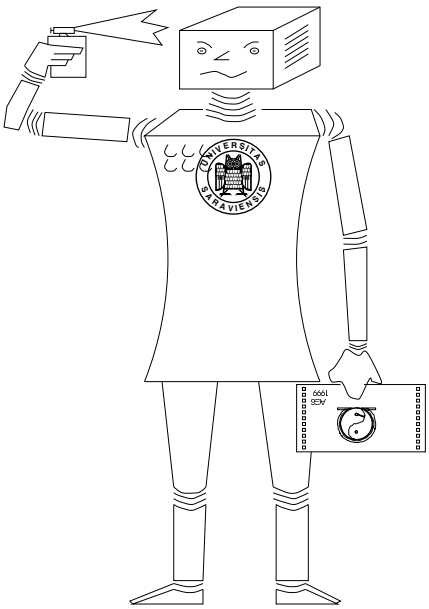
UNIVERSITÄT DES SAARLANDES
FACHBEREICH INFORMATIK
D-66041 SAARBRÜCKEN
GERMANY
WWW: http://www.ags.uni-sb.de/

SEKI Report
ISSN 1437-4447



CALCULEMUS Autumn School 2002: Student Poster Abstracts

Jürgen Zimmer and Christoph Benzmüller (Eds.)

{jzimmer|chris}@ags.uni-sb.de

FR 6.2 Informatik, Universität des Saarlandes, 66041 Saarbrücken, Germany

SEKI Report SR-02-06

CALCULEMUS Autumn School 2002: Student Poster Abstracts

Jürgen Zimmer and Christoph Benzmüller (Eds.)

Fachbereich Informatik
Universität des Saarlandes, Saarbrücken, Germany
{jzimmer|chris}@ags.uni-sb.de

test [1] foo

References

[1] Jürgen Zimmer and Christoph Benzmüller (Eds.). CALCULEMUS Autumn School 2002: Student Poster Abstracts. SEKI Technical Report SR-02-06, Fachbereich Informatik, Universität des Saarlandes, Saarbrücken, Germany, 2002.

CALCULEMUS Autumn School 2002: Student Poster Abstracts

Jürgen Zimmer and Christoph Benzmüller (Eds.)

Fachbereich Informatik
Universität des Saarlandes, Saarbrücken, Germany
{jzimmer|chris}@ags.uni-sb.de

test [1] foo

References

[1] Jürgen Zimmer and Christoph Benzmüller (Eds.). CALCULEMUS Autumn School 2002: Student Poster Abstracts. SEKI Technical Report SR-02-06, Fachbereich Informatik, Universität des Saarlandes, Saarbrücken, Germany, 2002.

CALCULEMUS Autumn School 2002: Student Poster Abstracts

Jürgen Zimmer and Christoph Benzmüller (Eds.)

Fachbereich Informatik
Universität des Saarlandes, Saarbrücken, Germany
{jzimmer|chris}@ags.uni-sb.de

Table of Contents

Synthesis of Object Models from Partial Models

Marco Alberti
Dipartimento di Ingegneria, Università degli Studi di Ferrara
Via Saragat, 1 - 44100 Ferrara (Italy)
telephone: +39 0532 974894 - email: malberti@ing.unife.it

1 Introduction

Model-based Object Recognition is one of the key problems in Computer Vision. One of the most widely accepted definitions of the problem is:

"Given a model of an object, and data in which the object may be present, discover whether, and where, the object is present in the data".

This definition assumes the model itself of the object as part of the data of the problem. In those cases in which, for some reason, the availability of the model is not easy to achieve, an automatic model construction procedure might be useful.

In practical cases, the representation of model of the object to be recognized is strongly dependent on the nature of data used for recognition. For example, for visual search in range images a representation of objects in terms of surfaces and relations among them is often the most suitable. Most of the automatic model construction methods proposed in literature are designed to fit a particular class of models [6, 7].

In this paper an approach to automatic model construction is presented which makes extensive use of the Constraint Satisfaction Problems paradigm, exploiting its generality to achieve a wider range of applicability. In fact, the concepts the work is based upon are not even intrinsically linked to Computer Vision: no strict assumptions have been made about the nature of the primitives used to describe the models, and the relationships among them.

This is made possible by assuming the data of the model construction problem to be descriptions of parts of the object, of the same nature of that of the resulting

complete model, which is built by a synthesis process. With this assumption, it is possible to treat the primitives used for model description as parameters, so they do not affect the algorithms here proposed directly. In other words, the problem here formulated and, to some extent, solved, is that of building the model of an object abstracting from the peculiar features of its representation.

The partial models (in the following, views) are expressed as CSPs. This choice allows for very general and flexible representations, which can take advantage of the extensive research work on CSPs of the recent years (also applied to Computer Vision: see, for instance, [5]). The resulting model is, as well, a CSP, obtained by establishing a correspondence between its elements and the views' elements, so to maintain the relations among elements.

The model synthesis problem is, itself, stated as a CSP.

For the problem solution, two algorithms have been implemented, and tested in a common problem in Computer Vision.

This paper is meant to only give a quick survey of the approach: further details can be found in [3].

2 Problem representation

To achieve higher generality, the concepts involved in the problem formulation are defined at the highest possible level of abstraction, avoiding mentioning a priori hypotheses about the kind of primitives chosen for the model representation.

2.1 (partial) Model representation

A (partial) object model is defined as composed by:

- *Elements*: a finite set of entities which are considered atomic at the chosen level of abstraction;
- *Attributes*: meant to represent features of n-ples of elements, attributes are functions which have an n-ple of elements as argument and an m-dimensional vector of values (where m is the *dimension* of the attribute). Value's components are bounded open intervals in $\Re$ for *real* attributes (so to take into account possible representation/measurement errors) and elements of a discrete set for *discrete* attributes.

- *Relations*: describing relations among elements, relations are defined in the usual mathematical ways, i.e., an n-ary relation is a subset of E^n, if E is the elements set.

Given such representation, it is straightforward to express the model as a CSP, where E is the set of variables, A and R define constraints respectively over and among elements of E.

2.2 Labeling

A *labeling* function defines the correspondence between a view of an object and a (complete) model. For any given element of a view the value of a *labeling* function is the correspondent element of the model.

Among the possible labeling functions, the only relevant for the purposes here considered are those which do not raise contraddictions in the model: for example, if the views' elements are surfaces, two elements having different values of the unary *shape* attribute (say, triangle and square) cannot be labeled with the same model element.

Having the notions of compatibility for attributes and relations been defined, it is possible to define a *compatible labeling* as a labeling of a view's elements with model elements such that all attributes' values and relations of the view's elements and of the labeling's image are compatible.

If there exists a compatible labeling of a given view with a given model, the view is said to be a *view of* the model.

Notice that compatibility constraints imply that only view-invariant attributes and relations can be used for views and model descriptions.

2.3 Problem statement and solutions

The problem considered here can thus be stated as:

"Given a set of views of an object, find a model such that all of the views are *view of* the model".

In the following, we will refer to this as a *model synthesis* problem.

A solution to the problem is defined by both the model and the labeling functions for the views.

For any model synthesis problem, there is at least one trivial solution, obtained by the juxtaposition of the views. This is achieved by constructing a model such

that each element of each view is labeled with a distinguished element of the model. Obviously, this solution is of no use (and no interest).

It makes more sense to search for solutions by performing a better synthesis of views, i.e., by labeling elements of different views with the same element of the model, keeping the *view of* relation between the views and the model valid.

The set of all the possible solutions is quite huge for practical problems. However, we identify two classes of especially significant solutions among them: *minimal solutions*, whose model has the minimal cardinality compatible with compatibility constraints; and *correct solutions*, whose labeling is *coherent*, i.e., if two elements of different views represent the same entity of the object, then they are labeled with the same element of the model and vice-versa.

3 Searching for solutions

For solving the synthesis problem, two algorithms have been developed.

The first, implemented in SWI-Prolog [1], is designed to search for minimal solutions and is based on a *standard backtracking* approach. For limiting its computational complexity to a tractable size, it has been necessary to devise a trade-off which does not guarantee minimality, but has proved satisfactory in the tests which have been run.

The second, implemented using the CHR [4] library of SICStus Prolog [2], follows a constraint-based approach, using compatibility constraints to build the labeling functions and exploiting a configurable heuristic procedure to guide the choices. The experimental results of this algorithm (for both correctness of solutions and performance) are heavily influenced by the quality of the information available as data.

However, it is worth to remind that both of the algorithms find actual solutions of the synthesis problem: thus, if the set of views taken as data is representative enough, all of the synthesized models are acceptable for Object Recognition applications.

Acknowledgments

I want to thank my laurea thesis advisor and now Ph.D. tutor, prof. Evelina Lamma, for her continuous help and guidance at all the stages of my work.

References

[1] SWI-Prolog Home Page, http://www.swi-prolog.org.

[2] SICStus Prolog Home Page, http://www.sics.se/sicstus.

[3] Marco Alberti and Evelina Lamma, 'Synthesis of Object Models from Partial Models: a CSP Perspetive', in *ECAI*, (2002). (to appear).

[4] Thom Frühwirth, 'Theory and Practice of Constraint Handling Rules', *Journal of Logic Programming, Special Issue on Constraint Logic Programming*, **37**(1-3), 95–138, (October 1998).

[5] Evelina Lamma, Paola Mello, Michela Milano, Rita Cucchiara, Marco Gavanelli, and Massimo Piccardi, 'Constraint Propagation and Value Acquisition: Why we should do it Interactively', in *IJCAI*, pp. 468–477, (1999).

[6] A.R. Pope and D.G. Lowe, 'Learning Object Recognition Models from Images', in *ICCV93*, pp. 296–301, (1993).

[7] K. Wu and M.D. Levine, 'Recovering Parametric Geons from Multiview Range Data', in *CVPR94*, pp. 159–166, (1994).

NUMERATION SYSTEM BASED ON GENERALIZED GOLDEN MEAN

Petr Ambrož, Edita Pelantová
Czech Technical University, Faculty of Nuclear Science and Physical Engineering
Trojanova 13, 120 00 Prague 2, Czech Republic
e-mail: ambry@linux.fjfi.cvut.cz

This work engage in arithmetic operation in numeration system with irrational base β, where β is the cubic golden mean, i.e. real root of equation $x^3 = x^2 + x + 1$, it is approximately 1.8392. Arithmetic with irrational base $\beta > 1$ yields interesting phenomenons, which cannot be observed in usual systems with natural base.

Basic term, which we will be working with, is the β-expansion. A β-expansion of a number $x > 0$ is its representation in the form of an infinite sequence $x = x_k\beta^k + x_{k-1}\beta^{k-1} + \ldots + x_1\beta + x_0 + x_{-1}\beta^{-1} + \ldots$, which can be computed by the greedy algorithm. Coeficients in the β-expansion obtained by the greedy algorithm are integers from the set $A = \{0, 1, \ldots, [\beta]\}$. Real numbers having a zero fractional part in their β-expansion are called β-integers. The set of all β-integers is denoted by $\mathbf{Z}_\beta$.

Except from ordinary β-expansion, we define for number 1 one more representation in powers of β – the so called Rényi development of number 1 [RN]. This expansion significantly influences properties of the set $\mathbf{Z}_\beta$ – while for $\beta \in \mathbf{N}, \beta \geq 2$, $\mathbf{Z}_\beta$ is equal to the set of all integers $\mathbf{Z}$, for $\beta \notin \mathbf{N}$ are sets $\mathbf{Z}_\beta$ and $\mathbf{Z}$ different.

If we inspect the distances between neighbours in the set $\mathbf{Z}_\beta$, we will find out that for most values of β the set $\{x \mid x \text{ is distance between nighbours in } \mathbf{Z}_\beta\}$ is infinite. Values of β having finite set of distances between neighbours in $\mathbf{Z}_\beta$ are called "β-numbers". A $\beta \in \mathbf{R}, \beta > 1$ is a β-number if and only if its Rényi's development of 1 is eventually periodic. Every β-number is an algebraic integer.

There is another difference between natural and irrational base: in a system with natural base every combination of admissible digis forms a β-expansion, whereas for the case of irrational base there are substrings which are not allowed to appear in any β-expansion. These forbidden combinations of digits can be simply characterized just by means of Rényi's development of 1.

The most studied "β-number" is the golden mean $\tau = \frac{1+\sqrt{5}}{2}$. Properties of $\mathbf{Z}_\tau$ are well described because $\mathbf{Z}_\tau$ can be equivalently defined by projection of points of $\mathbf{Z}^2$ lying between two parallel lines with slope τ. The coeficients in a

τ-expansion of arbitrary $x > 0$ are 0 and 1; substring 11 is forbidden. There are two possible distances between neighbours in the set $\mathbb{Z}_\tau$: $A = 1$ and $B = 1/\tau$, moreover the infinite sequence of distances in $\mathbb{Z}_\tau$ can be simply generated by substitution rules $A \mapsto AB$, $B \mapsto A$, starting from A.

In contrary to $\mathbb{Z}$, the set $\mathbb{Z}_\beta$ for $\beta \notin \mathbb{N}$ is not closed under addition, subtraction and multiplication, the result of these operations does not even need to have finite number of fractional digits. It is still open problem to describe the set of all numbers β, for which the result of addition, subtraction and multiplication of every pair of β-integers is finite. In [FS] it is shown that the greatest real root of the equation $y^n = a_{n-1}y^{n-1} + \ldots + a_1 y + a_0$, where $a_{n-1} \geq a_{n-2} \geq \ldots \geq a_1 \geq a_0 \geq 1$ belongs among these β. The golden mean as well as the cubic golden mean are roots of equations of this type.

When we perform arithmetic operations in these numeration systems by computer, we will need to know how much memory it will take to store the result, i.e. the maximal number of fractional digits that can arise. For addition and multiplication we denote these maxima by $L_\oplus(\beta)$ and $L_\odot(\beta)$ respectively.

Values $L_\oplus(\beta)$, $L_\odot(\beta)$ are known for some quadratic irrationalities [GMP]:

$L_\oplus(\beta)$	$L_\odot(\beta)$	β
1	1	β root of $x^2 = mx - 1$, where $m \in \mathbb{N}$
2	2	β root of $x^2 = mx + 1$, where $m \in \mathbb{N}$
2m		β root of $x^2 = mx + m$, where $m \in \mathbb{N}$
$\lceil \frac{m+1}{m-n+1} \rceil$		β root of $x^2 = mx + n$, where $m, n \in \mathbb{N}$, $m > n$

The aim of this work is to describe the set $\mathbb{Z}_\beta$ for β – cubic golden mean and to find out values of $L_\oplus(\beta)$, $L_\odot(\beta)$. From now on the symbol β will stand for the cubic golden mean.

First of all we will mention some properties of β-integers. Digits in the β-expansion of an arbitrary number are 0 and 1, the forbidden substring of digits is 111. That is why a real number x is a β-integer if and only if its absolute value $|x|$ can be written in the form $|x| = \sum_{i=0}^{n} a_i b^i$, where $a_0, a_1, \ldots, a_n \in \{0, 1\}$ and the product $a_i a_{i+1} a_{i+2} = 0$ for $\forall i = 0, 1, \ldots, n-2$. Several smallest elements of the set $\mathbb{Z}_\beta$ are drawn on the Figure 1.

The distances between neighbouring elements of the set $\mathbb{Z}_\beta$ take only three values: $A = 1$, $B = \beta - 1$, $C = 1/\beta$. The sequence of gaps in $\mathbb{Z}_\beta$ can be interpreted as an infinite word in the alphabet $\{A, B, C\}$. It is interesting to mention that this word is invariant under substitution

$$A \mapsto AB \qquad B \mapsto AC \qquad C \mapsto A.$$

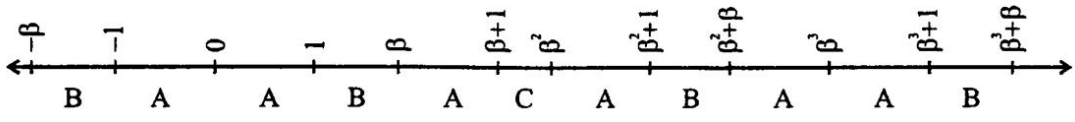


Figure 1: Several elements of the set $\mathbb{Z}_\beta$

An example which shows that $\mathbb{Z}_\beta$ is not closed under addition is the β expansion of $2 = 1 + 1 = 10.001$.

We have investigated two methods to obtain the upper bounds for $L_\oplus(\beta)$ and $L_\odot(\beta)$. Because $\beta^3 = \beta^2 + \beta + 1$, we can write also $\beta^4 = 2\beta^2 + 2\beta + 1$ and similarly an arbitrary power β^k can be represented in the form

$$\beta^k = F_k\beta^2 + G_k\beta + H_k, \quad \text{where} \quad F_k, G_k, H_k \in \mathbb{Z}. \tag{1}$$

It means, that an arbitrary β-integer can be written as an integer combination of $1, \beta$, and β^2, thus

$$\mathbb{Z}_\beta \subset \mathbb{Z}[\beta] := \{a + b\beta + c\beta^2 \mid a, b, c \in \mathbb{Z}\}. \tag{2}$$

$\mathbb{Z}[\beta]$ is a ring, because it is closed under addition and multiplication. Since $\mathbb{Z}[\beta]$ is dense in $\mathbb{R}$, we have $\mathbb{Z}_\beta \subsetneq \mathbb{Z}[\beta]$. Let us note that $\mathbb{Z}[\beta]$ can be obtained by the projection of the lattice $\mathbb{Z}^3$ on a properly chosen straight line in $\mathbb{R}^3$.

We denote the other roots of equation $x^3 = x^2 + x + 1$ by β' and β''. They are in modulus less than 1 and $\overline{\beta'} = \beta''$. The mapping $y = a + b\beta + c\beta^2 \mapsto y' = a + b\beta' + c\beta'^2$ for $a, b, c \in \mathbb{Q}$ is the isomorphism between the fileds $\mathbb{Q}[\beta]$ and $\mathbb{Q}[\beta']$.

For obtaining the upper bound on $L_\oplus$ and $L_\odot$ the following statement was crucial:

Theorem 1. *If $x \in \mathbb{Z}_\beta$, then*

$$|x'| \le h := \frac{1}{1 - |\beta'|^3}. \tag{3}$$

Let $x, y \in \mathbb{Z}_\beta$ be β-integers, such that $x + y > 0$ and let us define $z := \max\{w \in \mathbb{Z}_\beta \mid w \le x + y\}$. Since the biggest gap in $\mathbb{Z}_\beta$ is 1, we may rewrite $x + y = z + r$, where r (the fractional part of the result of $x + y$) satisfies $r \in \mathbb{Z}[\beta]$, $0 \le r < 1$. We have $r = x + y - z$ and thus according to Theorem 1, $|r'| = |x' + y' - z'| \le |x'| + |y'| + |z'| \le 3h$.

Note that the set $\mathcal{R} := \{r = a + b\beta + c\beta^2 \mid a, b, c \in \mathbb{Z}, 0 < r < 1, |r'| < 3h\}$ is finite. Explicit listing shows that the β-expansion of any $r \in \mathcal{R}$ has no more than six fractional digits.

The elements of $\mathcal{R}$ are only candidates for the fractional part of a sum on $\mathbb{Z}_\beta$. We obtain the estimate $L_\oplus \leq 6$. Similar method was used to get the estimate $L_\odot \leq 6$.

In order to explain the second method of estimation, we have to point out the properties of the sequence $(F_k)_0^\infty$ defined by equation (1). It is easy to see that $F_0 = F_1 = 0$, $F_2 = 1$ and $F_{k+3} = F_{k+2} + F_{k+1} + F_k$ for any $k = 0, 1, 2, \ldots$. Numbers F_k are sometimes called Tribonacci numbers. Every natural number m can be expressed as a combination of Tribonacci numbers $m = \sum_{i=3}^{n} a_i F_i$, where $a_i \in \{0, 1\}$ and the product $a_i a_{i+1} a_{i+2} = 0$ for $\forall i = 3, 4, \ldots, n-2$. This fact enables us to prove another important theorem:

Theorem 2. *For any $u \in \mathbb{Z}[\beta]$ there exist $z \in \mathbb{Z}_\beta$, $K, L \in \mathbb{Z}$ such that $u = z + K + L\beta$.*

In particular, any sum of β-integers x, y can be written as $x+y = z+K+L\beta$. Using the isomorphism mapping and Theorem 1 one obtains the relation

$$|K + L\beta'| = |x' + y' - z'| \leq 3h. \tag{4}$$

Again, the set of $K + L\beta$ satisfying (4) is finite. This implies that there exists a finite set F such that $\mathbb{Z}_\beta + \mathbb{Z}_\beta \subset \mathbb{Z}_\beta + F$, i.e. $\mathbb{Z}_\beta + \mathbb{Z}_\beta$ can be covered by a finite number of shifted copies of $\mathbb{Z}_\beta$. Inspecting the β-expansions of elements in F, we have deduced $L_\oplus \leq 8$ and similarly $L_\odot \leq 8$. We conclude that the first method provides better estimate on $L_\oplus$ and $L_\odot$.

In order to obtain the lower bounds on $L_\oplus$ and $L_\odot$ it is sufficient to find suitable sum/product of β-integers. For example

$1001011010 + 1001011011 = 10100100100.10101$

$1101001001011101 * 1101001001011101 = 1100100010001000010010011011011.0011$

We can conclude

$$5 \leq L_\oplus \leq 6, \qquad 4 \leq L_\odot \leq 6.$$

Note that both methods of upper estimation can be used for an arbitrary Pisot number β, i.e. an algebraic integer with Galois conjugates in modulus less than 1.

References

[FS] Ch. Frougny and B. Solomyak, *Finite beta expansions*, Ergod. Th. & Dynam. Sys., **12** (1992), 713–723.

[GMP] L. S. Guimond, Z. Masáková, E. Pelantová, *Arithmetics on beta expansions*, preprint FNSPE, 2001.

[RN] A. Rényi, *Representations for Real Numbers and their Ergodic Properties*, Acta Math. Acad. Sci. Hungary, **8** (1957), 477–493

Mechanizing proofs in Homological Algebra*

J. Aransay† C. Ballarin‡ J. Rubio†

Abstract

In recent years, there has been an increasing interest in Symbolic Computation in Homological Algebra and Algebraic Topology. An essential tool for the design of algorithms in this field is known as Basic Perturbation Lemma (or BPL, in short). We report here on our project of constructing a mechanized proof of the BPL, by means of the Isabelle tactical theorem prover.

1 Introduction

Kenzo [2] is a software system, written under the direction of Sergeraert, for Symbolic Computation in Algebraic Topology and Homological Algebra. Kenzo has a great computational power that has allowed to obtain some results (specifically, homology groups) which had never been determined before, using neither theoretical nor computational methods. The basis for this success is the intensive use of functional programming techniques, which enable in particular to encode and handle at runtime the infinite data structures appearing in the algorithms in Algebraic Topology (see [8, 9]).

In order to increase the reliability of the system, a project was undertaken to formally analyze fragments of the program (see some first results related to algebraic specification of data structures in [4]). As a part of this more general plan, we search for *certified* versions of some crucial fragments of the program.

*Partially supported by DGES PB98-1621-C02-01

†Departamento de Matemáticas y Computación. Universidad de La Rioja. Edificio Vives. Calle Luis de Ulloa s/n. 26004 Logroño (La Rioja, Spain). {jesus-maria.aransay,julio.rubio}@dmc.unirioja.es

‡Institut für Informatik. Technische Universität München. 80290 München (Germany). ballarin@in.tum.de

Due to the importance of functional programming, the tactical theorem prover Isabelle [7, 6] was chosen as a tool to mechanize our approach. As a first task, we focus on the Basic Perturbation Lemma, a result from homological algebra which is presented in the following section, and more exactly, on a previous lemma presented in section 4, an important step in the way to a full proof of the BPL (for fundamentals on homological algebra see, for instance, [8]).

2 The Basic Perturbation Lemma

Definition 2.1. A *reduction* $D_* \Rightarrow C_*$ between two chain complexes is a triple (f, g, h) where: (a) The components f and g are chain complex morphisms $f : D_* \to C_*$ and $g : C_* \to D_*$; (b) The component h is a homotopy operator $h : D_* \to D_*$ (degree 1); (c) The following relations are satisfied: (1) $fg = \mathrm{id}_{C_*}$; (2) $gf + d_{D_*}h + hd_{D_*} = \mathrm{id}_{D_*}$; (3) $fh = 0$; (4) $hg = 0$; (5) $hh = 0$.

Definition 2.2. Let D_* be a chain complex. A *perturbation* of the differential d_{D_*} is a morphism of graded modules $\delta_{D_*} : D_* \to D_*$ such that $d_{D_*} + \delta_{D_*}$ is a differential for the underlying graded module of D_*. A perturbation δ_{D_*} of d_{D_*} satisfies the *nilpotency condition* with respect to a reduction $(f, g, h) : D_* \Rightarrow C_*$ if the composition $\delta_{D_*} \circ h$ is pointwise nilpotent, that is, $(\delta_{D_*} \circ h)^n(x) = 0$ for an $n \in \mathbb{N}$ depending on x.

Theorem 2.3. Basic Perturbation Lemma — *Let $(f, g, h) : D_* \Rightarrow C_*$ be a chain complex reduction and $\delta_{D_*} : D_* \to D_*$ a* perturbation *of the differential d_{D_*} satisfying the nilpotency condition. Then a new reduction $(f', g', h') : D'_* \Rightarrow C'_*$ can be obtained where the underlying graded modules of D_* and D'_* (resp. C_* and C'_*) are the same, but the differentials are perturbed: $d_{D'_*} = d_{D_*} + \delta_{D_*}, d_{C'_*} = d_{C_*} + \delta_{C_*}$, and $\delta_{C_*} = f\phi\delta_{D_*}g$; $f' = f\phi$; $g' = (1 - h\phi\delta_{D_*})g$; $h' = h\phi$, where $\phi = \sum_{i=0}^{\infty}(-1)^i(\delta_{D_*} \circ h)^i$.*

Since the statement of the BPL in modern terms [1], plenty of proofs have been described in the literature. We are interested in a proof due to Sergeraert [8]. This proof is separated in two parts.

Part 1. Let ψ be $\sum_{i=0}^{\infty}(-1)^i(h \circ \delta_{D_*})^i$. From the BPL hypothesis, the following equations are proved: $\phi h = h\psi$; $\delta_{D_*}\phi = \psi\delta_{D_*}$; $\phi = 1 - h\delta_{D_*}\phi = 1 - \phi h\delta_{D_*} = 1 - h\delta_{D_*}$; $\psi = 1 - \delta_{D_*}h\phi = 1 - \psi\delta_{D_*}h = 1 - \delta_{D_*}\phi h$.

Part 2. Then, and *only by using the previous equations*, the BPL conclusion is proved.

3 Algebraic structures in Isabelle

Isabelle [6] is a theorem prover developed at the University of Cambridge which provides a tool to interactive proof, specification and verification in higher-order logic. Our main initial interest is to formalize in Isabelle mathematical structures such as chain complexes and morphisms. Since the algebraic structures that appear in the proof of the BPL are quite involved, we first focus on an elementary example: semigroups.

