MEng Individual Project Report

A KE Based Theorem Proving Assistant

Author: Ulrich Endriß
Supervisor: Jeremy Pitt

Imperial College, Department of Computing

June 20, 1996

Abstract

The project documented in this report mainly deals with the development of a pedagogic tool for teaching logic and reasoning. This program uses the KE Calculus, a relatively new method of automated deduction, which arguably has advantages, from a didactic point of view, over both the tableau method and natural deduction. There already existed a prototype program of the desired type, MacKE, which allows its user to build up a graphic KE proof tree, whose correctness can be checked automatically.

The first step undertaken in this project was an evaluation of MacKE. On this evaluation a redesign was to be based. Unlike MacKE, which runs on a Macintosh platform, the new program – WinKE – has been designed for a PC running under Windows 95 because of the relative availability of that platform.

This report presents the KE Calculus, comments on the evaluation of MacKE, documents the design of WinKE, and makes some remarks on its implementation. Version 1.2 of WinKE described in this report exists as a working stand-alone system.

Contents

Chapter 1

Introduction

1.1 The Project

There exists a program called MacKE that is to be used as a pedagogic tool to assist the teaching of logic and reasoning. It was intended to support a textbook by Marcello D'Agostino and Marco Mondadori, which is to be released in the near future. MacKE is implemented in Prolog for a Macintosh platform and uses the KE Calculus. The actual version (v0.34, 1994 by Jeremy Pitt) enables the user to perform a theorem proof in first order predicate logic by building up the respective proof tree via the graphical interface of the program according to the rules of KE. Two different modes are available, in the Pedagogue mode the user is forced to apply the rules correctly, in Supervisor mode generally any input is accepted, i.e. the program only serves as a blackboard to design the tree and the user him/herself is responsible for its correctness. The routines to check such a proof done in Supervisor mode are implemented, but haven't already been made available by the MacKE interface.

The projects aim is to create a new and improved version of such a KE based theorem proving assistant, this time to run on a PC platform under Windows 95. The port to PC is due to the relative availability of that platform. The first milestone to achieve is an evaluation of MacKE to spot advantages and disadvantages of the current version, regarding functionality as well as usability. This evaluation should then lead to a complete redesign of the system and finally to the implementation of WinKE.

In particular the new program should be equipped with a real Assistant option, that means apart from its teaching facilities the system should also offer the possibility to support users already experienced in Theorem Proving for example by being able to perform parts of the proofs automatically or allowing time efficient inputs via the mouse.

1.2 Overview

This report starts with an introduction to the KE Calculus in chapter 2. The rule systems for propositional as well as for first order logic are given and some aspects of the design of a KE calculus for modal logics are presented. This chapter closes with a short comparison with other techniques in the field of automated deduction and discusses some advantages of the system KE.

The next chapter deals with the program MacKE. After a description of the system from a user's point of view in the evaluation part several problems with the current design are addressed. As this evaluation is intended to produce guidelines for the design of WinKE chapter 3 finally lists some recommendations for the redesign.

In chapter 4 the design of WinKE is developed. First some general requirements are addressed. This is followed by a discussion of some important design decisions. The final specification of the program is then presented in the following chapter.

How the specification has been realised as a program is described in the chapter on implementation. To make the working of WinKE more transparent to the reader in that chapter also some screen dumps of the running program are shown.

This report ends with a short evaluation of the results gained and possible areas for further work in the field are pointed out.

1.3 Guide to Literature

At the end of this document a list of references is given, mainly covering the theoretical aspects of the project, i.e. theoretical presentations and discussions of the KE Calculus. Items [1] and [3] introduce the KE Calculus and also discuss its differences and in particular its advantages to other more classical methods of theorem proving like for example tableaux. In [7] besides a short introduction to KE the design of leanKE, a theorem prover that uses KE, is presented and its performance is evaluated. An introduction to modal logic is given by [6] and [4] discusses KE in conjunction with multi-modal logics.

The program MacKE is described in [9], a shorter description of its interface can be found in [8]. Also references to some other approaches towards building a teaching tool for theorem proving are given. Publications on the programs *Hyperproof* [2], *MacLogic* [5], and *Tableau II* [10] are listed.

Chapter 2

The KE Calculus

2.1 Introduction to KE

In his PhD thesis [1] Marcello D'Agostino presents the KE Calculus, a relatively new method for proving theorems automatically, a problem which classically has been tackled by either resolution, natural deduction, or semantic tableaux.

The typical problem the KE Calculus may be applied to is to show that when a set of formulas, the so called premises are known to be true, another formula also has to hold. The last formula is called the conclusion, because one believes, that it might be concluded from the premises. A solution to the given problem would be to prove that the complement of the conclusion is a contradiction to the premises. This could be done by deducing further formulas from the premises and the complement of the conclusion respectively adding theorems until two formulas are found, that obviously contradict each other (in the simplest case one of them would be the complement of the other).

Because the KE Calculus uses this idea it is called a refutation system. The first step to perform the proof of the conclusion is to write all the premises and the negation of the conclusion together one underneath the other as the trunk of a so called proof tree, which then has to be build up following the rules of the KE Calculus. Those rules, which will be described below, extend the proof tree by further formulas and may also branch the (at the beginning linear) tree. When on a branch of the proof tree one can identify two complement formulas, this branch is said to be closed and no further rules will be applied to it. If it is possible to close all branches, the proof is done, i.e. the conclusion is proven to hold. The reason why this is so lies in the special nature of the KE rules. They only add new formulas to the tree, that can either be deduced from the formulas given so far, or in one case (the rule *PB*, see below) split a branch and add to one of the new branches the complement of the formula that is added to the other one. As every single branch has to be closed, this splitting is equivalent to the proving of the refutation for the two possible cases that an arbitrary formula may be true or false.

2.2 Propositional KE

2.2.1 Formulas

Before the KE rules are presented a system of classification for formulas of propositional logic has to be introduced. A formula is called an α formula or said to be of *conjunctive type*, if it matches with one of the following patterns.

$$\alpha_1 \wedge \alpha_2 \qquad \neg(\alpha_1 \vee \alpha 2) \qquad \neg(\alpha_1 \rightarrow \alpha_2)$$

These formulas are called conjunctive, because the main connector of their canonical representation is $\wedge$. In addition also formulas that are syntactically the double negation of another formula are counted towards the α formulas. They can be symbolised as

$$\neg\neg\alpha$$

The second class consists of the β formulas, which are formulas of *disjunctive type*. The possible patterns for them are

$$\beta_1 \vee \beta_2 \qquad \neg(\beta_1 \wedge \beta_2) \qquad \beta_1 \rightarrow \beta_2$$

Formulas whose main connector is $\leftrightarrow$ form the class of η formulas, they may occur as one of the following patterns.

$$\eta_1 \leftrightarrow \eta_2 \qquad \neg(\eta_1 \leftrightarrow \eta_2)$$

2.2.2 Rules

For each of the presented types of formulas the KE Calculus has got a rule that may be applied to a formulas of the respective type on the proof tree. Such a rule describes what kind of formula(s) can be added to a branch of the tree, if one or two certain other formulas are already there. For example if $\alpha_1 \wedge \alpha_2$ is already on the tree respectively in the set of formulas that are assumed to be true (hoping that a contradiction to this assumption can be found), one can add α_1 and α_2 to the same branch, because it is possible to deduce α_1 and α_2 from the given formula $\alpha_1 \wedge \alpha_2$. This is in fact the so called α rule for the first one of the patterns above. Schematically such a rule can be written as a line with the given formula(s) above and the deduced one(s) below. If the complement[1] of a formula ϕ is denoted by ϕ^c, then the α rules for the first three patterns are

$$\frac{\alpha_1 \wedge \alpha_2}{\begin{array}{c}\alpha_1\\ \alpha_2\end{array}} \qquad \frac{\neg(\alpha_1 \vee \alpha_2)}{\begin{array}{c}{\alpha_1}^c\\ {\alpha_2}^c\end{array}} \qquad \frac{\neg(\alpha_1 \rightarrow \alpha_2)}{\begin{array}{c}\alpha_1\\ {\alpha_2}^c\end{array}}$$

The rule for the elimination of $\neg\neg$ is stated as

$$\frac{\neg\neg\alpha}{\alpha}$$

The β rules are slightly more complex as they take two given formulas and add one new formula to the tree. One of the formulas, that are already on the tree, is called the major formula – this could for example be $\beta_1 \vee \beta_2$ –, the other one, which has in this case to be the complement of one of the main subformulas of the major formula, is called the minor formula. If for example it would be ${\beta_1}^c$ the new formula β_2 could be deduced and added to the respective branch, because if $\beta_1 \vee \beta_2$ is true and because of ${\beta_1}^c$ the formula β_1 has to be false, one knows that "at least" β_2 needs to hold. So the following β rules can be listed. Note that for the third pattern two rules have to be listed, because it is not symmetric (i.e. a complement and a non-complement subformula occur when the pattern is transformed into

