Implementing Deep Inference in TOM

Ozan Kahramanoğulları[1], Pierre-Etienne Moreau[2], Antoine Reilles[3]

[1] Computer Science Institute, University of Leipzig
International Center for Computational Logic, TU Dresden
ozan@informatik.uni-leipzig.de
[2] LORIA & INRIA, Nancy, France
[3] LORIA & CNRS, Nancy, France
{Pierre-Etienne.Moreau,Antoine.Reilles}@loria.fr

Abstract. The calculus of structures is a proof theoretical formalism which generalizes sequent calculus with the feature of deep inference: in contrast to sequent calculus, the calculus of structures does not rely on the notion of main connective and, like in term rewriting, it permits the application of the inference rules at any depth inside a formula. Tom is a pattern matching processor that integrates term rewriting facilities into imperative languages. In this paper, relying on the correspondence between the systems in the calculus of structures and term rewriting systems, we present an implementation of system BV of the calculus of structures in Java by exploiting the term rewriting features of Tom. This way, by means of the expressive power due to Java, it becomes possible to implement different search strategies. Since the systems in the calculus of structures follow a common scheme, we argue that our implementation can be generalized to other systems in the calculus of structures for classical logic, modal logics, and different fragments of linear logic.

1 Introduction

Developing new representations of logics, which address properties that are central to computer science applications, has been one of the challenging goals of proof theory. One of the crucial needs of such a line of research is the appropriate set of implementation tools, which are in harmony with the underlying proof theoretical formalism. Such tools then make it possible to test conjectures on the logic, proof theory of which is studied. This way, they do not only allow researchers to save time by producing counter examples for false conjectures, but also shed light to potential applications of the logic being studied.

The calculus of structures [5] is a proof theoretical formalism, like natural deduction, sequent calculus, and proof nets. The calculus of structures generalizes the sequent calculus while keeping properties, such as locality and modularity (see, e.g. [3, 15]), in focus that are important for computer science applications. Structures are expressions intermediate between formulae and sequents which unify these two latter entities. This way, they provide a greater control over the mutual dependencies between logical relations. The main feature that distinguishes this formalism is *deep inference*: in contrast to the sequent calculus, the calculus of structures does not rely on the notion of main connective, and permits the application of the inference rules at any depth inside a structure. Applicability of the inference rules at any depth results in a richer combinatorial analysis of proofs than in the sequent calculus. Because proofs are constructed by manipulating and annihilating substructures, this formalism brings shorter proofs than all other formalisms supporting analytical proofs.

The calculus of structures was conceived, in [5], for introducing a logical system, called system BV, which extends multiplicative linear logic with the rules mix, nullary mix, and a self-dual, non-commutative logical operator. Due to the self-dual, noncommutative operator, system BV is of interest for applications where sequentiality plays an important role. In particular, as Bruscoli showed in [4], the non-commutative operator of BV captures precisely the sequential composition of process algebra, e.g. CCS. In fact, system BV can not be designed in the sequent calculus, as it was shown by Tiu in [17], since deep inference is crucial for deriving the provable structures of system BV. Kahramanoğulları showed, in [11], that system BV is NP-complete.

The calculus of structures also provides systems which bring insights to proof theory of different logics: in [2], Brünnler presents systems in the calculus of structures for classical logic; in [16], Straßburger presents systems for different fragments of linear logic; in [14], Stewart and Stouppa give systems for a collection of modal logics; in [18], Tiu presents a local system for intuitionistic logic. All the above mentioned systems follow a common scheme due to deep inference, which we exploit in this paper.

In the sequent calculus, because of the meta-level which causes branching while going up in the proofs, proofs are trees. However, in the calculus of structures, because meta-level of the sequent calculus is represented at the object level of the logical system [6], proofs are chains of inferences rather than trees. This observation and the applicability of the inference rules at any depth draws attention to a correspondence between the term rewriting systems [1] and systems of the calculus of structures. However, structures in a logical system are considered equivalent modulo an equational theory which makes it possible to observe the structures as equational classes of formulae with respect to the underlying equational theory of the system. Exploiting these observations, in [7], Hölldobler and Kahramanoğulları showed that systems in the calculus of structures can be expressed as term rewriting systems modulo equational theories.

Tom [13, 12] is a pattern matching preprocessor that integrates term rewriting and pattern matching facilities into imperative and functional languages such as C, Java, and Caml. In this paper, by resorting to the term rewriting features of Tom, we present a proof search implementation of system BV in Java. For this purpose, in several steps, we simulate the role played by the equational theory during proof search in the inference rules. We show that, instead of expressing commutativity and units as equalities in the underlying equational theory, role played by the equalities for unit and commutativity can be embedded into the inference rules of the system. This way, we express associativity in a list representation of the structures, and implement the inference rules as term rewriting rules which apply to terms that represent structures.

Because, of the expressive power of Java, it becomes possible to easily implement any search strategy for proof search. In our implementation, we resort to a *global search* strategy: we stack the structures which are premises of the all bottom-up instances of the inference rules with respect to a heuristic function, and proceed with applying this procedure to the topmost structure in the stack till the top-most structure is the unit. Since proofs are constructed by annihilating dual atoms, the heuristic function is chosen in a way which respects the mutual relations between dual atoms.

Because systems in the calculus of structures follow a common scheme, our implementation provides a recipe for implementing systems for other logics in the calculus of structures.

The rest of the paper is organized as follows: in Section 2, we re-collect the notions and notations of the calculus of structures and system BV. Then, in Sections 3 and 4, we remove the equalities for unit, and commutativity from the equational theory, respectively, by simulating their roles in the inference rules. After presenting some methods for reducing the nondeterminism in proof search in Section 5, we describe our implementation in Sections 6 and 7. We conclude with summary and discussions in Section 8.

2 The Calculus of Structures and System BV

In this section, we re-collect some notions and definitions of the calculus of structures and system BV, following [5].

