1. Introduction.

1.1 Aims of the project.

The axiomatic translation [1] is a method for converting propositional modal logic into first-order logic, which can then be solved with standard first-order logic theorem provers. Much of the effort that would otherwise be required (both in developing theory and software) to directly determine the satisfiability of problems in modal logic is hence avoided, by reusing pre-existing first-order theorem prover technology [32]. The classical semantic translation has been available for some time [25, 29, 31, 32]. In this modal axioms are incorporated into the translation as correspondence properties, which often show poor performance in first-order theorem provers. The axiomatic translation is a development in which the translated first-order clauses that are produced have much better computational properties in ordered resolution. The axiomatic translation can hence be used to solve problems in modal logic that were previously inaccessible.

SPASS is a sophisticated first-order theorem prover. It implements a large number of techniques for reducing the search space of resolution calculations (to determine satisfiability of the target formulae), and in particular, ordered resolution [28]. It is implemented in C, and the code is publicly available [16]. The classical semantic translation has already been implemented in SPASS [16, 17, 25]. In this project, SPASS has been extended by incorporation of an implementation of the axiomatic translation. The detailed aims of the project are listed below:

- To implement the axiomatic translation in SPASS.
- The implementation must cover at least the functionality described in [1]; the translation of arbitrary modal target formulae, with an arbitrary number of distinct modalities; schema encodings for all the modal axioms presented in [1]; support for combinations of modal axioms; support for the inclusion of compositional subformulae in the instantiation set used during the incorporation of modal axioms; support for the incorporation of axioms in a mixed mode together with correspondence properties; simple optimizations of the translation. Finally the implementation must interface with the SPASS resolution prover to allow local satisfiability of the problems in modal logic to be determined.
- To build a prototype interface, providing access to extended-SPASS, with a means of developing, debugging, and running experiments, and of storing the results.
- To document the implementation.
- To thoroughly test the implementation to ensure fidelity of the translation, providing a test suite against which further developments to extended-SPASS can be tested.

- To make simple extensions as suggested in [1], and in particular to determine the global satisfiability of target problems, to apply the axiomatic translation to other axioms, and to apply the axiomatic translation to bi-modal axioms
- To use the software developed to demonstrate the axiomatic translation on a wide range of target formulae in a wide variety of modal axioms and axiom combinations.
- To use the software to investigate aspects of the axiomatic translation, with a view to suggesting new features that might be implemented in future software releases, and in particular, pointing the way forward to the development of an automated method for incorporation of the schema encoding of an arbitrary modal axiom.

It will be seen, that all these aims were met by the project.

1.2 Outline of Dissertation.

A brief outline of the dissertation follows. In the introductory chapter, an introduction to logic, to modal logic and resolution is given. Only sufficient material is provided to support the remainder of the dissertation. Then some features of SPASS are presented, again only sufficient to understand the use made of SPASS in the dissertation.

The axiomatic translation is then described in detail. The formal definitions (and proofs) are given in [1], and so are not repeated. Instead, an explanation is presented, with many examples added that illustrate the translation. In addition, new material is developed by making extensions to the material presented in [1]. All of these extensions were suggested briefly in [1].

First a general overview of the materials and methods used is given. Next the software developed in the project is documented. Formal software engineering documentation is not provided. Instead, the requirement section describes the aims of the software project, and the general strategy adopted during the project. The specification section links the output produced by the software to details of the axiomatic translation, and forms the basis of a user manual for the software. Finally the details of the design and implementation are described. The detailed information provided on the implementation is required in order to ensure that future developers can continue the project with ease, and of course, it supports the explanation of the design. The software developed for this project is not only the axiomatic translation extensions to SPASS (in C), but also a web-based user interface that enables easy interaction with extended-SPASS, and a Java-based axiomatic translation that was developed to support software testing.

Testing of the software developed in this project was a lengthy process. The manipulations performed by extended-SPASS are very detailed; there are many options available to control translation, and many axioms have been implemented. Each needs testing against a large variety of target formulae. There is no shortcut available for testing software of this type. An overview of the strategy adopted is provided. Both manual and automated testing is described; both unit testing and integration testing is described. As a result of the test strategy, a test suite was developed that provides a baseline against which future developments can be tested.

The results of execution of extended-SPASS on a variety of target formulae and in many axioms is presented. These demonstrate the range of problems available to the user of extended-SPASS and provide a reference data set. Additional problems are reported that probe some features of the axiomatic translation. Finally, in the discussion section an assessment of two aspects of the project is made. First, the maintainability and usefulness of the software is assessed. And second, some features of the axiomatic translation are investigated using extended-SPASS. In both the results and discussion sections, a good deal of future work is suggested from the data produced, particularly in further theoretical work that needs to be undertaken. The likely outcome of this theoretical work is predicated.

In this dissertation theoretical aspects of the axiomatic translation have been illustrated using concrete examples, software has been developed, documented and tested, and extended-SPASS has been used to demonstrate and investigate features of the axiomatic translation. Much work remains to be done in this interesting area.

1.3 Basics of logic, modal logic and resolution.

1.3.1 Propositional and predicate logic. [adapted from 5, 6, 8, 11, 12, 13, 14].

A propositional (zeroth-order) formula is a well-formed combination of atomic propositions (usually represented by symbols such as A, B,…) and logical connectives (in this study { $\neg$, $\vee$, $\wedge$, $\leftarrow$, $\rightarrow$, $\leftrightarrow$; `not`, `or`, `and`, `implied`, `implies`, `bi-implication`} are used). Although precedence of these symbols is defined elsewhere, in this study parentheses are always used to avoid any confusion. Every formula in propositional logic takes either the value of true ($\top$) or false ($\bot$). *Truth tables*, in which every possible combination of true and false (*interpretation*) for the atoms in a formula is considered, can be used both to define the meaning of a connective and to characterize a propositional formula. Definitions are seen in figure 1.1.

First-order (predicate) logic builds upon the foundation of propositional logic by introducing quantifiers. The universal quantifier $\forall$x(F) is read as '*forall x in F*', and the existential quantifier $\exists$x(F) is read as there '*exists an x in F such that*'. These quantifiers allow relationships between variables (like x) to be expressed. In this study, all variables will be quantified (*bound*). If a variable is not within the scope of a quantifier, then it is *free*. (SPASS does not allow input with free variables). The terms *constants*, *variables*, *predicates*, *functions*, *relations* and *quantifiers* appear in this study, and need explanation in terms of predicate logic. The easiest way to understand these is to relate them to statements that can be interpreted in terms of real-world examples. In the statement 'Jasper is a dog', then Jasper is a constant, and dog is a predicate. This can be represented by *dog(Jasper)*, and will have a value of either true or false. The predicate dog has an arity of 1, and the constant Jasper is a predicate with arity 0. Binary predicates take two arguments (an arity of 2), and relate those arguments, for example if 'John is the owner of Jasper', then the predicate *ownerOf(John,Jasper)* has the value true. In a function the result of a statement is a unique object, for example, for *owner(Jasper)* the result is John (since Jasper is the only dog owned by John). A relation is similar to a function, except that several objects may be contained in the result, for example *inThePark(Today)* will yield many dogs including Jasper. All these formulae are ground terms since they contain no variables. If these concepts are extended from particular to general, then the argument is becomes a variable (rather that a constant), and quantifiers need to be introduced into the description. Hence if we need to encode the statement 'all animals in a given set are dogs', then this might be $\forall x(dog(x))$. Likewise, 'every owner of a dog is a person' can be translated as $\forall x \forall y(dog(x) \wedge ownerOf(y,x) \rightarrow person(y))$. In order to assign a truth value to to these formulae it may be necessary to assign values for the predicates, functions, and constants. Hence, $\forall x(dog(x))$ may be true in some *domains*, and false in others. In first-order logic quantification only takes place over variables.

