Available online at www.sciencedirect.com
ScienceDirect
Electronic Notes in Theoretical Computer Science
ELSEVIER
Electronic Notes in Theoretical Computer Science 219 (2008) 35–50
www.elsevier.com/locate/entcs

Maude as a Platform for Designing and Implementing Deep Inference Systems

Ozan Kahramanoğulları [1,2]

Department of Computing, Imperial College London

Abstract

Deep inference is a proof theoretical methodology that generalizes the traditional notion of inference in the sequent calculus: in contrast to the sequent calculus, the deductive systems with deep inference do not rely on the notion of main connective, and permit the application of the inference rules at any depth inside logical expressions, in a way which resembles the application of term rewriting rules. Deep inference provides a richer combinatoric analysis of proofs for different logics. In particular, construction of exponentially shorter proofs becomes possible. In this paper, aiming at the development of computation as proof search tools, we propose the Maude language as a means for designing and implementing different deep inference deductive systems and proof strategies that work on these systems. We give Maude implementations of deep inference systems together with an implementation that simulates sequent calculus proofs to serve as a benchmark. We demonstrate these ideas on classical logic, and argue that they can be analogously carried to other deep inference systems for other logics, as well as sequent calculus systems.

Keywords: deep inference, proof search, Maude, term rewriting

1 Introduction

In recent years, automated proof search has started to find broader applications, especially in the fields of automated theorem proving and software verification. In this regard, development of formalisms and tools that allow the construction of shorter analytic proofs is gaining more and more importance.

Deep inference is a proof theoretical methodology that generalizes the traditional notion of inference of the sequent calculus. In contrast to the sequent calculus, the deductive systems with deep inference do not rely on the notion of main connective and permit the application of the inference rules at any depth inside logical expressions, similar to the application of term rewriting rules.

[1] The author would like to thank Steven Eker, Kai Brünnler and the anonymous referees for their comments and suggestions.

[2] Email: ozank@doc.ic.ac.uk

doi:10.1016/j.entcs.2008.10.033

Deep inference has originally emerged as a means to conceive the logical system BV [7]. System BV is a conservative extension of multiplicative linear logic and it admits a self-dual non-commutative logical operator resembling the operators for sequential composition in process algebras. Although multiplicative linear logic is often represented as a sequent calculus deductive system, it is not possible to design system BV in a standard sequent calculus [20]. A notion of deep rewriting is necessary for deriving all the provable formulae of system BV.

Deep inference also provides deductive systems which bring new insights to the proof theory of other logics. The applicability of inference rules at arbitrary depths inside logical expressions brings about a rich combinatoric analysis of proofs, which previously has not been available by means of traditional approaches to proof theory: In [2], Brünnler presents deep inference systems for classical logic; in [18], Straßburger presents systems for different fragments of linear logic. In [16,17], Stewart and Stouppa give systems for a class of modal logics. Tiu presents, in [19], a local system for intuitionistic logic. All these systems follow a common scheme of inference rules which enjoys a rich proof theory.

Availability of deep inference provides shorter proofs than in the sequent calculus. For example, there is a class of theorems, called the *Statman's tautologies*, for which the size of proofs in the sequent calculus grows exponentially over the size of the theorems. However, over the same class, there are deep inference proofs that grow polynomially [3]. This is because applicability of the inference rules at any depth inside a formula makes it possible to start the construction of a proof by manipulating and annihilating sub-formulae without any prior branching. However, because inference rules can be applied in many more ways, nondeterminism in proof search is much greater than in the sequent calculus and the breadth of the search space grows rather quickly during proof search. In this respect, development of new techniques for reducing nondeterminism in proof search without sacrificing from proof theoretic cleanliness gains importance.

The language Maude [4,5] allows implementing term rewriting systems modulo equational theories due to the very fast matching algorithm that supports different combinations of associative, commutative theories, also with the presence of units. Furthermore, Maude allows to integrate conditional rules, equational, and meta-level reasoning in the modules. Exploiting these features, in this paper we propose the language Maude as a platform for designing and implementing deep inference systems where proof theoretic techniques for reducing nondeterminism [11] can be tested and further developed. We give Maude implementations of deep inference systems together with an implementation that simulates sequent calculus proofs to serve as a benchmark. We demonstrate these ideas on a system for classical logic and argue that they can be generalized to other deep inference systems.

2 Proof Theory with Deep Inference

In this section, we introduce the calculus of structures, the proof theoretic formalism that employs deep inference as its distinguishing feature from the sequent calculus.

The calculus of structures works with logical expressions called *structures*. From a syntactic point of view, structures can be seen as equivalence classes of formulae: The laws such as associativity and commutativity, which are usually implicitly imposed on formulae, become explicit on structures by means of an underlying equational system in a logical system of the calculus of structures. If one considers the notion of a structure from the point of view of the sequent calculus, structures can be seen as expressions intermediate between formulae and sequents which unify these two entities. Let us now see the classical logic structures:

Definition 2.1 [2] There are countably many *positive atoms* and *negative atoms* which are denoted by a, b, c,... Classical logic (KSg) *structures* are generated by

$$R ::= \mathsf{ff} \mid \mathsf{tt} \mid a \mid [R, R] \mid (R, R) \mid \overline{R}$$

where ff and tt are the units false and true, respectively. $[R, R]$ is a *disjunction* and (R, R) is a *conjunction*. $\overline{R}$ is the *negation* of the structure R. KSg structures are considered equivalent modulo the smallest congruence relation induced by the equational system consisting of the equations for associativity and commutativity for disjunction and conjunction, De Morgan equations for negation, and the equations

$$(\mathsf{ff}, \mathsf{ff}) \approx \mathsf{ff}\,, \quad [\mathsf{ff}, R] \approx R\,, \quad [\mathsf{tt}, \mathsf{tt}] \approx \mathsf{tt}\,, \quad (\mathsf{tt}, R) \approx R\,.$$