[1]Here complement stands for that syntactic representation of the complement with the minimal number of $\neg$'s, i.e. for a formula of the form $\neg\phi$ the complement is given by ϕ (and not by $\neg\neg\phi$).

the standard disjunctive form).

$$\frac{\begin{array}{c}\beta_1 \vee \beta_2 \\ {\beta_1}^c\end{array}}{\beta_2} \qquad \frac{\begin{array}{c}\neg(\beta_1 \wedge \beta_2) \\ \beta_1\end{array}}{{\beta_2}^c} \qquad \frac{\begin{array}{c}\beta_1 \to \beta_2 \\ \beta_1\end{array}}{\beta_2} \qquad \frac{\begin{array}{c}\beta_1 \to \beta_2 \\ {\beta_2}^c\end{array}}{{\beta_1}^c}$$

For the η formulas four rules can be derived. Their application is similar to the application of a β rule. Also a major and a fitting minor formula already have to be on the particular branch. As η formulas state equalities respectively non-equalities between two subformulas, if one of those subformulas is on the tree together with the η formula, then also the other one can be written on the same branch respectively for the case of non-equality the complement of the minor formula can be added. If the minor formulas are complements of the subformulas, the added formulas are also complements of subformulas. The rules are given as follows.

$$\frac{\begin{array}{c}\eta_1 \leftrightarrow \eta_2 \\ \eta_1\end{array}}{\eta_2} \qquad \frac{\begin{array}{c}\eta_1 \leftrightarrow \eta_2 \\ {\eta_1}^c\end{array}}{{\eta_2}^c} \qquad \frac{\begin{array}{c}\neg(\eta_1 \leftrightarrow \eta_2) \\ \eta_1\end{array}}{{\eta_2}^c} \qquad \frac{\begin{array}{c}\neg(\eta_1 \leftrightarrow \eta_2) \\ {\eta_1}^c\end{array}}{\eta_2}$$

The last rule to be described here is the so called *PB* rule, which is short for *Principle of Bivalence*. It is the only branching rule of the KE Calculus and takes no premises. It is always true that a given formula is either true or false. Looking at a proof tree this means, that any of its existing branches may be split into two new branches, if each of them is expanded by a new formula and those two new formulas are complementary. So the *PB* rule has the form

$$\frac{}{\phi \mid \phi^c}$$

Together with the strategy described earlier in 2.1 these rules form the KE Calculus for propositional logic.

2.2.3 Analytic PB and β Simplification

The algorithm described so far doesn't provide decidability despite the fact that propositional logic is in fact decidable. The reason for this deficiency is that the *PB* rule as presented above is not analytic. That means any arbitrary formula could be used to split an open branch. Perceiving that this isn't a very useful strategy is very straightforward. As the aim of the application of KE rules is to close a branch of the proof tree, and therefore to unificate a formula with the complement of another, it wouldn't make sense to introduce new terms during the application of the *PB* rule. So promising candidate formulas for the application of KE can only be those which already occur as a subformula on the particular branch. Rule applications that regard these considerations are said to satisfy the subformula property.

The application of *PB* becomes more efficient, if one doesn't choose those formulas to which an α, β or η rule already applies, as this doesn't provide any further information.

Another strategy to increase efficiency, which applies to methods of automated deduction in general, is not to use subsumed formulas during the construction of the proof. For the KE calculus this means that an application of the β rule might not always be useful. For example if the β formula $\beta_1 \vee \beta_2$ and one of its subformulas, say β_1, which subsumes the former one, are on the same branch, then the two possible minor premises $\neg\beta_1$ and $\neg\beta_2$ either close the branch directly (in the first case) or simply add the already present formula β_1 (in the second case). Such subsumed β formulas do not need to be considered for rule application. This strategy is called β simplification.

2.3 KE for First Order Logic

The KE Calculus described so far can be extended so that also sets of formulas of first order predicate logic can be handled. Only two rules to cope with quantified formulas have to be added, the refutation strategy as described above remains the same.

2.3.1 Formulas

In addition to the classes defined in the respective section in the presentation of propositional KE for first order logic classes for universally and existentially quantified formulas need to be introduced. Formulas that are *universally quantified*, like

$$\forall x : \gamma \qquad \neg\exists x : \gamma$$

are the so called γ formulas and those that are *existentially quantified* belong to the δ formulas, which can come up in one of the following forms.

$$\exists x : \delta \qquad \neg\forall x : \delta$$

2.3.2 Rules

Let $\gamma[x \leftarrow t]$ denote the formula resulting from substituting every x in γ by t. If there is a universally quantified formula $\forall x : \gamma$ on the tree, that means, if γ is true for any x, then γ also needs to hold for a specific value of x, for example t, and the formula $\gamma[x \leftarrow t]$ can be added to the branch. The rule is written as

$$\frac{\forall x : \gamma}{\gamma[x \leftarrow t]} \qquad \frac{\neg\exists x : \gamma}{\neg\gamma[x \leftarrow t]}$$

The δ rule states, that for an existentially quantified formula on a certain branch that formula with the quantified variable instantiated with a new (i.e. not on the branch) Skolem constant can be added. This is so, because if a formula δ is true for some x, one can define the mentioned new Skolem constant as that specific x and so also the formula yielded by the described substitution has to hold.

$$\frac{\exists x : \delta}{\delta[x \leftarrow sk_i]} \qquad \frac{\neg\forall x : \delta}{\neg\delta[x \leftarrow sk_i]} \qquad (sk_i \text{ new Skolem constant})$$

2.3.3 Restricting Instantiation and Fairness

Similarly to the *PB* rule also the accuracy of the application of the γ rule can be improved. Again, the aim is to close a branch by finding two complementary formulas on the same branch. Therefore it wouldn't make sense to introduce new subformulas when applying γ. To achieve this the quantified variable should not be instantiated with a new function symbol. In fact it is sufficient only to consider those terms that already occur as arguments of formulas on the same branch.

First order predicate logic is not decidable and therefore KE cannot provide a deterministic algorithm to prove arbitrary theorems. However there exist some guidelines on when and how to apply what specific rule to make KE more efficient. Firstly there are some preferences for the order of rule

α Rules (incl. $\neg\neg$-Elimination)

$$\begin{array}{c} \alpha_1 \wedge \alpha_2 \\ \hline \alpha_1 \\ \alpha_2 \end{array} \qquad \begin{array}{c} \neg(\alpha_1 \vee \alpha_2) \\ \hline {\alpha_1}^c \\ {\alpha_2}^c \end{array} \qquad \begin{array}{c} \neg(\alpha_1 \rightarrow \alpha_2) \\ \hline \alpha_1 \\ {\alpha_2}^c \end{array} \qquad \begin{array}{c} \neg\neg\alpha \\ \hline \alpha \end{array}$$

β Rules

$$\begin{array}{c} \beta_1 \vee \beta_2 \\ {\beta_1}^c \\ \hline \beta_2 \end{array} \qquad \begin{array}{c} \neg(\beta_1 \wedge \beta_2) \\ \beta_1 \\ \hline {\beta_2}^c \end{array} \qquad \begin{array}{c} \beta_1 \rightarrow \beta_2 \\ \beta_1 \\ \hline \beta_2 \end{array} \qquad \begin{array}{c} \beta_1 \rightarrow \beta_2 \\ {\beta_2}^c \\ \hline {\beta_1}^c \end{array}$$

η Rules

$$\begin{array}{c} \eta_1 \leftrightarrow \eta_2 \\ \eta_1 \\ \hline \eta_2 \end{array} \qquad \begin{array}{c} \eta_1 \leftrightarrow \eta_2 \\ {\eta_1}^c \\ \hline {\eta_2}^c \end{array} \qquad \begin{array}{c} \neg(\eta_1 \leftrightarrow \eta_2) \\ \eta_1 \\ \hline {\eta_2}^c \end{array} \qquad \begin{array}{c} \neg(\eta_1 \leftrightarrow \eta_2) \\ {\eta_1}^c \\ \hline \eta_2 \end{array}$$

γ Rules

$$\begin{array}{c} \forall x : \gamma \\ \hline \gamma[x \leftarrow t] \end{array} \qquad \begin{array}{c} \neg\exists x : \gamma \\ \hline \neg\gamma[x \leftarrow t] \end{array} \qquad (t \text{ a term already on the branch})$$

δ Rules

$$\begin{array}{c} \exists x : \delta \\ \hline \delta[x \leftarrow sk_i] \end{array} \qquad \begin{array}{c} \neg\forall x : \delta \\ \hline \neg\delta[x \leftarrow sk_i] \end{array} \qquad (sk_i \text{ new Skolem constant})$$

***PB* Rule**

$$\begin{array}{c|c} \hline \phi & \phi^c \end{array} \qquad (\phi \text{ satisfies subformula property})$$

Table 2.1: The Rules of the KE Calculus (Classical FOL)

applications. Whenever a branch can be closed this is definitely the first thing to do. Next any of the non-branching rules for propositional logic should be applied whenever possible, apart from the case of subsumption (β simplification) discussed earlier on. If this is not possible, before one starts splitting up branches, the quantifier rules should be tried.