In the language of BV atoms are denoted by $a, b, c, \ldots$ Structures are denoted by $R, S, T, \ldots$ and generated by

$$S ::= \circ \mid a \mid \langle \underbrace{S; \ldots; S}_{>0} \rangle \mid [\underbrace{S, \ldots, S}_{>0}] \mid (\underbrace{S, \ldots, S}_{>0}) \mid \overline{S} \quad ,$$

where $\circ$, the *unit*, is not an atom. $\langle S; \ldots; S\rangle$ is called a *seq structure*, $[S, \ldots, S]$ is called a *par structure*, and $(S, \ldots, S)$ is called a *copar structure*, $\overline{S}$ is the *negation* of the structure S. A structure R is called a *proper par structure* if $R = [R_1, R_2]$ where $R_1 \neq \circ$ and $R_2 \neq \circ$. Structures are considered equivalent modulo the relation $\approx$, which is the smallest congruence relation induced by the equations shown in Figure 1. There $\boldsymbol{R}$, $\boldsymbol{T}$ and $\boldsymbol{U}$ stand for finite, non-empty sequence of structures. A *structure context*, denoted as in $S\{\ \}$, is a structure with a hole that does not appear in the scope of negation. The structure R is a *substructure* of $S\{R\}$ and $S\{\ \}$ is its *context*. Context braces are omitted if no ambiguity is possible: for instance $S[R, T]$ stands for $S\{[R, T]\}$. A structure, or a structure context, is in *normal form* when the only negated structures appearing in it are atoms and no unit $\circ$ appears in it.

There is a straightforward correspondence between structures which do not involve seq structures and formulae of multiplicative linear logic (MLL) which

Associativity

$$\langle \boldsymbol{R}; \langle \boldsymbol{T} \rangle; \boldsymbol{U} \rangle \approx \langle \boldsymbol{R}; \boldsymbol{T}; \boldsymbol{U} \rangle$$
$$[\boldsymbol{R}, [\boldsymbol{T}]] \approx [\boldsymbol{R}, \boldsymbol{T}]$$
$$(\boldsymbol{R}, (\boldsymbol{T})) \approx (\boldsymbol{R}, \boldsymbol{T})$$

Context Closure

if $R = T$ then $S\{R\} = S\{T\}$ and $\bar{R} = \bar{T}$

Commutativity

$$[\boldsymbol{R}, \boldsymbol{T}] \approx [\boldsymbol{T}, \boldsymbol{R}]$$
$$(\boldsymbol{R}, \boldsymbol{T}) \approx (\boldsymbol{T}, \boldsymbol{R})$$

Units

$$\langle \circ; \boldsymbol{R} \rangle \approx \langle \boldsymbol{R}; \circ \rangle \approx \langle \boldsymbol{R} \rangle$$
$$[\circ, \boldsymbol{R}] \approx [\boldsymbol{R}]$$
$$(\circ, \boldsymbol{R}) \approx (\boldsymbol{R})$$

Negation

$$\overline{\circ} \approx \circ$$
$$\overline{\langle R; T \rangle} \approx \langle \overline{R}; \overline{T} \rangle$$
$$\overline{[R, T]} \approx (\overline{R}, \overline{T})$$
$$\overline{(R, T)} \approx [\overline{R}, \overline{T}]$$
$$\overline{\overline{R}} \approx R$$

Singleton

$$\langle R \rangle \approx [R] \approx (R) \approx R$$

Fig. 1. Equivalence relations underlying BV.

do not contain the units 1 and $\perp$. For example $[(a, b), \bar{c}, \bar{d}]$ corresponds to $((a \otimes b) \parr c^{\perp} \parr d^{\perp})$, and vice versa. Units 1 and $\perp$ are mapped into $\circ$, since $1 \equiv \perp$, when the rules mix and mix0 are added to MLL.

$$\mathsf{mix}\,\frac{\vdash \Phi \quad \vdash \Psi}{\vdash \Phi, \Psi} \qquad\qquad \mathsf{mix0}\,\frac{}{\vdash}$$

For a more detailed discussion on the proof theory of BV and the precise relation between BV and MLL, the reader is referred to [5].

In the calculus of structures, an *inference rule* is a scheme of the kind $\rho \dfrac{T}{R}$, where ρ is the *name* of the rule, T is its *premise* and R is its *conclusion*. A typical (deep) inference rule has the shape $\rho \dfrac{S\{T\}}{S\{R\}}$ and specifies the implication $T \Rightarrow R$ inside a generic context $S\{\ \}$, which is the implication being modeled in the system[4]. When premise and conclusion in an instance of an inference rule are equivalent, that instance is *trivial*, otherwise it is *non-trivial*. An inference rule is called an *axiom* if its premise is empty. Rules with empty contexts correspond to the case of the sequent calculus.

A (formal) *system* $\mathscr{S}$ is a set of inference rules. A derivation Δ in a certain formal system is a finite chain of instances of inference rules in the system. A derivation can consist of just one structure. The topmost structure in a derivation, if present, is called the *premise* of the derivation, and the bottommost structure is called its *conclusion*. A derivation Δ whose premise is T, conclusion is R, and inference rules are in $\mathscr{S}$ will be written as $\begin{matrix} T \\ \Delta \; \mathscr{S} \\ R \end{matrix}$. Similarly, $\begin{matrix} \Pi \; \mathscr{S} \\ R \end{matrix}$ will denote a *proof* Π which is a finite derivation whose topmost inference rule is an axiom. The *length* of a derivation (proof) is the number of instances of inference rules appearing in it.