Inference rules of the calculus of structures are applied to the structures, however these rule applications are not restricted to the top-level connective of the logical expressions as in the sequent calculus. In contrast, they can be applied at any depth inside logical expressions. The context, in which the rule is applied, is represented explicitly and denoted with $S\{\ \}$. Let us see a deductive system for classical logic:

Definition 2.2 [2] System KSg for classical logic is the system given by the rules

$$\mathsf{ai}{\downarrow}\ \frac{S\{\mathsf{tt}\}}{S[a, \bar{a}]}\,, \quad \mathsf{s}\ \frac{S([R, U], T)}{S[(R, T), U]}\,, \quad \mathsf{w}{\downarrow}\ \frac{S\{\mathsf{ff}\}}{S\{R\}}\,, \quad \mathsf{c}{\downarrow}\ \frac{S[R, R]}{S\{R\}}$$

which are called *atomic interaction*, *switch*, *weakening*, and *contraction*, respectively.

The inference rules above denote implications inside contexts, where the premise implies the conclusion. An application of an inference rule coincides with the rewritings in a term rewriting system modulo equational theory. Here, we would like to consider the application of the inference rules from a *bottom-up, proof search* point of view. Then, these rewritings are the rewritings defined by the rewrite relation R/E (see, e.g., [1]), where R is a rewriting system (corresponding to system KSg) and E is the equational theory (given in Definition 2.1) [9]. For instance, for the rule $\mathsf{s} \in \mathsf{KSg}$, we have that

$$\mathsf{s}\ \frac{(c, [a, (\bar{a}, [b, \bar{b}])])}{(c, [a, b, (\bar{a}, \bar{b})])} \quad \text{iff} \quad (c, [a, [b, (\bar{a}, \bar{b})]]) \approx_{\mathsf{E}} (c, [[(\bar{b}, \bar{a}), b], a]) \rightarrow_{\mathsf{s}} (c, [([\bar{b}, b], \bar{a}), a]) \approx_{\mathsf{E}} (c, [a, (\bar{a}, [b, \bar{b}])])\,.$$

Thus, a derivation in (system KSg of) the calculus of structures can be equivalently seen as a chain of instances of inference rules or a chain of rewrites. A derivation

Δ with premise T and conclusion R, and whose inference rules are in KSg will be written as $\Delta\dfrac{T}{R}\!\!\parallel_{\mathsf{KSg}}$ or equivalently as $R \xrightarrow{\Delta}_{\mathsf{KSg}} T$. The proof of a structure R in system KSg is a derivation where the conclusion is R and the premise is $\mathbb{t}$.

Apart from classical logic, the calculus of structures provides deductive systems for linear logic [18], modal logics [16,17], intuitionistic logic, and logics BV [7] and NEL [8]. All the calculus of structures deductive systems for these logics follow the same scheme, where the rules switch and atomic interaction are common components of these systems. However, these rules deal with different notions of conjunction and disjunction, dictated by the equations for the unit in the subject system. In this respect, the notion of a structure which provides a uniform syntax for these logics, allows to observe the common behavior in these systems.

In order to see this on an example, let us consider system BV. In fact, the calculus of structures was originally conceived to introduce system BV in order to capture the sequential composition of process algebras by means of a self-dual, non-commutative logical operator. This logic extends multiplicative linear logic (MLL) with the rules mix and nullary mix (see, e.g., [7]), and a non-commutative self-dual operator that resembles the prefixing in the process algebras. System BV cannot be expressed without deep inference, as Tiu proved in [20].

Definition 2.3 There are countably many *positive atoms* and countably many *negative atoms*. Atoms are denoted by $a, b, c, \ldots$ BV *structures* are generated by

$$R ::= \circ \mid a \mid [\,R\,,\,R\,] \mid (\,R\,,\,R\,) \mid \langle\,R\,;\,R\,\rangle \mid \overline{R}$$

where $\circ$, the *unit*, is not an atom. $[\,R\,,\,R\,]$ is called a *par structure*, $(\,R\,,\,R\,)$ is called a *copar structure*, and $\langle\,R\,;\,R\,\rangle$ is called a *seq structure*. $\overline{R}$ is the *negation* of the structure R. BV structures are considered equivalent modulo the smallest congruence relation induced by the equational system consisting of the equations for associativity and commutativity for par and copar, associativity for seq structures, and the equations

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

System BV is given with the rules

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

which are called *atomic interaction*, *switch*, and *seq*, respectively.

It is important to observe that the seq is a logical operator which is non-commutative and self-dual. A BV structure R has a proof if and only if there is a derivation with the conclusion R and the premise $\circ$. For an indepth exposure to the proof theory of system BV, the reader is referred to [7,20,12].

3 Implementing Deep Inference in Maude

The language Maude [4,5] allows implementing term rewriting systems modulo equational theories due to its very fast matching algorithm that supports different combinations of associative commutative theories, also in the presence of units. These features of language Maude can be used to implement the deductive systems of the calculus of structures in a straight-forward and simple way such that there is a one-to-one match between the definitions of the deductive systems and the corresponding Maude modules. Let us see this first on system KSg. The following Maude functional module implements Definition 2.1:

```
fmod KSg-Signature is
  sorts Unit Atom Structure .
  subsort Unit Atom < Structure .
  ops tt ff : -> Unit .
  op  -_   : Structure -> Structure [prec 50] .
  op [_,_] : Structure Structure -> Structure [assoc comm id: ff] .
  op {_,_} : Structure Structure -> Structure [assoc comm id: tt] .
  ops a b c d e f g h : -> Atom .
endfm
```

In this module, negation of a structure is represented with `-_`. We use the syntax `{_,_}` for conjunction instead of `(_,_)`. This way, we avoid ambiguities, because brackets are often used in meta-level programming and elsewhere in Maude. The information about the associativity and commutativity of the structures and their units are expressed simply by means of the operator attributes, e.g., `[assoc comm id: ff]` for the disjunction.