It has to be emphasised, that the γ rule is the only one that can be applied to the same formula on the same branch several times, as each time a new instantiation for the quantified variable may be chosen. The γ application should be *fair*, that means it should only be used up to a limited amount of times on the same branch and formula before other steps are tried.

The application of *PB* should always be last choice (apart from repeated instantiations of γ formulas), and as stated above it should always be restricted to subformulas already on the branch. When using KE fairly the semi-decidable character of first order logic can be captured.

A summary of all KE rules for first order predicate logic (including those treating propositional formulas) is provided by Table 2.1.

2.4 Modal KE

In the following we give a short introduction to prefixed modal KE (for the propositional case of distinct normal modal logics) as it is presented in [4].

2.4.1 Prefixed Formulas

Formulas valid in certain worlds can be distinguished by labels or prefixes, a (possibly empty) list of integers. Instead of giving the ground representation also metavariables for either single integers or sequences of them can be part of such a prefix. A non-ground prefix normally not only denotes one single world, but a set of worlds, i.e. all those that can be obtained by instantiation of the metavariables. [4] describes an algorithm for prefix unification that determines whether two given prefixes can be unified and if so gives back the unifier.

In addition to the types of formulas listed in the section on propositional KE for the modal case we also have formulas of *necessity* and *possibility*.

The former ones are either of one of the forms

$$\Box\nu \qquad \neg\Diamond\nu$$

and the latter have one of the following two patterns.

$$\Diamond\pi \qquad \neg\Box\pi$$

2.4.2 Accessibility and Modal Rules

Crucial about models of modal logics is the accessibility relation between the worlds involved. We are only dealing with normal modal logics, i.e. the axiom **K** and the rule of necessitation hold.

$$\mathbf{K:}\ \frac{\Box(A \to B)}{\Box A \to \Box B} \qquad \mathbf{nec:}\ \frac{A}{\Box A}$$

The general accessibility relation for prefixes can be defined as follows. For every i world σi is accessible from σ, where σ is a metavariable ranging over the domain of sequences of integers and i is a metavariable over the domain of integers. Further possible properties of the accessibility relation (like symmetry, transitivity, etc.) can easily be defined for prefixes.

The α rule and *PB* for modal KE are equivalent to the classical case, simply the conclusions have to get the same prefixes. As the β and the η rule involve two premises things are slightly more complex. Such a rule is not only applicable, if both premises have exactly the same label, but more general, if their prefixes can be unified. The conclusion can be assigned either of the two labels of the premises. To close a branch also the two complementary formulas' prefixes have to unify, but in addition their unifier has to be, or has to be able to be, ground. This latter condition is necessary to assure that the worlds used for the refutation actually do exist.

For formulas of necessity and possibility additional rules are introduced. The rule for possibility is the same for any distinct normal modal logic. For any ground i provided σi is a simple unrestricted extension of σ (to assure this make i a new integer on the particular branch) the rule is stated as follows.

$$\frac{\sigma : \Diamond\pi}{\sigma i : \pi} \qquad \frac{\sigma : \neg\Box\pi}{\sigma i : \neg\pi}$$

For example if $\Diamond\pi$ holds in world σ this means that π holds in *some* world accessible from σ, but apart from that we cannot make any statement about this world. σi is a newly introduced world, which is accessible from σ.

The rule for formulas of necessity is different for each logic. As $\sigma : \Box\nu$ means that ν is true in *every* world accessible from σ a formulation of an appropriate rule has to capture all worlds accessible from σ, which involves considerations about the underlying accessibility relation, which is different for each logic. A unified presentation of the rules is the following one.

$$\frac{\sigma : \Box\nu}{\tau : \nu} \qquad \frac{\sigma : \neg\Diamond\nu}{\tau : \neg\nu} \qquad (\tau \text{ accessible from } \sigma)$$

Here it depends on the axioms valid in the particular logic whether a world τ is accessible from σ or not.

2.4.3 A Problem

The presented rule for formulas of necessitation as presented above is not entirely correct for the two normal modal logics **K4** and **K45**. This is because the accessibility relations underlying those two logics are transitive but not deontic. The transitivity schema ($\Box A \rightarrow \Box\Box A$) allows to copy boxed formulas to newly accessible worlds, which might not exist as seriality cannot be guaranteed.

In [4] an a priori solution to this problem has been presented. Unfortunately that solution doesn't fit into the general framework of the rest of the presentation, and therefore has been judged as unsatisfactory by the authors.

2.5 Discussion

At this point the KE Calculus shall be compared with two other (classical) methods in the field of automated deduction, natural deduction and the tableau calculus. As argued for example in the articles [1], [3], and [7] KE poses some advantages over both tableaux and natural deduction.

The KE Calculus is strongly related to classical tableaux. The most important difference is, that it has only one branching rule (PB). This makes a theorem prover implemented according to the KE rules more space efficient than a similar one following the classical tableaux rules as less branches have to be maintained. This particular point has been examined in [7], where leanKE, a theorem prover using KE, is compared with a similar program based on the tableau method. The possibility to keep proof trees rather small might also help humans using the calculus to overview a proof. This is particularly interesting for teaching purposes.

Also the rule application corresponds more closely to the semantics of classical logic than in the tableau calculus. For example the β rule for a formula of the form $\beta_1 \to \beta_2$ and a minor premise β_1 can directly be interpreted as an application of *modus ponens*. The *Principle of Bivalence* of the notion of truth underlying classical logic, i.e. any formula is either true or false, is also immediately transparent through the KE Calculus in form of the *PB* rule. This is not the case for tableaux.

In [3] several examples for natural deduction proofs are given. Those proofs are much more complex than the rather simple nature of the proven theorems would suggest. As pointed out in that article this is due to the mismatch between the natural deduction rules and the classical meaning of the logical operators, a drawback not shared by the KE Calculus.

2.6 Summary

In this chapter the KE Calculus for propositional and for first order logic has been presented. Also some remarks on modal KE have been made. Strategies to reduce the search space have been mentioned. Those are analytic application of *PB* and the γ rule, β simplification, fairness (with respect to the number of applications of γ rules), and preferences of rules. A comparison of KE with other methods showed, that it arguably has some advantages over the classical approaches

Chapter 3

The Program MacKE

3.1 Description of MacKE

MacKE is a program to support the teaching of logic and reasoning. It is based on the KE Calculus and enables the user (who might be an undergraduate student in computer science) to perform the logic proof for a given exercise step by step. Such an exercise consists of a set of formulas of first order predicate logic, the so called premises, which are considered to be true, and one more formula, the conclusion, which has to be proven using the KE Calculus. In fact when loaded into the MacKE system the conclusions are already negated as required for the application of KE.

3.1.1 The Interface

After having started the program the interface described in the following paragraphs is presented to the user. As there hasn't yet been implemented a stand-alone version of MacKE, apart from the items belonging to MacKE itself the standard Prolog menu is visible respectively the MacKE menu is integrated in the main Prolog menu bar. MacKE has got the following four pulldown menus (see also Figure 3.1): File, Analysis, Mode, and Special, where File consists of the options New, Open, Next, Prev, Select, Save as, Print, and Quit. Under the Analysis menu Apply Alpha, Apply Beta, Apply Eta, $\neg\neg$-Elimination, Apply Delta, Apply Gamma, Apply PB, and finally Close Branch are available. The subitems to Mode are Assistant, Pedagogue, and

Figure 3.1: The Menus of MacKE

Supervisor. Special pulls down Check proof and Help. Not all of the subitems mentioned have been fully implemented so far, so that in fact some of them are only pseudo features, i.e. they do not react when being clicked. Those items are New, Select, Save as, Print, Quit, Assistant, Check proof, and Help. Exactly speaking parts of some of those features have been integrated in the program, but as they are not accessible via the userinterface of MacKE and can only be used by entering queries to the Prolog system, they won't be discussed here.

After having started the program at first only the Open item of the File menu can be activated. When clicked it confronts the user with a standard dialogue window for opening files. A file containing a number of proof problems has to be chosen (otherwise an error message will come up). Then the system opens the graphical KE window, which might look like shown in Figure 3.2. The KE window is divided into three parts. In the large one on the right hand side the proof tree is displayed. The area shown in that window is only a part of the space that is available for displaying the tree, and so the lower left window, that displays the whole area in a smaller scale, may be used to select (with the mouse) the part that should be shown in the big window on the right. The third window is the upper left one that contains three little icons (as shown in Figure 3.2). They represent the tools *select* (that's the arrow), *undo* (the rubber), and *hinter* (the magnifying glass). Note that the *select* tool has nothing to do with the (not implemented) menu option Select mentioned above. There's always one and only one of the tools active, they can be activated by a mouseclick on

Figure 3.2: The graphical KE window.

the corresponding icon. The icon of the active tool is highlighted, and which of the tools is activated is also visible through the form of the mouse-pointer, which takes the shape of the particular icon (respectively in the case of the *undo* tool becomes a cross), whenever the mouse passes the part of the KE window displaying the proof tree.