[4] Due to duality between $T \Rightarrow R$ and $\bar{R} \Rightarrow \bar{T}$, rules come in pairs of dual rules: a down-version and an up-version. For instance, the dual of the $\mathsf{ai}\!\downarrow$ rule in Figure 2 is the cut rule. However, in the calculus of structures, the down rules provide sound and complete systems.

$$\circ\!\downarrow \frac{}{\circ} \qquad \mathsf{ai}\!\downarrow \frac{S\{\circ\}}{S[a,\bar{a}]} \qquad \mathsf{s}\, \frac{S([R,T],U)}{S[(R,U),T]} \qquad \mathsf{q}\!\downarrow \frac{S\langle [R,U];[T,V]\rangle}{S[\langle R;T\rangle,\langle U;V\rangle]}$$

Fig. 2. System BV

Two systems $\mathscr{S}$ and $\mathscr{S}'$ are *equivalent* if for every proof of a structure T in system $\mathscr{S}$, there exists a proof of T in system $\mathscr{S}'$, and vice versa.

The system $\{\circ\!\downarrow, \mathsf{ai}\!\downarrow, \mathsf{s}, \mathsf{q}\!\downarrow\}$, shown in Figure 2, is denoted by BV, and called *basic system* V. The rules of the system are called *unit* ($\circ\!\downarrow$), *atomic interaction* ($\mathsf{ai}\!\downarrow$), *switch* (s) and *seq* ($\mathsf{q}\!\downarrow$).

3 Removing the Equalities for Unit

In this section, we present a system equivalent to system BV where the application of inference rules is explicit with respect to equalities for unit. We assume that these rules are applied to structures which are in normal form. However, this is not restrictive since a normal form of a structure can be equivalently obtained by applying the terminating and confluent term rewriting system resulting from orienting the equalities for negation and unit in Figure 1 from left to right [7]. Hence, the equalities for unit and negation can be equivalently removed from the underlying equational theory by considering only those structures that are in normal form.

$$ax\, \frac{}{[a,\bar{a}]} \qquad \mathsf{s}_1\, \frac{S([R,W],T)}{S[(R,T),W]}$$

$$\mathsf{ai}_1\!\downarrow \frac{S\{R\}}{S[R,[a,\bar{a}]]} \qquad \mathsf{ai}_2\!\downarrow \frac{S\{R\}}{S(R,[a,\bar{a}])} \qquad \mathsf{ai}_3\!\downarrow \frac{S\{R\}}{S\langle R;[a,\bar{a}]\rangle} \qquad \mathsf{ai}_4\!\downarrow \frac{S\{R\}}{S\langle [a,\bar{a}];R\rangle}$$

$$\mathsf{q}_1\!\downarrow \frac{S\langle [R,T];[U,V]\rangle}{S[\langle R;U\rangle,\langle T;V\rangle]} \qquad \mathsf{q}_2\!\downarrow \frac{S\langle R;T\rangle}{S[R,T]} \qquad \mathsf{q}_3\!\downarrow \frac{S\langle [W,T];U\rangle}{S[W,\langle T;U\rangle]} \qquad \mathsf{q}_4\!\downarrow \frac{S\langle T;[W,U]\rangle}{S[W,\langle T;U\rangle]}$$

Fig. 3. System BVu

Definition 1. *The system in Figure 3 is called* unit-free BV *or* BVu. *The equalities for unit do not apply to system* BVu.

Theorem 1. *[9] System* BV *and system* BVu *are equivalent.*

The inference rules of system BVu allow the unit to be completely removed from the language of the BV structures. Furthermore, trivial application of inference rules are not possible in system BVu.

4 Removing the Equalities for Commutativity

In this section, we will remove the equalities for commutativity from the equational theory underlying system BVu by making the role played by these equalities explicit in the inference rules. We first need some modifications on the inference rules:

Definition 2. *We put the following restriction on system* BVu*: The structures W in the rules are restricted to atoms, copar structures and seq structures. In other words, structure W is not a proper par structure. We will call this system* unit-free lazy BV *or* BVul.

Proposition 1. *System* BV *and system* BVul *are equivalent.*

Proof: Observe that the rules in BVul are instances of the rules in BVu with restrictions on switch and seq rules. The case where W is a proper par structure is derivable in BVul: the case of the rule $\mathsf{q}_4\!\downarrow$ being analogous to the case for the rule $\mathsf{q}_3\!\downarrow$, for the rules s, and $\mathsf{q}_3\!\downarrow$ take the following derivations:

$$\mathsf{s}_1\,\dfrac{S([R,T,V],U)}{\mathsf{s}_1\,\dfrac{S[([R,U],T),V]}{=\,\dfrac{S[[(R,T),U],V]}{S[(R,T),[U,V]]}}} \qquad \mathsf{q}_3\!\downarrow\dfrac{S\langle[R,V,T];U\rangle}{\mathsf{q}_3\!\downarrow\dfrac{S[R,\langle[V,T];U\rangle]}{=\,\dfrac{S[R,[V,\langle T;U\rangle]]}{S[[R,V],\langle T;U\rangle]}}}$$

Definition 3. *The system in Figure 4 is called* commutativity-free BV *or* BVc*, where W is either an atom or a copar structure or a seq structure, and the equalities for unit and commutativity do not apply to* BVc.

Proposition 2. *System* BV *and system* BVc *are equivalent.*

Proof: Inference rules of BVc are instances of the inference rules of BV. The proof of the other direction is by inductive case analysis on the commutative application of the inference rules of BVul: let Π be the proof of R in BVul. By induction on Π, we construct a proof Π' of R in BVc.

- If Π is $ax\,\dfrac{}{[a,\bar{a}]}$, take the same rule in BVc. (observe that $ax\,\dfrac{}{[\bar{a},a]}$ is an instance of this rule, also when commutativity does not apply, since $\bar{a}$ is an atom, and $\bar{\bar{a}} = a$.)
- If $\mathsf{ai}_1\!\downarrow$ is the last rule applied in Π, such that

$$\mathsf{ai}_1\!\downarrow\dfrac{S\{R\}}{=\,\dfrac{S\{R,[a,\bar{a}]\}}{Q}}$$

, there are the following possibilities for Q : If

  - $Q = S[R,a,\bar{a}]$; take $\mathsf{ai}_{11}\!\downarrow$.
  - $Q = S[a,\bar{a},R]$; take $\mathsf{ai}_{12}\!\downarrow$.
  - $Q = S[a,R,\bar{a}]$; take $\mathsf{ai}_{13}\!\downarrow$.
  - $Q = S(R,[a,\bar{a}])$; take $\mathsf{ai}_{21}\!\downarrow$.
  - $Q = S([a,\bar{a}],R)$; take $\mathsf{ai}_{22}\!\downarrow$.
  - $Q = S\langle R;[a,\bar{a}]\rangle$; take $\mathsf{ai}_{3}\!\downarrow$.