The formalization is based on the work by Naraschewski and Wenzel [5], where signatures are record types. The implementation of structures in Kenzo was made also trough records with functional fields. In the particular case of semigroups, we start with the following type definition.

```
record 'a semigroup = "'a carrier"
                      + prod :: "'a => 'a => 'a"
```

This gives only the *signature* of the semigroup. In order to include the *axioms* for semigroups, we specify a predicate which is true on the records of type semigroup on which the axioms of a real semigroup hold. It is done in the following Isabelle definition.

```
constdefs
 semigroup :: "\<lparr>carrier :: 'a set,
               prod :: 'a \<Rightarrow> 'a \<Rightarrow> 'a,
               \<dots> :: 'b\<rparr> \<Rightarrow> bool"
 "semigroup S \<equiv>
        \<forall>x \<in> carrier S.
        \<forall>y \<in> carrier S.
        \<forall>z \<in> carrier S.
            prod S (prod S x y) z = prod S x (prod S y z)"
```

This specification appropiately restricts the axiom to the carrier set of the concrete structure. This allows the convenient construction of arbitrary carriers: they are not restricted to types on higher order logic. Note that this construction uses the facility of dependent sets, which was provided by Florian Kammueller [3]. From this basis, it is possible to operate with semigroups in Isabelle, and for instance to prove that the cartesian product of two semigroups (with the canonical binary operation) is also a semigroup. This kind of results are necessary, with more complex structures, to mechanize a proof of the BPL.

4 Mathematical structures for the BPL

An additional benefit of the use of Naraschewski and Wenzel's perspective to formalize algebraic structures is that they are easily *extensible*. This allows algebraic specification with inheritance. For instance, both the declaration and the specification of the structure Group can be constructed from that of Semigroup, as it is shown in the following Isabelle fragment.

```
record 'a group = "'a semigroup" +
 inv :: "'a \<Rightarrow> 'a"
 one :: 'a
constdefs
 group :: "\<lparr>carrier :: 'a set,
          prod :: 'a \<Rightarrow> 'a \<Rightarrow> 'a,
          inv :: 'a \<Rightarrow> 'a, one :: 'a,
          \<dots> :: 'b\<rparr> \<Rightarrow> bool"
 "group G \<equiv> semigroup G \<and>
    (\<forall>x. prod G (inv G x) x = one G) \<and>
    (\<forall>x. prod G (one G) x = x)"
```

Then, it is easy to understand how the algebraic structures for the BPL can be constructed step-by-step. From Group to Abelian Group and from this to Differential Abelian Group (an abelian group G endowed with a group homomorphism $d : G \to G$ such as $d \circ d = 0$). This is "almost" a chain complex (only the degree information is missing there). We conjecture that the main parts of the BPL, more concretely, of the second part of Sergeraert's proof previously evoked, could be established in this more general setting.

While developing step-by-step the data structures, the *constructions* on these structures should be also produced in a modular (and "extensible") way. For instance, the cartesian product construction on semigroups should be extended to the cartesian product (direct sum) of chain complexes. At that stage, the first important lemmas to prove the BPL would be stated. Our first objective would be to prove the following result, an auxiliary lemma to our foreseen proof of the BPL.

Theorem 4.1. *Let A_* and B_* be two chain complexes such that A_* is endowed with a homotopy operator h (that is, $h : A_* \to A_*$, degree 1, such that: $d_{A_*}h + hd_{A_*} = id_{A_*}$). Then the triple $(p_2, i_2, (h, 0)) : A_* \oplus B_* \Rightarrow B_*$ is a reduction, where p_2 is the canonical projection and i_2 the canonical inclusion.*

5 Conclusions

The preliminary studies realized up to the date show that Isabelle can be a suitable tool to mechanize proofs in a complex application field as constructive homological algebra. The benefits for increasing the reliability of software systems as Kenzo [2] are quite obvious, if certified versions of the algorithms are produced. On the other side, to confront theorem provers with such complicated data structures and processes seems a challenging and fruitful task.

References

[1] R. Brown. *The twisted Eilenberg-Zilber theorem.* Celebrazioni Arch. Secolo XX, Simp. Top., 1967, pp 34-37.

[2] X. Dousson, F. Sergeraert and Y. Siret. *The Kenzo program.* `http://www-fourier.ujf-grenoble.fr/~sergerar/Kenzo/`

[3] F. Kammueller. *Modular Reasoning in Isabelle.* PhD thesis, Technical Report No. 470, Computer Laboratory, University of Cambridge.

[4] L. Lambán, V. Pascual and J. Rubio. *Specifying implementations.* In Proceedings ISSAC'99, ACM Press, 1999, pp. 245-251.

[5] W. Naraschewski and M. Wenzel *Object-oriented verification based on record subtyping in higher-order logic.* In J. Grundy and M. Newey, editors, Theorem Proving in Higher Order Logics: TPHOLs '98, LNCS, 1479 (1998).

[6] T. Nipkow, L. C. Paulson and M. Wenzel. *Isabelle,HOL: A proof assistant for higher order logic.* Lecture Notes in Computer Science, 2283.

[7] L. C. Paulson. *The foundation of a generic theorem prover.* Journal of Automated Reasoning 5(3), 1989, pp. 363-397.

[8] J. Rubio and F. Sergeraert. *Constructive Algebraic Topology.* Lecture Notes Summer School in Fundamental Algebraic Topology, Institut Fourier, 1997. `http://www-fourier.ujf-grenoble.fr/~sergerar/Summer-School/`

[9] J. Rubio and F. Sergeraert. *Constructive Algebraic Topology.* Bulletin des Sciences Mathématiques 126, 2002, pp. 389-412.

Introduction of redundant constraints for solving systems of distance constraints

Heikel Batnini

I3S INRIA CERMICS
2004 route des Lucioles, BP 93, 06902 Sophia Antipolis, France
Email :Heikel.Batnini@inria.fr
Tel : 04-92-38-76-21

Motivations

Distance constraints occur in numerous applications, for instance in molecular conformation problem or in the optimal design of mecanic systems. These constraints are often only a subpart of a more complex constraint system. The global form of the quadratic equation system to solve is :

$$\delta_{ij}^2 = \sum_{k=1}^{k=p} (x_i{}^k - x_j{}^k)^2$$

where p is the dimension of the euclidian space, $x_i{}^k$ is the k^e coordinate of i_e point P_i and δ_{ij} is the euclidian distance between P_i and P_j.
More precisely, consider n points of an euclidian space, for which an approximation of the euclidian distances between some pairs of points is known. The coordinates of these points and the distances are bounded by finite values, that is to say an interval $[\underline{x},\overline{x}]$ [1] is associated to each coordinate and each distance. The aim is to prune the domains of the coordinates to remove inconsistant values that do not verify distance constraints. Classical filtering algorithm (eg. 2B-consistency, Box-consistency [12, 11, 6]) usually achieve a very poor pruning on such constraints. We propose here two specific algorithms to handle distance constraints. The essential point is the introduction of redundant constraints and a more global filtering.

Before going into the details, let us explain why classical filtering techniques are unable to do this job. Local filtering methods are based on a

1. $\underline{x}$ (resp. $\overline{x}$) stands for the lower(resp. upper) bound of the domain of variation of x.

relaxation of arc-consistency[1]. The main drawback of these methods comes from their local processing. Then, the notion of point itself is lost during the pruning process.

A simple example

Let us consider a quad $ABCD$ where E is the middle of segment CD:
$A(0,0)$, $B(8,0)$, $C(x_C,y_C)$, $D(x_D,y_D)$, $E(x_E,y_E)$
$BC = CE = DE = 2$, $AD = 3$, $CD = 4$, $AB = 8$

$$D_{x_C} = [-20,20],\ D_{y_C} = [-20,20]$$
$$D_{x_D} = [-20,20],\ D_{y_D} = [-20,20]$$
$$D_{x_E} = [-20,20],\ D_{y_E} = [-20,20]$$

Then we have the following underconstrained system of distance constraints:

$$C_1: \quad (x_C - 8)^2 + {y_C}^2 = 4$$
$$C_2: \quad (x_C - x_E)^2 + (y_C - y_E)^2 = 4$$
$$C_3: \quad (x_D - x_C)^2 + (y_D - y_C)^2 = 16$$
$$C_4: \quad (x_D - x_E)^2 + (y_D - y_E)^2 = 4$$
$$C_5: \quad {x_D}^2 + {y_D}^2 = 9$$

where the unknowns are the coordinates of the points C, D and E. Figure 1

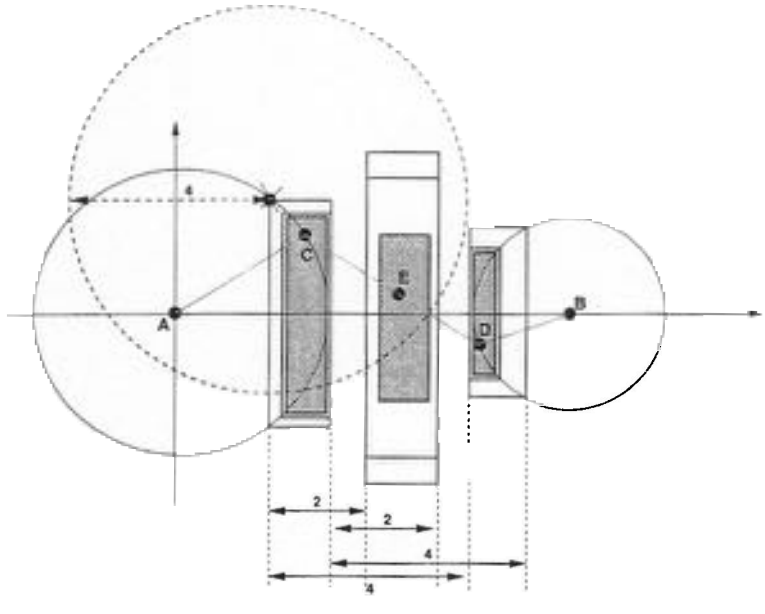


FIG. 1 – *Comparison of the results for 2B-consistency, 3B-consistency and 3B-consistency over triangles with redundant constraints*

shows that neither 2B-consistency (external box) nor 3B-consistency (middle box) are able to achieve a relevant pruning of the domain of E. The instan-

ciation of each coordinate by a value of its domain satisfies the constraints but no acceptable instanciation of both coordinates of E exists.

A refinement of the filtering

In distance constraint problems two types of entries can be distinguished: the domains of the coordinates of the points, for which we want an optimal filtering and distances relations. These distances relations may not always be satisfied since all triplet of distances must at least verify triangular inequalities. To ensure this property we can add the following redundant constraints:

$$\forall i,j,k \in [1..n] : i \neq j \neq k \begin{cases} \delta_{ij} \leq \delta_{ik} + \delta_{jk} \\ \delta_{ij} \geq |\delta_{ik} - \delta_{jk}| \end{cases}$$

Table 2 shows the results[2] of the filtering using these redundant constraints for the precedent example.

	$2B$ on I	$2B$ on I_f	CLOSE-FILTER
D_{x_C}	[2,3]	[2,3]	[2,3]
D_{y_C}	[-2.23,2.23]	[-2.23,2.23]	[-2.23,2.23]
D_{x_D}	[6,7]	[6,7]	[6,7]
D_{y_D}	[-1.73,1.73]	[-1.73,1.73]	[-1.73,1.73]
D_{x_E}	[4,5]	[4,5]	[4,5]
D_{y_E}	[-3.46,3.46]	[-2.64,2.64]	[-2.64,2.64]
CPU time	0.12s	0.12s	0.03s

FIG. 2 – *Comparison of the results for different experimentations*

A polynomial time algorithm for computing a closure

The *bound-smoothing* algorithm[5], based on Floyd's shortest path algorithm[14], provides a second way to improve the filtering of distance constraints. This algorithm computes distance graph closure(see fig. 3). So, we introduce a filtering procedure to prune coordinates domains each time a distance domain is updated by the closure algorithm. The filtering procedure uses the projection function defined by local filtering methods. The point is that the worst case complexity of this algorithm is independant from the size of the domains. This algorithm achieves a significant filtering but does not ensure 2B-consistency over the whole system.

Introduction of the barycenter for triangle filtering

The introduction of redundant constraints using triangle isobarycenter improves also the filtering of the domains of the coordinates. The coordinates

2. We used a computer with a bi-processor pentium 1Ghz and a memory of 256Mo

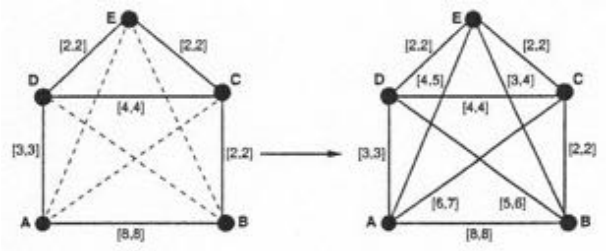


FIG. 3 – *Distance graph of the instance $\mathcal{I}$ is closed by the algorithm to generate a new closed instance $\mathcal{I}_f$.*

of the barycenter are defined by a simple linear function of vertices coordinates. The property of the closure allows us to introduce an approximation of the distance between the barycenter and each vertex of any triangle. So the following set of constraints is generated for each triangular system $P_iP_jP_k$:

$$\mathcal{C}_{ijk} = \begin{cases} (x_G - x_i)^2 + (y_G - y_i)^2 &= \frac{1}{9}(2\delta_{ik}^2 + 2\delta_{ij}^2 - \delta_{jk}^2) \\ (x_G - x_j)^2 + (y_G - y_j)^2 &= \frac{1}{9}(2\delta_{ij}^2 + 2\delta_{jk}^2 - \delta_{ik}^2) \\ (x_G - x_k)^2 + (y_G - y_k)^2 &= \frac{1}{9}(2\delta_{ik}^2 + 2\delta_{jk}^2 - \delta_{ij}^2) \\ x_G &= \frac{1}{3}(x_i + x_j + x_k) \\ y_G &= \frac{1}{3}(y_i + y_j + y_k) \end{cases}$$

Therefore, a 3B filtering can be performed on each triangle for which the barycenter constraints have been generated. Note that these constraints add implicit angular information that may improve the filtering quality in many cases. However slow convergence cycle may occur in a triangle whereas it not appears on the initial system. The figure 4 shows that this method provides a better filtering than usual methods and how it might be costly.

	3B on $\mathcal{I}$	3B on $\mathcal{I}_f$	3B on triangles
D_{x_C}	[2.31,3]	[2.31,3]	[2.31,3]
D_{y_C}	[−1.91,1.91]	[−1.91,1.91]	[−1.91,1.91]
D_{x_D}	[6,6.81]	[6,6.81]	[6,6.81]
D_{y_D}	[−1.61,1.61]	[−1.61,1.61]	[−1.61,1.61]
D_{x_E}	[4,5]	[4,5]	[4.15,4.9]
D_{y_E}	[−2.14,2.14]	[−1.82,1.82]	[−1.76,1.76]
CPU time	1.05s	0.74s	15.08s

FIG. 4 – *A global consistency for the exemple*

Further work concerns the integration of these algorithms in the more general framework introduced in [15].

Références

[1] Macworth A.K. Consistency in networks of relations. *Atificial Intelligence*, (8):99–118, 1977.

[2] Bessière C. and Régin J-C. Enforcing arc consistency on global constraints by solving subproblems on the fly. In *PPCP*, pages 103–117, 1999.

[3] Bliek C. and Sam-Haroud D. Path consistency on triangulated constraint graph. In *IJCAI-99*, pages 420–425, 1999.

[4] Sam-Haroud D. *Constraint.* PhD thesis, École Polytechnique fédérale de Lausanne, 1995.

[5] Crippen G.M and Havel T.F. *Distance geometry and molecular conformation.* John Wiley and sons, 1988.

[6] Collavizza H., Delobel F., and Rueher M. A note on partial consistencies over continuous domains. *Lecture Notes in Computer Science*, 1520:147–??, 1998.

[7] Régin J-C. Generalized arc consistency for global cardinality. In *AAAI-96*, pages 209–215, 1996. Oregon.

[8] Régin J-C. The symetric alldiff constraint. In *IJCAI-99*, pages 420–425, 1999.

[9] Régin J-C. and Rueher M. A global constraint combining sum and difference constraints. In *CP'2000 LNCS*, volume 1894, pages 384–396, 2000. Singapore.

[10] U. Montanari. Networks of constraints: Fundamental properties and applications to image processing. *Information science*, 7:95–132, 1974.

[11] Lhomme O. Consistency techniques for numerical csps. In *IJCAI-93*, pages 232–238, 1993.

[12] Lhomme O. *Contribution à la résolution de contraintes sur les réels par propagation d'intervalles.* PhD thesis, Université de Nice-Sophia Antipolis, 1994.

[13] Moore R. *Interval analysis.* Prentice-Hall, 1977.

[14] Cormen T.H., Leiserson C.E, and Rivest R.L. *Introduction to algorithms.* The MIT Press, 1991.

[15] Lebbah Y., Rueher M., and Michel C. A global filtering algorithm for handling systems of quadratic equations and inequations. In *CP'2002, Eighth International Conference on Principles and Practice of Constraint Programming*, Cornell University, Ithaca, NY, USA, Sept, 7-13 2002.

Proof carrying certification for lightweight devices

Thibault Candebat and David Gray
School of Computer applications,DCU,
Dublin 9, Ireland
{candebat,dgray}@computing.dcu.ie

1 Introduction

Current Public Key Infrastructures (PKI) are resource consuming and not suitable for thin clients such as mobile devices. Therefore, our aim is to design and prototype a PKI based on the concept of *proof-carrying certification* that can be used efficiently from a constrained mobile device. From a more theoretical point of view, we are also interested in proving the consistency of our system and its validity in terms of security : we will use a formalism like the π-calculus[1] to formally prove our assumptions.

2 Background

2.1 Lightweight Devices

The lightweight devices have limited processing power, memory, local storage and speed of network link and are typically mobile devices such as PDAs[2] and Java Phones[3]. These mobile devices will be used as part of our test bed.

2.2 Public Key Infrastructure

Asymmetric cryptography[4] uses a matching key pair to secure electronic commerce transactions. The private key is meant to be kept secret by its owner, while the public key needs to be made available so that any user who wants to use it can find it. The main problem with this scheme is public key distribution : how can Alice be sure that she's using Bob's public key in order to secure the transaction she's conducting with him ? In other words, how to bind a public key to a user ? This can be done by using a digital certificate[5], a document that is the electronic equivalent of physical proofs of identity and that is issued by a trusted authority (certificate authority) and stored in a directory. The infrastructure composed by all these entities is called a Public Key Infrastructure.

The majority of current Public Key Infrastructures (PKIs) are based on X.509[6] and while this technology is suitable for many existing applications, the limited processing power, storage and network connectivity of mobile devices makes full-blown X.509 unsuitable for most mobile applications. Considering the hierarchical model of X509, a mobile application must first retrieve a whole chain of digital certificates (called certification path) and check it to be able to use the certificate she requested. Then to be sure that none of them has been revoked [7], one may have to check the revocation status of the certificates.

The most widely used mechanism to check if a certificate has been revoked is to go through a CRL (Certificate Revocation List), issued by a certificate authority. However, this data can get very large (several megabytes) and therefore, downloading, validating and using a CRL on a small mobile device with constrained bandwidth can be problematic.

In a nutshell, the amount of network traffic and latency, and the number of cryptographic operations required to check the validity of a certificate within an X509 infrastructure makes X509 unsuitable for small devices.

2.3 Proof Carrying Authentication

This approach developed by Appel and Felten [8] is based on the work carried out by Georges Necula on Proof Carrying Code[9]. The idea is to use a distributed framework based on higher order logic to provide authentication techniques. Alice wants to have access to some resource on a server owned by Bob. To do so, she will construct a request using a language based on higher order logics and submit it to Bob who will decide whether or not he will grant her access to the resource. Considering that logics with quantification over predicates tend to be undecidable, a formal proof needs to be attached to the statement sent to Bob, so that he can verify that Alice has the rights to access the resource by using a proof checker.

The advantage of this approach is that all the burden of proving the access control is left to the client, i.e the client must construct a proof and the server needs only to check this proof. An application of that theory was developed in [10] to provide access control to some web pages.

The same kind of approach can be applied to our problem. Our strategy aims at delegating as much processing as possible on a server. Therefore, the server would produce a proof that the thin client would have to formally check.

3 Our Approach

3.1 Description

The main objective of the Public Key Infrastructure we are designing is to leave the mobile device with a minimum number of actions to perform. The two main tasks the lightweight device will have to carry out are :

- Performing Cryptographic operations (digital signature/validation, encryption/decryption)
- Retrieving Certificates and validating them

Some cryptographic operations can be very resource consuming. For example, operations such as prime number generation used for key generation in RSA[11] or random number generation can slow down a program. Therefore, our aim is to speed them up as much as possible, depending on the capacities of the small devices, and to reduce the number of operations that a device must perform.

In order to reduce the time required to perform the certification path discovery and validation on the mobile device, we will develop a new technique based on Proof Carrying Authentication [8]. This Proof carrying Certification will enable thin devices to take short cuts through the certificate validation process. Lightweight devices will act as clients of a server implementing the delegated path validation and discovery protocols (IETF) [12]. This server will collect information related to certificates, produce a formal proof and leave mobile devices with the only constraint to perform a simple formal proof checking to validate a single certificate or a certification chain.

Rather than having to verify multiple digital signatures, thin clients will take advantage of the concept of Proof carrying Certification to reduce the number of cryptographic operations needed during the proof checking.

A different formal proof will be designed for every PKI, guarantying interoperability between the various infrastructures.

3.1.1 Cryptographic operations

Two determining factors will influence the speed of these operations : the choice between different algorithms to sign and verify signatures and its implementation.

Given the limited processing power of most mobile devices, the efficiency of public-key encryption is crucial in determining the response time for users. Efficient ciphers

and implementations are required to avoid unacceptable delays. Elliptic Curve Cryptography (ECC)[13] is generally accepted as being more efficient than RSA; in particular, ECC requires smaller keys and key pair generation is simpler.

Java has been chosen to implement the cryptographic component of the PKI, so that the infrastructure will be platform independent. However, since mobile devices have different processing capabilities, several configurations will be implemented :

- One full Java version that can run on every device.
- One Java/C++ version using some native calls to speed up the key generation process, the signature/verification operations, a version that can run on more advanced devices like PDAs or 3G phones.

3.1.2 Certificate retrieval and validation

Our approach aims at delegating all the burden of retrieving the certificate chain to a server. To do so, the small device will communicate with a non trusted server [Figure 1] implementing the Delegation Path Discovery protocol (DPD) partially defined in [12].

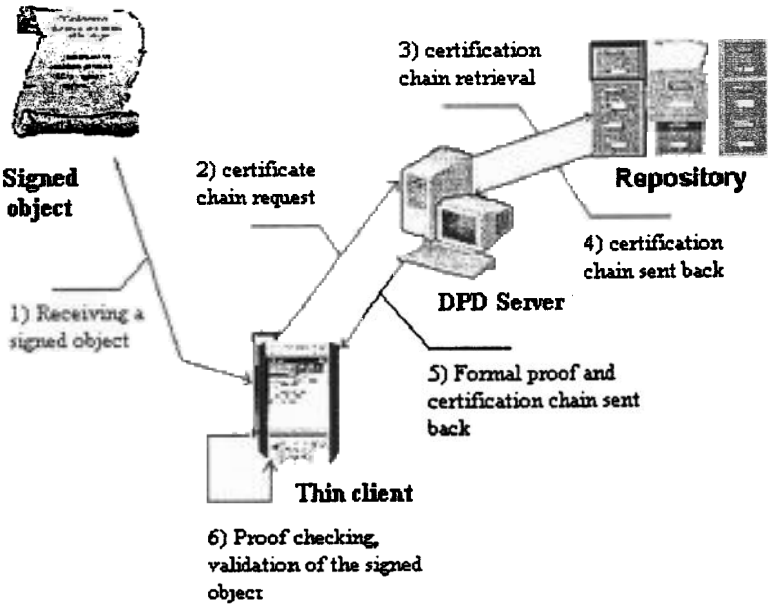


Figure 1 : Validation process

Once the certification chain is retrieved, the server will be able to construct the formal proof. In order to enable future developments on our infrastructure and remain interoperable with other infrastructures like SPKI/SDSI or PGP, a meta proof implemented as an XSLT file, different for each PKI, will be used to generate the actual proof [Figure 2]. The logic used to describe the formal proof is first order predicate and may include some temporal logic in order to deal with issues related to time.

Then, both the proof and the certification chain are sent back to be checked on the thin client side.

```
X509_Proof
R(1)::[trustedCA(C1)^keybind(C1,K1)^signed(K1,c2)=>isTrusted(c2)]
R(2)::[R(1)^contain(c2,C2,K2)=>keybind(C2,K2)]
R(3)::[R(2)^signed(K2,c3)=>isTrusted(c3)]
R(4)::[R(3)^contain(c3,C3,K3)=>keybind(C3,K3)]
R(5)::[R(4)^cert(C3,KA,A)^signed(KA,F)=>says(A,F)]
```

Figure 2: A formal proof example

The proof checker implemented in Java on the small device generates small proofs that are cached and that can be re-used for a certain amount of time, depending on how secure one wants the certificate checking to be. This idea is one of the shortcuts we are planning to apply in order to speed up computations on the thin client.

4 Conclusion

In this abstract, we describe a public key infrastructure suitable for constrained mobile devices, that is interoperable with existing public key infrastructures. We have started to build an implementation of a prototype using a Compaq iPAQ 3600 :

- The cryptographic side of the infrastructure is completely achieved : Signature and verification of digitally signed documents can be done in less than a second using the enhanced version of the cryptographic component.
- The proof checker used to validate the formal proof sent by the DPD server has been partially designed and will be implemented in Java.

However, some issues remain unsolved as the secure random number seed, used for key generation on the lightweight device and the secure private key storage. A revocation checking method has to be integrated as well in the formal proof so that the thin client can formally check that some certificates haven't been revoked.

Speeding up cryptographic operations on the thin client is efficient but reducing their number is even better. On the basis of that principle, we still need to develop our idea of shortcuts enabling small devices to skip some steps in the certification path validation.

In the long term, once our infrastructure is completely designed and implemented, we plan to formally prove that it is secure by developing a theory based on the π-calculus.

