Journal of Automated Reasoning **12:** 103–129, 1994.

Resolution Theorem Proving in Reified Modal Logics

J. STUART AITKEN[1], HAN REICHGELT[2] and NIGEL SHADBOLT[1]*
[1] *A.I. Group, Department of Psychology, University of Nottingham, University Park, Nottingham, NG7 2RD, England, email: stuart,nrs@psyc.nott.ac.uk*
[2] *Department of Computer Science, University of the West Indies, Mona, Kingston 7, Jamaica, e-mail: han@uwimona.edu.jm*

(Received: 30 November 1992; in final form: 23 April 1993)

Abstract. This paper is concerned with the application of the resolution theorem proving method to reified logics. The logical systems treated include the branching temporal logics and logics of belief based on K and its extensions. Two important problems concerning the application of the resolution rule to reified systems are identified. The first is the redundancy in the representation of truth functional relationships and the second is the axiomatic reasoning about modal structure. Both cause an unnecessary expansion in the search space. We present solutions to both problems which allow the axioms defining the reified logic to be eliminated from the database during theorem proving hence reducing the search space while retaining completeness. We describe three theorem proving methods which embody our solutions and support our analysis with empirical results.

Key words. Reified logic, model logic, theorem proving.

1. Introduction

In the reified approach to defining logical systems the semantics of the reified logics are defined by axioms in first-order logic. The advantage of the approach is that the reified logic is represented in a logical system (first-order logic) whose semantics and proof methods are well understood. First-order logic provides a sound framework in which to prove theorems **about** the reified logic.

The reified approach, although too complex for specifying standard propositional and first-order logics, is valuable for specifying propositional and quantified modal logics, and can be useful in gaining an understanding of the properties of such logics. A lack of clarity in both semantics and inference rules has been the source of many disagreements among the advocates and opponents of modal logics (see the discussion in Linsky (1971) and in Reichgelt (1989a) for more recent examples of this problem).

One consequence of adopting the reified approach is that if we wish to automate proofs for modal systems then any of the standard theorem proving methods for

* Much of the research reported in this paper was supported by DTI IED SERC grant No. GR/F 35968, and was carried out whilst Han Reichgelt was at the University of Nottingham.

first-order logic can be used to implement a theorem prover for a reified modal logic. However, to-date there has been little empirical or theoretical work on the properties of theorem provers for reified logics (Reichgelt, 1989a).

This paper explores the theoretical and practical issues raised by the attempt to improve the efficiency of theorem proving in reified logics. In doing so we identify two major problems in the application of resolution to the reified logics of branching time and of belief. The first problem is the redundancy in the representation of truth functional relationships in the database, for example a disjunction may be represented explicitly {HOLDS(or(p,q),w)} and may also be represented as: {HOLDS(p,w),HOLDS(q,w)}. This is the outcome of maintaining the axiomatic definitions of the logical operators in the database during theorem proving. Our first improvement to a naïve approach for theorem proving is the elimination of these axioms and their replacement by a rewriting procedure.

The second problem, which remains after the first has been eliminated, is the axiomatic reasoning about modal structure. This problem is particularly acute for reified temporal logics where the temporal framework is defined by a large number of axioms which generate a large search space. One approach is to use metareasoning during theorem proving. However we have adopted the more fundamental aim of defining a set of temporal axioms which specify the reified logic in such a way that the accessible worlds structure is encoded entirely within the second argument of HOLDS. The axioms which define the properties of the temporal logic, such as transitivity, modify the worlds structure argument only. Therefore in resolution-based theorem provers these axioms can be replaced by matching procedures. Our solution can be viewed as the extension of the method of Ohlbach (1988) to the temporal modality.

In the following sections we describe the design of three theorem provers all of which are based on the hyperresolution inference rule (Robinson, 1979, 1991). The first implements hyperresolution alone, the second additionally implements a rewriting procedure, and the third implements an additional matching procedure together with rewriting and hyperresolution. These three theorem provers have been implemented and their performance over a range of problems has been measured. We shall show that this empirical evidence supports our analysis of the alternative designs.

This paper continues, in Section 2, by presenting definitions of reified logics of belief, of branching time and of a multi-modal logic (QIL) which combines the epistemic and temporal modalities. The problem of theorem proving in modal logics is introduced in Section 3. In Section 4 the rewriting procedure is described and the process of reasoning about modality in the reified logic is contrasted with worlds-path method of Ohlbach (1988). Section 5 presents an alternative definition of the modal axioms where the modal semantics are encoded in a worlds-path. We also show how these axioms can be translated into a computational procedure. Section 6 presents experimental results which compare the performance of the methods.

2. Defining the Integrated Logic

Logical systems such as the modal logics and temporal logics are often defined axiomatically. A set of axioms together with inference rules and rules of necessitation (Hughes and Cresswell, 1968) define a particular logic. An alternative approach is to define the semantics of the modal or temporal logic (the object logic) in first-order logic. This is known as the reified approach. Examples of reified logics can be found in Moore (1985), Reichgelt (1989b) and Shoham (1986).

The reified approach requires all operators, predicates, variables and constants of the object, or reified, logic to be represented by functions or constants in the first-order language. This means that formulae of the reified logic are represented as domain terms of the first-order logic. The reified logic may be defined to correspond to a standard logic such as the propositional calculus or the modal system K. It can be shown that the expressivity of reified temporal logics is greater than that of the non-reified counterpart (see Reichgelt, 1989ab).

Throughout this paper we define the semantics of the logical operators in the reified language in terms of the standard first-order operators ($\wedge, \vee$ etc). The predicate *HOLDS* of the first-order language defines the truth or validity of (the reified representation of) a formulae of the object logic. The first-order language can be viewed as the metalanguage of the object logics we define.

The domain of the first-order logic is constructed from the following elements:

The set of functions:	and^2, or^2, if^2, bel^2, $poss^2$, $i\text{-}future^2$, $forall^2$, $exists^2$, not^1, $future^1$, $past^1$, $all\text{-}future^1$, $all\text{-}past^1$ to represent truth functional and modal operators
The set of functions:	$p^1, q^2 \ldots$ to represent predicates
The function:	$subst^3$ to represent the substitution of constants into terms
The set of constants *co*:	$a, b, c \ldots$ to represent constants of the object logic
The set of constants *d*:	$w_0, w_1 \ldots, ag_0, ag_1 \ldots, x, y \ldots$ to represent worlds, agent names and variables of the object logic.

The superscript indicates the arity of the function, the sets *co* and *d* are distinct, and their union defines the domain of the first-order logic.

The first-order logic has the standard interpretation with the modification that the following equivalence holds between terms in the domain: $subst(g(y,\ldots),x,y) = g(x,\ldots)$. The formula $P(subst(g(y),a,y))$ has the same interpretation as $P(g(a))$ where P is a predicate and $g(a)$ is the syntactic substitution of a for y in the term $g(y)$, for any constant y (y plays the role of a variable in the reified logic).

The semantics of the truth functional operators are defined as follows.[1]

TF1 $(\forall w)(\forall p)(\forall q)HOLDS(and(p,q),w) \leftrightarrow (HOLDS(p,w) \wedge HOLDS(q,w))$

TF2 $(\forall w)(\forall p)(\forall q)HOLDS(or(p,q),w) \leftrightarrow (HOLDS(p,w) \vee HOLDS(q,w))$

TF3 $(\forall w)(\forall p)(\forall q)HOLDS(if(p,q),w) \leftrightarrow (HOLDS(p,w) \rightarrow HOLDS(q,w))$

TF4 $(\forall w)(\forall p)HOLDS(not(p),w) \leftrightarrow \neg HOLDS(p,w)$.

These definitions are valid in all logics presented in this paper.

2.1. THE MODEL OF BELIEF

The model of belief is based on the K model of classical modal logic as defined by Kripke (1971). However this model is modified to account for the names of the believers (agents). The three place predicate R defines the accessibility relation. The first and third arguments of R denote worlds which are accessible from each other and the second argument denotes the name of the believer.

K $(\forall w)(\forall ag)(\forall p)HOLDS(bel(ag,p),w) \leftrightarrow (\forall u)(R(w,ag,u) \rightarrow HOLDS(p,u))$

P $(\forall w)(\forall ag)(\forall p)HOLDS(poss(ag,p),w) \leftrightarrow (\exists u)(R(w,ag,u) \wedge HOLDS(p,u))$.

Axioms which could be added to the K model of belief to give the usual extensions are:

D $(\forall w)(\forall ag)(\exists u)R(w,ag,u)$

4 $(\forall w)(\forall ag)(\forall u)(\forall v)(R(w,ag,u) \wedge R(u,ag,v)) \rightarrow R(w,ag,v)$

5 $(\forall w)(\forall ag)(\forall u)(\forall v)(R(w,ag,u) \wedge R(w,ag,v)) \rightarrow R(u,ag,v)$.

