Lemma 1 Ltd.
2nd Floor
31A Chain St.
Reading
Berks
RG1 2HX

Modal Logic in HOL

Abstract

The possible worlds semantics for the modal propositional calculus is defined in HOL and some well-known theorems about the semantics are proved. The use of such a semantics to give proof support for a modal logic is illustrated and discussed.

Version:	2.6
Date:	17 October 2002
Reference:	DS/FMU/IED/WRK022
Pages:	23

Prepared by:	R.D. Arthan
Tel:	+44 118 958 4409
E-Mail:	`rda@lemma-one.com`

1 Document Control

1.1 Contents List

1.2 Document Cross References

[1] George Boolos. *The Unprovability of Consistency.* Cambridge University Press, 1979.

[2] Michael J.C. Gordon. HOL:A Proof Generating System for Higher-Order Logic. In G. Birtwistle and P. A. Subrahmanyam, editors, *VLSI Specification, Verification and Synthesis.* Kluwer, 1987.

[3] Allan Ramsay. *Formal Methods in Artificial Intelligence.* Cambridge University Press, 1988.

[4] Johan van Benthem. *A Manual of Intensional Logic.* Center for the Study of Language and Information, 1988.

[5] *A Logic of Authentication.* M. Abadi M. Burrows and R. Needham, Digital Systems Research Center, February 28, 1989.

[6] *The HOL System: Description.* SRI International, 4 December 1989.

1.3 Changes History

Issue 2.1 (29 September 1992) First approved issue.

Issue 2.5 Copyright and banner updates for open source release.

Issue 2.6 PPHol-specific updates for open source release

1.4 Changes Forecast

2 GENERAL

2.1 Scope

This document reports on a brief investigation into embedding the possible worlds semantics for modal logic in HOL.

The purpose of the investigation was to demonstrate some of the new capabilities of the ICL HOL proof tool and to show by means of examples how a semantic embedding of this sort is used to provide proof support for formalisms other than HOL.

The treatment of the possible worlds semantics we give is by no means new. It is essentially just a translation into HOL of the sort of set-theoretic treatment which may be found in any text-book on modal logic, for example, [1].

2.2 Introduction

2.2.1 Modal Logics

Modal propositional calculus is the ordinary propositional calculus augmented by an additional connective, $\Box$. If A is a proposition then $\Box A$, the *necessitation* of A, was originally intended to connote the idea that A was, in some sense, a necessary rather than a contingent fact. The semantics for this calculus, due to Kripke, explicates this notion of necessity in terms of systems, called *frames* in the literature. A frame comprises a set of *possible worlds* supplied with a relation of *accessibility* between worlds. A proposition is viewed as necessary in a world if its truth in each world, x, implies its truth in every world accessible from x.

It turns out that by placing various constraints on the accessibility relation, we arrive at semantics which interpret the necessitation operator in interesting and useful ways, for example:

- transitive accessibility relations for which all ascending chains are finite correspond to a view of $\Box$ which is closely related to the provability predicate for Peano arithmetic. This leads to a useful conceptual framework for understanding provability, consistency and self-reference in the theory of arithmetic (see [1]).
- Various forms of linear accessibility relation correspond to a view of $\Box$ as a temporal operator, with $\Box A$ meaning that A will always hold if it holds now (see, e.g., [4]).
- Taking the worlds as the states of a computer and taking "x is accessible from y" to mean "x results when we execute program p in state y", we get a modal logic which is closely connected with weakest-precondition semantics for programming languages (see [4]).

- Taking the worlds as the possible states of knowledge (or belief) of some individual and taking the accessibility relation to represent ways in which these states grow through the acquisition of new knowledge (or belief), we get various forms of *epistemic modal logic* (or *doxastic modal logic*), in which $\Box A$ means that A is known (or believed). Such logics, usually extended to cover relationships between the knowledge (or belief) of several individuals, have many applications in artificial intelligence and in the study of communication protocols in distributed computer systems (see [3, 5]).

In the sequel we are going to formalise the possible worlds semantics in HOL and prove the semantic justification of two rules of inference for modal logic.

We will also prove some theorems, due to Kripke, about some axioms used in various modal calculi. To state the axioms we introduce the *possibility* operator, $\Diamond$. $\Diamond A$ is defined as $\neg(\Box(\neg A))$. In each case, the theorem we prove says that an axiom is valid provided the accessibility relation possesses a certain property. The axioms and properties are shown in the following table:

Axiom 1	$\Box A \Rightarrow A$	Reflexive
Axiom 2	$\Box A \Rightarrow \Box(\Box A)$	Transitive
Axiom 3	$A \Rightarrow \Box(\Diamond A)$	Symmetric
Axiom 4	$\Diamond A \Rightarrow \Box(\Diamond A)$	Euclidean

(The notion of a euclidean relation is defined in section 3 below.)