Bibliography

- Robin Milner
  Communicating and mobile systems : the π-calculus
  Cambridge University Press
- Craig C. Freudenrich, Ph.D.
  How Personal Digital Assistants (PDAs) Work
  http://www.howstuffworks.com/pda.htm
- Dylan Tweney
  Java on Your Mobile Phone?
  http://www.business2.com/articles/web/0,1653,38833,FF.html
- Whitfield Diffie, Martin E. Hellman
  New Directions in Cryptography
  IEEE Transactions on Information Theory
- Kohnfelder, LM
  Towards a practical public key cryptosystem
  Master's thesis, MIT Laboratory for computer science, 1978
- ITUT Recommendation
  X.509:The Directory Authentication Framework
  Technical report X.509, ITU, 1997
- Moni Naor, Kobbi Nissim
  Certificate Revocation and Certificate Update
  Proceedings 7th USENIX Security Symposium (San Antonio, Texas), 1998
- Andrew W. Appel and Edward W. Felten
  Proof Carrying Authentication,1999
- George C. Necula
  Proof-Carrying Code
  Presented at POPL97, January 1997.
- Lujo Bauer, Micchael A,Schneider, Edward W.Felten
  A proof-carrying authorization system
  Technical Report TR-638-01, 2001
- R.L. Rivest, A. Shamir, L. Adleman
  A Method for Obtaining Digital Signatures and Public-Key Cryptosystems (1978)
- Denis Pinkas, Bull, Russ Housley, RSA Laboratories
  Delegated Path Validation and Delegated Path Discovery Protocol Requirements (DPV&DPD-REQ)
  Internet Draft
- Don Johnson, Alfred Menezes
  The Elliptic Curve Digital Signature Algorithm (ECDSA)ăă, 1999

Proof Planning with Semantic Guidance

Seungyeob Choi

School of Computer Science
The University of Birmingham
Birmingham B15 2TT, England
email: S.Choi@cs.bham.ac.uk

1 Introduction

Proof planning [2, 10] is a reasoning process in which mathematical theorem proving is viewed as a planning problem. It incorporates domain-dependent and/or general mathematical knowledge, and shows a great potential in the area where the traditional logic-based theorem proving systems have been typically weak. The plan operator – *method* – encodes a piece of mathematical knowledge. When there are more than one method applicable, the proof planner has to determine which one to choose. Some methods may lead to a dead end where no more method is applicable. In this case the proof planner needs to backtrack to a previous stage, and apply another method. In order to minimise the backtrackings, good heuristics are needed. Up to now such heuristics are only useful in some particular cases. For instance, heuristic information can be encoded in form of specificity which prefers a specific method over a general one, or in form of an explicit ranking which rates methods independently of the concrete situation. None of these approaches provides a satisfying answer to what method to choose.

In this abstract, the idea of semantically guided proof planning is presented, and an implementation and experiments are discussed. For the research, the ΩMEGA theorem proving environment [1] is used. The semantically guided proof planner has been realised in MULTI – the multi-strategy proof planner of the ΩMEGA system. The model-guidance module plays the most important role in the semantic guidance. It updates a database of reference models and answers queries from MULTI whether a concrete method application is suitable with regard to the current reference models and how promising a method application is. In order to obtain the reference models, the model-guidance module provides an interface to FINDER [11], which is a first order model generator. Heuristic search control information is generated from the models.

2 Proof Planning with Semantic Guidance

2.1 Semantic Guidance in Automated Reasoning

Since human mathematicians make strong use of semantic information in order to obtain heuristics to solve problems, it is believed that a semantic approach is important for modelling the human reasoning capabilities in mathematics. There have been many attempts to use semantic information to guide proof search (see for instance [12, 8]). The basic idea is to use a *reference model* or a set of reference models to heuristically guide the choice of the clauses in the inference step. However, the generation of models in each inference step sometimes causes a significant overhead. Recently, a new method was introduced [6] in which the theorem prover generates models only at the beginning, and uses them in later stages of the proof search. Furthermore, in some problems where typically theorem provers fall into an exponential explosion, the assumption part of the problem can be refined to a simplified form before a refutation search is started [3].

Like the semantic guidance used in the traditional resolution-based theorem provers, semantic information can also be used in proof planning to guide the choice of the next method to apply. The semantics of the assumptions and the conclusion can be used to evaluate whether new premises can contribute to the proof (in forward planning), or whether a new open goal can be proved (in backward planning). The semantically guided proof planning approach (proposed by Choi and Kerber [5]) uses reference models as search control heuristics.

2.2 Semantics in Proof Planning

In proof planning, mainly two different types of planning strategies are used, forward planning and backward planning. In forward planning, the planner searches for a method that takes assumptions (like axioms, definitions, or already proved intermediate results) and produces new facts that can be used as new assumptions in the following planning steps. Initially there is a gap between the premises (the initial assumptions) and the theorem, the planner continues filling in the gaps between assumptions and the theorem until there are no more open goals. Likewise, in backward planning, a method takes a theorem and produces subgoals sufficient to deduce the theorem. Some of the subgoals may be identical to existing assumptions, but typically some others are new and to be added as open goals to be proved in later steps.

As in any search problem, the efficiency of proof planning depends on exploring the more promising parts of the search space first. In order to do so, we need good heuristics. Some heuristic information can be encoded in form of specificity, that is, prefer the application of a specific method over the application of a more general one. Another approach is to explicitly ranking methods, e.g., by a rating, and to apply the applicable method that has the highest ranking, independently of the concrete situation.

The main idea of the work proposed here is that the plausibility of a method can be estimated by means of comparing the semantics of new open goals generated by the method

with those of the assumptions and of the theorem. We extend the idea to intermediate results that any model of the assumptions Γ has also to be a model of the theorem ϕ. More concretely, assume we have a reference set of interpretations $\mathcal{M}$. The model sets $\mathcal{M}_\Gamma := \{m \in \mathcal{M} | m \models \Gamma\}$ and $\mathcal{M}_\phi := \{m \in \mathcal{M} | m \models \phi\}$ are defined; then it follows from $\Gamma \models \phi$ that $\mathcal{M}_\Gamma \subseteq \mathcal{M}_\phi$. If an intermediate result ψ is semantically not a superset of $\mathcal{M}_\Gamma$, that is, $\mathcal{M}_\Gamma \not\subseteq \mathcal{M}_\psi$, in backward planning, or not a subset of $\mathcal{M}_\phi$, that is, $\mathcal{M}_\psi \not\subseteq \mathcal{M}_\phi$, in forward planning, we can assume that ψ is not plausible. We will not be able to prove the theorem with ψ unless the models of the assumptions, i.e., of $\mathcal{M}_\Gamma \cap \mathcal{M}_\psi$, are a subset of the models of the theorem, i.e., of $\mathcal{M}_\phi$, and neither will we be able to prove ψ with the given assumptions unless its models form a superset of $\mathcal{M}_\Gamma$.

2.3 Semantic Restriction and Selection

A *semantic restriction* strategy is a strategy that refuses any methods whose applications produce an intermediate result with which the above semantic relation is not satisfied. This simple strategy can avoid many useless steps. For instance, the search for a proof plan for $S \rightarrow \forall x Q(x) \vdash \forall x (S \rightarrow Q(x))$ can be reduced from 10 steps to only 3 steps [7, 5] with the semantic restriction strategy in the ΩMEGA proof planner. Likewise, the search for a proof plan for a Topology theory example $\mathsf{int}(A \cap B) = \mathsf{int}(A) \cap \mathsf{int}(B)$ which would not be proved without semantic guidance, can also be proved [4].

A *semantic selection* strategy is a strategy which not only restricts the number of methods in the semantic way, but heuristically chooses one that seems to make most progress towards filling the gap. It evaluates the semantics of each method where all methods would seem equally applicable if semantics were not taken into consideration. More promising methods can be distinguished from the other candidates by estimating how well they semantically match the open goals with respect to the subset and superset relation with the given premises. In a forward reasoning approach, the methods that restrict the model class best in the direction to the theorem are better. Concretely, if the problem is $\Gamma \vdash \phi$ and two methods M_1 and M_2 transform the problem to $\Gamma_1 \vdash \phi$ and $\Gamma_2 \vdash \phi$, respectively, where $\mathcal{M}_\Gamma \subseteq \mathcal{M}_{\Gamma_i}$ and $(\mathcal{M}_\Gamma \cap \mathcal{M}_{\Gamma_i}) \subseteq \mathcal{M}_\phi$ with $i = 1$ or 2, select M_1 rather than M_2 if and only if $\mathcal{M}_{\Gamma_1}$ is bigger than $\mathcal{M}_{\Gamma_2}$. (If the sets have the same cardinality, make a random choice.) Likewise in backwards reasoning, a method should be selected which produces the smallest model class $\mathcal{M}_{\Gamma_i}$.

3 Implementation and Experiments

The semantic proof planner has been realised on the basis of the ΩMEGA system. Previously, ΩMEGA did not evaluate methods with respect to the concrete proof state when two or more methods are applicable. Instead, it depended on a fixed rating factor assigned to each method by the developer. The issues about implementation and experiments have been reported in [7]. In order to get a form of semantics, we generate models from the premises and the open goals with a model generator. However, since it is not practical

to keep a huge amount of models, we may use a smaller set of carefully defined typical models which is sufficient for determining the semantic relationship. The model set represents semantics and, in addition, the relation between two model sets correspond to the semantic relation between two premises. For instance, if the model set of the premises is not a subset of the model set of an open goal, we can assume that the open goal cannot be deduced from these premises. Thus, we should prevent the method from being applied and, as a consequence, avoid unnecessary steps that would have led to backtracking.

4 Further Work

In the semantically guided proof planner based on MULTI, a small number of simple problems in first order logic and Topology theory have been tested. The experimental results have been successful. Thus, our first task is to evaluate the implemented approach with respect to a broader case study. When applying our approach to problems from complicated mathematical domains, there are still many open questions. One important question is how to incorporate domain knowledge. Currently we pass only the proof assumptions to FINDER. If domain knowledge such as theorems and axioms is available, it could also be provided to FINDER to generate better models. However, if large sets of axioms and theorems are available, the question is how to restrict the set of candidate facts passed to FINDER. Another important question is how to select informative models. Currently we use FINDER to generate reference models. While this is already informative (and reduces search) in some examples we looked at, we want to improve it further in several ways. In particular, we want to keep the reference set of models small and informative at the same time. The idea of *typical models* as introduced by Kerber et al. [9] is considered promising for this purpose.

References

[1] C. Benzmüller et al. ΩMEGA: Towards a mathematical assistant. In W. McCune, editor, *Proceesings of CADE-14*, LNAI 1249, pp 252–255. Springer-Verlag, 1997.

[2] A. Bundy. The use of explicit plans to guide inductive proofs. In E. Lusk and R. Overbeek, editors, *Proceesings of CADE-9*, LNCS 310, pp 111–120. Springer-Verlag, 1988.

[3] S. Choi. Towards semantic goal-directed forward reasoning in resolution. In D. Scott, editor, *Proceedings of AIMSA'2002, Varna, Bulgaria*, LNAI 2443, pp. 243–252, Springer-Verlag, 2002.

[4] S. Choi. Semantic restriction of methods in proof planning. Talk presented at *CIAO-2002 workshop, The University of Edinburgh, Scotland.* April 2002.

[5] S. Choi and M. Kerber. Model-guided proof planning. In L. Magnani et al., editors, *Logical and Computational Aspects of Model-Based Reasoning.* Kluwer Academic Publishers, Dordrecht, The Netherlands, 2002.

[6] S. Choi and M. Kerber. Semantic selection for resolution in clause graphs. In *Proceedings of AI'02, Canberra, Australia*, LNAI, Springer-Verlag, 2002 (forthcoming).

[7] S. Choi and A. Meier. Proof planning in ΩMEGA with semantic guidance. Cognitive Science Research Papers CSRP-01-11, University of Birmingham, School of Computer Science, Birmingham, England, 2001.

[8] H. Chu and D.A. Plaisted. Semantically guided first-order theorem proving using hyper-linking. In A. Bundy, editor, *Proceedings of CADE–12, Nancy, France*, LNAI 814, pp. 192–206, Springer-Verlag, 1994.

[9] M. Kerber, E. Melis, and J. Siekmann. Analogical reasoning with typical examples. SEKI Report SR-92-13, Fachbereich Informatik, Universität des Saarlandes, Saarbrücken, Germany, 1992.

[10] E. Melis and J. Siekmann. Knowledge-based proof planning. *Journal of Artificial Intelligence*, 115(1):65–105, 1999.

[11] J. Slaney. FINDER: Finite Domain Enumerator. In A. Bundy, editor, *Proceedings of CADE–12, Nancy, France*, LNAI 814, pp. 798–801, Springer-Verlag, 1994.

[12] J. Slaney, E. Lusk, and W. McCune. SCOTT: Semantically Constrained Otter. In A. Bundy, editor, *Proceedings of CADE–12, Nancy, France*, LNAI 814, pp 764–768, Springer-Verlag, 1994.

Towards Automation of Real Analysis in Coq

Luís Cruz-Filipe*
Center for Logic and Computation, IST, Lisbon, Portugal
Department of Computer Science,
University of Nijmegen, The Netherlands
P. O. Box 9010, 6500 GL NIJMEGEN, The Netherlands
+31-24-3652610
lcf@cs.kun.nl

We have finished a constructive formalization in the theorem prover Coq of the Fundamental Theorem of Calculus, which states that differentiation and integration are inverse processes. This formalization is built upon the library of constructive algebra created in the FTA (Fundamental Theorem of Algebra) project, which is extended with results about the real numbers, namely about (power) series.

This formalization was done closely following the work of Bishop [1]; the real numbers were first axiomatically characterized as a complete ordered field with the archimedian property; later, this axiomatization was proved by Geuvers and Niqui [4] to be appropriate (in the sense that the construction of real numbers as Cauchy sequences of rationals satisfies the axioms) and categorical (as any two models of these axioms are isomorphic).

Using this work as a basis, partial functions are defined as a Coq record type consisting of a predicate and a total function on the set of real numbers that satisfy that predicate (see [3]). The usual operations (composition, addition, multiplication, division) are then defined as yielding partial functions from partial functions. We can then define continuity, differentiability and integration, and prove the usual properties of these: preservation of continuity and differentiability through algebraic operations and functional

*This author was supported by the Portuguese Fundação para a Ciência e Tecnologia, under grant SFRH / BD / 4926 / 2001.

composition, uniqueness of derivative and the rules for computing derivatives. Using these, we can then formalize Bishop's proofs of the constructive versions of Rolle's theorem, the Mean Law and Taylor's theorem, as well as of the Fundamental Theorem of Calculus.

One of the most important issues throughout this work is automating routine tasks, such as computing the derivative of a function. This was successfully done in Coq by using the general method of reflection as described in [7] and [5] adapted to our domain to define new tactics suitable for use with specific kinds of goals. With these tactics further development of the theory became much more high-level, allowing proofs to be done at a level of detail more similar to what is usual in mathematics.

Finally, the usual elementary transcendental functions (exponential, sinus, cosinus, tangent and their inverses) were defined as examples of partial functions and their properties were proved using the theoretical tools previously formalized.

Future work will include developing a higher level of automation, including the use of reflection to build a new tactic that can automatically prove a large class of equalities of real numbers.

References

[1] Bishop, E., *Foundations of Constructive Analysis*, McGraw-Hill Book Company, 1967

[2] The Coq Development Team, *The Coq Proof Assistant Reference Manual Version 7.2*, INRIA-Rocquencourt, December 2001

[3] Cruz-Filipe, L., *Formalizing Real Calculus in Coq*, to appear in *Procs. 15th International Conference on Theorem Proving in Higher Order Logics (TPHOLs2002)*, Carreõ, V., Munõz, C. and Tahar, S. (eds.), NASA, 2002

[4] Geuvers, H. and Niqui, M., *Constructive Reals in Coq: Axioms and Categoricity*, in Callaghan, P., Luo, Z., McKinna, J. and Pollack, R. (Eds.), *Proceedings of TYPES 2000 Workshop*, Durham, UK, LNCS 2277

[5] Geuvers, H. and Oostijk, M., *Proof by Computation in the Coq system*, in *Theoretical Computer Science*, vol. 272, pp. 293-314, Elsevier, 2002

[6] Geuvers, H., Pollack, R., Wiedijk, F. and Zwanenburg, J., *The algebraic hierarchy of the FTA project*, in *Calculemus 2001 Proceedings*, Siena, Italy, 13-27, 2001

[7] Geuvers, H., Wiedijk, F. and Zwanenburg, J., *Equational Reasoning via Partial Reflection*, in *Theorem Proving in Higher Order Logics*, 13th International Conference, TPHOLs 2000, Springer LNCS 1869, 162-178, 2000

Model Checking of Linear Programs into an infinite domain

Pasquale De Lucia

DIST–Università degli Studi di Genova,
via Opera Pia 13, 16145, Genova, Italia,
pasko@dist.unige.it

1 Introduction

The success of model-checking in hardware and in protocol analysis has led naturally to several attempts to develop similar techniques for the analysis of software. As pointed out in [2] a fundamental problem in this endeavor is to identify a model for programs (analogous to the finite state machines used for modeling hardware circuits) for which reasonably simple abstractions from conventional programming languages (such as C and Java) as well as efficient model-checking procedures do exist.

The SLAM project at MSR follows this path of action by proposing *(i) boolean programs* as a model for sequential programs and *(ii)* a procedure that abstracts any program given as input, say P, into a boolean program B having the same control-flow graph as P and whose execution traces are a superset of those of P. Boolean programs have the usual control-flow constructs, procedural abstraction with call-by-value parameter passing and recursion; program variables (both locals and globals) are restricted to range over the domain of the boolean values T and F. The core idea of SLAM project is that the study of the properties of a program may be reduced to the study of line reachability: in fact, considering a program P and a property ϕ, it is possible to rewrite P as an equivalent program P_1 where a line ℓ is reached iff ϕ is violated. Giving to ℓ the meaning of the `error` line, its reachability implies the violation of ϕ. The reachability of ℓ in P_1 is indirectly determined by checking the reachability of ℓ in B by means of a model-checking procedure for boolean programs. If ℓ is unreachable in B then this is detected by the model-checker and the unreachability of ℓ in P_1 is reported to the user. Otherwise an execution trace leading to ℓ in B is found by the model-checker and its feasibility in P_1 is checked. If the feasibility check succeeds, then an execution trace leading to ℓ in P_1 is returned, otherwise B is refined in some way so to rule out the unfeasible path and the whole procedure is iterated.

The efficiency of the approach depends heavily on the number of unfeasible paths allowed by the abstract program. In the SLAM approach, the abstraction to boolean programs is rather coarse and many unfeasible paths may be encountered by the procedure before finding a feasible one (if any).

We propose linear programs as a model for sequential programs and propose a model-checking procedure for this family of programs. Similarly to boolean

programs, linear programs have the usual control-flow constructs and procedural abstraction with call-by-value parameter passing and recursion. Linear programs differ from boolean programs in that program variables can range over a numeric domain (e.g. the integers or the reals); moreover, all conditions and assignments to variables involve linear expressions, i.e. expressions of the form $c_0 + c_1x_1 + \cdots + c_nx_n$, where $c_0, \ldots, c_n$ are numeric constants and $x_1, \ldots, x_n$ are program variables ranging over a numeric domain. Linear programs are considerably more expressive than boolean programs and can encode explicitly complex correlations between data and control that must necessarily be abstracted away when using boolean programs. However the reachability problem for linear programs is undecidable and therefore any sound and complete procedure for this problem may not terminate.

2 Linear Programs

A *linear program* is a conventional imperative program consisting of a sequence of declarations of global variables followed by a sequence of procedure definitions. Instructions are built out of basic instructions (`skip`, parallel assignment, and procedure invocation) using the usual control-flow constructs (`if-then-else`, `while-do`, `goto`, and `assert`). Instructions can be labeled. Procedures use call-by-value parameter passing and can be recursive. The distinguishing feature of linear programs is that all conditions and assignments to variables involve *linear expressions*, i.e. expressions of the form $c_0 + c_1x_1 + \cdots + c_nx_n$, where $c_0, \ldots, c_n$ are numeric constants and $x_1, \ldots, x_n$ are program variables ranging over the reals.

Let L be a linear program with n instructions and p procedures. We assign to each instruction a unique index from 1 to n and to each procedure an index from $n+1$ to $n+p$. In what follows s_i denotes the instruction associated with index i. For simplicity, we assume that variable and label names are globally unique in L.

With the notation *Globals(L)* we mean the set of global variables of L and so, intuitively, $Locals_L(i)$, will be the set of local variables at instruction s_i, $InScope_L(i)$ the set of visible variables and finally $Formals_L(i)$ and $Actuals_L(i)$ the sets of formal and actual parameters involved in procedure calls.

We briefly give all the definitions needed in the remaining of the paper: $First_L(p)$ denotes the first statement of a procedure p, and $ProcOf_L(i)$ is the procedure to which s_i belongs. $Exit_p$ is the exit node of the procedure and $Succ_L(i)$ the successor relation.

The *control-flow graph* of a linear program L is a directed graph $G_L = (V_L, Succ_L)$. The set $V_L = \{0, 1, \ldots, n+p\}$ contains one vertex for each instruction (vertices $1, \ldots, n$), one exit vertex $exit_p$ for each procedure (vertices $n+1, \ldots, p$), and the vertex 0 used to model the failure of an `assert` statement.

Let D be the domain of computation and $i \in V_L$, then a *valuation* for i is function $\omega : InScope_L(i) \to D$. A *state* of the program is a pair $\langle i, \omega \rangle$ where $i \in V_L$ and ω is a valuation for i. We assume that L contains a distinguished procedure

called `main`. A state $\langle i, \omega \rangle$ is *initial* iff $i = First_L(\texttt{main})$. State transitions in a linear program L are denoted by $\langle i_1, \omega_1 \rangle \rightarrow_L \langle i_2, \omega_2 \rangle$

A *path* is a sequence $\langle i_0, \omega_0 \rangle \rightarrow_L \langle i_1, \omega_1 \rangle \rightarrow_L \cdots \rightarrow_L \langle i_n, \omega_n \rangle$ such that $\langle i_k, \omega_k \rangle \rightarrow_L \langle i_{k+1}, \omega_{k+1} \rangle$ for $k = 0, \ldots, n-1$.

A *valid path* is a path $\langle i_0, \omega_0 \rangle \rightarrow_L \langle i_1, \omega_1 \rangle \rightarrow_L \cdots \rightarrow_L \langle i_n, \omega_n \rangle$ such to capture the transmission of effects from $\langle i_0, \omega_0 \rangle$ to $\langle i_n, \omega_n \rangle$ via a sequence of execution steps which may end with some number of activation records on the call stack. This allows us to reason about non-terminating or abortive executions.

A state $\langle i, \omega \rangle$ is *reachable* iff there exists a valid path from some initial state to $\langle i, \omega \rangle$. A vertex $i \in V_L$ is *reachable* iff there exists a valuation ω such that $\langle i, \omega \rangle$ reachable.

3 Model Checking of Linear Programs

Program reachability can be reduced to computing for each vertex i in the control-flow graph of the program the set of valuations Ω_i such that $\langle i, \omega \rangle$ is reachable iff $\omega \in \Omega_i$. Clearly the statement associated to vertex i is reachable iff Ω_i is not empty. In order to do this efficiently, the model-checking procedure proposed in [1] computes *(i)* "path edges" to represent the reachability status of vertices and *(ii)* "summary edges" to record the input/output behavior of procedures.

Let $i \in V_L$ and $e = First_L(ProcOf_L(i))$. A *path edge* $\pi_i = \langle \omega_e, \omega_i \rangle$ of i is a pair of valuations such that there exists a valid path $\langle First(\texttt{main}), \omega_0 \rangle \rightarrow_L^{\alpha_1} \cdots \rightarrow_L^{\alpha_k} \langle e, \omega_e \rangle$ and a same-level valid path $\langle e, \omega_e \rangle \rightarrow_L^{\alpha_{k+1}} \cdots \rightarrow_L^{\alpha_n} \langle i, \omega_i \rangle$ for some valuation ω_0. In other words, a path edge represents a suffix of a valid path from $\langle First(\texttt{main}), \omega_0 \rangle$ to $\langle i, \omega_i \rangle$.

Let $i \in V_L$ be such that $s_i = pr(e_1, \ldots, e_n)$, let $y_1, \ldots, y_n$ be the formal parameters of pr associated to the actuals $e_1, \ldots, e_n$ respectively, and let $\langle \omega_i, \omega_o \rangle$ a path edge of $exit_{pr}$. A *summary edge* $\sigma = \langle \omega_1, \omega_2 \rangle$ of $\langle \omega_i, \omega_o \rangle$ is a pair of valuations such that

1. $\omega_1(x) = \omega_2(x)$ for all $x \in Locals_L(i)$,
2. $\omega_1(x) = \omega_i(x)$ and $\omega_2(x) = \omega_o(x)$ for all $x \in Globals(L)$, and
3. $\omega_1(e_j) = \omega_i(y_j)$ for $j = 1, \ldots, n$.

Intuitively, a summary edge of i represents information about how the valuation after the procedure call depends on the valuation before the call. The creation of summary edges is the most important and original part in the algorithm: when a summary edge $\langle \omega_1, \omega_2 \rangle$ is built for a procedure, it is no more necessary to study again the same procedure for the input ω_1, because it will be immediately used the output valuation ω_2. In cases of frequently called procedures and of recursion, this will turn into a great improvement in performance.

Our procedure works by incrementally computing the ADLCs representing the set of path edges of the reachable nodes and the summary edges of the procedure calls.

4 The Constraint Solver

Our procedure represents path edges and summary edges symbolically by means of *abstract disjunctive linear constraints* (ADLCs for short), i.e. expressions of the form $\lambda \boldsymbol{x} \lambda \boldsymbol{x}'.c$ where c is a *disjunctive linear constraint* (DLC for short), i.e. a disjunction of conjunctions of linear constraints (which we represent as sets of sets of linear constrains). A *linear constraint* is an expression of the form $e \leq 0$, $e = 0$, or $e \neq 0$ where e is linear expression. Notice that Π is an infinite set of path edges but it is nevertheless compactly represented by the above ADLC. More precisely, let i be a vertex of the control-flow graph, $\boldsymbol{x} = InScope_L(i)$, and $\boldsymbol{x}'$ be a vector of distinguished variables obtained by priming the variables in $\boldsymbol{x}$. The set of path edges associated with i, say Π_i, is represented by the ADLC $\delta = \lambda \boldsymbol{x} \lambda \boldsymbol{x}'.c$ where c is a DLC in the variables $\boldsymbol{x}$ and $\boldsymbol{x}'$. In particular we have that $\Pi_i = \{\langle \omega_1, \omega_2 \rangle \text{s.t. } \omega_1 \cup \omega_2' \text{ satisfies } c(\boldsymbol{x}, \boldsymbol{x}')\}$ where $\omega' = \{\langle \boldsymbol{x}', d \rangle : \langle \boldsymbol{x}, d \rangle \in \omega\}$.

The constrain solver manipulates ADLCs and is abstractly characterized by the following interface functionalities:

- *Application.*
- *Conjunction.*
- *Disjunction.*
- *Projection.*
- *Entailment.*

In our prototype implementation the constraint solver is based on the Fourier-Motzkin elimination method [5].

5 Conclusions

Linear programs are considerably more expressive than boolean programs and thus we believe that their study will allow the construction of better models for minimizing inefficiencies in model description. The model-checking procedure for boolean programs introduced in [1] can be readily used to model-check linear programs by using a constraint solver for linear arithmetics.

In the future, we plan to cope with the non termination of our model-checking procedure by investigating the use of widening techniques [4] [3].

An alternative approach would be to consider subsets of linear arithmetics such as difference constraints [6] (i.e. constraints of the form $x - y \leq c$).