The following Maude system module implements Definition 2.2.

```
mod KSg is
    inc KSg-Signature .
    var R T U : Structure .   var A : Atom .
    rl [a_interaction] : [ A , - A ] => tt .
    rl [switch]        : [ { R , T } , U ] => { [ R , U ] , T } .
    rl [weakenning1]   : [ R , T ] => [ R , ff ] .
    rl [weakenning2]   : { R , T } => { R ,ff }
    rl [contraction]   : R => [ R , R ] .
    rl [tt]            : [ tt , tt ] =>  tt .
    rl [ff]            : { ff , ff } =>  ff .
endm
```

This module uses the module `KSg-Signature` above. It is important to observe that the rules of system KSg are expressed as bottom-up proof search term rewriting rules. In the calculus of structures inference rules can be applied only inside the contexts that are not under the scope of negation. Thus, in order to avoid the application of the rule weakening to negative atoms, in the module above, we have two rules for weakening. This way, by exploiting pattern matching, we avoid unsound

rewrites, e.g., the unsound rewrite $\bar{a} \to \bar{\mathbb{f}} \approx \mathbb{t}$ becomes impossible. Furthermore, the rules `[tt]` and `[ff]` implement the corresponding equations for unit in Definition 2.1. However, from the point of view of proof search, it suffices to consider these equations by orienting them from left to right, because the system remains complete without considering the right to left applications of these equations.

Similarly to the above module, we can implement system BV. The following two modules implement Definition 2.3:

```
fmod BV-Signature is
 sorts Atom Unit Structure .
 subsort Unit Atom < Structure .
  op o   : -> Unit .    op -_  : Structure -> Structure [prec 50].
  op [_,_] : Structure Structure -> Structure [assoc comm id: o] .
  op {_,_} : Structure Structure -> Structure [assoc comm id: o] .
  op <_;_> : Structure Structure -> Structure [assoc id: o] .
  ops a b c d e f g h l : -> Atom .
endfm

mod BV is
 inc BV-Signature .
  var R T U V : Structure . var A : Atom .
  rl [ai-down] : [ A , - A ] => o .
  rl [s]       : [ { R , T } , U ] => { [ R , U ] , T } .
  rl [q-down]  : [ < R ; T > , < U ; V > ] => < [R,U] ; [T,V] > .
endm
```

In order to compute the derivations of arbitrary length, we need the transitive closure of the rewrite relation $\mathsf{R/E}$. The language Maude implements the transitive closure of the rewriting relation $\mathsf{R/E}$ by means of its built-in breadth-first search (`search`) function. Because of this, it is possible to use these modules for proof search by resorting to the `search` function. This way, for example for the structure $[a, b, (\bar{a}, \bar{b})]$, one can explore all the possible one step rule applications, or search for derivations (or proofs), respectively:

```
search [ a , [ b , { - a , - b } ]  ] =>1 R .
search [ a , [ b , { - a , - b } ]  ] =>* [ a, - a ] .
```

Then, after a successful search, one can display the computed derivation:

```
Maude> show path 78 .
state 0, Structure: [a,[b,{- a,- b}]]
===[ rl [U,{R,T}] => {T,[R,U]} [label s] . ]===>
state 8, Structure: [a,{- a,[b,- b]}]
===[ rl [A,- A] => o [label ai-down] . ]===>
state 78, Structure: [a,- a]
```

In the calculus of structures, inference rules can be applied to the structures that are not in the scope of negation. For this reason, it is more favorable to consider only those structures that are in negation normal form. Furthermore, although the

equations for units can be easily expressed in Maude, these equations often cause redundant matchings of the inference rules where the premise and the conclusion of the instance of the inference rules are equivalent structures. In the following, we will consider the structures to be *in normal form* when they are in negation normal form, and no units can be equivalently removed. For this pupose, within functional modules, which we integrate to the above modules, we orient the equations for De Morgan laws, and equations for unit, in such a way that delivers the normal forms of the structures. By doing so, we can remove the operator attributes `id: ff` and `id: tt` from the module `KSg-Signature` and the operator attribute `id: o` from the module `BV-Signature`. Furthermore, we can move all the invertible rules [3] in the module `KSg` to the module `KSg-UNF` in the form of equations. The rules `[tt]`, `[ff]`, and `[interaction]` are such invertible rules. Because we are interested in proof search, we allow weakening only in the disjunctive contexts.

```
fmod KSg-UNF is including KSg-Signature .
 var R T U : Structure .  var A : Atom .
 eq  - tt = ff .  eq  - ff  =  tt .  eq  - - R = R .
 eq  - [ R , T ] = { - R , - T } .
 eq  - { R , T } = [ - R , - T ] .
 eq  [ ff , R ] = R .        eq { tt , R } = R .
 eq  [ tt , tt ] = tt .      eq { ff , ff } = ff .
 eq  [ A , - A ] = tt .
endfm
```

```
mod KSg is including KSg-UNF .
  var R T U : Structure .
  rl [switch]      : [ { R , T } , U  ] => { [ R , U ] , T } .
  rl [weakenning]  : [ R , T ] => [ R, ff ] .
  rl [contraction] : R => [ R , R ] .
endm
```