All theorems of the conventional KD model are true in the logic defined by K, P, D. Rule D specifies that R has the property of seriality which means that the quantification 'for all accessible worlds' specified in K can never be empty, or, equivalently that there is always some accessible belief world. Rules 4 and 5 define positive and negative introspection by specifying that the accessibility relation is transitive and euclidean respectively.

2.2. MODELS OF BRANCHING TIME

This section presents a branching model of time where different branches in time represent different possible futures. In this model (which is known as K_b in Rescher and Urquhart, 1971) different branches of time cannot joint up in the future. Time points on the same branch can be ordered but there is no ordering between time points on diverging branches. The model is defined in terms of a validity operator whose argument we may interpret as a logical world at point in time or simply as an accessible world in a worlds structure. These notions are equivalent. The accessible worlds structure is defined by the two place predicate T. The first

argument of T denotes a world which precedes the world denoted by the second argument. The branching model has the property of transitivity:

TR $(\forall t)(\forall u)(\forall v)(T(t,u) \wedge T(u,v)) \rightarrow T(t,v)$.

In the branching model of time, time points are not totally ordered but two time points on a particular branch can be ordered. This property is known as backwards linearity (and is defined by BLIN). In the simplest branching system it is not possible to refer to a specific branch of time. In order to represent the alternative branches explicitly we extend the basic logic by allowing a branch to be labelled. The label appears as an argument in the immediate-future (i-future) operator thereby connecting the temporal structure and the object language. The label can be interpreted as being the name of an action, and as we shall see at the end of this section, this gives a framework in which we can model planning.

The constraints on the temporal precedence relation T in BL, the core branching system, are defined by TR, BLIN and EQ.

BLIN $(\forall t)(\forall u)(\forall v)T(t,v) \wedge T(u,v) \rightarrow (T(t,u) \vee T(u,t) \vee equal(t,u))$

EQ $(\forall u)(\forall v)(\forall p)equal(u,v) \rightarrow (HOLDS(p,u) \leftrightarrow HOLDS(p,v))$.

The temporal operators are defined below.

AF $(\forall t)(\forall p)HOLDS(all\text{-}future(p),t) \leftrightarrow (\forall u)(T(t,u) \rightarrow HOLDS(p,u))$

F $(\forall t)(\forall p)HOLDS(future(p),t) \leftrightarrow (\exists u)(T(t,u) \wedge HOLDS(p,u))$

AP $(\forall t)(\forall p)HOLDS(all\text{-}past(p),t) \leftrightarrow (\forall u)(T(u,t) \rightarrow HOLDS(p,u))$

P $(\forall t)(\forall p)HOLDS(past(p),t) \leftrightarrow (\exists u)(T(u,t) \wedge HOLDS(p,u))$.

There are weak and strong versions of the temporal logic. The definition of the temporal modal operators is the same for both systems but the definition of the worlds framework varies. In the weak logic the *all-future*(p)/*all-past*(p) formulae do not imply that there are any future/past time points where p is true (there is an analogous definition of K) as can be seen from the definitions above.

In the weak model

all-future$(p) \wedge$ *all-future*$(not(p))$

can be satisfied if there are no future points in time. A stronger temporal model would specify that *all-future*(p) implies that there is some future time when p is true. We can obtain a strong logic by adding the axioms ST1, ST2. These axioms ensure that for all times t there is always a future and a past point in time. Therefore if *all-future*(p) is true at time $t0$ there is a point in time in the future of $t0$ (by ST1) and p is true at that point (by AF).

ST1 $(\forall t)(\exists u)T(t,u)$

ST2 $(\forall t)(\exists u)T(u,t)$

The branching system (BL) is defined by TF1–TF4, TR, EQ, BLIN plus the modal

operator definitions AF, F, AP, P. BL^{inf} is defined by adding ST1, ST2 to BL. Definitions of completeness are presented in Appendix 1.

Now we can extend the basic system. The logic BL^{e} is specified by adding BI and BIF to BL. BI further defines the branching temporal accessibility relation. The definition of *i-future* shows how the label of a particular branch is encoded into the temporal structure. If an action l occurs at a time point t then the next time point is named $s(l,t)$. The temporal logic is not extended by the induction axiom.

BL $(\forall u)(\forall l)T(u, s(l,u))$

BIF $(\forall t)(\forall l)(\forall p)HOLDS(i\text{-}future(l,p),t) \leftrightarrow (T(t,s(l,t)) \wedge HOLDS(p,s(l,t))$.

The motivation behind this particular formulation of a branching logic is the desire for a formal framework in which we can model planning. Such a framework must be able to represent propositions, actions and temporal concepts. We believe that the branching systems satisfy these needs. A brief discussion of the uses of the branching logic completes this section.

The label which appears as the first argument of *i-future* is intended to be the name of an action thus we may have:

$$if(forall(block1, forall(block2, if(and(clear(block1), clear(block2)),$$
$$i\text{-}future(stack(block1, block2), and(on(block1, block2), clear(block1)))))))$$

or

$$i\text{-}future(pc20\text{-}test, known\text{-}airway\text{-}reactivity)$$

(see Section 2.4 for quantifier definitions) where the first rule defines the action of stacking one block on another in the classic planning domain. The second rule defines the outcome of the pulmonary function diagnostic test, the PC-20 test, such a rule may be present in a knowledge based system in the pulmonary function domain.

For the sake of convenience in defining agent models an additional temporal operator *always* is defined (for example see the models in Aitken, 1992). This is not a new operator but is defined in terms of existing operators as follows:

$$(\forall w)(\forall p)HOLDS(always(p), w) \leftrightarrow HOLDS(and(p, all\text{-}future(p)), w).$$

The interpretation of *always* is 'now and all times future'. This operator is useful in defining rules and relationships which are true throughout time, the block-stacking rule given above is one such example.

2.3. THE INTEGRATION OF TIME AND BELIEF

Now we can combine models of time and models of belief as both have been defined in an accessible worlds framework.

Let W be a set of worlds, $v \epsilon W$ be the real world at an origin of time and let R, T be accessibility relations defined on W. The conditions placed on R correspond to the model of belief, the conditions placed on T depend on the model of time. The modal

structure is defined as: $\Theta = \langle v, W, R, T, P_1, P_2, \ldots \rangle$. The sets of worlds P_i define the worlds where the formula i is true.

We define a logical system which is an instance of Θ. In this model R has the property of seriality, that is we use a model of belief corresponding to the *KD* modal system and T imposes the backwards linearity ordering on (a subset of) W to implement the BL branching model of time. This system is defined by TF1–TF4, K, P, D, TR, BLIN, EQ, AF, F, AP, P, BI and BIF and we name it Integrated Logic (IL). Adding ST1, ST2 to IL defines IL^{inf} which contains the stronger temporal model of BL^{inf}. In all definitions of modal operators given in this section the worlds structure is explicitly represented in the metalanguage. We will refer to this as the modal-structural approach.

2.4. FIRST-ORDER MODELS

In this section we shall define a constant domain version of IL named QIL. We can define a varying domains model but we do not do so here.

By introducing an exists predicate (E) into the metalanguage we can reason explicitly about domain constants and their substitution. The *forall* operator can be defined as follows:

$$(\forall w)(\forall x)(\forall p)HOLDS(forall(x,p),w)$$
$$\leftrightarrow (\forall k)(E(k) \rightarrow HOLDS(subst(p,k,x),w)).$$

The sequence $subst(p,k,x)$ stands for the formula p with all occurrences of x replaced by k. This is a purely syntactic notion of substitution which is all that is required to define the common domain quantificational model. A constant domain of object language terms is associated with every world, it is a simple matter to define these terms as a type CO in the metalanguage such that:

$$(\forall x)(x \epsilon CO \leftrightarrow E(x))$$

The definitions of the *forall* and *exists* operators can now be more simply defined.

QU $(\forall w)(\forall x)(\forall p)HOLDS(forall(x,p),w)$
$$\leftrightarrow (\forall k : co)(HOLDS(subst(p,k,x),w)$$

QE $(\forall w)(\forall x)(\forall p)HOLDS(exists(x,p),w)$
$$\leftrightarrow (\exists k : co)(HOLDS(subst(p,k,x),w)).$$

Adding these definitions to TF1–TF4 gives the quantified logic QPC, adding them to IL gives QIL.

To define a varying domain quantificational model it is necessary to distinguish between the substitution into a predicate-function term (atomic) and substitution into a more complex term which consists of modal or truth functional functions. In the former case the *subst* function can be applied immediately, in the latter case the interpretation of reified modal formulae such as: $HOLDS(sub(future(p),a,x),w)$

must be defined explicitly e.g. by a rule such as:

$$(\forall w)(\forall x)(\forall p)HOLDS(sub(future(p), a, x), w)$$
$$\leftrightarrow (\exists u)(T(w, u) \wedge HOLDS(sub(p, a, x), u))$$
$$(\forall w)(\forall x)(\forall p)HOLDS(forall(x, p), w)$$
$$\leftrightarrow (\forall k)(E(k, w) \rightarrow HOLDS(sub(p, k, x), w))$$
$$(\forall w)(\forall x)(\forall p : atomic)HOLDS(sub(p, k, x), w)$$
$$\leftrightarrow (E(k, w) \wedge HOLDS(subst(p, k, x), w))$$
$$(\forall w)(\exists k)E(k, w).$$

In addition the quantifier elimination rules must be modified to introduce the function *sub*, as shown above, and the interpretation of *sub* is defined in terms of the *subst* function defined earlier. The remainder of this paper is concerned with constant domain modal logics.