The representation of a tree mainly consists of its formulas, which are written in the standard way. Edges are only drawn where a branch is split. Below the last formula of every branch a so called branch marker, a little circle, is displayed. Also the split branches are marked with such a circle, which is placed between the two edges indicating the splitting. According to the context in which they are used the branch markers have different appearances. The open branch marker is a simple unfilled circle, whereas the closed branch marker is a filled one. A marker denoting a split branch is grey and if an open branch is selected (what that means will be described below), its marker is a nearly filled circle.

With the features Next and Prev (obviously short for Previous) one of the exercises of the opened file can be selected. Whenever a certain problem is left, in other words whenever **Next** or **Prev** are used, the steps done so far on the old proof are forgotten by the system and when returning one has to start again from the beginning. The procedure of performing a proof is in principle the same for any chosen problem, so the rest of this section deals only with steps to be performed after a certain one has been opened.

3.1.2 Using the Graphic Tools

At the beginning the proof tree displayed in the KE window only consists of the premises and the negation of the conclusion, that one wants to prove. These formulas together form the trunk of the tree, which has to be extended by the user in order to perform the proof by applying the KE rules. To alter shape and contents of the tree the user first has to choose one of the graphic tools described above by clicking the corresponding icon. At the beginning the *select* tool is activated.

According to the chosen tool clicking on either formulas or branch markers has different effects. If the *select* tool is in use in general clicking on any item of the tree means selecting that item, whether this might be a marker, a premise or a derived formula. Selecting a formula causes the system to highlight it. A selected item can be deselected by clicking on it again with the *select* tool. Closed and split branch markers cannot be selected.

When the *undo* tool is applied to a closed branch marker, the respective branch is reopened, i.e. its marker becomes an open branch marker. Clicking on a formula or a split branch marker with the *undo* tool makes MacKE delete everything on the proof tree underneath that point. Of course premises cannot be deleted, so in the case that such a formula is selected only every other formula apart from the premises and the negated conclusion is deleted, in other words the proving process is restarted. In addition to the deleting the *undo* tool also opens the respective branch, i.e. alters the corresponding branch marker.

---

[1]If the formula has been derived using the α rule or PB, then do the same for its sibling. If the formula is a premise or a negated conclusion, then restart the proof.

[2]While using the *undo* tool no formula can be a selected one.

[3]When the *hinter* tool has been used on a branch marker before, some formulas are highlighted, which makes them look like selected formulas. For the same reason also "selected" open branch markers may be clicked on with the *hinter* tool.

Click On	*select*	*undo*	*hinter*
formula	select	delete every-thing beneath[1] & open branch	show open branches it may be used on
selected formula	deselect	n/a[2]	show open branches it may be used on[3]
open branch marker	select	none	show formulas not been used on this branch
selected open branch marker	deselect	n/a	show formulas not been used on this branch
closed branch marker	none	open branch	none
split branch marker	none	delete every-thing beneath & open branch	none

Table 3.1: Functionality of the Graphic Tools

Using the *hinter* tool can help the user to identify formulas that haven't yet been used for the proof. If a formula is clicked on with the *hinter* tool all those open branches, which this formula may be used on, will be shown. This is done by changing the corresponding branch markers to selected open branch markers. Analogously clicking on a open branch marker causes all the formulas that may be used on the respective branch to be highlighted. The functionality of the graphic tools described is summarised in Table 3.1, where for every tool and every tree object that might be chosen the corresponding actions of the system are listed. Note that whenever the tool is changed by the user, all objects are deselected.

3.1.3 Applying KE Rules to the Proof Tree

By pulling down the Mode menu (see Figure 3.1) the user can choose between the two possible modes Pedagogue and Supervisor. The general difference between the two is, that in Supervisor mode the user may type in whatever s/he feels like when entering the derived formulas during an application of one of the KE rules, whereas in Pedagogue mode only correct applications of the rules are accepted by the system. First the behaviour of the system in Pedagogue mode will be described, and then the differences to the Supervisor mode will be stated.

To apply a KE rule to the proof tree the user has to select an open branch and (depending on what rule is to be used) a specific number of formulas. Via the Analysis menu (see Figure 3.1) one of the rules Apply Alpha, Apply Beta, Apply Eta, $\neg\neg$-Elimination, Apply Delta, Apply Gamma, Apply PB, or Close Branch may be chosen. If a wrong number of formulas has been selected, an error message comes up and – of course – the rule isn't applied. Otherwise a dialogue window (like the one shown in Figure 3.3) for the particular rule is opened and the user has to enter the derived formulas. In Pedagogue mode the system will reject wrong inputs like syntactically incorrect formulas or formulas that do not satisfy the KE rules and bring up a window with the respective error message. The selected formulas on which the chosen rule is to be applied are shown in the dialogue window and the userinput is supported by cut-and-paste facilities. If the dialogue hasn't been cancelled before, as soon as the input is correct, the rule is applied to the proof tree and the new one is displayed in the graphical KE window. Then again the user has the possibility to apply another rule respectively to use any other of the menu items available.

To apply the α rule one has to click on Apply Alpha. Before that an open branch and one conjunctive formula on this branch have to be selected. In the dialogue window the two subformulas [4] of the selected formula have to be entered. The selected branch will then be extended by those two subformulas.

The β rule requires a selection of two formulas on the same selected open branch, where one of these formulas has to be of a disjunctive type and

[4] When speaking of a subformula in this context we mean a derived formula of the respective KE rule. This might not always be a subformula but the complement of a subformula.

Figure 3.3: Sample Dialogue Window

the second one a matching minor premise. After having started the Apply Beta dialogue the conclusion is expected as input and will be added to the respective branch.

With Apply Eta the η rule can be used. Therefor similar to the β case two formulas have to be selected and one new formula will be put to the tree.

To eliminate a double negation one has to select the particular formula as well as an open branch marker. Then clicking on $\neg\neg$-Elimination will add the simplified formula to the branch. In this case no further input is necessary.

Also for the δ rule no dialogue is necessary as the system determines the name of the Skolem constant to be introduced automatically. When an existentially quantified formula has been selected (as well as an open branch), activating the Apply Delta item causes MacKE to extend that branch by the new skolemized formula, after a message about the skolemization has been given to the user.

Clicking on Apply Gamma invokes an application of the γ rule on a selected universally quantified formula on a selected open branch. In the dialogue window an instance of the quantified variable has to be entered. The instantiation is done automatically.

The *PB* rule can be applied by choosing Apply PB. Apart from the open branch marker one formula has to be selected, and the new formulas entered have to be subformulas of the selected one to guarantee the subformula property.

Finally one can choose Close Branch. For a successful closure of the selected branch two formulas, of which one is the complement of the other, on that branch have to be selected. The branch marker then becomes a closed branch marker.

The description of the application of the KE rules so far only refers to Pedagogue mode. In Supervisor mode the input of formulas in the dialogue windows is not checked respectively the user is not warned if s/he enters something wrong and MacKE carries on building up the graphical proof tree, even if wrong conclusions are made. As indicated by the not activateable item Check proof under the Special menu a feature for checking – possibly wrong – proves performed in Supervisor mode was planned and has also been implemented in parts. To use this feature one has to enter the respective queries directly to the Prolog system.

When all branches are closed, the proof is done and the user may switch to the next problem. The system itself doesn't care whether a proof is finished or not, that means the user alone decides when s/he wants to stop working on a certain exercise respectively starting a new one.

As the Quit item is inactive MacKE has to be exited by ending the Prolog session via the respective option in the Prolog menu.

3.2 Evaluation

The program MacKE has been evaluated with the intention to acquire guidelines for the design of WinKE. Considering the number of omissions the prototype character of the present implementation of MacKE is obvious from its description. Still the main idea underlying its design has found to be very useful and has therefore been adopted for the development of the new system. With this main idea mentioned we mean the direct manipulation of a proof tree by the user via a graphic interface. This procedure closely corresponds to what normally is done on paper. Also the availability of different teaching modes proved to be useful. In addition of the Supervisor and the Pedagogue modes implemented in MacKE also the already planned Assistant should be realised.

In the following we list a couple of problems encountered when using MacKE. The first one clearly is, that MacKE isn't a stand-alone system. To use it first Prolog has to be started before it can be used. Once compiled it becomes part of the Prolog interface, and the only way to quit it is to exit Prolog.

