculations with equivalence between logical formulas, but also used implication and backward implication quite frequently. The format obviously works for any reflexive transitive relation between any kinds of mathematical entities, like equality or ordering between arithmetic and algebraic expressions, or equality and inclusion ordering between set expressions. The underlying logic assumed by Dijkstra for calculational derivations is first order logic with an underlying Hilbert-style reasoning.[1]

Example 1 We exemplify calculations with a classical problem in high school mathematics: solving arithmetic equations. This example illustrates both the format for calculations and the use of explicit logical notation in mathematical derivations. The problem is to solve the equation

$$x^3 - x^2 + x - 1 = 0$$

The solution is found as follows:

•	$x^3 - x^2 + x - 1 = 0$
$\equiv$	{grouping}
	$x^2(x-1) + (x-1)s = 0$
$\equiv$	{factorization}
	$(x-1)(x^2+1) = 0$
$\equiv$	{zero product rule: $ab = 0 \equiv a = 0 \vee b = 0$}
	$x - 1 = 0 \vee x^2 + 1 = 0$
$\equiv$	{add 1 to both sides in left disjunct and simplify}
	$x = 1 \vee x^2 + 1 = 0$

[1] Check original writing by Dijkstra and by van Gasteren

$\equiv$ {add -1 to both sides in right disjunct and simplify}

$x = 1 \vee x^2 = -1$

$\equiv$ {a square is never negative}

$x = 1 \vee F$

$\equiv$ {disjunction rule: $p \vee F \equiv p$}

$x = 1$

□

The answer is thus that there is exactly one solution to the equation, $x = 1$. Note that the use of equivalence is important here. If we only had forward implication, then we might get too many values for x, and if we only had backward implication, we could get too few values for x.

Discussion The advantages of the calculational proof style is that the derivation steps are very clearly laid out, and that there is ample room for both terms and motivations for each step. The explicit justification of each step is a characteristic property of this proof style, and makes it very easy to follow the derivation, step by step. Checking the correctness of a calculational proof style is local: we only need to check that each relation step is true, for the reasons indicated in the justification for the step. If that is the case, then the overall conclusion holds.

Compare this to how the example derivation would be carried out in a more traditional way on the blackboard. The relationship between the proof steps (equivalence) would probably not be indicated explicitly, and the motivation for each step would be terse and often omitted. The argument would break up into two different subderivations, one for the first term and one for the second term, without explicit indication of how the two cases are related. The use of explicit disjunction in our example allows us to keep the two cases together.

The main perceived disadvantages of the calculational style is that proofs become longer and they take longer to write down and explain. However, in a pedagogical setting this is often an advantage. It is much easier for students to follow the proof as it is constructed step by step on the blackboard in this fashion: the motivation for each step is explicitly written down, and the teacher takes more time in explaining each step (because writing down the motivation takes time).

There is also a definite advantage in forcing the student to write down the motivations for each step when he solves a problem on his own. This helps drive home the lesson that a calculation is a proof, where each step has to be carefully justified, in order to be certain that the result is correct. The explicit justifications also help the teacher to see where the students derivation has gone astray, and what the student has and has not understood.

2.2 Explicit task and assumptions

Dijkstra's calculational style focuses on the calculation or proof itself. The original problem statement is given elsewhere, usually in the surrounding text. Our first extension to the calculational style is to add the problem statement as an explicit part of the derivation: we state the task to be solved (or conjecture to be proved) and list the assumptions that we are allowed to make in the derivation.

Syntax for structured derivation A structured derivation with explicit task and assumptions has the following syntax (here $m \geq 0$ and $k \geq 0$):

derivation ::=	
•	*task*
–	*assumption*$_1$
	⋮
–	*assumption*$_m$
⊩	*justification*
	term$_0$
rel_1	*justification*$_1$
	term$_1$
	⋮
	term$_{k-1}$
rel_k	*justification*$_k$
	term$_k$
□	

The task is given on the first line of the derivation. The task is usually expressed in an imperative way, as something that needs to be done: "Show that ..." , "Compute a such that ...", "Solve equation ...", etc. The task is followed by the assumptions, each assumption on a line of its own. The default symbol for an assumption is a dash.

The start of the derivation is indicated as before with a bullet and the end of the derivation with an optional square. The start of the proof is indicated with the proof symbol "⊩" ("is proved by"). The proof symbol is followed by a justification that explains why the calculation that follows does in fact solve the task. In

many cases this trivial and may be omitted, but sometimes we need to justify this last step more carefully.

This style tries to enforce a systematic method for solving problems. The student reads the problem carefully, and extracts from the problem formulation the specific task that needs to be solved, as well as the assumptions that he or she is allowed to make. This should be done before starting to solve the problem.

Naming tasks and assumptions The default symbols for the task and assumptions are fine in many cases, when we do not want to make the derivation overly formal. However, we can replace the bullet with a task identifier (followed by a colon) when we need to be more specific in our reference (like in "Theorem 1: ...", or "Problem 5a: ...", etc.). Similarly, we may replace the dash with an assumption identifier (in round brackets) when we need to refer to a specific assumptions in a derivation.

We could, of course, have turned the defaults around, and assumed that all tasks and assumptions are uniquely identified by default (e.g., by numbering them explicitly). However, there are usually not that many assumptions in school mathematics problems, so it is quite sufficient to indicate that a specific derivation step follows from the assumptions. Moreover, not having explicit numbering for assumptions makes it easier to change the assumptions (adding, modifying or deleting assumptions) during the derivation, without having to change all references to them. We can still choose to identify some crucial assumption explicitly, while leaving the other assumptions anonymous.

Example 2 Show that

$$(1+a)(1+b)(1+c) \geq 1+a+b+c$$

when a, b and c are non-negative real numbers.

The following derivation starts with the task to be solved (the conjecture that we want to prove), then lists the assumptions that we are allowed to make, and finally proves that the conjecture is indeed correct.

- Show that $(1+a)(1+b)(1+c) \geq 1+a+b+c$

- when $a, b, c \geq 0$

$\Vdash$ {combining $=$ and $\geq$ gives $\geq$, transitivity}

$(1+a)(1+b)(1+c)$

$=$ {multiply two last parenthesis}

$(1+a)(1+b+c+bc)$

$=$ {multiply remaining parenthesis}

$1 + b + c + bc + a + ab + ac + abc$

$\geq$ {expression $ab + ac + bc + abc$ is non-negative by assumption}

$1 + a + b + c$

This example illustrates the use of structured derivations for non-logical relations. The motivation following the proof symbol ("combining $=$ and $\geq$ gives $\geq$") explains why the calculational proof does prove the conjecture. The motivation is trivial in this case, and would usually be omitted.

Calculations as structured derivations We have two different forms of structured derivations, the short calculational derivation and a derivation with explicit task and assumptions. The latter is useful when teaching a more systematic approach to problem solving, and when solving more complex problems. A simple calculation is, however, often sufficient in situations where the task is more or less evident from the context, and there are no assumptions (or the assumptions are given in the preceding text or derivation).

The shorter format is then seen as an abbreviation for the longer format, where the task and the justification after the proof symbol are left implicit. The correspondence is shown below. On the left, we have derivation in the form of a calculation, on the right its translation into a derivation with explicit task and assumptions.

•	$term_0$	stands for	•	Show that $term_0\, rel\, term_k$
rel_1	*justification*$_1$		$\Vdash$	$\{transitivity\}$
	$term_1$			$term_0$
	$\vdots$		rel_1	*justification*$_1$
	$term_{k-1}$			$term_1$
rel_k	*justification*$_k$			$\vdots$
	$term_k$			$term_{k-1}$
$\Box$			rel_k	*justification*$_k$
				$term_k$
			$\Box$	

We assume here that each rel_i is either equality or some specific reflexive transitive relation rel.

2.3 Nested derivations

Dijkstra's calculational style assumes that each derivation step can be motivated with one or more sentences within brackets. A motivation that requires a more detailed argument, or a subproof, has to be given somewhere else. This breaks up a single proof into a number of separate proofs, and the context in which these subproofs are carried out is lost. This is in particular problematic when the subproofs are to be carried out under the same assumptions as in the main proof.

The original form for structured derivations (SDS1,[4]), introduced subderivations as a way of justifying more complex derivation steps in a proof, and also introduced assumptions in calculational proofs (the task to be solved was not, however, explicitly stated). The extended structured derivations style (SDS2) combines explicit tasks and explicit assumptions with nested derivations.

Syntax with nested derivations We generalize the previous syntax for derivations by extending the definition of a justification, as shown below ($l \geq 0$).

justification ::=
{*motivation*}
$derivation_1$
$\vdots$
$derivation_l$

In other words, the justification for a derivation step is a motivation (inside curly brackets) as before, but the motivation may optionally be followed by one or more subderivations. These subderivations establish additional facts that are needed for the motivation. The nested derivations are indented one step to the right, so that they stand out clearly from the main derivation.

Example 3 We want to determine when (i.e., for which values of x) the expression $\frac{x-1}{x^2-1}$ is well-defined.

- Determine when $\frac{x-1}{x^2-1}$ is well-defined

$\Vdash$ $\frac{x-1}{x^2-1}$ is well-defined

$\equiv$ {definedness of rational expressions}

$x^2 - 1 \neq 0$

$\equiv$ {switch to logic notation}

$\neg(x^2 - 1 = 0)$

$\equiv$ {solve equation in brackets}

- $x^2 - 1 = 0$

$\equiv$ {factorization rule}

$(x + 1)(x - 1) = 0$

$\equiv$ {rule for zero product}

$x = -1 \vee x = 1$

$\ldots$ $\neg(x = -1 \vee x = 1)$

$\equiv$ {de Morgans laws}

$\neg(x = -1) \wedge \neg(x = 1)$

$\equiv$ {change notation}

$x \neq -1 \wedge x \neq 1$

$\square$

The outer derivation uses the long format, with explicit task, whereas the nested derivation is just a short calculation.

