To appear in J. Calmet, C. Limongelli (Eds.), Proceedings of the International Symposium on Design and Implementation of Symbolic Computation Systems (DISCO'96), Karlsruhe, LNCS, Springer.

Structures for Symbolic Mathematical Reasoning and Computation

Karsten Homann and Jacques Calmet

Universität Karlsruhe
Institut für Algorithmen und Kognitive Systeme
Am Fasanengarten 5 · 76131 Karlsruhe · Germany
{homann,calmet}@ira.uka.de

Abstract. Recent research towards integrating symbolic mathematical reasoning and computation has led to prototypes of interfaces and environments. This paper introduces computation theories and structures to represent mathematical objects and applications of algorithms occuring in algorithmic services. The composition of reasoning and computation theories and structures provide a formal framework for the specification of symbolic mathematical problem solving by cooperation of algorithms and theorems.

Keywords: Integration of computing and reasoning paradigms, Interfaces

1 Introduction

The combination of systems performing any kind of mathematical computation is a young and active research field. We call such systems *mathematical services* which include computer algebra systems (CAS), theorem provers and proof checkers (TPS), mathematical knowledge representation systems, tools for visualization and editing, ... A major requirement to qualify as mathematical service is the ability to cooperate by incremental and restartable computation and deduction.

There has been work to integrate homogeneous services, i.e. combining CAS in CAS/π [13] and OpenMath [1], combining TPS in OMRS [9] and many others. The integration of CAS and TPS in a common environment has not yet led to powerful systems. However, some prototypes were developed which prove the advantages of such a combination, e.g. Analytica [6], interfaces between Hol and Maple [10], Isabelle and Maple [2], and Nuprl and Weyl [12].

We classify communication and cooperation methods for such environments in [5]. However, we believe that the lack of a general formal framework is one reason why nowadays prototypes do not qualify as mathematical services. [1] introduces a semantics of mathematical objects and interfaces and [9] initiates the formal specification of reasoning theories and reasoning structures for combining logical services. However, a formal framework to combine CAS and TPS is not given.

Another reason is that nowadays CAS behave like black boxes. They are not designed to allow provisional or restartable computation and do not provide access to their context. To implement contexts in mathematical reasoning and computation has been initiated in [8] by proposing extensions to the interactive mathematical proof system IMPS [7]. Again, there is no semantics of the integration.

The long term goal of this work is to provide a methodology for constructing complex systems by composing mathematical services while the actual goal of this paper is to provide a formal method for their specification. Such a methodology has to consist of both a formal theory and structures for its representation. We introduce *computation structures* to represent objects and applications of algorithms appearing in algorithmic services by extending and modifying the concept of reasoning structures given in [9]. The notions, definitions and theorems given here are intentively kept very close to reasoning theories to illustrate their similarities and to construct natural combinations of deduction and computation. Because of the restricted length of the paper we left some notions informal or without examples and give references. The composition of reasoning and computation theories and structures is subject of ongoing research.

The paper is organized as follows. Mathematical services are defined in section 2. They include interactive and automated open systems performing any kind of mathematical computation. The formal specification of reasoning structures is sketched by some examples of IMPS in section 3. Section 4 and 5 introduce computation structures and their construction and manipulation. Finally, section 6 illustrates examples of combining structures for symbolic mathematical reasoning and computation.

2 Specification of Mathematical Services

A formal description of mathematical services has to ensure interaction capabilities to combine several systems, to contain meta information and justifications on functions, it should allow easy extension by subpackages and interaction of existing systems. Such a description should consist of several levels such as the communication level for dynamic distribution of messages and events among the services and the abstract functionality of the interface (see [17] and [16] for examples).

The formal approach given in this paper provides a description of theories for reasoning and computation respectively. [9] introduces the concept of *Open Mechanized Reasoning Systems* which we call *logical services* $\mathcal{LS}$. These systems are based on reasoning theories and structures. By defining computation theories for *algorithmic services* $\mathcal{AS}$ and composition of theories we introduce a framework for the structures given in the next section.

Definition 1.
A ***mathematical service*** ($\mathcal{MS}$) is an implementation of a mathematical computation or processing and an interaction component.