Another important issue is, starting from the backstanding abstraction technique used in the SLAM project and described in [2], to develop a an abstraction tool and so create a stand-alone model checker for real programs.

A final significant step in our research will be to find a suitable set of real problems, which could also suggest some optimizations tailored to this new application domain.

References

1. T. Ball and S.K. Rajamani. Bebop: a symbolic model checker for boolean programs. In *Workshop on Model-Checking of Software (SPIN 2000). August-september. LNCS*, volume 1885, pages 113–130, 2000.
2. T. Ball and S.K. Rajamani. Boolean programs: a model and process for software analysis. In *MSR Technical Report*, volume 14, 2000.
3. P. Cousot and R. Cousot. Abstract interpretation: a unified lattice model for static analysis of programs by construction or approximation of fixpoints. In *Conference Record of the Fourth Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, pages 238–252, Los Angeles, California, 1977. ACM Press, New York, NY.
4. N. Halbwachs, Y.-E. Proy, and P. Raymond. Verification of linear hybrid systems by means of convex approximations. In *International Static Analysis Symposium, SAS'94*, Namur (Belgium), September 1994.
5. J.-L. Lassez and M.J. Maher. On Fourier's Algorithm's for Linear Arithmetic Constraints. *Journal of Automated Reasoning*, 9:373–379, 1992.
6. J. Møller, J. Lichtenberg, H. R. Andersen, and H. Hulgaard. Difference decision diagrams. Technical Report IT-TR-1999-023, Department of Information Technology, Technical University of Denmark, Building 344, DK-2800 Lyngby, Denmark, February 1999.

The Use of Data-Mining Techniques for the Automatic Formation of Proof Plans

Alan Bundy John Levine Hazel Duncan

July 1, 2002

1 The Authors

Prof. Alan Bundy and Dr John Levine are members of the Mathematical Reasoning Group (MRG) and the Artificial Intelligence Applications Institute, which are both part of the Centre for Intelligent Systems and their Applications within the Division of Informatics at the University of Edinburgh. The Division of Informatics was one of only six computing departments in UK to have obtained a 5* ranking in the 2001 Research Assessment Exercise. It returned the highest number of research active staff and was the only 5*A department. It contains world-class research groups in the areas of theoretical computer science, artificial intelligence and cognitive science.

1.1 Prof. Alan Bundy

Prof. Bundy has been active in automated mathematical reasoning research since 1971 and has become a world authority. His international reputation is witnessed by his being made a founding fellow of both of the two international AI societies: AAAI and ECCAI, in addition to the UK society, AISB, and serving terms as Chair of both IJCAI Inc and CADE Inc. He is also a Fellow of the Royal Society of Edinburgh and of the British Computer Society. He won the SPL Insight Award in 1986, was an SERC Senior Fellow (1987-92), a member of the Hewlett-Packard Research Board (1989-91), Head of the Division of Informatics at Edinburgh (1998-2001), a member of the ITEC Foresight Panel (1994-96), a member of the Computer Science RAE panel (1999-2001) and is the founding Convener of UKCRC (2000-date). He is the author of over 140 publications.

1.2 Dr. John Levine

Dr. John Levine is an Informatics Research Fellow at AIAI working on AI planning systems and evolutionary computation. His current research program consists of applying genetic search and learning techniques to AI planning problems, applying AI planning and evolutionary computation techniques to real-world problems, and mixed-initiative systems for planning and optimisation. He has

an MA in Computer Science, an MPhil in Computer Speech and Language Processing, and a PhD in Computer Science, all from the University of Cambridge. He is a member of the review board of Applied Intelligence Journal and is the chair of the PLANSIG 2001 conference on AI Planning and Scheduling systems. He has taught a wide range of undergraduate, postgraduate and commercial courses in Artificial Intelligence, including Common Lisp, Planning and Search, Knowledge Representation, Fuzzy Logic and Practical Reasoning Methodologies.

1.3 Miss. Hazel Duncan

Miss. Hazel Duncan is completing an Honours degree in Maths and Artificial Intelligence from the University of Edinburgh and will graduate in 2002. Her final-year project involved mathematical theorem proving using Isabelle.

2 Description of Proposed Research and its Context

This project will explore the application of data-mining techniques to the automatic construction of proof plans from a large corpus of proofs.

2.1 Background

Proof planning is a technique for the guidance of proof search in automated theorem proving. Until recently, this has required the manual construction of proof plans. This is a time-consuming and highly skilled process which has limited the more widespread uptake of this technology. The automatic construction of proof plans would remove this impediment.

2.1.1 Proof Planning

A *proof method* is the computational representation of a common pattern of proof in a family of related proofs [Bundy, 1991]. A *proof critic* similarly represents a common technique for patching an initially failed proof attempt [Ireland and Bundy]. Both methods and critics consist of a tactic plus a specification of its preconditions and effects, expressed in a meta-language describing the syntactic properties of the formulae input to and output by the tactic. A *tactic* is a computer program for applying the rules of inference of a mathematical theory [Gordon *et al*, 1979]. Tactics are combined with tacticals to produce higher-level tactics. *Tacticals* include operations of sequencing, nondeterminism and repetition. *Proof planning* is the use of AI plan formation technology to guide proof search by constraining it to a set of proof methods and critics. Proof planning limits the combinatorial explosion of potential proof steps, which occurs if exhaustive search techniques are used.

The MRG invented the technique of proof planning, implemented it in the *λClam* proof planner [Div2000] and applied it particularly to the kind of inductive proofs that arise in verification and synthesis of IT systems. It has extended the range of problems that can be solved without human intervention. In particular, the use of proof critics has automated the discovery of intermediate lemmas and generalisations [Ireland and Bundy] – so called, "eureka" steps, which were previously thought to require human intervention.

2.2 Previous Work on Learning Proof Methods

There have been several previous attempts to learn new proof methods from example proofs.

In his PhD project with MRG, Bernard Silver applied techniques of explanation-based learning to the automated learning of proof methods for equation solving [Silver,1984]. His Learning-Press system analysed successful solutions to equations and generalised these solutions to form methods for guiding the Press system. In this way, he was able to automatically rediscover simplified versions of many of the previously hand-coded methods of Press.

Similarly, another MRG PhD, Roberto Desimone, automated the reconstruction of inductive proof plans, [Desimone, 1989]. The techniques of both Silver and Desimone generalised from single successful proofs and required the system to be primed with some key meta-level concepts for expressing the preconditions and effects of the methods they learnt.

More recently, Kerber, Jamnik and Benzmüller, from Birmingham University, have applied the techniques of least general generalisation to a family of similar proofs to learn new proof methods for algebraic reasoning [Jamnik *et al*, 2000]. Note that this technique required all the proofs in the family to be examples of the learned method.

2.3 Data-Mining

In this project we intend to data-mine a large corpus of proofs to extract new proof tactics. Unlike the previous projects at Edinburgh, we will initially only attempt to learn the tactics, but not the *specifications* of these tactics, which would also be required in order to learn the proof methods. This simplification will enable us to postpone the problem of either anticipating or learning the meta-language that the tactics will require. Unlike the previous project at Birmingham, we will not assume that the proofs in the corpus are all examples of the tactic to be learnt. Rather we will try to identify common patterns of proof within a heterogeneous corpus. Note that a pattern may correspond to only part of a proof.

The data-mining will consist of two stages. Firstly, frequently occurring sequences of proof steps will be identified using probabilistic reasoning. Secondly, these sequences will be used to seed a genetic programming process to combine the sequences with tacticals. This two stage process will create a family of tactics, whose effectiveness will then be evaluated.

2.4 Identifying the Corpus of Proofs

The corpus of proofs to be used in this project must meet exacting requirements.

1. It must be stored in computational form, so that it is available for data-mining.
2. It must be sufficiently large to contain many examples of multiply occurring patterns of proof.
3. There must be an appropriate diversity of kinds of proof steps, *i.e.* sufficient different kinds of proof steps that patterns can be identified, but not so much diversity that patterns do not recur. Note that the appropriateness of diversity is relative to corpus size: the larger the diversity the larger the corpus required for the re-occurrence of patterns.

We have identified several corpuses that may meet these requirements.

Note first that the huge search spaces generated by resolution-style theorem provers are, unfortunately, mostly unsuitable because of requirement 3 above: typically only one or two rules of inference are used. We could try to differentiate rule applications by the formulae they manipulate, but these formulae are generated during the proof search and are too diverse, *e.g.* millions of derived clauses. A possible exception that deserves examination is model elimination [Loveland, 1978], in which one parent clause of each non-ancestor resolution must be an input clause – thus limiting the diversity. Using search spaces, rather than proofs, would have the potential advantage that negative information from the unsuccessful branches could also be used in tactic formation, so this possibility is well worth investigation.

Better corpus candidates may be generated by the sequent calculus based theorem provers, since the wider range of rules of inference provides the necessary diversity to meet requirement 3. Most of these provers are interactive, so the proofs have been structured by human users, making it more likely that proof patterns are present[1]. Also, most of these provers are tactic based; these tactics provide additional proof step diversity and permit our new tactics to be built on top of existing ones: leading to a hierarchically structured family of tactics, as required in proof planning.

One set of candidates arises from the Isabelle prover [Paulson, 1986]. Jacques Fleuriot, in MRG, has developed a corpus of several thousand proofs in non-standard analysis. Worldwide, there are many other users with similar corpuses. Unfortunately, there is a technical problem that Isabelle normally stores its proofs as derived rules generalised from the proof using explanation-based learning. If this technical problem can be overcome then Isabelle could provide some ideal corpuses.

Another candidate is the Mizar proof library [Rudnicki, 1992]. This contains several thousand proofs of major theorems in Mathematics, built up over several

[1] In automatically generated proofs it is more likely that two or more proof patterns are interleaved, making them hard to detect.

decades. There are some technical problems in making this library available, but Paul Jackson, in MRG, is already studying this problem.

References

[Bundy, 1991] Bundy, Alan. (1991). A science of reasoning. in Lassez, J-L. and Plotkin, G., (eds.), *Computational Logic: Essays in Honor of Alan Robinson*, pages 178-198. MIT press. Also available from Edinburgh as DAI Research paper 445.

[Desimone, 1989] Desimone, R. V. (1989). *Learning Control Knowledge within an Explanation-Based Learning Framework.* Unpublished Ph.D. thesis, Dept. of Artificial Intelligence, University of Edinburgh.

[Div2000] Division of Informatics, University of Edinburgh, Edinburgh. (v2000). *User/Programmer Manual for the λClam proof planner*, v2.0.0 edition.

[Gordon *et al*, 1979] Gordon, M. J., Milner, A. J. and Wadsworth, C. P. (1979). *Edinburgh LCF - A mechanised logic of computation*, volume 78 of *Lecture Notes in Computer Science.* Springer-Verlag.

[Ireland and Bundy] Ireland, A. and Bundy, A. (1996). Productive use od failure in inductive proof. *Journal of Automated Reasoning*, 16(1-2):79-111. Also available from Edinburgh as DAI Research Paper No 716.

[Jamnik *et al*, 2000] Jamnik, M., Kerber, M. and Benzmüller, C. (2000). Towards learning new methods in proof planning. In *Proceedings of the 2000 Calculemus Symposium: Systems for Integrated Computation and Deduction.* Also presented at CADE 2000 Workshop: The role of Automated Deduction in Mathematics, Pittsburgh, USA.

[Loveland, 1978] Loveland, D. W. (1978). *Automated theorem proving: A logical basis*, volume 6 of *Fundamental Studies in Computer Science.* North Holland.

[Paulson, 1986] Paulson, L. C. (1986). Natural deduction as higher order resolution. *Journal of Logic Programming*, 3:237-258

[Rudnicki, 1992] Rudnicki, P. (1992). An overview of the Mizar project. In *1992 Workshop on Types for Proofs and Programs*, Bastad. Chalmers University of Technology. See http://mizar.org for up-to-date information on Mizar and the Journal of Formalized Mathematics.

[Silver,1984] Silver, B. (1984) *Using Meta-Level Inference to Constrain Search And To Learn Strategies In Equation Solving.* Unpublished Ph.D. thesis, Dept. of Artificial Intelligence, University of Edinburgh, Published as a book by North Holland.

Proof Automation for High Integrity Software Engineering

Bill J. Ellis*

School of Mathematical and Computer Sciences
Heriot-Watt University
Riccarton, Edinburgh, Scotland, EH14 4AS
E-mail: bill@cee.hw.ac.uk

1 Introduction

Critical systems are those where error is unacceptable. Historically, critical systems have been restricted to safety-critical systems, where failure could risk human life. However, the economic and technological advantages of computers have led to wide spread use of computer systems, resulting in other categories of systems that must meet the same high integrity constraints.

There is a need to develop tools and techniques for constructing the high integrity software demanded of critical systems. Praxis Critical Systems have had commercial success developing critical systems using their SPARK approach [2]. However, the SPARK approach does not exploit modern techniques in automated reasoning. It is our intention to enhance the applicability of the SPARK approach by extending their proof checker SPADE using existing and developing new techniques in automated reasoning.

2 Overview

2.1 Software Verification

Software verification involves constructing a formal proof that a program meets its specification. Our work is intended for procedure based imperative programming languages.

Floyd [10] and Hoare [14] introduced an inductive assertion method for proving the partial correctness of software, that if a program terminates, it meets its specification. To support this a formal description of the semantics of the programming is supplied as Hoare style proof rules. The inductive assertion

*Supported by the EPSRC Critical Systems Programme - EPSRC grant GR/R24081.

method involves adding the program specification to procedures in the form of assertions. A precondition assertion is added stating what must be true when entering the procedure and a postcondition is added to state what must be true if the procedure terminates. An invariant or inductive assertion is required to specify the behaviour within each loop. Conjectures called verification conditions (VCs) are automatically generated for each path of execution between assertions, or in the case of paths round loops, from the invariant back to itself. Proving these VCs is sufficient to prove that the program is partially correct. Conversely, proving that any one of these VCs is false will prove that their is an error in the specification, code or both.

The Floyd-Hoare method for proving programs was traditionally conducted in a batch style after the program has been written. It is often the case that many of resulting VCs are trivial, and can be discharged by an automatic theorem prover. Those VCs that can not be discharged are then tackled inside an interactive theorem prover.

A number of systems have adopted this batch style, proving many programs. However, it became increasingly clear that the batch style did not scale up to larger problems. It became difficult to determine if failed VCs relate to errors in the code or specification and which parts of these. Further, the process of generating good invariants for code from scratch is difficult. Finally, the proof effort required to discharge the VCs typically involved frequent and sophisticated human interaction.

Building on the work of Floyd-Hoare, Dijkstra [9] elaborated on the inductive assertion method, adding a proof of termination to the proof of partial correctness to give total correctness. Further, Dijkstra refined the methods by which programming language semantics are described using predicate transformers over Hoare proof rules. Most significantly, Dijkstra and Gries [12] advocated that a program and proof should be developed in cooperation, with the proof ideas leading the way. Although not overcoming the difficulties of Floyd-Hoare verification, this cooperation style has the potential to lessen the problems.

Praxis Critical Systems is a commercial company building high integrity software. They advocate correctness by construction [13, 8] building on the Dijkstra-Gries cooperation style of software verification by considering issues of verification from the start of software development. The SPARK approach [2] has been shown to be successful in developing a number of critical systems (LOCKHEED C130J [8, 7], CA [13, 7]). Of particular merit, the SPARK approach was used to build the first system (SHOLIS [19, 7]) that met the stringent standards of the MOD Defence Standards 00-55 and 00-56 for SIL4, the most critical class of software.

Underpinning the SPARK approach is a formal description of the semantics for a subset of Ada[1]. The subset has been carefully carved out of Ada to provide an unambiguous, useful and most significantly provable programming language called SPARK. In addition to the functional Ada parts of SPARK, the language also allows additional information to be provided through annotations.

[1] Subsets are defined for both Ada 83 and Ada 95.

These include information and data flow annotations and Floyd-Hoare inductive assertions. The SPARK approach prescribes best practise for using the SPARK language as REVEAL for requirements capture and INFORMED [1] for software design.

There are two different kinds of tools supporting the SPARK approach. The EXAMINER performs static analysis on SPARK code. It enforces the SPARK syntax, performs sophisticated static analysis using the flow annotations [4], and generates VCs for proof of exception freedom (no run time errors) and partial correctness. The other tools are geared toward formal proof. The SIMPLIFIER is a trivial VC eliminator and SPADE is an interactive theorem prover.

In spite of the successes of SPARK it does not overcome the difficulties in performing Floyd-Hoare verification. It is not uncommon for several months to be spent tackling the VCs for a realistically sized SPARK project.

2.2 Automated Reasoning

Two broad styles of automated reasoning have emerged, machine and human oriented. The machine oriented approach replaces the human operator with an algorithm executed by the computer. To provide full automation, machine oriented approaches necessarily operate in logics where the cut rule[2] is not available. This prevents machine oriented approaches from providing full automation where reasoning about iteration, a key concept in software verification. Machine oriented approaches include decision procedures (Davis-Putnum, Presenburger Arithmetic) uniform proof procedures (Resolution) and analytical tools (constraint logic programming, computer algebraic systems).

The human oriented approach relies on interaction with the user where automation is unsuccessful. Automation is achieved through heuristic strategies. Heuristics are general rules of thumb and it is expected that they will sometimes fail. Proof planning [6, 5] provides a mechanism to make explicit heuristic search strategies. It separates the search for a proof from the soundness of the logical argument. This decoupling means that flexible search strategies can be used. Tactics are procedures that perform large grain proof steps, following the LCF style of theorem proving [11]. Methods are partial tactic specifications, outlining the preconditions that should be true before applying a tactic and the resulting effects having applied the tactic. Proof planning involves a search using the methods alone. During this search methods may fail to apply and critics [15, 17] may be invoked to patch the proof. Critics have been successfully applied in patching induction proofs [16, 17, 18]. Once a proof plan is found it is converted into its corresponding tactic sequence and executed inside a sound theorem prover. Proof planning has been implemented in ΩMEGA [3], CLAM and λ-CLAM. A project also practically demonstrated the separation between soundness and search by changing the object level of the CLAM proof planner from OYSTER to HOL in CLAM-HOL [20].

[2] Of the form: $\frac{\Gamma \vdash \alpha \quad \Gamma, \alpha \vdash \beta}{\Gamma \vdash \beta}$

2.3 Progress Toward Automated VC Proof

There has been extensive research on automating the discovery of loop invariants. Key to performing this is information discovery. Most research has been placed on gaining the additional information by employing a top-down strategy, where information is extracted from the program specification. This strategy has been implemented successfully within the proof planning paradigm using critics to automate the discovery of invariants and lemmas [21]. An opposing bottom-up approach is also applicable, extracting information directly from the program source code. The branching factor of a bottom-up strategy is high, requiring some constraints to to provide more targeted information.

Industrial strength verification systems have yet to adopt these techniques for automating the proof of VCs. There is a need to develop systems in the spirit of CLAM but with an industrial context.

3 Research Proposal

The SPARK approach stands out as one of the few examples of software verification being successfully applied in industry. However, the SPARK approach has not directly addressed the difficulties associated with performing Floyd-Hoare verification. Research into automated reasoning has produced many promising ideas for automating the proof of theorems in general and VCs in specific. However, the exploration of this research has typically been outside an industrial setting.

We propose to further enhance the applicability of the the SPARK approach by improving the automation of its tools supporting formal proof. The proof planning paradigm will be used as a flexible framework within which this automation can be explored. Although focus will be placed on the SPARK verification problems, as a regular imperative language the techniques discovered are likely to be useful in other contexts. Our research hypothesis is:

> *Significant productivity gains can be made in automating the verification of high integrity software by developing new heuristics and implementing these within the proof planning paradigm.*

Some possible classes of heuristics have already been identified. Exploiting existing tools inside a NuSPADE context could bring some benefits. Decision procedures, uniform proof procedures and analytical tools may all have a role to play as heuristics. Further, information discovery is key for supporting many heuristics. The strong foundations for the top-down approach can be built upon, while some exploratory work using bottom-up techniques may also be advantageous. Finally, proving that a VC is false indicates that there is an error in the specification, code or both. Heuristics could be constructed that target such non-theorems. Experience from the testing community may bring valuable insights into where to look for non-theorems.

References

[1] Peter Amey. The informed design method for spark, 1999, 2001.

[2] J. G. P. Barnes. *High Integrity Ada The SPARK Approach*. Addison-Wesley, 1997.

[3] C. Benzmüller, L. Cheikhrouhou, D Fehrer, A. Fiedler, X. Huang, M. Kerber, K. Kohlhase, A Meirer, E. Melis, W. Schaarschmidt, J. Siekmann, and V. Sorge. Ωmega: Towards a mathematical assistant. In W. McCune, editor, *14th International Conference on Automated Deduction*, pages 252–255. Springer-Verlag, 1997.

[4] Jean-Francois Bergeretti and Bernard A. Carr. Information-flow and data-flow analysis of while-programs. *ACM Transactions on Programming Languages and Systems*, 7(1), 1985.

[5] A. Bundy. A science of reasoning. Research Paper 445, Dept. of Artificial Intelligence, University of Edinburgh, 1989.

[6] Alan Bundy. The use of explicit plans to guide inductive proofs. Research Paper 349, Dept. of Artificial Intelligence, University of Edinburgh, 1988.

[7] R. Chapman. Industrial experience with spark, 2000.

[8] Martin Croxford and James Sutton. Breaking through the v and v bottleneck, 1995.

[9] E. Dijkstra. *A Discipline of Programming*. Prentice-Hall, 1976.

[10] R. W. Floyd. Assigning meanings to programs. In J. T. Schwartz, editor, *Mathematical Aspects of Computer Science, Proceedings of Symposia in Applied Mathematics 19*, pages 19–32. American Mathematical Society, 1967.

[11] M. J. Gordon, A. J. Milner, and C. P. Wadsworth. *Edinburgh LCF - A mechanised logic of computation*, volume 78 of *Lecture Notes in Computer Science*. Springer-Verlag, 1979.

[12] David Gries. *The Science of Programming*. Springer-Verlag, New York, 1981.

[13] A. Hall and R. Chapman. Correctness by construction: Developing a commercial secure system, 2002.

[14] C.A.R. Hoare. An axiomatic basis for computer programming. *Communications of the ACM*, 12:576–583, 1969.

[15] A. Ireland. The Use of Planning Critics in Mechanizing Inductive Proofs. In A. Voronkov, editor, *International Conference on Logic Programming and Automated Reas oning – LPAR 92, St. Petersburg*, Lecture Notes in Artificial Intelligence No. 624, pages 178–189. Springer-Verlag, 1992.

[16] A. Ireland and A. Bundy. Extensions to a Generalization Critic for Inductive Proof. pages 47–61. Springer-Verlag, 1996. Springer Lecture Notes in Artificial Intelligence No. 1104.

[17] A. Ireland and A. Bundy. Productive use of failure in inductive proof. *Journal of Automated Reasoning*, 16(1–2):79–111, 1996.

[18] A. Ireland and A. Bundy. Automatic Verification of Functions with Accumulating Par ameters. *Journal of Functional Programming: Special Issue on Theorem Proving & Functional Programming*, 9(2):225–245, March 1999.

[19] S.King, J. Hammond, and R. Chapman A. Pryor. Is proof more cost-effective than testing? volume 26, pages 675–686. IEEE, 2000.

[20] K. Slind, M. Gordon, R. Boulton, and A. Bundy. System description: An interface between CLAM and HOL. volume 1421, pages 134–138, Lindau, Germany, July 1998. Springer.

[21] J. Stark and A. Ireland. Proof planning for strategy development. In *Annals of Mathematics and Artificial Intelligence*.

Cooperation between the Mathematical Knowledge Base MBase and the Theorem Prover ΩMEGA

Andreas Franke, Markus Moschner, Martin Pollet

Department of Computer Science, University of Saarbrücken, Germany;
{afranke,moschm,pollet}@mathweb.org

The poster will describe the cooperation between the mathematical knowledge repository MBase [5] and the mathematical proof assistant ΩMEGA [2]. The cooperation aims to assist a user of ΩMEGA during the proof construction, namely to suggest possible theorems and lemmas that could be applied in the current proof situation. For this task, we exploit the particular features provided by the two systems. MBase is a repository for mathematical content ranging from mathematical texts to formal theories. Besides simple retrieval functionality MBase allows to find specific pieces of knowledge by structural queries. ΩMEGA manages the process of proof construction integrating different approaches: interactive and automated theorem proving, proof planning, use of external systems.

MBase

MBase is a mathematical knowledge management system based on OMDoc [4], which is an extension of the OpenMath standard (http://www.openmath.org/). OpenMath provides a universal syntax for formal mathematical objects and is appropriate for communication with automated deduction services and computer algebra systems. Based on this representation for terms, OMDoc can be used to represent mathematical documents in a varying degree of formalization, ranging from text in natural language containing OpenMath formulae up to completely formalized theories. From a practical point of view, the MBase system can be seen as a repository of OMDoc collections, which can be accessed and searched by humans and software systems. Thus MBase provides separate interfaces for these two different types of "clients": one can browse through MBase content via the web interface whereas software systems can access it via an *XML-RPC* interface.

One query service offered by MBase is the search for theorems containing a subterm that matches a given pattern. We are using simple structural matching here. In principle, one can supply any OpenMath object as a pattern. Additionally, meta-variables can be used to match arbitrary subterms consistently. Examples for patterns are commutativity $f(x,y) = f(y,x)$ or monotonicity $f(x,y) \Rightarrow f(g(x),g(y))$ where f, g, x, y are meta-variables and $=, \Rightarrow$ are OpenMath symbols.

ΩMEGA

ΩMEGA is designed as a mathematical assistant for the construction of mathematical proofs. The main features of ΩMEGA are automatic generation of proofs based on the proof planner Multi and the support of interactive theorem proving with the agent-based command suggestion mechanism ΩAnts [1]. Proofs are represented in the three-dimensional proof data structure $\mathcal{PDS}$ that allows to have proofs at different levels of abstractness with the base calculus as least abstract level. ΩMEGA has a graphical user interface called *LΩUI* [3], allows for the verbalization of formal proofs into natural language via the proof explanation system *P.rex* and has access to different mathematical services via *MathWeb* [4]. Available mathematical services are first-order and

higher-order automated theorem proving systems, computer algebras systems, model checkers, constraint solvers, and the mathematical knowledge base MBASE.

MBASE is used to store and provide theories (containing the signature, definitions, theorems and problems) of formalized mathematics for ΩMEGA.

Cooperation between MBASE and ΩMEGA

On the side of ΩMEGA the ΩANTS mechanism is used to suggest possible commands to the user. ΩANTS is a blackboard architecture where for each inference rule and its arguments so-called agents check for possible instantiations of the inference rule in the current proof context. The inference rules are either rules of the basic calculus, tactics or methods from proof planning, and have premises, conclusions, and possibly additional parameters as arguments. If the agents found instantiations for inference rules, the corresponding commands are presented to the user in a window of the graphical user interface *LΩUI*.