Removing the equations for unit in system KSg does not require the modification of the inference rules of system KSg. However, for the case of system BV, when we remove the operator attribute `id: o` from the module `BV-Signature`, some applications of the rule `[q-down]` are broken. In order to maintain these aplications, thus the completeness, we must include the following rules in the module `BV`:

```
 rl [q2]  : [ R , T ] => < R ; T > .
 rl [q3]  : [ R , < T ; U > ] => < [ R , T ] ; U > .
 rl [q4]  : [ R , < T ; U > ] => < T ; [ R , U ] > .
```

Because these modifications disable the redundant instances of the inference rules due to the applications of the equations for unit, they provide a better performance in proof search for system BV [10]. However, because of `[contraction]`, it is not possible to use system KSg for proof search: In breadth-first search, instances of

[3] Invertible rules are those rules for which the premise and the conclusion of every instance of these rules are equivalent logical expressions.

this rule, which copy arbitrary substructures, cause the search space to grow rather quickly. In order to get over this, the application of this rule must be controlled. In the following, we will address this issue in conjunction with some proof theoretical ideas that aim at reducing nondeterminism in proof search.

4 Implementing Proof Theoretic Strategies:

In the calculus of structures, we can construct proofs which consist of separate phases such that in each phase only certain inference rules are used. In particular, we can easily simulate the construction of the sequent calculus proofs and use this as a proof search strategy with the cost of an exponential grow in the size of the proof in some cases (see, e.g., [3]):

Theorem 4.1 *If a structure R has a proof in system* KSg*, then there exist structures R_1, R_2, R_3, R'_1, R'_2, and R'_3 and proofs of the following forms:*

$$\begin{array}{c} \mathtt{tt} \\ \Delta_3 \Big\| \{\mathsf{w}\downarrow\} \\ R_3 \\ \Delta_2 \Big\| \{\mathsf{ai}\downarrow\} \\ R_2 \\ \Delta_1 \Big\| \{\mathsf{s},\mathsf{c}\downarrow\} \\ R \end{array} \quad \overset{i.}{\rightsquigarrow} \quad \begin{array}{c} \mathtt{tt} \\ \Delta_3 \Big\| \{\mathsf{w}\downarrow\} \\ R_3 \\ \Delta_2 \Big\| \{\mathsf{ai}\downarrow\} \\ R_2 \\ \Delta_{1,b} \Big\| \{\mathsf{s}\} \\ R_1 \\ \Delta_{1,a} \Big\| \{\mathsf{c}\downarrow\} \\ R \end{array} \quad \overset{ii.}{\rightsquigarrow} \quad \begin{array}{c} \mathtt{tt} \\ \Delta_2 \Big\| \{\mathsf{ai}\downarrow\} \\ R'_3 \\ \Delta_3 \Big\| \{\mathsf{w}\downarrow\} \\ R_2 \\ \Delta_{1,b} \Big\| \{\mathsf{s}\} \\ R_1 \\ \Delta_{1,a} \Big\| \{\mathsf{c}\downarrow\} \\ R \end{array} \quad \overset{iii.}{\rightsquigarrow} \quad \begin{array}{c} \mathtt{tt} \\ \Delta_2 \Big\| \{\mathsf{ai}\downarrow\} \\ R'_3 \\ \Delta'_{1,b} \Big\| \{\mathsf{s}\} \\ R'_2 \\ \Delta'_3 \Big\| \{\mathsf{w}\downarrow\} \\ R'_1 \\ \Delta_{1,a} \Big\| \{\mathsf{c}\downarrow\} \\ R \end{array}$$

Proof. We can derive the rule, that we call *distributive*(d), as follows:

$$\begin{array}{c} S([R,U],[T,U]) \\ \mathsf{s} \\ S[([R,U],T),U] \\ \mathsf{s} \\ S[(R,T),U,U] \\ \mathsf{c}\downarrow \\ S[(R,T),U] \end{array}$$

By applying this rule exhaustively to structure R bottom up, we obtain the derivation Δ_1 with the premise R_2 which is in conjunctive normal form. Because R_2 is provable, each disjunction in R_2 must have an atom a and its dual $\bar{a}$. By applying the rule $\mathsf{ai}\downarrow$ bottom up to each one of these pairs of dual atoms, we obtain the derivation Δ_2 with the premise R_3, where each disjunction has an instance of the unit $\mathtt{tt}$. By applying the rule $\mathsf{w}\downarrow$ exhaustively to all the remaining structures in each disjunction which are different from the unit $\mathtt{tt}$, we obtain the derivation Δ_3.
i. With structural induction on R, we obtain the derivations $\Delta_{1,a}$ and $\Delta_{1,b}$ from the derivation Δ_1. If R is an atom or the unit $\mathtt{tt}$ or $\mathtt{ff}$, then it is already in conjunctive normal form. If $R = (T,U)$ or $R = [T,U]$ then we have the derivations (1.) and (2.) below by induction hypothesis where T_2 and U_2 are in conjunctive normal form. Let n be the number of disjunctions in U_2. We assume that n is greater than one. Otherwise, we can exchange T_2 with U_2, or if in both T_2 and U_2, there are less than