Note that the appropriateness of the above axioms depends on the application. For example, they are all arguably appropriate for the epistemic reading, for which, say, 2 is the so-called principle of "positive introspection": 'if A is known, then it is known that A is known'. However, axiom 1 is inappropriate for the doxastic reading: we cannot assert that a proposition is true just because it is believed.

2.2.2 The HOL Proof Tool

HOL (Higher Order Logic) is a polymorphic version of Church's type theory due to M.J.C. Gordon [2]). A system produced by Cambridge University supporting machine-checked reasoning in HOL has been available in the public domain for several years [6]. The Cambridge HOL system has been widely used for a range of verification tasks in academia and has been successfully exploited by a number of industrial users including the Formal Methods Unit at ICL.

ICL are currently engaged in a research programme into formal proof technology which includes a re-engineering of the HOL proof tool to meet more fully the requirements of industrial use and to give a basis for exploiting more recent research on the HOL technology. The ICL HOL proof tool has been specifically designed with a view to its use to provide support for

specification and proof in formalisms other than the HOL logic itself. After a significant prototyping exercise early in the project, a first version of this system is currently being integrated and tested.

As the HOL logic is well-established and uncontroversial mathematically, and as the HOL proof tool is constructed so as to maximise assurance in the correctness of the theorems it proves with respect to that logic, use of HOL to support other formalisms means that the soundness of such support tools does not have to be established on an *ad hoc* basis.

Space does not permit us to give a full exposition of the operation of the HOL proof tool here. However, we hope that some aspects of this will be clear from the examples we give. It may be worth mentioning some basic concepts:

The system is implemented in the interactive functional programming language Standard ML (or, strictly speaking, an extension of Standard ML giving a special syntax for entering HOL terms and types and supporting the use of an extended character set for mathematical notations). ML, often referred to as the *metalanguage*, also acts as the command language through which the user interacts with the system.

The types and terms of the HOL language are implemented as abstract data types, $TYPE$ and $TERM$. The constructors of the data type of terms guarantee that all values of type $TERM$ obey the typing rules of the HOL language. Proof is conducted, at the most primitive level, by computing theorems, i.e. values of an abstract data type, THM. The constructors of this abstract data type implement the primitive inference rules of the logic. Thus, the only way to compute a theorem is via a sequence of primitive inferences, and so any value of type THM is indeed a theorem of the HOL logic. On top of this logical kernel are implemented a wide range of proof procedures which assist the user in performing proofs. The great merit of this approach to implementing a proof tool is that the logical kernel guarantees that the soundness of the system cannot be compromised by infelicities in the coding of these derived proof procedures.

2.2.3 Notation

The present document is a *literate script* containing a mixture of narrative text and input for the ICL HOL system. The appendix contains a listing of the HOL theory set up by the script and section 9 contains an index of the objects defined in the script. Defining occurrences of names are shown in **bold**.

The inputs for HOL consists of a sequence of commands in an extension of the interactive programming language Standard ML.

HOL terms and types appear enclosed by the symbols '⌜' and '⌝' (with a ':' after the '⌜' for a type), the text between the symbols being parsed as HOL and resulting in an ML value of type $TERM$ or $TYPE$. For example, consider the following fragment of ML:

SML
```
val t = ⌜∀m n•m + n − n = m⌝;
```

This causes the HOL term $\forall m\ n \bullet m\ +\ n\ -\ n\ =\ m$ to be parsed and the resulting value of type $TERM$ to be bound to the ML variable t.

HOL constants are introduced using constant definition boxes which have the form:

HOL Constant

$$\frac{c : ty}{P}$$

The intention of this is to introduce a new constant, c, of type ty, satisfying the property P. In the present document P will always be a (possibly universally quantified) equation or bi-implication defining a value or a function.

The definitions introduced by these boxes are *conservative*. The HOL system maintains a distinction between conservative extensions and the introduction of arbitrary axioms.

2.2.4 HOL Preamble

The following HOL commands create a new HOL theory in which we will save our definitions and theorems, and prepare the specification proof facilities for the task in hand.

SML
```
open_theory"hol";
new_theory"wrk022";
set_pc "hol";
```

3 AUXILIARY DEFINITIONS

We need definitions of the concepts of *reflexive*, *transitive*, *symmetric* and *euclidean* relations in HOL.

As we have already mentioned, HOL is a polymorphic variant of simple type theory. In such a system properties of values of type τ are represented as propositional functions, that is to say they are objects of type $\tau \rightarrow BOOL$, where $BOOL$ is the two-point type of truth values. A binary relation on a type τ is a two-argument propositional function, i.e., it has type $\tau \rightarrow \tau \rightarrow BOOL$.