3. Theorem Proving Methods for Modal Logics

Many theorem proving methods for modal logics are designed for particular classes of epistemic, temporal or dynamic logics. The most commonly used proof techniques are the tableau, matrix and resolution methods originally developed for first-order logic. In general these techniques cannot be immediately applied to modal logics. Additional inference rules are required to account for the modal context of a formula or literal.

Tableaux and sequent based proof methods for modal logics have been automated by Fitting (1988), Catach (1991) and Jackson and Reichgelt (1989). Proofs obtained by these methods have a more natural style than resolution style proofs as the methods were originally developed for human use. Of particular note is the TABLEAUX system of Catach who has extended the tableaux method to cover temporal logics. The matrix method of Wallen (1987) and the resolution methods of Farinas del Cerro (1985), Konolige (1986), Ohlbach (1988) and Auffray (1989) produce proofs of a less natural style reflecting their design as automated proof methods. These methods cover the modal systems D, S4, S5 plus K and its extensions[2] and as yet none has been extended to the temporal case.

It is not our central concern here to evaluate the strengths and weaknesses of the various approaches but rather to develop and assess one particular approach, the resolution method, as applied to reified logics. Perhaps by presenting some empirical data we shall be able to contribute to some future assessment of the alternative theorem proving methods. In this paper we present two modifications to a standard first-order theorem prover which reduce the search space and hence improve theorem proving performance. Unlike the direct approaches to theorem

proving (i.e. those methods mentioned in the preceeding paragraph), the modifications we propose are not made in order to account for the translation of the modal logic into first-order logic. Such translations are required in order that first-order proof methods can be applied to modal logics. We require no such translation because in the reified method the semantics of the object logic (the modal logic) is defined in first-order logic.

Theorem proving in any reified logic can be carried out using an un-modified first-order theorem prover. All that is required is the conversion of the definitional rules of the logic into clause form and the application of binary resolution. This implements a complete and correct proof procedure (assuming the first-order theorem prover to be complete). This naïve approach provides a method of theorem proving in all classes of object logics including the Integrated Logic. However there is a major problem with the naïve method. This is the massive search space generated by the logical axioms. In the following sections we identify two sources of this problem and propose two modifications to a standard first-order theorem prover which qualitatively reduce these problems.

The first modification we propose makes use of the fact that the rules defining the semantics of the modal operators are equivalence relations. This means that we can choose to replace all occurrences of all truth functional and modal functions by an equivalent and unique clause form. We shall show that this procedure eliminates the representational redundancy which makes a naïve approach infeasible.[3]

The second modification we propose requires the semantics of the object logics to be represented in a more computationally efficient manner than in the definitions of Section 2. The intuition is that a worlds-path argument can be introduced into the *HOLDS* predicate to replace the predicates R and T of the metalanguage. The use of a worlds-path construction is common to many of the direct methods of modal theorem proving in belief logics, especially Ohlbach (1988). We apply this device to the modalities of time and belief, and consequently to the Integrated Logic.

The remainder of this section describes the worlds-path method of Ohlbach in order to illustrate the central ideas and to serve as a reference for an analysis of proof search space in reified logics.

3.1. ENCODING MODALITY BY A WORLDS-PATH

The worlds-path method of encoding modal context is defined by the rules given below.[4] The worlds-path is denoted by the superscript of a formula. The symbol : stands for the *append* function i.e. $[0] : [w] = [0, w]$. The successive application of these rules (and others which account for quantifiers) results in formulae where all modal operators have been eliminated. Theorem proving can be carried out by resolution, plus additional unification algorithms which determine whether the

worlds-paths of two literals being matched can be unified. Matching requires the unification of the variables of the formulae and the unification of the worlds-path.

$$(\forall\alpha) \models bel(p)^{\alpha} \leftrightarrow (\forall w)(\models p^{\alpha:[w]})$$

$$(\forall\alpha) \models poss(p)^{\alpha} \leftrightarrow (\exists w)(\models p^{\alpha:[w]})$$

$$(\forall\alpha)(\forall p)(\forall q) \models or(p,q)^{\alpha} \leftrightarrow (\models p^{\alpha} \vee \models q^{\alpha})$$

$$(\forall\alpha)(\forall p) \models \neg p^{\alpha} \leftrightarrow \neg \models p^{\alpha}.$$

The superscript denotes the worlds-path.

No additional worlds-path unification steps are necessary in order to implement the KD model of belief. This is because the condition of seriality which is imposed on the accessibility relation is correctly modelled by the universal quantifier where $(\forall w)p \rightarrow (\exists w)p$ for any p. It is a simple matter to deduce the D axiom:

$$(\forall\alpha) \models bel(p)^{\alpha} \rightarrow \models poss(p)^{\alpha}$$

for all p.

In the KD4 system the accessibility relation is both serial and transitive. Transitivity is implemented by mapping elements of one worlds-path onto elements of another. For example:

If $\alpha = [0, a, b]$ (where a and b are constants)

and $\beta = [0, w]$ (where w is a variable)

then the mappings $[0] \rightarrow [0]$ and $[a, b] \rightarrow [w]$ unify the worlds-paths α and β. A set of such mappings can be defined for most belief logics (KD and stronger normal systems) Ohlbach (1988) and it is these mappings which are implemented by the worlds-paths unification algorithm. The theorem proving problem is made more complex by the additional unification steps but this method does not induce an expansion of the search space in order to reason about the modal structure.

4. Making Theorem Proving Feasible by a Rewriting Procedure

In this section we describe a theorem proving method which is applicable to all reified systems. This method has two stages, the first can be implemented as a rewriting procedure which eliminates all reified operators from the database. This procedure is applied prior to the second stage, which is theorem proving proper. We describe two implementations of the operator-elimination procedure and show that completeness is maintained. We compare the proposed two stage method with worlds-path method and analyse the differences between the two approaches.

Taken together, the rules which define the semantics of the logical operators define a large search space most of which is redundant. This is a redundancy in the representation of the disjunction, conjunction and worlds model. For example

(i) $HOLDS(and(p, bel(ag, q)), w)$ may be represented as
(ii) $HOLDS(p, w) \wedge HOLDS(bel(ag, q), w)$ or
(iii) $HOLDS(p, w) \wedge (\neg R(w, ag, u) \vee HOLDS(q, u))$.

The situation is actually much worse as for any formula p, for all formulae q, if $HOLDS(p, w)$ is true then $HOLDS(or(p, q), w))$ is true. The search space clearly contains redundancies as many of the different representations of a formula are equivalent, in addition many are logically weaker.

We begin by defining *data* as formulae containing *HOLDS* predicates which take a ground proposition as an argument or in the quantified case take a reified formula which has a ground predicate symbol. By this definition the formulae (i) to (iii) above are data while the meaning postulates of the previous section contain no ground terms and are not considered as data. The general approach we adopt to obtaining proofs is to compute the extension of the logical model defined by the data formulae in order to demonstrate a contradiction.

We can choose to eliminate all object language operators from all data to obtain the most expanded form of the proposition. As the expanded form is obtained from the original data by applying equivalence rules the expanded form and the original data are equivalent. Theorem proving can be carried out on this expanded (or rewritten) dataset without the need for the operator definition clauses to be present in the database. Completeness is maintained and the search space is reduced as we shall now show by considering a procedure which employs resolution. A procedural rewriting method can achieve the same purpose. The derivation of the rewriting algorithm is also presented below.

4.1. THE REWRITING PROCEDURE

Let DC be the set of clauses obtained by converting the definitional rules of a reified logic into clause form. Let OC be the data set, also in clause form. The aim of resolution is to derive a contradiction from DC ∪ OC. DC consists of two types of rules, equivalence rules which define an operator, ER, and modal rules, MR. The set DC is consistent – applying resolution to rules within this set will never derive a contradiction. By noting that the set ER are equivalence relations we propose the following two stage resolution procedure.

The first stage is to define the set OC^e by the resolution of one clause from ER with one clause from OC or OC^e. The resolution strategy of this stage ensures that each application of binary resolution eliminates an operator from OC^e.[5] This procedure terminates when all operators have been eliminated. By the arguments above OC and OC^e are equivalent and resolutions among ER are unnecessary. The second stage of theorem proving applies resolution to OC^e ∪ MR. The advantages of this method are that the set ER has been eliminated from the database during the main theorem proving stage and that completeness is maintained.