Figure 1.1. <u>Definitions in propositional logic and predicate logic</u>. (11, 12, 14, 16)

Basic syntax of propositional logic:

If F is a formula, P is an atom, then

$F ::= \top \mid \bot \mid P \mid F \mid \neg F \mid F \vee F \mid F \wedge F \mid F \leftarrow F \mid F \rightarrow F \mid F \leftrightarrow F$ (infix notation)

$F ::= P \mid F \mid \neg F \mid \vee(F,F) \mid \wedge(F,F) \mid \leftarrow(F,F) \mid \rightarrow(F,F) \mid \leftrightarrow(F,F)$ (prefix notation)

Truth table for common logical connectives:

A	B	A∧B	A∨B	A→B
⊤	⊤	⊤	⊤	⊤
⊤	⊥	⊥	⊤	⊥
⊥	⊤	⊥	⊤	⊤
⊥	⊥	⊥	⊥	⊤

Truth table for implication:

A	B	A→B
⊤	B	B
⊥	B	⊤
A	⊤	⊤
A	⊥	¬A

Useful definitions and tautologies of propositional logic:

$A \vee \top \dashv\vDash \top$	$A \wedge \top \dashv\vDash A$	$A \wedge \neg A \dashv\vDash \bot$	$\neg\neg A \dashv\vDash A$
$A \vee \bot \dashv\vDash A$	$A \wedge \bot \dashv\vDash \bot$	$A \vee \neg A \dashv\vDash \top$	$\neg\bot \dashv\vDash \top$
$A \leftrightarrow B \dashv\vDash (A \rightarrow B) \wedge (A \leftarrow B)$			
$A \rightarrow B \dashv\vDash \neg A \vee B$			
$A \vee (B \wedge C) \dashv\vDash (A \vee B) \wedge (A \vee C)$		$A \wedge (B \vee C) \dashv\vDash (A \wedge B) \vee (A \wedge C)$	[Distributive Law]
$\neg(A \wedge B) \dashv\vDash (\neg A \vee \neg B)$		$\neg(A \vee B) \dashv\vDash (\neg A \wedge \neg B)$	[De Morgan's Law]

$(A \rightarrow B) \wedge A \models B$	[Inference rule, modus ponens]

- The symbol F1 $\Dashv\models$ F2 is used where formulae are *logically equivalent*, meaning that, the interpretation is equal for every valuation. A, B and C are propositional literals (or modal formulae).
- For two formulae, X and F, $X \models F$ denotes that F is a *logical consequence* of X; when X is true, then F is true; any model satisfying X, also satisfies F
- Formulae created by simultaneously replacing all occurrences of all instances of any subcomponent are created by *substitution*, for example [y/x] replaces all occurrences of y by x.

Definitions in First-order Predicate Logic:

$\forall xF \Dashv\models \neg\exists x\neg F$	$\forall x\neg F \Dashv\models \neg\exists xF$	$\neg\forall xF \Dashv\models \exists x\neg F$	$\neg\forall x\neg F \Dashv\models \exists xF$

1.3.2 <u>Modal logic</u>. [adapted from 7, 9, 10].

The syntax of the propositional modal logic considered in this study is formulated by adding two unary operators, box □ (or the *necessity* operator) and diamond ◇ (or the *possibility* operator) to the syntax of propositional logic. □ is a quantifier similar to universal quantification in predicate logic, and ◇ is similar to existential quantification. In modal logic, statements are true, or false, or somewhere in-between (the are modes of truth like ‘possibly false’ or ‘true in the future’ [see 9]). Concepts can be expressed in first-order logic, but is often easier to use the expressiveness of modal logic to model real world concepts. The propositional modal logic described here can be extended to description logics, temporal logics, etc [see 27].

The semantics of modal logic is best visualized in terms directed graphs (di-graphs) used to illustrate Kripke semantics [24]. A Kripke frame F is an ordered pair (W, R), where W is the (non-empty) set of *possible worlds*, and R is a binary *accessibility relation* connecting some of those worlds ($(w_1,w_2) \in R$ is taken to mean that world w_2 is accessible from world w_1). In a di-graph, the nodes represent possible worlds and the edges represent the relationships between these worlds. Such figures can be taken to illustrate transitions between conceivable states (a view which is useful when model checking is applied to program verification). For example, the *frame* F = {W={1,2,3,4}, R={(1,3), (2,3), (3,2), (2,2)}} is illustrated in figure 1.2. Here for example, worlds 2 and 3 are successor worlds of world 2. In a *model* M = (W, R, I), the frame can be is given a particular *interpretation function*, for example I = {(p,{1,4}), (q,{2,3})}. The labeling of the nodes in the di-graph represents these *valuations*. In figure 1.2, for example, p is true in world 1 (so, $1 \in I(p)$), and false in world 2.

In Kripke semantics, the box operator (□ϕ) takes as an argument a particular world, and returns true if the formula (ϕ) is true in *all* the direct successor worlds, of the starting world, that are defined by the accessibility relation. In a similar way, the diamond operator returns true if a particular formula is true in *at-least-one* possible successor world of the starting world. Hence, if we consider the statement $M,1 \models \Box\Box p \vee \Diamond\neg p$ (applied to the frame in figure 1.2), meaning that the formula $\Box\Box\Box p\vee\Diamond\neg p$ is true (*holds*) at world 1, then this can be verified by visual inspection: It is possible to see that $\Box\Box\Box p$ is false (consider the successive transitions from worlds 1-to-3-to-2-to-2 in which p is false; likewise for successive steps 1-to-3-to-2-to-3), but that $\Diamond\neg p$ is true (consider the transition from worlds 1-to-3 in which p is false), and so the disjunction is true.

A more formal derivation procedure has been developed for determining the truth of modal formulae. Modal formulae are subject to the *Model Checking* algorithm [7], in which the structure of the modal formula, taking account of the properties of a given frame, is expanded inductively using the transformations seen in figure 1.3. The generality of proofs can be extended to cover an arbitrary model, an arbitrary world, or at the most general, all Kripke frames. (For properties valid in all frames see figure 1.4). Two inter-related properties can be determined for modal formulae [7]:

- A modal formula ϕ is *satisfiable* iff in is some model and some world $M,w\models\phi$ is true.
- A modal formula ϕ is valid ($\models\phi$) iff it is true in all Kripke frames, for every model and every world (also known as *global satisfiability*).

Validity and satisfiability are related. Often the satisfiability of the negated formula is checked, since a modal formula ϕ is valid iff $\neg\phi$ is *unsatisfiable* (and ϕ is satisfiable iff $\neg\phi$ is not valid).

In standard modal logic the two formulae *axiom K* and *necessitation* (figure 1.4) are valid for all Kripke frames. They represent the weakest restriction that can be placed on the properties of Kripke frames. Subclasses of frames can be defined by adding other constraints, which can often be represented in terms of geometric constraints (correspondence properties) on the accessibility relation (R). Some common examples are listed in figure 1.5. (Others will be seen later in this study). The properties of these Kripke frames can of course be modeled in di-graphs. For example, in figure 1.7, it can be seen that axiom D (seriality) excludes the existence of dead-end worlds (that is, worlds which are unable to see any other world). It can be demonstrated that the correspondence properties listed in figure 1.6 are equivalent to modal formulae. These modal formulae are not valid in axiom K, but are however theorems of (that is, valid in) the subclass of frames restricted by the appropriate correspondence property. Hence they are known as *modal axioms*. In some cases the relationship between a modal axiom and a correspondence property can be determined automatically [10, 15, 30]. One simple algorithm takes a modal axiom of the type $\Diamond^h\Box^i\phi\rightarrow\Box^j\Diamond^k\phi$ (where, for example, values h=0, i=1, j=2, k=3 gives an axiom $\Box\phi\rightarrow\Box\Box\Diamond\Diamond\Diamond\phi$), and defines the equivalent correspondence property as

$\boldsymbol{\forall vyz\ (R^h(v,y)\wedge R^j(v,z))\rightarrow \exists x(R^i(y,x)\wedge R^k(z,x))}$ (where R^0 means equivalence($\approx$); and R^2 means composition of R(x,y) with itself, giving $\exists z(R(x,z)\wedge R(z,y))$, etc) [Scott-Lemmon algorithm, 15, 30]. Using this algorithm, the equivalence of the modal axioms and correspondence properties listed in figure 1.6 can be demonstrated, as shown in figure 1.5. There are more sophisticated algorithms available [10, 15, 26] that can generate first-order correspondence properties from other classes of target modal axiom, for example from Sahlqvist formulae, as implemented in the SCAN tool [19]. It is also important to note that there are many classes of Kripke frame, restricted by a modal axiom, for which there is no first-order correspondence property (for example, McKinsey's axiom M, $\Box\Diamond\phi\rightarrow\Diamond\Box\phi$). Likewise, there are many correspondence properties do not have modal axiom equivalents (for example irreflexivity, $\forall x\in W, (x,x)\notin R$) [7]. Note also that in the general case, modal axioms are second order properties [26].