In the rest of the paper we restrict $\mathcal{MS}$ to reasoning and computation systems.[1]

$$\begin{aligned}
\mathcal{MS} &= \mathcal{LS} \mid \mathcal{AS} \\
\mathcal{LS} &= \textit{Reasoning System} + \textit{Interaction} \\
\mathcal{AS} &= \textit{Computation System} + \textit{Interaction} \\
\textit{Reasoning System} &= \textit{Reasoning Theory} + \textit{Control} \\
\textit{Computation System} &= \textit{Computation Theory} + \textit{Control} \\
\textit{Reasoning Theory} &= \textit{Sequents} + \textit{Rules} \\
\textit{Computation Theory} &= \textit{Objects} + \textit{Algorithms}
\end{aligned}$$

Reasoning theories are defined in [9] to consist of a sequent system $S_{sys} = \langle S, C, \models, I, \_[\_] \rangle$, a set of identifiers Id and a rule set $\tilde{r} \in Rset[S_{sys}, Id]$. Sequents allow the use of schematic variables and can be instantiated. A rule is a relation on tuples consisting of a non-empty sequence of sequents and a finite set of constraints.

Definition 2.
An ***object system*** is a structure

$$O_{Sys} = \langle O, C, \models, I, \_[\_] \rangle$$

with a set of ***objects*** O, a set of ***constraints*** C, a ***constraint satisfaction mechanism*** $\models \subseteq (P_\omega(C) \times C)$, a set of ***instantiations*** I, and an ***application of instantiations*** $\_[\_] : [O \times I \to O]$ and $\_[\_] : [C \times I \to C]$.

Objects in CAS include polynomials, numbers, matrices, equations, sequences, sets, expressions and many others. Conditions in terms of constraints are provided as local context of objects, e.g. type constraints. We allow objects and constraints to be schematic and they can be instantiated.

Constraints are also introduced to guarantee certain properties when applying algorithms. Such an algorithm is a labelled relation on tupels of objects, input parameters and a unique result, which is closed under instantiation, with Id a set of identifiers.

Definition 3.
A ***set of algorithms over an object system*** $A \in Algs[O_{Sys}, Id]$ is such that

$$\begin{aligned}
Algs[O_{Sys}, Id] = [Id \longrightarrow \{ A \subseteq (O^* \times O \times P_\omega(C)) \mid\ & (\forall \langle \bar{o}, o, \tilde{c} \rangle \in A)(\forall \iota \in I) \\
& (\langle \bar{o}, o, \tilde{c} \rangle [\iota] \in A) \}]
\end{aligned}$$

A computation theory specifies a set of objects and a set of algorithms. They are designed to provide a methodology for the design and implementation of future CAS according to the ideas given in [3]. The intermediate computations are represented by computation structures which are introduced in the next section.

[1] $\mathcal{LS}$ corresponds to OMRS in [9], $\mathcal{AS}$ to OMCS and $\mathcal{MS}$ to OMME in [4].

Definition 4.
Let O_{Sys} be an object system, Id a set of identifiers and $\tilde{a} \in Algs[O_{Sys}, Id]$. A ***computation theory*** CT is a structure

$$CT = \langle O_{Sys}, Id, \tilde{a} \rangle \ .$$

To define a disjoint composition of theories for composing proofs and computations is subject of ongoing research. It is done by gluing together seperate reasoning and computation theories using additional rules called *bridges*. Such bridges may include syntax transformations or instructions for rigorous systems how to verify results of external theories. Section 6 sketches two examples of bridges.

3 Reasoning Structures

Let $RT = \langle S_{Sys}, Id, \tilde{r} \rangle$ be an arbitrary but fixed reasoning theory. Reasoning structures [9] were designed to represent the structures appearing in the construction of derivations and proofs. They can be illustrated as labelled graphs with edges and two kinds of nodes: sequent nodes and link nodes. These nodes are labelled by their corresponding sequents and justifications respectively. To enable vertical flexibility the justifications may contain nested reasoning structures together with an instantiation as well as premiss and conclusion nodes.

