

Co-Synthesis of New Complex Selection Algorithms and their Human Comprehensible XML Documentation

Jutta Eusterbrock

JEusterbrock@seamless-solutions.de

Abstract. In this paper, an approach for program and algorithm synthesis within a higher-order framework is presented that allows us to generate new algorithm structures together with readable documentation which enable users to conveniently inspect the results of this synthesis process. The synthesis approach is based on feature graphs as a higher-order data type for specifying synthesis types like proof terms, theorems and document types. The document synthesis method uses constructive reasoning about a user-defined knowledge base to synthesise documents from metaobjects which are composed of diagrams, text, and references to articles, theorems and algorithms. The data format used for reasoning is a variant of XML syntax and thus enables the organisation of comprehensive knowledge in XML repositories and the conversion of document terms into \LaTeX , PDF or XHTML documents which can be displayed by browsers.

This framework has been applied for the automated synthesis of several new selection algorithms and their documentation. As one new result, an algorithm that proves selecting the 4th element of 24 elements needs at most 34 comparisons was synthesised using 50 seconds CPU time on an AMD Athlon 3200. The correctness of the synthesised algorithm was manually checked. It improves the known upper bounds for the specific selection problems. The running time for synthesis is several orders of magnitude more efficient than comparative approaches.

1 Introduction

The idea of automatically discovering solutions to mathematical conjectures or proving the non-existence of solutions which are beyond human comprehension is intriguing. In recent years, computers have been used for a number of famous algorithmic problems to facilitate analysis based upon a detailed problem specific formalisation. For instance, C. Lam proved that a projective plane of order 10 does not exist [Lam91]. As the “proof” took several years of computer search (the equivalent of 2000 hours on a Cray-1), it is considered to be the most time-intensive computer assisted single proof. Recently, G. Gonthier, a mathematician who works at Microsoft Research in Cambridge, England, used the CoQ system (cf. [Dev05]) to verify the computer supported proof of the famous Four Color

Theorem, which was originally proven in 1976 by Appel and Haken who used computer programs to check a very large number of cases.

However, for a mathematician it is unsatisfying to know that there exists a solution or no solution for a problem, because thousands, hundreds of thousands or millions of states have been explored by a theorem prover whose rules have been verified, and at the same time, not be able to comprehend the tedious machine-generated proofs and not be able to draw conclusions from the automated proof.

In previous work [Eus92b] on automated algorithm synthesis, we designed and implemented a metalevel methodology and system to assist algorithm synthesis and the proof of lower complexity bounds. The system evaluated the search process and derived new knowledge which was abstracted and added to a dynamically growing knowledge base. The system was applied to assist the proof of a number-theoretic conjecture for the selection problem. It was possible to prove the values for very small n within a few minutes on a Sparc Workstation. Surprisingly, the system synthesised an algorithm which constructed a counter example to a set of rules which should be verified by the system [Eus92a]. As a side effect, an isolated counter-example for a published lower bound for the selection problem was constructed. The computation took several days and the machine generated proofs were cumbersome because the nested proof graphs are hard to comprehend. However, treating proofs as graphs enables various forms for generating explanations. In [EN96], a visualisation component was implemented in order to assist the exploration of huge graphs. Techniques like zooming, organising of graphs on various hierarchy levels, folding and unfolding techniques enabled the interactive exploration of large graphs. However, it is assumed that presenting algorithms by documents which are similar to scientific or textbook descriptions, making use of different formats and establishing links to published results is cognitively more appropriate for proof presentation than uniform means such as visualisation techniques. Meanwhile, XML-based data formats have emerged as a standard for data encoding and exchange. XML documents can be comparatively easily converted into web pages, \LaTeX or PDF document files which facilitate the display of natural language, graphics, mathematical symbols and references in an appropriate form.

In this paper, it is illustrated how XML based documentation for automatically synthesised algorithms and programs can be generated from specifications and proof terms and further processed by XML tools is analysed. Synthesised theorems and programs should be presented at the right level of granularity such that experts can check the automatically synthesised programs in the same way as they verify a proof in a publication. The key to a solution is abstraction. It is achieved by a higher-order formalisation of abstract synthesis objects and document components and a higher-order approach towards automated synthesis which raises the level of abstraction (cf. [Kre93]). A method is devised that transforms program specifications and synthesised algorithms into documents which include graphics, text, and references. The method is based upon general correspondences between synthesis types and document types. User-defined

context-specific rules as how to decompose the proof graph, when to generate lemmata and what kind of additional visual information to present at various stages can be provided. The method has been applied to automatically synthesise documents for new complex selection algorithms which were automatically synthesised.

The paper is organised as follows. In section 2, the selection problem is introduced. Section 3 gives an overview of the synthesis framework. In section 4, the synthesis types and the building blocks of the metatheory for the encoding of mathematical knowledge are defined. The document synthesis method is explained in section 5. Section 6 summarises some experimental results. Section 7 concludes this paper. Appendix A contains the automatically synthesised document for the automatically constructed new selection algorithm.

2 The Selection Problem

The selection problem is the problem of finding the i -th largest element, given a set of n distinct unordered numbers, $1 < i < n$. The special case $i = \lceil n/2 \rceil$ is the median problem. The worst-case, minimum number of comparisons is denoted by $V_i(n)$. The problem goes back to Rev. C. L. Dodgson's (aka Lewis Carroll) essay on how prizes were awarded unfairly in tennis tournaments (see Knuth [8:5.33]). In the classic book *The Art of Computer Programming, Volume 2, Sorting and Searching* [Knu73a], D. Knuth introduces the problem and states the combinatorial bounds for $n \leq 10$. In [Eus85], the present author constructed the formula $H_i(n)$

$$H_i(n) = n - i + \sum_{l=1}^{i-1} (\lceil \lg(\frac{n-i+2}{i-l+3}) \rceil + 2). \quad (1)$$