- $Q = S\langle [a,\bar{a}]; R\rangle$; take $\mathsf{ai}_4\!\downarrow$.

– If s_1 is the last rule applied in Π, such that

$$\mathsf{s}_1 \frac{S([R,W],T)}{\dfrac{S[(R,T),W]}{Q}{=}} \quad , \quad \text{there are the following possibilities for } Q : \text{If}$$

- $Q = S[(R,T),W]$; take s_{11a}.
- $Q = S[(T,R),W]$; take s_{12a}.
- $Q = S[W,(R,T)]$; take s_{13a}.
- $Q = S[W,(T,R)]$; take s_{14a}.
- $Q = S[(R,T,U),W]$; take s_{15a}.
- $Q = S[W,(R,T,U)]$; take s_{16a}.
- $Q = S'[(R,T),P,W]$ such that $S\{\ \} = S'[\{\ \},P]$; take s_{11b}.
- $Q = S'[(T,R),P,W]$ such that $S\{\ \} = S'[\{\ \},P]$; take s_{12b}.
- $Q = S'[W,P,(R,T)]$ such that $S\{\ \} = S'[\{\ \},P]$; take s_{13b}.
- $Q = S'[W,P,(T,R)]$ such that $S\{\ \} = S'[\{\ \},P]$; take s_{14b}.
- $Q = S'[(R,T,U),P,W]$ such that $S\{\ \} = S'[\{\ \},P]$; take s_{15b}.
- $Q = S'[W,P,(R,T,U)]$ such that $S\{\ \} = S'[\{\ \},P]$; take s_{16b}.

– If $\mathsf{q}_1\!\downarrow$ is the last rule applied in Π, such that

$$\mathsf{q}_1\!\downarrow \frac{S\langle [R,T];[U,V]\rangle}{\dfrac{S[\langle R;U\rangle,\langle T;V\rangle]}{Q}{=}} \quad , \quad \text{there are the following possibilities for } Q : \text{If}$$

- $Q = S[\langle R;U\rangle,\langle T;V\rangle]$; take $\mathsf{q}_{11}\!\downarrow$.
- $Q = S'[\langle R;U\rangle,P,\langle T;V\rangle]$ such that $S\{\ \} = S'[\{\ \},P]$; take $\mathsf{q}_{12}\!\downarrow$.

– If $\mathsf{q}_2\!\downarrow$ is the last rule applied in Π, such that

$$\mathsf{q}_2\!\downarrow \frac{S\langle R;T\rangle}{\dfrac{S[R,T]}{Q}{=}} \quad , \quad \text{there are the following possibilities for } Q : \text{If}$$

- $Q = S[R,T]$; take $\mathsf{q}_{21}\!\downarrow$.
- $Q = S[T,R]$; take $\mathsf{q}_{22}\!\downarrow$.
- $Q = S'[R,P,T]$ such that $S\{\ \} = S'[\{\ \},P]$; take $\mathsf{q}_{23}\!\downarrow$.
- $Q = S'[T,P,R]$ such that $S\{\ \} = S'[\{\ \},P]$; take $\mathsf{q}_{24}\!\downarrow$.

– If $\mathsf{q}_3\downarrow$ is the last rule applied in Π, such that

$$\mathsf{q}_3\downarrow \frac{\dfrac{S\langle [W,T];U\rangle}{S[W,\langle T;U\rangle]}}{Q}= \quad ,$$ there are the following possibilities for Q : If

- $Q = S[W, \langle T;U\rangle]$; take $\mathsf{q}_{31}\downarrow$.
- $Q = S[\langle T;U\rangle, W]$; take $\mathsf{q}_{32}\downarrow$.
- $Q = S'[W, P, \langle T;U\rangle]$ such that $S\{\ \} = S'[\{\ \}, P]$; take $\mathsf{q}_{33}\downarrow$.
- $Q = S'[\langle T;U\rangle, P, W]$ such that $S\{\ \} = S'[\{\ \}, P]$; take $\mathsf{q}_{34}\downarrow$.

– If $\mathsf{q}_4\downarrow$ is the last rule applied in Π, such that

$$\mathsf{q}_4\downarrow \frac{\dfrac{S\langle T;[W,U]\rangle}{S[W,\langle T;U\rangle]}}{Q}= \quad ,$$ there are the following possibilities for Q : If

- $Q = S[W, \langle T;U\rangle]$; take $\mathsf{q}_{41}\downarrow$.
- $Q = S[\langle T;U\rangle, W]$; take $\mathsf{q}_{42}\downarrow$.
- $Q = S'[W, P, \langle T;U\rangle]$ such that $S\{\ \} = S'[\{\ \}, P]$; take $\mathsf{q}_{43}\downarrow$.
- $Q = S'[\langle T;U\rangle, P, W]$ such that $S\{\ \} = S'[\{\ \}, P]$; take $\mathsf{q}_{44}\downarrow$.

5 Reducing the Nondeterminism

In a proof search episode, inference rules can be applied to a structure in many different ways, however only few of these applications can lead to a proof. For example, to the structure $[(a,b),\bar{a},\bar{b}]$ switch rule can be applied bottom-up in twelve different ways, but only two of these instances can lead to a proof. With the below definition, we will redesign the inference rules such that the instances of the inference rules which do not provide a proof will not be possible. For an extensive exposure to these ideas the reader is referred to [10].