Definition 5.
Basic reasoning structures are elements of $RS_0[RT, SN, LN]$, which is the set of structures $rs = \langle Sn, Ln, g, sg, sL, lL \rangle$ such that

1. $Sn \in P_\omega(SN)$ set of sequent nodes of rs,
2. $Ln \in P_\omega(LN)$ set of link nodes of rs,
3. $g : [Ln \to Sn]$ maps to associated goal sequent node,
4. $sg : [Ln \to Sn^*]$ maps to possibly empty set of subgoal sequent nodes,
5. $sL : [Sn \to S]$ sequent node labelling map;
6. $lL : [Ln \to [Id \times P_\omega(C)]]$ link node labelling map such that $\forall ln \in Ln$:

$$lL(ln) = \langle id, \tilde{c} \rangle \wedge \bar{s} = sL(sg(ln)) \wedge s = sL(g(ln)) \Rightarrow \exists \tilde{c}' : \tilde{c} \models \tilde{c}' \wedge \langle \bar{s}, s, \tilde{c}' \rangle \in \tilde{r}(id)$$

For example, a basic reasoning structure rs_I in the proof construction of theorem 6 is illustrated in figure 1 and expanded in figure 2. The labelling of the sequent nodes was omitted because of readability.

Theorem 6.
$\forall x, c \in \mathbb{R}$ *and partial functions* $f : \mathbb{R} \to \mathbb{R}$ *such that* $\mathcal{D}(f)(x) \downarrow$:

$$\mathcal{D}(c * f(x)) = c * \mathcal{D}(f)(c)$$

Vertical flexibility allows to nest structures within others to achieve better presentations and readability of proofs.

```
((#{IMPS-sqn 1}
  (FORCE-SUBSTITUTION
   (#{IMPS-sqn 2}
    (PRODUCT-RULE
     (#{IMPS-sqn 5})
     (#{IMPS-sqn 6} GROUNDED)))
   (#{IMPS-sqn 3} GROUNDED))))
```

Fig. 1. Illustration of a deduction graph in IMPS

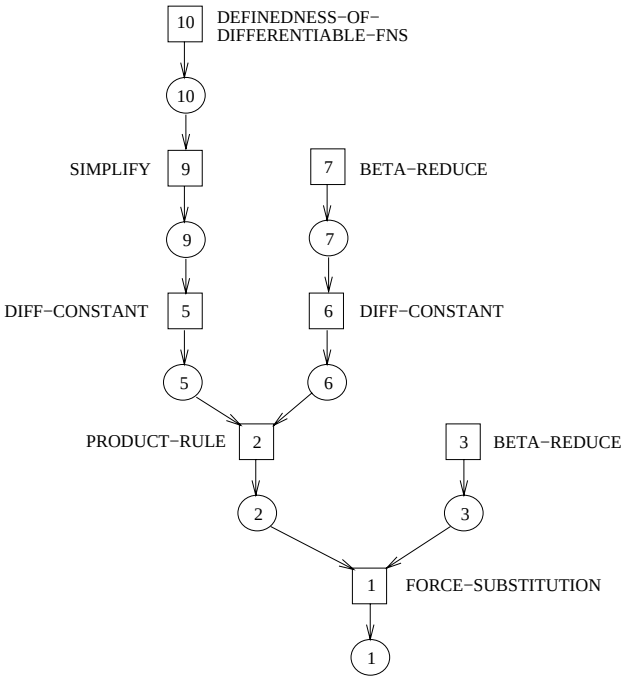

Fig. 2. Reasoning structure rs_I

Definition 7.
A ***reasoning structure*** is an element of

$$RS[RT, SN, LN] = \bigcup_{n \in \mathbb{N}} RS_n[RT, SN, LN]$$

with $RS_n[RT, SN, LN]$ the set of ***reasoning structures of level*** n. RS_{n+1} is recursively defined as the set of structures $rs = \langle Sn, Ln, g, sg, sL, lL \rangle$ satisfying (1.-5.) of definition 5 and

6_{n+1}. $vB : [Vk \to [Id \times P_\omega(C)] + [P_\omega(C) \times I \times [Sk^*, Sk] \times DS_n[DT, SK, VK]]]$ is a natural extension of the link labelling function.

Reasoning structures can be instantiated and can represent derivations and proofs. Such proofs are derivations without open assumptions, e.g. figure 2. Theorems about reachability, instantiation of derivation, elimination of nesting and derivation of trees are given in [9].

4 Computation Structures

Let $CT = \langle O_{Sys}, Id, \tilde{a} \rangle$ be an arbitrary but fixed computation theory. Corresponding to reasoning structures we define computation structures to represent graphs of computations which are constructed by the application of algorithms. The graphs consist of edges and two kinds of nodes:

- *object nodes*, ON, represented by circles and
- *algorithm nodes*, AN, represented by squares.