2 disjunctions, then they are already in conjunctive normal form. We construct the derivations for $R = (T, U)$ and $R = [T, U]$, respectively, as in (3.) and (4.) below:

$$\begin{array}{cccc}
(1.) & (2.) & (3.) & (4.) \\
 & & & R_2 \\
 & & & \Big\|_{\{\mathsf{s}\}} \\
 & & (T_2, U_2) & [T_2, \ldots, T_2, U_2] \\
T_2 & U_2 & \Big\|_{[\Delta'_T, \Delta'_U]\{\mathsf{s}\}} & \Big\|_{[\Delta'_T, \ldots, \Delta'_T, \Delta'_U]\{\mathsf{s}\}} \\
\Big\|_{\Delta'_T\{\mathsf{s}\}} & \Big\|_{\Delta'_U\{\mathsf{s}\}} & (T_1, U_1) & [T_1, \ldots, T_1, U_1] \\
T_1 & U_1 & \Big\|_{[\Delta_T, \Delta_U]\{\mathsf{c}\downarrow\}} & \Big\|_{[\Delta_T, \ldots, \Delta_T, \Delta_U]\{\mathsf{c}\downarrow\}} \\
\Big\|_{\Delta_T\{\mathsf{c}\downarrow\}} & \Big\|_{\Delta_U\{\mathsf{c}\downarrow\}} & (T, U) & [T, \ldots, T, U] \\
T & U & & \Big\|_{\{\mathsf{c}\downarrow\}} \\
 & & & [T, U]
\end{array}$$

ii. We trivially permute each instance of $\mathsf{w}\downarrow$ under the instances of $\mathsf{ai}\downarrow$.
iii. We permute the instances of the rule s over the rule $\mathsf{w}\downarrow$: Other cases being trivial, we consider the following: (*a*.) The redex is of $\mathsf{w}\downarrow$ is inside the contractum of s. (*b*.) The contractum of s is inside the redex of $\mathsf{w}\downarrow$.

$$\begin{array}{ccc}
S(\mathsf{ff}, T) & & S(\mathsf{ff}, T) \\
\mathsf{w}\downarrow S([R, U], T) & \overset{a.}{\rightsquigarrow} & \mathsf{w}\downarrow S(R, T) \\
\mathsf{s}\ S[(R, T), U] & & \mathsf{w}\downarrow S[(R, T), U]
\end{array}
\qquad
\begin{array}{ccc}
S([R, U], \mathsf{ff}) & & S([R, U], \mathsf{ff}) \\
\mathsf{w}\downarrow S([R, U], T, P) & \overset{b.}{\rightsquigarrow} & \mathsf{s}\ S([(R, \mathsf{ff}), U], \mathsf{ff}) \\
\mathsf{s}\ S([(R, T), U], P) & & \mathsf{w}\downarrow S([(R, T), U], \mathsf{ff}) \\
 & & \mathsf{w}\downarrow S([(R, T), U], P)
\end{array}$$

□

In [2] and [18], Brünnler and Straßburger, respectively, present classes of theorems, called decomposition theorems, for classical logic and linear logic. The leftmost derivation in the theorem above is given in the semantic cut elimination proof in [2]. When these theorems provide normal forms at intermediate stages between phases, they can be used as search strategies in proof search. The availability of conjunctive normal form provides such a strategy, however with an exponential cost in the transformation, for some classes of formulae. However, this is the scheme in which sequent calculus proofs are constructed where, in the worst case, an exponentially branching proof tree is generated. In the following, we will present an implementation of this strategy to serve as a benchmark for comparison with the implementations of proof search where deep inference is used.

4.1 Using the Meta-level Features to Implement Decomposition of Proofs

In order to implement the ideas in Theorem 4.1, we require a mechanism that allows to pass information between modules for the rules at different phases of the proof. For this purpose, we employ the meta-level features of Maude (see, e.g., [6]), which allow to represent such information as meta-data in the presence of normal forms. We need to include the meta-level module in module `KSg-Signature` and also add

an operator (`error`) which serves as an error token in the meta-level computation:

```
inc META-LEVEL .      op error : -> [Structure] .
```

Instead of exploring the search space by using the `search` function to find a proof, below we use functional modules which deterministically compute the proof by means of a strategy corresponding to the left-most derivation of Theorem 4.1.

```
fmod distribute is inc KSg-Signature .   var R T U : Structure .
  eq  [ U , { R , T } ]  = { [ U , R ] , [ U , T ] } .
endfm

fmod interaction is inc KSg-Signature .   var A : Atom .
  eq   [ A , - A ]  = tt .
endfm

fmod weakening is  inc KSg-Signature .   var R : Structure .
  eq   [ tt , R ]  = tt .   eq   { tt , R }  = R .
endfm

fmod KSg-Strat is
     inc KSg-Signature . inc KSg-UNF . inc distribute .
     inc interaction . inc weakening .
     op prove_ : Structure -> Structure .
     var R : Structure .

eq prove R =
  downTerm(   getTerm(   metaReduce(['weakening],
    getTerm(       metaReduce(['interaction],
      getTerm(      metaReduce(['distribute],
        getTerm(metaReduce(['KSg-UNF], upTerm( R ) )))))))), error) .
endfm
```

In the implementation above, the different phases of the proof, where different sets of inference rules are used, are represented by functional modules which are called by the operator `prove` of the functional module `KSg-Strat`. Seen procedurally, by means of the operation `upTerm`, this operator first converts the object level representation of the input query term to a Maude meta-level representation of the same term with respect to the module `KSg-Signature`. Then the meta-level term corresponding to the negation normal form of the input term is computed by means of the operation `metaReduce` which takes the meta-representation of the functional module `KSg-UNF` as argument. Then the computed meta-level terms are passed similarly to the meta-level representations of the functional modules `distribute`, `interaction` and `weakening`, respectively, which reduce these meta-level terms with respect to their rules.