Polymorphism allows us to use variables which range over types, such type variables are distinguished syntactically by having names beginning with the character $'$. The propositional functions $Reflexive$, $Transitive$, etc. which we will shortly define are polymorphic constants, they may be applied to any value whose type has the form $\tau \to \tau \to BOOL$.

The definitions of the four properties of relations we need follow:

HOL Constant

$$Reflexive\text{: } ('a \to 'a \to BOOL) \to BOOL$$

$$\forall rel\bullet\ Reflexive\ rel \Leftrightarrow \forall x\bullet\ rel\ x\ x$$

HOL Constant

$$Transitive\text{: } ('a \to 'a \to BOOL) \to BOOL$$

$$\forall rel\bullet\ Transitive\ rel \Leftrightarrow \forall x\ y\ z\bullet\ rel\ x\ y \wedge rel\ y\ z \Rightarrow rel\ x\ z$$

HOL Constant

$$Symmetric\text{: } ('a \to 'a \to BOOL) \to BOOL$$

$$\forall rel\bullet\ Symmetric\ rel \Leftrightarrow \forall x\ y\bullet\ rel\ x\ y \Rightarrow rel\ y\ x$$

HOL Constant

$$Euclidean\text{: } ('a \to 'a \to BOOL) \to BOOL$$

$$\forall rel\bullet\ Euclidean\ rel \Leftrightarrow \forall x\ y\ z\bullet\ rel\ x\ y \wedge rel\ x\ z \Rightarrow rel\ y\ z$$

4 POSSIBLE WORLD SEMANTICS IN HOL

4.1 Frames

In a set-theoretic treatment a *frame* consists of a non-empty set W, of *possible worlds*, equipped with a binary relation, R, the *accessibility* relation. We will use a type variable $'W$ to represent the set of possible worlds, so that our general treatment can be instantiated to a particular type of possible worlds. We can thus capture the notion of a frame using the following type abbreviation:

SML

$$declare\_type\_abbrev\ (\texttt{"}\boldsymbol{FRAME}\texttt{"},\ [\texttt{"}'W\texttt{"}],\ \ulcorner :'W \to 'W \to BOOL\urcorner);$$

The above declaration introduces a new type abbreviation $FRAME$ with a single formal parameter $'W$; The effect of the declaration is that, for example, the type expression $\ulcorner:(\mathbb{N})FRAME\urcorner$ will represent the type $\ulcorner : \mathbb{N} \to \mathbb{N} \to BOOL \urcorner$ of binary relations on the type $\mathbb{N}$ of natural numbers. Thus a value of type $\ulcorner:(\mathbb{N})FRAME\urcorner$ can be viewed as a frame in which the possible worlds are numbers.

We will use the variable R for accessibility relations, and x, y and z for worlds.

4.2 Valuations

A *valuation* will be a function assigning a truth-value to every possible world. In a syntactic treatment we would work with an *evaluator* assigning valuations in this sense to the propositional variables. In our approach we can use HOL variables of the appropriate type to represent the semantics directly. We use the following type abbreviation for valuations:

SML

```
declare_type_abbrev ("VALUATION", ["'W"], ⌜:'W →BOOL⌝);
```

Thus, for example, a value of type $\ulcorner:(\mathbb{N})VALUATION\urcorner$ is a propositional function on the natural numbers.

We will use the variables A, B and C for valuations.

4.3 Propositional Connectives

It is straightforward to give the semantics of the ordinary propositional connectives in their modal guise. In each case the modal version of a connective combines the valuations which are its operands to give a valuation which asserts that for every world the corresponding propositional connective holds between the values taken by the operands in that world. The definitions of these connectives are therefore independent of any accessibility relation.

We will take implication and negation as our primitive connectives and define others in terms of them:

HOL Constant

```
⇒_modal: ('W) VALUATION → ('W) VALUATION → ('W) VALUATION
```

$$\forall A\ B\ x\bullet \Rightarrow\_modal\ A\ B\ x \Leftrightarrow A\ x \Rightarrow B\ x$$

HOL Constant

$$\neg\_\boldsymbol{modal}\colon (\prime W)\,VALUATION \to (\prime W)\,VALUATION$$

$$\forall A\ x\bullet\ \neg\_modal\ A\ x \Leftrightarrow \neg A\ x$$

In making the above definitions, we had to distinguish the names for the modal connectives from those already reserved for the propositional connectives in HOL. The *alias* mechanism supported by ICL HOL allows us to use the usual names instead as syntactic abbreviations. The following declarations achieve this:

SML

```
declare_alias
    ("⇒", ⌜⇒_modal: ('W) VALUATION → ('W) VALUATION → ('W) VALUATION⌝);
declare_alias
    ("¬", ⌜¬_modal: ('W) VALUATION → ('W) VALUATION⌝);
```

To see how this works, we can now use a more natural syntax for the definition of modal disjunction and conjunction:

HOL Constant

$$\vee\_\boldsymbol{modal}\colon (\prime W)\,VALUATION \to (\prime W)\,VALUATION \to (\prime W)\,VALUATION$$

$$\forall A\ B\bullet\ \vee\_modal\ A\ B = (\neg A \Rightarrow B)$$

HOL Constant

$$\wedge\_\boldsymbol{modal}\colon (\prime W)\,VALUATION \to (\prime W)\,VALUATION \to (\prime W)\,VALUATION$$

$$\forall A\ B\bullet\ \wedge\_modal\ A\ B = \neg(A \Rightarrow \neg B)$$

Note here that $\neg$ and $\Rightarrow$ refer to the modal connectives. The HOL $\neg$ and $\Rightarrow$ may still be used — the ICL HOL system identifies the appropriate internal representation on the basis of the types of the operands.

As with the other connectives we make alias declarations for the modal disjunction and conjunction:

SML

```
declare_alias("∨", ⌜∨_modal: ('W) VALUATION → ('W) VALUATION → ('W) VALUATION⌝);
declare_alias("∧", ⌜∧_modal: ('W) VALUATION → ('W) VALUATION → ('W) VALUATION⌝);
```

4.4 Necessitation

The necessitation operator, $\Box$, is defined with respect to a given frame R; The necessitation of a valuation, A, is the valuation which is true at a world x if and only if A is true at every world accessible from x. The HOL definition of this is as follows

HOL Constant

$\Box$: $('W)FRAME \rightarrow ('W)VALUATION \rightarrow ('W)VALUATION$

$\forall R\ A\ x\bullet\ \Box R\ A\ x \Leftrightarrow \forall y\bullet\ R\ x\ y \Rightarrow A\ y$

4.5 Possibility

The possibility operator, $\Diamond$, is defined in terms of necessitation and negation as follows:

HOL Constant

$\Diamond$: $('W)FRAME \rightarrow ('W)VALUATION \rightarrow ('W)VALUATION$

$\forall R\ A\bullet\ \Diamond R\ A = \neg(\Box R(\neg A))$

4.6 Validity

A valuation is valid if it is true in every world. Thus:

HOL Constant

$\boldsymbol{Valid}$: $('W)VALUATION \rightarrow BOOL$

$\forall A\bullet\ Valid\ A \Leftrightarrow \forall x\bullet\ A\ x$

4.7 A Rewrite System

In the sequel, we will use the above definitions to prove some theorems about the semantics. The proofs will have a common pattern, in which the first step is to expand out the above definitions to reduce the goal to be proved to a proposition in the predicate calculus. The following ML command gives us an ML value containing the list of defining theorems which we use to do this.

SML
```
val modal_rewrites = map snd (get_defns"-");
```

This pattern of proof is common in most applications of HOL: at the beginning of building a theory, one often has to reduce problems to first principles. Usually, once one has established a basic repertoire of theorems characterising the problem domain, subsequent proofs are performed at a higher-level using the characterising theorems.

5 INFERENCE RULES

Using the definitions of the previous section we can now prove some results about the modal operators. In this section we prove two theorems which are the semantic justifications for the two inference rules usually associated with modal logics.

5.1 Modus Ponens

The rule of modus ponens for modal logic is given by the following HOL theorem. As an inference rule, modus ponens says that from (the theoremhood of) $A \Rightarrow B$ and (the theoremhood of) A we may infer (the theoremhood of) B. The semantic justification of this is the theorem we shall now prove which asserts that if $A \Rightarrow B$ and A are valid, then so is B.

In the following statement of this theorem note that the conjunction and the second implication are the HOL logical connectives. The first implication is the modal one.

SML
```
push_goal([], ⌜∀A B• Valid (A ⇒ B) ∧ Valid A ⇒ Valid B⌝);
```

The above command initiates a session with the ICL HOL subgoal package, the standard means of finding proofs by a goal oriented search. Goals are reduced to subgoals by applying tactics. A discussion of how proofs are conducted is outside the scope of this document. The proofs given here follow a common pattern. First we rewrite with the definitions to reduce the goal to a predicate calculus proposition. We then break this down using the standard tactic for simplifying such propositions, *strip_tac*. This simplification gives us a simpler goal and some assumptions with which to prove it. In most of the present proofs, one or more of the assumptions turns out to be a universally quantified formula, which we use to prove the goal by specialisation and rewriting.