Object nodes are labelled by $\Gamma|o$ consisting of a local context Γ of constraints and an object o. Algorithm nodes are labelled by *explanations* which consist of the name of an algorithm with its parameters or a quadrupel $\langle C, \iota, [on], cs \rangle$ of a *nested computation structure* cs such that C is the set of constraints, ι instantiation, and $[on]$ sequence of object nodes. Each algorithm node has a unique link to its result and there are links from object nodes to algorithm nodes. These graphs allow the representation of contexts for symbolic algebraic computation.

Definition 8.

$$cs = \langle On, An, r, p, oL, aL \rangle \in CS_0[CT, ON, AN]$$

is a ***basic computation structure*** with

1. $On \in P_\omega(ON)$ set of object nodes of cs,
2. $An \in P_\omega(AN)$ set of algorithm nodes of cs,
3. $r : [An \to On]$ maps to resulting object node,
4. $p : [An \to On^*]$ maps to possibly empty set of input parameter objects,
5. $oL : [On \to O]$ object node labelling map,
6. $aL : [An \to [Id \times P_\omega(C)]]$ algorithm node labelling map such that $\forall an \in An$:

$$aL(an) = \langle id, \tilde{c} \rangle \wedge \bar{o} = oL(p(an)) \wedge o = oL(r(an)) \Rightarrow \exists \tilde{c}' : \tilde{c} \models \tilde{c}' \wedge \langle \bar{o}, o, \tilde{c}' \rangle \in \tilde{a}(id)$$

Definition 9.
The ***labelled graph*** $Graph(cs)$ of a basic computation structure $cs = \langle On, An, r, p, oL, aL \rangle$ is the graph with nodes $On \cup An$ and edges

$$\{(an, r(an)) \mid an \in An\} \cup \{(on, an) \mid an \in An \wedge on \in e(an))\}$$

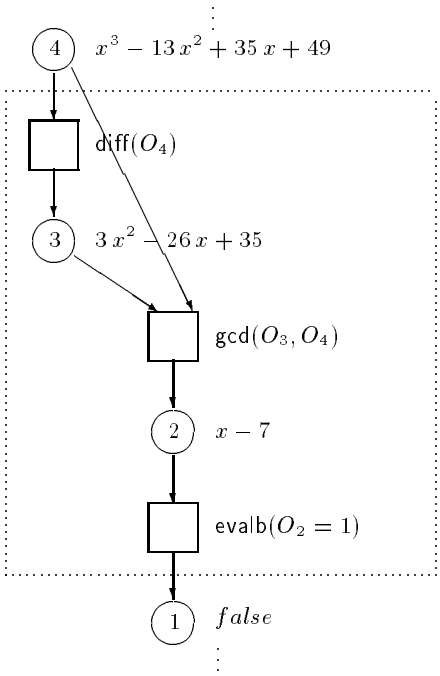

Fig. 3. Parts of a computation structure cs_I

Relying on basic structures we introduce vertical flexibility for algorithmic services by nesting structures. Computation structures are defined by their depth and recursive elements. Figure 3 illustrates the graph of a basic computation structure without constraints.

To increase the readability of computations and to group several steps we introduce vertical flexibility by nesting computation structures. The nested structure usually includes schematic variables and one can assign a unique identifier, e.g. `Squarefree`. Nesting computation structures correspond to the concept of subroutines in symbolic computation systems. To allow for cooperation within the computation of the squarefree property a CAS must handle structures for subroutines.

Definition 10.

$$cs = \langle On, An, r, p, oL, aL \rangle \in CS[CT, ON, AN] = \bigcup_{n \in \mathbb{N}} CS_n[CT, ON, AN]$$

is a ***computation structure***. The set CS_{n+1} of ***computation structures of depth*** n is recursively defined as the set of structures such that (1.-5.) in definition 8 holds and

6_{n+1}. $aL : [An \to [Id \times P_\omega(C)] + [P_\omega(C) \times I \times [On^*, On] \times CS_n[CT, ON, AN]]]$
the algorithm nodes labelling map such that (6.) holds for basic structures

and if $aL(an) = \langle \tilde{c}, \iota, [\bar{o}n, on], cs' \rangle \ \wedge \ cs' = \langle On', An', r', p', oL', aL' \rangle$ then

$$[\bar{o}n, on] \in (On')^* \ \wedge \ oL'([\bar{o}n, on])[\iota] = [oL(p(an)), oL(r(an))]$$

Computation structures can include schematic variables to allow for schematic computations. It was defined for such a computation to be closed under instantiation. Examples for objects with schematic variables are $gcd(n, 2n) = n$ or $\int x^n \, dx = \frac{x^{n+1}}{n+1}$.