We implemented two commands for the application of theorems and the corresponding agents in ΩMEGA, namely assertion-application and rewrite-application. The first command applies a theorem of the form $\forall_{x_1,\ldots,x_n} P_1 \wedge \ldots \wedge P_m \Rightarrow C$ when C matches with the formula to prove and introduces $P_1, \ldots, P_m$ as new unproved formulas. The latter applies conditional equations $\forall_{x_1,\ldots,x_n} P_1 \wedge \ldots \wedge P_m \Rightarrow t_1 = t_2$ where t_1 (or t_2) matches a subterm of the formula to prove and introduces $P_1, \ldots, P_m$ and the old formula where t_1 is replaced by t_2, or t_2 is replaced by t_1 resp., as new unjustified formulas.

Agents that try to provide a suggestion for a theorem would have to test all theorems which is rather expensive. At this point we use the pattern matching facilities of MBASE to get a preselected set of theorems which are tested for applicability. In detail, MBASE is queried for theorems containing a subformula matching the current formula to prove in case of assertion-application, or for an equation containing a subterm of the current goal as right hand side or left hand side of an equation for rewrite-application. The argument agent filters from this preselection all theorems that are not of the right form or not applicable and presents the remaining theorems to the user.

WWW–References:

- MBASE: `http://www.mathweb.org/mbase`
  Demo `http://mbase.mathweb.org:8080/mbase/`
- ΩMEGA: `http://www.ags.uni-sb.de/~omega/`
  Demo `http://www.ags.uni-sb.de/~omega/demo/`

References

[1] Christoph Benzmüller and Volker Sorge. Oants – an open approach at combining interactive and automated theorem proving. In M. Kerber and M. Kohlhase, editors, *Proc. of the Calculemus Symposium 2000*, St. Andrews, UK, 6–7 August 2000. AK Peters, New York, NY, USA.

[2] Christoph Benzmüller et al. ΩMEGA: Towards a mathematical assistant. In William McCune, editor, *Proc. of the 14th Conference on Automated Deduction*, number 1249 in *LNAI*, pages 252–255, Townsville, Australia, 1997. Springer Verlag.

[3] Jörg Siekmann et al. Loui: Lovely omega user interface. *Formal Aspects of Computing*, 11:326–342, 1999.

[4] Andreas Franke and Michael Kohlhase. System description: MATHWEB, an agent-based communication layer for distributed automated theorem proving. In Harald Ganzinger, editor, *Proc. of the 16th Conference on Automated Deduction*, number 1632 in LNAI, pages 217–221. Springer Verlag, 1999.

[5] Andreas Franke and Michael Kohlhase. System description: MBASE, an open mathematical knowledge base. In David McAllester, editor, *Automated Deduction – CADE-17*, number 1831 in LNAI, pages 455–459. Springer Verlag, 2000.

Extended Abstract

Normalforms and Termorders for Recognizing Types of Equations

Matthias Goldgruber

August 14, 2002

University of Technology
Institute for Softwaretechnology
Inffeldgasse 16b/2 8010 Graz
Austria

0043 316 817356
goldy@sbox.tugraz.at

1 Introduction

Algebra systems solve equations with just one function (e.g. Solve) by deciding to which equation type the given equation belongs to, and by choosing the right method for solving that equation type. In order to acchieve this functionality, the equation solvers of algebra systems implement a sophisticated search on a huge hierarchy of equation types.

The equation solvers of algebra systems do their work hidden from the user, and just come up with the solution. The *ISAC*-project, in contrary, aims at making the functions transparent to the user. In the case of equation solving this means to show how the system determines the type of the equation, which is the first step of solving the equation.

Another aim of *ISAC* is to provide for a sound logical base for doing mathematics. As I have been told so far, this aim is not my concern, except an involvement in typing (*ISAC* is based on Isabelle and uses the types implemented in HOL) and sub-typing, eventually.

In this extended abstract I will describe the problems I have to solve within my task in the *ISAC*-project, and I will give an example, and show how the task can be done using *ISAC*'s mathematics engine so far. The notions *pattern*, *matching*, *normal form*, *orderd rewriting*, *term order* , *rewriting*, *rewrite system*, *reduction system*, *reduction order* I use as defined in [Baader 1998].

2 My task within the *ISAC*-project

2.1 Equation types

An equation type can be discribed with a *pattern*. For example the pattern for a linear eqaution could look like

$$a + b * boundVariable = 0$$

where a and b stand for any (sub-)term *not* containing *boundVariable*, the bound variable of the equation. The equation

$$4 + 2 * x = 0$$

matches the pattern if x is declared to be the bound variable.

2.2 Normal forms

The equation

$$1 + 2 * x * r + s = 0 \tag{1}$$

does not match the pattern although it is an linear eqaution in x. For matching the pattern it first has to be transformed into a *normal form*. For the above example the normal form would be

$$(1 + s) + (2 * r) * x = 0 \tag{2}$$

if x is the bound variable. If r is the bound variable the normal form would be

$$(1 + s) + (2 * x) * r = 0.$$

2.3 Ordered rewriting

The standard method for transforming equations in such a way is *ordered rewriting*. Therefore you need a *set of (rewrite-)rules* and a *term order*. To transform the above equation (1) into the normal form (2) you would need the commutative rules for + and *. So the respective rule set would look like

$$\begin{array}{ll} r_1: & (n + m \Rightarrow m + n) \\ r_2: & (n * m \Rightarrow m * n) \end{array} \quad (3)$$

In the first step you need to use the rule r_1 to change $2 * x * r + s$ to $s + 2 * x * r$ and then you need to use the rule r_2 to change $2 * x * r$ to $2 * r * x$. To use rules for transforming eqautions into a different form is called *rewriting* and the set of such rewrite rules is called the *rewrite system*.

If you want to automate the transformation the system needs to be smart enought to decide, when to use which rule to get the desired result (e.g. the normal form). It also could happen that the transformation doesn´t terminate. For example if the system would apply the rule r_1 to the same two operands consecutively and thus never would terminate.

Therefore you need an order which can compare two terms in respect to special criterias. In *Ordered rewriting* you just apply a rule to a term if the resulting term is smaller in respect to the order. Such a rewrite system is also called a *reduction system* with a respective *reduction order* .

2.4 The scope of my task

My task is restricted to all equations which are taught at Austrian high-schools (actually to all equations contained in a frequently used textbook). For these equations I have to do the following:

1. determine normal forms for all equation types such that they are distinct
2. design a hierarchy for the equation types such that they can be searched efficiently
3. for each normal form search for an appropriate term order, and develop a toolbox of term orders.

3 Some trials with *ISAC*'s math engine

On this point I would like to show with an example how *ISAC*'s math engine works.

I will take the above equation (1) and show how to get it into the normal form (2) using the *ISAC*'s math engine. First I want to do this step by step using the rules in (3) and then automate the procedure by using the rules as an rule set with a term order.

3.1 Step by step calculation

The rules (3) implemented in *ISAC* look like the following:

```
ML> radd_commute;
val it = "?m + ?n = ?n + ?m" : thm
ML> rmult_commute;
val it = "?m * ?n = ?n * ?m" : thm
```

The equation (1) is stored in the variable t and has the following notation:

```
ML> val t = "#1 + #2 * x * r + s = #0";
val t = "#1 + #2 * x * r + s = #0" : string
```

The equation can be transformed to the normalform in two steps using the function *rewrite* with some additional parameters, applying the two rules to it:

```
ML> val Some (t,_) =
    rewrite "LinArith.thy""tless_true""e_rls"false ("radd_commute","") t;
val t = "s + (#1 + #2 * x * r) = #0" : string

ML> val Some (t,_) =
    rewrite "LinArith.thy""tless_true""e_rls"false ("rmult_commute","") t;
val t = "s + (#1 + r * (#2 * x)) = #0" : string
```

3.2 Automated calculation

In *ISAC*you can implement a rule set as a set of *theorems*:

```
ML> assoc_add_mult;
val it =
  Rls
    {preconds=[],rew_ord=("tless_true",fn),
     rules=[Thm ("radd_commute","?m + ?n = ?n + ?m"),
            Thm ("rmult_commute","?m * ?n = ?n * ?m")],
     scr=Script (Free ("xxx","RealDef.real"))} : rls
```

The term order *rew-ord* is an order which prefers the bound variable shifted to the right within a term. The system now tries to apply one rule after the other to the equation but just doing so if the term is getting smaller. If it can´t apply a rule any more the system stops and returns the result.

```
ML> val Some (t,_) = rewrite_set "LinArith.thy" "eval_rls" false "assoc_add_mult" t;
### trying thm 'radd_commute'
### rewrite_set_: s + (#1 + #2 * x * r) = #0
### trying thm 'radd_commute'
```

```
### not: "s + (#1 + #2 * x * r)" > "#1   #2 * x * r   s"
### not: "#1 + #2 * x * r" > "#2 * x * r + #1"
### trying thm 'rmult_commute'
### rewrite_set_: s + (#1 + r * (#2 * x)) = #0
### trying thm 'rmult_commute'
### not: "r * (#2 * x)" > "#2 * x * r"
### not: "#2 * x" > "x * #2"
### trying thm 'radd_commute'
### not: "s + (#1 + r * (#2 * x))" > "#1 + r * (#2 * x) + s"
### not: "#1 + r * (#2 * x)" > "r * (#2 * x) + #1"
### trying thm 'rmult_commute'
### not: "r * (#2 * x)" > "#2 * x * r"
### not: "#2 * x" > "x * #2"
val t = "s + (#1 + r * (#2 * x)) = #0"   string
```

4 Summary

This abstract presents the problem I have to solve within my diploma thesis and within the *ISAC*-project: use term orders for rewriting to normal forms, which can be matched with patterns of types of equations.

My work started a few weeks ago, and I hope to proceed during the time until the Calculemus Autumn School such that I may present my work in a more advanced way. And I hope to get background information (deduction systems, partiality etc.) on the work I am going to do.

References

[Baader 1998] Baader, F.; Nipkow, T.: *Term Rewriting and All That*; Cambridge University Press, Cambridge (1998).

[Dick 1990] Dick, J.; Kalmus, J.; Martin, U.: *Automating the Knuth Bendix ordering*; Verlag Springer, Acta Informatica, 28:95-119 (1990).

Supporting Interactive Theorem Proving in a calculus-free Framework

Malte Hübner

Fachbereich Informatik, Universität des Saarlandes
D-66041 Saarbrücken, Germany

Abstract. This paper describes work aimed at extending the recently developed proof-planning framework CORE [1] by a suggestion mechanism that supports interactive proof search.

1 Introduction

Problems inherent to fully automated theorem provers as well as the need for intelligent mathematical assistant systems have led to a growing interest in interactive theorem proving environments. However, most of the current systems are not very intuitive to use and require a strong background in mathematical logic in order to apply them successfully.
These shortcomings are mainly caused by the fact that interactive theorem provers are usually too tightly related to a logical calculus: Interactive proof search typically proceeds by sequentially selecting and applying inference rules of the calculus that underlies the respective system. In order to assist the user, these systems are often able to pre-select a set of applicable rules from the set of all available inference rules and might even be able to automatically instantiate parameters of these rules (cf. the Ω-ANTS [3] suggestion mechanism of the ΩMEGA [2] system).
However, the need to apply rules from a given calculus commits the user to perform very fine grained proof steps that often do not correspond to the broader steps that one would find in a proof of a human mathematician. Moreover, most calculi make it necessary to perform many proof steps that would not occur in human generated text book proofs at all. One example is focusing on subtasks which is often done automatically by humans but which is very hard to mimic when being committed to the application of a fixed set of calculus rules.
As an attempt to overcome these problems Autexier [1] recently presented a more intuitive proof framework which does not directly depend on any calculus. Rather, the system allows the user to focus on a part of a given formula and then to rewrite this subformula by applying rewrite rules which are computed by the system from the context of the subformula under focus.
As an example, consider the situation, where a formula of the form

$$F \wedge (A_1 \vee \ldots \vee A_n \Rightarrow B) \Rightarrow G$$

is to be proven. In this situation it would be natural to focus on the subformula G and to show that G is true by using information encoded in

the antecedent of the formula (In fact, showing that G holds corresponds to rewriting it to *true* in the new system). The system supports this style of reasoning by suppling amongst others, the rewrite rules $B \rightarrow A_i$ to manipulate the subformula G. That is, any occurrence of B in G can be replaced by any of the A_i. Intuitively this expresses the fact that in order to show B, it is sufficient to show A_i, and then conclude B by making use of the fact that $(A_1 \vee \ldots \vee A_n \Rightarrow B)$ belongs to the context of G. In general, the rewrite rules generated by the system are of the form

$$[\Phi]\ h \rightarrow v$$

where Φ is a set of conditions under which application of the rule is valid. If a rule is applied, the conditions are introduced as new open goals. h is called the *head* of the rule. Any subformula in the current focus that matches this head can be replaced by the value v. A formula is a tautology if it can be rewritten to $\top$.

In this paper I will briefly report on an ongoing project which aims at extending the system by a suggestion mechanism that supports the user by making suggestions about the subformulas to be focussed on and suggests rewrite rules for application.

The remainder of the paper is organized into six short, informal sections. In the following section I will informally characterize the search space of the new system. Each of the then following sections describes one of the central problems that have to be addressed in the project. The final section describes the current state of the project.

2 Proof Search

The style of reasoning supported by the system requires the user to make one out of two kinds of choices at any proof state, i.e. the user can either

1. apply any of the available rewrite rules to transform the focus, or
2. narrow or widen the focus, or set the focus onto a completely different subformula.

However, none of the choices is trivial. First, there are usually many rewrite-rules available, not all of which are applicable because their heads do not match any term in the focus. Selecting the right inference rule might therefore sometimes become difficult. In real-world scenarios there are easily 30 rewrite rules available in each proof state. In these cases it is a non-trivial task to identify the applicable rules.

Second, focusing on the right subformula is crucial for structuring the search; e.g. if there is more than one disjunctive goal, choosing the write goal can be crucial for successful search. The system would therefore benefit from a suggestion mechanism that

1. is able to make suggestions about which subformulas to focus on, and
2. presents only the applicable rewrite rules, ordered according to a heuristic in such a way that the preferred rule can be suggested for application.

3 Tactics

As is possible for other systems, there should be a way to encode the definition of mathematical concepts or larger proof steps (induction, diagonalization, etc.) in form of a tactic in the system. Providing a way to use such tactics again causes two problems:

1. It has to be decided on how tactics and their expansion can be represented in the system.
2. Application of tactics should be supported by a suggestion mechanism in a similar way as was described for the rewrite rules.

4 Representation of Tactics

In many theorem proving systems tactics are represented as a set of premises and a conclusion (for instance in ΩMEGA). When reasoning backward, premises that cannot be instantiated with a proof line are introduced as open problems when the tactic is applied. Forward application of a tactic leads to an extension of the support-lines by the conclusion of the tactic.
It was decided to chose a similar representation of tactics for the new system. The reason for this is twofold:
First, specifying tactics in this way is very intuitive for the designer of a tactic. Second, if tactics are encoded in this way it is a straightforward task to adapt ΩMEGA s already existing suggestion mechanism Ω-ANTS to the new framework.
The tactic interpreter of the new system now allows the user to specify tactics in the manner described above. Informally, the interpreter converts a tactic of the form $\frac{P_1 \quad \ldots \quad P_n}{C}$ into a rewrite rule of the form $C \rightarrow P_{i_1} \wedge P_{i_j}$ where the P_{i_k} are those premises of the tactic for which no instantiation could be found. Application of a tactic then corresponds to application of the computed rewrite rule. (Compare this to ΩMEGA s natural deduction like calculus where the P_{i_k} are introduced as new goals).
However, application of a tactic yields the new open goal $(P_{i_1} \wedge P_{i_n} \Rightarrow C)$ which has to be closed through tactic expansion. Currently, we intend to expand tactics by simply handing the newly introduced subgoal along with some necessary definitions or axioms to an external theorem prover (such as OTTER, etc.). We hope that most problems that are introduced by tactic application are simple enough to be solved immediately by such a system. However, eventually a proper mechanism for tactic expansion will have to be provided, although this is not part of the project described here.

5 Supporting Proof Search

To support proof search it was decided to adapt the agent based suggestion mechanism Ω-ANTS [3, 4] to the new system. Ω-ANTS is an agent-based, concurrent suggestion mechanism that was developed to support interactive proof search in the ΩMEGA system.

Because the representation of tactics in the new systems is similar to the one used in ΩMEGA the adaption of Ω-ANTS to the new system is quite straightforward.
Currently, Ω-ANTS already supports backward application of tactics in the new system by searching for instances of the conclusion amongst the positive subformulas in the current focus. (This can later be extended to search through all negative subfomulas, which would correspond to considering all possible foci). Once an instance for a conclusion of the tactic is found, the system searches for instances of the parameters of the tactic in all positive subformulas that are α-related[1] to the instance of the conclusion.
More difficult is the task of extending Ω-ANTS to support the dynamically generated rewrite rules that change when the focus is shifted. A suggestion mechanism that supports reasoning with this rules should ideally be able to:

- Filter out those rules whose head matches a subformula in the focus, i.e. determine the set of applicable rules.
  For all applicable rules, try to find instances for the conditions of the rules. If such an instance is found, the corresponding condition no longer has to be introduced as a new open subgoal but can be closed immediately.
- Order the applicable rules according to a given heuristic.

Although these demands seem to be quite similar to those required to support application of tactics, they are not directly available in the Ω-ANTS system. The main reason for this is that Ω-ANTS allows to specify agents that search for instantiation of parameters for a given inference rule. However, because the dynamically available rules are only computed during the proof, no agents can be specified in advance.
To solve this problem it is intended to extend the system by an agent that sequentially checks for the applicability of each rewrite rule and -if a rule is applicable- also searches for instances of the condition of the respective rule.

6 Focusing

Focusing on the correct subformula of a goal is crucial for successful proof search. This can be seen as follows: a rewrite rule is applicable in a given focus if its head matches any of the subformulas in the focus. Hence, the more subformulas a focus contains, the more rewrite-rules are generally applicable in that situation. With more applicable rules at hand the task of selecting or suggesting the correct rule for application becomes more difficult. Moreover, in case of automated proof search, where each rules has to be applied systematically, the search space becomes larger if more rules are available. Furthermore, not every subformula needs to be focused on;i.e. when a goal consists of certain disjunctive subgoals it is

[1] In the new system formulas are annotated with a primary type α, β, γ or δ. Intuitively, formulas of type β are those formulas that cause a split in a sequent calculus proof. However, see [5] for details

sufficient to select one of the subgoals and rewrite it to true. Hence, focus placement is important to structure the proof search in an intelligent way. Consequently, a suggestion mechanism for the framework should be able to identify a minimal set of foci required to show the validity of a formula where each of the selected foci should contain as small formulas as possible.
To enable the system to make suggestions on where to place the focus it is planned to add a so called focus agent to the suggestion mechanism. This agent should analyze the current goal and identify a set of subformulas that have to be manipulated in order to prove the overall goal. By carefully selecting those subformulas the agent guides the user by suggesting subgoals which have to be proven in turn. Ideally, the agent is able to identify the crucial foci and to present them to the user one after another, so that the user only needs to concentrate on selecting the correct rewrite rules.

7 Current State of Implementation

Currently, the Ω-ANTS suggestion mechanism has been successfully adapted to support the use of tactics in the new framework. However, the suggestion mechanism cannot yet deal with the dynamically generated rewrite rules. Furthermore, a simple tactic interpreter has been implemented which allows one to execute tactics which can be specified in the manner described above.

References

1. Serge Autexier. A proof-planning framework with explicit abstractions based on indexed formulas. In Maria Paola Bonacina and Bernhard Gramlich, editors, *Electronic Notes in Theoretical Computer Science*, volume 58. Elsevier Science Publishers, 2001.
2. C. Benzmüller, L. Cheikhrouhou, D. Fehrer, A. Fiedler, X. Huang, M. Kerber, M. Kohlhase, K. Konrad, E. Melis, A. Meier, W. Schaarschmidt, J. Siekmann, and V. Sorge. ΩMega: Towards a Mathematical Assistant. In W. McCune, editor, *Proceedings of the 14th Conference on Automated Deduction (CADE-14)*, LNAI, Townsville, Australia, 1997. Springer Verlag, Berlin, Germany.
3. Christoph Benzmüller and Volker Sorge. Critical Agents Supporting Interactive Theorem Proving. In P. Barahona and J. J. Alferes, editors, *Progress in Artificial Intelligence, Proceedings of the 9th Portuguese Conference on Artificial Intelligence (EPIA-99)*, volume 1695 of *LNAI*, pages 208–221, Évora, Portugal, 21–24, September 1999. Springer Verlag, Berlin, Germany.
4. Christoph Benzmüller and Volker Sorge. Critical Agents Supporting Interactive Theorem Proving. Seki Report SR-99-02, Computer Science Department, Universität des Saarlandes, 1999.
5. Lincoln A. Wallen. *Automated Proof Search in Non-Classical Logics. Efficient Matrix Proof Methods for Modal and Intuitionistic Logics.* MIT Press, Cambridge, Massachusetts;London, England, 1990.

Automated Reasoning and Proof Planning

Baoqing Li

Department of Information Engineering
The Chinese University of Hong Kong
Shatin, New Territories, Hong Kong
TEL (+852) 2609-8367 FAX (+852) 2603-5032
E-Mail bqli1@ie.cuhk.edu.hk

ABSTRACT

A problem of reasoning about actions is given in terms of an initial situation, a terminal situation, a set of feasible actions, and a set of constraints. The task of a problem solver is to find the best sequence of permissible actions that can transform the initial situation into the terminal situation. Proof planning is an approach to theorem proving which uses proof methods rather than low-level logical inference rules to prove a theorem at hand. It is a powerful technique for guiding the search for proofs in automated reasoning for it often dramatically reduces the search space, allows reuse of proof methods, and moreover generates proofs where the reasoning strategies of proofs are transparent.

In general, realistic problems have enormous associated spaces of possible solutions that must be explored (searched) to find an actual solution that meets the requirements of the problem. These spaces are much too large to be searched in their entirety, and ways must be found to focus or short-circuit the search for solutions if systems are to have any practical utility. Put another way, almost every interesting problem class is computationally intractable (NP-complete or worse), meaning that in the worst case problems may take time exponential in the problem size to solve. Important questions include ways to reduce problem size and to focus search, and techniques for finding approximate solutions quickly.

Suppose you have left your keys somewhere in the house but cannot remember where, and need to find them quickly. You could do a systematic

search of the entire house, perhaps starting in the garage and exploring every nook and cranny to be absolutely sure the keys are not there before moving to other rooms. However, experience suggests that this ought to be a last resort and that a quick search following your instinct or "following your nose" is often likely to be successful more quickly. This suggests that when the search space is very large and a complete search will take too long, then it can be worthwhile to give up any hope of ever doing a complete systematic search and instead use an algorithm that concentrates on first exploring a good selection of the most likely places.

Many such algorithms are based on some form of "iterative repair." We propose a solution, look at its deficiencies and then try to repair one of these deficiencies, obtaining a new state that we hope is closer to an acceptable solution. This process is repeated many times until the state is acceptable, or we run out of patience. In order to save memory such algorithms cannot remember all the places that have already been searched. Hence, they often intentionally involve some degree of randomness: if decisions are made by tossing a coin then it is less likely that we will repeat exactly the same mistakes on each attempt to solve the problem.

Also, we can encode an optimized reasoning strategy (e.g. expert knowledge) in proof methods, which can be used in proof planning, by employing all kinds of different reasoning techniques in particular reasoning systems (e.g. computer algebra system).

On Complete Sequent Extensions of SLD-Resolution

Olexandr Lyaletsky
Faculty of Cybernetics, Kyiv National Taras Shevchenko University,
2, Glushkov av., build. 6, 03022 Kyiv, Ukraine, tel: +38 (044) 266 1218
E-mail: aal@tc.unicyb.kiev.ua

Introduction. The problem of construction of a goal-oriented technique for inference search based on the sequent formalism of first-order classical logic is solved. This technique gives an simple way for extension of logic-programming tools using SLD-resolution in the form of SLD-trees.

In this connection, we note that Gentzen calculi [1], including Kanger's calculus [2], significantly yield proof search efficiency to resolution-type methods (such that resolution-type methods, Maslov's inverse method, etc.), which rely upon results of Skolem [3] and Herbrand [4]. To overcome this obstacle, special goal-oriented sequent calculi are constructed here. Two peculiarities are inherent in the calculi: preliminary skolemization is used for increasing their proof search efficiency (and this permits to use a technique of finding most general unifiers) and selection of an appropriate rule is driven by a goal under consideration. Soundness and completeness of these calculi are proven.

As to SLD-resolution, we must note that results obtained give a simple way to transform sequent trees to trees, which can be considered as a general conclusion of the usual notion of SLD-trees. This gives a "key" for constructing complete extensions of SLD-resolution when logical consequence of arbitrary first-order formulas skolemized is under consideration.

Preliminaries. A sequent form of first-order classical logic without equality is considered. It is known [5] that deducibility of any sequent can be reduced to deducibility of a sequent with eliminated positive quantifiers and with bound variables only. Therefore, we can assume that any sequent consists of quantifier-free formulas, which in twos have no common variables.

Notions of terms, formulas, and literals are considered to be known. If L is a literal, then $\sim L$ denotes its complement. The *expression* $F^{\neg}$ denotes the

result of one-step carrying of the negation into a formula F and $\sharp$ denotes the empty formula. Also note that we understand *positive* ($P\lfloor F^+\rfloor$) *and negative* ($P\lfloor F^-\rfloor$) *occurrences* of a formula F in a formula P in the usual sense.

We treat the notion of (simultaneous) most general unifier as in [6].

An *equation* is a pair of terms s and t, which is written as $s \approx t$.

Let L be a literal of a form $R(t_1, \ldots, t_n)$ $(\neg R(t_1, \ldots, t_n))$ and M be a literal of a form $R(s_1, \ldots, s_n)$ $(\neg R(s_1, \ldots, s_n))$, where R is a predicate symbol and $t_1, \ldots, t_n, s_1, \ldots, s_n$ are terms. Then $\Sigma(L, M)$ denotes the set of equations $\{t_1 \approx s_1, \ldots, t_n \approx s_n\}$. In this case, L and M are said to be *equal modulo* $\Sigma(L, M)$($L \approx M$ *modulo* $\Sigma(L, M)$).

We understand sequents in the usual sense. Further we consider only *one-goal sequents*. Note that a sequent of the form $\Gamma \rightarrow \sharp$ is called an axiom.

A notion of trees uses in the usual sense. In what follows, so-called sequent trees are considered. A *sequent tree* is a tree with nodes labeled by sequents.

When a proof of an initial sequent S is searched, an *inference tree* Tr w.r.t. S is constructed. At the beginning, Tr contains only S. The subsequent nodes are generated when suitable rules are applied "from top to bottom".

A set $Eq(Tr)$ of equations is connected with every inference tree Tr. We suppose $Eq(Tr)$ is equal to $\emptyset$ for every initial tree Tr. For an inference tree Tr different from an initial tree, $Eq(Tr)$ is determined as $Eq(Tr') \cup \Sigma(L, M)$, where Tr' is such a tree that Tr is "inferred" from Tr' with some rule application and $\Sigma(L, M)$ is determined by this rule application.