SML
```
a(rewrite_tac modal_rewrites);
a(REPEAT strip_tac);
a(POP_ASM_T (ante_tac o ∀_elim⌜x⌝) THEN asm_rewrite_tac[]);
```

This completes the proof of our goal. We save the theorem in the HOL theory as follows:

```
SML
val modal_mp_thm = save_thm("modal_mp_thm", pop_thm());
```

5.2 Necessitation

The rule of necessitation says that from A we may infer $\Box A$. Again this rule holds for any accessibility relation.

```
SML
push_goal([], ⌜∀R A• Valid A ⇒ Valid (□R A)⌝);
```

```
SML
a(rewrite_tac modal_rewrites);
a(REPEAT strip_tac THEN asm_rewrite_tac[]);
```

```
SML
val necessitation_thm = save_thm("necessitation_thm", pop_thm());
```

6 THE DISTRIBUTION AXIOM SCHEMATA

The distribution axiom schemata contains all sentences of the form $\Box(A \Rightarrow B) \Rightarrow (\Box A \Rightarrow \Box B)$. The semantic justification for this is proved as follows:

```
SML
push_goal([], ⌜∀R A B• Valid (□R(A ⇒ B)) ⇒ Valid (□R A ⇒ □R B)⌝);
```

```
SML
a(rewrite_tac modal_rewrites);
a(REPEAT strip_tac);
a(DROP_NTH_ASM_T 3 (ante_tac o list_∀_elim[⌜x⌝, ⌜y⌝]));
a(DROP_NTH_ASM_T 2 (ante_tac o ∀_elim ⌜y⌝) THEN asm_rewrite_tac[]);
a(REPEAT strip_tac);
```

```
SML
val distribution_thm = save_thm("distribution_thm", pop_thm());
```

The reader may be reassured to know that the above is the longest subgoal package proof in the document.

7 FOUR AXIOMS

In this section we prove the promised four theorems about the interplay between certain modal axioms and properties of the accessibility relation.

7.1 Axiom 1

This axiom holds for reflexive accessibility relations:

SML
```
push_goal([], ⌜∀R• Reflexive R ⇒ ∀A• Valid(□R A ⇒ A)⌝);
```

SML
```
a(rewrite_tac modal_rewrites);
a(REPEAT strip_tac);
a(POP_ASM_T (ante_tac o ∀_elim⌜x⌝) THEN asm_rewrite_tac[]);
```

SML
```
val axiom1_thm = save_thm("axiom1_thm", pop_thm());
```

7.2 Axiom 2

This axiom holds for transitive accessibility relations:

SML
```
push_goal([], ⌜∀R• Transitive R ⇒ ∀A• Valid(□R A ⇒ □R (□R A))⌝);
```

SML
```
a(rewrite_tac modal_rewrites);
a(REPEAT strip_tac);
a(DROP_NTH_ASM_T 4 (ante_tac o list_∀_elim[⌜x⌝, ⌜y⌝, ⌜y'⌝]) THEN asm_rewrite_tac[]);
```

SML
```
val axiom2_thm = save_thm("axiom2_thm", pop_thm());
```

7.3 Axiom 3

This axiom holds for symmetric accessibility relations:

SML

```
push_goal([], ⌜∀R• Symmetric R ⇒ ∀A• Valid(A ⇒ □R (◇R A))⌝);
```

SML

```
a(rewrite_tac modal_rewrites);
a(REPEAT strip_tac);
a(DROP_NTH_ASM_T 3 (ante_tac o list_∀_elim[⌜x⌝, ⌜y⌝]) THEN asm_rewrite_tac[]);
a(REPEAT strip_tac THEN simple_∃_tac⌜x⌝ THEN asm_rewrite_tac[]);
```

SML

```
val axiom3_thm = save_thm("axiom3_thm", pop_thm());
```

7.4 Axiom 4

This axiom holds for euclidean accessibility relations:

SML

```
push_goal([], ⌜∀R• Euclidean R ⇒ ∀A• Valid(◇R A ⇒ □R (◇R A))⌝);
```

SML

```
a(rewrite_tac modal_rewrites);
a(REPEAT strip_tac);
a(DROP_NTH_ASM_T 4 (ante_tac o list_∀_elim[⌜x⌝, ⌜y'⌝, ⌜y⌝]) THEN asm_rewrite_tac[]);
a(REPEAT strip_tac THEN ∃_tac⌜y⌝ THEN asm_rewrite_tac[]);
```

SML

```
val axiom4_thm = save_thm("axiom4_thm", pop_thm());
```

8 PROOF SUPPORT FOR A MODAL LOGIC

In this section we implement inference rules for modal logic, and also implement the axiom schemata for the modal logic traditionally called T. T has as its axioms all distribution axioms, all instances of ordinary propositional tautologies and all sentences of the form $\Box A \Rightarrow A$.