In MacKE it is not possible to enter new problems via the program itself. Instead Prolog files have to be edited, which than can be loaded to the system. This is particularly uncomfortable as the user has to know the exact file format required and errors might lead to unpredictable behaviour of MacKE.

Logically the *PB* rule doesn't require any premises. Nevertheless in MacKE the user has to select a formula before applying *PB*. The system then checks whether one of the typed in formulas is a subformula of that premise. It would correspond more directly to the KE Calculus, if that input of a premise could be omitted.

MacKE has branch markers for open, closed, and split branches. A split branch marker doesn't serve any real purpose. It can be clicked on with the *undo* tool, but the effect is the same as clicking on one of its direct son formulas.

The design of the interface has found to be quite useful, but is still improveable. In particular the part where the graphic tools are displayed is unnecessarily large and therefor looks a bit unready. The fact that under the File options like Open and Save that relate to files are listed together with options like Next that relate to problems as parts of files is not satisfactory.

The usability of MacKE would increase if more help functions would be available. Some minor errors for example in the texts of dialogues have been found. When using the rule application dialogues putting in logical symbols is very difficult, as one has to know which key combination belongs to what symbol.

The algorithm that determines how a proof tree is drawn on the screen is not ideal with respect to the space requirements of the displayed graphics. We will discuss this problem in detail in the next chapter.

3.3 Recommendations for the Redesign

In general MacKE seems to be a good starting point for a redesign. Most of its features will also appear in the new program. What follows is a couple of recommendations for the redesign based on the above evaluation.

As mentioned before the File menu of MacKE combines options of two different areas. This should be divided in two different main menus. One called File as before to handle actions that really directly relate to files, another one called Problem for the options treating single problems.

The problem with the interface of MacKE could be solved by dividing it into several separate windows that can be treated independently. The usability could be improved by adding buttons for shortcuts to certain menu items. In particular for the rule applications this should prove to be useful. To make the input of logical symbols easier they should be made available via mouse click where necessary. The cut-and-paste facilities proved to be very useful and will be used in WinKE as well.

As mentioned above split branch markers can be omitted, in particular as there is no correspondence for them neither to the paper-and-pencil procedure nor to the underlying logic.

Further to the *hinter* tool, which should follow over to the new system, more ways to support the user in finding a proof should be added. For example bookkeeping facilities could be included, and as already mentioned in an Assistant mode to be set up the application of KE rules will be made more comfortable for the user.

3.4 Summary

In this chapter we described the working of MacKE in detail. It proved to be a promising approach to the task of building a pedagogic tool for teaching logic and reasoning. Nevertheless the section on evaluation pointed out some disadvantages of and omissions in the current version, which have to be considered during the redesign.

Chapter 4

Design 1: Requirements

4.1 Tasks

In this section the main requirements that should be met by the program to be designed are stated. The following sections will then deal with some of the important design decisions. A detailed specification of the final product from a user's point of view will be given in the next chapter.

The aim is to design a program that shall be used mainly by students that are introduced to logic and automated deduction by means of the KE Calculus. What is classically done on paper should be simulated on the computer. That means the user should be able to set up a problem (either by typing it in by her/himself or by loading an appropriate file) and then construct a corresponding proof tree by applying the different rules of the Calculus. In addition the program should have some teaching facilities, it should be possible to check whether a proof is correct, respectively wrong steps might be prohibited in the first place. Besides the teaching options it would be useful to be able to use the program as a proof assistant, for example parts of a proof could be done automatically, hints could be given etc..

To make the system more practical options to take back steps and book-keeping facilities to make the proofs more transparent to the user should be added. Also the program should be equipped with the standard options for loading and saving files, printing, editing commands and more.

A number of the specified tasks are met by MacKE, which makes it a practical starting point for the design.

4.2 Human Computer Interaction

4.2.1 Design of the Interface

The interface of WinKE has to be designed in such a way that all functions provided by the program are easily accessible to the user. Where possible the user should be given the possibility to make inputs using the mouse rather than the keyboard. As far as it is allowed by the special nature of WinKE the system should share common standards with other software products for Windows. This means for example that the File or the Help menus should be designed in the familiar way. The appearance and the functionality of the graphic tools can be similar to standard graphic programs. The windows associated with WinKE should be movable individually. WinKE should not interfere with other programs that might run at the same time.

For example in *Tableau II* (see [10]) the user has to determine the appearance of the proof tree by dragging the nodes to the desired position on the screen. This takes more time than necessary and it is very difficult to build up a regularly drawn tree. If a way of drawing proof trees in an ideal way considering space requirements and aesthetics can be found, it is justified to omit any user interference in the positioning of the nodes. An algorithm producing such trees will be presented below.

4.2.2 Transparency

To make it more transparent to the user what is going on a couple of bookkeeping facilities will be introduced. Every formula is given a number so that for any of them clear information how it was obtained can be provided. The user him/herself decides how much information is given directly on the proof tree, the whole range can always be displayed using a *bookkeeping* tool.

4.2.3 Control

The user should be given as much control over the his work as possible. That means he should be able to change to other problems or files at any time without losing information on what s/he is currently working on. It has to be possible to take back steps individually. This will be provided by the *delete* and the *undo* tools. The first one works like the corresponding one in MacKE (there called *undo*), the latter one in addition provides the possibility to eliminate only the selected node and all those that depend on it directly with respect to rule application (rather than to the tree structure like for *delete*).

During the construction of a proof tree numbers for the bookkeeping are automatically assigned to the nodes. As the order of rule application can be arbitrary and as it is possible to take back steps, after a while it is unlikely that the tree will still be numbered in preorder. Normally after a manipulation of the tree it should not be renumbered automatically, as it might confuse a user, when certain formulas suddenly change their numbers. On the other hand the user should be given the opportunity to force a renumbering whenever this is desired.

At any time the user should be able to change the problem without losing those parts of the proof that are still valid. Control is also given by offering the possibility to change the mode at any time.

For the checking of proofs done in Supervisor mode and for the checking of rule application in Pedagogue mode the user can determine whether the subformula property and analytic instantiation for the γ rule should be tested.

4.2.4 Assistance

For every dialogue and each of the graphic tools help has to be available to assist the user during the usage of the program. Another kind of assistance is how WinKE supports its users in finding proofs. Like in MacKE a *hint* tool will be available to highlight formulas not been used on open branches at a particular state. In Assistant mode it will no longer be necessary for the user to type in the conclusions for standard rule applications (that's everything apart from *PB* and the γ rule). For the application of the *PB* rule all subformulas on the branch will be provided by the system and for the γ rule any possible value for the instantiation will be made available as well.

4.3 Drawing the Proof Tree

4.3.1 Trees to Display KE Proofs

The state of a proof using the KE Calculus is represented by a tree, which has some certain properties. Every node (i.e. every formula) has either one or two sons (the latter if *PB* has been applied). To save space the edge between a node and its single son can be omitted. The splitting of a branch is represented by two lines from the father node to the left and to the right son. So on every level of a tree (level with respect to the y-coordinate) either a formula or two lines indicating a split are displayed. This structure can be handled best by the internal data structure, if for every split an additional node is introduced, which is made the single son of the father node, and the two original sons are connected to the newly introduced pseudo node. To offer the user the possibility to choose a certain branch of the proof tree, objects that represent the branches have to be added. This is done by asserting to every branch a so called branch marker as a son of the last node of the particular branch for example in form of a little circle.

A tree drawing algorithm specifies how the x- and the y-coordinates for displaying each of the nodes are found. Calculating the y-coordinates is straightforward. An initial value is associated with the root node and then while traversing the tree in preorder every node is asserted the value obtained by adding a fixed number (the height needed to display one level) to the y-coordinate of the father node. Note that when doing so it doesn't matter whether the specific node represents a formula, a split branch or a branch marker.

Crucial about the algorithm is how the x-coordinates are calculated. In the following first the method that has been used in MacKE is presented. As will be shown that method bears certain disadvantages to be pointed out. Next some requirements an ideally drawn tree should meet are stated and finally an algorithm that produces trees according to these "aesthetics" is given.

4.3.2 The Tree Drawing Algorithm of MacKE

First for every branch the formula requiring the most space is identified and its leaf is asserted the space of this longest formula. The x-coordinate of each leaf is then given by the sum of the spaces asserted to all those leaves preceding the particular leaf with respect to a preorder traverse of the tree (that means "which are to its left"). Next the tree is traversed in postorder and every node is centred above its son(s). That this can be done without the risk of overlapping has been assured by asserting the maximal space required by any formula on a branch to the leaves.