An inference tree Tr is considered to be a *proof tree w.r.t.* S if and only if the following conditions are satisfied: every leaf of Tr is an axiom and there exists a simultaneous general unifier of all equations from $Eq(Tr)$.

A Goal-Oriented Sequent Calculus. A calculus GS described below is intended to establish that a formula G is the logical consequence of formulas $P_1, ..., P_n$. (In this case, $P_1, \ldots, P_n$, and G are called *initial formulas.*) Clearly, for defining GS it only remains to give its inference rules (cf. [7]).

Goal Splitting rules. These rules are used for elimination of the principal logical connective from the goal of a sequent under consideration. Any rule application results in generation of a new sequent (sequents) with a new goal (goals) and, possibly, with new premises. Note that for all Goal Splitting rules $\Sigma(L, M)$ is equal to $\emptyset$.

$$\frac{\Gamma \rightarrow F \supset F_1}{\Gamma, F \rightarrow F_1} \qquad \frac{\Gamma \rightarrow F \supset F_1}{\Gamma, F_1^{\neg} \rightarrow \neg F} \qquad \frac{\Gamma \rightarrow F \vee F_1}{\Gamma, F^{\neg} \rightarrow F_1}$$

$$\frac{\Gamma \rightarrow F \vee F_1}{\Gamma, F_1^{\neg} \rightarrow F} \qquad \frac{\Gamma \rightarrow F \wedge F_1}{\Gamma \rightarrow F \quad \Gamma \rightarrow F_1} \qquad \frac{\Gamma \rightarrow \neg F}{\Gamma \rightarrow F^{\neg}}$$

Premise Duplication Rule.

$$\frac{\Gamma_1, F\lfloor M^+\rfloor, \Gamma_2 \to L}{\Gamma_1, F', F\lfloor M^+\rfloor, \Gamma_2 \to L}$$

where $L \approx M$ *modulo* $\Sigma(L, M)$, and F' is a variant of a formula F.

Auxiliary Goals rules. As in [7], an order of applications of the rules of this group is "controlled" by a succedent literal L. Note that below M denotes a literal satisfying the following condition: $L \approx M$ *modulo* $\Sigma(L, M)$

$$\frac{\Gamma_1, F\lfloor M^-\rfloor \supset F_1, \Gamma_2 \to L}{\Gamma_1, (F\lfloor M^+\rfloor)^\neg, \Gamma_2 \to L \quad \Gamma_1, \Gamma_2 \to F_1^\neg} \qquad \frac{\Gamma_1, F \supset F_1\lfloor M^+\rfloor, \Gamma_2 \to L}{\Gamma_1, F_1\lfloor M^+\rfloor, \Gamma_2 \to L \quad \Gamma_1, \Gamma_2 \to F}$$

$$\frac{\Gamma_1, F \vee F_1\lfloor M^+\rfloor, \Gamma_2 \to L}{\Gamma_1, F_1\lfloor M^+\rfloor, \Gamma_2 \to L \quad \Gamma_1, \Gamma_2 \to \neg F} \qquad \frac{\Gamma_1, F\lfloor M^+\rfloor \vee F_1, \Gamma_2 \to L}{\Gamma_1, F\lfloor M^+\rfloor, \Gamma_2 \to L \quad \Gamma_1, \Gamma_2 \to \neg F_1}$$

$$\frac{\Gamma_1, F\lfloor M^+\rfloor \wedge F_1, \Gamma_2 \to L}{\Gamma_1, F\lfloor M^+\rfloor, F_1, \Gamma_2 \to L} \qquad \frac{\Gamma_1, F \wedge F_1\lfloor M^+\rfloor, \Gamma_2 \to L}{\Gamma_1, F_1\lfloor M^+\rfloor, F, \Gamma_2 \to L}$$

$$\frac{\Gamma_1, \neg(F\lfloor M^-\rfloor), \Gamma_2 \to L}{\Gamma_1, F^\neg\lfloor M^+\rfloor, \Gamma_2 \to L} \qquad \frac{\Gamma_1, M, \Gamma_2 \to L}{\Gamma_1, M, \Gamma_2 \to \sharp}$$

Contrary Closing rule($CC-rule$). Let Tr be a sequent tree and Br be its branch with a leaf Lf labeled a sequent $\Gamma \to L$, where Γ is a sequence of formulas, and L is a literal. Let Br contain a sequent $\Gamma' \to M$, where Γ' is a sequence of formulas, and M is a literal such that $\sim L \approx M$ *modulo* $\Sigma(\sim L, M)$. If Tr' is obtained from Tr by means of adding one successor labeled by $\Gamma \to \sharp$ to Lf, then Tr' is said to be inferred from Tr by CC-rule. The set $Eq(Tr')$ is determined as $Eq(Tr) \cup \Sigma(\sim L, M)$.

By defining all the rules of GS, we have the following result.

Proposition 1. Let formulas $P_1, \ldots, P_n$ form a consistent finite set of formulas. A formula G is the logical consequence of $P_1, \ldots, P_n$ if and only if there exists a proof tree w.r.t. the sequent $P_1, \ldots, P_n, \neg G \to G$ in GS.

Modifications of GS. A separate consideration requires the case, when we examine only sequents of the form $M_{1,1} \vee \ldots \vee M_{1,r_1}, \ldots, M_{n,1} \vee \ldots \vee M_{n,r_n} \to L_1 \wedge \ldots \wedge L_k$, where $M_{1,1}, \ldots, M_{n,r_n}, L_1, \ldots, L_k$ are literals. Because any first-order formula can be reduced to the conjunctive (disjunctive) normal form by means of logical-equivalence preserving transformations, it is easy to see that establishing deducibility of any sequent is equivalent to establishing

deducibility of a sequent of the form $M_{1,1} \vee ... \vee M_{1,r_1}, ..., M_{n,1} \vee ... \vee M_{n,r_n} \rightarrow L_1 \wedge ... \wedge L_k$. That is why we can investigate these sequents only. In this connection note that if G is $L_1 \wedge \ldots \wedge L_k$, then $G^{\neg}$ is $\sim L_1 \vee \ldots \vee \sim L_k$.

Taking the above into account, the calculus GS can be transformed into a calculus LS of literal sequents having the following rules.

Goal Splitting rule. The rule below generalizes $(\rightarrow \wedge)$-rule from GS in the case, when $\wedge$ is considered as a multiple-place operation.

$$\frac{\Gamma \rightarrow L_1 \wedge \ldots \wedge L_m}{\Gamma \rightarrow L_1, \ldots, \Gamma \rightarrow L_m}$$

Auxiliary Goals rule. This rule is applied when the goal of a sequent is a literal.

$$\frac{\Gamma_1, A_1 \vee \ldots \vee A_n \vee M \vee B_1 \vee \ldots \vee B_r, \Gamma_2 \rightarrow L}{\Gamma' \rightarrow\!\sim A'_1, \ldots, \Gamma' \rightarrow\!\sim A'_n, \Gamma' \rightarrow\!\sim B'_1, \ldots, \Gamma' \rightarrow\!\sim B'_r}$$

where $A_1, \ldots, \vee A_n, B_1, \ldots, B_r, L$, and M are literals, Γ' is the sequence $\Gamma_1, A_1 \vee \ldots \vee A_n \vee M \vee B_1 \vee \ldots \vee B_r, \Gamma_2$, $L \approx M$ *modulo* $\Sigma(L, M)$, and $A'_1, \ldots, A'_n$, $B'_1, \ldots, B'_r$ are new variants of $A_1, \ldots, A_n$, $B_1, \ldots, B_r$, respectively.

The calculus LS has the same $(\rightarrow \sharp)$-rule and CC-rule, as GS has.

Proposition 2. Let clauses $P_1, \ldots, P_n$ form a consistent finite set of clauses. A conjunction of literals G is the logical consequence of $P_1, \ldots, P_n$ if and only if there exists a proof tree w.r.t. the sequent $P_1, \ldots, P_n, G^{\neg} \rightarrow G$ in the calculus LS.

The peculiarity of LS is that antecedents of inferred sequents coincide with an antecedent of an initial sequent, say, Γ. This permits to consider Γ as a set of input clauses and to transform any inference tree Tr in LS into a tree $\gamma(Tr)$ having the same nodes, as Tr has, and labelling by only goals of corresponding sequents. Such a tree $\gamma(Tr)$ is said to be a *goal-tree* corresponding to Tr and we have an easy way to go from LS to SLD-resolution.

The completeness of SLD-resolution is a well-known result in Logic Programming (see, for example, [8, 9]). The proposition below contains it.

Corollary (Soundness and Completeness of SLD-resolution). Let positive Horn clauses $P_1, \ldots, P_n$ form a consistent finite set of clauses and G be a conjunction of atomic formulas. The goal G is the logical consequence of $P_1, \ldots, P_n$ if and only if there exists a proof tree w.r.t. the sequent $P_1, \ldots, P_n, G^{\neg} \rightarrow G$ in the calculus LS without any CC-rule application.

In conclusion, let us note that the corollary asserts that the calculi GS and LS can be considered as complete extensions of SLD-resolution.

References

[1] Gentzen, G.: Untersuchungen uber das Logische Schliessen. I, II, Math. Zeit. **39** (1934) 176–210, 405–443.

[2] Kanger, S.: Simplified proof method for elementary logic. In: Comp. Program. and Form. Sys.: Stud. in Logic. Amsterdam, North-Holland, Publ. Co. (1963) 87–93

[3] Skolem, T.: Logisch-kombinatorische Untersuchungen uber die Erfullbarkeit oder Beweisbarkeit mathematischer Satze. Skriftner utgit ar Videnskapsselskaper i Kristiania, **4** (1920) 4–36.

[4] Herbrand, J.: Recherches sur la Theorie de la Demonstration. Travaux de la Societe des Sciences et de Lettres de Varsovie, Class III, Sciences Mathematiques et Physiques, **33** (1930).

[5] Mints, G.: Herbrand Theorem (in Russian). In: Mathematical Theory of Logical Inference. Nauka, Moscow (1967) 311–350

[6] Robinson, J.: A Machine-Oriented Logic Based on Resolution Principle. JACM (1965) 23–41

[7] Degtyarev, A., Lyaletski, A., and Morokhovets, M. : Evidence Algorithm and Sequent Logical Inference Search. In: LNAI, **1705** (1999) 44–61.

[8] Apt, K.R., van Emden M. H. : Contributions into the Theory of Logic Programming. In: JASM, **3**, No 29 (1982) 841-862.

[9] Lloyd, J.V. Foundations of Logic Programming. Springer, Berlin (1987).

A Parallel Approach to Minimal Involuive Basis Algorithm

Vladimir A. Mityunin
Moscow State University
Department of Mechanics and Mathematics
Laboratory of Computing Methods
Vorobjovy Gory, Moscow, 119899, Russia
vmit@metric.ru

Alexandr S. Semenov
Moscow State University
Department of Mechanics and Mathematics
Laboratory of Computing Methods
Vorobjovy Gory, Moscow, 119899, Russia
semyonov@mccme.ru

ABSTRACT

The main result of [2] was the probabilistic algorithm to obtain the autoreduced Gröbner Basis of an algebraic ideal over the ring of integer numbers. This algorithm leaves a place for a possibility, that the basis obtained is not valid Gröbner one (with certain, usually very small probability). The important idea is that we perform computation of the autoreduced Gröbner Basis for the integer ideal I and for its projection I_p over the modular field Z_p simultaneously. If we see the zero reduction in modular case, we don't perform integer one either. We extended this idea for the Minimal Involutive Basis algorithm [6]. Using the method described in [2], it was shown, that this algorithm is also very sequential.

1. INTRODUCTION

In this work we present an attempt to parallelize the algorithm of computation of the minimal involutive bases. As Faugere has shown, the task of the Gröbner-like basis computation is very sequential and, therefore, every attempt to parallelize it essentially breaks the original algorithm. The main reason is that the result of the polynomial reduction often depends on the other polynomial, in particular, on the last reduced polynomial.

We present also a pseudo-probabilistic Faugère-type version of the computation method of the minimal involutive basis of the ideal. For example, we need to launch the Minimal Involutive Basis algorithm [6] on an integer system. Usually most of the time will be spent on involutive reducing the unnecessary polynomials (which have zeros as normal form). Instead of true integer computation we can perform its modular analog, and see which involutive prolongations are redundant. And later, having obtained the modular involutive reduction protocol, we can perform corresponding integer involutive reductions step by step. Certainly, this algorithm is not correct everywhere, but it gives the valid minimal involutive basis with very high probability. Upon completion of this part we have to check the involutive basis property. If we have failed we have to execute slow integer computation.

Surprisingly, for most systems we can obtain a correct minimal involutive basis using this method without slow integer computation and in many cases the check phase is very quick. We can also perform parallel checking using as much processors as we have.

2. PSEUDO-PROBABILISTIC RECOMPUTING APPROACH, SEQUENTIAL VERSION

In paper [2] a general probabilistic approach to the Buchberger-like algorithms is scetched. In our paper we use this approach to the Minimal Involutive Basis algorithm presented in [6]

In the following algorithm we will denote by NF_L the involutive normal form, by NM_L the set of non-multiplicative variables and by *Criterion* the involutive interpretation of the standard Buchberger criterion.

Criterion$(g, \mathbf{u}, T)$ is true provided that if there is $(f, v, D) \in T$ such that $lm(f)|_L lm(g)$ and $lcm(\mathbf{u}, v) \prec lm(g)$.

The Integer Probabilistic Minimal Involutive Basis Algorithm works as follows: we assume the sets F, G, T, Q to be the sets of the polynomials with the integer coefficients. By F_p, G_p, T_p, Q_p we will denote their projections to the sets of the polynomials over Z_p. The symbol g_p stands for the modular projection of an polynomial with integer coefficients g.

Procedure $GetModularProjection(g)$ projects the polynomial g with the integer coefficients to a polynomial with the coefficients over the ring Z_p and returns its projection g_p. If we have the set N of polynomials we will denote by $N_p = GetModularProjection(N)$ the set of projections of polynomials which are contained in N.

Procedure $CheckValidity()$ does the check phase of the algorithm. It checks, whether the initial monomials of the elements of integer and modular set coincide(first case), or the leading monomials of a polynomial and its modular projection are equal. If that equivalence doesn't hold, that means, that there's sufficient divergence between the integer and modular case, and there is no sense to continue the process. In that case $CheckValidity()$ stops the algorithm.

Algorithm **Integer Probabilistic Minimal Involutive Basis**
Input: F, a finite polynomial set
Output: G, probably, the minimal involutive basis of the ideal $Id(F)$
begin
  $F := Autoreduce(F)$
  $F_p := GetModularProjection(F)$
  $CheckValidity(F, F_p)$
  choose $g \in F$ with the lowest $lm(g)$ w.r.t. $\prec$
  $T := \{(g, lm(g), \emptyset)\}; Q := \emptyset; G := \{g\}$
  for each $f \in F \setminus \{g\}$ **do**
    $Q := Q \cup \{(f, lm(f), \emptyset)\}$
  $g_p := GetModularProjection(g); G_p := \{g_p\}$
  $CheckValidity(g, g_p)$
  repeat
    $h := 0$
    while $Q \neq \emptyset$ **and** $h = 0$ **do**
      choose g in $(g, u, P) \in Q$ with the lowest $lm(g)$ w.r.t. $\prec$
      $g_p = GetModularProjection(g)$
      $CheckValidity(g, g_p)$
      $Q := Q \setminus \{(g, u, P)\}$
      if $Criterion(g, u, T)$ is false **then** $h_p := NF_L(g_p, G_p)$
      if $h_p \neq 0$ **then** $h = NF_L(g, G)$
        $CheckValidity(h, h_p)$
    end
    if $h \neq 0$ **then** $G := G \cup \{h\}$; $G_p := G_p \cup \{h_p\}$
      if $lm(h) = lm(g)$ **then** $T := T \cup \{(h, u, P)\}$
      else $T := T \cup \{(h, lm(h), \emptyset)\}$
        for each f in $(f, v, D) \in T$ s.t. $lm(f) \succ lm(h)$ **do**
          $T := T \setminus \{(f, v, D)\}; Q := Q \cup \{(f, v, D)\}; G := G \setminus \{f\}$
          $f_p := GetModularProjection(f); G_p := G_p \setminus \{f_p\}$
    while exist $(g, u, P) \in T$ and $x \in NM_L(g, G) \setminus P$ and, if $Q \neq \emptyset$,
    s.t. $lm(gx) \prec lm(f)$ for all f in $(f, v, D) \in Q$ **do**
      choose such (g, u, P), x with the lowest $lm(g)x$ w.r.t. $\prec$
      $T := T \setminus \{(g, u, P)\} \bigcup \{(g, u, P \cup \{x\})\}$
      if $Criterion(gx, u, T)$ is false **then** $h_p := NF_L(g_p x, G_p)$
        if $h_p \neq 0$ **then** $h := NF_L(g, G)$
          $CheckValidity(h, h_p)$
          if $lm(h) = lm(gx)$ **then** $T := T \cup \{(h, u, \emptyset)\}$
          else $T := T \cup \{(h, lm(h), \emptyset)\}$
            for each f in $(f, v, D) \in T$ with $lm(f) \succ lm(h)$ **do**
              $T := T \setminus \{(f, v, D)\}; Q := Q \cup \{(f, v, D)\}; G := G \setminus \{f\}$
              $f_p := GetModularProjection(f); G_p := G_p \setminus \{f_p\}$
    end
  until $Q \neq \emptyset$
end

This algorithm seems to be efficient, when the integer coefficients of the polynomials proceeded are very large. The zero reductions are very common during the execution of the algorithm. Experience shows, that the coefficients of these reduced to zero polynomials in process of the reduction are often the largest during the entire algorithm. Avoiding of this work is the

main reason of the effectiveness of the algorithm presented.

Of course, we can give some examples, where such a procedure doesn't give the valid Minimal Involutive Basis. In that case we should perform the canonical algorithm.

After the execution of the probabilistic algorithm, we have to test, whether the system of polynomials obtained is the valid Minimal Involutive Basis. The procedure is simple. Let G be the obtained set. We make all non-multiplicative prolongations of the elements of G and reduce them with respect to G. If all prolongations are reducible to zero, the basis is obtained. Else we have to perform canonical algorithm.

3. PSEUDO-PROBABILISTIC RECOMPUTING APPROACH, PARALLEL VERSION

The detailed analysis shows that involutive reducing of the redundant prolongations (the prolongations whose normal form is zero) take a lot of running time. The only way to eliminate them correctly is to use criteria, described in [7]. But this is only a part of the task. Some zero prolongations remain. For integer computations, the cost of one involutive reduction is rather expensive, and the elimination of redundant prolongations is very important.

In integer computations, probabilistic approaches play a significant role. The word "probabilistic" means that, after termination of the algorithm, we are to check, whether the obtained polynomial system is the minimal involutive basis of the initial ideal. The theory requires that a Buchberger-like algorithm should be run again. (We generate prolongations and try to get the normal form of them.) But empirical results show that, if the initial system is an involutive basis of an ideal, a Buchberger-like algorithm quickly finishes the work.

Let I be an initial integer polynomial system and I_p its modular projection. The algorithm consists of three steps:

- Sequential calculation of the minimial involutive basis of I_p and recording the non-zero involutive reductions into the list RL.
- Parallel re-execution of RL-reductions on the set I. We finally obtain an integer system $RL(I)$.
- Parallel checking whether $RL(I)$ is an involutive basis.

We record the reductions in the following form. Assume that we have added $n-1$ polynomials to the basis. We are proceeding the polynomial number n. Initially, it can be formed as the polynomial prolongation of the polynomials k by the variable y or may be one of the initial generators. Then, it is reduced by polynomials number $n_1, \ldots, n_k$ to a non-zero involutive normal form. Its protocol is, by definition, $[Prolongation, k, y, n_1, \ldots n_k]$ in prolongation case, and $[FromInitialSet, n_1, \ldots n_k]$ otherwise, where $Prolongation$ and $FromInitialSet$ are the two opposite values of a Boolean indicator.

We do the second step as follows. Let us call a pair $\{polynomial, protocol\}$ a reduction map. A one-step transformation of a reduction map is one operation of the following forms:

$$\{0, [Prolongation, k, y, n_1, \ldots n_k]\} > \{Prolongation(k, y), [n_1, \ldots n_k]\} \tag{1}$$

$$\{0, [FromInitialSet, n_1, \ldots n_k]\} > \{NextFromInitialSet, [n_1, \ldots n_k]\} \tag{2}$$

$$\{p, [n_i, \ldots n_k]\} > \{invoReduce(p, n_i), [n_{i+1}, \ldots n_k]\} \tag{3}$$

The transformation 2 can be always performed. Operations 1 and 3 can be done only when the polynomials k or n_i have been yet computed. Otherwise, the map is called locked. A map $\{p, []\}$, where p is an arbitrary polynomial, is terminal.

We have one master and N slaves. Every slave keeps all the previously computed basis polynomials. At each step, the master reads protocols in RL and distributes the maps of the form

$\{0, [Prolongation, k, y, n_1, \ldots n_k]\}$ or $\{$polynomial from initial set$, [n_1, \ldots n_k]\}$

over the slaves. Each slave transforms (reduces) the maps, which are not terminal or locked, and then sends polynomials from terminal maps back to master. Sometimes, a slave interrupts its work to get all new computed basis polynomials from the master and the new maps. After that, some locked pairs become non-locked and master can continue its work with them.

When the algorithm ends its work, the master (and all slaves) will have the system which is likely to be the minimal involutive basis. But it not always so, and, hence, we need checking.

The checking is done in the following way. We create non-multiplicative prolongations of polynomials from $RL(I)$ and send them to the slaves. Since every slave has the whole basis list, it involutively reduces the received pairs itself and sends the error message if the result is not zero. One can see that with increasing of the slave number, the speed of testing also linearly increases.

Also we present results of the theoretical estimation of the maximum possible parallelization quality for number of test systems. This is done using Faugère's technique, for detailed description refer to [2].

Pseudo-Probabilistic Faugère's algorithm is a mighty tool for computing the involutive basis of integer polynomial systems. The most important fact for parallelization of this algorithm is the possibility to check basis using all available processors.

Our program complex supports both integer and modular field computations. It is implemented in Microsoft Visual C++ 6.0 and can be compiled on many platforms. Parallelization is supported by means of MPI 1.1 standard.

4. REFERENCES

[1] A.Giovanni, T.Mora, G. Niesi, L. Robbiano, C. Traverso, One sugar cube, please or Selection Strategies in Buchberger Algorithm, *Proceedings of ISSAC '91, ACM*, 1991, 49-54

[2] J.C. Faugère, Parallelization of Gröbner Basis, *Proceedings of PASCO'94*

[3] Kurt Siegl, A Parallel Factorization Tree Gröbner Basis Algorithm, *Proceedings of PASCO'94*

[4] Alyson A. Reeves, A Parallel Implementation of Buchberger's Algorithm over Zp for p $\leq$ 31991, *J.Symbolic Computation*, 1998, 229-244.

[5] R.Gebauer, H.M.Möller, On an Installation of Buchberger's Algorithm, *J. Symbolic Computation*, (1988) 6, 275-286.

[6] Gerdt V.P., Blinkov Yu. A. Minimal Involutive Bases, *Mathematics and Computers in Simulation*. - 1998. - 45.

[7] V.P.Gerdt, Yu.A.Blinkov, Denis A. Yanovich, Construction of Janet Bases, 2001

[8] D. A. Yanovich, Parallelization of an Algorithm for Computation of Involutive Janet Bases, *Programming and Computer Software, vol.28, No. 2, 2002*

[9] V.A.Mityunin, A.I.Zobnin, A.I.Ovchinnikov, A.S.Semenov, Involutive and Classical Gröbner Bases Construction from The Computational Viewpoint, *Proceedings of CAAP'2001*

[10] Mityunin V.A., Semenov A.S., Parallel Implementations of Honey Strategy Buchberger Algorithm, *Proceedings of "Workshop on Under- and Over-Determined Systems of Algebraic or Differential Equations" (Karlsruhe, Germany, 2002)*

Stern-Brocot binary representation of Rational and Real Numbers

Milad Niqui
University Of Nijmegen
Department of Computer Science
Postbus 9010, 6500 GL Nijmegen, The Netherlands
+31 24 36562610
milad@cs.kun.nl

In this poster we present the Stern-Brocot tree and a (binary) representation for rational and real numbers based on this tree. We show some arithmetical algorithms for this binary representation.

The Stern-Brocot tree is a binary tree containing all positive rational numbers with the important property that each rational number occurs exactly once. For $\frac{p_1}{q_1}$ and $\frac{p_2}{q_2}$ we define the *mediant* of them to be the fraction $\frac{p_1+p_2}{q_1+q_2}$. In this tree every row consists of the fractions that are mediants of elements of previous rows. We start to write the two pseudo-fractions $\frac{0}{1}$ and $\frac{1}{0}$, we proceed to construct the tree row by row. The first row is the mediant of the two initial pseudo-fractions, that is $\frac{0+1}{1+0} = \frac{1}{1}$. We write this mediant in the middle of the initial pseudo-fractions. The second row consists of the mediant of $(\frac{0}{1}, \frac{1}{1})$ and the mediant of $(\frac{1}{1}, \frac{1}{0})$. We continue in this way and each time we choose two neighbouring fractions (i.e. two fractions which when moving from left to right there is no other fraction between them); and we place their mediant in a new row and in the middle of the two parents. This construction is illustrated in Figure I. The resulting tree has many interesting properties. We present some of the basic properties in this poster. For more properties including some combinatorial properties see [4, 1].

Lemma 1 *All the fractions occurring in the Stern-Brocot tree are irreducible.*

Lemma 2 *Every positive rational number occurs exactly once in the Stern-Brocot tree.*

It has already been observed [4] that this property can be used to give a binary encoding of positive rational numbers. We will formally introduce this representation as a basis for inductively defined rational numbers and present the algorithms for computing the arithmetical expressions on them. This representation has the benefit that it is easily formalisable in a theorem prover using inductive types. Thus we define the set **USB** of *unsigned Stern-Brocot rational numbers* as:

$$\mathbf{USB} = \mathbf{L}\ \mathbf{USB} \mid \mathbf{R}\ \mathbf{USB} \mid \mathbf{I}$$

Therefore we represent any positive rational number by a finite sequence of L's and R's which ends in I. Subsequently following the methods for computing with continued fractions [3, 7, 5, 6] we will show the algorithms for computing *homographic functions*:

$$\begin{array}{l} H_A : \mathbb{Q}^+ \longmapsto \mathbb{Q}^+ \\ H_A(x) = \frac{ax+b}{cx+d} \qquad a,b,c,d \in \mathbb{Z} \quad A = \left[\begin{smallmatrix} a & b \\ c & d \end{smallmatrix}\right] \end{array}$$

and *quadratic functions*:

$$\begin{array}{l} Q_A : \mathbb{R}^+ \longmapsto \mathbb{R}^+ \\ Q_A(x,y) = \frac{axy+bx+cy+d}{exy+fx+gy+h} \quad a,b,c,d,e,f,g,h \in \mathbb{Z} \\ A = \left[\begin{smallmatrix} a & b & c & d \\ e & f & g & h \end{smallmatrix}\right] \end{array}$$

for this representation. The latter will give us the algorithms for computing the field operations.