Definition 11.
An ***instantiation of a computation structure***

$$cs = \langle On, An, r, p, oL, aL \rangle \in CS[CT, ON, AN] \text{ and } \iota \in I$$

is $cs[\iota] = \langle On, An, r, p, oL[\iota], aL' \rangle$ such that

$\forall an \in An$
(ra) $\quad aL(an) = \langle id, \tilde{c} \rangle \ \Rightarrow aL'(an) = \langle id, \tilde{c}[\iota] \rangle$ and
(nest) $\quad aL(an) = \langle \tilde{c}, \iota_1, [\bar{o}k, ok], cs_1 \rangle \ \Rightarrow aL'(an) = \langle \tilde{c}[\iota], \iota \circ \iota_1, [\bar{o}k, ok], cs_1 \rangle$.

Computation structures represent the computation of a resulting object from a set of given input parameters if no algorithm application has unsolved constraints. The following lemma states that vertical flexibility can be eliminated by vertical unfolding.

Lemma 12. ***Elimination of nesting***
Let $cs \in CS[CT, ON, AN]$ be a computation structure with result o and input parameters $\tilde{o}$ then there exists a computation structure $cs_0 \in CS[CT, ON, AK]$ of depth 0 with result s and input parameters $\tilde{o}$.

5 Construction and Manipulation of Structures

Operations $\mathcal{O}_{rs} = \{addS, linkR, solveC, linkN\}$ for constructing and manipulating reasoning structures are given in [9], including the empty reasoning structure $emptyRS$, introduction of sequents, link by forward and backward application of rules, constraint solving, and nesting. This section introduces the corresponding operations on computation structures.

Definition 13.

$$emptyCS = \langle \emptyset, \emptyset, \emptyset, \emptyset, \emptyset, \emptyset \rangle$$

is the empty computation structure.

Since $emptyCS = emptyRS$ we denote empty structures as $empty$.

Definition 14.
Objects $o \in O$ are introduced by $addO$

$$addO(cs, o) = \langle On \cup \{on\}, An, r, p, oL\{on \mapsto o\}, aL\rangle$$

with new object node $on \in ON \setminus On$.

Definition 15.
Let $on \in On, o\bar{n} \in On^*, a = \langle id, \bar{o}, o, \tilde{c}\rangle \in \tilde{a}, \models \{\bar{o} \sim oL(o\bar{n}), o \sim oL(on)\}$. Links by application of algorithms are defined as

$$linkA(cs, o\bar{n}, on, a) = \langle On, An \cup \{an\},$$
$$r\{an \mapsto on\}, p\{an \mapsto o\bar{n}\}, oL, aL\{an \mapsto \langle id, \tilde{c}\rangle\}\rangle$$

with new algorithm node $an \in AN \setminus An$.

Definitions of operations for solving constraints, nesting links, extending operations to nested computation structures and additional operations to combine structures by introducing bridge rules correspond to [9] and are given in [11]. One can easily proof that the operations of $\mathcal{O}_{cs} = \{addO, linkA, solveC, linkN\}$ are sound, independent, and complete generators of any computation structure out of $empty$.

Let $a = \langle id, \bar{o}, o_0, \tilde{c}\rangle$ be an algorithm such that $\bar{o} = [o_1, \ldots, o_n]$ and $\models \tilde{c}$. We define the application of an algorithm to be a computation structure as a sequence of operations from $\mathcal{O}_{cs}$.