We write "$\ldots$" at the start of the first term following the subderivation, to indicate where the main derivation continues. We may also have an end of proof symbol after a subderivation, but the "$\ldots$" notation seems to be more useful (and concise, it saves one line).

Selectively hiding and revealing subderivations The use of nested derivations allows much more complex proofs to be handled in a single derivation. The proof is structured at different levels of detail: the overall proof idea is shown on the outermost level, while nested subderivations show more and more of the details needed to understand the proof.

An outlining text editor with a capability to selectively hide and show the nested subderivations will allow a reader to see the the derivation at different levels of detail. A derivation step with one or more subderivations might be hidden at first reading, when one is trying to get an overall understanding of the proof. The subderivation can later be shown and checked in more details, if needed. Similarly, we may write a derivation so that at first stage, we just indicate the major proof steps, and later fill in the details of the proof as subderivations. Structured derivations support both this kind of top-down proof construction and the dual bottom-up proof construction (described later).

The above derivation looks as follows when the subderivation is hidden.

- Determine when $\frac{x-1}{x^2-1}$ is well-defined

$\Vdash$ $\frac{x-1}{x^2-1}$ is well-defined

$\equiv$ {definedness of rational expressions}

$x^2 - 1 \neq 0$

$\equiv$ {switch to logic notation}

$\neg(x^2 - 1 = 0)$

$\equiv$ {solve equation in brackets}

$\ldots$ $\neg(x = -1 \vee x = 1)$

$\equiv$ {de Morgans laws}

$\neg(x = -1) \wedge \neg(x = 1)$

$\equiv$ {change notation}

$x \neq -1 \wedge x \neq 1$

$\Box$

The three dots indicates that the subderivation for this step is hidden.

2.4 Inherited assumptions

The proof symbol "$\Vdash$" indicates that the preceding assumptions are implicitly available in all subderivations (in addition to the assumptions that are explicitly listed in the subderivation). This is the preferred way to treat assumptions in structured derivations. In case we want to have more control over the assumptions, we can use the proof symbol $\Vdash_0$ in stead of the default symbol $\Vdash$. This prevents the assumptions of the outer level from being inherited by the subderivations. Any assumption we want to make in the subderivation has then to be explicitly listed, including assumptions from the outer level that we want to reuse.

The following example, taken from analytic geometry, illustrates inheritance of assumptions and subderivations with additional assumption, as well as how to reason about quantifiers in structured derivations.

This derivation also illustrates a technique for calculating the witness needed to establish an existential term in a subderivation. We write the assumption in the subderivation below in the form $p \,\#\#\, q$. The symbol $\#\#$ should be seen as a cross-out symbol: initially, we make the assumption p in the proof, but when we learn more, we cross out p and replace it with the assumption q.

Example 4 The two vectors $\bar{a}$ and $\bar{b}$ ($\bar{a} \neq 0$, $\bar{b} \neq 0$) satisfy the equation $3(\bar{a} - \bar{b}) = -4\bar{a} + 3\bar{b}$. Show that the vectors are parallel.

- Show that vectors $\bar{a}$ and $\bar{b}$ are parallel, when

(1) $\bar{a} \neq 0$ and $\bar{b} \neq 0$, and

(2) $3(\bar{a} - \bar{b}) = -4\bar{a} + 3\bar{b}$

$\Vdash$ Vectors $\bar{a}$ and $\bar{b}$ are parallel

$\equiv$ {assumption (1), definition of two vectors being parallel}

$\left(\exists r > 0 \cdot \bar{a} = r\bar{b}\right)$

$\equiv$ {witness rule, there is a value for r that satisfies this condition}

- Show that $r > 0$ and $\bar{a} = r\bar{b}$

(3) when $r =?$ ## $r = \frac{6}{7}$

$\Vdash$ T

$\equiv$ {assumption (2)}

$3(\bar{a} - \bar{b}) = -4\bar{a} + 3\bar{b}$

$\equiv$ {multiply on the left}

$3\bar{a} - 3\bar{b} = -4\bar{a} + 3\bar{b}$

$\equiv$ {add $4\bar{a} + 3\bar{b}$ to both sides}

$7\bar{a} = 6\bar{b}$

$\equiv$ {divide by 7}

$\bar{a} = \frac{6}{7}\bar{b}$

$\equiv$ {make assumption (3), $r = \frac{6}{7}$}

$\bar{a} = r\bar{b}$

$\Rightarrow$ {$r = \frac{6}{7} > 0$}

$r > 0$ and $\bar{a} = r\bar{b}$

$\ldots$ T

The example shows that the assumptions on the outer level are available in the subderivation. It also shows an example where the subderivation can introduce new assumptions. The witness rule says that that $(\exists x \cdot p(x))$ is true if we can find a value x_0 for x such that $p(x_0)$ is true. The subderivation starts off by wanting to prove that $r > 0$ and $\bar{a} = r\bar{b}$ holds for some yet unknown value r (indicated by first writing $r =?$). Later in the derivation we find out that this condition will be true if we choose $r = \frac{6}{7}$. We then replace the original assumption $r =?$ with the assumption $r = \frac{6}{7}$, to make the subderivation correct. This is indicated in assumption (3) by writing it as $r =?$ ## $r = \frac{6}{7}$.

Changing text in proofs The symbol $\#$ is used to indicate replacement: $p \,\#\#\, q$ means that we replace proposition p with proposition q. Think of this as a purely textual edit, similar to crossing out some text p and writing the correct version q next to it. We can have a number of changed text like this: $p_1 \,\#\#\, p_2 \,\#\#\, \ldots \,\#\#\, p_n$. Only the last version p_n is taken into account when checking that the proof is correct, the previous ones are ignored.

We use the $\#\#$ symbol to indicate that we change some assumption while constructing the proof. In most cases, one should just overwrite an erroneous assumption with the correct one, and proceed with the proof. However, in some cases, it is important to indicate for the reader how the proof was constructed. The example above illustrates one quite common situation. We could present the proof linearly, guessing directly in the subderivation that $r = \frac{6}{7}$ is a good value for the existentially quantified variable. However, this would not be pedagogically very wise, as it teaches the student that the only way to prove an existential formula is by making very smart and insightful guesses. In stead, we prefer to show that a better way is to proceed with the derivation with r as yet unknown ($r =?$), and try to calculate a suitable value for r. When we find a good value for r, then we replace the original assumption with a new assumption that gives this value for r.

Assuming $r = \frac{6}{7}$ directly at the start of the derivation makes it look like the person writing the proof had some divine insight into the problem, so that he or she could directly see that this is the right choice. An ordinary reader would not necessarily be able to see how this value was figured out, and is lost at this point. Revealing the trick makes the proof easier to read and understand, and also helps the reader in constructing similar proofs in the future.

2.5 Choosing the level of rigour

The examples above have all been quite simple. This might give the impression that the structured derivations are intended to be used only in rather mechanical calculations on simple domains of discourse (like solving equations and inequations). This is not the case, rather the opposite. The syntax is designed so that it in principle can be used for any domain of discourse, and at any level of precision. The way we write mathematical terms, as well as how we motivate the derivation steps, can be chosen freely to fit the problem at hand and the level of rigour desired.

We exemplify this with the following problem, taken from analysis, where the argumentation is carried out with larger steps than in the previous examples.

Example 5 We want to determine the values of the constant a such that the function $f(x) = -x^2 + ax + a - 3$ is always negative.

The following structured derivation shows how to solve this problem.

- Determine a such that $(\forall x \cdot f(x) < 0)$

- where $f(x) = -x^2 + ax + a - 3$

$\Vdash$ $(\forall x \cdot f(x) < 0)$

$\equiv$ {the function f is a parabola that opens downwards, as the coefficient for the square term is negative; such a function is always negative if it does not intersect the x- axis}

$(\forall x \cdot f(x) \neq 0)$

$\equiv$ {this is equivalent to the discriminant D_f for the equation being less than zero}

$D_f < 0$

$\equiv$ {determine the discriminant D_f}

- D_f

$=$ {the discriminant for the equation $Ax^2 + Bx + C = 0$ is $B^2 - 4AC$}

$a^2 - 4(-1)(a - 3)$

$=$ {simplify}

$a^2 + 4a - 12$

$\ldots$ $a^2 + 4a - 12 < 0$

$\equiv$ {solve equation $a^2 + 4a - 12 = 0$; the function is a parabola that opens upwards, because the coefficient for a^2 is positive, such a function is negative between the intersection points with the x- axis}

- $a^2 + 4a - 12 = 0$

$\equiv$ {square root formula}

$a = \frac{-4 \pm \sqrt{4^2 - 4 \cdot 1 \cdot (-12)}}{2 \cdot 1}$

$\equiv$ {simplify}

$a = 2 \vee a = -6$

$\ldots$ $-6 < a < 2$

Discussion The example illustrates the use of nested subderivations in a non-trivial way. Both subderivations are necessary, because the computations needed are too complex to be carried out in head. The problem is solved on the ordinary level of discourse used in an analysis class in high school. The motivations are quite detailed, and make use of known facts about parabolas. The first subderivation also recalls the definition for a discriminant, to help the student to understand the substitutions that are done.

2.6 Observations

Solving a problem starts with identifying the task to be solved, as well as the assumptions that we are allowed to make when solving the task. In many situations, we also want to do some preliminary observations, to explore the solution space and fix some basic facts that we think will be useful. We add observations of this kind to structured derivations with the following extension of the syntax. This last extenion give us the most general form of structured derivations.

Syntax with observations Observations are added immediately after the assumptions, and before the proof symbol. Here $m \geq 0$, $n \geq 0$, and $k \geq 0$.

