

Electronic Notes in Theoretical Computer Science 219 (2008) 35–50

[www.elsevier.com/locate/entcs](http://www.elsevier.com/locate/entcs)

Maude as a Platform for Designing and Implementing Deep Inference Systems

# Ozan Kahramanog˘ulları [1](#_bookmark0) *,*[2](#_bookmark0)

*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

# 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 connec- tive 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 Bru¨nnler and the anonymous referees for their comments and suggestions.

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

1571-0661 © 2008 Elsevier B.V. Open access under [CC BY-NC-ND license.](http://creativecommons.org/licenses/by-nc-nd/3.0/)

doi:10.1016/j.entcs.2008.10.033

Deep inference has originally emerged as a means to conceive the logical system BV [[7](#_bookmark17)]. 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](#_bookmark30)]. 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 the- ory: In [[2](#_bookmark10)], Bru¨nnler presents deep inference systems for classical logic; in [[18](#_bookmark28)], Straßburger presents systems for different fragments of linear logic. In [[16](#_bookmark26),[17](#_bookmark27)], Stewart and Stouppa give systems for a class of modal logics. Tiu presents, in [[19](#_bookmark29)], 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 cal- culus. 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](#_bookmark11)]. 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. How- ever, 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](#_bookmark13),[5](#_bookmark14)] 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](#_bookmark18)] 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.

# 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 formu- lae: 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](#_bookmark10)] There are countably many *positive atoms* and *negative atoms*

which are denoted by *a*, *b*, *c*,. . . Classical logic (KSg) *structures* are generated by

*R* ::= ff | tt | *a* | [ *R, R* ] | ( *R, R* ) | *R*

where ff and tt are the units false and true, respectively. [*R, R*] is a *disjunction* and (*R, R*) is a *conjunction*. *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

(ff*,* ff) ≈ ff *,* [ff*, R*] ≈ *R ,* [tt*,* tt] ≈ tt *,* (tt*, R*) ≈ *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](#_bookmark10)] System KSg for classical logic is the system given by the rules

*S*{tt}

ai↓ *,*

*S* [*a, a*¯]

*S*([*R, U* ]*,T* )

s *,*

*S* [(*R, T* )*,U* ]

*S*{ff}

w↓ *,*

*S*{*R*}

*S* [*R, R*]

c↓

*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 rewrit- ings 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](#_bookmark12)]), where R is a rewriting system (corresponding to system KSg) and E is the equational theory (given in Definition [2.1](#_bookmark1)) [[9](#_bookmark19)]. For instance, for the rule s ∈ KSg, we have that

(*c,* [*a,* (*a*¯*,* [*b,* ¯*b*])])

s (*c,* [*a, b,* (*a*¯*,* ¯*b*)]) iff

(*c,* [*a,* [*b,* (*a*¯*,* ¯*b*)]]) ≈E (*c,* [[(¯*b, a*¯)*, b*]*, a*]) →s

¯ ¯

(*c,* [([*b, b*]*, a*¯)*, a*]) ≈E (*c,* [*a,* (*a*¯*,* [*b, 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

*T*

Δ KSg

*R*

or equivalently as *R* −Δ→KSg *T* . The proof of a structure *R* in

system KSg is a derivation where the conclusion is *R* and the premise is tt.

Apart from classical logic, the calculus of structures provides deductive sys- tems for linear logic [[18](#_bookmark28)], modal logics [[16](#_bookmark26),[17](#_bookmark27)], intuitionistic logic, and logics BV

[[7](#_bookmark17)] and NEL [[8](#_bookmark20)]. 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](#_bookmark17)]), 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](#_bookmark30)].

Definition 2.3 There are countably many *positive atoms* and countably many *neg- ative atoms*. Atoms are denoted by *a, b, c,...* BV *structures* are generated by

*R* ::= ◦ | *a* | [ *R, R* ] | ( *R, R* ) | ⟨ *R* ; *R* ⟩ | *R*

where ◦, the *unit*, is not an atom. [ *R, R* ] is called a *par structure*, ( *R, R* ) is called a *copar structure*, and ⟨ *R* ; *R* ⟩ is called a *seq structure*. *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

[◦*, R*] ≈ *R ,* (◦*, R*) ≈ *R ,*

⟨◦*, R*⟩≈ *R ,* (*R,* ◦) ≈ *R ,*

[*R, T* ] ≈ (*R, T* ) *,* ⟨*R, T* ⟩≈ ⟨*R, T* ⟩ *,*

(*R, T* ) ≈ [*R, T* ] *, R* ≈ *R ,* ¯◦≈◦ *.*

System BV is given with the rules

*S*{◦}

ai↓ *,*

*S* [*a, a*¯]

*S*([*R, U* ]*,T* )

s *,*

*S* [(*R, T* )*,U* ]

*S*⟨[*R, U* ]; [*T, V* ]⟩

q↓

*S* [⟨*R*; *T* ⟩*,* ⟨*U* ; *V* ⟩]

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 ◦. For an indepth exposure to the proof theory of system BV, the reader is referred to [[7](#_bookmark17),[20](#_bookmark30),[12](#_bookmark22)].