The first stage need not be implemented by resolution, a rewriting algorithm can be derived automatically from ER. The rewriting algorithm performs two functions. First, it eliminates all occurrences of reified functions. That is, after rewriting, the database consists of sentences from which the reified functions such as *and*, *or*, *past* etc have been eliminated. Secondly, it converts input sentences to clausal form.

The rewriting procedure consists of two subprocedures, one for rewriting positive *HOLDS* literals, and one for rewriting negative *HOLDS* literals, called *rw* and *nrw* respectively. Both are constructed automatically from the meaning postulates defining the semantics of the reified functions. In order to construct a rewriting procedure, we first convert those meaning postulates that define a reified function into clausal form. The resulting clauses are then ordered so that the leftmost literal in each is either the negative or the positive literal which was the antecedent of the original meaning postulate. Since each meaning postulate defining the meaning of a reified function will always have exactly one *HOLDS* predicate in its left-hand side, the existence of such a literal is guaranteed for each clause. Moreover, if a meaning postulate yields more than one clause, the literal in question will be identical for each clause. Thus, for example, TF1 will yield the following two sets of clauses:

$$\{[\neg HOLDS(and(p,q),v), HOLDS(p,v)]$$
$$[\neg HOLDS(and(p,q),v), HOLDS(q,v)]\}$$
$$\{[HOLDS(and(p,q),v), \neg HOLDS(p,v), \neg HOLDS(q,v)]\}$$

These sets of clauses can then be used to generate a separate statement in the definition of the rewriting algorithm. Each set of clauses whose first literal is negative yields an additional statement in the definition of the *nrw* sub-routine, while each set of clauses whose first literal is positive yields one for *rw*. The right-hand side of the clause is obtained as follows. First, delete the left-most literal from each clause, i.e. the *HOLDS* literal defining the reified function. Then, replace each negative literal $\neg p$ by $rw(p)$ and each positive literal by $nrw(p)$. For each clause, if there is more than one expression remaining, construct a statement applying the function *append* to each rewritten literal. Finally, if there is more than one such *append* statement, then form a new statement applying the function *merge* to the different append statements; otherwise simply return the *append* statement.

The functions *merge* and *append* act as expected. They both take lists of clauses as arguments. The function *merge* forms new clauses by combining clauses in the first argument list with those in the second list. The function *append* simply appends two lists of clauses. The effect of these functions is to distribute disjunction and conjunction over the clauses generated by subsequent *rw* or *nrw* operations.

To illustrate the generation of a rewriting procedure, consider the sets of clauses generated from TF1. After applying the first step, we have:

$$\{[HOLDS(p,v)][HOLDS(q,v)]\}$$
$$\{[\neg HOLDS(p,v), \neg HOLDS(q,v)]\}.$$

The first set of clauses will generate the right-hand side for $nrw(HOLDS(and(p,q),v))$, while the second one will generate that for $rw(HOLDS(and(p,q),v))$. The second step yields:

$$\{[nrw(HOLDS(p,v))][nrw(HOLDS(q,v))]\}$$
$$\{[rw(HOLDS(p,v)), rw(HOLDS(q,v))]\}$$

which after applying the final steps, result in:

$$merge(nrw(HOLDS(p,v)), nrw(HOLDS(q,v)))$$

$$append(rw(HOLDS(p,v)), rw(HOLDS(q,v))).$$

In addition to the clauses for *rw* and *nrw* defined in this way, each rewriting algorithm contains two further clauses, which are in fact the first ones to be applied, namely:

$$rw(HOLDS(p,v)) = \{[HOLDS(p,v)]\}$$

if p is the name of an atomic proposition.

$$nrw(HOLDS(p,v)) = \{[\neg HOLDS(p,v)]\}$$

if p is the name of an atomic proposition.

The full rewriting algorithm for any reified logic is obtained by applying the above procedure to the definitional rules.

The Two-Stage resolution procedure is independent of the object logic under consideration. It is applicable to the belief logics, temporal logics and the Integrated Logics. The 'architecture' of this approach is derived from the properties of the rules defining the reified logics. Experiments with Two-Stage theorem provers have shown that this method makes the theorem proving problem feasible, the experimental results are presented in Section 6.

4.2. COMPARISON OF THE WORLDS-PATH AND REWRITING METHODS

Comparing proofs employing the reified method with those employing the worlds-path construction reveals that there is a greater branching of the search space in the reified method. As an example consider the proof of $bel(p)$ from a database consisting of this formula alone. The proof procedure is to add $\neg bel(p)$ to the database and derive the empty clause $\square$, as illustrated in Table I.

The explicit representation of the R predicate, in the Two-Stage reified method, causes the search space to branch to a greater degree than in the worlds-path method. For example atomic clause (ii) will resolve with all clauses containing the

Table I. Comparison of proof methods.

	Worlds-path method	Two-Stage reified method
Data	a) $\{\models p^{[0,w]}\}$	i) $\{\neg R(0,a,w) HOLDS(p,w)\}$
Goal	b) $\{\models \neg p^{[0,c]}\}$	ii) $\{R(0,a,c)\}$ iii) $\{\neg HOLDS(p,c)\}$
Inference Steps	$\square[c/w]$ by a + b	iv) $\{HOLDS(p,c)\}[c/w]$ by i + ii $\square$ by iv + iii
Computation	R = 1, U = 2	R = 2, U = 5

Key: R = the number of applications of resolution, U = the total number of variables unified.

literal $\neg R(0, a, w)$. This literal occurs in all clauses derived from statements about a's beliefs. The result of applying binary resolution is to derive new clauses concerning the truth of *HOLDS* literals at world c for all clauses concerning a. A contradiction will only be deduced by resolutions involving the other goal clause (iii). Hence many unnecessary computations are made. The equivalent step of modal reasoning in the worlds-path method is the unification of terms in the worlds-path, but this does not happen until matching propositions have been identified and therefore the problem does not arise.

In the above example there are no additional axioms defining the worlds structure, however we must consider the cases where they do occur. The clause $\{R(x, y, f(x))\}$ defines KD, the serial extension of K. Adding this clause to the set (i)–(iii) listed above expands the search space yet more. In the temporal modality, reasoning about the modal structure is even more problematic since the temporal axioms themselves define a large search space.

Our conclusion is that the Two-Stage method has made automated proof in reified logics a possibility, through the elimination of representational redundancy. The problem of reasoning about modal structure in an efficient manner is not solved and this approach compares unfavourably with the worlds-path approaches in this respect. This problem will be tackled in the following section.

5. Encoding Modal Semantics in Terms of Worlds-Paths

We can combine the reified method of defining modal logics with the idea of encoding the modal structure in a worlds-path. The central idea is to complicate the world argument of *HOLDS*( ) by making it a more complex construction, in order to remove the explicit representation of worlds structure. The idea of a worlds-path is common to many of the syntactic methods of modal resolution. In this section we redefine the reified logics making use of the worlds-path to eliminate the R and T predicates from the first-order language. Previously the syntactic methods have used this device to derive efficient belief logics. We show how this device can also be used to replace T in the strong temporal logics. All of the systems presented in this section are extensions of common domain object language defined by TF1–TF4, QU, QE.

This section begins by defining three logical models in terms of a worlds-path index. The rules defining these logics include those which eliminate an operator of the object language, as was the case previously. In addition we define rules which modify the worlds-path index of formulae and inference rules. The next step is to rewrite these definitions so that they can apply to atomic formulae only (i.e. literals in a clause) and the logic remain complete. Having achieved this, the index modification rules can be implemented as unification rules and the inference rules as resolution rules.

5.1. THE MODEL OF BELIEF

We must encode the semantics of all accessible worlds (where there is at least one accessible world) in the metalanguage. In contrast to the definitions K and P of Section 2.1 the predicate R must not appear in the metalanguage for the reasons given above. This is achieved by employing the exact correspondence between this notion of 'all accessible worlds' and $\forall x$. This approach is common to Ohlbach (1988), Jackson and Reichgelt (1987) and has the consequence that the model of belief is based on KD. The argument of the *HOLDS*( ) operator becomes a list construction with the following convention for writing lists:

Let $\cdot$ be a list construction function then $a \cdot b = \cdot(a, b)$, and $a \cdot b \cdot c = \cdot(a, \cdot(b, c))$ where the left expressions are shorthand for the linear lists on the right. We shall refer to these constructions as index-lists.

The KD model of belief is defined by operator definition rules B and PB in addition to TF1–TF4, QU, QE.

$$\text{B} \quad (\forall i)(\forall ag)(\forall p)HOLDS(bel(ag,p),i) \leftrightarrow (\forall j : w)(HOLDS(p, j \cdot ag \cdot i))$$

$$\text{PB} \quad (\forall i)(\forall ag)(\forall p)HOLDS(poss(ag,p),i) \leftrightarrow (\exists j : w)(HOLDS(p, j \cdot ag \cdot i))$$

The definitions make use of a type theory where w is the type 'worlds'. The index-list is built up from right to left. This is because it is easier to add a first element to a list than to add a last element while keeping the list linear. For example $\cdot(c, \cdot(b, a))$ plus d as a first element $= \cdot(d, \cdot(c, \cdot(b, a)))$ where the $\cdot(\ )$ function is applied to both existing structures. Adding d as the last element means finding the last element and applying the $\cdot(\ )$ function – all these operations would have to be defined in the logic.