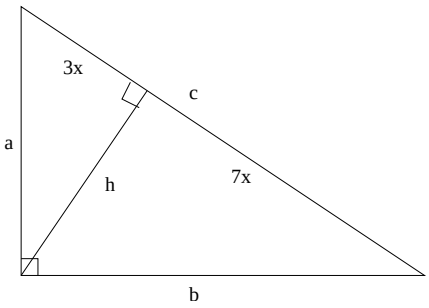
derivation ::=	
•	*task*
-	*assumption*$_1$
	⋮
-	*assumption*$_m$
+	*justification*$_1$
	observation$_1$
	⋮
+	*justification*$_n$
	observation$_n$
⊩	*justification*
	term$_0$
rel_1	*justification*$_1$
	term$_1$
	⋮
	term$_{k-1}$
rel_k	*justification*$_k$
	term$_k$
□	

The idea is that we make one observation after the other. Each observation is justified by explaining why it follows from the assumptions and the preceding observations.

The default identification for an observation is the plus symbol. The plus symbol can be replaced with an observation identifier (within square brackets), when we need to give a specific reference to s the observation. We illustrate observations with the following problem, taken from classical geometry. This example illustrates the use of observations, as well as the use of figures in structured derivations.

Example 6 Consider a right triangle with catheters a and b and hypotenuse c. Assume that the height of the triangle on the hypotenuse divides the hypotenuse in the proportion 3:7. Determine the proportion $\frac{a}{b}$.

We first draw a figure of the situation, with notation that can be referred to in the proof.



The requested proportion is then derived as follows.

- Determine $\frac{a}{b}$, when

- the triangle is right, with hypotenuse c and catheters a and b,

- the height of the triangle on the hypotenuse divides the hypotenuse in the proportion 3:7, and

- h and x are as defined in the figure

[1] {from figure}

$c = 10x$

[2] {figure and Pythagorean theorem}

$h^2 + 9x^2 = a^2$

[3] {figure and Pythagorean theorem}

$h^2 + 49x^2 = b^2$

[4] {figure and Pythagorean theorem}

$a^2 + b^2 = 100x^2$

[5] {subtract equation [2] from equation [3] and simplify}

$b^2 - a^2 = 40x^2$

[6] {add equations [4] and [5] and simplify}

$b^2 = 70x^2$

[7] {substitute equation [6] into equation [4]}

- $a^2 + b^2 = 100x^2$

$\equiv$ {equation [6]}

$a^2 + 70x^2 = 100x^2$

$\equiv$ {solve a^2}

$a^2 = 30x^2$

$\ldots$ $a^2 = 30x^2$

$\Vdash$ $\frac{a}{b}$

$=$ {square root definition, a and b are positive numbers}

$\sqrt{\frac{a^2}{b^2}}$

$=$ {observations [6] and [7]}

$\sqrt{\frac{30x^2}{70x^2}}$

$=$ {simplify}

$\frac{\sqrt{3}}{\sqrt{7}}$

$\square$

Discussion The example illustrates the classical way of solving a geometry problem, by making observations inspired by the figure, and finally combining them into a solution for the problem. The assumptions are followed by altogether seven observations. The last observation uses a calculational style subderivation as justification. The proof of the main conjecture is also done with a calculational derivation. We have chosen this solution because it illustrates how two very different proof styles, a Hilbert-style forward chaining proof and a calculation are combined in a single structured derivation.

2.7 Reduction proofs

We have previously stated that the justification following the proof symbol can be omitted when it is obvious from the calculation (e.g., when the justification is that the relation used in the calculation is reflexively transitive). The proof then just consists of the calculation. We can also do this the other way around, writing the justification but omitting the calculation in the proof (omitting both justification and calculation is, however, not permitted).

Omitting the calculation part, and instead working just with the justification and its possible subderivations gives us a *backward chaining* (or *reduction)* derivations. Here we start from the given task, and try to reduce it step by step to ever simpler problems, until we get down to problems that are so simple that they can be solved directly.

Example 7 We will exemplify this style of proof with a classical induction proof. The problem is to prove that

$$0 + 1 + 2 + \ldots + n = \frac{n(n+1)}{2}$$

The proof is as follows.

- Show that $(\forall n \in \mathbb{N} \bullet 0 + 1 + \ldots + n = \frac{n(n+1)}{2})$

$\Vdash$ {induction proof}

Base step: $0 + 1 + \ldots + n = \frac{n(n+1)}{2}$

- when $n = 0$

$\Vdash$ $0 + 1 + \ldots + n = \frac{n(n+1)}{2}$

$\equiv$ {assumption $n = 0$, definition of sum }

$0 = \frac{0(0+1)}{2}$

$\equiv$ {simplification}

T

Induction step: $0 + 1 + \ldots + n' = \frac{n'(n'+1)}{2}$

- when $n' = n + 1$ and
- $0 + 1 + \ldots + n = \frac{n(n+1)}{2}$

$\Vdash$ $0 + 1 + \ldots + n'$

$=$ {assumption }

$0 + 1 + \ldots + n + (n + 1)$

$=$ {induction assumption}

$\frac{n(n+1)}{2} + (n + 1)$

$=$ {write with common denominator}

$\frac{n^2+n+2n+2}{2}$

$=$ {simplify}

$\frac{n^2+3n+2}{2}$

$=$ {factorize}

$\frac{(n+1)(n+2)}{2}$

$=$ {assumption $n' = n + 1$}

$\frac{n'(n'+1)}{2}$

□

Here the problem is immediately reduced to proving the two cases in an induction proof: the base case and the induction step. These two subproblems are again established with calculational derivations. The base case is proved using equivalences between the equations, while the induction step is proved using equality between terms. We have identified the two subderivations explicitly, as "basic step" and "induction step", to clearly identify the two proof obligations in induction proofs. This example also illustrates how to combine different proof styles in a single structured derivation. In this case, we combine a Gentzen-style backward chaining proof with calculations for solving the subproofs.

2.8 A final example

The examples that we have given above are all taken from a standard high school mathematics curriculum. In fact, most examples are taken from the collection of Finnish Matriculation Exam questions for Upper Secondary Level Mathematics (advanced level). We will end here with a slightly more advanced example, taken from a university level introductory course on analysis.

The example is to show that a given function is continuous in a given point. We recall the definition of continuity. Consider two sets of real numbers, I and

D, and a function $f : I \to D$. The function f is continuous in point $c \in I$, if for all $\epsilon > 0$ there exists a $\delta > 0$ such that for all $x \in I$,

$$|x - c| < \delta \Rightarrow |f(x) - f(c)| < \epsilon$$

This is the epsilon-delta definition of continuity, originally proposed by Bolzano and Cauchy. We express this definition as a single logical formula:

$$(\forall \epsilon > 0 \cdot \exists \delta > 0 \cdot \forall x \geq 0 \cdot |x - x_0| < \delta \Rightarrow |f(x) - f(x_0)| < \epsilon)$$

This formulation shows clearly the reason why the epsilon-delta method presents so much difficulty to beginning math students. The formulation has three alternating quantifiers, when for most people already two quantifiers are quite tough to master conceptually.

Example 8 Show that the function $f(x) = \sqrt{x}$ is continuous for $x_0 > 0$.

The proof is as follows.

- Show that f is continuous in x_0, when

(1) $f(x) = \sqrt{x}$, and

(2) $x_0 > 0$

$\Vdash$ f is continuous in x_0

$\equiv$ {definition of continuity}

$(\forall \epsilon > 0 \cdot \exists \delta > 0 \cdot \forall x \geq 0 \cdot |x - x_0| < \delta \Rightarrow |f(x) - f(x_0)| < \epsilon)$

$\equiv$ {prove for arbitrary ϵ (generalization rule)}

- Show that $(\exists \delta > 0 \cdot \forall x \geq 0 \cdot |x - x_0| < \delta \Rightarrow |f(x) - f(x_0)| < \epsilon)$, when

- $\epsilon > 0$

$\Vdash$ {find a suitable value for δ (witness rule)}

- Show that $\delta > 0 \wedge (\forall x \geq 0 \cdot |x - x_0| < \delta \Rightarrow |f(x) - f(x_0)| < \epsilon)$,

(3) when $\delta =? \; \# \# \; \delta = \epsilon \cdot \sqrt{x_0}$

$\Vdash$ {conjunction rule}

- Show that $\delta > 0$

$\Vdash$ δ

$=$ {assumption}

$\epsilon \cdot \sqrt{x_0}$

$>$ {$\epsilon > 0$ by assumption, $x_0 > 0$ by assumption (2)}

0

- Show that $(\forall x \geq 0 \cdot |x - x_0| < \delta \Rightarrow |f(x) - f(x_0)| < \epsilon)$

$\Vdash$ {show for arbitrary x (generalization rule)}

- Show that $|f(x) - f(x_0)| < \epsilon$, when

- $x \geq 0$, and

- $|x - x_0| < \delta$

$\Vdash$ $|f(x) - f(x_0)|$

$=$ {assumption (1), $f(x)$ and $f(x_0)$ defined, by assumption $x \geq 0$ and (2)}
$\left|\sqrt{x} - \sqrt{x_0}\right|$

$=$ {extend with conjugate value for $\sqrt{x} - \sqrt{x_0}$, i.e., with $\sqrt{x} + \sqrt{x_0} > 0$}
$$\frac{|x - x_0|}{\sqrt{x} + \sqrt{x_0}}$$

$\leq$ {make divisor smaller, $\sqrt{x} \geq 0$}
$$\frac{|x - x_0|}{\sqrt{x_0}}$$

$<$ {assumption $|x - x_0| < \delta$}
$$\frac{\delta}{\sqrt{x_0}}$$

$=$ {choose $\delta = \epsilon \cdot \sqrt{x_0}$ in (3)}
ϵ

$\ldots \quad T$

Discussion This example illustrates the way that assumptions are inherited by subderivations. It also shows how to use basic inference rules for quantifiers. We have to deal with three alternating quantifiers, so the argumentation looks quite complex. However, the proof is actually not that complicated, we just need to apply the obvious quantifier rules to handle the nested quantifiers. The main calculation is carried out on the innermost level. The value of δ is determined at the innermost level, and then substituted for the question mark in the outer assumption. The value of δ is chosen such that the innermost derivation is correct and that the condition $\delta > 0$ is satisfied.