Note that we use the term "axioms" here for what are actually theorems derived from our definitions of the semantics. The axiom schemata are implemented as proof procedures, i.e., ML functions which prove rather than merely postulate the "axioms" in question.

Examples of the use of the rules and axiom schemata are given in section 8.3 below.

8.1 Implementing the Inference Rules

In this section we implement derived rules in HOL corresponding to the theorems proved in section 5 above.

The first rule is modus ponens. Given as its arguments theorems of the form $\Gamma \vdash Valid(A \Rightarrow B)$[1]and $\Delta \vdash Valid\, A$ it will prove the theorem $\Gamma \cup \Delta \vdash Valid\, B$. This is done with HOL rules to instantiate our theorem on modus ponens and to use the theorem arguments to discharge the antecedents of the resulting implication.

SML
```
fun modal_mp_rule (thm1 : THM) (thm2 : THM) : THM = (
        simple_⇒_match_mp_rule modal_mp_thm (∧_intro thm1 thm2)
);
```

The second rule is necessitation. Given as arguments a term, $\ulcorner R \urcorner$, and a theorem of the form $\Gamma \vdash Valid\, A$, it will prove $\Gamma \vdash Valid(\Box RA)$. This is done by instantiating our theorem on necessitation.

SML
```
fun nec_rule (R : TERM) (thm : THM) : THM = (
        inst_term_rule [(R, ⌜R:('a)FRAME⌝)]
        (simple_⇒_match_mp_rule necessitation_thm thm)
);
```

8.2 Implementing the Axiom Schemata

It is usual in modal calculi to take all ordinary propositional tautologies as axioms. The following ML code implements this rule by proving all such axioms. This proof procedure works by rewriting with the definitions of validity and all the modal connectives except $\Box$, and then using *strip_tac* to prove the resulting proposition.

SML
```
fun modal_taut_rule (tm : TERM) : THM = (
        tac_proof( ([], tm),
        rewrite_tac
        (map(get_defn"−")
        ["◇", "∧_modal", "∨_modal", "¬_modal", "⇒_modal", "Valid"])
        THEN REPEAT strip_tac)
);
```

[1]The HOL logic is formulated as a *sequent calculus*: the assertions one proves comprise a list of assumptions and a conclusion. We write $\Gamma \vdash t$ for a theorem with assumptions Γ and conclusion t.

It is also usual to take all instances of the distribution axiom as axioms. This is implemented by the following proof procedure which takes the accessibility relation, $\ulcorner R \urcorner$, and terms $\ulcorner A \urcorner$ and $\ulcorner B \urcorner$ as arguments and proves $\vdash \Box R(A \Rightarrow B) \Rightarrow (\Box R\, A \Rightarrow \Box R\, B)$. The proof is done by instantiating our theorem on the distribution axiom appropriately.

SML

```
fun dist_rule (R : TERM) (A : TERM) (B : TERM) : THM = (
        inst_term_rule
        (combine [R, A, B] (fst(strip_∀(concl distribution_thm))))
        (all_simple_∀_elim distribution_thm)
);
```

With reference to our theorem about axiom 1, to complete our rules and axioms for the system T we need assume that some unspecified accessibility relation R is symmetric. The following proof procedure proves instances of axiom 1 on the assumption of a symmetric accessibility relation, R.

SML

```
fun axiom1_rule (tm : TERM) : THM = (
        ⇔_t_elim(rewrite_conv[undisch_rule(all_simple_∀_elim axiom1_thm)] tm)
);
```

8.3 Example Proofs

As examples, we will prove the (semantic justifications of) the following two theorems of the system T. The two results are given as theorem 6 in chapter 1 of [1] and the proofs we give follow the ones give there.

Informal Example

$\vdash A \Rightarrow \Diamond A$

$\vdash \Box A \Rightarrow \Diamond A$

For expository purposes, we have written the proof out step by step. In actual use, the main purpose of the sort of rules we have implemented is to enable the construction of higher-level and more powerful facilities for finding proofs. Proofs at the level of detail seen here would not be seen by the user.

The following sequence of computations, then, is the HOL proof of the two results. After each step we show the output produced by the HOL system after executing that step. Note that, essentially, we are now working purely in modal logic: we no longer see any HOL logical operators, just terms of the form $\ulcorner Valid\, A \urcorner$ where A involves only modal connectives; all proof steps are done using the modal inference rules and axiom schemata we have coded above.