Figure 1.2 <u>A directed graph illustrating a Kripke frame</u>. An example of a Kripke frame. It is used to illustrate features of modal logic in section 1.3.2 and figure 1.3.

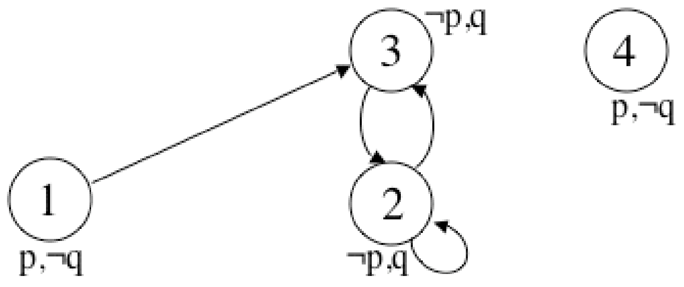


Figure 1.3 <u>The Model Checking Algorithm</u>. The definition of the model checking algorithm is given, followed by an example of use of the algorithm, applied to the frame in figure 1.2.

Inductive Definitions for Model Checking:

•	$M,w \models p$	iff	$w\in I(p)$	(where p is a propositional variable)
•	$M,w \models \top$			
•	$M,w \not\models \bot$			
•	$M,w \models \neg\phi$	iff	$M,w \not\models \phi$	
•	$M,w \models (\phi * \psi)$	iff	$M,w \models \phi * M,w \models \psi$	(where $* \in \{\wedge\ \vee\ \rightarrow\ \leftrightarrow\}$)
•	$M,w \models \Box\phi$	iff	$\forall w`\in W, (w,w`)\in R \rightarrow M,w` \models \phi$	
•	$M,w \models \Diamond\phi$	iff	$\exists w`\in W, (w,w`)\in R \wedge M,w` \models \phi$	all taken from [7]

The following formula (in the frame in figure 1.2) is false, M, 1 $\models \Box\Box p$

iff $\forall w\in\{1, 2, 3, 4\}, (1,w)\in R\rightarrow M, w \models \Box p$ …unfold definition □

iff $((1, 1)\in R\rightarrow M,1 \models \Box p)\wedge((1, 2)\in R\rightarrow M,2 \models \Box p)\wedge((1, 3)\in R\rightarrow M,3 \models \Box p)\wedge((1, 4)\in R\rightarrow M,4 \models \Box p)$
…expand quantifier

iff $(\bot\rightarrow M,1\models\Box p)\wedge(\bot\rightarrow M,2\models\Box p)\wedge(\top\rightarrow M,3\models\Box p)\wedge(\bot\rightarrow M,4\models\Box p)$ …substituting values from R

iff $\top\wedge\top\wedge(M,3\models\Box p)\wedge\top$

iff $M,3\models\Box p$

iff $(\forall w\in\{1, 2, 3, 4\}, (3,w)\in R\rightarrow M, w \models p$ …unfold definition □

iff $(\forall w\in\{1, 2, 3, 4\}, (3, w)\in R\rightarrow w\in I(p)$ …unfold, truth of p

iff $((3, 1)\in R\rightarrow 1\in I(p)\wedge(3, 2)\in R\rightarrow 2\in I(p)\wedge(3, 3)\in R\rightarrow 3\in I(p)\wedge(3, 4)\in R\rightarrow 4\in I(p)$ …expand quantifier

iff $((\bot\rightarrow\top)\wedge(\top\rightarrow\bot)\wedge(\bot\rightarrow\bot)\wedge(\bot\rightarrow\top)$ …substituting values from R and I

iff $(\top\wedge\bot\wedge\top\wedge\top)$

iff $\bot$

Figure 1.4 <u>Valid propositional modal formulae</u>.

These properties are valid in all possible Kripke frames) [7, 9]

- All propositional tautologies
- $\models\Diamond\phi \leftrightarrow \neg\Box\neg\phi$ $\models\neg\Diamond\phi \leftrightarrow \Box\neg\phi$ $\models\Diamond\neg\phi \leftrightarrow \neg\Box\phi$
- $\models\Box\top \leftrightarrow \top$
- $\models\Diamond\bot \leftrightarrow \bot$
- $\models\Box(\phi\rightarrow\psi) \rightarrow (\Box\phi\rightarrow\Box\psi)$ or $\models(\Box(\phi\rightarrow\psi)\wedge\Box\phi) \rightarrow \Box\psi$ [*axiom K*]
- $\models\Box(\phi\wedge\psi) \leftrightarrow (\Box\phi\wedge\Box\psi)$
- $\models\Diamond(\phi\vee\psi) \leftrightarrow (\Diamond\phi\vee\Diamond\psi)$
- $\models\Diamond(\phi\wedge\psi) \rightarrow (\Diamond\phi\wedge\Diamond\psi)$
- $\models(\Box\phi\vee\Box\psi) \rightarrow \Box(\phi\vee\psi)$
- $(\models(\phi\rightarrow\psi)\wedge\models\phi)\rightarrow\models\psi$ [*necessitation*]
- $\models\phi \rightarrow \models\Box\phi$ [*modus ponens*]

Figure 1.5 <u>The equivalence of some modal axioms and correspondence properties.</u>

The correspondence property of various modal axioms is derived, from the modal representation, using the Scott-Lemmon algorithm. See section 1.3.2 for more information.

Axiom T is $\Box\phi \rightarrow \phi$ with h=0, i=1, j=0, k=0, and correspondence property:

$\forall vyz\, (R^0(v,y)\wedge R^0(v,z))\rightarrow\exists x(R^1(y,x)\wedge R^0(z,x))$

which reduces to … $((v\approx y)\wedge(v\approx z))\rightarrow\exists x(R(y,x)\wedge(z\approx x))$ and $(y\approx z)\rightarrow R(y,z)$

$\forall y\, R(y,y)$ Reflexive

Axiom D is $\Box\phi \rightarrow \Diamond\phi$ with h=0, i=1, j=0, k=1, and correspondence property:

$\forall vyz\, (R^0(v,y)\wedge R^0(v,z))\rightarrow\exists x(R^1(y,x)\wedge R^1(z,x))$

which reduces to … $(y\approx z)\rightarrow\exists x(R(y,x)\wedge R(z,x))$ and $\exists x(R(y,x)\wedge R(y,x))$

$\forall y\exists x(R(y,x))$ Serial

Axiom B is $\Diamond\Box\phi \rightarrow \phi$ with h=1, i=1, j=0, k=0, and correspondence property:

$\forall vyz\, (R^1(v,y)\wedge R^0(v,z))\rightarrow\exists x(R^1(y,x)\wedge R^0(z,x))$

which reduces to … $(R(v,y)\wedge(v\approx z))\rightarrow\exists x(R(y,x)\wedge(z\approx x))$

$\forall yz\, R(z,y)\rightarrow R(y,z)$ Symmetry

Axiom 4 is $\Box\phi \rightarrow \Box\Box\phi$ with h=0, i=1, j=2, k=0, and correspondence property:

$\forall vyz\, (R^0(v,y)\wedge R^2(v,z))\rightarrow\exists x(R^1(y,x)\wedge R^0(z,x))$

which reduces to … $((v\approx y)\wedge R(v,u)\wedge R(u,z))\rightarrow\exists x(R(y,x)\wedge(z\approx x))$