The structured derivation shown here combines Gentzen like deductions with Dijkstra like calculations. The outermost and innermost derivations are in calculational style, while the intermediate derivations are in Gentzen style. The proof has five levels of nesting: the outermost level states the problem, the following three levels correspond to the three nested quantifiers, and the innermost level is used to carry out the actual calculation.

2.9 Structured derivations in practice

We finally describe some of the experiences that we have from using structured derivations in practice. Our work here can be divided into three main categories: experience of using structured derivations in practice when solving mathematical problems (mostly on high school level), experiences that we have gained from teaching mathematics with this method in schools and universities, and experiences that we have gained from building computer support for the method.

Pragmatics Structured derivations are intended to support a careful and systematic way of solving mathematical problems. We start by analysing the given task: identify the problem to be solved, and identify and list the assumptions that we are allowed to make. Auxiliary figures are drawn when needed, and are annotated so that we can refer to them in assumptions and in derivation steps. We may also recall central definitions, theorems and lemmas that we will use in the proof.

We continue by making observations that seem to be relevant to the problem being solved. It does not matter if these observations are not used in the end, superfluous observations do not make the proof erroneous, and they can be deleted once we have solved the problem.

After laying the ground work, we start with the proof itself. The preferred way is to use backward reasoning, i.e., we start from a logical formulation of the problem and try to reduce it step by step to simpler problems, until the original problem is solved. In a reduction proof, we use basic inference rules in a Gentzen-like proof system. In a calculational proof, we reduce the original problem in a sequence of calculation steps to a form that shows the solution explicitly.

We can try to first work out the overall proof structure, checking the details later. But in many cases this may be too difficult, because the way we structure the proof may crucially depend on some key results that we first need to check out. In those cases, we can try to make the key results into observations, before deciding on the overall proof strategy. Or, we just proceed step by step, checking each step carefully.

We use subderivations to work out the details of some proof step and observations as a simple lemma mechanism. Results that are needed in two or more places in the proof should be made into observations. Otherwise, we have to repeat the same argumentation in different subderivations. Results that are needed only once should again be done as subderivations, in order not to clutter up the overall proof structure (solving equations, calculating results, etc).

We emphasize the use of logical notation throughout. It makes the formulas simpler to understand and reason about, and avoids logical errors. Derivations steps should be motivated with explicit logical inference rules (or sequence of rules), whenever possible.

Simpler mathematical problems can be solved directly in a single derivation. More complex problems may require a sequence of derivations leading up to the

required result. The last derivation in this sequence solves the original problem, whereas the preceding derivations are lemmas. Using a sequence of derivations provides a more general lemma mechanism than observations. Observations cannot have their own assumptions, whereas each derivation in a sequence can come with its own notation and assumptions.

Solving mathematical problems and programming Writing down a solution to a mathematical problem as a structured derivation becomes much easier if we use an editor that supports mathematical text and outlining. It is then easy to change, add, and delete derivation steps, task descriptions, assumptions, or observations in the different parts of the proof. The analogue between constructing a mathematical proof and constructing a computer program then also becomes much stronger. Rather than considering the construction of a proof as a process that proceeds linearly from start to end, we see a structured derivation as the end product of the proof process. The proof process proceeds by successive iterations of the structured derivation, which is repeatedly extended, modified and shortened, until it is deemed correct. We can work at different parts of the derivation in arbitrary order, improving and changing these parts as well as the overall structure of the derivation, until we are satisfied that the derivation as a whole is correct.

Empirical studies We have done quite a lot of experiments and empirical studies of structured derivations in high school mathematics. The method has been extended and refined during the last eight years in a close feedback loop with experiments where the method has been tested in practice.

We started experimenting with structured derivations by testing the method on a large sample of standard exercises in high school mathematics. The examples were all taken from the Finnish Matriculation Exam in Mathematics [3]. This provides an unbiased collection of examples that all Finnish students are expected to master after they finish high school. Since then, the method has been tried in a number of smaller and larger empirical studies in Finnish high schools. The most comprehensive study was carried out in 2001 - 2004 (and repeated in 2002 - 2005). The whole mathematics curriculum in a Finnish high school (Kupittaan lukio, Turku) was taught using structured derivations to one group (the test group), while a control group was taught in the standard way. The results of this study were very good, with the test group systematically outperforming the control group in all mathematics courses, as well as in the final matriculation exam [17, 2]. The method has also been used in individual mathematics courses in high school, where we have in particular measured students acceptance and attitudes toward using structured derivations in practice. The results have also here been very encouraging [18, 15].

The Finnish National Board of Education has a program for continuous education of mathematics teachers. Structured derivations are now being actively

taught in this program, as a new way of teaching mathematics, and is gaining support among mathematics teachers at the secondary education level.

Tool support The syntax and layout of structured derivations is defined precisely, and can be easily parsed by a computer. The precise syntax and semantics of structured derivations makes it easy to provide computer support for structured derivations. This includes specialized editors for writing, editing and reading structured derivations, as well as support for using this approach in web-based teaching. We can also use computers to check the correctness of derivations, with the help of mechanized theorem provers. Automatic or semi-automatic checking of student assignments would be a great help to all mathematics teachers, as well as a providing strong support for training and self-studies. Our present work has focused on building enhancements to Lyx that supports structured derivations. Lyx is an open source wysiwyg Latex editor, that directly displays a good approximation of the final mathematical text as we write it. The editor is quite stable, and has a large developer and user base .

3 Combining different proof styles

Structured derivations are an extension of calculational proofs in Dijkstra's style, so obviously proofs of this kind can be expressed as structured derivations. We show here that two other central proof styles, Hilbert-style proofs and Gentzen-style proofs, also can be expressed in a straightforward way as structured derivations.

3.1 Hilbert-style proofs

A Hilbert-style proof is constructed by writing down a sequence of propositions, where each proposition is either an axiom, an assumption, or can be inferred by an inference rule from previous propositions in the sequence. A Hilbert-style proof system for predicate calculus usually has only two inference rules, *modus ponens* and *generalization*, together with a small collection of axioms.

Assume that we want to prove that $proposition$ follows from $proposition_1$, ...,$proposition_m$. A typical Hilbert-style proof of this looks as follows:

1.	$proposition_1$	(assumption)
$\vdots$		
m.	$proposition_m$	(assumption)
$m+1$.	$proposition_{m+1}$	(inference rule$_1$, from ...)
$\vdots$		
$m+k-1$.	$proposition_{m+k-1}$	(inference rule$_{k-1}$, from ...)
$m+k$.	$proposition$	(inference rule$_k$, from ...)

The justification "(inference rule$_{k-i}$, from ...)" stands for either an axiom or the application of an inference rule to some preceding propositions.

Hilbert-style proofs as structured derivations We use observations to model Hilbert-style axiomatic proofs as structured derivations. The proof above, written as a structured derivation, looks as follows:

•	$proposition$
(1)	$proposition_1$
	⋮
(m)	$proposition_m$
$[m+1]$	{inference rule_1, from...}
	$proposition_{m+1}$
	⋮
$[m+k-1]$	{inference rule_{k-1}, from...}
	$proposition_{m+k-1}$
⊩	{inference rule_k, from...}
□	

Note that the task is the last proposition proved in the Hilbert-style proof. The proof itself is here very short, all the work is done by the observations, and there are no calculations in the proof.

A Hilbert-style proof in its purely axiomatic form is seldom used outside the domain of mathematical logic. However, most proofs that are written follow in practice this proof style, in that the proof is seen as a sequence of propositions that are established one by one, where each proposition may depend on the assumptions and on the propositions proved before. This is sometimes known as *forward chaining* derivations: we start from the assumptions and derive more and more consequences, until we finally establish the result that we are looking for. Structured derivations provide a formalization of this general proofs style.

3.2 Gentzen-style proofs

We now show that also Gentzen-style proofs can be expressed as structured derivations. Let us start by defining somewhat more carefully what we mean by a Gentzen-style proof.

A *sequent* is a logical statement of the form $A \vdash con$. Here $A = ass_1, \ldots, ass_m$ (the *antecendent*) is a list of propositions, $m \geq 0$, and con (the *consequent*) is also a proposition. Intuitively, $A \vdash con$ says that con can be proved from the *assumptions* A. If the logic is sound, then this means that con holds whenever all the assumptions in the list A are true. [2]

An inference rule R is of the form

$$\frac{A_1 \vdash con_1, \ldots, A_n \vdash con_n}{A \vdash con}$$

where $n \geq 0$. The sequents above the horizontal line are referred to as the *hypothesis* and the sequent below the horizontal line is referred to as the *conclusion.* Intuitively, a rule of this form says that if we know that all the hypothesis are true, then we also know that the conclusion is true. An inference rule with $n = 0$ (no hypothesis) is an axiom.

A *(Gentzen-like) derivation* has the following general structure.

$$der \; = \; \frac{der_1, \ldots, der_n}{A \vdash con} \{R\}$$

where $n \geq 0$.

We say that der is a *derivation (proof)* of the sequent $A \vdash con$, if

1. der_i is a derivation of some sequent $A_i \vdash con_i$, for $i = 1, \ldots, n$,

2. we have an inference rule R such that

$$\frac{A_1 \vdash con_1, \ldots, A_n \vdash con_n}{A \vdash con}$$

   is an instance of the rule that we get by substituting permitted values for the meta variables in the rule R, and

3. there are only finitely many sequents in the derivation der.

The last clause says that each branch in the derivation tree must eventually end in an axiom, i.e., an application of an inference rule that does not have any hypothesis.

[2] The original formulation by Gentzen assumes that a sequent is of the form $A \vdash C$, where A is as here. but $C = con_1, \ldots, con_k$. Intuitively, this says that one of the conclusions con_i is true whenever all the assumptions $ass_1, \ldots, ass_m$ are true. Our formulation is due to Gordon and Melham [9]. We can obviously express the original formulation in our notation, as $ass_1, \ldots, ass_m \vdash con_1 \vee \ldots, \vee con_k$.