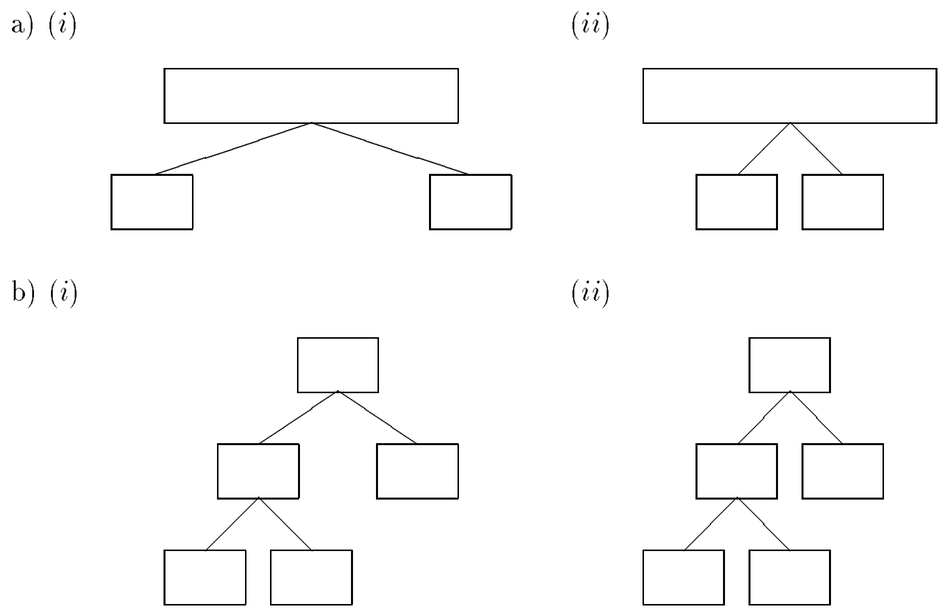

Figure 4.1: Examples of Drawn Trees

Trees whose appearance has been determined by the described algorithm have some certain disadvantages concerning space requirements as well as aesthetic considerations. To illustrate this claim two examples are given in figure 4.1. For two different logical trees (called a) and b)) in each case two different graphic representations are given. The graphics are simplified, only the size of the nodes is indicated, it doesn't matter what they actually look like. In both examples under (*i*) the tree that the algorithm used in MacKE would produce is given and (*ii*) displays the desired form, which takes less space and also looks tidier.

4.3.3 Some Aesthetics for Drawing a Tree

In [11] the authors define some requirements (or "aesthetics" as they call it) to be met by a tidily drawn tree. That discussion is followed by the presentation of an algorithm that can produce such trees. Unfortunately that article only deals with trees whose node have a fixed (very small) size and in addition nodes have only left and right nodes, also single sons are either put to the left or the right of their father node. So the results obtained in that work cannot be applied directly to the design of an appropriate algorithm to calculate trees for displaying KE proofs. But still the following listing of aesthetics for such trees partly follows the discussion in [11].

Note how KE trees have been defined above. Nodes are either formulas or splits or branch markers. All leaves (and only they) are branch markers. Splits are the only nodes that have two sons, which are formulas in any case. Formulas always have exactly one son node.

The drawing of a tree should satisfy the following aesthetics.

Aesthetic 1: Nodes at the same level should lie along a straight line, and the straight lines defining the levels should be parallel.

Aesthetic 2: A left son should be positioned to the left of its father and a right son to the right.

Aesthetic 3: A father should be centred over its son respectively its sons.

Aesthetic 4: A tree and its mirror image should produce drawings that are reflections of one another; moreover, a subtree should be drawn in the same way regardless of where it occurs in the tree.

Aesthetic 5: Any two nodes on the same or neighbouring level(s) should be placed next to another as close as possible in horizontal direction but without violating a predefined minimal distance.

The algorithm used in MacKE, which is described above, doesn't always preserve aesthetics number 4 and 5. Although it guarantees that nodes do not overlap, the distances in horizontal direction are not always minimal, as both examples in figure 4.1 illustrate. Also the subtree consisting of the two small sibling nodes is drawn differently in the two examples; this violates aesthetic 4.

4.3.4 An Aesthetic Tree Drawing Algorithm

In this section an algorithm working according to the aesthetics defined earlier on is presented. It is the algorithm that will be used for WinKE.

The y-coordinates are calculated in the standard way as pointed out in 4.3.1. Note that by doing so aesthetic 1 is guaranteed to hold.

Before the following algorithm can be applied the space requirements for the graphical representation of all the nodes need to be known. At this point values for markers and splits have to be defined. The value of a split can be set to 0, as the displaying of the two lines definitely takes less space in horizontal direction than the left and the right son below, and nodes on neighbouring levels are not allowed to overlap with respect to their x-coordinates (aesthetic 5).

The formulation of aesthetic 4 leads to the most important guideline when designing an appropriate algorithm. As subtrees should be drawn in the same way regardless where they appear, the algorithm has to start in the lowest levels of the tree, i.e. the tree will be traversed in postorder when asserting the x-coordinates. When doing so in the first traverse it won't be possible to obtain the final coordinates immediately. First only a relative coordinate for every node with respect to the position of its father has to be calculated.

For single sons this relative coordinate is clearly 0, as they are supposed to be centred below their fathers. Note that with x-coordinate (whether they might be relative or absolute) we mean the coordinate where the middle of the node's graphical representation should be placed.

For left and right sons things become more complicated. To be able to assert the correct values for each node respectively the subtree it is associated with (the subtree whose root it is) two lists representing its left and right contour have to be created and maintained. With the contour of a subtree we mean the information for each level how much space is required to the left respectively to the right from the middle of the root node. If this information is available for two sibling nodes it can be determined how far from each other they have to be put by consulting the right contour of the left node and the left contour of the right one. At this point aesthetic 5 is important. For every level it has to be checked that the nodes in that level don't come too close to each other respectively to the nodes in the levels above and below. Next the father node (which is a split) can be centred above the left and the right son.

The absolute x-coordinate of the root is set to the middle of the window where the tree is to be displayed. After that the tree is again traversed, this time in preorder. For every node the absolute value of its x-coordinate is obtained by adding the one of its father and its own relative coordinate calculated before.

Finally, before the tree can be drawn, for the nodes that are formulas the upper left corner has to be computed from the x-coordinate (that indicates where the middle should appear) and the half of length needed to display the graphical object.

4.4 The Undo Tool

WinKE will be equipped with a *delete* tool, which works like the *undo* of MacKE. Its implementation is rather straightforward as the proof tree is simply cut at the selected node. Just formulas obtained during application of either α or *PB* require additional care because of the sibling nodes, which also have to be deleted.

For WinKE in addition a different *undo* tool is planned. Unlike *delete*, that operates with respect to the tree structure, it is supposed to work on the logical structure of a proof. That means only the selected formula and all its logical offspring should be taken off the tree. For this tool as well in the case of an α or *PB* application also the siblings have to be eliminated. Difficulties might occur when a formula obtained from an application of the *PB* rule is chosen with the *undo* tool.

In that case for both the subtrees *undo* has to be applied on the two sibling nodes respectively. Then the remaining subtrees and the father branch have to be rearranged to form a single resulting branch. If both subtrees are linear this can be done without further problems by just putting them one underneath another. This is obviously not possible, if both of them are branching. In that case one of them has to be attached to the last node of the father branch and the second one to each of the open branches of the first. To minimise the complexity of the resulting tree that subbranch with fewer open branches should be the one placed above the other.

When the second subbranch is attached to each of the open branches of the first one, several copies of it have to be made. As they will be treated independently from then on, their nodes have to have different numbers. So new numbers have to be introduced. In fact the resulting tree after an application of the *undo* tool could have more nodes than the original one. To guarantee transparency in general during the manipulation of the trees numbers of certain nodes shouldn't be changed (unless of course they are newly introduced). So after a couple of manipulations of the tree it is likely not to be numbered in preorder, in the case of an *undo* application the range of numbers might even have gaps (some nodes/numbers have been deleted and other/greater ones have been newly introduced). This makes the option to renumber a tree as mentioned earlier on particularly useful in conjunction with the *undo* tool.

If one of the subbranches is completely closed the other one can be thrown away, as its information is not needed anymore. The only disadvantage is, that, if in Supervisor mode that subbranch has been closed incorrectly, after proof checking the information of the subbranch thrown away might be useful again.

4.5 Data Structure

In this section an appropriate data structure for problems respectively the formulas as part of them shall be defined. The data associated with a certain formula has to handle all information that is bound directly to it, like the formula itself, details about its derivation and any other information that should be available as part of the bookkeeping system.

In addition to the formula specific data, the problem as a whole, i.e. the connections between the single formulas has to be dealt with. This is first of all the structure of the proof tree, but it has also to be borne in mind, that it has to be possible to store a problem sequentially (i.e. in a file). To be able to use the tree printing algorithm presented in section 4.3 for every node its son(s) has/have to be stored and it has to be transparent whether the specific node has one or two sons. The latter is only the case, if the node represents a split. The dynamic Prolog clauses shown in figure 4.2 can represent the tree structure of a KE proof. The arguments of these clauses are integers representing the numbers of the respective nodes. For

```
son( Node, Son)
left_son( Node, LeftSon)
right_son( Node, RightSon)
```

Figure 4.2: Prolog Clauses for Tree Structure

nodes that represent formulas we will use positive and for splits and branch markers negative integers.