$\forall uyz\, (R(y,u)\wedge R(u,z))\rightarrow R(y,z)$ Transitive

Axiom 5 is $\Diamond\Box\phi \rightarrow \Box\phi$ with h=1, i=1, j=1, k=0, and correspondence property:

$\forall vyz\, (R^1(v,y)\wedge R^1(v,z))\rightarrow\exists x(R^1(y,x)\wedge R^0(z,x))$

which reduces to … $(R(v,y)\wedge R(v,z))\rightarrow\exists x(R(y,x)\wedge(z\approx x))$

$\forall vyz\, (R(v,y)\wedge R(v,z))\rightarrow R(y,z)$ Euclidean

Axiom alt_1 is $\Diamond\phi\rightarrow\Box\phi$ with h=1, i=0, j=1, k=0, and correspondence property:

$\forall vyz\, (R^1(v,y)\wedge R^1(v,z))\rightarrow\exists x(R^0(y,x)\wedge R^0(z,x))$

which reduces to … $(R(v,y)\wedge R(v,z))\rightarrow\exists x((y\approx x)\wedge(z\approx x))$

$\forall vyz\, (R(v,y)\wedge R(v,z))\rightarrow(y\approx z)$ Functional

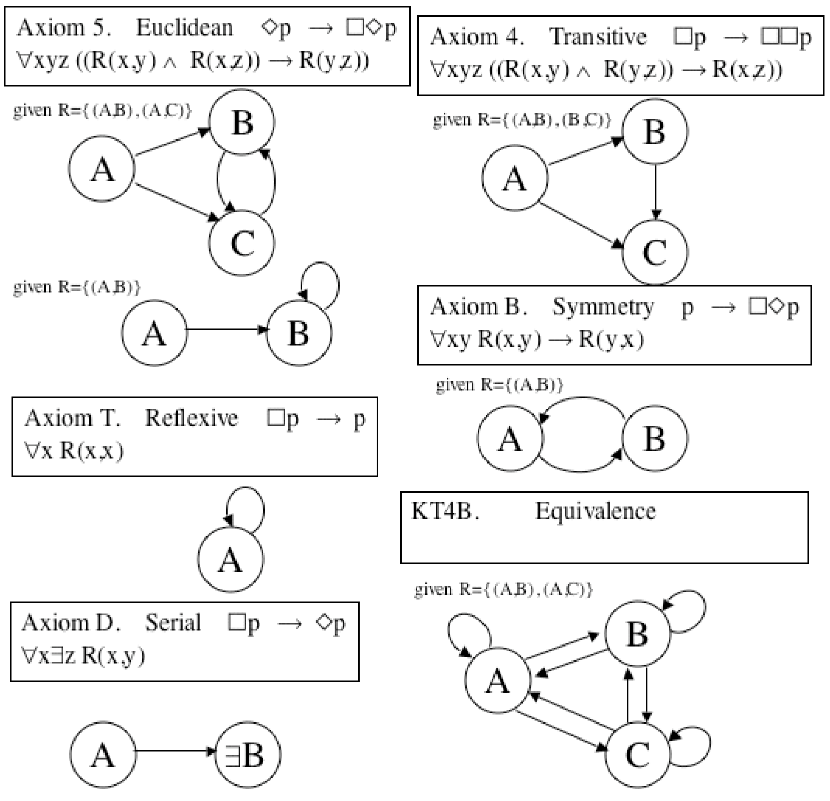
Figure 1.6 Modal axioms and correspondence properties. [adapted from 1, 7, 9]

The restrictions on a Kripke frame, the first-order logic correspondence property and the modal representation of several axioms are given.

Axiom	Geometric description	Correspondence property as a restriction on Kripke frame, F = (W,R)	Correspondence property in first-order logic	Modal axiom
T	Reflexive	$\forall x\in W, (x,x)\in R$	$\forall x\, R(x,x)$	$\Box\phi \rightarrow \phi$
D	Serial	$\forall x\in W,\ \exists y\in W, (x,y)\in R$	$\forall x\exists y\, R(x,y)$	$\Box\phi \rightarrow \Diamond\phi$ or $\Diamond\top$
B	Symmetry	$\forall x,y\in W, ((x,y)\in R \rightarrow (y,x)\in R)$	$\forall xy(R(x,y)\rightarrow R(y,x))$	$\phi \rightarrow \Box\Diamond\phi$ or $\Diamond\Box\phi \rightarrow \phi$
4	Transitive	$\forall x,y,z\in W, (((x,y)\in R\wedge(y,z)\in R)\rightarrow(x,z)\in R)$	$\forall xyz((R(x,y)\wedge R(y,z))\rightarrow R(x,z))$	$\Box\phi \rightarrow \Box\Box\phi$
5	Euclidean	$\forall x,y,z\in W, (((x,y)\in R\wedge(y,z)\in R)\rightarrow(x,z)\in R)$	$\forall xyz((R(x,y)\wedge R(x,z))\rightarrow R(y,z))$	$\Diamond\phi \rightarrow \Box\Diamond\phi$ or $\Diamond\Box\phi \rightarrow \Box\phi$
alt_1	Functionality	$\forall x,y,z\in W, (((x,y)\in R\wedge(x,z)\in R)\rightarrow(y\approx z))$	$\forall xyz((R(x,y)\wedge R(x,z))\rightarrow(y\approx z))$	$\Diamond\phi\rightarrow\Box\phi$

Figure 1.7 Examples of Kripke frames in which modal axioms hold:

Kripke frames are given illustrating the 'geometric' nature of correspondence properties. In each case starting nodes are given, to which the axiom is applied.



1.4 Translation of modal logic into first-order logic [adapted from 1, 7].

In order to use tools such as SPASS to attempt to prove, or to disprove, the satisfiability of modal formulae it is necessary to first translate the modal formula into first-order logic. Such a translation has been available for some time [25, 29, 31, 32, 37, 38], and has been implemented in SPASS. This semantic translation preserves the truth state of any modal formula (and in particular the property of satisfiability) under first-order resolution (see section 1.5). The semantic translation function π is defined as seen in figure 1.8, taking as arguments the modal formula and a first-order variable symbol.

Propositional variables are mapped to a unary predicate symbol (for example, p maps to Q_p), and accessibility relations are mapped to binary predicates (R is used in the uni-modal case below). The function π is applied inductively (iteratively) until no more transformations are possible. Modal axioms are introduced into the translation as the *correspondence properties* (see table 1.6). These correspondence properties are incorporated into the formula to be submitted to SPASS as follows.

$$\bigwedge(\text{Correspondence Property for each Axiom}) \rightarrow \forall x\ \pi(\phi,x)$$

So for example, if a modal formula φ is valid in KT45, then the following is valid in first-order logic:

$$(\forall x R(x,x) \wedge \forall xyz(((R(x,y) \wedge R(x,z)) \rightarrow R(y,z)) \wedge \forall xyz((R(x,y) \wedge R(x,z)) \rightarrow R(y,z))) \rightarrow \forall x.\pi(\phi,x)$$

Definition of the semantic translation provides the opportunity to follow the derivation of these correspondence properties. The derivations in figure 1.9 rely upon rearranging the translation of the modal formula representing the axiom, to give a formula in which quantifiers can be eliminated by comparison to a propositional formula (provable by truth tables), and from which the correspondence properties can then be derived by substitution. These are my own derivations of these properties. A more rigorous derivation from the literature is shown in figure 1.10 for axiom 4.

The axiomatic translation implemented in this study builds upon the ideas presented in this translation. Note, that there are other translation methods (for example, functional, semi-functional, and optimized-functional translations), some of which have previously been implemented in SPASS [17, 25].

Figure 1.8 <u>Inductive definition of the function, π</u>.

The definition of the function π is given. It is used to perform the semantic translation of modal logic to first-order logic.

$\pi(p,x) = Q_p(x)$	$\pi(\neg\phi,x) = \neg\pi(\phi,x)$
$\pi(\top,x) = \top$	$\pi(\bot,x) = \bot$
$\pi(\phi * \psi ,x) = \pi(\phi,x) * \pi(\psi,x)$ where $* \in \{\rightarrow,\leftrightarrow,\vee,\wedge\}$	
$\pi(\Box\phi,x) = \forall y(R(x,y) \rightarrow \pi(\phi,y))$	$\pi(\Diamond\phi,x) = \exists y(R(x,y) \wedge \pi(\phi,y))$
x and y are distinct first order logic variables; φ and ψ are any modal formula	

Figure 1.9 <u>Derivations of the correspondence property for various modal axioms</u>.