It was shown in [Eus85] that the numbers $H_i(n)$ unify the published results for the worst-case behaviour of selection-algorithms as follows. For admissible combinations of i, n the numbers $H_i(n)$:

- match the exact numbers $V_i(n)$ for $i = 1, 2, 3$ [Kis64, Knu73b, Aig82];
- are equal to or less than the lower bounds [Kis64, Yao74, FG79, Aig82, MP82, BJ85];
- are equal or greater than the upper bounds [Kis64, HS69, BFP⁺73, FR75, SPP76, Yap76, Aig82, RH84]

known at that time. Furthermore, using the numbers $H_i(n)$, novel combinatorial algorithms for small values of i, n were constructed which prove the upper bounds for small values of i, n :

$$V_i(n) \leq H_i(n), \text{ iff } i \leq 4, n \leq 14 \text{ and } i \leq 5, n \leq 12. \quad (2)$$

The approximate behaviour for the medians is given by the formula below

$$H_{n/2}(n) \approx 2.5n - 3\lceil \lg(n+4) \rceil + 5 \quad (3)$$

The author stated the hypothesis $V_i(n) = H_i(n)$ for all i, n . For all known upper and lower bounds either $V_i(n) \geq L_i(n)$ or $V_i(n) \leq U_i(n)$ with one exception which will be described later on. The hypothetical formula for the median coincides with the conjecture of Yao and the conjecture of Paterson $V_{n/2} \approx 2.4094n$. In [Eus92b] we designed a system in order to assist the refinement of rules. Using this system, it was possible to prove the values for very small n within a few minutes on a Sun Workstation. Surprisingly the system synthesised an algorithm which proves $V_3(22) < H_3(22)$ and thus constructed a counter example to a set of rules which should be verified by the system [Eus92a]. The computation took several days. Computerised searches for the selection problem were subsequently performed by [GKP96] and [Oks05] which use alpha-beta search, a transposition table and some optimisations. Oksanen's system constructs decision graphs which in summary suggest $V_i(n) \leq H_i(n)$, iff $i \leq 6, n \leq 14$. However, it is also claimed that $V_5(12) \geq 19 = H_5(12) - 1$ while in [Eus85] an algorithm is presented that proves $V_5(12) \leq 18$ and thus contradicts the automatically proven statement. The constructed decision graphs presented in [Oks05] are too large to comprehend and in the case $i = 7, n = 14$ consist of more than 600 nodes.

The process for constructing algorithms or lower bounds for selection problems is similar to minimum-comparison sorting. M. Peczarski has devised a system and analysed the optimal lower bounds $S(n)$ for sorting n elements, $n = 13, 14, 22$ based on an algorithm for counting the linear extensions of partial orders. The proof $S(22) > 70$ took 1740 hours on a computer with a 650 MHz processor (cf. [Pec04]).

3 The Higher-order Synthesis Framework

In this paper, a reformalisation and reimplementaion of our knowledge-based synthesis framework, called SEAMLESS (cf. [Eus95]) is analysed. The synthesis system consists of verified generic methods which take a specification as input and generate metalevel proofs by metalevel reasoning about domain-specific knowledge which may be interpreted as programs. The synthesis system evaluates the search processes, generalises the case solutions which result from successful and failed proof attempts and stores them for further reuse in the knowledge base. A documentation synthesis component has been added. The document synthesis component transforms specifications, derived theorems and synthesised proofs into XML documents including graphical visualisations, references and text. The XML documents can be converted by freely available tools into human-readable documents in multiple formats such as XHTML, L^AT_EX or PDF. The resulting system structure is shown in Figure 1.

In order to achieve an integrated formal framework for the logic-based co-synthesis of proofs and their documentation, principles of a higher-order logic of program synthesis are applied (cf. [Kre93]). The core for the integration of the different types of knowledge and the correct design of the synthesis methods is a metatheory or, in other terms, an ontology. Types and higher-order predicates to represent the structure of algorithm design knowledge fragments and properties

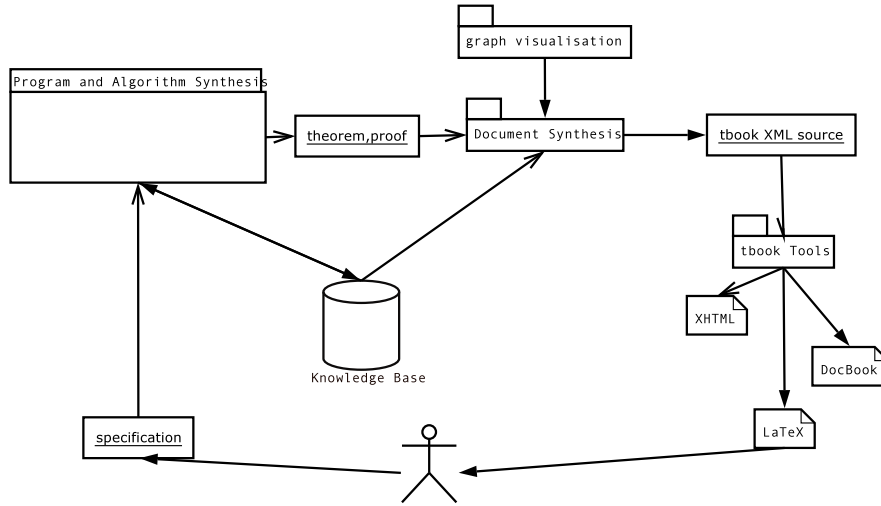


Fig. 1. Scenario for Knowledge-based Synthesis

or relationships between them have been defined. Figure 2 lists the major types of the SEAMLESS framework.

Basic types *Bool, Integer, Constants, Vars, String*
 Object logic *Atom, Clause, Algebraic Expression, Constraint*
 Synthesis types *Precondition, Postcondition, Program, Proof*

Fig. 2. Types for Program Synthesis

In this paper, the notation of [Kre93] is adopted and used in a semi-formal way. A definition *New Object Type* \equiv *Composition of Defined Object Types* defines a new object in terms of already existing object types. Meta theorems are written in the form $Goal \Leftarrow Subgoal_1 \wedge \dots \wedge Subgoal_r$, or they are stated as facts.