SML

```
val lemma1 = axiom1_rule ⌜Valid(□R (¬A) ⇒ ¬A)⌝;
```

HOL Output

```
val lemma1 = Reflexive R ⊢ Valid (□ R (¬ A) ⇒ ¬ A) : THM
```

SML

```
val lemma2 = modal_taut_rule⌜Valid((□R(¬A) ⇒ ¬A) ⇒ (A ⇒ ◇R A))⌝;
```

HOL Output

```
val lemma2 = ⊢ Valid ((□ R (¬ A) ⇒ ¬ A) ⇒ A ⇒ ◇ R A) : THM
```

SML

```
val result1 = save_thm("result1", modal_mp_rule lemma2 lemma1);
```

HOL Output

```
val result1 = Reflexive R ⊢ Valid (A ⇒ ◇ R A) : THM
```

SML

```
val lemma3 = modal_taut_rule⌜Valid((□R A ⇒ A) ⇒ (A ⇒ ◇R A) ⇒ (□R A ⇒ ◇R A))⌝;
```

HOL Output

```
val lemma3 = ⊢ Valid ((□ R A ⇒ A) ⇒ (A ⇒ ◇ R A) ⇒ □ R A ⇒ ◇ R A) : THM
```

SML

```
val lemma4 = axiom1_rule ⌜Valid(□R A ⇒ A)⌝;
```

HOL Output

```
val lemma4 = Reflexive R ⊢ Valid (□ R A ⇒ A) : THM
```

SML

```
val lemma5 = modal_mp_rule lemma3 lemma4;
```

HOL Output

```
val lemma5 = Reflexive R ⊢ Valid ((A ⇒ ◇ R A) ⇒ □ R A ⇒ ◇ R A) : THM
```

SML

```
val result2 = save_thm("result2", modal_mp_rule lemma5 result1);
```

HOL Output

```
val result2 = Reflexive R ⊢ Valid (□ R A ⇒ ◇ R A) : THM
```

9 PRACTICAL SYSTEMS

While based on a semantics similar to the semantics discussed here, systems of modal logic intended for practical applications, e.g. for program verification or protocol verification, will involve application-specific syntactic features. It is important for the ease of use of proof tools that the user should interact with the tool using the natural formalism for the task at hand. As we have seen, this can come with very little extra work in the case of a sufficiently simple logical language.

Proof support offering this feature for more complex languages may be supplied by combining a semantic approach like the one given here with use of the facilities offered by the HOL proof tool for manipulating syntax. This technique is usually referred to as *semantic embedding* (as opposed to a *syntactic* treatment, in which one would use HOL to reason about syntactic notions, e.g. inference rules, rather than semantic ones, e.g. the accessibility relation in the present treatment of modal propositional logic).

In the semantic embedding technique, sentences of the language to be supported are represented by semantically equivalent HOL terms. A parser maps sentences in the desired concrete syntax to their HOL representation and a pretty-printer automatically inverts this mapping when terms are displayed. In the simple example we have given here the parser and pretty-printer could be very easily constructed modifications of the HOL parser and pretty-printer which would suppress the appearance of the validity operator. Tools assisting in the production of parsers and pretty-printers are supplied as part of the ICL HOL system.

On the basis of theorems proved about the semantic objects, proof procedures, analogous to the rules we have implemented in the present document, may be produced which preserve the required syntax as far as the user is concerned. Such procedures may often be produced quite readily by customising or specialising existing proof procedures for HOL. As in our example, if valid inference rules for the language are already known, then they can be used as the basis for the design of such proof procedures.

An important advantage of the semantic embedding techniques is that, compared with other approaches, such as coding a complete system for manipulating syntax from scratch, it drastically reduces the amount of code in which errors can make the system inconsistent (i.e. allow the user to prove an invalid result).

A system of this sort offering proof support for Z is currently being prototyped in ICL and good results are emerging. Z is in many respects much harder to accommodate than most of the formalisms based on modal logics which have been proposed for computer science applications, and so such formalisms promise to be good applications for this semantic embedding techniques.

10 INDEX

A THE THEORY wrk022

A.1 Parents

hol

A.2 Constants

Reflexive	$('a \to 'a \to BOOL) \to BOOL$
Transitive	$('a \to 'a \to BOOL) \to BOOL$
Symmetric	$('a \to 'a \to BOOL) \to BOOL$
Euclidean	$('a \to 'a \to BOOL) \to BOOL$
$\Rightarrow$***_modal***	$('W \to BOOL) \to ('W \to BOOL) \to 'W \to BOOL$
$\neg$***_modal***	$('W \to BOOL) \to 'W \to BOOL$
$\vee$***_modal***	$('W \to BOOL) \to ('W \to BOOL) \to 'W \to BOOL$
$\wedge$***_modal***	$('W \to BOOL) \to ('W \to BOOL) \to 'W \to BOOL$
$\Box$	$('W \to 'W \to BOOL) \to ('W \to BOOL) \to 'W \to BOOL$
$\Diamond$	$('W \to 'W \to BOOL) \to ('W \to BOOL) \to 'W \to BOOL$
Valid	$('W \to BOOL) \to BOOL$