Gentzen-style proofs as structured derivations Let

$$der \;=\; \frac{der_1,\,\ldots,\,der_n}{ass_1,\ldots,\,ass_m \vdash con}\,\{R\}$$

be a Gentzen-style proof. Then this proof can be written equivalently as the following structured derivation:

- con
- ass_1

$\vdots$

- ass_m

$\Vdash_0$ $\{R\}$

der_1

$\vdots$

der_n

$\square$

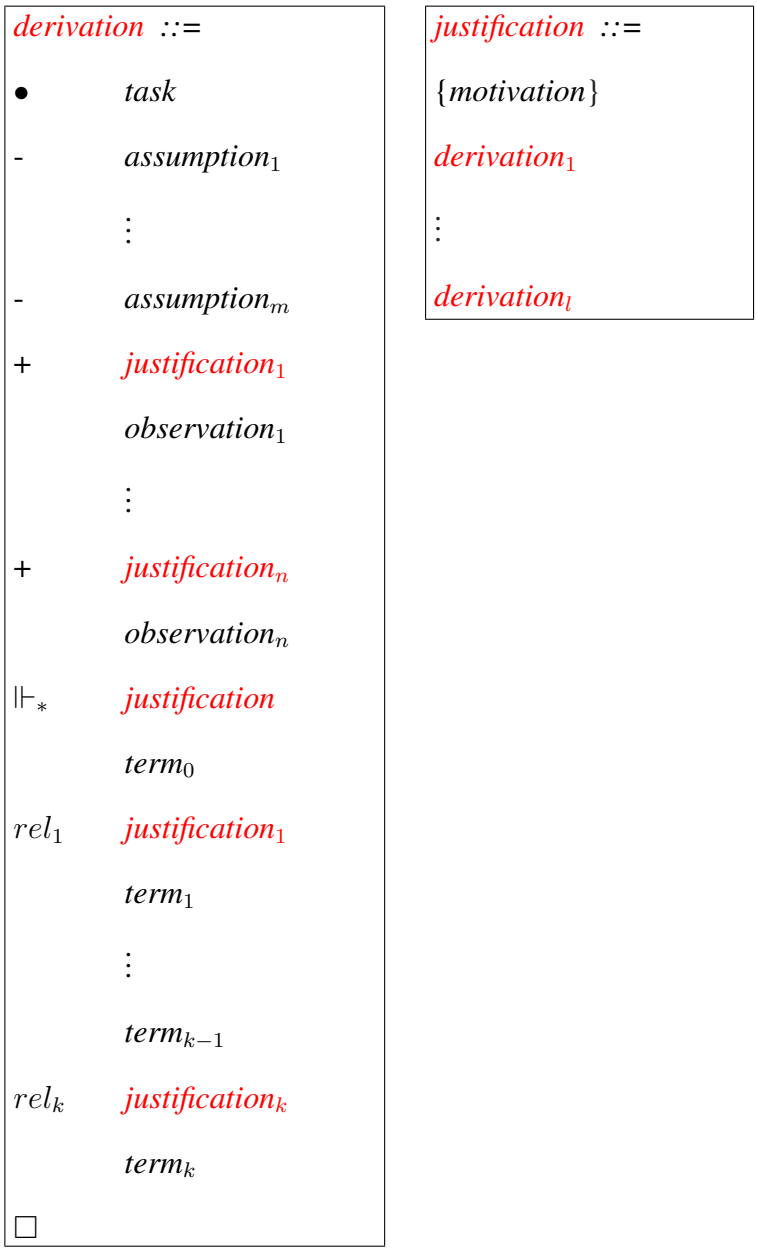
The horizontal line in Gentzen-style derivations is here indicated by the symbol "$\Vdash_0$" and by indenting the hypothesis one step to the right. The Gentzen-style derivation grows upwards, while the equivalent structured derivation grows sideways, nesting suderivations inside subderivations. This is a much more convenient way of presenting a proof on paper or on the computer screen, as has been noticed many times before.

Note that the translation to structured derivations uses the non-inheriting proof symbol "$\Vdash_0$", because the assumptions can change at each level in a Gentzen-style derivation. Note also that the symbol "$\vdash$" in a sequent calculus derivation is implicit in the structured derivation: the task (preceded by a bullet) is the consequent while the assumptions are the antecendent.

3.3 Combining different proof styles in a single derivation

The fact that calculational proofs, Hilbert-style proofs and Gentzen-style proofs all can be expressed as structured derivations is useful, but the real power of the notation comes from the fact that these proof styles can be combined freely in a single structured derivations. This means that we can use different proof styles in different parts of the derivation, choosing the proof style that best fits the problem at hand.

Consider the most general form of a structured derivation:



Here we assume that $m \geq 0$, $n \geq 0$, $k \geq 0$ and $l \geq 0$. If $m = 0$, then there are no assumptions in the derivation, if $n = 0$, then there are no observations, $k = 0$ indicates (by convention) that there is no calculation part, and if $l = 0$, there are no subderivations. The symbol $\Vdash_*$ stands for either $\Vdash$ or $\Vdash_0$.

The different proof styles are all special cases of the general structured derivations style. A pure Hilbert-style proof has no calculation part and no subderivations. A pure Gentzen-style proof has no calculation part and no observation part, and uses only $\Vdash_0$ for subderivations. A pure calculational proof has no observations and no subderivations.

A combination of Hilbert-style and calculational style has both observations and calculations but no subderivations. A combination of Hilbert-style and Gentzen-

style proofs has both observations and subderivations with $\Vdash_0$, but no calculations. A combination of calculational style and Gentzen-style had calculations and subderivations with $\Vdash_0$, but no observations. Finally, allowing calculations, observations and subderivations with $\Vdash$ and $\Vdash_0$ gives us the general structured derivations style, combining all proof styles in a single style.

The general style for structured derivation permit any combination of the three proof styles on any level of the proof. The examples given earlier illustrate the power of this combination of different proof styles in a single derivation. Examples 3 - 8 all use nested derivations, which is a Gentzen-style proof step. Examples 3, 4 and 5 show how to combine calculational style derivations with Gentzen-style nesting. Example 6 shows how to combine a Hilbert-style main derivation with a calculational proofs of observations and the main result. Example 7 combines a main derivation in Gentzen-style with calculational style proofs in the subderivations. Example 8 combines calculational style proofs at the outermost and innermost level, with Gentzen-style proof steps in the intermediate levels.

4 Syntax of structured derivations

We have previously described the syntax of structured derivations in an intuitive way, and illustrated its use in practice. However, we also need to give a formal syntactic definition of structured derivations, to fix the notation and to allow computer based syntax checking.

Structured derivations are not primarily intended to be read by computers but by humans. This puts a somewhat different set of constraints on the language design. The basic syntax of the language should be very simple and intuitive, so that users can remember it easily. It is also important that the syntax does not become too verbose, and that shortcuts are allowed whenever possible. Structured derivations are also often written by hand, another reason for not being too verbose.

For the same reason, we need clear and unambiguous rules for how to lay out a structured derivation. A computer parsing a programming language like Java does not really need to know how the program has been laid out in the text file, because the programming language syntax contains enough extra information to determined the structure. The price to pay for this is extra verbosity in the syntax (begin ... end, if ... then ... else ... end, etc). For structured derivations, this is a price that is too heavy. In stead, we have opted for describing the basic structure using indentation, in the same way as in some programming languages (Python, Occam, etc). There are some well-known problems associated with using indentation as a structuring element in computer programs, like confusion about the indentation level over page brakes. On the other hand, this kind of problems are easily solved by using an outlining editor. In addition to indentation, we identify the different parts of a structured derivation using special notation in the first column ("•, -, +,$\Vdash$,□", together with assumption and observation identifiers and

relation symbols).

Another design decision is that we do not want to fix the syntax of the basic constituents of a structured derivation (tasks, assumptions, observations, motivations, terms, and relations). The user should be allowed to use any specific notation that he or she is used to, and which he or she deems most suitable for the problem at hand. This does not mean that it would not make sense to fix a more specific notation for these basic constituents, but rather that should be a separated decision (we discuss this issue in more detail below). This design decision means that almost any text string can occur as a basic constituents. We then need to clearly delineate where a basic constituent starts and where it ends. Our solution is to have each constituent (except a relation) occupy the rest of the line (and possibly more successive lines), on a line of its own, from the first tab-character on the line to the last return-character.

We will below describe the syntax of structured derivations in three parts. The first part describes the general structure of structured derivations (the abstract derivation structure). The second part describes the concrete layout of a structured derivation. The third part lists the basic constituents of the syntax, which have to be defined in some way before we can actually parse a derivation. This gives the parameters of the language definition. Defining these basic constituents in different ways gives rise to different dialects of structured derivations. The parameter defintions has to satisfy some additional constraints, so that the first and second part definitions are not compromised.

4.1 Structure

The abstract syntax of structured derivations determines the overall structure and information content of a derivation. The derivation starts with the task to be solved (theorem to be proved, quantity to be computed, construction to be found). We then list the assumptions that may be used in the derivation. After the assumptions comes the observations that we can infer from the assumptions. Each observation must be justified by reference to assumptions or to previous observations.

Then comes the solution/proof of the task/conjecture itself. The proof may be either a justification alone, a calculation alone, or a justification followed by a calculation. The justification alone gives us a Gentzen-like proof of the task. A calculation alone gives us a Dijkstra-like proof of the task. The calculation is given in the Dijkstra style, with terms on their own lines, and a relation between the terms together with a justification on a line of its own. The proof contains both a justification and a calculation in cases where we need to explain why the calculation solves the problem, or when there are side conditions to the relations used in the calculations that need to be verified.