# Implementing Deep Inference in Maude

The language Maude [[4](#_bookmark13),[5](#_bookmark14)] allows implementing term rewriting systems modulo equa- tional theories due to its very fast matching algorithm that supports different com- binations 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 cor- responding Maude modules. Let us see this first on system KSg. The following Maude functional module implements Definition [2.1](#_bookmark1):

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 ambigui- ties, because brackets are often used in meta-level programming and elsewhere in Maude. The information about the associativity and commutativity of the struc- tures 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](#_bookmark2).

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 ap- plication 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 *a*¯ → ¯ff ≈ tt becomes impossible. Furthermore, the rules [tt] and [ff] implement the corresponding equations for unit in Defi- nition [2.1](#_bookmark1). 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](#_bookmark3):

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 R*/*E. The language Maude implements the transitive closure of the rewriting relation 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,* (*a*¯*,* ¯*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](#_bookmark4) 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 perfor- mance in proof search for system BV [[10](#_bookmark21)]. 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.

# 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](#_bookmark11)]):

Theorem 4.1 *If a structure R has a proof in system* KSg*, then there exist struc-*

*tures R*1*, R*2*, R*3*, R*' *, R*' *, and R*'

*and proofs of the following forms:*

1 2

tt

tt Δ3

3

{ w↓}

tt

Δ2 { ai↓}

tt

Δ2 { ai↓}

Δ3 { w↓} *R*3 ' '

*R*

*R*

Δ

3

3

*R*3 Δ2

2

{ ai↓}

Δ3 { w↓}

'

1*,b*

*R*

~

{ s }

Δ2 { ai↓}

2

~*i. R*

~*ii. R*

*iii.* '

2

Δ

3

*R*2 Δ1*,b*

{ s }

Δ1*,b*

{ s }

' { w↓}

Δ1 { s*,* c↓} *R*1 *R*1 '

*R*

1

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| *R* | Δ1*,a* | { c↓} | Δ1*,a* | { c↓} | Δ1*,a* | { c↓} |
|  | *R* |  | *R* |  | *R* |  |

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

*S*([*R, U* ]*,* [*T, U* ])

s

*S* [([*R, U* ]*,T* )*,U* ]

s

*S* [(*R, T* )*, U,U* ]

c↓

*S* [(*R, T* )*,U* ]

By applying this rule exhaustively to structure *R* bottom up, we obtain the deriva- tion Δ1 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 *a*¯. By applying the rule ai↓ bottom up to each one of these pairs of dual atoms, we obtain the derivation Δ2 with the premise *R*3, where each disjunction has an instance of the unit tt. By applying the rule w↓ exhaustively to all the remaining structures in each disjunction which are different from the unit tt, we obtain the derivation Δ3.

1. With structural induction on *R*, we obtain the derivations Δ1*,a* and Δ1*,b* from the derivation Δ1. If *R* is an atom or the unit tt or 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:

(1*.*) (2*.*) (3*.*) (4*.*)

*R*2

{ s }

(*T*2*, U*2)

*T U*

[*T*2*,... , T*2*, U*2 ]

' 2 ' 2 [Δ' *,*Δ' ] { s }

[Δ' *,...,*Δ' *,*Δ' ] { s }

Δ*T* {s}

Δ*U* {s} *T U*

*T T U*

*T*1 *U*1

(*T*1*, U*1)

[*T*1*,... , T*1*, U*1 ]

Δ*T* {c↓}

Δ*U* {c↓}

[Δ*T ,*Δ*U* ] { c↓}

[Δ*T ,...,*Δ*T ,*Δ*U* ]

{ c↓}

*T U* (*T, U* )

[*T,... ,T,U* ]

 { c↓}

[*T, U* ]

1. We trivially permute each instance of w↓ under the instances of ai↓.
2. We permute the instances of the rule s over the rule w↓: Other cases being trivial, we consider the following: (*a.*) The redex is of w↓ is inside the contractum of s. (*b.*) The contractum of s is inside the redex of w↓.