The Floyd-Hoare logic [Hoa69] is used for specifying the semantics of imperative programs and to associate logical specifications with programs. In Floyd-Hoare's logic, triples of the form $\{Pre\}Prog\{Post\}$ state that if program *Prog* starts in an input state satisfying *Pre* then if and when *Prog* halts, it does so in a state satisfying *Post*. Programs are sequences of statements. Hoare provided a set of logical rules in order to reason about the correctness of computer programs. It is well-known that the Floyd-Hoare rules and axioms can be embedded in higher order logic and become derived rules. In the SEAMLESS framework, a statement is a variable assignment, procedure or conditional. Loop statements together with their specifications can be added to the knowledge base, however, their automatic synthesis is currently not supported as it is based on non-trivial

mathematical induction. Consequently, the key conceptual building elements of the SEAMLESS theory include the abstractions precondition, postcondition, proofs which may be interpreted as programs, and, moreover, range constraints for a cost function as part of the postcondition. The metatheory is defined in terms of generic predicates and formal axioms for them which provide correctness axioms for the suitable domain theories. For example, the correctness axiom for the higher-order predicate *Know* states that a proof for the validity of a Hoare triple is known.

$$Know(Pre, Post, Prog, True) \equiv \vdash \{Pre\}Prog\{Post\}$$

Domain-specific design knowledge can be provided by definitions for the open generic higher-order predicates, if the correctness axioms are satisfied. The SEAMLESS knowledge base of the system entails theorems which are relations among truth values, specifications and programs. They are stated as higher-order theorems, once types for the corresponding abstractions and the semantics of the higher-order predicates have been defined. In this application scenario, the knowledge bases contain published domain specific theorems about the complexity of selection problems, as summarised in section 2, and various related algorithms encoded as arguments of the higher-order predicate *Know*. The knowledge base of the synthesis system also comprises theorems whose proof is only given by a bibliographic reference to the corresponding document.

Generic synthesis methods have been derived from the generic predicates as metatheorems. The proofs-as-programs paradigm is adapted to the Hoare logic for the purpose of extending it to synthesising imperative programs. A synthesis method is a set of rules including metavariables for programs which are instantiated while proving the synthesis task. A very basic synthesis strategy is to retrieve solutions from the knowledge base which is specified below.

$$Synthesis(Pre, Prog, Post, Bool) \Leftarrow Know(Pre, Prog, Post, Bool)$$

Experience has shown that theorems as they are published in the literature are often not directly applicable for solving a problem specification. Equivalences, generalisations, and reductions are considered to establish semantic relations between specifications and the theorems in the literature to obtain proofs. These relations can be modelled by corresponding higher-order predicates.

4 Graphterms as a Higher-order Datastructure and XML

The extensible Markup Language (XML) has emerged as a quasi-standard for knowledge exchange, document processing and Web applications. XML is a metalanguage used to create generalised markup languages. XML annotations facilitate the retrieval of document fragments based upon their semantic annotations. The focus in this paper is on a more convenient and efficient data structure than arbitrary terms for the organisation of the knowledge fragments. The data structures are optimised in order

- to facilitate the automatic synthesis of human comprehensible XML-based documents;
- to elicit the interplay between design knowledge - which has been published in scientific documents and Web resources - and the corresponding encoding as theorem in the knowledge base of a reasoning system;
- to tackle the huge search complexities.

Linear (serial) term representations, named *graphterms*, were devised as a data structure for the encoding of labelled DAGs and feature graphs [Eus97]. Graphterms (cf. [Eus01,Eus97]) are used as the core data structure to encode formalised knowledge fragments in SEAMLESS.

Definition 1 *Suppose that there are given an infinite set of variables, a set of features, and a set of constants. From features, constants attribute variables, and variables, terms are constructed, if features, constants attribute variables, and variables that must be interpreted as functional relations. Let f be a feature symbol, X a variable. A feature graphterm is an expression $f(X, \text{Graphtermlist})$, where Graphtermlist is either the empty list $[],$ or denotes a list of constants, variables and feature graphterms.*

To ensure that graphterms are directed acyclic graphs, further axioms constrain the valid terms. A graphterm algebra, that is term rewriting operations that implement graph operations and canonical forms for classes of isomorphic objects was constructed (cf. [Eus97]). Composed types can be defined using feature graphs and the objects of a type are instantiated feature graphs. The following assertion defines a type *Spec*

$$\text{Spec}([\text{Id} = \text{No}], [\text{Pre}([], [\text{Pre}]), \text{Post}([], [\text{Post}])]) \Leftarrow \\ \text{Conjunction_of_Atom}(\text{Pre}) \wedge \text{Conjunction_of_Atom}(\text{Post}).$$

Each attribute variable of a feature graph may be instantiated by a thuslist of attributes. In the examples above, specifications are assigned the attribute identifier. Attributes don't change the logical semantics of the terms, however, they may be used to design more efficient synthesis methods. It is possible to attach hash values to a graphterm by means of attributes. Objects may be substituted by references to them. An algorithm reference can be an automatically generated counter, e.g., *Alg12345*. Then it refers to an automatically generated object in the knowledge base or it refers to published algorithms, e.g., *Kislitsyn, Aigner* which were constructed outside the synthesis system. The use of references facilitates sharing of terms. It decreases the size of the knowledge base and makes relationships more obvious. The encoding of knowledge is demonstrated by the higher-order formula 4.

$$\text{Know}([], \text{Pre}([], [\text{poset}]), \text{Prog}([\text{Id} = \text{id}, \text{Refid} = \text{Kislitsyn}], []), \\ \text{Post}([], [\text{Select}(i, \text{poset}), \text{Bound}(r..r)], \text{True}) \Leftarrow \\ i = 2 \wedge \text{Forest}(\text{poset}) \wedge \text{CostK}(\text{poset}) = r. \quad (4)$$

The advantage of using graphterms for encoding higher-order formulas is that feature graphterms directly correspond to XML Document Type Definitions (DTDs) and the instantiated ground terms are syntactic variants of XML (cf. [Eus01]). This facilitates the storing and maintenance of structured design knowledge fragments in XML repositories or XML databases, their retrieval based upon semantic annotations, and at the same time enables formal reasoning about them.