The following is the syntax definition of structured derivations:

$$
\begin{array}{rcl}
derivation & ::= & derivationId\, task \\
 & & (assumptionId\, assumption)^* \\
 & & (observationId\, justification\, observation)^* \\
 & & startproof\, proof\, [endproof] \\
derivation & ::= & derivationId\, calculation\, [endproof] \\
proof & ::= & justification \mid calculation \mid justification\, calculation \\
calculation & ::= & term\, (relation\, justification\, term)^+ \\
justification & ::= & motivation\, [startsub\, derivation^+\, endsub]
\end{array}
$$

The syntax is here defined in terms of five nonterminals. The syntax definition uses standard conventions, like a star superscript to indicate zero or more repetitions, a plus superscript to indicate one or more repetitions, and square brackets to indicate optional parts.

4.2 Layout

The next part of the syntax definition describes how to lay out a structured derivation in two dimensions, with explicit markers to indicate the different sections of the proof. The proof style is essentially a simple two-column format with indentation to show subderivations. The first column is for markers that indicate the kind of information coming next: a bullet "•" to indicate the task (as well as the start of a new derivation), a minus sign (a dash) to indicate that the following text is an assumption, a plus sign to indicate the start of an observation, a proof symbol ("$\Vdash$" or "$\Vdash_0$") to indicate where we start the proof of the task, and an end of proof symbol "□" to indicate that the proof is complete.

A subderivation is indicated by indenting the whole derivation one step to the right. This makes it easy to see where the subderivation starts and ends. Further indentation is used for nested subderivations. A list of subderivations is followed by an ellipsis ". . ." in the first column when the outer derivation continues. This is visible even when the nested derivation is hidden, so we know that there is a hidden derivation. The motivation of a justification is written inside curly brackets, "{ ... motivation ... }".

The tabbing distance is usually chosen to be the same as the indentation-dedentation distance, but these need not be the same. In some cases, it might be pedagogically motivated to explain the difference between the two column format and the indentation of subderivations by choosing the indentation to be larger that the tabbing distance.

The layout is defined by the following addition to the syntax above:

$$
\begin{array}{rcl}
derivationId & ::= & \text{" } \bullet \text{ " } | \, label \text{ " : "} \\
assumptionId & ::= & \text{" } - \text{ " } | \text{ "(" } label \text{ ")"} \\
observationId & ::= & \text{" } + \text{ " } | \text{ "[" } label \text{ "]"} \\
task & ::= & \text{tab } tasktText \text{ ret} \\
assumption & ::= & \text{tab } assumptionText \text{ ret} \\
observation & ::= & \text{tab } observationText \text{ ret} \\
term & ::= & \text{tab } termtText \text{ ret} \\
relation & ::= & relationText \\
motivation & ::= & \text{tab "\{" } motivationtText \text{ "\}" ret} \\
startproof & ::= & \text{" } \Vdash \text{ " } | \text{ " } \Vdash_0 \text{ "} \\
endproof & ::= & \text{"} \square \text{" ret} \\
startsub & ::= & \text{indent} \\
endsub & ::= & \text{dedent " } \ldots \text{ "}
\end{array}
$$

The terminal symbols are here either character sequences (like "•","-", ...) or special characters, like **tab** (tabulator), **ret** (return), **indent** (indent the line one step to the right) or **dedent** (indent the line one step to the left).

4.3 Parameters

The syntax leaves the exact syntax of the basic constituents as parameters. The parameters are the following:

$$
\begin{array}{rcl}
label & ::= & \ldots \text{ notation for labels } \ldots \\
taskText & ::= & \ldots \text{ notation for tasks } \ldots \\
assumptionText & ::= & \ldots \text{ notation for assumptions } \ldots \\
observationText & ::= & \ldots \text{ notation for observations } \ldots \\
termText & ::= & \ldots \text{ notation for terms } \ldots \\
relationText & ::= & \ldots \text{ notation for relations} \ldots \\
motivationText & ::= & \ldots \text{ notation for motivations } \ldots
\end{array}
$$

The syntax of these constructs has to be defined in such a way that it does not affect the syntax of the previous two parts. None of these constructs may contain a tab or return character, or an indent or dedent indication. A label may also not contain brackets (round, square or curly), nor start with a bullet, dash, plus, proof symbol or square.

4.4 Dialects

We get different dialects of structured derivations by instantiating the parameters in different ways. We will consider here four important dialects.

Informal use of structured derivations We often use structured derivations informally, using whatever ad hoc notation feels most suitable for the problem at hand. In this case, we allow any text string for describing the task, the assumptions, the observations, the terms and the motivations. Labels are used in a traditional way, and are usually Arabic or roman numerals, or alphabetic characters. This style is very loose, it does not even require any systematic use of logical notation in the derivations, and the motivations can be very informal and intuitive.

Structured derivations with logic The prefered way of using structured derivations is to insist that we use a systematic logical notation in tasks, assumptions, observations and terms, and that we motivate our derivation steps with logical inference rules or with explicit results from the underlying domain of discourse. We can characterize this as structured derivations with logic.

Structured derivations with logic fixes some specific logical framework as the basis for the argumentation. We prefer to use higher order logic, but one can also choose set theory or first order predicate calculus, or some more restricted framework like equational logic. In each case, we can be more precise about the basic constructs

$$
\begin{array}{rcl}
label & ::= & \ldots \text{ notation for labels } \ldots \\
taskText & ::= & imperative\, proposition \\
assumptionText & ::= & proposition\, correction \\
observationText & ::= & proposition \\
termText & ::= & expression \\
relationText & ::= & relationSymbol \\
motivationText & ::= & \ldots \text{ notation for motivations } \ldots \\
imperative & ::= & \text{... expression for what needs to be done ...} \\
correction & ::= & (\text{"}\#\#\text{"}\ proposition)^* \\
\end{array}
$$

The syntax of propositions, expressions and relation symbols is determined by the specific logical framework that we use, and by the specific logical theory that we are working in. For ease of writing proofs and derivations, it is often useful to allow informal natural language expressions in propositions, expressions and relation symbols, in order to make the proof more readable. We still retain the possibility to use specific logical inference rules to justify the steps in the proof.

We have added here a little bit of information to make the derivations more readable: an imperative that explicitly states what needs to be done, and a correction facility that can be used to indicate the order in which the derivation is to be read. Neither of these two features is really needed for the logical correctness of the derivation, and can be ignored when checking the correctness of a derivation.

Axiomatic proofs The use of a fixed logical framework for structured derivations is not the same as a fully axiomatic proof. A fully axiomatic proof is in most cases much too detailed and cumbersome to write down. However, it is important that one is aware of what is the minimum information that needs to be given in a fully axiomatic proof. We can get a fully axiomatic proof by fixing the undefined basic constructs in the previous syntax, as follows:

$$\begin{aligned} label &::= number \\ motivationText &::= ruleName\,[rule]\,assignment\,validity \\ imperative &::= \text{""} \end{aligned}$$

Here we fix the labels to be (distinct) numbers. The motivation starts by a rule name (like "Modus ponens", "Excluded middle", "Distributivity of multiplication over addition", etc.). The rule itself may be optionally given in the motivation. If the rule is given in terms of meta variables, then the assignment part describes how these meta variables are instantiated in this application of the rule. Finally, the validity part states that the side condition for the rule is satisfied. The side condition is a constraint on the meta variables that must be satisfied before the rule can be applied in the proof. Violating the side condition may lead to unsound proofs. There is no imperative expression in the task.

Structured derivations for proof checkers Structured derivations can be seen as a front end for both mechanized proof checkers like HOL, PVS, Isabelle, etc and for fully automatic proof checkers like ... . In that case, we choose the syntax for structured derivations with logic, and chose the logical framework to be the one used in the proof checker. This means giving precise syntactic definitions for *proposition*, *expression* and *relationSymbol.* The motivations would in this case be strategies that the proof checker is asked to follow:

$$motivationText ::= strategy$$

Using structured derivations with mechanized proof checkers gives a slightly different working method than the usual way of doing mathematical proofs by hand. In most cases, a step would be carried out interactively with the computer. Starting from some given expression, the user would indicate the strategy to be

tried out as the motivation for the next step. The computer would then generate the result of this step, as well as the possible subderivations that need to be carried out. The user would try to come up with a proof in this way, backtracking whenever necessary, until the final proof has been constructed.

Alternatively, we can have a structured derivation written in this form, and we can use a mechanized proof checker to verify that each step in the derivation is correct. This can be very useful in an educational context, e.g. for automatic checking of student assignments and exams.

5 Semantics of structured derivations

We have in section 2 shown that structured derivations can be used to express proofs in three main proof styles: calculational proofs, Hilbert-style proofs and Gentzen-style proofs. We now show that each structured derivation can be reduced to an equivalent Gentzen like proof. In other words, structured derivations and Gentzen-like proofs are equivalent in proof power. We can see structured derivations as a more user friendly notation for Gentzen-like proofs. The reduction of a structured derivation proof to a Gentzen- style proof provides the semantic definition of structured derivations.

5.1 Structured derivations notation for sequent calculus derivations

Let us start by introducing a very a simple subset of structured derivations, where we only allow the non-inheriting proof symbol $\Vdash_0$ and do not have any observations or calculations. The syntax of such a *basic structured derivation* is as follows:

- con
- ass_1

$\vdots$

- ass_m

$\Vdash_0$ $\{R\}$

der_1

$\vdots$

der_n

$\square$

A structured derivation of this form is just an alternative notation for a sequent calculus derivation, as explained earlier. Let us make this more precise. For each basic structured derivation der, we define the corresponding sequent calculus derivation der^{α}, as follows:

$$der^{\alpha} = \frac{der_1^{\alpha}, \ldots, der_n^{\alpha}}{ass_1, \ldots, ass_m \vdash con}\,\{R\}$$

In other words, $ass_1, \ldots, ass_m \vdash con$ can be concluded from the derivations $der_1^{\alpha}, \ldots, der_n^{\alpha}$, using the proof rule R. Notice that the transformation is applied recursively to the subderivations. This transformation gives us a standard sequent calculus derivation for each structured derivation in the simple format. Vice verse, any sequent calculus derivation can be written as a basic structured derivation using this format, as we explained earlier.