Correspondence properties are derived below, as described in the text of section 1.4.

Axiom T:	$\pi(\Box \mathbf{p} \rightarrow \mathbf{p},x)$	=	$\forall x(\ \pi(\Box p,x) \rightarrow\ \pi(p,x)\ )$
		=	$\forall x(\ \forall y(R(x,y) \rightarrow \pi(p,y)) \rightarrow\ Q_p(x)\ )$
		=	$\forall x(\ \forall y(R(x,y) \rightarrow Q_p(y)) \rightarrow\ Q_p(x)\ )$
		=	$\forall x(\ \neg\forall y(R(x,y) \rightarrow Q_p(y)) \vee\ Q_p(x)\ )$
		=	$\forall x(\ \exists y(\neg(R(x,y) \rightarrow Q_p(y)) \vee\ Q_p(x))\ )$
		=	$\forall x(\ \exists y(\ (R(x,y) \rightarrow Q_p(y)) \rightarrow\ Q_p(x))\ )$

$= \quad \forall x(((R(x,x) \rightarrow Q_p(x)) \rightarrow Q_p(x)))$ substituting [y/x]

$\Dashv \quad \mathbf{\forall x(R(x,x))}$ $A \models (A \rightarrow B) \rightarrow B$

proof by refutation $\neg((A \rightarrow B) \rightarrow B) \wedge A$

$\equiv \neg(\neg(\neg A \vee B) \vee B) \wedge A \equiv ((\neg A \vee B) \wedge \neg B) \wedge A \equiv ((\neg B \wedge \neg A) \vee (\neg B \wedge B)) \wedge A$

$\equiv ((\neg B \wedge \neg A) \vee (\bot)) \wedge A \equiv \neg B \wedge \neg A \wedge A \equiv \neg B \wedge \bot \equiv \bot$

Axiom D: $\pi(\mathbf{\Box p \rightarrow \Diamond p}, x) \quad = \quad \forall x\, \pi(\Box p, x) \rightarrow \pi(\Diamond p, x)$

$= \quad \forall x(\forall y(R(x,y) \rightarrow \pi(p,y)) \rightarrow \exists z(R(x,z) \wedge \pi(p,z)))$

$= \quad \forall x(\forall y(R(x,y) \rightarrow Q_p(y)) \rightarrow \exists z(R(x,z) \wedge Q_p(z)))$

$= \quad \forall x(\exists y(R(x,y) \wedge \neg Q_p(y)) \vee \exists z(R(x,z) \wedge Q_p(z)))$

$= \quad \forall x \exists y \exists z((R(x,y) \rightarrow Q_p(y)) \rightarrow (R(x,z) \wedge Q_p(z)))$

$= \quad \mathbf{\forall x \exists y(R(x,y))}$ substituting [z/y],

and $(A \rightarrow B) \rightarrow (A \wedge B) \Dashv\models A$

proof: $(A \rightarrow B) \rightarrow (A \wedge B)$

$\equiv \neg(\neg A \vee B) \vee (A \wedge B) \equiv (A \wedge \neg B) \vee (A \wedge B) \equiv A \wedge (\neg B \vee B) \equiv A \wedge \top \equiv A$

Axiom B: $\pi(\mathbf{p \rightarrow \Box\Diamond p}, x) \quad = \quad \forall x\, \pi(p,x) \rightarrow \pi(\Box\Diamond p, x)$

$= \quad \forall x(Q_p(x) \rightarrow \forall y(R(x,y) \rightarrow \pi(\Diamond p, y)))$

$= \quad \forall x(Q_p(x) \rightarrow \forall y(R(x,y) \rightarrow \exists z(R(y,z) \wedge \pi(p,z))))$

$= \quad \forall x(Q_p(x) \rightarrow \forall y(R(x,y) \rightarrow \exists z(R(y,z) \wedge Q_p(z))))$

$= \quad \forall x \forall y \exists z((R(x,y) \wedge Q_p(x)) \rightarrow (R(y,z) \wedge Q_p(z)))$

$= \quad \forall x \forall y((R(x,y) \wedge Q_p(x)) \rightarrow (R(y,x) \wedge Q_p(x)))$ substituting [z/x]

$\Dashv \quad \mathbf{\forall xy\ (R(x,y) \rightarrow R(y,x))}$ $A \rightarrow B \models (A \wedge C) \rightarrow (B \wedge C)$

proof by refutation $(A \rightarrow B) \wedge \neg((A \wedge C) \rightarrow (B \wedge C))$

$\equiv (\neg A \vee B) \wedge \neg(\neg(A \wedge C) \vee (B \wedge C)) \equiv (\neg A \vee B) \wedge ((A \wedge C) \wedge \neg(B \wedge C)) \equiv (\neg A \vee B) \wedge (A \wedge (C \wedge (\neg B \vee \neg C)))$

$\equiv (\neg A \vee B) \wedge (A \wedge ((C \wedge \neg B) \vee (C \wedge \neg C))) \equiv (\neg A \vee B) \wedge (A \wedge ((C \wedge \neg B) \vee \bot)) \equiv (\neg A \vee B) \wedge (A \wedge \neg B \wedge C)$

$\equiv (\neg A \vee B) \wedge \neg(\neg A \vee B) \wedge C \equiv \bot \wedge C \equiv \bot$

Axiom 4: $\pi(\mathbf{\Box p \rightarrow \Box\Box p}, x) \quad = \quad \forall x\, \pi(\Box p, x) \rightarrow \pi(\Box\Box p, x)$

$= \quad \forall x(\forall u(R(x,u) \rightarrow \pi(p,u)) \rightarrow \forall y(R(x,y) \rightarrow \pi(\Box p, y)))$

$= \quad \forall x(\forall u(R(x,u) \rightarrow \pi(p,u)) \rightarrow \forall y(R(x,y) \rightarrow \forall z(R(y,z) \rightarrow \pi(p,z))))$

$= \quad \forall x(\forall u(R(x,u) \rightarrow Q_p(u)) \rightarrow \forall y(R(x,y) \rightarrow \forall z(R(y,z) \rightarrow Q_p(z))))$

$= \quad \forall x(\neg\forall u(\neg R(x,u) \vee Q_p(u)) \vee \forall y(\neg R(x,y) \vee \forall z(\neg R(y,z) \vee Q_p(z))))$

$= \quad \forall x(\exists u(R(x,u) \wedge \neg Q_p(u)) \vee \forall y(\neg R(x,y) \vee \forall z(\neg R(y,z) \vee Q_p(z))))$

$= \quad \forall x(\exists u \forall y \forall z\ ((R(x,y) \wedge R(y,z) \wedge \neg Q_p(z)) \rightarrow (R(x,u) \wedge \neg Q_p(u))))$

$= \quad \forall x \forall y \forall z\ ((R(x,y) \wedge R(y,z) \wedge \neg Q_p(z)) \rightarrow (R(x,z) \wedge \neg Q_p(z)))$ substituting [u/z]

$\Dashv \quad \mathbf{\forall xyz\ ((R(x,y) \wedge R(y,z)) \rightarrow R(x,z))}$ $(A \wedge B) \rightarrow C \models (A \wedge B \wedge D) \rightarrow (C \wedge D)$

proof by refutation $((A \wedge B) \rightarrow C) \wedge \neg((A \wedge B \wedge D) \rightarrow (C \wedge D))$

$\equiv (\neg(A \wedge B) \vee C) \wedge \neg(\neg(A \wedge B \wedge D) \vee (C \wedge D)) \equiv (\neg(A \wedge B) \vee C) \wedge ((A \wedge B \wedge D) \wedge \neg(C \wedge D))$

$\equiv (\neg(A \wedge B) \vee C) \wedge ((A \wedge B) \wedge ((D \wedge \neg C) \vee (D \wedge \neg D))) \equiv (\neg(A \wedge B) \vee C) \wedge ((A \wedge B) \wedge ((D \wedge \neg C) \vee \bot))$

$\equiv ((\neg(A \wedge B) \vee C) \wedge \neg C) \wedge (A \wedge B \wedge D)) \equiv ((C \wedge \neg C) \vee (\neg(A \wedge B) \wedge \neg C)) \wedge (A \wedge B \wedge D)) \equiv (\neg(A \wedge B) \wedge (A \wedge B) \wedge D \wedge \neg C) \equiv \bot$

Axiom 4^κ: $\Box p \rightarrow \Box^\kappa \Box p$ is the general case. (Note axiom 4^K is equivalent to axiom 4^1).