The data associated with a particular node is maintained through the dynamic clauses given in figure 4.3. Here `Formula` is a compound term whose

```
formula( Node, Formula)
derivation_rule( Node, DerivationRule)
parents( Node, Parents)
sibling( Node, Sibling)
used_on_all_branches( Node)
```

Figure 4.3: Prolog Clauses for Node Representation

functors are logical operators that need having been defined beforehand. If the particular node is not representing a formula, then the Variable `Formula` takes one of the values `split`, `open`, or `closed` respectively. In those cases also `Rule` doesn't give the KE rule but is simply set to `marker`. Otherwise `Rule` is asserted one of the atoms `double_neg`, `alpha`, `beta`, `eta`, `delta`, `gamma`, or `pb`. `Parents` is a list holding the numbers of the parent nodes, those nodes that have been used for the derivation. In the case of a derivation using *PB* `Parents` is the empty list. This is the same for splits or open branch markers. For closed ones `Parents` gives the numbers of those formulas that have been used to close the branch. `sibling( Node, Sibling)` is only defined in the case of the α or the *PB* rule. Whenever `used_on_all_branches( Node)` exists for a certain `Node` this indicates that the respective formula has already been used (as a major premise) on all open branches it is on. For γ formulas `used_on_all_branches/1` is never set as the γ rule can be applied more than once to the same branch and formula.

The third group of clauses are those handling a problem respectively a proof tree as a whole. They are listed in figure 4.4. `Preorder` is a list of all the numbers of nodes on the tree in preorder and the list `SelectedNodes` holds the numbers of those nodes that are selected (highlighted) at the current

```
preorder( Preorder)
selected_nodes( SelectedNodes)
```

Figure 4.4: Prolog Clauses for the Entire Problem

state. In addition two further clauses will have to be defined to handle the corresponding graphic objects. These will be the clause `graphic_objects( GraphicObjects)` to maintain all graphic objects to be displayed and the clause `selected_graphic_objects( SelectedGraphicObjects)` to maintain which of them are selected at a certain state.

With the data structure defined so far a problem and its associated proof tree can be handled. But the problem dealt with at a certain state will normally be just one out of a couple of problems stored in a file. The entire information of a problem and possibly parts of its proof can be stored in a single data object. For this purpose the information provided by `formula/2`, `derivation_rule/2`, `parents/2`, and `sibling/2` for each node is put together in a list of each four elements. Here the value of `Sibling` is stored in a list which might be empty, if `sibling/2` is not defined for the particular node. The information given by `used_on_all_branches/1` can be reconstructed from the rest, so it is not necessary to keep it in the list as well. Then all these lists each representing a node are put together in one big list, the nodes' order is given by `Preorder`.

As nodes for which `DerivationRule` is either `open` or `closed` have no sons, those for which it is `split` have two and the remaining ones all have exactly one son, the tree structure can be reconstructed from that preordered list. Finally the head of that list is made an integer giving the index number of that particular problem.

A couple of problems can now be stored as a list of single problems, each of them maintained as just described. At a certain state during the running of WinKE there are a number of problems loaded, which come either from a file or have been edited by the user. One of those problems is the current one, i.e. it is displayed on the screen. This information is maintained by the Prolog clauses shown in figure 4.5. the variable `Problems` is a list of the

```
problems( Problems)
current_problem_nr( CurrentProblemNr)
```

Figure 4.5: Prolog Clauses for All Problems

type just defined and `CurrentProblemNr` is an integer giving the number of that problem that is currently being displayed.

Modal Logics It is planned to extend WinKE for modal logics in the near future. The data structure defined in this section is open towards such an extension. In addition for every formula on a proof tree the prefix denoting the world it is valid in needs to be stored. Such a prefix is a list of integers and metavariables over either single integers or sequences of integers. Therefore a way to distinguish those two types of metavariables (for example when the are typed in by the user) has to be found. Also for every problem the associated logic has to be stored.

4.6 File Handling

The problem of how to save a problem respectively a number of problems to a file has already been solved by the definition of the data structure. The list `Problems` (see 4.5) can simply be written or read from a file. Whenever the user wishes to save the current state of the problems the chosen file is updated with the value of `Problems` (after `Problems` has been updated according to the actual state of the current problem).

4.7 Rule Checking

It makes sense to introduce different levels of rule respectively proof checking. For the sake of efficiency as well as for pedagogic reasons usually we want to assure that the subformula property holds for any application of the *PB* rule. Similarly a γ formula should only be instantiated in a promising way, that means with an argument already present on the branch. On the other hand logically speaking an application of *PB* that does not involve a formula that already occurs on the branch is not incorrect, it's just not very efficient. Imagine for example a readily done proof built up in Supervisor mode that uses such an unnecessary *PB* rule is checked by WinKE. If still all branches have been closed correctly the proof should be accepted and no error messages should come up.

For these reasons the user should be offered the possibility state whether s/he wants the system to check for analytic rule application or not. This decision affects the behaviour of the rule checker in Pedagogue and Assistant mode as well as the proof checking called after building up a proof tree in Supervisor mode.

The implementation of non-analytic rule checking is quite simple. It just has to be checked that a formula is of the correct type has been selected and then the user input(s) are compared with the correct conclusion(s) that can be extracted directly from the premise(s). To verify whether a particular application of the *PB* rule satisfies the subformula property all subformulas occuring on the selected branch have to be collected. To test an instantiated γ formula all terms that are arguments of formulas on the branch have to be reviewed. Checking a whole proof is done by traversing the tree in preorder and applying the rule checking procedure to every node encountered.

Modal Logics When WinKE will be extended to handle modal logics, the rule checking routines have to be changed appropriately. First of all a prefix unification algorithm like in [4] has to be implemented. Then the rule checking for the classical propositional formulas can easily be extended with a check for the correctness of the prefixes involved. For the modal rules the accessibility relation has to be considered as well.

4.8 Architecture

To define the architecture of the system it can be seen as divided in three layers. The topmost is the *interface layer* which deals with user inputs and routes them to the next layer, and which produces outputs for example in form of messages, dialogues or displayed graphics. The modules in the interface layer are the Graphic Window Manager, the Tool Manager, the Dialogue Manager, and the Menu Manager. The actual computations take place in the *internal layer*. It consists of the Internal Graphics Manager, the Tree Manager, and the KE Manager. Finally the *data layer* has got two modules, the Graphic Database and the Tree Database.

The diagram of figure 4.6 shows the architecture of WinKE schematically. The Tree Database maintains the data specified earlier on in this chapter (see 4.5). The corresponding graphic objects are stored in the Graphic Database. All data in the latter one can be constructed by applying the tree drawing algorithm to the information obtained from the Tree Database.