5 Higher-order XML-based Document Synthesis

To an increasing degree widely used XML Document Type Definitions like DocBook are being used for the structuring and mark-up of software documentation and scientific publications. Common document structuring elements are, for example, figure, graphics, theorem, proof or enumeration. These XML documents then can be transformed into different target formats like L^AT_EX, XHTML or PDF and provided with a professional layout using freely available tools. Document synthesis is a process that automatically generates a complete structured document. In this section, how to synthesise documents, given a program specification and its proof will be examined.

In order to enable constructive synthesis of documentation, a subset of the `tbook` DTD is used to formalise the types of the theory for document synthesis. The `tbook` DTD was chosen because it is an XML file format that is suitable for scientific texts, but it is also as simple and small as possible, uses similar names to L^AT_EX and it accepts MathML's presentation and contents markup. The `tbook` tools for XML authoring (cf. [Bro05]) may be used to transform the XML document into XHTML, DocBook or L^AT_EX documents. The `tbook` document types are used in the SEAMLESS knowledge base to model document structure. Type definitions are encoded by non-ground facts using feature graphs as knowledge representation format. The simplified document type is sketched in Figure 3. Automatic document synthesis is imple-

| | | |
|------------------------|---|---|
| <i>Document</i> | ≡ | <i>Sequence of Header and Body</i> |
| <i>Body</i> | ≡ | <i>Sequence of Theorems or Lemmata and their Proofs</i> |
| <i>Theorem, Lemma</i> | ≡ | <i>Sequence of Statements or Enumeration of Items</i> |
| <i>Proof</i> | ≡ | <i>Sequence of Statements or Enumeration of Items</i> |
| <i>Item, Statement</i> | ≡ | <i>Natural language sentence, Figure, Reference or Algebraic expression</i> |

Fig. 3. Simplified Document Type

mented by the generic method *Doc_synthesis* which when invoked by a goal $\Leftarrow \text{Doc_synthesis}([\text{spec}, \text{proof}], \text{docterm})$, given a pair $[\text{spec}, \text{proof}]$, causes the instantiation of the metavariable `docterm` by a document term that describes the document structure. Scripts are provided that convert document terms of type *document* into the corresponding XML syntax. The resulting documents

can be processed by the `tbook` tools. The synthesised method is formalised by higher-order predicates which relate programming types and document types. A method which creates the basic document structure is outlined in Figure 4 and need to be augmented by the specifications for the methods *Doc_syn_theorem*, *Doc_syn_proof*, *Doc_syn_lemmata*:

- *Doc_syn_theorem* synthesises theorem content from formal specifications.
- *Doc_syn_proof* generates the proof content and a set of proof cases which shall be treated as lemmata.
- *Doc_syn_lemmata* generates the presentation for the proof cases.

$$\begin{aligned}
 \text{Doc_syn}([\text{spec}, \text{proof}], \text{doc}) &\Leftarrow \text{Doc_header}(\text{header}) \wedge \\
 &\quad \text{Doc_body}([\text{spec}, \text{proof}], \text{body}) \wedge \\
 &\quad \text{doc} = \text{Document}(\text{att}, [\text{header}, \text{body}]). \\
 \text{Doc_body}([\text{spec}, \text{proof}], \text{body}) &\Leftarrow \text{Doc_theorem}(\text{spec}, \text{theo}) \wedge \\
 &\quad \text{Doc_proof}(\text{proof}, \text{doc_proof}, \text{subcases}) \wedge \\
 &\quad \text{Doc_syn_lemmata}(\text{subCases}, \text{seq_lem_proof}) \wedge \\
 &\quad \text{body} = \text{Body}(\text{att}, [\text{theo}, \text{doc_proof}, \text{seq_lem_proof}]).
 \end{aligned}$$

Fig. 4. Skeleton of the document synthesis method

The above synthesis skeleton prescribes the main structure of the document to be synthesised. Specifications are mapped into theorems and programs into proofs. However, the content of these elements needs to be determined in more detail. It requires the partition of the specifications and their proofs into manageable parts, each part should be presented adequately, avoiding low-level details, but compiling all the information which is necessary for the comprehension of a proof part and required to keep the coherence between the parts. Each of the 3 methods *Doc_syn_theorem*, *Doc_syn_proof*, *Doc_syn_lemmata* is realised as a divide-and-conquer strategy. The specifications of the generic methods consist of metarules which define how to decompose the program synthesis fragments and how to compose the document fragments. Synthesis involves various decisions which are based on context-specific knowledge:

- how to decompose the program into subprograms which are represented as one major step;
- when to use graph visualisations or textual explanations for proof parts;
- which parts of the proof graph to split and to treat separately as lemmata.

The knowledge base of the document synthesiser can be augmented by heuristic layout knowledge, e.g., “acceptable” sizes for decision trees. The knowledge base entails parameterised text templates which are associated with corresponding objects of the program synthesis theory. The graph visualisation tool `graphviz` [Res05] is called on demand to generate graph layouts in the desired format.

6 Experimental Results

A previous version of SEAMLESS included generic methods for synthesis and machine learning which were devised independently of the data structure used for knowledge encoding. Domain specific knowledge was provided through an intermediate component as definitions for open generic predicates. The system was used to prove $V_3(22) \leq 28$ which constructed a counter example for a published theorem. Although the time for constructing the algorithm that has verified $V_3(22) \leq 28$ was not exactly measured, the computation had taken several days on a Sun Solaris workstation.