5.2 Inheriting assumptions

We now look at the different new features introduced in structured derivations (as compared to standard Gentzen-style derivations), and explain them one by one, by showing what they correspond to in a Gentzen-style derivation. We start by showing how assumptions are inherited in structured derivations.

A structured derivation uses the inheriting proof symbol $\Vdash$ in addition to the standard proof symbol $\Vdash_0$. In practice, the inheriting proof symbol is used almost exclusively in structured derivations. We define the use of the inheriting proof symbol by showing how a derivation using this symbol can be translated into an equivalent derivation that only uses the symbol $\Vdash_0$.

Let us first introduce a notation for propagating assumptions in derivations. Assume that we have a derivation

$$der = \frac{der_1, \ldots, der_n}{A \vdash con}\,\{R\}$$

Let $B = ass_1, \ldots, ass_m$ be a list of assumptions. Then we denote by $[B]der$ the derivation that we get from *der* by adding B as assumptions in the conclusion, i.e.,

$$[B]der = \frac{der_1, \ldots, der_n}{B,\ A \vdash con}\,\{R\}$$

Assume that we have an extension of basic structured derivations, where we allow both the inheriting proof symbol $\Vdash$ and the standard proof symbol $\Vdash_0$. Consider now the transformation that removes the inheriting proof symbol, described below. On the left, we have a derivation der that uses the proof symbol $\Vdash$, on the right we have the corresponding derivation der^{β} using only the proof symbol $\Vdash_0$.

$$
\begin{array}{lll}
\bullet & con & \longrightarrow \quad \bullet \quad con \\
\text{-} & ass_1 & \text{-} \quad ass_1 \\
\vdots & & \vdots \\
\text{-} & ass_m & \text{-} \quad ass_m \\
\Vdash & \{R\} & \Vdash_0 \quad \{R\} \\
 & der_1 & \quad ([ass_1, \ldots, ass_m]der_1)^{\beta} \\
 & \vdots & \quad \vdots \\
 & der_n & \quad ([ass_1, \ldots, ass_m]der_n)^{\beta} \\
\Box & & \Box
\end{array}
$$

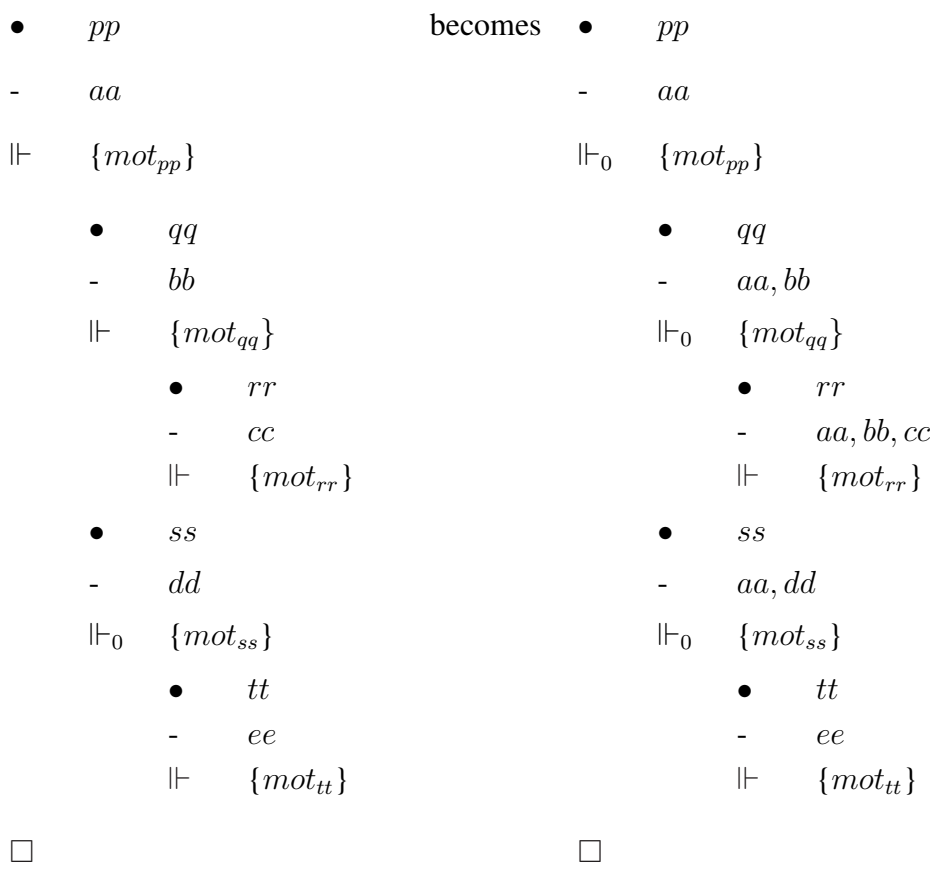
The transformation thus replaces the inheriting proof symbol with the standard proof symbol. At the same time, it adds the assumptions of the conclusion to all hypothesis, before (recursively) removing the inheriting proof symbol also from these hypothesis.

We also need to define what it means to remove the inheriting proof symbol from a derivation that has the standard proof symbol in the conclusion. This is necessary, because the inheriting proof symbol may still be used in some subderivations. On the left, we have a structured derivation with a standard proof symbol, on the right we have the corresponding derivation where all inheriting proof symbols have been removed.

$$
\begin{array}{lll}
\bullet & con & \longrightarrow \quad \bullet \quad con \\
\text{-} & ass_1 & \text{-} \quad ass_1 \\
\vdots & & \vdots \\
\text{-} & ass_m & \text{-} \quad ass_m \\
\Vdash_0 & \{R\} & \Vdash_0 \quad \{R\} \\
 & der_1 & \quad der_1^{\beta} \\
 & \vdots & \quad \vdots \\
 & der_n & \quad der_n^{\beta} \\
\Box & & \Box
\end{array}
$$

In this case, the assumptions are not pushed into the subderivations. However, the subderivations are transformer as before, so nested inheriting proof symbols will be treated in the right way.

The following example shows how the transformation works:



5.3 Observations

We next define derivations with observations, by showing how to transform a structured derivation that contains observations (but no calculations) to one that does not contain observations (nor calculations). On the left hand side, we have a derivation der with observations, on the right we have an equivalent derivation der^{γ} without observations.

$\bullet$ con

$-$ ass_1

$\vdots$

$-$ ass_m

$+$ $\{R_1\}$

$der_{1,1}$

$\vdots$

der_{1,n_1}

$\ldots$ obs_1

$\vdots$

$+$ $\{R_k\}$

$der_{k,1}$

$\vdots$

der_{k,n_k}

$\ldots$ obs_k

$\Vdash$ $\{R\}$

der_1

$\vdots$

der_n

$\square$

$\bullet$ con

$-$ ass_1

$\vdots$

$-$ ass_m

$\Vdash$ $\{lemma\, rule\}$

$\bullet$ obs_1

$\Vdash$ $\{R_1\}$

$der^{\gamma}_{1,1}$

$\vdots$

der^{γ}_{1,n_1}

$\vdots$

$\bullet$ obs_k

$-$ obs_1

$\vdots$

$-$ obs_{k-1}

$\Vdash$ $\{R_k\}$

$der^{\gamma}_{k,1}$

$\vdots$

der^{γ}_{k,n_k}

$\vdots$

$\bullet$ con

$-$ obs_1

$\vdots$

$-$ obs_k

$\Vdash$ $\{R\}$

der^{γ}_1

$\vdots$

der^{γ}_n

$\square$

The lemma rule used to justify the transformation is as follows:

$$\frac{A \vdash obs,\ A, obs \vdash con}{A \vdash con}\ \{lemma\}$$

The lemma rule thus says that we can prove that $A \vdash con$ is true, by first proving that a lemma obs follows from A, and then proving that the required conclusion con follows from assumptions A together with the lemma obs. This is the basic inference rule that justifies using lemmas in proofs.

The general form for the lemma rule is as follows:

$$\frac{A \vdash o_1,\ A, o_1 \vdash o_2,\ \ldots,\ A, o_1, o_2, \ldots, o_{k-1} \vdash o_k,\ A, o_1, o_2, \ldots, o_k \vdash con}{A \vdash con}\ \{lemma\}$$

In other words, if we can prove each proposition o_i from the assumptions A and the preceding propositions $o_1, \ldots, o_{i-1}$, for $i = 1, \ldots, k$, and we can prove con from the assumptions A and the propositions $o_1, \ldots, o_k$, then $A \vdash con$ holds.