4.2 Interaction Rules with Controlled Contraction in Proof Search

Availability of deep inference provides shorter proofs than in the sequent calculus [3]: Applicability of the inference rules at any depth inside a structure makes it

possible to start the construction of a proof by manipulating and annihilating substructures. This provides many more different proofs of a structure, some of which are shorter than in the sequent calculus. However, deep inference causes a greater nondeterminism: Because the inference rules can be applied at many more positions than in the sequent calculus, the breadth of the search space increases rather quickly. In order to get over this problem, in [11] we have introduced the following modification on the rule s which exploits an interaction scheme on the structures.

Definition 4.2 For a structure R, let $\mathsf{at}\,R$ denote the set of atoms appearing in structure R. The rule *lazy interaction switch* (lis) is the rule

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

where $\mathsf{at}\,\overline{W} \cap \mathsf{at}\,R \neq \emptyset$ and W is not a disjunction (par structure).

The intuition behind the rule lis can be seen as follows: Let us consider the subformulae which are in a disjunction relation as interacting formulae, whereas those formula in a conjunction relation as non-interacting formula. For example, when we consider the formula $[a, b, (\bar{a}, \bar{b})]$, a is interacting with b, $\bar{a}$, and $\bar{b}$, whereas $\bar{a}$ is interacting with a and b, but it is not interacting with $\bar{b}$. The interacting formulae have the potential to annihilate each other to construct a proof, whereas the non-interacting formulae do not. In [11], we have shown that the rule switch can be replaced with the rule lis in systems BV and KSg without losing completeness.

Theorem 4.3 *[11] Systems* $\{\mathsf{ai}{\downarrow}, \mathsf{s}\}$ *and* $\{\mathsf{ai}{\downarrow}, \mathsf{lis}\}$ *are equivalent, that is, they prove the same structures.*

In the following, by integrating the contraction rule to the rule lis we will obtain a system where the nondeterminism in proof search is reduced and the application of the contraction rule is controlled.

Definition 4.4 The rule cis is the rule

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

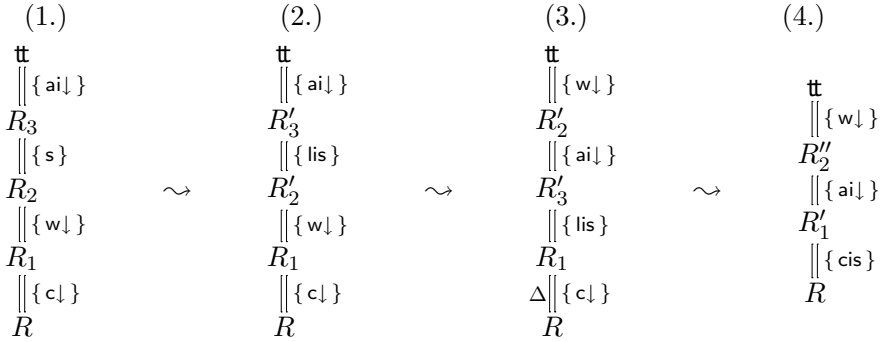
where $\mathsf{at}\,W \cap \mathsf{at}\,R \neq \emptyset$, and W is not a disjunction (par structure).

Definition 4.5 System KSgic is the system resulting from replacing the rule s and $\mathsf{c}{\downarrow}$ with the rule cis.

Theorem 4.6 *Systems* KSg *and* KSgic *are equivalent, that is, they prove the same structures.*

Proof. Every proof in system KSgic is a proof in system KSg. For the proof of the

other direction, let R be a provable KSg structure.



Consider the proof (1.) which we construct by Theorem 4.1. By Theorem 4.3, we construct the proof (2.). By trivial permutations of the rule $\mathsf{w}\!\downarrow$ over the rule lis, we then construct the proof (3.). In order the construct the proof (4.), we repeat the following procedure inductively: We take the top-most instance of the rule $\mathsf{c}\!\downarrow$ in derivation Δ: If the redex of this rule is a par structure, we replace it as follows:

$$\mathsf{c}\!\downarrow \frac{S[R_1, R_2, \ldots, R_n, R_1, R_2, \ldots, R_n]}{S[R_1, R_2, \ldots, R_n]} \quad \rightsquigarrow \quad \begin{array}{l} \mathsf{c}\!\downarrow \dfrac{S[R_1, R_2, \ldots, R_n, R_1, R_2, \ldots, R_n]}{\vdots} \\ \mathsf{c}\!\downarrow \; S[R_1, R_2, \ldots, R_n, R_1] \\ \mathsf{c}\!\downarrow \; S[R_1, R_2, \ldots, R_n] \end{array}$$

We then permute the top-most instance of the rule $\mathsf{c}\!\downarrow$ until it cannot be permuted and where its contractum is used in an instance of the rule lis and we replace these two rule instances with an instance of the rule cis. □

The condition imposed on the rule s reduces the breadth of the search space by reducing the numbers of the possible rule instances of this rule. This situation delays the exponential blow-up in proof search and makes it plausible to consider more complex formulae for proof search.