Definition 4. *Given a structure S, the notation* $\mathsf{at}\,S$ *indicates the set of all the atoms appearing in S. Let* lazy interaction switch *be the rule*

$$\mathsf{lis}\,\frac{S([R,W],T)}{S[(R,T),W]} \quad ,$$

where structure W is not a proper par structure and $\mathsf{at}\,\overline{W} \cap \mathsf{at}\,R \neq \varnothing$*. The following rules are called* interaction seq rule 1*,* lazy interaction seq rule 3*, and* lazy interaction seq rule 4*, respectively,*

$$\mathsf{iq}_1\downarrow\frac{S\langle [R,T];[U,V]\rangle}{S[\langle R;U\rangle,\langle T;V\rangle]} \qquad \mathsf{liq}_3\downarrow\frac{S\langle [R,W];T\rangle}{S[W,\langle R;T\rangle]} \qquad \mathsf{liq}_4\downarrow\frac{S\langle T;[R,W]\rangle}{S[W,\langle T;R\rangle]}$$

where structure W is not a proper par structure and, in $\mathsf{iq}_1\!\downarrow$, $\mathsf{at}\,\overline{R} \cap \mathsf{at}\,T \neq \varnothing$ *and* $\mathsf{at}\,\overline{U} \cap \mathsf{at}\,V \neq \varnothing$*; in* $\mathsf{liq}_3\!\downarrow$ *and in* $\mathsf{liq}_4\!\downarrow$, $\mathsf{at}\,\overline{R} \cap \mathsf{at}\,W \neq \varnothing$*. The system resulting from replacing the rules* s_1, $\mathsf{q}_1\!\downarrow$, $\mathsf{q}_3\!\downarrow$, *and* $\mathsf{q}_4\!\downarrow$, *in* BVu *with the rule* lis, $\mathsf{iq}_1\!\downarrow$, $\mathsf{q}_2\!\downarrow$, $\mathsf{liq}_3\!\downarrow$, *and* $\mathsf{liq}_4\!\downarrow$ *is called* interaction system BV, *or* BVi.

Theorem 2. *[10] System* BV *and system* BVi *are equivalent.*

With the below definition, we will combine the ideas from systems BVc and BVi in a single system, that is, we will impose the restrictions on the rules of BVi analogously on the inference rules of system BVc. This way, we will obtain a system where the equalities for unit and commutativity are redundant, and non-determinism is reduced.

Definition 5. *Let* commutativity-free interaction system BV *or system* BVci *be the system obtained by imposing the following restrictions on system* BVc*: in the rules* $\mathsf{s}_{11a}, \mathsf{s}_{12a}, \mathsf{s}_{13a}, \mathsf{s}_{14a}, \mathsf{s}_{11b}, \mathsf{s}_{12b}, \mathsf{s}_{13b}, \mathsf{s}_{14b}$ *we have* $\mathsf{at}\,\overline{R} \cap \mathsf{at}\,W \neq \varnothing$*; in the rules* $\mathsf{s}_{15a}, \mathsf{s}_{16a}, \mathsf{s}_{15b}, \mathsf{s}_{16b}$ *we have* $\mathsf{at}\,\overline{(R,U)} \cap \mathsf{at}\,W \neq \varnothing$*; in the rules* $\mathsf{q}_{11}\!\downarrow, \mathsf{q}_{12}\!\downarrow$ *we have* $\mathsf{at}\,\overline{R} \cap \mathsf{at}\,T \neq \varnothing$ *and* $\mathsf{at}\,\overline{U} \cap \mathsf{at}\,V \neq \varnothing$*; in the rules* $\mathsf{q}_{31}\!\downarrow, \mathsf{q}_{32}\!\downarrow, \mathsf{q}_{33}\!\downarrow, \mathsf{q}_{34}\!\downarrow$ *we have* $\mathsf{at}\,\overline{W} \cap \mathsf{at}\,T \neq \varnothing$*; in the rules* $\mathsf{q}_{41}\!\downarrow, \mathsf{q}_{42}\!\downarrow, \mathsf{q}_{43}\!\downarrow, \mathsf{q}_{44}\!\downarrow$ *we have* $\mathsf{at}\,\overline{W} \cap \mathsf{at}\,U \neq \varnothing$.

Theorem 3. *[10] System* BV *and system* BVci *are equivalent.*

Proof: Follows immediately from Proposition 2 and Theorem 2.

6 From Inference Rules to Term Rewriting Rules

The systems in the calculus of structures can be represented as term rewriting systems modulo equational theories.[5] In such a representation, the notion of a structure is replaced by a notion of term, considering terms over variables. Thus, bottom up application of an inference rule is represented as a rewriting rule that rewrites the conclusion to the premise of the inference rule. Similarly, inference rules with conditions are represented as conditional rewriting rules. For instance, consider the following rewrite rules for the inference rules switch and interaction seq rule 1, respectively:

$$\begin{array}{lcl}
\mathsf{s} & : & [(R,T),U] \quad \rightarrow \quad ((R,U),T) \\
\mathsf{iq}_1\!\downarrow & : & [\langle R;U\rangle,\langle T;V\rangle] \quad \rightarrow \quad \langle [R,T];[U,V]\rangle \quad \text{if} \quad \mathsf{at}\,\overline{R} \cap \mathsf{at}\,T \;\wedge\; \mathsf{at}\,\overline{U} \cap \mathsf{at}\,V
\end{array}$$

Such rewrite rules are applied modulo the equational theory underlying the proof theoretical system. Because we use structures as terms the equalities for context closure and singleton become redundant. This leaves us with only equalities for associativity for system BVci when expressed as term rewriting system. In the next section, by resorting to a list representation of n-ary terms which captures associativity, we will present an implementation of the term rewriting system for system BVci.

[5] For an indepth exposure on the correspondence between systems of the calculus of structures and term rewriting systems the reader is referred to [7].

7 Implementation in TOM

Tom is a language extension which adds syntactic and associative pattern-matching facilities to existing languages like Java, C, and OCaml. This hybrid approach is particularly well-suited when describing transformations of structured entities like trees/terms and XML documents, for example. In this work, we use Tom, combined with Java, to implement our prototype.