If we try to implement a K model by the worlds-path method then we could modify the definitions of P and PB to account for the case where there is no accessible world, as below.

$$(\forall i)(\forall ag)(\forall p)HOLDS(bel(ag,p),i)$$
$$\leftrightarrow (\forall j : w)(HOLDS(p, j \cdot ag \cdot i) \vee End(ag,i)).$$

Alternatively the *End* predicate can be introduced into the definition of the unification procedure (Ohlbach (1988)). Both approaches involve explicit reasoning about the worlds structure therefore neither approach is a significant improvement on the definitions K, P in Section 2.1. We conclude that the worlds-path approach is an improvement on the modal-structural approach only in the case where the core modal system is KD i.e. where the correspondence between the modal operators and $\forall, \exists$ is exact.

In the modal-structural approach the extensions of KD were defined by logical rules constraining R. In the worlds-path approach this is not possible as R has been eliminated from the axiomatisation. An alternative way to define the extensions of KD is to add the axioms for 4, 5 as rules in the metalanguage.[6] This

solution satisfies all formal semantic considerations but we must also take implementational factors into account. Rules 4 and 5 appear to have adverse implications for the efficiency of theorem proving. If these rules were converted into clause form and added to the database then the problems of representational redundancy would reappear. However we can look at these rules from another perspective: the effect of axioms 4 and 5 is to modify the index-list and to leave the reified proposition unaltered, therefore we can view these rules as specifying computational procedures which can be built into the theorem prover. Once these procedures have been defined there is no need to retain the clause form of the axioms in the database.

Our solution to the problem of reasoning about modal structure has two components: the first is the definition of the semantics of the modal operators (rules B and PB above) and the definition of rules which modify the worlds-path to specify particular modal properties. The second component is the mapping of the two sets of logical rules into rewriting procedures and index-list unification procedures respectively.

As the **logical** implication operator and the **procedural** implication operator of computer languages have different semantics, the definition of the extensions 4, 5 of KD must be modified in order to account for these differences.

We shall refer to 'path rules' as those rules which are part of the definition of the logic and are of the format: $HOLDS(p,i) \rightarrow HOLDS(p,j)$. The extensions of KD are defined by operator definition rules and path rules, KD is defined by the operator definition rules alone. The path rules must be implemented as a procedure in the theorem prover. The implementation must be derived from the path rules and must be equivalent to them. Therefore the semantics of the operators in the path rules must match the semantics of the operators in the procedure. A procedural implication operator $\Rightarrow$ is introduced. The $\Rightarrow$ operator has the following semantics:

$$P \wedge (P \Rightarrow Q) \rightarrow Q$$

The $\Rightarrow$ operator is weaker than standard implication as from $(P \Rightarrow Q)$ and $\neg Q$ it is not possible to deduce $\neg P$. That is, the procedural implication operator can only be applied in the forwards direction. This is exactly the interpretation that the rules of the implemented unification procedure will have.

From the above considerations the path rules must be of the form $HOLDS(p,i) \Rightarrow HOLDS(p,j)$. These rules must also specify a consistent strategy for modifying the index-list. The most obvious strategy is that the length of the index-list of the conclusion must be shorter than, or equal to, the length of the index-list of the condition. If this is not the case then the procedure of applying path rules may not terminate as a rule increasing the length of the index-list may be recursively applied.

The following axiom sets are used in place of the standard definitions of 4 and 5 in

the modal systems KD4 and KD5 respectively.

$4'$ $\Diamond\Diamond P \Rightarrow \Diamond P$

$5'$ $\Diamond\Box P \Rightarrow \Box P$

$5''$ $\Diamond\Box P \Rightarrow \Box\Box P$

They are defined in terms of procedural implication and no rule increases the length of the modal path. These axioms have the desired properties of path rules and are translated into the index-list formulation below.

$$\text{E4}' \quad (\forall l : w)(\forall k : w)(\exists j : w)(\forall i)(\forall ag)(\forall p)HOLDS(p, k \cdot ag \cdot l \cdot ag \cdot i) \Rightarrow HOLDS(p, j \cdot ag \cdot i)$$

$$\text{E5}' \quad (\forall l : w)(\forall j : w)(\exists k : w)(\forall i)(\forall ag)(\forall p)HOLDS(p, k \cdot ag \cdot l \cdot ag \cdot i) \Rightarrow HOLDS(p, j \cdot ag \cdot i)$$

$$\text{E5}'' \quad (\forall l : w)(\forall h : w)(\forall j : w)(\exists k : w)(\forall i)(\forall ag)(\forall p)HOLDS(p, k \cdot ag \cdot l \cdot ag \cdot i) \Rightarrow HOLDS(p, h \cdot ag \cdot j \cdot ag \cdot i).$$

The specification of KD5 by two additional axioms, instead of one, arises from the interaction of 5 with itself i.e. $\Diamond\Box P \rightarrow \Box\Diamond\Box P \rightarrow \Box\Box P$ by two applications of 5. This derivation requires an increase in the degree of modality or, in terms of index-lists, the introduction of a variable of type w in the first step of the deduction. We remove the need to introduce such variables by the particular choice of axioms E5′, E5″.

The derivation of rewrite rules from B and PB is straightforward (see Section 4) the derivation of a procedural algorithm from path rules is presented in Section 5.3. In contrast with the method of Ohlbach (1988) it is not guaranteed that the axiom set defining a transitive relation among accessible worlds can simply be added to an axiom set defining symmetry, euclideanness or reflexivity in order to define an accessibility relation which combines several properties. The interaction of these properties must be accounted for, as shall be illustrated in the following section.

5.2. THE BRANCHING MODEL OF TIME

We wish to retain the efficient features of the worlds-path approach in our redefinition of the branching temporal logic. Consequently we restrict our treatment to the strong branching system. The basic approach is to add terms to the worlds-path which stand for 'all points in time in the future' and 'some point in time in the future'. The past time operators are treated similarly, however they are replaced by terms defined as a different type to future-path terms in order to avoid confusion.

Let the types *ft* and *pt* be future and past time points respectively (both types are distinct from *w*). Future time points may also be denoted by domain terms (type *co*) as these terms are used as arguments of the *i-future* operator and therefore must be

encoded into the worlds-path. Let $tt = pt \cup ft$. In the following rules we define the modal operators of BL^e. If we wished to implement BL^{inf} we would delete rule EBI.

EAF $(\forall i)(\forall p)HOLDS(all\text{-}future(p), i) \leftrightarrow (\forall u : ft)(HOLDS(p, u \cdot i))$

EF $(\forall i)(\forall p)HOLDS(future(p), i) \leftrightarrow (\exists u : ft)(HOLDS(p, u \cdot i))$

EAP $(\forall i)(\forall p)HOLDS(all\text{-}past(p), i) \leftrightarrow (\forall \alpha : pt)(HOLDS(p, \alpha \cdot i))$

EP $(\forall i)(\forall p)HOLDS(past(p), i) \leftrightarrow (\exists \alpha : pt)(HOLDS(p, \alpha \cdot i))$

EBI $(\forall i)(\forall l : co)(\forall p)HOLDS(i\text{-}future(l, p), i) \leftrightarrow HOLDS(p, l \cdot i).$

The temporal model must have the properties of partial ordering of time points and transitivity. We can model transitivity by the combination of terms in the index-list. The partial ordering of time points can also be implemented in part by path rules however we must also define an inference rule (PO). The fact that we must implement an additional inference rule in our theorem prover is a source of inefficiency. However this situation is still an improvement on the previous approach where there were 6 temporal inference rules whose use had to be controlled.

As was stated previously the path rules must satisfy a set of constraints. The rules EB1–EB7 meet all conditions and together with EAF, AF, EAP, AP, EBI and PO define a complete logic of branching time.