We implement the conditional inference rules as conditional rewrite rules in Maude. In order to compute the condition of the rules lis and cis, we use the functional module below which implements the function `can-interact`.

```
fmod Can-interact is inc KSg-Signature .
  sort Interaction_Query .
  op can-interact   : -> Interaction_Query .
  op empty-set : -> Interaction_Query .

  op _or_ : Interaction_Query Interaction_Query
                        -> Interaction_Query [assoc comm prec 70] .
  op _ci_ : Atom Structure -> Interaction_Query [prec 60] .
```

```
  var R T U V : Structure .      var A B : Atom .
  var C : Interaction_Query .

  eq    A  ci - A = can-interact .
  eq  - A  ci   A = can-interact .
  eq    A  ci   B = empty-set [owise] .

  eq [ T , U ] ci R =  T ci R  or  U ci R .
  eq { T , U } ci R =  T ci R  or  U ci R .
  eq A ci [ R , T ] =  A ci R  or  A ci T .
  eq A ci { R , T } =  A ci R  or  A ci T .

  eq can-interact or  C  = can-interact .
  eq empty-set    or  C  = C .
endfm
```

The following system module implements system KSgic.

```
mod KSgic is  inc KSg-UNF .  inc Can-interact .
   var R T U V P Q : Structure .  var A  : Atom .

  crl  [rls11] : [ { R , T } , A ] => [ { [ R , A ] , T } , A ]
                          if  ( R ci A ) == can-interact .

  crl  [rls21] : [ { R , T } , { U, V } ] =>
                     [ { [ R , { U , V } ] , T } , { U , V } ]
                           if ( R ci  { U , V } ) == can-interact .
endm
```

Remark 4.7 In proof search, the rule cis copies many structures which are often superfluous and weakened during the construction of the proofs. When we consider the way sequent calculus proofs are constructed, an alternative to this rule is as follows: The rules cis_1 and cis_2 are the rules

$$\mathsf{cis}_1\ \frac{S([R,W],[T,W])}{S[(R,T),W]} \qquad \mathsf{cis}_2\ \frac{S([R,W],T)}{S[(R,T),W]}$$

where W is not a disjunction, in cis_1 we have $\mathsf{at}\,W \cap \mathsf{at}\,R \neq \emptyset$ and $\mathsf{at}\,W \cap \mathsf{at}\,T \neq \emptyset$, and in cis_2 we have $\mathsf{at}\,W \cap \mathsf{at}\,R \neq \emptyset$ and $\mathsf{at}\,W \cap \mathsf{at}\,T = \emptyset$. Let us call KSgic' the system obtained by replacing the cis in system KSgic with the rules cis_1 and cis_2. Although for some formulae KSgic seems to be more advantegous, for pigeon-hole formulae, system KSgic performs better than system KSgic' (See Subsection 4.3).

4.3 Experiments

We performed experiments on the modules discussed by running them on the formulae below. The results are displayed in Table 1. There, `fDKSg` denotes the

	System	# states explored	# of trewrites
1.	`fDKSg`	–	29
	`DKSg`	7	61
	`KSgic`	8	265
	`KSgic'`	8	202
3.	`fDKSg`	–	46
	`DKSg`	158	1150
	`KSgic`	15	313
	`KSgic'`	11	466

	System	# states explored	# of rewrites
2.	`fDKSg`	–	32
	`DKSg`	187	2468
	`KSgic`	18	265
	`KSgic'`	18	822
4.	`fDKSg`	–	86
	`DKSg`	–	–
	`KSgic`	302	34126
	`KSgic'`	10846 (*)	1765578

Table 1
Results of proof search experiments with the modules for the systems `fDKSg`, `DKSg`, `KSgic` and `KSgic'`.

module `KSg-Strat`; `DKSg` denotes the module where the switch rule is replaced with the following rule:

```
rl [distributive] : [ { R , T } , U ] => { [ R , U ] , [ T , U ] } .
```

The proof marked with (*) is computed in 920 ms. All other displayed proofs are computed in less than 20ms.

1. $[(a,b),([\bar{a},\bar{b}],[\bar{a},\bar{b}])]$
2. $[a,b,(\bar{a},\bar{b},[c,d,(\bar{c},\bar{d},[e,f,(\bar{e},\bar{f})])])]$
3. $[\bar{c},(\bar{d},\bar{e}),(e,\bar{a}),(c,[a,d])]$
4. $[([a,b],[\bar{c},[a,b]]),(\bar{a},\bar{d}),([(c,d),\bar{b}],[c,d])]$

5 Discussion

We have presented a general procedure for implementing deep inference deductive systems by using term rewriting features of Maude. In particular, we have presented implementations of the systems KSg and BV. We have also shown that proof theoretical strategies can be implemented using the meta-level features and conditional rewriting rules of this language. By resorting to the meta-level features of Maude, we have presented an implementation that simulates sequent calculus proofs with the purpose of serving as a benchmark for performance comparison. We have analogously applied the ideas of this paper to other deep inference systems for linear logic [18], and system NEL [8]. These implementations are available for download. [4]

In [13], Marti-Oliet and Meseguer present a Maude implementation of linear logic as a sequent calculus system. There, in order to capture the branching at the application of multiple premise sequent calculus inference rules, they introduce an operator, called *configuration*, which provides a representation of the meta-level at the object-level. In deep inference deductive systems, because the meta-level merges

[4] `http://www.doc.ic.ac.uk/~ozank/maude_cos.html`

with the object level, and hence there is no multiple premise inference rules, the deep inference implementation of linear logic does not require additional operators on top of those of linear logic. Another aspect that distinguishes our implementation of linear logic is due to the *promotion* rule. In the sequent calculus, promotion rule is defined as the rule on the left below, which involves a global knowledge of the context: the application of this rule requires each formula in the context of $!A$ to be checked to have the form $?B$. In the calculus of structures this rule is replaced with the rule on the right, which does not require such a global view of the formulae.

$$\frac{\vdash A, ?B_1, \ldots, ?B_n}{\vdash\ !A, ?B_1, \ldots, ?B_n}\ ! \qquad \frac{S\{!\,[R,T]\}}{S[?R, !T]}\ \mathsf{p}\downarrow$$

Schäfer has developed a graphical proof editor, called GraPE [15], which functions as a graphical user interface to the Maude modules discussed in this paper. This tool makes it possible to use the Maude implementations interactively: By using the GraPE tool, the user can guide the proof construction and choose between automated proof search and user-guided proof construction. Then the output derivation can be exported as LaTeXcode. The GraPE tool is available online [5].