The system has been partially reimplemented by reusing the formally correct generic methods, however, changing the implementation of the data structures and the knowledge representation format. The knowledge representation schema and graph-based data structure which have been devised in this paper have been used to structure and implement the specialised domain knowledge about the selection problem. Indexing techniques for graphs have been introduced to tackle the huge search complexities structure. The results of these changes are

1. a radical improvement of the search complexities;
2. more abstract well-defined metalevel proofs with a higher degree of reuse which is reflected by references and links to objects and theorems;
3. a clearer distinction between informal or formal knowledge which has been derived outside of the system and automatically derived knowledge.

This implementation was used to re-synthesise algorithms for small n and it could be automatically proven that $V_i(n) \leq H_i(n)$, $1 \leq i \leq 6$, $1 \leq n \leq 14$. In addition, the implementation was used to automatically synthesise various new complex algorithms which prove $V_4(21) \leq 30$, $V_4(23) \leq 33$, $V_4(24) \leq 34$, $V_4(25) \leq 35$ and $V_4(26) \leq 26$ and confirm the number-theoretic hypothesis for the selection problem.

The re-implementation was experimentally analysed on an Athlon 3200 64 bit computer with 1 GB memory and on a 1.3 GHZ Intel Centrino laptop with 512 MB memory. In this experimental application scenario, Yap Prolog turned out to be the fastest Prolog compared against Sicstus, GNU, B and SWI Prolog. In Figure 6 some complexity indicators for automated synthesis are collected: 1) a small example whose solution is described in the classic book by Knuth; 2) a more complex example; 3) the previously discovered selection algorithm which took days and now can be handled within seconds; 4) much more complex problems whose solutions improve the known upper bounds and confirm the stated hypothesis. The test environment was as follows: a) Centrino + Yap Prolog, b) Centrino + SWI Prolog, c) Athlon 3200 + Yap Prolog, d) Athlon 3200 + SWI Prolog. Some experimental results are summarised in Figure 5. The CPU time is measured in seconds. During the synthesis process, for each case solution, a canonical form is computed that represents the solution up to isomorphism. The abstracted case solutions can be reused and adapted. The number of generated case solutions for isomorphism classes is given in the last column of the table. A missing entry means that a solution couldn't be constructed within a couple of

hours. It should be noted that these indicators are snapshots of an implementation in progress. Each minor modification may alter the search space. This can decrease or increase the complexities to solve a problem drastically.

| <i>Spec</i> | a | b | c | d | #examined isomorphism classes |
|-------------------|-------|----------|-------|----------|-------------------------------|
| $V_4(7) \leq 10$ | 0.026 | 0.05 | 0.014 | 0.01 | 68 |
| $V_3(22) \leq 28$ | 1.87 | 8.6 | 0.84 | 2.93 | 3,062 |
| $V_4(14) \leq 21$ | 9.0 | 56.0 | 3.3 | 18.1 | 12,122 |
| $V_4(21) \leq 30$ | 52.0 | 996.4 | 19.3 | 321.2 | 61,859 |
| $V_4(23) \leq 33$ | 288.9 | 16,239.7 | 107.1 | 5,210.2 | 277,178 |
| $V_4(24) \leq 34$ | 136.8 | 6,163.3 | 50.4 | 1,951.6 | 127,572 |
| $V_4(25) \leq 35$ | 390.4 | – | 146.5 | 13,034.3 | 362,458 |
| $V_4(26) \leq 36$ | – | – | 525.1 | – | 1,048,665 |

Fig. 5. Complexity indicators for correct search

In all experimental analysed cases, it can be experimentally verified that reuse of derived knowledge from previous proof searches when solving new problems improves the search complexities by up to 25%. Although the system is able to examine around 1,500,000 isomorphic states in less than one half hour and store them in 1 GB memory, the complexities of the selection problem fairly soon exceed the capabilities of the system. The experiments also showed that when using the computers to full capacity, odd runtime errors occur, e.g., uninstantiated variables, which cannot attributed to programming errors.

The reuse and knowledge-based synthesis approach generates proof graphs whose size has been decreased enormously compared to the uniform approaches. However, taking the constructed selection algorithms, in most of the more complex cases, the sizes of the generated decision graphs are still unsatisfying, because they cannot be manually checked with reasonable effort.

The generic synthesis method is based upon a 3-valued logic. Heuristic knowledge marked with the truth value *Maybe* can be provided. In order to decrease the sizes of the proof graphs various incorrect lower bounds were experimentally added to the knowledge base. The simplest form of an incorrect lower bound is a depth restriction. The incorrect lower bounds restrict the search space. Employing these heuristics, an unsuccessful search may yield the result *Maybe*. Using heuristic search a solution for $V_4(24) \leq 34$ could be constructed within 6.5 secs, having examined 13695 isomorphism classes and having used Yap Prolog on the AMD Athlon 3200. The size of this proof graph was also substantially decreased.

The generated graph is still too large to be easily comprehensible, therefore it is embedded in a system-generated documentation which provides a mathematically skilled reader background information about the proofs, such as cases which can be reduced to problems known from the literature, or are solved using the dedicated lemmata. The automatically synthesised documentation of this automatically synthesised algorithm is presented in appendix A.

A major advantage of combining higher-order program synthesis and document synthesis is that the generated proof terms describe programs at the algorithmic abstraction level. There is no need to raise the level of abstraction or to integrate a planning module into the document synthesis component. The synthesised document is a direct mapping from specifications and proof terms. An improved knowledge base of the system directly causes an improved documentation. Comprehensibility depends on the size of the automatically synthesised proof graphs. For small i, n the automated co-synthesis of algorithms and their documentation achieves human-comprehensible documents. For complex problems, synthesis of proof terms of reasonable sizes is an experimental endeavour.