EB1 $(\forall x : ft)(\forall y : ft)(\exists z : ft)(\forall p)(\forall i)(HOLDS(p, y \cdot x \cdot i)$
$\Rightarrow HOLDS(p, z \cdot i))$

EB2 $(\forall \alpha : pt)(\forall \beta : pt)(\exists \gamma : pt)(\forall p)(\forall i)(HOLDS(p, \beta \cdot \alpha \cdot i)$
$\Rightarrow HOLDS(p, \gamma \cdot i))$

EB3 $(\forall \alpha : pt)(\exists x : ft)(\forall p)(\forall i)(HOLDS(p, x \cdot \alpha \cdot i) \Rightarrow HOLDS(p, i))$

EB4 $(\forall x : ft)(\exists \alpha : pt)(\forall p)(\forall i)(HOLDS(p, \alpha \cdot x \cdot i) \Rightarrow HOLDS(p, i))$

EB5 $(\forall \alpha : pt)(\forall y : ft)(\exists x : ft)(\forall p)(\forall i)(HOLDS(p, x \cdot \alpha \cdot i)$
$\Rightarrow HOLDS(p, y \cdot i))$

EB6 $(\forall x : ft)(\forall \beta : pt)(\exists \alpha : pt)(\forall p)(\forall i)(HOLDS(p, \alpha \cdot x \cdot i)$
$\Rightarrow HOLDS(p, \beta \cdot i))$

EB7 $(\forall x : co)(\exists y : ft)(\forall p)(\forall i)(HOLDS(p, x \cdot i) \Rightarrow HOLDS(p, y \cdot i))$

PO $(\forall i)(\forall \alpha : pt)(\forall \beta : pt)(\exists \gamma : pt)(\exists \delta : pt)(\forall p)(\forall q)(\forall r)$
$(\neg HOLDS(p, \alpha.i) \vee \neg HOLDS(q, \beta.i) \vee \neg HOLDS(r, \alpha.i)$
$HOLDS(p, \beta.i) \vee HOLDS(q, \delta.\alpha.i) \vee HOLDS(r, \gamma.\beta.i))$

Rules EB1, EB2 define the property of transitivity and correspond to T4 and T8 of

the axiom set as defined in Appendix 1. The rules EB3–EB6 define the interaction between the past and future operators. The inferences that *past*(*all-future*(p)) implies p and *future*(*all-past*(p)) implies p are defined by EB3, EB4 respectively. These rules correspond to the axioms T3 and T7.

The rule EB5 defines the inference *past*(*all-future*(p)) implies *all-future*(p) and EB6 defines *future*(*all-past*(p)) implies *all-past*(p). These rules arise from the interaction of transitivity and the sequences *past*(*all-future*(p)), *future*(*all-past*(p)). The deductions they define are implicit in the axiom set (they can be derived from T3, T4, T7, T8 or from EB1–EB4 if $\Rightarrow$ is replaced by $\rightarrow$). All deductions which could be made by the interaction of the converses of EB1–EB4 with the set EB1–EB4 itself, are no longer valid due to the use of the $\Rightarrow$ operator. Therefore these deductions must be explicitly defined by path rules. Rule PO is derived from the following two rules:

$$(\forall i)(\forall \alpha : pt)(\forall \beta : pt)(\exists \gamma : pt)(\exists \delta : pt)(\forall p)$$
$$((HOLDS(p, \alpha.i) \wedge \neg HOLDS(p, \beta.i)) \rightarrow \alpha.i = \gamma.\beta.i \vee \beta.i = \delta.\alpha.i)$$
$$(\forall x : tt)(\forall y : tt)(\forall p)(\forall i)(x = y \rightarrow (HOLDS(p, x) \rightarrow HOLDS(p, y))).$$

The first rule states that if p is true sometime α in the past of i and p is false sometime β in the past of i then either α precedes β or β precedes α. The right hand side of this rule is refuted if both the assumptions $\alpha = \gamma \cdot \beta$ and $\beta = \delta \cdot \alpha$ can be refuted. The second rule connects equality between points in time with the truth of reified formula at the named points. We assume that no other rules include the equality predicate which allows us to combine these rules in order to eliminate this predicate. The result after some simplification is PO, a rule which has little intuitive value but which serves to complete the definition of the branching logic.

The path rules which define the temporal model must ultimately be implemented in a resolution algorithm. They must define a procedure which terminates. This is guaranteed as each application of a path rule EB1–EB7 reduces the length of the (finite) index-list therefore at some point the list is either reduced to length 1, or no path rule is applicable. In both cases the procedure terminates. Another consequence of implementing the path rules as part of a resolution inference procedure is the need to restrict the quantification of p to the case where p is atomic. The path rules given above must be transformed and a method for doing this is given in Section 5.3.

The component models of belief and branching time can again be combined to obtain the strong Integrated Logic IL^{inf}.

5.3. SPECIFYING THE INDEX-LIST UNIFICATION ALGORITHM

Index-list unification can be viewed as an extra condition which must be satisfied in order for the matching of two literals to succeed. This can be expressed by the

resolution rule (RR) given below in which the predicate *IL-UNIFY*(i, j) is true where the index-lists i and j can be unified.

RR $(\forall p : atomic)(\forall i)(\forall j)$

$$((HOLDS(p,i) \vee A) \wedge (\neg HOLDS(p,j) \vee B) \wedge IL\text{-}UNIFY(i,j))$$

$$\rightarrow (A \vee B)$$

where A and B are disjunctions

Rule RR shows how index-list unification is integrated into the binary resolution rule. The predicate *IL-UNIFY* defines the metalevel concept of unifiable worlds-paths, just as the *HOLDS* predicate defines the metalevel concept of truth at a possible world.

We must now define the conditions when *IL-UNIFY*(i, j) is true. These must be precisely the conditions defined by the path rules EB1–EB7. The path rules cannot be directly translated into statements in the unification algorithm without accounting for the restriction in the quantification of the variable p in RR to reified atomic formulae. The problem is that while a modal formulae may be substituted for p in any of rules EB1–EB7, in the theorem prover all modal operators will be eliminated by the rewriting procedure. Hence we may have an index-list such as $v \cdot ag \cdot x \cdot y$ where v is of type world (w), ag is an agent name and x, y are future time points, in which case rule EB1 is no longer applicable which would be incorrect. The solution is to view the sequence $x \cdot y$ as a sub-path of the index-list and redefine EB1 to allow the deduction of:

$$(\exists z : ft)HOLDS(p, h \cdot z \cdot i) \text{ from } HOLDS(p, h \cdot y \cdot x \cdot i)$$

for any values of h, i or p. The pattern of quantification of EB1 specifies that z is dependent upon x and y alone and hence rule EB1 can be interpreted as defining a modification to any sub-path of any reified formulae, that is, we can now view index-list unification as a process which is independent of reified formulae. This can be expressed as follows:

$$EB1'(\forall x : ft)(\forall y : ft)(\exists z : ft)(\forall h)(\forall i)\langle h \,|\, y \cdot x \,|\, i \rangle \Rightarrow \langle h \,|\, z \,|\, i \rangle$$

where $\langle\ \rangle$ represents an index-list and $\|$ represents a sub-path.

This transformation of the path rules allows the statements of the unification algorithm to be derived in a straightforward manner. The algorithm given below defines the conditions under which sub-paths can be unified as the derivation of *TERMN* from *SUB-PATH*(i, j) (UT1, UT2). The algorithm defines that unifiable sub-paths can be combined to obtain unifiable index-lists (U1, U2). Each path rule contributes two statements to the definition of the sub-path algorithm. Statements TL1, TR1 are derived from Rule EB1′ and define that EB1′ may be applied to either of the

index-lists being tested for unification.

U1 $(\forall x)(\forall y)(SUB\text{-}PATH(x,y) \Rightarrow TERMN) \rightarrow IL\text{-}UNIFY(x,y)$

U2 $(\forall x)(\forall y)(\forall u)(\forall v)(IL\text{-}UNIFY(x,y) \wedge IL\text{-}UNIFY(u,v))$
$\rightarrow IL\text{-}UNIFY(x \mid u, y \mid v)$

UT1 $(\forall x)SUB\text{-}PATH(x,x) \Rightarrow TERMN$

UT2 $(\forall x)(\forall y)SUB\text{-}PATH(x \cdot y, x \cdot y) \Rightarrow TERMN$

TL1 $(\forall y : ft)(\forall x : ft)(\exists z : ft)(\forall i)SUB\text{-}PATH(y \cdot x, i) \Rightarrow SUB\text{-}PATH(z,i)$

TR1 $(\forall y : ft)(\forall x : ft)(\exists z : ft)(\forall i)SUB\text{-}PATH(i, y \cdot x) \Rightarrow SUB\text{-}PATH(i,z)$

Each of the path rules EB1–EB7 adds two sub-path modification rules to the unification algorithm. There are fourteen rules which may be selected in order to show that two paths unify and there may be more than one substitution under which two literals resolve. For example:

$HOLDS(p, c \cdot b \cdot a \cdot i)$ and $\neg HOLDS(p, y \cdot x \cdot i)$

can be resolved under the substitutions

$\{[c/y], [g(b,a)/x]\}$ or $\{[g(c,b)/y], [a/x]\}$

where a, b, c are constants of type ft, x, y are variables of type ft and g is the skolem function introduced by TL1. This is the case because both $IL\text{-}UNIFY(c \cdot b \cdot a \cdot i, g(c,b) \cdot a \cdot i)$ and $IL\text{-}UNIFY(c \cdot b \cdot a \cdot i, c \cdot g(b,a) \cdot i)$ are true.

The search space defined by the path rules may seem large but in fact it is limited as the type theory and the pattern of quantification restrict the number of rules which actually apply for any particular pair of index-lists.

6. Theorem Proving

There is a common theorem proving strategy which is applicable to each logic described in section 5. The three steps in the strategy correspond to the different functions of the reified rules defining the logical system.