$\forall x(\pi(\Box p \rightarrow \Box^\kappa \Box p, x) \models \forall xy\ (R^{\kappa+1}(x,y) \rightarrow R(x,y))$

Axiom 5: $\pi(\mathbf{\Diamond\Box p \rightarrow \Box p}, x) \quad = \quad \forall x\, \pi(\Diamond\Box p, x) \rightarrow \pi(\Box p, x)$

$= \quad \forall x(\exists y(R(x,y) \wedge \pi(\Box p, y)) \rightarrow \forall z(R(x,z) \rightarrow \pi(p,z)))$

$= \quad \forall x(\exists y(R(x,y) \wedge \forall u(R(y,u) \rightarrow \pi(p,u))) \rightarrow \forall z(R(x,z) \rightarrow \pi(p,z))))$

$= \quad \forall x(\exists y(R(x,y) \wedge \forall u(R(y,u) \rightarrow Q_p(u))) \rightarrow \forall z(R(x,z) \rightarrow Q_p(z))))$

$= \quad \forall x(\forall y \neg(R(x,y) \wedge \forall u(\neg R(y,u) \vee Q_p(u))) \vee \forall z(\neg R(x,z) \vee Q_p(z))))$

$= \quad \forall x(\forall y(\neg R(x,y) \vee \exists u(R(y,u) \wedge \neg Q_p(u))) \vee \forall z(\neg R(x,z) \vee Q_p(z))))$

$= \quad \forall xyz \exists u(\neg R(x,y) \vee (R(y,u) \wedge \neg Q_p(u))) \vee (\neg R(x,z) \vee Q_p(z)))$

$= \quad \forall xyz \exists u((R(x,y) \wedge R(x,z) \wedge \neg Q_p(z)) \rightarrow (R(y,u) \wedge \neg Q_p(u))))$

$= \quad \forall xyz((R(x,y) \wedge R(x,z) \wedge \neg Q_p(z)) \rightarrow (R(y,z) \wedge \neg Q_p(z))))$ substituting [u/z]

$\Dashv \quad \mathbf{\forall xyz\ ((R(x,y) \wedge R(x,z)) \rightarrow R(y,z))}$ $(A \wedge B) \rightarrow C \models (A \wedge B \wedge D) \rightarrow (C \wedge D)$

Axiom 5^κ: $\neg\Box^\kappa\neg\Box p \rightarrow \Box p$ is the general case. (Note axiom 5^κ is equivalent to axiom 5^1).

$\forall x(\pi(\neg\Box^\kappa\neg\Box p \rightarrow \Box p, x) \models \forall xyz\ ((R^\kappa(x,y) \wedge R(x,z)) \rightarrow R(y,z))$

Axiom alt_1: $\pi(\mathbf{\Diamond p \rightarrow \Box p}, x) \quad = \quad \forall x\, \pi(\Diamond p, x) \rightarrow \pi(\Box p, x)$

$= \quad \forall x(\exists y(R(x,y) \wedge \pi(p,y)) \rightarrow \forall z(R(x,z) \rightarrow \pi(p,z)))$

$= \quad \forall x(\exists y(R(x,y) \wedge Q_p(y)) \rightarrow \forall z(R(x,z) \rightarrow Q_p(z)))$

$= \quad \forall x(\forall y \neg(R(x,y) \wedge Q_p(y)) \vee \forall z(\neg R(x,z) \vee Q_p(z)))$

$= \quad \forall x \forall y \forall z(\neg R(x,y) \vee \neg Q_p(y) \vee \neg R(x,z) \vee Q_p(z))$

$= \quad \forall x \forall y \forall z(\neg R(x,y) \vee \neg R(x,z) \vee \neg Q_p(y) \vee Q_p(z))$

$= \quad \forall x \forall y \forall z((R(x,y) \wedge R(x,z)) \rightarrow (Q_p(y) \rightarrow Q_p(z)))$

$\dashv \quad \boldsymbol{\forall xyz ((R(x,y) \wedge R(x,z)) \rightarrow (y \approx z))}$

if y equivalent to z, then statement is tautology

Axiom alt$_1^{\kappa1,\kappa2}$: $\neg\Box^{\kappa1}\Box p \rightarrow \Box^{\kappa2}\Box\neg p$ is the general case.

(Note axiom alt$_1^{\kappa,\kappa}$ is equivalent to axiom alt$_1^{0,0}$).

$\forall x(\pi(\neg\Box^{\kappa1}\Box p \rightarrow \Box^{\kappa2}\Box\neg p,x) \vDash \forall xyz ((R^{\kappa1+1}(x,y) \wedge R^{\kappa2+1}(x,z)) \rightarrow (y \approx z))$

Figure 1.10 **<u>A derivation of the correspondence property for axiom 4 from the literature</u>**. A rigorous way to derive correspondence properties using second order quantifier elimination techniques applied to semantic translation is shown, adapted from [26]. This process has been automated in the SCAN algorithm. The translated formula is negated, transformed to clausal form, and subject to resolution (see section 1.5).

For axiom 4: $\forall p(\boldsymbol{\Box p \rightarrow \Box\Box p}) \quad = \quad \forall Q_p \forall x\ \pi(\Box p \rightarrow \Box\Box p,x)$

$= \quad \forall Q_p \forall x(\forall u(R(x,u) \rightarrow Q_p(u)) \rightarrow$

$\forall y(R(x,y) \rightarrow \forall u(R(y,z) \rightarrow Q_p(z))))$ see above

so in the negation (for resolution, see section 1.4)

$\exists Q_p \exists x\ \neg\pi(\Box p \rightarrow \Box\Box p,x) = \quad \exists Q_p \exists x(\forall u(R(x,u) \rightarrow Q_p(u)) \wedge \exists y(R(x,y) \wedge \exists z(R(y,z) \wedge \neg Q_p(z))))$

and in clausal form (following Skolemization)

	1.	$\neg R(a,u) \vee Q_p(u)$	
	2.	$R(a,b)$	
	3.	$R(b,c)$	
	4.	$\neg Q_p(c)$	
applying C-resolution	5.	$\neg R(a,u) \vee c \not\approx u$	from 1, 4
	6.	$\neg R(a,c)$	from 5, by c-elimination

allowing clauses 1 and 4 to be deleted by purification, leaving clauses 2, 3, 6: $R(a,b) \wedge R(b,c) \wedge \neg R(a,c)$

relating clauses 2, 3, and 6 to the original formula, by reverse Skolemisation gives

$\exists v_a \exists v_b \exists v_c\ [\ R(v_a,v_b) \wedge R(v_b,v_c) \wedge \neg R(v_a,v_c)\ ]$,

which is then negated transitive property, so negation (to reverse the first negation) gives

$\forall v_a \forall v_b \forall v_c[\ \neg R(v_a,v_b) \vee \neg R(v_b,v_c) \vee R(v_a,v_c)\ ] \ = \ \forall v_a \forall v_b \forall v_c[\ (R(v_a,v_b) \wedge R(v_b,v_c)) \rightarrow R(v_a,v_c)\ ]$

1.5 <u>Resolution</u>. [adapted from 5, 7, 11, 12]

Figure 1.11 outlines the principle of resolution for propositional and then predicate formulae. SPASS is a resolution prover for first-order logic based on *refutation*. In SPASS, if a formula ϕ is valid, then $\neg\phi$ is unsatisfiable (the formula is negated for proof by refutation), and the result of resolution is *Proof Found*. On the other hand, if a formula $\neg\phi$ is satisfiable, then the result of resolution is *Completion Found*. SPASS can assess the satisfiability of modal formulae by performing a semantic translation of the modal formula into first-order logic (that is then submitted to the resolution prover), and by including the correspondence properties of modal axioms. Hence, a modal formula ϕ is valid in the series of logics $K\Sigma$ (where Σ represents a series of axioms) iff $\bigwedge_{A\in\Sigma}(\mathrm{Corr}_A) \wedge \exists x.\pi(\neg\phi,x)$ is unsatisfiable in first order logic.