7 Conclusions and Related Work

In [Kre93], it has been stated “We believe that a *formalization of the metatheory of programming* is one of the most important steps towards the development of program synthesisers which are flexible and reliable and whose derivations are both formally correct and comprehensible for human programmers.” As a step towards this objective, this paper has experimentally analysed the interplay between mathematical documents and knowledge-based algorithm synthesis based upon a metatheory for structuring problem solving knowledge. Structuring elements are identified to formalise algorithm design knowledge which is implicit in scientific documents and include abstractions like pre- and postconditions based upon the Floyd-Hoare program logics. Types, higher-order predicates and synthesis methods have been defined. The resulting framework is more fine-grained than related approaches for program synthesis which are based upon higher-order logic or proof planning [Kre93, IS04, RicCA]. The SEAMLESS framework also aims at the formalisation of heuristic problem solving strategies and resource restrictions. The framework devised has been applied to structure comprehensive knowledge relating to the selection problem and encode it for use in a knowledge base. The knowledge base has been used for the synthesis of complexity bounded algorithms using generic methods that were previously devised. Extended graph-terms are used as the core data structure for knowledge encoding. They provide the key for the efficiency of the implementation as they support various optimisation techniques. The huge search complexities are tackled by a combination of indexing techniques, isomorphism abstraction, machine learning and knowledge reuse through references. It was possible to synthesise several new complex selection algorithms which improve known bounds and confirm our hypothesis.

Based upon the higher-order synthesis framework, a method that synthesises human-comprehensible XML-based documents from specifications and constructed proofs has been devised and implemented. The synthesised documents include diagrams and references to reused knowledge. The method has been applied for the automatic synthesis of documentation for automatically synthesised algorithms. It allows logic-based synthesis of documents on a higher abstraction level than a number of current low-level XML-based approaches which aim at the exchange or presentation of arbitrary mathematical documents, Website synthe-

sis or program documentation and merely exploit the correspondences between XML data and logical terms, e.g., [BDHG,LR03,SR01]. The presented framework distinguishes itself from related approaches (cf. [Pec04,GKP96,Oks05]) in which computer have been used for the complexity analysis of unsolved mathematical problems. The results synthesised by SEAMLESS are constructive, comprehensible and can be manually checked for correctness, provided that the sizes of the generated proof graphs are reasonable. In one case, it was possible to synthesise a new complex algorithm of reasonable length, and, thus the synthesised documentation is comprehensible (cf. Appendix A).

Our future work will include experimental work on automated abstraction to decrease the size of the generated algorithms and to get more abstract explanations for series of single proof steps or comprehensive decision trees. The objective is to discover a few rules which describe new correct algorithms for infinitely many n .

References

- [Aig82] M. Aigner. Selecting the top three elements. *Discrete Applied Mathematics*, 4:247–267, 1982.
- [BDHG] Dietmar A. Seipel, Bernd D. Heumesser and Ulrich Güntzer. An expert system for the flexible processing of XML-based mathematical knowledge in a Prolog-environment.
- [BFP⁺73] M. Blum, R.W. Floyd, V. Pratt, R.L. Rivest, and R.E. Tarjan. Time bounds for selection. *J. Comp. Syst. Sci.*, 7:448–461, 1973.
- [BJ85] S.W. Bent and J.W. John. Finding the median requires $2n$ comparisons. In *Proc. 17th ACM Symposium Theory of Computing*, pages 213–216, 1985.
- [Bro05] Torsten Bronger. The tbook system for XML authoring. <http://tbookdtd.sourceforge.net>, 25.9.2005.
- [Dev05] Keith Devlin. Last doubts removed about the proof of the four color theorem. http://www.maa.org/devlin/devlin_01_05.html, January 2005.
- [EN96] J. Eusterbrock and M. Nicolaides. The visualization of constructive proofs by compositional graph layout: A world-wide web interface. *Proc. CADE Visual Reasoning Workshop, Rutgers University*, 1996.
- [Eus85] J. Eusterbrock. Ein rekursiver Ansatz zur Bestimmung der Anzahl von Vergleichen bei kombinatorischen Selektionsproblemen. Diplomarbeit, Universität Dortmund, 1985.
- [Eus92a] J. Eusterbrock. Errata to “Selecting the top three elements” by M. Aigner: A Result of a computer assisted proof search. *Discrete Applied Mathematics*, 41:131–137, 1992.
- [Eus92b] J. Eusterbrock. *Wissensbasierte Verfahren zur Synthese mathematischer Beweise: Eine kombinatorische Anwendung*, volume 10 of *DISKI*, 1992.
- [Eus95] J. Eusterbrock. SEAMLESS: Knowledge based evolutionary system synthesis. *ERCIM News*, 23, October 1995.
- [Eus97] J. Eusterbrock. Canonical term representations of isomorphic transitive DAGs for efficient knowledge-based reasoning. In *Proceedings of the International KRUSE Symposium, Knowledge Retrieval, Use and Storage for Efficiency*, pages 235–249, 1997.