In [3], Bruscoli and Guglielmi show that for a class of classical tautologies called Statman's tautologies, deep inference provides an exponential speed up in contrast to the sequent calculus proofs. The restrictions imposed by the rules discussed in this paper preserve the shortest proofs of [3]. However, proof search applications of these deductive systems require further restrictions in the application of the inference rules, which is a topic of on going work in conjunction with an extensive comparison of these implementations and proof complexity analysis. A deep inference system for the logic of bunched implications [14] is also a potential application of the ideas above for future work. Other topics of future investigation include introducing strategies for partitioning the search space by resorting to the *splitting theorem* (see, e.g., [7,12]).

References

[1] F. Baader and T. Nipkow. *Term Rewriting and All That*, volume 1. Cambridge University Press, 1998.

[2] Kai Brünnler. *Deep Inference and Symmetry in Classical Proofs*. PhD thesis, TU Dresden, 2003.

[3] Paola Bruscoli and Alessio Guglielmi. On the proof complexity of deep inference. Available on the web at http://cs.bath.ac.uk/ag/p/PrComplDI.pdf, 2007.

[4] M. Clavel, F. Durán, S. Eker, P. Lincoln, N. Martí-Oliet, J. Meseguer, and J. Quesada. Maude: specification and programming in rewriting logic. *Theoretical Computer Science*, 285:187–243, 2002.

[5] M. Clavel, F. Durán, S. Eker, P. Lincoln, N. Martí-Oliet, J. Meseguer, and C. Talcott. The Maude 2.0 system. In Robert Nieuwenhuis, editor, *Rewriting Techniques and Applications, Proceedings of the 14th International Conference*, volume 2706. Springer, 2003.

[6] M. Clavel, F. Durán, S. Eker, J. Meseguer, and M-O. Stehr. Maude as a formal meta-tool. In Jeannette M. Wing, Jim Woodcock, and Jim Davies, editors, *FM'99 — Formal Methods, World Congress on Formal Methods in the Development of Computing Systems, Toulouse, France, September 20–24, 1999 Proceedings, Volume II*, volume 1709 of *LNCS*, pages 1684–1703. Springer, 1999.

[5] http://grape.sourceforge.net/

[7] Alessio Guglielmi. A system of interaction and structure. *ACM Transactions on Computational Logic*, 8(1):1–64, 2007.

[8] Alessio Guglielmi and Lutz Straßburger. A non-commutative extension of MELL. In M. Baaz and A. Voronkov, editors, *LPAR 2002*, volume 2514 of *LNAI*, pages 231–246. Springer-Verlag, 2002.

[9] Ozan Kahramanoğulları. Implementing system BV of the calculus of structures in Maude. In L. A. i Alemany and P. Égré, editors, *Proc. of the ESSLLI-2004 Student Session*, pages 117–127, Université Henri Poincaré, Nancy, 2004. 16th European Summer School in Logic, Language and Information.

[10] Ozan Kahramanoğulları. System BV without the equalities for unit. In C. Aykanat, T. Dayar, and I. Körpeoğlu, editors, *Proceedings of the 19th International Symposium on Computer and Information Sciences, ISCIS'04*, volume 3280 of *LNCS*. Springer, 2004.

[11] Ozan Kahramanoğulları. Reducing nondeterminism in the calculus of structures. In M. Hermann and A. Voronkov, editors, *LPAR 2006, Phnom Penh, Cambodia*, volume 4246 of *LNCS*, pages 272–286. Springer, 2006.

[12] Ozan Kahramanoğulları. System bv is np-complete. *Annals of Pure and Applied Logic*, 2007. to appear.

[13] N. Martí-Oliet and J. Meseguer. Rewriting logic as a logical and semantic framework. In J. Meseguer, editor, *Proc. 1st Internat Workshop on Rewriting Logic and its Application, WRLA' 96*, volume 4 of *Electronic Notes in Theoretical Computer Science*, pages 189–224. Elsevier, 1996.

[14] P.W. O'Hearn and D.J. Pym. The logic of bunched implications. *Bulletin of Symbolic Logic*, 5(2):215–244, 1999.

[15] Max Schäfer. The design and implementation of the grape graphical proof editor. Master's Project Report, TU Dresden. Available at http://grape.sourceforge.net/grape.pdf, 2006.

[16] Charles Stewart and Phiniki Stouppa. A systematic proof theory for several modal logics. In Renate Schmidt, Ian Pratt-Hartmann, Mark Reynolds, and Heinrich Wansing, editors, *Advances in Modal Logic*, volume 5 of *King's College Publications*, pages 309 – 333, 2005.

[17] Phiniki Stouppa. A deep inference system for the modal logic S5. *Studia Logica*, 2006. to appear.

[18] Lutz Straßburger. *Linear Logic and Noncommutativity in the Calculus of Structures*. PhD thesis, TU Dresden, 2003.

[19] Alwen Fernanto Tiu. A local system for intuitionistic logic. In M. Hermann and A. Voronkov, editors, *LPAR 2006, Phnom Penh, Cambodia*, volume 4246 of *LNCS*, pages 242–256. Springer, 2006.

[20] Alwen Fernanto Tiu. A system of interaction and structure II: The need for deep inference. *Logical Methods in Computer Science*, 2(2-4):1–24, 2006.