*S*(ff*,T* )

*S*(ff*,T* )

*S*([*R, U* ]*,* ff)

*S*([*R, U* ]*,* ff)

s

w↓

*S*([*R, U* ] *,T* )

s

*S* [(*R, T* )*,U* ]

~*a.*

w↓

*S*(*R, T* )

w↓

*S* [(*R, T* )*, U* ]

w↓

*S*([*R, U* ]*,T,P* )

s

*S*([(*R, T* )*,U* ]*,P* )

~*b.*

*S*([(*R,* ff)*,U* ]*,* ff)

w↓

*S*([(*R, T* )*,U* ]*,* ff)

w↓

*S*([(*R, T* )*,U* ]*,P* )

□

In [[2](#_bookmark10)] and [[18](#_bookmark28)], Bru¨nnler and Straßburger, respectively, present classes of theo- rems, called decomposition theorems, for classical logic and linear logic. The left- most derivation in the theorem above is given in the semantic cut elimination proof in [[2](#_bookmark10)]. 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 ex- ponentially 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.

* 1. *Using the Meta-level Features to Implement Decomposition of Proofs*

In order to implement the ideas in Theorem [4.1](#_bookmark5), 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](#_bookmark15)]), 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](#_bookmark5).

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 procedu- rally, 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.

* 1. *Interaction Rules with Controlled Contraction in Proof Search*

Availability of deep inference provides shorter proofs than in the sequent calculus

[[3](#_bookmark11)]: 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 sub- structures. 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 posi- tions than in the sequent calculus, the breadth of the search space increases rather quickly. In order to get over this problem, in [[11](#_bookmark18)] 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 at *R* denote the set of atoms appearing in structure *R*. The rule *lazy interaction switch* (lis) is the rule

lis

*S*([*R, W* ]*,T* )

*,*

*S* [(*R, T* )*,W* ]

where at *W* ∩ at *R* /= ∅ 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,* (*a*¯*,* ¯*b*)], *a* is interacting with *b*, *a*¯, and ¯*b*, whereas

*a*¯ is interacting with *a* and *b*, but it is not interacting with ¯*b*. The interacting

formulae have the potential to annihilate each other to construct a proof, whereas the non-interacting formulae do not. In [[11](#_bookmark18)], 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*](#_bookmark18)*] Systems* {ai↓*,* s} *and* {ai↓*,* 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

cis

*S* [([*R, W* ]*,T* )*,W* ]

*S* [(*R, T* )*,W* ]

where at *W* ∩ at *R* /= ∅, and *W* is not a disjunction (par structure).

Definition 4.5 System KSgic is the system resulting from replacing the rule s and

c↓ 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.

(1*.*) (2*.*) (3*.*) (4*.*)

tt

{ ai↓}

tt

{ ai↓}

tt

{ w↓} tt

*R*

*R*3

{ s }

*R*2 ~

'

3

*R*

{ lis } '

*R*

~

2

'

2

{ ai↓} '

*R*

~

3

{ w↓}

''

*R*

2

{ ai↓}

|  |  |  |  |
| --- | --- | --- | --- |
| { w↓} | { w↓} | { lis } | *R*'  1 |
| *R*1 | *R*1 | *R*1 | { cis } |
| { c↓}  *R* | { c↓}  *R* | Δ { c↓}  *R* | *R* |

Consider the proof (1*.*) which we construct by Theorem [4.1](#_bookmark5). By Theorem [4.3](#_bookmark6), we construct the proof (2*.*). By trivial permutations of the rule w↓ 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 c↓ in derivation Δ: If the redex of this rule is a par structure, we replace it as follows:

*S* [*R*1*, R*2*,... , Rn, R*1*, R*2*,... , Rn* ]

c↓

*S* [*R*1*, R*2*,... , Rn*]

*S* [*R*1*, R*2*,... , Rn, R*1*, R*2*,... , Rn* ]

c↓

.

~ c↓

*S* [ *R ,R ,... ,R , R* ]

1 2 *n* 1

c↓

*S* [ *R*1*, R*2*,... , Rn* ]

We then permute the top-most instance of the rule c↓ 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 cis1 and cis2 are the rules

cis1

*S*([*R, W* ]*,* [*T, W* ])

*S* [(*R, T* )*,W* ]

cis2

*S*([*R, W* ]*,T* )

*S* [(*R, T* )*,W* ]

where *W* is not a disjunction, in cis1 we have at *W* ∩ at *R* /= ∅ and at *W* ∩ at *T* /= ∅, and in cis2 we have at *W* ∩ at *R* /= ∅ and at *W* ∩ at *T* = ∅. Let us call KSgic' the system obtained by replacing the cis in system KSgic with the rules cis1 and cis2. 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](#_bookmark7)).