Many alternative proof systems exist (for example Hilbert and sequent calculi, and natural deduction, [7, 5, 12]). The only one of interest to this study is tableau, in which a tree-like diagram is created by exhaustive application of pre-formed expansion rules to the input clauses, and in which an attempt is made to close each branch of the tree that appears by derivation of an *empty clause*. If all branches are closed then the input clauses are unsatisfiable. This method is closely related to resolution. Under certain circumstances, the proof steps in resolution can be used to derive a tableau [1].

Figure 1.11 **<u>Principles of resolution:</u>** [adapted from 5, 7, 11 (especially chapters 2 & 7), 12 especially chapter 6)]. A brief overview of the main features of resolution is given. The reader is directed to further information.

<u>Some Definitions:</u>

- A formula F is *satisfiable* (true in one or more interpretations) iff ¬F is *falsifiable* (false in one or more interpretations). $X \nvDash F$ iff $X \wedge \neg F$ is satisfiable.
- F is *valid* (a tautology) iff ¬F is *unsatisfiable* (always false). $X \vDash F$ iff $X \wedge \neg F$ is unsatisfiable.
- A system is *sound* if it is only possible, using the rules of the system, to draw correct conclusions regarding the above properties of any formula. It is *complete* if it is possible to prove all valid formulae with just the given rules. It is *decidable* if all calculations will terminate with a result in finite time (which may however be a very long time) for any arbitrary input.

<u>Propositional Resolution:</u>

- Resolution is a technique derived directly from modus ponens.

A propositional formula is transformed into *conjunctive normal form* (CNF), is a conjugation of disjunctions $((L_1 \vee L_2 \vee \ldots) \wedge (L_n \vee \ldots) \wedge \ldots.)$. A simple algorithm exists to derive CNF from any propositional formula (see [5]). The CNF is often modified to give a stylized representation of the *clauses,* in the format $[L_1;\neg L_2]$, representing the set $\{L_1 \vee \neg L_2\}$

- During *resolution*, *clashing clauses* are identified as complimentary literals, of the form $\boxed{L} \vee \ldots L_n$ (or C_1) and $\boxed{\neg L} \vee \ldots L_m$ (C_2), and then clashing literals are eliminated to give a *resolvent* of the form $Res(C_1,C_2) = L_n \vee L_m$.
- The resolvent clause preserves the (joint) satisfiability state of the parent clauses, and hence resolution gradually pairs-down a set of clauses until either (i) the *empty clause* is derived indicating that the original set of clauses were *unsatisfiable*, or (ii) no further resolution steps are possible, indicating that the original set of clauses were satisfiable.

  Resolution is illustrated below using the same example:

  Res([¬A;¬B;C], [¬C]) = [¬A;¬B]
  Res([¬A;¬B], [B]) = [¬A]
  Res([¬A], [A]) = []

  The *empty clause* "[]" is false, so the clause set is unsatisfiable, and the original logical consequence holds.
- Propositional resolution is sound, complete and decidable. Conversion of formulae into CNF may give rise to an exponential increase in the number of sub-formulae, and resolution itself may produce a large number of steps, potentially making resolution a lengthy process [36]. Many optimizations are available to reduce the size of the clause set before resolution takes place [see 6].

<u>Predicate (first-order) Resolution:</u>

- Predicate resolution follows the rules above, but has the additional complications arising from quantified variables.
- After the CNF is formed, a *prenex* format is generated in which all quantifiers are extracted from the body of the formula. All bound variables are renamed apart, and quantifiers are then extracted. (For an explanation see chapter 8 of [12], chapter 9 of [5]). When possible existential quantifiers are extracted first, since this makes Skolemisation easier.
- Existential quantifiers are removed by *Skolemisation*. An ∃x at the head (leftmost) of the list of quantifiers is removed, and bound x-variables in the body of the formula are replaced by a unique *Skolem constant*. Where ∃x appears behind a list of universal quantifiers, the bound x-variables in the body are replaced by a unique function in each of the universally quantified variables (for example, ∀y∀z∃x raises the replacement of x by f(y,z); an excellent explanation is given on chapter 7 of [11]. Note that Skolemisation preserves satisfiability, but not logical equivalence). Finally all the universal quantifiers are dropped.
- Clashing clauses are resolved by substitution of variables. A *most general unifier* is formulated that describes a set of simultaneous substitutions that allows sub-formulae involving variables to be unified [see chapter 7 of 11 for a excellent explanation].

- Predicate resolution is only complete if another technique, such as positive *factoring* is included. In factoring, duplicate copies of a term are eliminated from any clause to remove repetition.
- Predicate resolution is **not a decision procedure** (and hence may never terminate, that is never produce a result), since for even very simple clause sets there may be an infinite number of unique substitutions possible (for example the clause set $[\neg p(x);p(f(x))][p(a)]$ from [6]).

  A simple example of resolution illustrates these principles.

  $\forall x(P(x)\rightarrow(Q(x)\rightarrow R(x))) \models \forall x((P(x)\wedge Q(x))\rightarrow R(x))$

  $\forall x(P(x)\rightarrow(Q(x)\rightarrow R(x))) \wedge \neg(\forall y\ ((P(y)\wedge Q(y))\rightarrow R(y)))$ renaming apart/negation for refutation

  $\forall x(\neg P(x)\vee\neg Q(x)\vee R(x)) \wedge \exists y(P(y)\wedge Q(y)\wedge \neg R(y))$

  $\exists y(\forall x(\neg P(x)\vee\neg Q(x)\vee R(x)) \wedge (P(y)\wedge Q(y)\wedge \neg R(y)))$

  $\exists y\forall x((\neg P(x)\vee\neg Q(x)\vee R(x)) \wedge (P(y)\wedge Q(y)\wedge \neg R(y)))$

  $\forall x((\neg P(x)\vee\neg Q(x)\vee R(x)) \wedge (P(c)\wedge Q(c)\wedge \neg R(c)))$ c is a Skolem constant

  $[\neg P(x);\neg Q(x);R(x)]\ [P(c)]\ [Q(c)]\ [\neg R(c)]$ clause set

  $\mathrm{Res}([\neg P(c);\neg Q(c);R(c)]\ [P(c)]) = [\neg Q(c);R(c)]$ substituting $[x/c]$

  $\mathrm{Res}([\neg Q(c);R(c)]\ [Q(c)]) = [R(c)]$

  $\mathrm{Res}([R(c)]\ [\neg Q(c);R(c)]\ [\neg R(c)]) = []$ unsatisfiable, so inference holds
- Again, a large number of improvements to the basic resolution are possible, which improve efficiency, and may be capable of avoiding non-terminating calculations. Many of these techniques eliminate unnecessary resolution steps, and hence reduce the space over which to search for a solution. See [28] for a comprehensive review. In terms of this study ordered resolution with selection and hyper-resolution are most relevant.