A.3 Aliases

$\Rightarrow$	$\Rightarrow\_modal : ('W \to BOOL) \to ('W \to BOOL) \to 'W \to BOOL$
$\neg$	$\neg\_modal : ('W \to BOOL) \to 'W \to BOOL$
$\vee$	$\vee\_modal : ('W \to BOOL) \to ('W \to BOOL) \to 'W \to BOOL$
$\wedge$	$\wedge\_modal : ('W \to BOOL) \to ('W \to BOOL) \to 'W \to BOOL$

A.4 Type Abbreviations

$'W$ ***FRAME*** $'W \to 'W \to BOOL$
$'W$ ***VALUATION*** $'W \to BOOL$

A.5 Definitions

Reflexive $\vdash \forall\ rel\bullet\ Reflexive\ rel \Leftrightarrow (\forall\ x\bullet\ rel\ x\ x)$

Transitive $\vdash \forall\ rel$
$\bullet\ Transitive\ rel$
$\Leftrightarrow (\forall\ x\ y\ z\bullet\ rel\ x\ y \wedge rel\ y\ z \Rightarrow rel\ x\ z)$

Symmetric $\vdash \forall\ rel\bullet\ Symmetric\ rel \Leftrightarrow (\forall\ x\ y\bullet\ rel\ x\ y \Rightarrow rel\ y\ x)$

Euclidean $\vdash \forall\ rel$
$\bullet\ Euclidean\ rel$
$\Leftrightarrow (\forall\ x\ y\ z\bullet\ rel\ x\ y \wedge rel\ x\ z \Rightarrow rel\ y\ z)$

⇒_modal $\vdash \forall\ A\ B\ x\bullet\ (A \Rightarrow B)\ x \Leftrightarrow A\ x \Rightarrow B\ x$

¬_modal $\vdash \forall\ A\ x\bullet\ (\neg\ A)\ x \Leftrightarrow \neg\ A\ x$

∨_modal $\vdash \forall\ A\ B\bullet\ (A \vee B) = (\neg\ A \Rightarrow B)$

∧_modal $\vdash \forall\ A\ B\bullet\ (A \wedge B) = (\neg\ (A \Rightarrow \neg\ B))$

□ $\vdash \forall\ R\ A\ x\bullet\ \Box\ R\ A\ x \Leftrightarrow (\forall\ y\bullet\ R\ x\ y \Rightarrow A\ y)$

◇ $\vdash \forall\ R\ A\bullet\ \Diamond\ R\ A = (\neg\ \Box\ R\ (\neg\ A))$

Valid $\vdash \forall\ A\bullet\ Valid\ A \Leftrightarrow (\forall\ x\bullet\ A\ x)$

A.6 Theorems

modal_mp_thm $\vdash \forall\ A\ B\bullet\ Valid\ (A \Rightarrow B) \wedge Valid\ A \Rightarrow Valid\ B$

necessitation_thm
$\vdash \forall\ R\ A\bullet\ Valid\ A \Rightarrow Valid\ (\Box\ R\ A)$

distribution_thm
$\vdash \forall\ R\ A\ B$
$\bullet\ Valid\ (\Box\ R\ (A \Rightarrow B))$
$\Rightarrow Valid\ (\Box\ R\ A \Rightarrow \Box\ R\ B)$

axiom1_thm $\vdash \forall\ R\bullet\ Reflexive\ R \Rightarrow (\forall\ A\bullet\ Valid\ (\Box\ R\ A \Rightarrow A))$

axiom2_thm $\vdash \forall\ R$
$\bullet\ Transitive\ R$
$\Rightarrow (\forall\ A\bullet\ Valid\ (\Box\ R\ A \Rightarrow \Box\ R\ (\Box\ R\ A)))$

axiom3_thm $\vdash \forall\ R$
$\bullet\ Symmetric\ R$
$\Rightarrow (\forall\ A\bullet\ Valid\ (A \Rightarrow \Box\ R\ (\Diamond\ R\ A)))$

axiom4_thm $\vdash \forall\ R$
$\bullet\ Euclidean\ R$
$\Rightarrow (\forall\ A$
$\bullet\ Valid$
$(\Diamond\ R\ A \Rightarrow \Box\ R\ (\Diamond\ R\ A)))$

result1 $Reflexive\ R \vdash Valid\ (A \Rightarrow \Diamond\ R\ A)$

result2 $Reflexive\ R \vdash Valid\ (\Box\ R\ A \Rightarrow \Diamond\ R\ A)$