STEP 1: The rules defining the semantics of the reified operators are applied exhaustively to the domain language formulae. All operators are eliminated to give a data set in clausal form which is equivalent to the original data.

STEP 2: The binary resolution rule is applied. This involves index-list unification by procedural rules derived from the path rules which define the logic.

STEP 3: If the definition of the logic includes inference rules then at some point in the search a meta-control decision must be made to apply an inference rule.

For KD and its extensions there is no third step to the theorem proving process. For the temporal logics the application of the inference rule is necessary in order to construct proofs which depend on the total or partial ordering of time points. There is a

natural division between simple temporal reasoning and more complex inferences derived from the ordering theorems. The former includes reasoning about transitivity and is implemented by unification, the latter requires the inference rules to be applied. This division can be used in the meta-control strategy. If the branching logic is restricted to future time operators then step 3 can be eliminated from the theorem proving procedure. This would implement a system with enough expressivity to model planning with a more efficient proof procedure than the complete logic.

Now we turn to the question of the efficiency of the theorem proving methods. We shall examine one aspect of this issue: that of the size of search space generated by hyperresolution. The two measures that we are interested in are the number of clauses generated during the proof and the number of applications of the hyper-resolution rule required in the derivation of the empty clause. The former measure indicates the magnitude of the search space, the latter indicates the depth of search required. The problems that we consider in this study are listed in Table II, they are a subset of the formulae which define completeness in the propositional calculus, belief logic and temporal logic (see Appendix 1 for the full set). Formula PC1 defines a fundamental truth functional relationship in the propositional calculus. B1 is true by the definitions of the belief and possibility operators while B2 defines a truth functional relationship which holds among beliefs. T4 and T7 hold because the temporal structure is transitive and T5 defines the property of backwards linearity. Each of these formulae illustrate a particular property of the reified logics. The other formulae of the completeness-set depend on similar properties: the definition of the operators, transitivity or the combination of modal and truth functional reasoning, therefore the 6 formulae of Table II are representative of the 18 which specify completeness in IL. The experimental results are presented in Table III.

The results show that as the definitional rules of the reified logics are replaced by rewriting and unification procedures the initial number of clauses in the database is reduced. As was stated earlier, this is one of our aims. The Naïve theorem prover was unable to solve any of the problems in this trial, in fact no solution was found to PC1 within the first 14,000 clauses (however the Naïve Method was able to solve simpler

Table II. Problem set.

PC1	$(\forall w)(\forall p)HOLDS(if(or(p,p),p),w)$
B1	$(\forall w)(\forall ag)(\forall p)(HOLDS(poss(ag,p),w) \rightarrow HOLDS(not(bel(ag,not(p))),w))$
B2	$(\forall w)(\forall ag)(\forall p)(\forall q)(HOLDS(bel(ag,if(p,q)),w) \rightarrow (HOLDS(bel(ag,p),w) \rightarrow HOLDS(bel(ag,q),w)))$
T4	$(\forall w)(\forall p)(HOLDS(all\text{-}past(p),w) \rightarrow HOLDS(all\text{-}past(all\text{-}past(p)),w))$
T5	$(\forall w)(\forall p)(\forall q)(HOLDS(and(past(p),past(q)),w) \rightarrow (HOLDS(past(and(p,q)),w) \vee HOLDS(past(and(p,past(q))),w) \vee HOLDS(past(and(q,past(p))),w)))$
T7	$(\forall w)(\forall p)HOLDS(future(all\text{-}past(p)),w) \rightarrow HOLDS(p,w))$

Table III. Experimental results

Problem	Naïve Method		Two-Stage Method		Three-Stage Method	
	Clauses	HR	Clauses	HR	Clauses	HR
PC1	* (41)	–	3 (10)	2	2 (2)	2
B1	* (42)	–	1 (11)	1	1 (2)	1
B2	* (43)	–	8 (12)	3	2 (3)	2
T4	* (42)	–	72 (12)	3	1 (2)	1
T5	* (44)	–	* (15)	–	15 (5)	5
T7	* (42)	–	12 (11)	2	1 (2)	1

Key: Clauses = I(J) means that I clauses were generated from an initial set of J clauses, HR = K means that K applications of hyperresolution were required, and * indicates that no solution was found after generating 200 clauses.

problems). The addition of the rewriting algorithm in the Two-Stage Method makes all problems soluble, with the exception of T5. The Three-Stage Method solves all problems.

The performance of Two-Stage and Three-Stage Methods is comparable for problems PC1 and B1. This is due to the use of hyperresolution which combines several binary resolutions into one step, the greater number of binary resolutions required in the Two-Stage Method is therefore hidden. However for B2 the Two-Stage Method requires three applications of hyperresolution in the proof in comparison with 2 applications by the Three-Stage Method. The extra depth of search is due to explicit reasoning about the R predicate and leads to a greater expansion of the search space. This finding is in accordance with the analysis of Section 3. The greatest difference between the Two-Stage and Three-Stage Methods occurs in the temporal problems. In problems T4 and T7 the Three-Stage Method finds a solution with one application of hyperresolution while the Two-Stage Method requires three and two applications of hyperresolution with the consequential expansion in search space. Only the Three-Stage Method is able to prove T5—the formulae whose proof requires reasoning about the total ordering of time points on a particular branch of the temporal structure.

The additional computational complexity introduced by the worlds-path unification rules must be included in a full analysis of the efficiency of modal theorem proving, such an analysis would also include the gains of structure sharing methods (for example those of Boyer and Moore (1972) or Staples and Robinson (1990)) the use of reduction rules (Eisinger *et al.*, 1991) and other advanced techniques. We have focussed on one component of the efficiency question, that of induced search space, which we regard as being of primary importance.

From the experimental results and from the analysis presented above we conclude that the addition of the rewriting stage makes theorem proving in reified logics feasible. The second modification of encoding the possible worlds structure in

terms of a worlds-path, and defining unification rules to unify paths, further improves theorem proving performance.

7. Conclusions

This paper has presented new empirical and theoretical work on theorem proving in reified logics. The rewriting methods and worlds-path methods we have employed are not new but have been used in a novel application. The extension of the worlds-path method to the temporal modality is an original result. Adopting the reified approach aided this work by providing a logical metalanguage for the definition of our concepts.

We conclude that in defining theorem provers for reified logics the elimination of representational redundancy is an essential step. This can be implemented as a rewriting procedure which is applied prior to theorem proving. Reasoning about modality remains a significant burden. This problem can be tackled for logics whose possible worlds semantics are directly analogous to the universal and existential quantifiers of first-order logic. In these cases a worlds-path approach can be taken so that reasoning about modality is implemented by an algorithm which is guaranteed to terminate. For temporal logics the situation is more complex and a single rule from the axiomatisation of the logic must be retained during theorem proving.

Appendix 1. Showing the Axiomatisation of the Reified Logics to be Complete

As we have specified the reified logics by defining the semantics of the individual operators our logics are correct in the semantic sense. It remains to show that the definitions are complete. This is achieved by showing that combinations of operators have the correct equivalences and relations. The axiomatisations for most of the logical models presented above are well known and we use these res and quantification. The derivation in QIL of the axioms of the standard axiom sets of the object logic shows that QIL contains the object logic. Proof of completeness also requires that this containment be strict, but this question will not be dealt with here.

There are two advantages in using the reified method. These are firstly, that the definition of theorems is entirely within first-order logic. All variables in a theorem have a well defined status and rules of substitution for formulae and domain constants are also well defined. A second consequence of the first-order formulation is that proof of theorems can be carried out using the well understood inference rules of first-order logic. The definitions of modal systems often include rules of necessitation such as $\vdash \alpha \rightarrow \vdash \Box\alpha$ (Hughes and Cresswell, 1968, p. 31) and $\vdash \alpha \rightarrow \vdash \mathit{all\text{-}future}(\alpha)$ (Rescher and Urquhart, 1971, p. 55). In the reified method these rules are replaced by the formulation $(\forall w)HOLDS(\alpha, w)$. The interaction of modality and quantification can be complex and it is an advantage to eliminate

ambiguity by explicitly defining substitution and necessitation rules. As a result of these features of the reified logics proofs can be derived automatically by means of a conventional theorem prover for first-order logic.

We prove the theorems of the component models by proving

$$(\forall w)(\forall p)HOLDS(\alpha, w)$$

which is equivalent to $\vdash \alpha$ in conventional notation, where α is a theorem which includes p as a formula-variable.

The axiomatisation of PC

PC1 $(\forall w)(\forall p)HOLDS(if(or(p,p),p),w)$

PC2 $(\forall w)(\forall p)(\forall q)HOLDS(if(q,or(p,q)),w)$

PC3 $(\forall w)(\forall p)(\forall q)HOLDS(if(or(p,q),or(q,p)),w)$

PC4 $(\forall w)(\forall p)(\forall q)(\forall r)HOLDS(if(if(q,r),if(or(p,q),or(p,r))),w)$

The definition of *and*

PC5 $(\forall w)(\forall p)HOLDS(and(if(and(p,q),not(or(not(p),not(q)))),$
$if(not(or(not(p),not(q))),and(p,q))),w).$

For the sake of clarity we shall replace the *if* operator and the sequence $and(if(p,q),if(q,p))$ in the theorems below by the $\rightarrow$ and $\leftrightarrow$ metalanguage operators.