We will show that this inductive representation can be generalised to a *coinductive* representation. That is to say we define the set of *unsigned Stern-Brocot real numbers*:

$$\texttt{CoInductive}\ \ \overline{\mathbf{USB}} = \mathbf{L} : \overline{\mathbf{USB}} \longrightarrow \overline{\mathbf{USB}} \mid \mathbf{R} : \overline{\mathbf{USB}} \longrightarrow \overline{\mathbf{USB}} \mid \mathbf{I} : \overline{\mathbf{USB}}$$

As defined here, a real number is a possibly infinite binary stream of L's and R's (and if it is finite it ends in I). As it is the case with all efficient representation of real numbers, some redundancy will arise here. In the poster we will briefly discuss this redundancy.

Similarly the algorithms that we defined for computation on the rational Stern-Brocot representation (the homographic and quadratic algorithms), can be generalised to obtain lazy algorithms for the case of real numbers.

Furthermore, we shall give some generalisations of quadratic algorithm, to calculate more complex arithmetical expressions, such as polynomials and rational functions. We will also mention how one can use the methods given in [6] to evaluate the power series of transcendental functions.

The formalisation of algorithms in the Coq theorem prover [2], is in progress. We conclude the presentation by showing some of the difficulties peculiar to this formalisation.

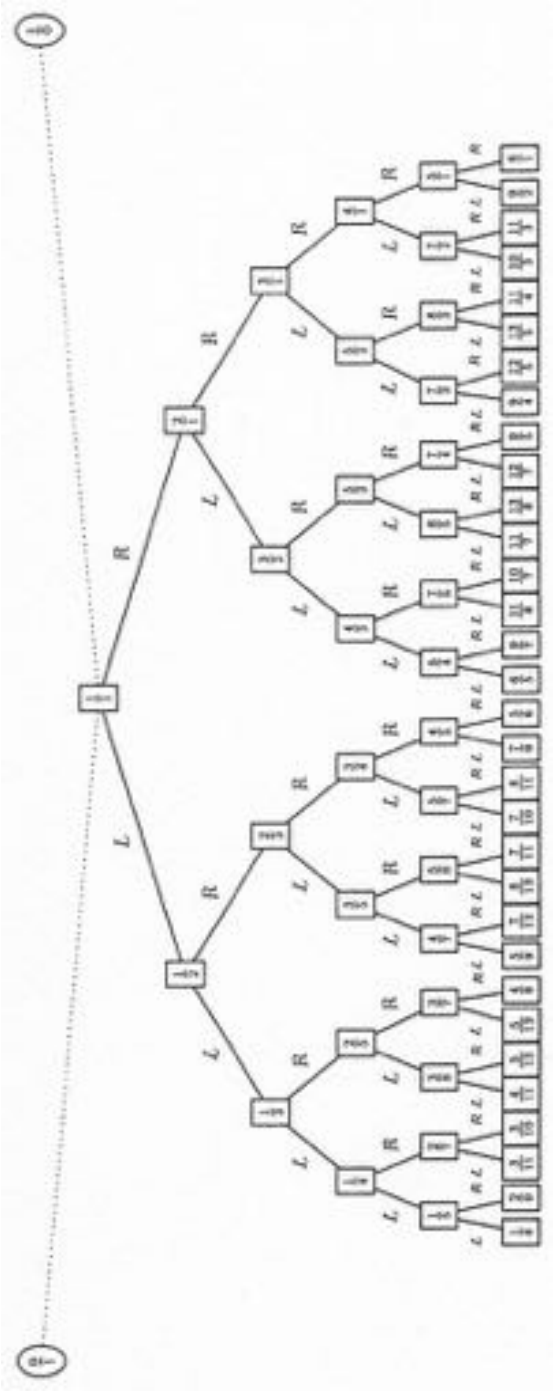


Figure I: Stern-Brocot tree

References

[1] Bruce P. Bates. *Self-Matching and Interleaving in Some Integer Sequences and the Gauss Map.* PhD thesis, University of Wollogong, 2001.

[2] Bruno Barras et al. *The Coq Proof Assistant Reference Manual, Version 7.0.* INRIA, `http://pauillac.inria.fr/coq/doc/main.html`, apr 2001.

[3] Ralph W. Gosper. HAKMEM, Item 101 B. `http://www.inwap.com/pdp10/hbaker/hakmem/cf.html#item101b`, feb. 1972. MIT AI Laboratory Memo No.239.

[4] Ronald E. Graham, Donald E. Knuth, and Oren Patashnik. *Concrete Mathematics.* Addison-Wesley, Reading, Massachusetts, second edition, 1994.

[5] Valérie Ménissier-Morain. *Arithmétique exacte, conception, algorithmique et performances d'une implémentation informatique en précision arbitraire.* Thèse, Université Paris 7, dec 1994.

[6] Peter J. Potts. *Exact Real Arithmetic using Möbius Transformations.* PhD thesis, Uiversity of London, Imperial College, jul 1998.

[7] Jean E. Vuillemin. Exact real computer arithmetic with continued fractions. *IEEE Transactions on Computers*, 39(8):1087–1105, aug 1990.

Prototype User Interface for an Interactive Theorem Prover

I.Normann

August 19, 2002

Fachbereich Informatik, Universität des Saarlandes
D-66041 Saarbrücken, Germany
normann@ags.uni-sb.de

Abstract

We present a prototype of a browser based GUI for the interactive reasoning system CORE. Very sophisticated reasoning and proof planning systems such as CORE and ΩMEGA exist for experts. The purpose of our research is to make these systems usable by the non-experts. We consider this a prerequisite to establish automated reasoning and proof planning as an advanced assistant tool for maths education and human mathematical reasoning. On the proof of "$\sqrt{2}$ is irrational" we provide a demo of a prototypical browser based GUI of CORE.

Introduction

A calculator is such an easy to use helper for basic mathematical operation as a simple text editor for writing words. To calculate log(8) (up to a certain precision) only three actions are necessary and within a millisecond we see the result on the display. One reason why we use the calculator is obviously its speed, but another reason is that we would not know how to calculate it by hand. Either because we have never learned it or we cannot remember. If we calculated log once in school by hand then probably just to get a better "feeling" of the function log in general rather then for the result itself. As long as we can interprete the meaning of log(8) and its computed result we actually do not need to know how to calculate it. So we can dispatch the job of the mechanical calculation to the calculator which does it correctly.

The success of a calculator is (besides its computational speed) due to its user interface: It is very straightforward to type in what you want to be calculated. So we learn very early in school how to use such a tool.

Success of Computer Algebra Systems in Maths Education

For maths education the use of a calculator is of course very limited, mainly as service for pure calculation. With the rise of computer algebra systems more powerful tools are available. Many of them are capable of symbolic computation and they provide their own programming language. But the more features a tool offers the more difficult it is to make the whole functionality transparent to the user.

Nowadays there are CAS with a very complex functionality (such as Maple, Mathematica, MuPAD, etc.). In spite of their complexity they seem to be user friendly enough to be employed in maths education.

We can dispatch such tasks as, e.g., integration of functions, simplification of fractions or equations etc. to such a CAS (with some restrictions to correctness). Even though there is no longer a keyboard that provides a key for each operation (like e.g. a "log"-key on calculator) the instruction to the CAS is almost as simple as the "log"-key. One types just such more or less self explaining commands like "Integrate[f(x), {x, a, b}]".

Interactive and Automated Theorem Proving

CAS are even able to do proofs if we consider a proof a mapping from a statement to a truth value (a trivial example: Mathematica maps "$0 < a - a$" to "false"). But a CAS does not supply a framework

for interactive or automated theorem proving. The user does not have access to any kind of logical calculus within the system not to speak of proof planning.

Reasoning systems such as CORE and ΩMEGA [1, 2] are built on logical calculi and they provide means to do logical justified proof steps. They even provide methods (ΩMEGA now and CORE in future) which abstract from the calculus-level and allow proof steps at a level that is closer to human mathematical reasoning.

As such systems already exist, why dont we use them in education or as proof assistant? The answer is: They are far from being user friendly! It is still a challenge how to model formally human like mathematical reasoning in methods, control knowledge, strategies etc. (which is not part of this work). Moreover it is the missing intuitive interface to the user (the primal goal of this work).

Difficulties of Building an UI for a Theorem Prover

If we look inside a mathematical text book, we find that all those operations that could be done by a CAS - like integration of functions, simplification of fractions or equations etc. - have a (quasi) formal presentation. For instance, the equation "$\int \sin(x)dx = \cos$" can be read as an procedure in a CAS: "The operator Integrate applied on sin yields cos". Or "$(a^2 - b^2)/(a + b) = a - b$" as "The operator Simplify applied on $(a^2 - b^2)/(a + b)$ yields $a - b$".

In contrast we do not find formal presentation of logical operations that could be done by a theorem prover. For instance we never find (quasi) formal presentation of the application of modus ponens, or the forall introduction in mathematical text book. Actually, mathematicians rarely use a logical calculus for mathematical reasoning. Russell and Whitehead [4] showed in their Principia Mathematica how tedious simple proofs become if they obey to a formal calculus. Therefore, human mathematical reasoning has still a rather informal look.

Unfortunately, a theorem prover does only understand formal input. Even if we try to model human like mathematical reasoning by an interactive proof planning system it is still a system which needs a formal input. But there is no corresponding formal presentation of tactics, methods, strategies, etc. in mathematical text books. In order to tell the system to apply one of the proof planning operations we first have to learn its meaning. This includes the understanding of its formal presentation. But there is no cultural evolved (quasi) formal and widely understood presentation of proof planning objects which could be the basis of the UI.

So one difficulty of the design of a UI for a theorem prover is that the presentation of operations should suggest their meaning much more than in CAS where the UI is rather concerned with presentation of already understood operations.

UI for Core on the calculus level

Since the definition of the operations and components which constitutes proof planning in the new system CORE is still in progress, it is to soon to design a UI for proof planning. But one can already prove very interesting theorems interactively in CORE without the help of proof planning just using its elegant calculus.

Actually, this calculus itself is a great improvement towards a rather human oriented logical reasoning. Also ΩMEGA did already a step in this direction as its underlying calculus is natural deduction rather then a machine-oriented calculus like resolution of the traditional automated theorem provers. But also natural deduction appears too low-level to be practicable even for simple proofs.

In a real mathematical proof scenario the formal presentation of the theorem to be proved is a more or less complex logical formula. From formal perspective the mathematician typically wants to focus on some subformula in order to refine it to subgoals. Using natural deduction to get access to the subformula under focus someone is concerned with a lot of technical inference steps like e.g. quantifier elimination.

In CORE's underlying calculus you have a straight access to subexpressions. This means that you set your focus there and the system provides you the complete collection of replacement rules which are allowed to apply on the focus (within a given theory). Of course only those are applicable which match the subexpression. The application of replacement rules yield new subgoals.

Even a user with almost no formal logic background who has no glue how the system finds the replacement rules to a given focus has immediately an idea what to do with replacement rules namely replacing expressions. So as soon as the user knows which subexpression of a given complex formula he wants to manipulate CORE does a valuable service.

Though the handling of CORE is very easy it is absolutely not user friendly in practice although it has a GUI namely $\mathcal{LOUI}$.

Features of our New Prototype UI

To reveal the power of CORE to a non developer we must be aware that he gets soon impatient if basic interactions with the system are awkward to handle. This is the case in $\mathcal{LOUI}$ so far: Putting a focus on subformula forces the user to type in the corresponding position as list of numbers.

The much more intuitive interaction for this goal is just pointing with the mouse to the corresponding subformula. Before we switch to the new focus we want to see the selected focus highlighted in order to be sure that the system selects the intended subformula. Alternatively to mouse selection it is comfortable to navigate through a formula tree with arrow keys. The selection of replacement rules should work in a similar manner.

Before we can do anything with CORE, the problem (i.e. the theorem + the theory) must be passed to the system. Up to now this is done via a specification file written in the specification language CASL (in future OpenMath will be supported too) [3]. Existing problems must be loadable, of course. In addition the user may modify or extend a problem or even write his own one. So we need an input editor. An input editor is a project on its own, but we try to provide a restricted input editor for demonstration purpose. We leave the input editors list of features open.

During the proof the user may recall some previous steps. So some means to browse through proof history and the proof plan must be provided. Also, a graphical tree that represents the proof plan does a good service to survey the proof procedure. A click on node yields the concerning proof status of the history.

Of course we want a state of the art rendering of mathematical expressions. We abandon completely verbalization (as verbalization is a research field on its own). Therefore, the user must be capable to understand logic syntax.

Implementation of the Prototype

We try to provide most of the features mentioned above in our prototype. The basic requirement is the facility of interaction on state of the art rendered mathematical expression. In modern browser (e.g. Mozilla, Internet Explorer) we get this for free. Therefore, we decided to implement a browser UI. Interaction events can be sufficiently controlled by JavaScript. This script language is also predestinated for fast prototyping.

The purpose of the prototype is also to find the basic system architecture of the final UI. In principle we follow the Model View Control design pattern. Roughly spoken the control handles the interaction events, the view renders the presentational layout and the model manages the logical functionality of the interactive objects and communicates with CORE. View and control lie on client side and the model on server side. We take a Lisp server which enables an easy communication with CORE which is also written in Lisp.

Conclusion and Outlook

We restricted our prototype GUI for CORE to an interactive proof frontend on CORE's calculus level. In a demo proof of "$\sqrt{2}$ is irrational", we hope to make CORE's elegant calculus visible. In this case it shows that real world proofs can be carried out in an almost mathematical natural fashion at a calculus level.

For more complex theorem proving a more abstract framework as developed in the proof planning research is needed. As soon as this framework is established for CORE we go ahead prototyping a GUI that supports it too.

The long term goal is an interactive maths environment for theorem proving and proof presentation that can be connected to a reasoning system like CORE and to a mathematical tutor system that supports meta reasoning in a Polya style. But it should be also possible to use it stand alone as tool that supports the user structuring his proof ideas just like with the pencil and paper but with an intelligent support of elaborated interactive features.

The demo prototype is a component within this large vision.

References

[1] Serge Autexier. A proof-planning framework with explicit abstractions based on indexed formulas. In *Proceedings of the 4^{th} Workshop on Strategies in Automated Deduction (STRATEGIES'01 pages = 87-99, year = 2001, editor = Bonacina, Maria-Paola and Gramlich, Bernhard, volume = TR DII 10/01, month = june, publisher = Universitá degli studi di Siena,.*

[2] C. Benzmüller, L. Cheikhrouhou, D. Fehrer, A. Fiedler, X. Huang, M. Kerber, M. Kohlhase, K. Konrad, E. Melis, A. Meier, W. Schaarschmidt, J. Siekmann, and V. Sorge. ΩMega: Towards a Mathematical Assistant. In W. McCune, editor, *Proceedings of the 14th Conference on Automated Deduction (CADE-14)*, LNAI, Townsville, Australia, 1997. Springer Verlag, Berlin, Germany.

[3] OpenMath. http://www.openmath.org/.

[4] Alfred N. Whitehead and Bertrand Russell. *Principia Mathematica*, volume I. Cambridge University Press, Cambridge, Great Britain; 2nd edition, 1910.

On bigenerated subalgebras of the univariate polynomial ring

Hans Öfverbeck
telephone: +46 46 222 34 08
email: hans@maths.lth.se

Coauthors:

Anna Torstensson
telephone: +46 46 222 85 68
email: annat@maths.lth.se

Victor Ufnarovski
telephone: +46 46 222 41 46
email: ufn@maths.lth.se

All authors are affiliated to:
Lund University
Centre for Mathematical Sciences
Box 118, SE-221 00 Lund
Sweden

Take a look at the following problem:

Let K be a field and let $K[x]$ be the commutative polynomial ring in one variable over K. Let $K[f,g]$ be the subalgebra generated by the nonconstant polynomials f and g, what are the necessary and sufficient conditions for $K[f,g] = K[x]$?

In our research group we are inspired by this problem and study other problems which are related to subalgebras generated by two polynomials in the univariate polynomial ring. Even though most of the problems are fairly easy to formulate, the proofs are often far from straightforward. Most results rely on the characteristic of the field, as for instance the theorem, which was first proved in [1]:

Theorem 1 *Let $f, g \in K[x] \setminus 0$. Assume that the characteristic of K does not divide* $\gcd(\deg(f), \deg(g))$. *If $K[f,g] = K[x]$ then either* $\deg(f)$ *divides* $\deg(g)$ *or* $\deg(g)$ *divides* $\deg(f)$.

Since most results have similar restrictions on the characteristic of the field, or are even restricted to characteristic zero, it is natural to ask whether these limitations are imposed by the problem or if they are a result of the method of proof. We have tried to find a characteristic independent approach to

these problems and have managed to prove some results which are indeed independent of the characteristic.

SAGBI bases, as introduced in [3] and independently in [2], have many natural connections to these problems. This is illustrated by the appearance of the SAGBI concept in the theorems below, which are a selection of the results we have managed to prove:

Theorem 2 *$\{f, g\}$ is a SAGBI-basis if* $\gcd(\deg(f), \deg(g)) = 1$.

Theorem 3 *$\{f, g\}$ is a SAGBI-basis if and only if* $[K(x) \quad K(f, g)] = \gcd(\deg(f), \deg(g))$.

Theorem 4 *Let $n = \deg(f)$, $m = \deg(g)$, $d = \gcd(n, m)$, $n' = n/d$ and $m' = m/d$ If h is the S-reduced remainder of the S-polynomial $f^{m'} - g^{m'}$ and $\gcd(d, deg(h)) = 1$ then $\{f, g, h\}$ is a SAGBI-basis.*

Note that a consequence of Theorem 2 and Theorem 3 is that if $\gcd(\deg(f), \deg(g)) = 1$ then $K(x) = K(f, g)$, this however does not imply that $K[x] = K[f, g]$ as can be seen from the example $f = x^2$, $g = x^3$.

References

[1] Shreeram S. Abhyankar and Tzuong Tsieng Moh. Embeddings of the line in the plane. *J. Reine Angew. Math.*, 276:148–166, 1975.

[2] Deepak Kapur and Klaus Madlener. A completion procedure for computing a canonical basis for a k-subalgebra. In *Computers and mathematics (Cambridge, MA, 1989)*, pages 1–11. Springer, New York, 1989.

[3] Lorenzo Robbiano and Moss Sweedler. Subalgebra bases. In *Commutative algebra (Salvador, 1988)*, volume 1430 of *Lecture Notes in Math.*, pages 61–87. Springer, Berlin, 1990.

Reasoning Inside a Formula

Andrey Paskevich
Faculty of Cybernetics, Kyiv National Taras Shevchenko University,
2, Glushkov avenue, building 6, 03022 Kyiv, Ukraine
Tel.: +38 (044) 266 3108, E-mail: andrey@raptor.kiev.ua

When we deal with real mathematical problems in the automated reasoning, we rarely meet "absolute", unconstrained rules, definitions, statements. Usually, everything we use is supplied with certain conditions (e.g. type constraints), so that we have to eliminate them first. For example, we can't reduce the fraction $\frac{xy}{x}$ until we prove that x is non-zero.

Consider a big formula of the kind $(\cdots \forall x\,(x \in \mathbb{R}^+ \supset (\cdots \frac{xy}{x} \cdots)) \cdots)$. It's evident that we can replace $\frac{xy}{x}$ with y, but how to prove it? The task itself seems to be absurdive: as soon as a term depends on bound variables, we can't reason about it. In the traditional fashion, we should first split our big formula up to the quantifier bounding x, make a substitution (or skolemization), separate $x \in \mathbb{R}^+$, and only then make the simplification.

One can argue that there is no gain in any simplifications while a formula to be simplified lies deep inside, that we would split our big formula anyway to use that fraction in a proof. However, we believe that it's useful and instructive to simplify our problem *in it's initial form* as far as possible in order to select the most perspective way of the proof search.

Besides constrained simplification, there is a question of well-definedness of a given formula. Every occurrence of a defined symbol must conform the definition guards to be consistent. And it's quite important to check that our problem is correctly formulated before investigating it seriously. But such a verification is connected as well with the reasoning about terms containing bound variables (regard the example above).

The same problem exists on the propositional level. Suppose that our theory contains an axiom $A \supset B$, where A and B are fixed propositional letters, and consider a formula of the kind $(\cdots (A \wedge (\cdots B \cdots)) \cdots)$. Again, it would be reasonable to replace B with $\top$ (truth) and then simplify the whole formula. Again, we have no means to do it directly.

Let us return to our first example. While the statement "x is non-zero" is

surely meaningless, we can say "x is non-zero in this occurrence of $\frac{xy}{x}$". Our intuition prompts that side by side with the usual truth value, there exists a so called *local truth value* defined with respect to a context, to some position in a formula. A statement that is generally false or even meaningless can become locally true being considered in that position.

In the paper we develop the notion of a locally true statement. and show that we can consistently reason about interiors of a formula. For the sake of brevity, we give our statements without proofs.

Preliminaries. We consider the first-order language with the equality $\approx$, the logical connectives $\neg$, $\wedge$, $\vee$, $\supset$, and $\equiv$, and the quantifiers $\forall$ and $\exists$. Two formulas F and G are said to be *equivalent* (written $F \simeq G$) if their logical equivalence is valid in the first-order logic, i.e. $\models F \equiv G$. We denote the universal closure of a formula U by $\forall U$.

A *position* is a word in $\{0,1\}^*$. The set of positions of a formula F, denoted by $\Pi(F)$, is defined as follows (below, $i.\Pi$ denotes $\{\, i.\pi \mid \pi \in \Pi \,\}$):

$$\begin{aligned}
\Pi(A) &= \{\varepsilon\} && \text{where } A \text{ is an atom}\\
\Pi(* F) &= \{\varepsilon\} \cup 0.\Pi(F) && \text{where } * \in \{\forall x, \exists x, \neg\}\\
\Pi(F * G) &= \{\varepsilon\} \cup 0.\Pi(F) \cup 1.\Pi(G) && \text{where } * \in \{\wedge, \vee, \supset, \equiv\}
\end{aligned}$$

An *occurrence* is a pair (F, π), where F is a formula and $\pi \in \Pi(F)$. For every such pair, we denote by $F|_\pi$ the subformula of F at position π: $F|_\varepsilon = F$, $(* F)|_{0.\pi} = F|_\pi$, $(F * G)|_{0.\pi} = F|_\pi$, $(F * G)|_{1.\pi} = G|_\pi$.

Given an occurrence (F, π) and a formula U, the formula $F[U/\pi]$ is defined as follows (free variables of U are allowed to be bound in $F[U/\pi]$):

$$\begin{aligned}
F[U/\varepsilon] &= U & (* F)[U/0.\pi] &= (* F[U/\pi])\\
(F * G)[U/0.\pi] &= (F[U/\pi] * G) & (F * G)[U/1.\pi] &= (F * G[U/\pi])
\end{aligned}$$

Local images. Given an occurrence (F, π) and a formula U, we define the (F, π)-*image* of U (denoted $\langle U \rangle_\pi^F$) by the following identities:

$$\begin{array}{lll}
\langle U \rangle_\varepsilon^F = U & \langle U \rangle_{0.\pi}^{F \wedge G} = G \supset \langle U \rangle_\pi^F & \langle U \rangle_{1.\pi}^{F \wedge G} = F \supset \langle U \rangle_\pi^G\\
\langle U \rangle_{0.\pi}^{\neg F} = \langle U \rangle_\pi^F & \langle U \rangle_{0.\pi}^{F \vee G} = G \vee \langle U \rangle_\pi^F & \langle U \rangle_{1.\pi}^{F \vee G} = F \vee \langle U \rangle_\pi^G\\
\langle U \rangle_{0.\pi}^{\forall x F} = \forall x \langle U \rangle_\pi^F & \langle U \rangle_{0.\pi}^{F \supset G} = G \vee \langle U \rangle_\pi^F & \langle U \rangle_{1.\pi}^{F \supset G} = F \supset \langle U \rangle_\pi^G\\
\langle U \rangle_{0.\pi}^{\exists x F} = \forall x \langle U \rangle_\pi^F & \langle U \rangle_{0.\pi}^{F \equiv G} = \langle U \rangle_\pi^F & \langle U \rangle_{1.\pi}^{F \equiv G} = \langle U \rangle_\pi^G
\end{array}$$

The formula $\langle U \rangle_\pi^F$ says "*U is true at position π in F*" Note that it doesn't depend on the subformula $F|_\pi$.

Example 1 *Let F be the formula*

$$\forall x\,(x \in \mathbb{N} \supset \forall n\,(n \in \mathbb{N} \supset (x \approx \mathbf{fib}(n) \equiv \\ \equiv ((n \leqslant 1 \wedge x \approx 1) \vee x \approx (\mathbf{fib}(n-1) + \mathbf{fib}(n-2))))))$$

This formula represents a recursive definition, i.e. it contains occurrences of the symbol being defined to the right of the "main" equivalence sign. Indeed, for $\pi = 0.1.0.1.1.1$, $F|_\pi = (x \approx (\mathbf{fib}(n-1) + \mathbf{fib}(n-2)))$. We want to verify that the arguments $(n-1)$ and $(n-2)$ satisfy guards of the definition and are strictly less than n (note that the ordering must be well-founded).

Consider the second term. We should prove $\langle (n-2) \in \mathbb{N} \wedge (n-2) < n \rangle^F_\pi$. The latter formula is equal to

$$\forall x\,(x \in \mathbb{N} \supset \forall n\,(n \in \mathbb{N} \supset ((n \leqslant 1 \wedge x \approx 1) \vee ((n-2) \in \mathbb{N} \wedge (n-2) < n))))$$

But this formula is false given $n = x = 0$. And this points out a mistake in our definition: $x = 0$ makes false the left side of the disjunction $F|_{0.1.0.1.1}$, so we have to consider the right side with $n = 0$ in order to evaluate the truth value of the whole disjunction. Now it's easy to build a good definition F':

$$\forall x\,(x \in \mathbb{N} \supset \forall n\,(n \in \mathbb{N} \supset (x \approx \mathbf{fib}(n) \equiv \\ \equiv ((n \leqslant 1 \wedge x \approx 1) \vee (n \geqslant 2 \wedge x \approx (\mathbf{fib}(n-1) + \mathbf{fib}(n-2)))))))$$

Let us sketch some important properties of local images.