- [Eus01] J. Eusterbrock. Knowledge mediation in the world-wide web based upon labelled dags with attached constraints. *Electronic Transactions on Artificial Intelligence*, 5:"<http://www.ida.liu.se/ext/epa/ej/etai/2001/D>", 2001.
- [FG79] F. Fussenegger and H.N. Gabow. A counting approach to lower bounds for selection problems. *J. Assoc. Comput. Mach.*, 26:227–238, 1979.
- [FR75] R.W. Floyd and R.L. Rivest. Expected time bounds for selection. *Comm. ACM*, 18:165–172, 1975.
- [GKP96] William Gasarch, Wayne Kelly, and William Pugh. Finding the i th largest of n for small i, n . *SIGACT News*, 27(2):88–96, 1996.
- [Hoa69] C.A.R. Hoare. An axiomatic basis for computer programming. *CACM*, 12(10):576–581, 1969.
- [HS69] A. Hadian and M. Sobel. Selecting the t -th largest using binary errorless comparisons. In P. Erdős, A. Renyi, and V.T. Sos, editors, *Combinatorial Theory and its Applications II*, pages 585–600. North Holland, 1969.
- [IS04] A. Ireland and J. Stark. Combining proof plans with partial order planning for imperative program synthesis. *Journal of Automated Software Engineering*, Accepted for publication, 2004.
- [Kis64] S. S. Kislitsyn. On the selection of the k -th element of an ordered set by pairwise comparisons. *Sib. Mat. Z.*, 5:557–564, 1964.
- [Knu73a] D.E. Knuth. *Fundamental algorithms*, volume 1 of *The Art of Computer Programming*. Addison Wesley, Reading, MA, 1973.
- [Knu73b] D.E. Knuth. *Sorting and Searching*, volume 3 of *The Art of Computer Programming*. Addison Wesley, Reading, MA, 1973.
- [Kre93] Chr. Kreitz. Metasynthesis - deriving programs that develop programs. Habilitationsschrift, Technische Hochschule Darmstadt, 1992.
- [Lam91] Clement Lam. The search for a finite projective plane of order 10. *American Mathematical Monthly*, 98:305–318, 1991.
- [LR03] S. Leung and D. Robertson. Automated website synthesis. <http://www.ukuug.org/events/linux2003/papers/leung.pdf>, 2003.
- [MP82] J.I. Munro and P.V. Poblete. A lower bound for determining the median. Technical report, University of Waterloo Research Report CS-82-21, 1982.
- [Oks05] Kenneth Oksanen. Selecting the i th largest of n elements. <http://www.cs.hut.fi/cessu/selection/>, 2005.
- [Pec04] Marcin Pecarski. New results in minimum comparison sorting. *Algorithmica*, 40:133–145, 2004.
- [Res05] AT&T Research. Graphviz - graph visualization software. <http://www.graphviz.org/>, 25.9.2005.
- [RH84] P.V. Ramanan and L. Hyafil. New algorithms for selection. *J. of Alg.*, 1:557–578, 1984.
- [RicCA] C.S Richardson, J.D. Proof planning and program synthesis: a survey. In *Logic-Based Program Synthesis: State-of-the-Art & Future Trends, AAAI 2002 Spring Symposium*, March 25-27, 2002, Stanford University, CA.
- [SPP76] A. Schönhage, M. Paterson, and N. Pippenger. Finding the median. *J. Comp. System Sci.*, 13:184–199, 1976.
- [SR01] Johann Schumann and Peter Robinson. [] or success is not enough: Current technology and future directions in proof presentation. <http://www.cs.bham.ac.uk/mmk/events/ijcar01-future/>, 2001.
- [Yao74] F.F. Yao. On lower bounds for selection problems. Technical report, TR-121, MIT, Project Mac, Cambridge, Mass., 1974.
- [Yap76] C.K. Yap. New upper bounds for selection. *Comm. ACM*, 19:501–508, 1976.

A Automatically Synthesised Documentation¹

Select(4,24)

©Jutta Eusterbrock

theorem 1 $V_4(24) \leq 34$

Proof. Let $KEYS$ be a totally ordered set, $|KEYS| = 24$. The 4 -th largest element of $KEYS$ is computed by Algorithm 1. The computation takes in the worst-case at most 34 comparisons.

algorithm 1

1. Partition the set $KEYS$ into disjoint subsets $|K1| = 16, |K2| = 8$. Determine the maxima of $K1, K2$ by setting-up balanced tournaments. The resulting poset is isomorphic to the poset as shown in Figure 6. For setting-up the balanced tournaments 22 comparisons are needed.

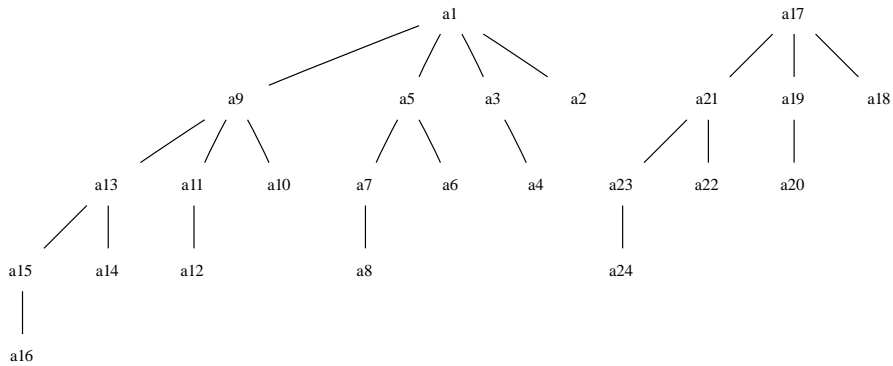


Fig. 6. Balanced Tournaments of Set Partition

¹This is an automatically composed documentation of a new, automatically synthesised algorithm **Select(4, 24)** with the worst-case complexity $V_4(24) \leq 34$. Based on the proof structure derived during the synthesis, the system selected, instantiated, and composed appropriate text templates to generate the explanatory text and document structure. The graphs were visualised by the graphviz system. The proof was checked for correctness by the present author.

2. Perform the comparisons in accordance with the decision graph in Figure 7. The nodes in the decision graph represent:
- Comparisons $X : Y$, shown as circles. The left child node represents the case $X > Y$ and the right one the case $X < Y$.
 - Subgraph place holders, shown by references to a diamond which correspond to solutions which can be obtained by instantiating theorems and algorithms known from published literature. The cases 1, 27, 48, 49, 739, 743, 744, 873, 874, 7772, 7773, 7823, 7824, 7839, 7840 are instances of published theorems.
 - Place holders for calls to simplification functions, especially isomorphism functions, which are denoted by boxes. There are two type of function calls. Firstly, solutions for the simplified subcases are represented by a subgraph which is then referenced. This concerns case 7844. Secondly, in some situations more detailed explanations for the simplified subcases, e.g. lemmata, have been generated. In the given decision graph, the cases 876 and 7775 are handled separately.

