

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

Jutta Eusterbrock



SEAMLESS Solutions

Empirically Successful Automated Reasoning in Higher-Order Logic

- Given a formal specification of a complex problem
 $\Leftarrow \exists Prog : \{Pre\}Prog\{Post\}$
- instruct your digital proof assistant to synthesise a proof and its documentation in a form which can be checked by domain experts in the same way as scientific publications.

\Rightarrow Is this science fiction?

Motivation

General skepticism of computer-based theorem proving

- Impossible for any expert to verify all the computations
- Various types of errors (algorithms, implementation, runtime, hardware) are possible
- Lack of mathematical abstraction, tedious, cryptic notation

Famous computer-based proofs of theorems

- Non-existence of a **projective plane of order 10** by C. Lam in 1991 (backtrack search, equivalent of 2000 hours on a Cray-1)
- Proof of **four color theorem**
 - 1976 [Appel & Haken], 1936 specific cases, 1200 hours
 - 1996 [Robertson, Sanderson, Seymour, Thomas] 633 cases
 - 2004 [G. Gonthier, Microsoft Research, England, on leave from INRIA], used the CoQ system to verify the proof
 - Nov, 2005 [J. Ferro] purports *short* proofs

SEAMLESS Knowledge-based Program Synthesis

- **Synthesis tasks** are logical expressions containing pre-, postconditions, complexity constraints, and existentially quantified meta-variables which stand for an unknown program
- **Higher-order definitions** $\text{Spec} \equiv \text{Expr}$ define the members of synthesis types. Type **Program** defines *Hoare*-like imperative programs with additional function calls
- **Axiomatisation of higher-order predicates** (**Decompose**, **Generalise**, **Know...**) models generic properties and relations
- **Correct generic synthesis methods** $\text{Synthesis}(\text{pre}, \text{prog}, \text{post})$ are defined in terms of the higher-order predicate as set of rules
- **Domain-specific knowledge** is provided as definitions for the higher-order predicates
- **3-valued logic** to enable reasoning about incomplete knowledge

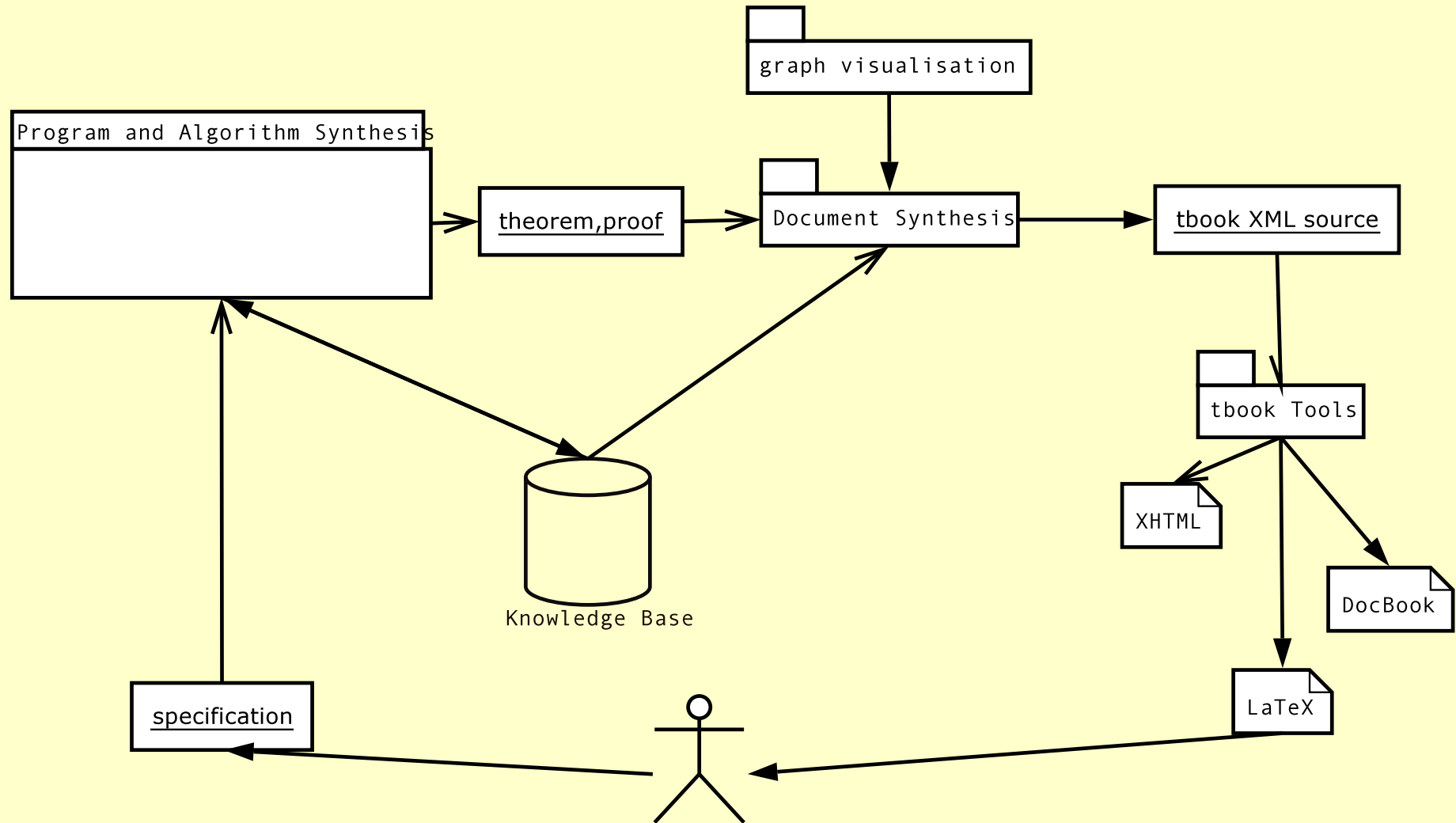
SEAMLESS Knowledge-based Program Synthesis