Lemma 2 *For any occurrence (F, π) and a formula U, $\forall U \models \langle U \rangle^F_\pi$.*

Lemma 3 *For any occurrence (F, π) and formulas U and V,*

$$\models \langle U \equiv V \rangle^F_\pi \supset (\langle U \rangle^F_\pi \equiv \langle V \rangle^F_\pi) \qquad \models \langle U \wedge V \rangle^F_\pi \equiv (\langle U \rangle^F_\pi \wedge \langle V \rangle^F_\pi)$$

Corollary 4 $\models \langle U \supset V \rangle^F_\pi \supset (\langle U \rangle^F_\pi \supset \langle V \rangle^F_\pi)$

Corollary 5 *Given a quantifier-free context C,*

$$\models (\langle U_1 \equiv V_1 \rangle^F_\pi \wedge \cdots \wedge \langle U_n \equiv V_n \rangle^F_\pi \wedge \langle t_1 \approx s_1 \rangle^F_\pi \wedge \cdots \wedge \langle t_m \approx s_m \rangle^F_\pi) \supset \\ \supset \langle C[U_1, \ldots, U_n, t_1, \ldots, t_m] \equiv C[V_1, \ldots, V_n, s_1, \ldots, s_m] \rangle^F_\pi$$

These lemmas show that we really can reason about locally true statements: combine them in conjunctions, use the substitutivity of equivalence, apply *modus ponens*. However, the key property of local images is given by the following theorem.

Theorem 6 *For any occurrence (F, π) and formulas U and V*

$$\models \langle U \equiv V \rangle^F_\pi \supset (F[U/\pi] \equiv F[V/\pi])$$

Corollary 7 *Given an occurrence (F, π) and a formula U,*

$$\models \langle U \rangle^F_\pi \supset (F \equiv F[\pi/(U \wedge F|_\pi)]) \qquad \models \langle U \rangle^F_\pi \supset (F \equiv F[\pi/(U \supset F|_\pi)])$$

So we can safely replace subformulas not only with equivalent formulas but with locally equivalent ones as well. Note that the inverse of Theorem 6 holds in the propositional logic: $\models_0 \langle U \equiv V \rangle^F_\pi \equiv (F[U/\pi] \equiv F[V/\pi])$. The local equivalence is there a *criterion* of the substitutional equivalence. It's not the case for the first-order logic, where $(\exists x\, x \approx 0) \simeq (\exists x\, x \not\approx 0)$.

Now, why can we trust the reasonings from Example 1? Let us define a *definition* as a closed formula of the form $\forall \vec{x}(C(\vec{x}) \supset (P(\vec{x}) \equiv D(\vec{x})))$, with P a predicate symbol, $\vec{x}$ a sequence of distinct variables, C a *guard formula* that doesn't contain P, and D a *expansion formula*. Consider any occurrence (F, π), where $F|_\pi = P(\vec{s})$. If we can prove $\langle C(\vec{s}) \rangle^F_\pi$, then we have $\langle P(\vec{s}) \equiv D(\vec{s}) \rangle^F_\pi$ by Lemma 2 and Corollary 4 (sure, on condition that the definition makes part of the theory). Then we can replace $P(\vec{s})$ in (F, π) with $D(\vec{s})$ by Theorem 6, i.e. expand the definition.

Returning back to Example 1, we can guarantee that such an expansion is always possible (since $\langle n - 1 \in \mathbb{N} \wedge n - 2 \in \mathbb{N} \rangle^F_\pi$ holds) and never infinite (since $\langle n - 1 < n \wedge n - 2 < n \rangle^F_\pi$ holds).

Conclusion. We hope that proposed notion of a locally true statement can be useful in automated reasoning, especially, in the frame of interactive theorem proving with various proof assistants. It allows to formalize and check certain correctness properties of formulas with regard to a given theory. It can also serve for the expansion of definitions and for simplification routines.

Note that all our results are intuitionistic. Furthermore, the definition of the (F, π)-image can be easily extended to the language of the (uni)-modal logic: take $\langle U \rangle^{\Box F}_{0.\pi} = \Box \langle U \rangle^F_\pi$ and $\langle U \rangle^{\Diamond F}_{0.\pi} = \Box \langle U \rangle^F_\pi$. Then our statements can be proved for the modal logic **K**, hence for any normal modal logic.

In future, we plan to integrate this technique into a deductive framework and to investigate further its capabilities. In particular, we would like to learn how transformations of a problem during a proof search influence its "local properties", i.e. statements that are locally true in it.

Poster Abstract: Frame-based Representation for Mathematical Concepts

Martin Pollet

School of Computer Science
The University of Birmingham
Birmingham, B15 2TT
England, UK
M.Pollet@cs.bham.ac.uk
www.cs.bham.ac.uk/~mxp

Fachbereich Informatik
Universität des Saarlandes
D-66041 Saarbrücken
Germany
pollet@ags.uni-sb.de
www.ags.uni-sb.de/~pollet

1 Introduction

The representation formalisms for mathematics of current theorem proving systems are usually based on first-order logic, higher-order logic, λ-calculus, type systems or set theory. In [BMM+01] we identified a phenomenon that we called 'loss of information', that is, some important aspects of the knowledge a mathematician has about concepts is hard to reconstruct when it is expressed in one of the formalisms mentioned before. A more elaborated analysis of the differences between representations used in mathematical textbooks and logical formalisms is given in [KP02].

With the poster I want to address one of these shortcomings, namely propose a frame-based data structure to model the hierarchal knowledge mathematicians have about concepts and describe its application for theorem proving.

2 Motivation

New concepts in mathematical textbooks are typically introduced by its definition followed by lemmas stating the basic properties of the concepts, examples are given and connections to already known concepts are introduced.

As an example we take the *degree of a polynomial*. The degree $\deg[p(x)]$ is defined as the highest exponent of the polynomial $p(x)$. The immediate properties are

$$\begin{aligned} \deg[p(x) \cdot q(x)] &= \deg[p(x)] + \deg[q(x)], \\ \deg[p(x) + q(x)] &\leq \max\{\deg[p(x)], \deg[q(x)]\}, \\ \deg[p(x)] &= 0 \text{ iff } p(x) \text{ is a constant.} \end{aligned}$$

These equations are so strongly connected to the concept degree, that their use would not be explicitly referenced.

As another example, a *group* can be defined as an non-empty set G and associative operation $\circ : G \times G \to G$ such that there exists an neutral element $e \in G$ so that for all $x \in G$ holds $x \circ e = x = e \circ x$, and for each $y \in G$ exists an inverse element $z \in G$ with $y \circ z = e = y \circ z$. Immediate results are that the neutral element is unique and there is exactly one inverse element of for each member of G. This property is so fundamental that the inverse element is usually denoted by a function $^{-1} : G \to G$ with $x \circ x^{-1} = e$. The new concept group can be connected to already introduced concepts as monoid, since every group is a monoid.

It seems that the immediate results are used by mathematicians as if they would belong to the definition of the concept itself. Theorems about subsumed definitions allow the transfer of knowledge to other concepts. This hierarchical structure is hard to retrieve when the concepts are defined by a formula and every property is uniformly encoded as theorem.

I argue that the hierarchical knowledge about concepts can be important for theorem proving system, because:

- The structure of mathematical textbooks should instruct the reader how to work with the introduced concepts. Since this structure is important for human reasoning it could be important to enhance the reasoning capabilities of theorem proving systems.
- A mathematician as user of a theorem proving system would expect that the system is able to apply the knowledge he trivially connects with a concept.

3 Frames

Kerber introduced in [Ker92] a frame data structure for mathematical concepts to be used for mathematical databases. A very similar object-oriented encoding is proposed by Vernitski [Ver] for the use in education systems and databases. A frame is data structure with the slots given in the following table:

concept
parameters
equivalent definitions
simple properties
typical examples

The slots contain lists of objects corresponding to the label of the slot. The information encoded in the frame representation shall be used to allow the following functionalities:

- Inheritance of frames: properties of a concept can be accessed from subsumed concepts.

- Flexible formalisation: different views on objects can be expressed by different definitions given in the slot `equivalent definitions`.
- Choose simple condition to establish an assertion: `equivalent definitions` allows to choose how to prove that an object is an instance of a concept.
- Detection of 'situations': detection of a property becomes more independent from the actual problem encoding, it can be accessed via the inheritance of frames.
- Context attached to objects: it becomes possible to get an answer to the question "What is known about the object x."
- Prove trivial assertions: a mechanism to prove assertions that are direct conclusions of the knowledge encoded in frames. This can be used to introduce the notion 'trivial.'
- (Presentation: use the knowledge about concepts to explain proof steps.)

With the poster I will present an implementation of the frame representation in the ΩMEGA system [SBB+02] and possible applications for proof planning.

References

[BMM+01] Christoph Benzmüller, Andreas Meier, Erica Melis, Martin Pollet, Jörg Siekmann, and Volker Sorge. Proof planning: A fresh start? In Manfred Kerber, editor, *IJCAR-Workshop: Future Directions in Automated Reasoning*, pages 30–37, Siena, Italy, 2001.

[Ker92] Manfred Kerber. *On the Representation of Mathematical Concepts and their Translation into First Order Logic.* PhD thesis, Fachbereich Informatik, Universität Kaiserslautern, Kaiserslautern, Germany, 1992.

[KP02] Manfred Kerber and Martin Pollet. On the design of mathematical concepts. Technical Report CSRP-02-06, The University of Birmingham, School of Computer Science, May 2002.

[SBB+02] Jörg Siekmann, Christoph Benzmüller, Vladimir Brezhnev, Lassaad Cheikhrouhou, Armin Fiedler, Andreas Franke, Helmut Horacek, Michael Kohlhase, Andreas Meier, Erica Melis, Markus Moschner, Immanuel Normann, Martin Pollet, Volker Sorge, Carsten Ullrich, Claus-Peter Wirth, and Jürgen Zimmer. Proof development with omega. In Andrei Voronkov, editor, *18th Conference on Automated Deduction (CADE-18)*, pages 143–148, Copenhagen, Denmark, 2002. To appear.

[Ver] Alexei Vernitski. Object-oriented formal mathematical writing style: Mathcaffeine. `http://www.cs.mdx.ac.uk/staffpages/alexei/mathcaffeine.htm`.

OpenMath related Software and Tools

M. N. Riem E. Reinaldo-Barreiro A. M. Cohen H. Cuypers
Eindhoven University of Technology
Den Dolech 2, 5600MB Eindhoven, NL
{mriem,ereinald,amc,hansc}@win.tue.nl

O. Caprotti
RISC-Linz, Johannes Kepler University
A-4020 Linz, Austria
olga.caprotti@risc.uni-linz.ac.at

June 28, 2002

Abstract

This poster is about the OpenMath related Java software developed at RIACA ([4]). This list of software includes: an API for handling OpenMath objects ([5]), MathBook ([2]) an OpenMath-based document markup language, OpenMath shells ([7]), several phrasebooks, and a JSP tag library for handling OpenMath ([6]), among other products.

OpenMath([3]) is a language designed for the representation of mathematical objects. It has evolved from its initial goal of being the lingua franca among computer algebra packages to that of being the language used for communicating mathematics among computational packages. OpenMath focuses on the content side of mathematics with the understanding that for presentation it is better to use MathML([1]).

References

[1] MathML. http://www.w3.org/Math/.

[2] MathBook. http://www.riaca.win.tue.nl/public/mathbook/

[3] OpenMath Standard. http://www.openmath.org/standard/

[4] Research Institute for Applications of Computer Algebra (RIACA). http://www.riaca.win.tue.nl.

[5] RIACA OpenMath library (ROML). http://www.riaca.win.tue.nl/public/openmath/

[6] RIACA OpenMath JSP tag library. http://www.riaca.win.tue.nl/openmath/taglib/

[7] RIACA OpenMath shell. http://www.riaca.win.tue.nl/webstart/

[8] RIACA OpenMath CD Editing tool. http://www.riaca.win.tue.nl/webstart/

Formal Concept Analysis in Isabelle/HOL

Barış Sertkaya
Department of Computer Engineering
Middle East Technical University
06531 ANKARA TURKEY
+90 312 2105548
baris@ceng.metu.edu.tr

August 14, 2002

Formal Concept Analysis arose around 1980's as a result of a systematic framework development of lattice theory applications by a research group in Darmstadt University of Technology, Germany.It is a field of applied mathematics used for deriving implicit conceptual structures out of explicit knowledge. Formal Concept Analysis is based on the mathematical formalization of the philosophical understanding of concept and conceptual hierarchy, borrowing mathematical foundations from lattice theory.A concept is constituted by two parts: its extension, which consists of all the objects belonging to the concept, and its intention which contains the attributes common to all objects of the concept.This formalization allows to form all concepts of a context and introduce a subsumption hierarchy between the concepts, giving a complete lattice called the concept lattice of the context.

Formal Concept Analysis looks at knowledge representation and processing from a mathematical order theoretic point of view.It allows graphical representation of structured knowledge as conceptual hierarchies and mathematical thinking for conceptual data analysis and knowledge processing.Knowledge is represented as concepts in the nodes of the concept lattice ordered by the subsumption relation, giving a taxonomy.Concept lattice is used to query the knowledge and to derive implicit information about the knowledge.

Isabelle is a generic theory development environment for implementing

logical formalisms.It has been instantiated to support reasoning in several object-logics like first-order-logic, higher-order-logic, Zermelo-Fraenkel set theory and T, S4 and S43 modal logic systems.Specialization of Isabelle for Higher Order Logic is called Isabelle/HOL [1].Isabelle/HOL theorem proving environment provides an ML programming language-like interactive interface for theory and proof development.Isar extension of Isabelle [2] provides a conceptually different view on machine-checked proofs through an interpreted language environment which is a more robust and comfortable development platform, with support for theory development graphs and single-step transactions.

Isabelle theories consist of a signature which contains necessary typing and parsing information, and a collection of axioms.A theory definition extends an existing theory with new signature specifications and axioms.Development of a theory may involve many other theories in a hierarchy.Each Isabelle proof works in a single theory associated with the proof state, but many theories may exist at the same time.

My aim is to implement the theory of Formal Concept Analysis in Isabelle/HOL proof assistant.My motivation is to build a Formal Concept Analysis tool for knowledge representation and query on the concept lattice with machine-checked theory. In my implementation I will use the Isar extension for better readability of proofs and ease of implementation.For the ordering and lattices in Formal Concept Analysis, I will use Markus Wenzel's orders, bounds, lattices and complete lattices theories [3] in Isabelle/HOL.Implementation of the theory will proceed following the textbook Formal Concept Analysis [4].I will start the implementation with formal definitions of context, concept, common objects (extent) and common attributes (intent) of a concept and will proceed with the proof of basic theorem.My aim is to be able to prove the major theorems of the subject covered in [4].

Keywords: Knowledge Representation, Formal Concept Analysis, concept lattice, Isabelle/HOL Proof Assistant.

References

[1] Tobias Nipkow, Lawrence C. Paulson and Markus Wenzel. *Isabelle/HOL.A Proof Assistant for Higher-Order Logic*, Springer-Verlag, 2002.

[2] Markus Wenzel. *The Isabelle/Isar Reference Manual*, TU München, 2000 .http://isabelle.in.tum.de/doc/isar-ref.pdf.

[3] Markus Wenzel. *Lattices and Orders in Isabelle/HOL*, TU München, 2001.

[4] Bernhard Ganter and Rudolf Wille. *Formal Concept Analysis*, Springer-Verlag, 1996.

[5] Bernhard Ganter and Rudolf Wille. *Applied Lattice Theory: Formal Concept Analysis*, 1997.

Modelling JavaSpaces™ in μCRL*
Extended abstract

Jaco van de Pol and Miguel Valero Espada
Centrum voor Wiskunde en Informatica,
P.O. Box 94079, 1090 GB Amsterdam, The Netherlands
{Jaco.van.de.Pol, Miguel.Valero.Espada}@cwi.nl

Abstract

We study a formal specification of the shared data space architecture, JavaSpaces. This Java technology provides a virtual space for entities, like clients and servers, to communicate by sharing objects.

We use μCRL [7, 2], a language that combines abstract data types with process algebra [1, 5], to model an abstraction of this coordination architecture. Besides the basic primitives write, read and take, our model also captures the event notification mechanism, transactions and leasing.

The formal model can be used to verify distributed JavaSpaces applications and to evaluate the architecture itself and its possible extensions or modifications.

1 Introduction

It is well known that the design of reliable distributed system can be an extremely arduous task. The parallel composition of processes with a simple behavior can even produce a wildly complicated system. The use of coordination architectures is a suitable way to manage the complexity of specifying and programming large distributed applications.

We study the JavaSpaces technology which is based in Linda [4] and provides a platform for designing distributed computing systems. Protocols built under this technology are modeled as a flow of objects between the space and the external processes. The space handles the details of concurrent access to the common repository. The interface provided by JavaSpaces is essentially composed by insertion (*write*) and lookup primitives (*read, take, readIfExists and takeIfExists*). The objects in the space are *leased* for a period of time, after it they are automatically removed by the space. Programmers can specify a *timeout* to the look up operations meaning how long the operation is willing to wait for a matching entry. JavaSpaces supports a transactional model ensuring that a set of grouped operations are performed atomically. Processes can register their interest in future incoming entries, the space notifies the arrival of new entries by sending events. Figure 1 shows an overview of the architecture, for more information, see: [6, 10].

*Partially supported by PROGRESS, the embedded systems research program of the Dutch organisation for Scientific

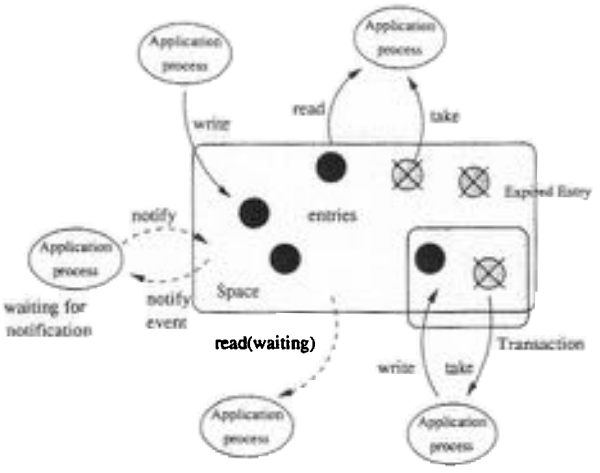


Figure 1. JavaSpaces architecture overview

The goals of this research are: to evaluate the features of the JavaSpaces architecture and its possible variations and to verify the correctness of JavaSpaces applications. Therefore, we specify a formal model covering the main characteristics of the technology in μCRL, a language which combines abstract data types with process algebra [7]. The verification of the systems is done by simulation, visualization of the state space and model checking using the combination of μCRL and CADP tool sets.

2 Specification

A μCRL specification is composed by two parts: the definition of the data types, called **sorts** and the processes definitions which are composed by parameterized actions and process variables combined using the operators: .,+,||, *sum* ($\sum$), that express the possibility of infinite choice of one element of a sort, and the conditional expression "if-then-else" denoted $p \lhd b \rhd q$, where b is a boolean expression, p and q process terms. If b is true then the system behaves as p otherwise it behaves like q.

The space is modeled as a single process called *javaspace*. External agents are implemented as separate processes executed in parallel with the space. A JavaSpaces system is specified in μCRL as follows:

$$\mathsf{System} = \mathsf{javaspace}(...) \parallel \mathsf{external\_P_0}(\mathsf{id_0}:\mathsf{Nat},...) \parallel \mathsf{external\_P_1}(\mathsf{id_1}:\mathsf{Nat},...) \parallel ...$$

The arguments of the *javaspace* process represent the current state of the space. They are composed by: stored objects, active transactions, the current time, active operations, notify registrations, et cetera... External processes interact with the space by means of a set of synchronous actions, derived from the JavaSpaces API. Every process has a unique identification number used by the space to control the access to the common repository. Processes use the sort *Entry* to encapsulate the shared data.

The insertion of a new entry into the space is done with the action *write* which has four arguments: the process identification number of the sort *Nat* (naturals), the entry of the sort *Entry*, the lease of naturals and the reference to a transaction (*null* if it is not submitted to any one). When the space receives a write request, it automatically encapsulates the entry, with its lease and the reference to the transaction, in an new data sort (*Object*) and stores it in the database which has the structure of a *Set*.

Research NWO, the Dutch Ministry of Economic Affairs and the Technology Foundation STW, grant CES.5009.

The JavaSpaces specification says that a look up request searches in the space for an Entry that matches the template provided in the action. If the match is found, a reference to a copy of the matching entry is returned. If no match is found, *null* is returned. We do not use templates to model the matching operation but by adding to every invocation one predicate, as argument, which determines if an Entry matches or not the action. This predicate belongs to the sort *Query*, defined by the user according to the specification of the *Entry*. The sort must include the operator *test* used to perform the matching. An entry of the space will match a look up action if it satisfies the associated *test* predicate. Look up operations are not atomic, they are performed by means of two synchronous actions. First, the process makes the request and blocks waiting for a matching entry or for the timeout expiration. The space stores this request in a set with other pending requests and afterward it returns, using a different action, a matching entry, if it is found, or a *null* value, if the timeout has expired.

Processes can join several operations in such a way that either all of them complete or none are executed. The behavior of all the primitives would be slightly different depending on whether they are executed under a transaction or not. Transactions are modeled by a system of lock sets in which the changes done by the primitives are stored until the transaction commits or aborts.

The space also manages a set of registered processes that wait for the arrival of new entries. When a new entry matches a registration query, the space send a notification event to the associated listener through a non reliable network (modeled as a different process), that can always loose or duplicate it.

3 Example

In this section, we are going to formalize a simple JavaSpaces application to show the possibilities of the μCRL tool set for system model checking. The system is inspired by the classical arcade game Ping-Pong, in which two players throw one ball from one to the other. This example has been taken from the chapter 5 of the book "JavaSpaces™ Principles, Patterns, and Practice" [6]. The players are modeled by two processes called Ping and Pong which communicate by means of an entry that encapsulates the ball. They play during a fixed number of rounds. The following fragment of μCRL code specifies the players process:

```
proc  player(id:Nat,name:Name,round:Nat) =
      take(id, NULL, FOREVER, forMe(name))
      .∑e:Entry (Return(id,e)
        .print(name)
        .write(id, ball(other(name)),
           NULL, FOREVER))
      .player(id,name,S(round))
      ◁ lt(round, maxRounds) ▷ δ
```

A player, first tries to get the ball from the space by performing a *take* action, then he executes the external action *Print* used to inform the environment that he has got the entry. Finally, the player writes the ball back and loops increasing the counter of rounds. The complete system is composed by the parallel composition of the two instances of the player process, *Ping* and *Pong*, and the *javaspaces* process, which initially has a ball directed to Ping.

To each μCRL specification belongs a labeled transition system (LTS) being a directed graph, in which the nodes represent states and the edges are labeled with actions. If this transition system has a finite

number of states the μCRL tool set can automatically generate this graph. Subsequently, the CÆSAR ALDÉBARAN DEVELOPMENT PACKAGE (CADP) can be used to visualise and to analyse this transition system. Figure 2 shows the generated LTS of a the external behavior of the two rows game.

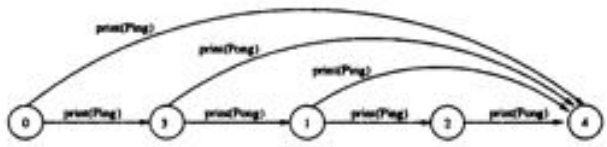


Figure 2. External behavior of 2 rows simple Ping-Pong game

The fair execution of the game is 0-3-1-2-4. If the time reaches the bound the system halts, and it's possible that the system stops before all the rows have been completed, this behavior corresponds to the transitions 0-4, 3-4 and 1-4.

Properties can be automatically verified by the Evaluator tool from the CADP package. These properties are expressed in in the regular alternation-free μ-calculus [9]. For example, a safety property expresses the prohibition of "bad" execution sequences. The following formula means that the player *Ping* cannot throw the ball twice in a row:

$$[\ \text{true}^*\ .\ \textit{"print(Ping)"}\ .\ (\text{not}\ \textit{"print(Pong)"})^*\ .\textit{"print(Ping)"}\ ]\ \text{false}$$

Our model supports more complicated systems, for example, we introduce a small change in the rules of the game: once a player catched the ball, he has one time unit to put it back in the space, otherwise he looses the game. We model this approach by using transactions. After a player has performed the *take*, he creates a transaction leased for one second. Once the *write* operation is done, the transaction can commit:

```
proc  player(id:Nat,name:Name,round:Nat) =
      take(id, NULL, FOREVER, forMe(name))
      .∑e:Entry (Return(id,e)
        .∑trc:Nat (create(id, trc, S(0))
          .(write(id, ball(other(name)),
              trc, FOREVER)
            + Exception(id, trc).looser(name))
          .(print(name)
            + Exception(id, trc).looser(name))
          .(commit(id, trc))
            + Exception(id, trc).looser(name)))
      .player(id,name,S(round))
      ◁ lt(round, maxRounds) ▷ δ
```

4 Conclusion

We have built a formal specification of the JavaSpaces technology that covers its main features, see references [11, 12]. The formal model attempts to study and clarify the issues that were unclear or ambiguous in the JavaSpaces specification. Furthermore, the model has been successfully used to model check some simple distributed applications. Other approaches to verification of coordination architectures can be found in [8, 3].

References

[1] J.A. Bergstra and J.W. Klop. Algebra of communicating processes with abstraction. *TCS*, 37(1):77–121, 1985.

[2] S.C.C. Blom, W.J. Fokkink, J.F. Groote, I.A. Langevelde, B. Lisser, and J.C. van de Pol. μCRL: a toolset for analysing algebraic specifications. In *Proc. of CAV*, LNCS 2102, pages 250–254. Springer, 2001.

[3] Nadia Busi and Gianluigi Zavattaro. Publish/subscribe v.s. shared dataspace coordination infrastructures. In *Workshop on Web-based Infrastructures and Coordination Architectures for Collaborative Enterprises (WETICE-2001)*. IEEE Computer Society Press, 2001.

[4] N. Carriero and D. Gelernter. *How to Write Parallel Programs: A First Course.* MIT Press, 1990.

[5] W. Fokkink. *Introduction to Process Algebra.* Texts in Theoretical Computer Science. Springer, 2000.

[6] E. Freeman, S. Hupfer, and K. Arnold. *JavaSpaces principles, patterns, and practice.* Addison-Wesley, Reading, MA, USA, 1999.

[7] J.F. Groote and M.A. Reniers. Algebraic process verification. In J.A. Bergstra et al., editor, *Handbook of Process Algebra*, chapter 17. Elsevier, 2001.

[8] J.M.M. Hooman and J.C. van de Pol. Formal verification of replication on a distributed data space architecture. In *Proceedings ACM SAC, Coordination Models, Languages and Applications*, page (to appear), Madrid, 2002. ACM press.

[9] R. Mateescu. *Verification des proprietes temporelles des programmes paralleles.* PhD thesis, Institut National Polytechnique de Grenoble, 1998.

[10] SUN Microsystems. *Jini*^{tm} *Technology Core Platform Specification*, 1.1 edition, October 2000. See `http://www.sun.com/jini/specs/`.

[11] J.C. van de Pol and M. Valero Espada. Formal specification of JavaSpaces^{TM} architecture using μcrl. In *Proc. of COORDINATION*, pages 274 – 290. Springer, 2002.

[12] J.C. van de Pol and M. Valero Espada. μcrl specification of event notification in javaspaces^{TM}. In *JJCC 2002*, pages (191–204). Kronos, 2002.