An interesting feature of the language is to provide support for matching modulo sophisticated theories. In particular, pattern-matching modulo associativity and neutral element (also known as list-matching) is both useful and efficient to model the exploration of a search space.

For expository reasons, we assume that Tom only adds two new constructs: `%match` and *back-quote* (`). The first construct is similar to the `match` primitive of ML and related languages: given a term (called subject) and a list of pairs pattern-action, the `match` primitive selects a pattern that matches the subject and performs the associated action. The second construct is a mechanism that allows one to easily build ground terms over a defined signature. This operator, called *back-quote*, is followed by a well-formed term, written in prefix notation.

A main originality of this system is to be data-structure independent. This means that a *mapping* has to be defined to connect algebraic data-structure, on which pattern matching is performed, to low-level data-structures, that correspond to the implementation. Most of the time, Tom is used in conjunction with the ApiGen system [19], which generates abstract syntax tree implementations and a mapping, from a given datatype definition. The input format for ApiGen is a concise language defining sorts and constructors for the abstract syntax. The output is an efficient, in time and memory, Java implementation for this datatype. This implementation is characterized by strong typing and maximal sub-term sharing, providing both memory efficiency and constant-time equality checking.

For an interested reader, design and implementation issues related to Tom are presented in [13, 12].

7.1 Data structures

A main difficulty, when implementing the systems of the calculus of structures, is to find a good representation for the *par*, *cop*, and *seq* structures ($[R, T]$, (R, T) and $\langle R; T\rangle$). In our implementation of BVci, we considered these constructors as unary operators which take a *list of structures* as argument. Using ApiGen, the considered data-type can be described by the following signature:

```
module Struct
  public sorts Struc StrucPar StrucCop StrucSeq
  abstract syntax
    a -> Struc
    b -> Struc
    ...other atom constants
    neg(Struc)         -> Struc
    par(StrucPar)      -> Struc
    cop(StrucCop)      -> Struc
    seq(StrucSeq)      -> Struc
    concPar( Struc* ) -> StrucPar
    concCop( Struc* ) -> StrucCop
    concSeq( Struc* ) -> StrucSeq
```

The grammar rule `par(StrucPar) -> Struc` defines a unary operator `par` of sort `Struct` which takes a `StrucPar` as unique argument. The grammar rule `concPar(Struc*) -> StrucPar` defines the `concPar` operator of sort `StrucPar`. The special syntax `Struc*` indicates that `concPar` is a *list-operator* which takes a list of `Struc` as argument. Thus, by combining `par` and `concPar` it becomes possible to represent the structure $[a,[b,c]]$ by `par(concPar(a,b,c))`. Note that structures are flattened. In Tom, list-operators are interesting because their arity is not fixed. Thus, `concPar(a,b,c)` corresponds to a list of 3 elements, `concPar(a)` corresponds to a list of single element, namely `a`, whereas `concPar()` denotes the empty list. (R,T) and $\langle R;T\rangle$ are represented in a similar way, using `cop, seq`, `concCop`, and `concSeq`.

A problem with this approach is that we can manipulate objects, like `par(concPar())`, which do not necessarily correspond to intended structures. It is also possible to have several representations for the same structure. Hence, `par(concPar(a))` and `cop(concCop(a))` both denote the structure `a`. To avoid such situations, we have encoded, in the defined mapping, a notion of canonical form which avoids building uninteresting terms. Thus, we ensure that

– $[]$, $\langle\rangle$ and $()$ are reduced when containing only one sub-structure:
  $par(concPar(x)) \rightarrow x$
– nested structures are flattened:
  $par(concPar(...,par(concPar(x_1,...,x_n)),...) \rightarrow par(concPar(...,x_1,...,x_n,...)$
– subterms are sorted (according to a given total lexical order $<$):
  $concPar(...,x_i,...,x_j,...) \rightarrow concPar(...,x_j,...,x_i,...)$ if $x_j < x_i$.

This notion of canonical form allows us to efficiently check if two terms represents the same structure with respect to commutativity of those connectors.

7.2 Rewrite rules

The rewrite rules define the deduction steps in system BVci. They are implemented by a *match* construct which matches a sub-term with the left-hand side of the rewrite rule. Then the right-hand side of the rule builds the deduced structure.

For instance, the rules $[(R,T),U] \rightarrow ([R,U],T)$ and $[(R,T),U] \rightarrow ([T,U],R)$ are encoded by the following match construct.

```
%match(Struc t) {
  par(concPar(X1*,cop(concCop(R*,T*)),X2*,U,X3*)) -> {
    if(‘T*.isEmpty() || ‘R*.isEmpty() ) {
    } else {
      StrucPar context = ‘concPar(X1*,X2*,X3*);
      if(canReact(‘R*,‘U)) {
        StrucPar parR = cop2par(‘R*);
                            // transform a StrucCop into a StrucPar
        Struc elt1 = ‘par(concPar(
                cop(concCop(par(concPar(parR*,U)),T*)),context*));
        c.add(elt1);
      }
      if(canReact(‘T*,‘U)) {
        StrucPar parT = cop2par(‘T*);
        Struc elt2 = ‘par(concPar(
                cop(concCop(par(concPar(parT*,U)),R*)),context*));
        c.add(elt2);
} } } }
```

We ensure that we do not execute the right-hand side of the rule if either `R` or `T` are empty lists. The other tests implements the restrictions on the application of the rules detailed in Section 5 for reducing the non-determinism. This is done by using an auxiliary predicate function `canReact(a,b)` which collects all atoms in structures `a` and `b` and returns `true` only if `a` contains at least one atom which is contained in a negated form in `b`. This function can be made efficient by using the features of the host language of Tom, in our case, by using an efficient hash-set implementation in Java. The remaining rules are expressed in a similar way.