The axiomatisation of K

B1 $(\forall w)(\forall ag)(\forall p)HOLDS(poss(ag,p),w)HOLDS(not(bel(ag,not(p))),w)$

B2 $(\forall w)(\forall ag)(\forall p)(\forall q)HOLDS(bel(ag,if(p,q)),w)$
$\rightarrow (HOLDS(bel(ag,p),w) \rightarrow HOLDS(bel(ag,q),w))$

B3 $(\forall w)(\forall ag)(\forall p)HOLDS(bel(ag,p),w) \rightarrow HOLDS(poss(ag,p),w).$

The axiomatisation of BL

T1 $(\forall t)(\forall ag)(\forall p)HOLDS(future(p),t) \leftrightarrow HOLDS(not(all\text{-}future(not(p))),t)$

T2 $(\forall t)(\forall p)(\forall q)HOLDS(all\text{-}past(if(p,q)),t)$
$\rightarrow (HOLDS(all\text{-}past(p),t) \rightarrow HOLDS(all\text{-}past(q)),t)$

T3 $(\forall t)(\forall p)HOLDS(past(all\text{-}future(p)),t) \rightarrow HOLDS(p,t)$

T4 $(\forall t)(\forall p)HOLDS(all\text{-}past(p),t) \rightarrow HOLDS(all\text{-}past(all\text{-}past(p)),t)$

T5 $(\forall t)(\forall p)(\forall q)HOLDS(and(past(p),past(q)),t)$
$\rightarrow HOLDS(past(and(p,q)),t) \vee HOLDS(past(and(p,past(q))),t)$
$\vee HOLDS(past(and(q,past(p))),t)$

T6 $(\forall t)(\forall p)(\forall q)HOLDS(all\text{-}future(if(p,q)),t)$
$\rightarrow (HOLDS(all\text{-}future(p),t) \rightarrow HOLDS(all\text{-}future(q),t))$

T7 $(\forall t)(\forall p)HOLDS(future(all\text{-}past(p)),t) \rightarrow HOLDS(p,t))$

T8 $(\forall t)(\forall p)HOLDS((all\text{-}future(p),t) \rightarrow HOLDS(all\text{-}future(all\text{-}future(p)),t)$

T9 $(\forall t)(\forall p)HOLDS(all\text{-}future(p),t) \rightarrow HOLDS(future(p),t)$

T10 $(\forall t)(\forall p)HOLDS(all\text{-}future(p),t)$
$\rightarrow HOLDS(forall(l, i\text{-}future(l, and(p, all\text{-}future(p)))),t).$

The axiomatisation of the quantifiers

Q1 $(\forall w)(\forall p)(\forall x)HOLDS(forall(x,p),w)$
$\leftrightarrow HOLDS(not(exists(x, not(p))),w)$

Q2 $(\forall w)(\forall p)(\forall x)(\forall y)HOLDS(forall(y, if(forall(x,p),p)),w)$

Q3 $(\forall w)(\forall p)(\forall x)HOLDS(forall(x,p),w) \rightarrow HOLDS(exists(x,p),w)$

Theorems on the interaction of quantification and modality

Q4 $(\forall w)(\forall ag)(\forall p)(\forall x)(\forall y)HOLDS(exists(x, bel(ag,p)),w)$
$\rightarrow HOLDS(bel(ag, exists(y,p)),w)$

Q5 $(\forall w)(\forall p)(\forall x)(\forall y)HOLDS(exists(x, future(p)),w)$
$\rightarrow HOLDS(future(exists(y,p)),w)$

Q6 $(\forall w)(\forall p)(\forall x)(\forall y)HOLDS(forall(x, bel(ag,p)),w)$
$\rightarrow HOLDS(bel(ag, forall(y,p)),w)$

Q7 $(\forall w)(\forall p)(\forall x)(\forall y)HOLDS(bel(ag, forall(x,p)),w)$
$\rightarrow HOLDS(forall(y, bel(ag,p)),w)$

Q8 $(\forall w)(\forall p)(\forall x)(\forall y)HOLDS(future(exists(x,p)),w)$
$\rightarrow HOLDS(exists(y, future(p)),w).$

Not provable

$(\forall w)(\forall p)HOLDS(bel(ag, exists(x,p)),w) \rightarrow HOLDS(exists(x, bel(ag,p)),w).$

Proof of these axioms is achieved by steps in first-order logic with no modifications in syntax, inference rules or the need for necessitation rules. The reified method puts these common modifications to first-order logic proof theory onto a more formal basis.

Notes

[1] In considering the completeness problem, there are two advantages in using the reified method. The first

is that theorems (including modal theorems) can be expressed entirely within first-order logic and the second is that proofs of theorems can be carried out using the well understood inference rules of first-order logic. The issue of completeness is dealt with in Appendix 1.

[2] In fact most cover only a subset of this range.

[3] By feasible we mean refutable by resolution under a simple control strategy, i.e. forwards or backwards chaining. We do not consider metalevel control strategies in this evaluation.

[4] The syntax of our account differs from that of Ohlbach (1988).

[5] This can be done by ordering the literals in clauses of ER such that the left most contains a reified function and restricting the application of the binary resolution rule to the left most clauses of ER only.

[6] In fact these rules must be expressed in an appropriate form, see E4′, E5′ and E5″.

References

Aitken, S., Reichgelt, H. and Shadbolt, N. (1992), Planning, knowledge division and agency, *Proc. 11th UK Planning Special Interest Group.*

Auffray, Y. and Enjalbert, P. (1989), Modal theorem proving: An equational viewpoint, *Proc. IJCAI-11*, 441–445.

Auffray, Y., Enjalbert, P. and Hebrard, J. (1990), Strategies for modal resolution: results and problems *J. Automated Reasoning* **6**, 1–38.

Boyer, R. S. and Moore, J. S. (1972), The sharing of structure in theorem proving programs, in B. Meltzer and D. Michie (eds.), *Machine Intelligence* Vol 7, Edinburgh University Press, pp. 101–116

Catach, L. (1991), TABLEAUX: A general theorem prover for modal logics, *J. Automated Reasoning* **7**, 489–510.

Eisinger, N., Ohlbach, H. J. and Pracklein, A. (1991), Reduction rules for resolution based systems, *Artificial Intelligence* **50**, 141–181.

Farinas del Cerro, L. (1985), Resolution modal logic, *Logique et Analyse* **28**, 153–172.

Fitting, M. (1988), First-order modal tableaux, *J. Automated Reasoning* **4**, 191–213.

Hughes, G. E. and Cresswell, M. J. (1968), *An Introduction to Modal Logic*, Methuen, London.

Jackson, P. and Reichgelt, H. (1989), A general proof method for modal predicate logic, in Jackson, P., Reichgelt, H. and van Harmelen, F. (eds.), *Logic Based Knowledge Representation*, MIT Press.

Konolige, K. (1986), *A Deduction Model of Belief*, Pitman, London.

Kripke, S. A. (1971), Semantical considerations on modal logic, in Linsky, L. (1971).

Linskey, L. (1971), *Reference and Modality*, Oxford University Press.

Moore, R. C. (1985), A formal theory of knowledge and action, in Hobbs, R. J. and Moore, R. C. (eds.), *Formal Theories of the Commonsense World*, Ablex Publishing, New Jersey.

Ohlbach, H. J. (1988), A resolution calculus for modal logics, *Proc. 9th Int. Conf. on Automated Deduction*, Lecture Notes in Computer Science No. 310, Springer-Verlag, New York, pp. 500–516.

Reichgelt, H. (1989a), Logics for reasoning about knowledge and belief, *The Knowledge Engineering Review* **4**, 119–139.

Reichgelt, H. (1989b), A comparison of modal and first order logics of time, in Jackson, P., Reichgelt, H. and van Harmelen, F. (eds.), *Logic Based Knowledge Representation*, MIT Press.

Rescher, N. and Urquhart, A. (1971), *Temporal logic*, Springer-Verlag, New York.

Robinson, J. A. (1979), *Logic Form and Function*, Edinburgh University Press.

Robinson, J. A. (1992), Logic and logic programming, *Comm. ACM* **35**, 41–65.

Shoham, Y. (1986), Reified temporal logics: Semantical and ontological considerations, *Proc. ECAI-7*, 390–397.

Staples, J. and Robinson, P. J. (1990), Structure sharing for quantified terms: fundamentals, *J. Automated Reasoning* **6**, 115–145.

Wallen, L. A. (1987), Matrix proof methods for modal logics, *Proc. IJCAI-10*, 917–923.