Definition 16.
Let $cs = \langle On, An, r, p, oL, aL\rangle$ be a computation structure such that $o\bar{n} = [on_1, \ldots, on_n] \in On^* \wedge \forall\ 1 \leq j \leq n : oL(on_j) = o_j$. Let $on_0 \notin On \wedge an_0 \notin An$. $applyA(cs, a, o\bar{n})$ is the sequence of operations

- $cs_0 = addO(cs, o_0)$ by introducing on_0;
- $cs_1 = linkA(cs_0, o\bar{n}, on_0, a)$ by introducing an_0;
- $cs' = solveC(cs_1, vk_0, \emptyset)$.

6 Examples

6.1 Isabelle and Maple

We designed and implemented an interface between the tactical theorem prover Isabelle [15] and Maple [14] by extending the simplifier of Isabelle ([2], figure 4). The simplifier is extended by new kinds of rules to call external functions of the CAS. The interactive proof involves computation structures with only one algorithm application. As an example, the inductive proof of

$$\forall n \in \mathbf{N} : 5 \leq n \Longrightarrow n^5 \leq 5^n.$$

expands all of the products in the induction step. $x+1$ is an object in the sequent of a reasoning structure which serves as input node to the `expand` algorithm of Maple.

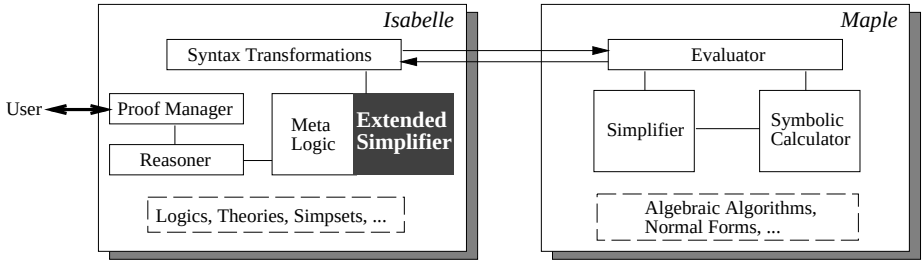

Fig. 4. Schematic Link between Isabelle and Maple

```
- by (res_inst_tac [("P",
= "%y.expand((x + 1) ^ 5) <= y")] expandE 1);
n ^ 5 <= 5 ^ n
 1. !!x. [| x : Nat; 5 <= x; x ^ 5 <= 5 ^ x |] ==>
        expand((x + 1) ^ 5) <= expand(5 ^ (x + 1))
- by (asm_simp_tac Nat_simplify_ss 1);
n ^ 5 <= 5 ^ n
 1. !!x. [| x : Nat; 5 <= x; x ^ 5 <= 5 ^ x |] ==>
         x ^ 5 + 5 * x ^ 4 + 10 * x ^ 3 +
         10 * x ^ 2 + 5 * x + 1 <= 5 * 5 ^ x
```

The interface was implemented without explicit construction of computation structures. The semantic of the interaction is given in [2].

6.2 Imps and Calvin

To introduce external function calls in a rigorous theorem prover is difficult. Either algorithms must be proven to be correct or the results of algorithms must be fully proven by the proof system because rigorous systems can not trust external tools. [10] introduces the concept of trust in cooperated reasoning and computation. Such cooperations can be used to guide proofs where verification is easier as computation by theorems. Since a huge subset of mathematical algorithmic computation can be easily verified by simple equations we started to implement an interface between Imps [7] and Maple along the concept of external macetes introduced by [8]. External computation and verification of results could be successfully implemented, e.g., for factorisation of polynomials, anti-derivatives, gcd computations, Hensel lifting, and chinese remainder. Problems with completeness of results occur by applications of the `solve` operator, i.e. solving linear and differential equation systems, recurrence relations and equational simplification. The verification of numerical or "direct" computations, however, is of same complexity.

A link with an existing CAS allows only to apply computation structures with just one algorithm node. To implement a CAS capable to manage contexts is one of our long term research projects. The system, called Calvin, allows

restartable and incremental application of algorithms. These contexts can be used to guarantee correctness of computations, e.g. requesting the context of n when integrating x^n.

7 Conclusion and Further Research

This paper introduces computation theories and structures which serve as a formal framework and tool for representing mathematical objects and applications of algorithms appearing in algorithmic services. The composition of reasoning and computation theories and structures will provide a theoretical and technical framework for the specification and implementation of symbolic mathematical problem solving by cooperation of algorithms and theorems.