7.3 Strategy

When designing a proof search procedure, implementing the set of inference rules is very important, but this is only one part of the job. The second part consists in defining a strategy which describes how to apply the rules. In rule based systems like ELAN or MAUDE, it is very easy to describe such strategies, using primitive operators or meta-level capabilities. However, in some cases, it may be difficult to express strategies which take time and space into consideration. In ELAN for example, the search is implemented using a backtracking mechanism. This is a good approach to implement depth-first search strategies. However, while efficient in space, such a strategy may lead to explore infinite branches and non-terminating programs. On the other hand, breadth-first search, as in MAUDE, usually terminates when a proof exists, but the memory needed can be considerably huge. In languages like ELAN, the backtracking based mechanism makes the definition of strategies difficult.

In Tom, there is no particular support for implementing search space exploration strategies. Thus, the search space has to be handled explicitly. On one hand, this leads to more complex implementations, but on the other, this allows us to define very fine and efficient search strategies.

In our implementation, we have implemented a *global search* strategy which combines the advantages of both depth- and breadth-first search strategies: given an ordered list of elements, we select the first term and compute the set of its successors by applying all rules at every position. Implementing a breadth-first search strategy can be done by adding this resulting set of elements at the end of the list. To implement a depth-first search strategy, the set has to be inserted at the beginning of the list. In our case, the elements of the set are inserted and inter-mixed in the initial list, according to the given order. In one sense, the order implements a heuristic which characterizes the *interesting* structures that have to be explored first, since they may lead to the proof in an efficient way. This mechanism is iterated until the main list contains the unit element.

The order used for those lists is the main parameter of the method, and can be changed at will to find a suitable order for fast proof finding.

8 Summary and Discussions:

We have presented a proof search implementation of system BV of the calculus of structures by resorting to the correspondence between the systems of the calculus of structures and term rewriting systems modulo equational theories. The term rewriting rules corresponding to inference rules of system BV are applied modulo an equational theory which admits associativity, commutativity and a unit for different logical operators. By making the role played by the equalities for unit and commutativity in the application of the inference rules explicit, we presented a system equivalent to system BV where these equalities become redundant. This way, we expressed associativity in a list representation.

Our implementation, in Java, uses the pattern matching preprocessor TOM in order to integrate term rewriting features into Java. By exploiting the expressive power due to Java, we provided a search algorithm which combines different search strategies and heuristic search. The source code of the implementation is available at the TOM distribution[6]. A representative applet of this implementation can be found at `http://tom.loria.fr/examples/structures/BV.html`.

Our implementation respects a common scheme which is shared by all the systems of the calculus of structures for classical logic, modal logics, and linear logic. For this reason, our implementation can be easily generalized for implementing different tools for these systems, also by employing different search strategies at will.

In [8], Kahramanoğulları presents an implementation of system BV in MAUDE by using the *search* function of this system which implements breadth-first search on the search space of term rewriting systems modulo equational theories. Although this implementation benefits from the simple high-level language of MAUDE, implementation of a certain strategy, different from breadth-first search, remains complicated due to the complex meta-level language. However, in the implementation presented in this paper, the availability of the Java language provides a great ease on implementing different search strategies.

[6] `http://tom.loria.fr`

References

1. Franz Baader and Tobias Nipkow. *Term Rewriting and All That*, volume 1. Cambridge University Press, 1998.
2. Kai Brünnler. *Deep Inference and Symmetry in Classical Proofs*. PhD thesis, Technische Universität Dresden, 2003.
3. Kai Brünnler and Alwen Fernanto Tiu. A local system for classical logic. In R. Nieuwenhuis and A. Voronkov, editors, *LPAR 2001*, volume 2250 of *Lecture Notes in Artificial Intelligence*, pages 347–361. Springer-Verlag, 2001.
4. Paola Bruscoli. A purely logical account of sequentiality in proof search. In Peter J. Stuckey, editor, *Logic Programming, 18th International Conference*, volume 2401 of *Lecture Notes in Computer Science*, pages 302–316. Springer-Verlag, 2002.
5. Alessio Guglielmi. A system of interaction and structure. Technical Report WV-02-10, TU Dresden, 2002. To app. in ACM Transactions on Computational Logic.
6. Alessio Guglielmi. Mismatch. Available on the web at http://iccl.tu-dresden.de/~guglielm/p/AG9.pdf, 2003.
7. Steffen Hölldobler and Ozan Kahramanoğulları. From the calculus of structures to term rewriting systems. Technical Report WV-04-03, TU Dresden, 2004.
8. Ozan Kahramanoğulları. Implementing system BV of the calculus of structures in maude. In L. A. i Alemany and P. Égré, editors, *Proceedings of the ESSLLI-2004 Student Session*, pages 117–127, Université Henri Poincaré, Nancy, France, 2004.
9. Ozan Kahramanoğulları. System BV without the equalities for unit. In C. Aykanat, T. Dayar, and I. Körpeoğlu, editors, *Proc. of the 19th Int. Symp. on Computer and Information Sciences, ISCIS'04*, volume 3280 of *LNCS*. Springer, 2004.
10. Ozan Kahramanoğulları. Reducing the non-determinism in the calculus of structures. Technical report, TU Dresden, 2005. Available at http://www.informatik.uni-leipzig.de/~ozan/reducingNondet.pdf.
11. Ozan Kahramanoğulları. System BV is NP-complete. to appear in Proceedings of WoLLIC'05, http://www.informatik.uni-leipzig.de/~ozan/BVnpc.pdf, 2005.
12. Claude Kirchner, Pierre-Etienne Moreau, and Antoine Reilles. Formal validation of pattern matching code. 2005. To appear in PPDP '05: Proceedings of the 7th ACM SIGPLAN Int. Conf. on Principles and practice of declarative programming.
13. Pierre-Etienne Moreau, Christophe Ringeissen, and Marian Vittek. A Pattern Matching Compiler for Multiple Target Languages. In G. Hedin, editor, *12th Conference on Compiler Construction, Warsaw (Poland)*, volume 2622 of *LNCS*, pages 61–76. Springer-Verlag, May 2003.
14. Charles Stewart and Phiniki Stouppa. A systematic proof theory for several modal logics. Technical Report WV-03-08, TU Dresden, 2003. Accepted at Advances in Modal Logic 2004, to app. in proceedings published by King's College Publications.
15. Lutz Straßburger. A local system for linear logic. In Matthias Baaz and Andrei Voronkov, editors, *Logic for Programming, Artificial Intelligence, and Reasoning, LPAR 2002*, volume 2514 of *LNAI*, pages 388–402. Springer-Verlag, 2002.
16. Lutz Straßburger. *Linear Logic and Noncommutativity in the Calculus of Structures*. PhD thesis, TU Dresden, 2003.
17. Alwen Fernanto Tiu. Properties of a logical system in the calculus of structures. Technical Report WV-01-06, Technische Universität Dresden, 2001.
18. Alwen Fernanto Tiu. A local system for intuitionistic logic: Preliminary results. http://www.loria.fr/~tiu/localint.pdf, 2005.
19. Mark van den Brand, Pierre-Etienne Moreau, and Jurgen Vinju. A generator of efficient strongly typed abstract syntax trees in java. Technical report SEN-E0306, ISSN 1386-369X, CWI, Amsterdam (Holland), November 2003.