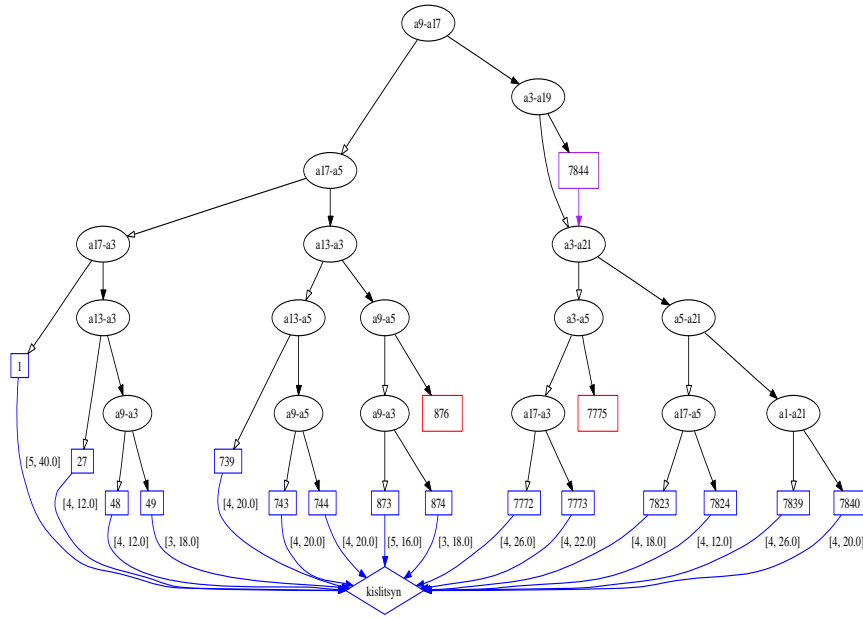


Fig. 7. Decision Tree

3. Generate solutions for specific cases:
- Case 876 is described as follows. Let P be the poset as visualised in Figure 8. It needs to be proven $V_3(P) \leq 8$. The specification is up-to-isomorphism handled by Lemma 1.
 - Case 7775 is described as follows. Let P be the poset as visualised in Figure 9. It needs to be proven $V_3(P) \leq 8$. The specification is up-to-isomorphism handled by Lemma 2.

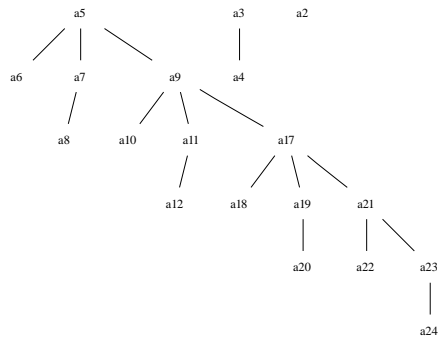


Fig. 8. Poset P

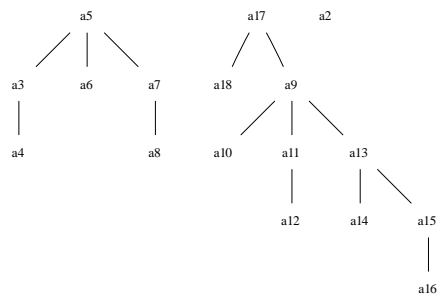


Fig. 9. Poset P

lemma 1 *Let P be a poset as visualised in Figure 10. The 3-rd largest element of P can be computed by at most 8 comparisons.*

Proof. **algorithm 2**

1. Compare a_9 and a_3 .
 - (a) $a_9 > a_3$. a_9 is greater than the 3-rd largest element and is reduced. The resulting poset P satisfies the precondition of the algorithm [Kis64]. Hence, selecting the 2-nd largest element takes at most $f(5,14.0)=7$ comparisons.
 - (b) $a_9 < a_3$. a_3 is greater than the 3-rd largest element and is reduced. The resulting poset P satisfies the precondition of the algorithm [Kis64]. Hence, selecting the 2-nd largest element takes at most $f(3,10.0)=5$ comparisons.

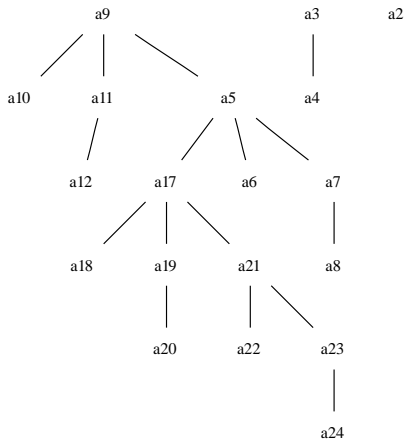


Fig. 10. Poset P

lemma 2 *Let P be a poset as visualised in Figure 11. The 3-rd largest element of P is computed by at most 8 comparisons.*

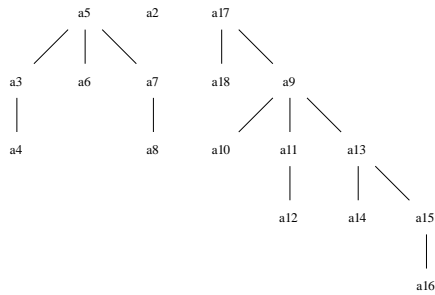


Fig. 11. Poset P

Proof. **algorithm 3**

1. Compare $a5$ and $a17$.
 - (a) $a5 > a17$. $a5$ is greater than the 3-rd largest element and is reduced. The resulting poset P satisfies the precondition of the algorithm [Kis64, kislitsyn]. Hence, selecting the 2-nd largest element takes at most $f(5,10.0)=7$ comparisons.
 - (b) $a5 < a17$. $a17$ is greater than the 3-rd largest element and is reduced. The resulting poset P satisfies the precondition of the algorithm [Kis64]. Hence, selecting the 2-nd largest element takes at most $f(4,18.0)=7$ comparisons.