- Learning from failure and success: isomorphic case solutions are unified and stored in a knowledge base for reuse
- Application domains: *Selection Problem, Configuration*

Re-implementation

- Refinement of feature graphterm logical data structures to improve efficiency
- XML interface
- Incorporation of a document synthesis component that synthesises structured scientific documents (LateX, XHTML), including text, graphics and references, similar to textbook descriptions

SEAMLESS for Knowledge-based Synthesis



eXtensible Markup Language (XML)

- **XML document** is a nested element
< tag attributename = ' value ' > contents < /tag > and *contents* may consist of text or list of elements
- fast emerging **W3C standard** for describing and interchanging data and documents among various systems and databases on the Internet.
- offers the **Document Type Definition (DTD)** as a formalism for defining the syntax and structure of XML documents.
- widely available **tool support** for XML data and document management, processing, type validation, transformation, storing, querying and layout

Specification Language: Feature graphterms

SEAMLESS Specification language

- Horn clauses with feature graphterms as arguments
- Higher-order type definitions using feature graphterms

Definition

- Let F be a feature symbol, x an attribute 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. A feature graphterm is **wellformed** if the term represents a DAG.

Practical considerations

- Attributes have no logical semantics: *identifier, hash value*
- Concrete subterms can be substituted by **references**

Feature graphterms and XML

- Definition of the type *Spec* using feature graphterms

$$\text{Spec}([\text{Id} = \text{no}], [\text{Pre}([], \text{pre}), \text{Prog}([], \text{prog}), \text{Post}([], \text{post})]) \Leftarrow \\ \text{Pre}([], \text{pre}) \wedge \text{Prog}([], \text{prog}) \wedge \text{Post}([], \text{post})$$

...

$$\text{Post}([], [\text{select}(\text{I}, \text{N})]) \Leftarrow \text{Integer}(\text{I}) \wedge \text{Integer}(\text{N})$$

- Substituted feature graphterms which satisfy a type definition define its members, ie., synthesis objects

$$\text{Spec}([\text{specid}=0], [\text{Pre}([], [\text{true}]), \\ \text{Prog}([], [\text{skip}]), \text{Post}([], [\text{true}])])$$

- Synthesis objects can be easily converted into XML syntax

```
<spec specid=0><pre> true </pre>
  <prog> skip </prog>
  <post> true </post></spec>
```

Synthesis of `tbook` XML Documents

`tbook` XML documents

- consist of text, graphics and structural elements like *sections*, *theorems*, *proof*, *figures*, *formulae*, *references*
- can be automatically transformed into different document formats (LateX, docBook) using the `tbook` tools

Document synthesis method

is devised by defining

1. a higher-order type `docterm` by translating the `tbook` DTD into definitions feature graph types
2. a generic method `Doc_synthesis(+spec, -docterm)` that may be used to construct corresponding document terms for formal specifications

Synthesis of tbook XML Documents

Logical Document Types

Semi-formal description

Document \equiv *Sequence of Header and Body*

Body \equiv *Sequence of Theorems or Lemmata and their Proofs*

Theorem, Lemma \equiv *Sequence of Statements or Enumeration of Items*

Proof \equiv *Sequence of Statements or Enumeration of Items*

Item, Statement \equiv *Natural language sentence, Figure,
Reference or Algebraic expression*

a bit more formal type description

$$\text{Docterm}(\text{att}, [\text{Header}(\text{att}, \text{header}), \text{Body}(\text{att}, \text{body})]) \Leftarrow \\ \text{Header}(\text{att}, \text{header}) \wedge \text{Body}(\text{att}, \text{body})$$