$$ax\,\frac{}{[a,\bar{a}]}$$

$$\mathsf{ai}_{11}{\downarrow}\,\frac{S\{R\}}{S[R,a,\bar{a}]}\qquad \mathsf{ai}_{12}{\downarrow}\,\frac{S\{R\}}{S[a,\bar{a},R]}\qquad \mathsf{ai}_{13}{\downarrow}\,\frac{S\{R\}}{S[a,R,\bar{a}]}$$

$$\mathsf{ai}_{21}{\downarrow}\,\frac{S\{R\}}{S(R,[a,\bar{a}])}\qquad \mathsf{ai}_{22}{\downarrow}\,\frac{S\{R\}}{S([a,\bar{a}],R)}\qquad \mathsf{ai}_{3}{\downarrow}\,\frac{S\{R\}}{S\langle R;[a,\bar{a}]\rangle}\qquad \mathsf{ai}_{4}{\downarrow}\,\frac{S\{R\}}{S\langle [a,\bar{a}];R\rangle}$$

$$\mathsf{s}_{11a}\,\frac{S([R,W],T)}{S[(R,T),W]}\qquad \mathsf{s}_{12a}\,\frac{S([R,W],T)}{S[(T,R),W]}\qquad \mathsf{s}_{13a}\,\frac{S([R,W],T)}{S[W,(R,T)]}\qquad \mathsf{s}_{14a}\,\frac{S([R,W],T)}{S[W,(T,R)]}$$

$$\mathsf{s}_{15a}\,\frac{S([(R,U),W],T)}{S[(R,T,U),W]}\qquad \mathsf{s}_{16a}\,\frac{S([(R,U),W],T)}{S[W,(R,T,U)]}$$

$$\mathsf{s}_{11b}\,\frac{S[([R,W],T),P]}{S[(R,T),P,W]}\qquad \mathsf{s}_{12b}\,\frac{S[([R,W],T),P]}{S[(T,R),P,W]}$$

$$\mathsf{s}_{13b}\,\frac{S[([R,W],T),P]}{S[W,P,(R,T)]}\qquad \mathsf{s}_{14b}\,\frac{S[([R,W],T),P]}{S[W,P,(T,R)]}$$

$$\mathsf{s}_{15b}\,\frac{S[([(R,U),W],T),P]}{S[(R,T,U),P,W]}\qquad \mathsf{s}_{16b}\,\frac{S[([(R,U),W],T),P]}{S[W,P,(R,T,U)]}$$

$$\mathsf{q}_{11}{\downarrow}\,\frac{S\langle [R,T];[U,V]\rangle}{S[\langle R;U\rangle,\langle T;V\rangle]}\qquad \mathsf{q}_{12}{\downarrow}\,\frac{S[\langle [R,T];[U,V]\rangle,P]}{S[\langle R;U\rangle,P,\langle T;V\rangle]}$$

$$\mathsf{q}_{21}{\downarrow}\,\frac{S\langle R;T\rangle}{S[R,T]}\qquad \mathsf{q}_{22}{\downarrow}\,\frac{S\langle R;T\rangle}{S[T,R]}\qquad \mathsf{q}_{23}{\downarrow}\,\frac{S[\langle R;T\rangle,P]}{S[R,P,T]}\qquad \mathsf{q}_{24}{\downarrow}\,\frac{S[\langle R;T\rangle,P]}{S[T,P,R]}$$

$$\mathsf{q}_{31}{\downarrow}\,\frac{S\langle [W,T];U\rangle}{S[W,\langle T;U\rangle]}\quad \mathsf{q}_{32}{\downarrow}\,\frac{S\langle [W,T];U\rangle}{S[\langle T;U\rangle,W]}\quad \mathsf{q}_{33}{\downarrow}\,\frac{S[\langle [W,T];U\rangle,P]}{S[W,P,\langle T;U\rangle]}\quad \mathsf{q}_{34}{\downarrow}\,\frac{S[\langle [W,T];U\rangle,P]}{S[\langle T;U\rangle,P,W]}$$

$$\mathsf{q}_{41}{\downarrow}\,\frac{S\langle T;[W,U]\rangle}{S[W,\langle T;U\rangle]}\quad \mathsf{q}_{42}{\downarrow}\,\frac{S\langle T;[W,U]\rangle}{S[\langle T;U\rangle,W]}\quad \mathsf{q}_{43}{\downarrow}\,\frac{S[\langle T;[W,U]\rangle,P]}{S[W,P,\langle T;U\rangle]}\quad \mathsf{q}_{44}{\downarrow}\,\frac{S[\langle T;[W,U]\rangle,P]}{S[\langle T;U\rangle,P,W]}$$

Fig. 4. System BVc