1.6 Essential knowledge of SPASS. [adapted from 17, 18, 25]

There are a large variety of input modes for SPASS, but the discussion here is restricted to those appropriate for this study. The anatomy of a .dfg file is illustrated by the example in figure 1.12. Both problems and program control options may be input from a .dfg file. The location of a .dfg file is input from the command line as … *SPASS <path>example.dfg*. Key features of the .dfg file in figure 1.12 are described below:

- The sections can include (i) Description (lines 2-7), (ii) Symbols (lines 8-10), (iii) Axiom formulae (lines 11-16), (iv) Conjecture formulae (lines 17-22), (v) Settings (lines 23-34).
- **Symbols** list: Defines symbols (and functions) occurring in the subsequent first order and modal formulae. Contents of interest to this study are

(i) *predicates*: used to define non-standard symbols used in the formulae sections, in the format `(symbol, arity)`

(ii) *translpairs*: (Not shown in the example). Used to map a first-order predicate symbols to a modal logic propositional symbol. In this study a typical use is, for example, `translpairs[(r,R)]` with predicates[(R,2),(r,0)…]., linking the nullary modality index symbol `r` from modal formulae, to the binary first order predicate `R` in a correspondence property defined in the dfg file.

- **Formulae** (for first order logic problems) or **prop_formulae** (for modal problems, converted by the software into first-order logic problems; see figure 7.1 for the typical syntax) may occur in either the **axioms** or **conjectures** lists. Formulae are entered in prefix notation, and for this study, the operators used in first-order formulae are `and`, `or`, `not`, `implies`, `implied`, `equiv`, `equal`, `true`, `false`, with quantifiers `forall` and `exists` (requiring two arguments; a list of terms, and the quantified sub-formulae), and in modal prop_formulae the quantifiers are `box` and `dia`, For more details see figure 7.1 and [17]. Conjectures and axioms are combined as described in section 5.2.

• **Settings** list: flags are set which control command-line options (see figure 7.1; [17]). Many options can also be set at the command-line: *SPASS –AnOption=value filename.dfg*.

• The **precedence** of symbols defined in the symbols list can be defined with the syntax `set_precedence(a,B,z,C).`

Typical options used to run SPASS are set in the dfg file in figure 1.12 and can be seen in figure 7.1. Some options important for this study are listed below [17, 18]. In most cases, setting the flag value to 1 enables the option, and 0 disables the option. The reader is directed to the following options in [18]; for control of the configuration of the resolution technologies in SPASS see *-Auto, -Splits, -Sorts=0*, *-CNFOptSkolem*, *-CNFStrSkolem*, *-CNFRenaming*; for options modifying the amount of information printed to the user see *–DocProof*, *-PProblem*, *-PGiven*, *-PKept*, *-PFlags*; and since first-order resolution is not decidable, it is wise to bound the resources allocated to any problem with *-TimeLimit*.

An example of the output produced in seen in figure 1.12, and illustrates some features of SPASS output. Of particular interest are:

• The input problem is reported in a modified clausal form (lines 1-5), which correspond to $\top\rightarrow P(c)$, $\top\rightarrow Q(c)$, $R(c)\rightarrow\bot$, $(Q(x)\wedge P(x))\rightarrow Q(c)$. The maximal literal of a clause is marked with *.

• The resolution proof is reported in lines 40-46, with the clauses used listed first, followed by the resolution steps (in figure 1.12, for example, line 45 is Res(2,4)=[$\neg$P(c); R(c)]), and the final line (46) describes the derivation of the empty clause.

• SPASS is a refutation-based resolution prover for first order logic. The final result (line 30) may contain (i) `SPASS beiseite: Proof found` if the input formulae have a model and are valid (`unsatisfiable`), (ii) `SPASS beiseite: Completion found` if the input formula is not valid (`satisfiable`), and since (iii) validity is not a decidable problem, the calculation may run forever, or be terminated without result by a time limit.

• The total CPU-time used is reported as seen on line 34, and further breakdowns of execution times by module (for example the eml-module) may be reported (lines 34-39).

It is important to mention two internal data structures within SPASS.

- TERM – is a struct with the principle components, a SYMBOL, and a LIST of arguments for this symbol. This list will contain other TERMs (and is of an arbitrary length, or empty). SYMBOLS are for example fol_Exist, for_And, fol_TRUE or a predicate.
- LIST – is a singly linked list of POINTERs that can be traversed in the direction head to tail. The pointers contained in the list can be to arbitrary components, but are frequently pointers to TERMs.

Using these two data structures it is possible to define formulae in a nested fashion; a symbol and it's associated arguments is present at the topmost level, and each of these arguments is itself a TERM composed of a symbol with its own arguments. The recursive structure is terminated by an empty list at, for example, the argument to a variable.

Figure 1.12 <u>A .dfg input for SPASS and the results output by SPASS</u>.

An example SPASS session is illustrated. The problem calculated is the same as used in figure 1.12.

INPUT .dfg file	OUTPUT of SPASS
1 begin_problem(example).	1 Input Problem:
% NO line numbers in .dfg	2 1[0:Inp] || -> P(skc1)*.
	3 2[0:Inp] || -> Q(skc1)*.
2 **list_of_descriptions.**	4 3[0:Inp] || R(skc1)* -> .
3 name({**}).	5 4[0:Inp] || Q(U) P(U) -> R(U)*.
4 author({**}).	6 This is a monadic Horn problem without equality.
5 status(unknown).	7 This is a problem that has, if any, a finite domain model.
6 description({**}).	8 There are no function symbols.
7 end_of_list.	9 This is a problem that contains sort information.
	10 The conjecture is ground.
	11 The following monadic predicates have finite
8 **list_of_symbols.**	extensions:Q,P.
9	12 Axiom clauses: 1 Conjecture clauses: 3
predicates[(Q,1),(P,1),(R,1)].	13 Inferences: IEmS=1 ISoR=1 IORe=1
10 end_of_list.	14 Reductions: RFMRR=1 RBMRR=1 RObv=1 RUnC=1 RTaut=1 RSST=1
	RSSi=1
	15 RFSub=1 RBSub=1 RCon=1
11 **list_of_formulae(axioms).**	16 Extras : Input Saturation, Always Selection, No
12 formula(	Splitting, Full
13 forall([x],implies(P(x),	17 Reduction, Ratio: 5, FuncWeight: 1, VarWeight: 1
14 implies(Q(x),R(x))))	18 Precedence: div > id > R > Q > P > skc0 > skc1
15).	19 Ordering : KBO
16 end_of_list.	20 Processed Problem:
	21 Worked Off Clauses:
	22 Usable Clauses:
17	23 1[0:Inp] || -> P(skc1)*.
list_of_formulae(conjectures).	24 2[0:Inp] || -> Q(skc1)*.
18 formula(	25 3[0:Inp] || R(skc1)* -> .
18 forall([x],	26 6[0:Res:1.0,4.1] || Q(skc1) -> R(skc1)*.
20	27 7[0:Res:4.2,3.0] || P(skc1) Q(skc1)* -> .
implies(and(P(x),Q(x)),R(x)))	28 4[0:Inp] || P(U) Q(U) -> R(U)*.
21).	29 SPASS V 3.0
22 end_of_list.	30 **SPASS beiseite: Proof found.**
	31 Problem: /tmp/mspass13564.in
	32 SPASS derived 3 clauses, backtracked 0 clauses and kept 5
23 **list_of_settings(SPASS).**	clauses.
24 {*	33 SPASS allocated 575 KBytes.
25 set_flag(DocProof,1).	34 **SPASS spent 0:00:00.03 on the problem.**
26 set_flag(PProblem,1).	35 0:00:00.00 for the input.
27 set_flag(PGiven,1).	36 0:00:00.00 for the FLOTTER CNF translation.
28 set_flag(Sorts,0).	37 0:00:00.00 for inferences.
29 set_flag(Splits,0).	38 0:00:00.00 for the backtracking.
30 set_flag(CNFRenaming,0).	39 0:00:00.00 for the reduction.
31 set_flag(CNFOptSkolem,0).	40 Here is a proof with depth 1, length 6 :
32 set_flag(CNFStrSkolem,0).	41 1[0:Inp] || -> P(skc1)*.
33 *}	42 2[0:Inp] || -> Q(skc1)*.
34 end_of_list.	43 3[0:Inp] || R(skc1)* -> .
	44 4[0:Inp] || P(U) Q(U) -> R(U)*.
35 end_problem.	45 5[0:Res:2.0,4.0] || P(skc1) -> R(skc1)*.
	46 8[0:MRR:5.0,5.1,1.0,3.0] || -> .
	47 Formulae used in the proof : conjecture0 axiom0