Nowadays CAS behave like black boxes. Cooperation with such systems result in computation structures with just one algorithm node. Although the benifit of the given theory still is a formal description of the cooperation between provers as well as several CAS and horizontal flexibility, major redesign of CAS is required to achieve high level cooperation. We have shown how computation structures naturally correspond to reasoning structures and give access to intermediate knowledge. To develop the CAS Calvin along the concepts presented in this paper is part of an ongoing research project.

Cooperating services must provide some kind of access to reasoning and computation structures. The functionality of interfaces for distributed reasoning and computation are given in [16] and [11] respectively. Although they might not explicitly be implemented cooperations among mathematical services are based on manipulation of structures for symbolic mathematical reasoning and computation.

References

1. J. Abbott, A. van Leeuwen, A. Strotmann
   Objectives of OpenMath. Submitted to Journal of Symbolic Computation, 1995.
2. C. Ballarin, K. Homann, J. Calmet
   Theorems and Algorithms: An Interface between Isabelle and Maple. In A.H.M. Levelt (Ed.), Proceedings of International Symposium on Symbolic and Algebraic Computation (ISSAC'95), pp. 150–157, ACM press, 1995.
3. B. Buchberger
   Mathematica: Doing Mathematics by Computer? In A. Miola (Ed.), Advances in the Design of Symbolic Computation Systems, Texts and Monographs in Symbolic Computation, Springer, 1996.
4. J. Calmet, K. Homann
   Distributed Mathematical Problem Solving. In Proceedings of 4th Bar-Ilan Symposium on Foundations of Artificial Intelligence (BISFAI'95), pp. 222-230, 1995.
5. J. Calmet, K. Homann
   Classification of Communication and Cooperation Mechanisms for Logical and Symbolic Computation Systems. In F. Baader, K.U. Schulz (Eds.), Proceedings of First International Workshop on Frontiers of Combining Systems (FroCoS'96), pp. 133–146, Kluwer Series on Applied Logic, 1996.

6. E. Clarke, X. Zhao
Combining Symbolic Computation and Theorem Proving: Some Problems of Ramanujan. In A. Bundy (Ed.), Automated Deduction (CADE-12), pp. 758–763, LNAI 814, Springer, 1994.
7. W.M. Farmer, J.D. Guttman, F.J. Thayer
IMPS: An Interactive Mathematical Proof System. In Journal of Automated Reasoning 11:213–248, 1993.
8. W.M. Farmer, J.D. Guttman, F.J. Thayer
Contexts in Mathematical Reasoning and Computation. In Journal of Symbolic Computation 19:201–216, 1995.
9. F. Giunchiglia, P. Pecchiari, C. Talcott
Reasoning Theories – Towards an Architecture for Open Mechanized Reasoning Systems. In F. Baader, K.U. Schulz (Eds.), Proceedings of First International Workshop on Frontiers of Combining Systems (FroCoS'96), pp. 97–114, Kluwer Series on Applied Logic, 1996.
10. J. Harrison, L. Théry
Extending the HOL Theorem Prover with a Computer Algebra System to Reason About the Reals. In J.J. Joyce, C.-J.H. Seger (Eds.), Higher Order Logic Theorem Proving and its Applications (HUG'93), pp. 174–184, LNCS 780, Springer, 1993.
11. K. Homann
Symbolic Mathematical Problem Solving by Cooperation of Algorithmic and Logical Services, (in german), Dissertation, University of Karlsruhe, 1996.
12. P. Jackson
Exploring Abstract Algebra in Constructive Type Theory. In A. Bundy (Ed.), Automated Deduction (CADE-12), pp. 590–604, LNAI 814, Springer, 1994.
13. N. Kajler
CAS/PI: a Portable and Extensible Interface for Computer Algebra Systems. In Proceedings of ISSAC'92, pp. 376–386, ACM press, 1992.
14. M.B. Monagan, K.O. Geddes, K.M. Heal, G. Labahn, S. Vorkoetter
Maple V Programming Guide, Springer, 1996.
15. L.C. Paulson
Isabelle — A Generic Theorem Prover, LNCS 828, Springer, 1994.
16. C. Talcott
Handles – Specialized logical services, Working Draft, 1994.
17. N. Venkatasubramanian, C. Talcott
Reasoning about Meta Level Activities in Open Distributed Systems, to appear in PODC, 1995.

This article was processed using the LaTeX macro package with LLNCS style