Synthesising tbook XML Documents

Goal $\Leftarrow \text{Doc_synthesis}(\text{spec}, \text{docterm})$

Divide-and-Conquer Method

- Program specifications are decomposed into their atomic pieces such as formulae
- Each specification piece is related to a document fragment
- The document fragments are assembled into a document term

$\text{Doc_synthesis}(\text{spec}, \text{docterm}) \Leftarrow$

$\text{spec} = \text{Spec}([\text{Id} = \text{no}], [\text{Pre}([], \text{pre}), \text{Prog}([], \text{prog}), \text{Post}([], \text{post})]) \wedge$

$\text{Doc_synthesis}(\text{Pre}([], \text{pre}), \text{Post}([], \text{post}), \text{doctheorem}) \wedge$

$\text{Doc_synthesis}(\text{Prog}([], \text{prog}), \text{docproof}) \wedge$

$\text{Compose}(\text{doctheorem}, \text{docproof}, \text{docterm})$

Synthesising tbook XML Documents

Context-specific Rules

- 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

Knowledge Base

- parameterised text templates for corresponding program types
- visualisations which are generated by graphviz

Challenging Problem: The Selection Problem

Selection problem : finding the i -th largest element, given a set of n distinct unordered numbers, $1 < i < n$. The worst-case, minimum number of comparisons is denoted by $V_i(n)$.

presented in the classic book of D. Knuth. *The Art of Computer Programming 3. Sorting and Searching*.

The present author constructed the formula $H_i(n)$

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

and proved

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

Challenging Problem: The Selection Problem

It was shown that the numbers $H_i(n)$:

- match the exact numbers $V_i(n)$ for $i = 1, 2, 3$
- are equal to or less than the lower bounds
- are equal or greater than the upper bounds
- coincide with the conjecture of Yao and the conjecture of Paterson $V_{n/2} \approx 2.4094n$ for the median problem

It was proven

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

Conjecture $V_i(n) = H_i(n)$ for all i, n

Challenging Problem: The Selection Problem

Computer-assisted optimum selection

- Constructive algorithm synthesis [Eusterbrock, 1992] proved $V_3(22) < H_3(22)$ & is a counterexample to a published theorem
- Intelligent searches [Gasarch, Kelly, Pugh, 1996] independently reconfirmed $H_i(n) = V_i(n)$, $n \leq 12$, ($V_4(11) = 17$, 4116.7 secs) non-optimal upper & lower bounds by heuristic search
- Lower bounds and algorithms [Oksanen, 2005] by intelligent search for $n \leq 15$. Decision graph $i = 7, n = 14 \geq 600$ nodes. Computer result $V_5(12) \geq 19$ conflicts with Lemma $V_5(12) \leq 18$

Computer-assisted optimum sorting

- optimal lower bounds $S(n)$ for sorting n elements, $n = 13, 14, 22$. $S(22) > 70 = 1740$ hours on a computer with a 650 MHz processor based upon intelligent search [M. Peczarski, 2004]

Challenging Problem: The Selection Problem

Formalisation as program synthesis task \Rightarrow

New experimental SEAMLESS results

- higher-order approach decreases the proof sizes substantially
- for small i, n optimal algorithms and human comprehensible scientific documentations can be synthesised automatically
- several new selection algorithms were synthesised that “prove”
 $V_4(n) \leq H_4(n), n = 21, 23, 24, 25, 26$
- generally, the sizes of the generated proof graphs for more complex problems are too huge and grow exponentially
- a new human-verified algorithm of reasonable size that proves $V_4(24) \leq H_4(24)$ and its scientific documentation could be synthesised (6.5 secs) using heuristics for search space reduction

Challenging Problem: The Selection Problem

Complexity indicators for complete search

<i>Spec</i>	a	b	c	d	#non-isomorphic cases
$V_4(7) \leq 10$	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

CPU time in secs: a) Centrino (512MB) + Yap Prolog, b) Centrino + SWI, c) Athlon 3200 (1GB) + Yap, d) Athlon 3200 + SWI.

Challenging Problem: The Selection Problem

New experimental SEAMLESS results

Computation times

- have been reduced by several orders of magnitude: for example $V_4(11) = H_4(11)$ 1.2 secs compared against 4116.7 secs
- on different platforms may differ by the factor 150

Resource usage constraint for computation is memory consumption

- up-to 1,5 million non-isomorphic case solutions can be stored in the memory and reused (AMD Athlon 3200 1GB)

Examples of automatically synthesised algorithms →
SEAMLESS V424

Conclusions

- **Higher-order knowledge-based synthesis** yields abstract constructive proofs which can be fully automatically transformed into document structure and content
- **Feature graphterms** provide a data structure that facilitates higher-order reasoning, conversion into XML syntax and efficiency
- **Experiments** have shown that it is possible to tackle the search complexities for construction of solutions for challenging problems and to generate proofs of reasonable size
- Size of the generated proofs is exponentially growing
- Future work will examine machine learning for automated abstraction of generated proofs