The derivation that we get after the transformation combines the lemma rule with the specific rules for proving the conclusion and the observations. The transformed derivation corresponds to the following proof in standard notation:

$$\frac{\dfrac{der_{11}, \ldots}{A \vdash o_1}\ \{R_1\},\ \ldots,\ \dfrac{der_{k1}, \ldots}{A, o_1, \ldots, o_{k-1} \vdash o_k,}\ \{R_k\}\ \dfrac{der_1, \ldots}{A, o_1, \ldots, o_k \vdash con}\ \{R\}}{A \vdash con}\ \{lemma\}$$

5.4 Calculations

Let us finally define the meaning of calculations, by showing how to transform any structured derivation der with a calculation to an equivalent structured derivation der^{δ}that does not contain a calculation. We show der on the left and der^{δ} on the right:

$$
\begin{array}{ll}
\bullet & con \\
\text{-} & assumptions \\
+ & observations \\
\Vdash_x & \{R\} \\
 & der_1 \\
 & \vdots \\
 & der_n \\
\ldots & term_0 \\
rel_1 & \{R_1\} \\
 & der_{1,1} \\
 & \vdots \\
 & der_{1,n_1} \\
\ldots & term_1 \\
\vdots & \\
 & term_{k-1} \\
rel_k & \{R_k\} \\
 & der_{k,1} \\
 & \vdots \\
 & der_{k,n_k} \\
\ldots & term_k \\
\square &
\end{array}
\qquad
\begin{array}{ll}
\bullet & con \\
\text{-} & assumptions \\
+ & observations \\
\Vdash_x & \{lemma'\} \\
 & \bullet \quad con \\
 & \text{-} \quad term_0\, rel_1\, term_1 \\
 & \vdots \\
 & \text{-} \quad term_{k-1}\, rel_k\, term_k \\
 & \Vdash \quad \{R\} \\
 & \quad der_1^\delta \\
 & \quad \vdots \\
 & \quad der_n^\delta \\
 & \bullet \quad term_0\, rel_1\, term_1 \\
 & \Vdash \quad \{R_1\} \\
 & \quad der_{1,1}^\delta \\
 & \quad \vdots \\
 & \quad der_{1,n_1}^\delta \\
 & \vdots \\
 & \bullet \quad term_{k-1}\, rel_k\, term_k \\
 & \Vdash \quad \{R_k\} \\
 & \quad der_{k,1}^\delta \\
 & \quad \vdots \\
 & \quad der_{k,n_k}^\delta \\
\square &
\end{array}
$$

In other words, each relational term in the calculation is proved in a separate subderivation.

Discussion We can justify this transformation with the following variant of the lemma rule:

$$\frac{A \vdash b_1,\ \ldots,\ A \vdash b_n,\ A, b_1, \ldots, b_n \vdash con}{A \vdash con}\ \{lemma'\}$$

The difference here, as compared to the previous lemma rule, is that the proofs of the lemmas $b_1, \ldots, b_n$ are independent of each other. This rule is easily proved with the lemma rule, by using the rule

$$\frac{A \vdash con}{A, B \vdash con}$$

This says that we can always add additional assumptions to a sequent without loosing correctness.

The transformed derivation, written in standard form, is as follows:

$$\frac{\dfrac{der_{11}, \ldots}{A, O \vdash t_0\, r_1\, t_1}\,\{R_1\},\ \ldots,\ \dfrac{der_{k1}, \ldots}{A, O \vdash t_{k-1}\, r_k\, t_k}\,\{R_k\},\ \dfrac{der_1,\ \ldots,\ der_n}{A, O,\ t_0\, r_1\, t_1,\ \ldots \vdash con}\,\{R\}}{A, O \vdash con}\,\{lemma'\}$$

5.5 Semantics of structured derivations

Consider now an arbitrary structured derivation der. The meaning of this derivation is defined as the equivalent sequent calculus derivation that we get by first removing all calculations in the derivation, then removing all observations, then removing all inheriting proof symbols, and finally writing the derivation in the standard sequent calculus notation. In other words, the meaning $\overline{der}$ of the structured derivation der is

$$\overline{der} = (((der^{\delta})^{\gamma})^{\beta})^{\alpha}$$

The fact that we define the meaning of a structured derivation by a reduction to a standard Gentzen-style proof means that structured derivations can be seen as an alternative way to express Gentzen-style derivations. This means, in particular, that we can freely use all the standard inference rules available in Gentzen-style systems.

6 Conclusions

We have above described SDS2, an extensions to the original structured derivation style SDS1 which allows us to unify the three main proof styles in use today: Gentzen-style backward chaining proofs, Hilbert-style forward chaining proofs and Dijkstra-style calculational proofs. These three proof styles can be freely mixed in a single structured derivation, so that one can in each part of the derivation choose the style that is most suitable for the problem at hand. The main aim of structured derivations is to provide a human readable and well-structured description of a mathematical argument, and a style that can be used on any kind of mathematical domain.

Our primary goal is to try to improve teaching mathematics at secondary and tertiary education level, by providing the student with a standard format for writing down the solution to a mathematical problem. Structured derivations require

that each step in the argumentation is explicitly justified by some accepted mathematical facts or rules of inference. This makes it easier for students to understand the derivation, both when the teacher presents it at the white board and when he or she is reading the derivation later, when solving assignments or preparing for an exam. It also makes it easier for the teacher to understand the students solutions and to correct errors in them, and to see what the student has has and has not understood. The structured derivations style has performed well in empirical tests in Finland, and students at high school and university level have learned to use the style without much problems.

The exact syntax of structured derivations makes it rather easy to check the syntax of a derivation, and to build editors for structured derivations that understand and support the syntax. Building extensive tool support for structured derivations is one of our main research topics for the moment. This could ultimately pave the way for a more wide spread adaption of computers in mathematics education, both using computers to write mathematical text, communicating mathematical texts between teacher and students over the net, and as a tool for automatically analysing and checking student solutions to mathematical exercises.

Acknowledgements The work on developing the structured derivations approach has been carried out in close cooperation with the members of the Learning and Reasoning laboratory at Abo Akademi University and University of Turku. I want to thank the following persons, who all have in one way or the other contributed to the development of the method, testing the method in practical teaching experiments, or worked on tool support for the method (the list is in alphabetic order): Stefan Asikainen, Johannes Eriksson, Tanja Kavander, Linda Mannila, Martin Nylund, Mia Peltomäki, Viorel Preoteasa, Teemu Rajala, Tapio Salakoski, Petri Sallasmaa, Fredrik Sandström, Patrick Sibelius, Solveig Wallin and Joakim von Wright. The research has been financed by the Academy of Finland, as part of the funding for the Center of Excellence in Formal Methods in Programming, and by the Finnish Technology Industry Centennial Fund.

References

[1] Ralph-Johan Back, Jim Grundy, and Joakim von Wright. Structured calculational proof. *Formal Aspects of Computing*, 9:469–483, 1998.

[2] Ralph-Johan Back, Linda Mannila, Mia Peltomaki, and Patrick Sibelius. Structured derivations: A logic based approach to teaching mathematics. In *FORMED 2008: Formal Methods in Computer Science Education, Budapest*, 2008.

[3] Ralph-Johan Back, Mats Sjöberg, and Joakim von Wright. Field tests of the structured derivations method. Technical Report 491, TUCS - Turku Centre for Computer Science, Turku, Finland, Nov 2002.

[4] Ralph-Johan Back and Joakim von Wright. *Refinement Calculus: A Systematic Introduction*. Springer-Verlag, 1998. Graduate Texts in Computer Science.

[5] Ralph-Johan Back and Joakim von Wright. A method for teaching rigorous mathematical reasoning. In *Proceedings of Int. Conference on Technology of Mathematics*, University of Plymouth, UK, Aug 1999.

[6] A. Church. A formulation of the simple theory of types. *Journal of Symbolic Logic*, 5:56–68, 1940.

[7] Edsger W. Dijksta and C. S. Scholten. *Predicate Calculus and Program Semantics*. Springer-Verlag, 1990.

[8] E.W. Dijkstra. The notational conventions i adopted, and why. *Formal Aspects of Computing*, 14:99 – 107, 2002.

[9] M.J.C. Gordon and T.F. Melham. *Introduction to HOL.* Cambridge University Press, New York, 1993.

[10] David Gries. Teaching calculation and discrimination: A more effective curriculum. *Communications of the ACM*, (34):45 – 54, 1991.

[11] David Gries. *Teaching and Learning Formal Methods*, chapter Improving the curriculum through the teaching of calculatio and discrimination, pages 181–196. Academic Press, London, 1996.

[12] David Gries and Fred Schneider. *A Logical Introduction to Discrete Mathematics*. Springer-Verlag, 1993.

[13] David Gries and Fred Schneider. Teaching math more effectively through calculational proofs. *Am. Math. Monthly*, pages 691–697, October 1995.

[14] Leslie Lamport. How to write a proof. *American Math. Monthly 102 (1995), no. 7, 600–608.*, 102(7):600–608., 1995.

[15] Linda Mannila and Solveig Wallin. Promoting students justification skills using structured derivations. In *ICMI 19 studies*, 2009.

[16] L. C. Paulson. Isabelle: the next 700 theorem provers. In P. Odifreddi, editor, *Logic and Computer Science*, pages 361–386. Academic Press, 1990.

[17] Mia Peltomaki and Ralph-Johan Back. An empirical evaluation of structured derivations in high school mathematics. In *ICMI-19 studies*, Taipei, Taiwan, 2009.

[18] Linda Mannila Ralph-Johan Back and Solveig Wallin. Student justifications in high school mathematics. In *CERME 6*, January 2009.

[19] Natarajan Shankar Sam Owre and John Rushby. Pvs: A prototype verification system. In *CADE 11*, Saratoga Springs, NY, June 1992.

[20] A Trybulec. The mizar logic information language. In *Studies in Logic, Grammar and Rhetoric*, volume 1. Bialystok, 1980.

[21] Rudnicki P. Trybulec, A. On equivalents of well-foundedness. an experiment in mizar,. *Journal of Automated Reasoning*, 23:197–234, 1999.

[22] Jan L. A. van de Snepscheut. *What computing is all about*. Springer Verlag, 1993.

[23] A. J. M. van Gasteren. *On the Shape of Mathematical Arguments*. Lecture Notes in Computer Science. Springer-Verlag, Berlin, 1990.

[24] Markus Wenzel. Isar - a generic interpretative approach to readable formal proof documents. In A. Hirschowitz C. Paulin L. Thery Y. Bertot, G. Dowek, editor, *Theorem Proving in Higher Order Logics, 12th International Conference, TPHOLs'99*, volume 1690 of *LNCS*. Springer Verlag, 1999.

Turku
Centre *for*
Computer
Science

Lemminkäisenkatu 14 A, 20520 Turku, Finland | www.tucs.fi

University of Turku
- Department of Information Technology
- Department of Mathematics

Åbo Akademi University
- Department of Computer Science
- Institute for Advanced Management Systems Research

Turku School of Economics and Business Administration
- Institute of Information Systems Sciences

ISBN 978-952-12-2318-1
ISSN 1239-1891