* 1. *Experiments*

We performed experiments on the modules discussed by running them on the formulae below. The results are displayed in Table [1](#_bookmark8). 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*)*,* ([*a*¯*,* ¯*b*]*,* [*a*¯*,* ¯*b*])] 2*.* [*a, b,* (*a*¯*,* ¯*b,* [*c, d,* (*c*¯*, d*¯*,* [*e, f,* (*e*¯*, f*¯)])])]

3*.* [*c*¯*,* (*d*¯*, e*¯)*,* (*e, a*¯)*,* (*c,* [*a, d*])] 4*.* [([*a, b*]*,* [*c*¯*,* [*a, b*]])*,* (*a*¯*, d*¯)*,* ([(*c, d*)*,* ¯*b*]*,* [*c, d*])]

# 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 theo- retical 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 anal- ogously applied the ideas of this paper to other deep inference systems for linear logic [[18](#_bookmark28)], and system NEL [[8](#_bookmark20)]. These implementations are available for download. [4](#_bookmark9) In [[13](#_bookmark23)], 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.

▶ *A,* ?*B*1*,... ,* ?*Bn*

!

▶ !*A,* ?*B*1*,... ,* ?*Bn*

*S*{! [*R, T* ]}

p↓

*S* [?*R,* !*T* ]

Sch¨afer has developed a graphical proof editor, called GraPE [[15](#_bookmark25)], which func- tions 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 be- tween automated proof search and user-guided proof construction. Then the output derivation can be exported as LATEXcode. The GraPE tool is available online [5](#_bookmark16) .

In [[3](#_bookmark11)], 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](#_bookmark11)]. However, proof search applications of these deductive systems require further restrictions in the application of the in- ference rules, which is a topic of on going work in conjunction with an extensive comparison of these implementations and proof complexity analysis. A deep infer- ence system for the logic of bunched implications [[14](#_bookmark24)] 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](#_bookmark17),[12](#_bookmark22)]).

# References

1. F. Baader and T. Nipkow. *Term Rewriting and All That*, volume 1. Cambridge University Press, 1998.
2. Kai Bru¨nnler. *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,](http://cs.bath.ac.uk/ag/p/PrComplDI.pdf) 2007.
4. M. Clavel, F. Dur´an, S. Eker, P. Lincoln, N. Mart´ı-Oliet, J. Meseguer, and J. Quesada. Maude: specification and programming in rewriting logic. *Theoretical Computer Science*, 285:187–243, 2002.
5. M. Clavel, F. Dur´an, S. Eker, P. Lincoln, N. Mart´ı-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.

1. M. Clavel, F. Dur´an, 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/>

1. Alessio Guglielmi. A system of interaction and structure. *ACM Transactions on Computational Logic*, 8(1):1–64, 2007.
2. 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.

1. Ozan Kahramano˘gulları. Implementing system BV of the calculus of structures in Maude. In L. A. i Alemany and P. E´gr´e, editors, *Proc. of the ESSLLI-2004 Student Session*, pages 117–127, Universit´e Henri Poincar´e, Nancy, 2004. 16th European Summer School in Logic, Language and Information.
2. Ozan Kahramano˘gulları. System BV without the equalities for unit. In C. Aykanat, T. Dayar, and

I. Ko¨rpeo˘glu, editors, *Proceedings of the 19th International Symposium on Computer and Information Sciences, ISCIS’04*, volume 3280 of *LNCS*. Springer, 2004.

1. Ozan Kahramano˘gulları. 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.

1. Ozan Kahramano˘gulları. System bv is np-complete. *Annals of Pure and Applied Logic*, 2007. to appear.
2. N. Mart´ı-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.
3. P.W. O’Hearn and D.J. Pym. The logic of bunched implications. *Bulletin of Symbolic Logic*, 5(2):215– 244, 1999.
4. Max Sch¨afer. The design and implementation of the grape graphical proof editor. Master’s Project Report, TU Dresden. Available at [http://grape.sourceforge.net/grape.pdf,](http://grape.sourceforge.net/grape.pdf) 2006.
5. 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.
6. Phiniki Stouppa. A deep inference system for the modal logic S5. *Studia Logica*, 2006. to appear.
7. Lutz Straßburger. *Linear Logic and Noncommutativity in the Calculus of Structures*. PhD thesis, TU Dresden, 2003.
8. 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.

1. 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.