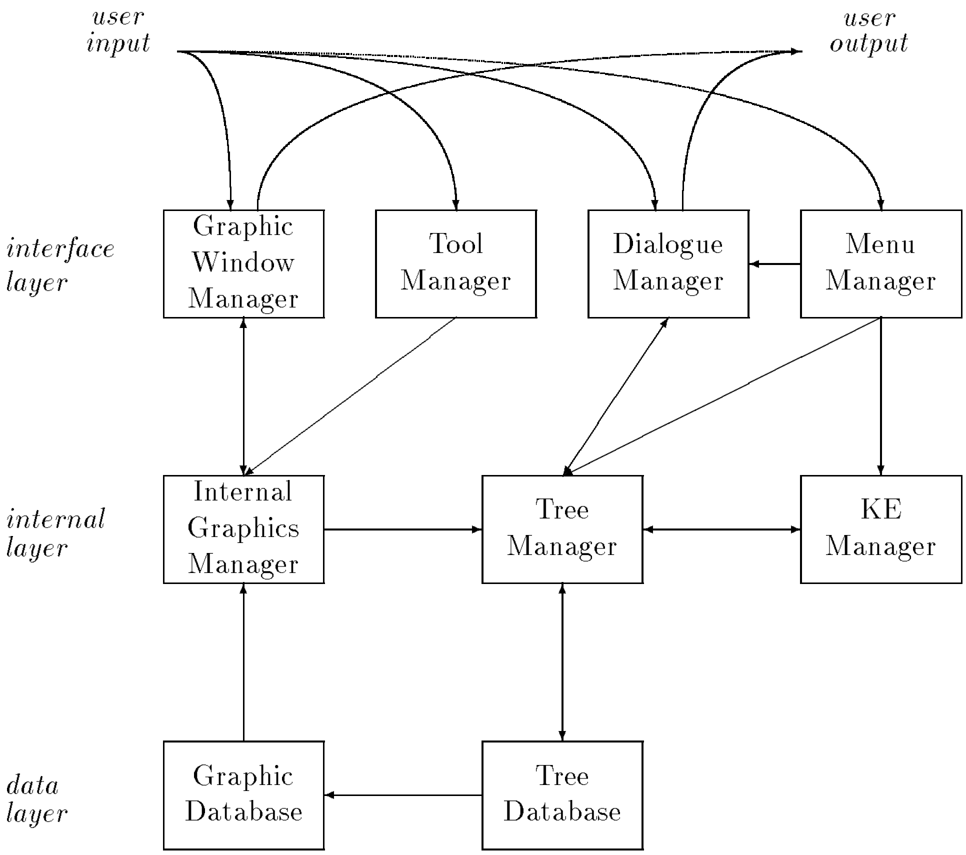

Figure 4.6: Schematic Architecture of WinKE

The Internal Graphics Manager takes the information from the Graphic Database and passes it on to the Graphic Window Manager, which finally displays it on the screen. User inputs that directly affect the graphics come either through the Graphic Window Manager (e.g. mouse clicks) or the Tool Manager to the Internal Graphics Manager. The Internal Graphics Manager informs the Tree Manager about such events. The Tree Manager, which works on the data provided by the Tree Database, is the most important module. Here all manipulations on the proof tree take place. A motivation for such a manipulation can come from either the Internal Graphics Manager, the Dialogue Manager or the Menu Manager. If the desired change is due to a rule application, then the Tree Manager has to consult the KE

Manager to find out whether that particular manipulation is approved by the rule checker provided by that module. The information the KE Manager needs for such a decision is actually provided by the Tree Database and the Dialogue Manager (which maintains the user input), but from a more systematic point of view this information can also first be collected by the Tree Manager and then be passed on to the KE Manager as a whole.

The performance of the KE Manager is determined by the Menu Manager as the teaching mode is set via the menus and also analytic rule application can be switched on and off through a menu. The Tree Manager passes messages back to the Dialogue Manager, where they are put out for the user. Also the Tree Database is updated by the Tree Manager.

4.9 Summary

In this chapter we pointed out the main requirements for the new system. The first section gave a very general statement on the tasks of WinKE. Then the main design decisions were discussed and two algorithmic problems, the tree drawing and the realisation of the new *undo* tool, have been covered.

A systematic description of the program eventually designed is given in the next chapter.

Chapter 5

Design 2: Interface and Functionality

5.1 Interface

When WinKE is started four windows come up: the menu window, two graphic windows, and the tool box. In one of the graphic windows the proof tree will be displayed and it will be used to manipulate the tree. This window shall simply be called the graphic window from now on. The other one we will refer to as the little graphic window. It displays the entire area available for the proof tree in a much smaller scale. It will be used to determine the visible part of that area to be displayed in the (other) graphic window. All four windows have minimise boxes. The graphic window is the only one that also has got a maximise box and that is resizable. Only the menu window is equipped with a system menu.

5.1.1 The Menu Window

The title of this window is "WinKE". The menu window holds all the menus of the program and a couple of buttons that provide shortcuts to some of the menu items. There are five main menus, they are called File, Problem, Analysis, Options, and Help.

The items available from the File menu are New, Open, Save, Save as, and Exit. From the Problem menu the option Next, Previous, Select, New, Edit, Reset, Renumber, Check, and Print can be chosen. Next comes the Analysis menu, which provides the different rule applications. They are Double Negation, Apply Alpha, Apply Beta, Apply Eta, Apply Delta, Apply Gamma, Apply PB, and Close Branch. The first entry under Options is Mode, which in turn has a submenu consisting of the items Pedagogue, Supervisor, and Assistant. The remaining items are Bookkeeping, Further Settings, and Save Settings. Finally the Help menus consists of the options Contents, How to use WinKE Help, and About WinKE.

The buttons beneath the menus offer direct access to some of the most frequently needed functions available via the menus. Those items are New, Open, Save, Previous, Next, all the subitems of the Analysis menu, and Help/Contents.

5.1.2 The Tool Box

The tool box window holds six buttons. The first, called the *select* button, shows an arrow. The second one displaying a cross is called *delete*. Next comes a button named *undo*, which displays the standard undo icon. The *hint* button shows a field-glass and the *bookkeeping* one an open book. Button number six finally shows a question mark for *help*. The title of the tool box window is "Tools".

5.1.3 The Graphic Window

The graphic window's title is "Problem" followed by the active file name and the problem number. The window simply consists of one graphic control together with a vertical and a horizontal scrollbar. It is resizable and can be maximised.

5.1.4 The Little Graphic Window

The little graphic window also consists of just one graphic control. Its title is "Visible Area".

5.2 Functionality

In this section the intended functionality of WinKE is described. Actions are either initiated by a menu selection, the choice of a graphic tool, or a mouse click in one of the graphic windows. So we structure the following documentation according to those different kinds of origins of actions.

5.2.1 The File Menu

The choice of Open brings up a standard dialogue for the opening of files. Files storing KE problems should have the extension `.ke`, those kinds of files in the active directory will be displayed for choice in a listbox. When a certain file has been chosen the first problem is displayed in the graphic window. The New option works as if an empty file has been opened. The system then expects the input of a problem by the user. The Save and the Save as items start up the standard dialogues for these options. The state of all problems currently maintained by the `problems/1` predicate is written to the desired file. Exit closes all four windows and WinKE is quit. Warnings or messages, for example to save a file before a new one is opened, are included in the usual way.

5.2.2 The Problem Menu

The first three options, Next, Previous, and Select, work in the obvious way: the next, previous, or a problem selected by its number is made the active one unless a non-existing problem is addressed. For example if Previous is selected from the first problem of a file an error message comes up.

New and Edit both lead to the same kind of dialogue, where a problem can be defined by typing in its premises and its conclusion. If Edit is chosen the premises already on the tree appear in a listbox and may or may not be changed or deleted, as well as new formulas may be added. If a proof has already been started, its correctness cannot be assumed anymore (as premises might have been deleted). Therefore after that it should be checked again. Within the New respectively the Edit dialogue the user is supported with cut-and-paste facilities as well as with a "virtual keyboard" for the logical operators, which are not directly available via the normal keyboard.

Reset deletes all formulas on the tree apart from the problems' premises and the negated conclusion. Renumber renumbers the proof tree in preorder. Check traverses the tree in preorder and calls the rule checker for every node encountered. Throughout the checking the user is guided by a dialogue which allows direct pruning of false formulas, jumping on to the next error, or cancelling. The Print option sends the active proof tree to the printer.

5.2.3 The Graphic Windows

The main graphic window is used to display the proof tree. Its scrollbars work in the obvious way. The little graphic window can be used for orientation during proofs requiring more space than the main graphic window provides. In the little one the whole tree is displayed in a smaller scale and a little rectangle indicates the area visible in the other graphic window. By dragging that rectangle the display in the large graphic window can be changed.

5.2.4 The Graphic Tools

A graphic tool can be chosen by clicking on the respective button of the tool box. Whenever this is done all current selections of nodes are rejected. If the *select* tool is chosen, mouse clicks on either formulas or open branch markers selects them, i.e. they are highlighted.

If the *delete* tool is active, a click on a node causes the deletion of that node and its entire subbranch. Only premises cannot be deleted, i.e. if one of them is clicked, the deleting starts at the first derived node (which has the same effect as a call of Reset). Using the *undo* tool only the clicked node and its logical offspring are deleted from the proof tree (see also discussion earlier on in chapter 4).

The *hint* tool highlights all open branches the formula being clicked has not been used on so far. If a open branch marker is chosen, those formulas not used on it are highlighted. Clicking on a node when the *bookkeeping* tool is active causes a message listing all the bookkeeping values of that particular node to come up. For formulas bookkeeping values are the node's number, the formula itself, the rule used for its derivation (unless it is a premise or the negated conclusion), the parent formulas (where available) and possibly a sibling formula. For closed branch markers the same is done. Here parent formulas are those that were used to close the branch. For open branch markers there are no bookkeeping values defined.

The last button in the graphic tool box can be used to enter the WinKE help system at the page on the usage of the graphic tools.

5.2.5 The Analysis Menu

The Analysis menu's entries are used for rule application. The chosen rule is applied to the currently selected formula(s) and the selected branch marker. Before a dialogue is entered it is checked whether a correct selection has been made, i.e. whether exactly one open branch marker and the correct number of formulas on it have been selected.

The actual rule dialogues ask the user for the input of the appropriate conclusion(s). Like for the New/Edit problem dialogue cut-and-paste and a keyboard for logical symbols are provided. In Assistant mode for the α, β, η, and the δ rule as well as for Close branch the dialogues are omitted, as after a correct selection a user input is not necessary anymore to find the right conclusion(s). For *PB* the user can use a *Next* button to get all the possible subformulas displayed one after the other. Similarly for the γ rule possible arguments for instantiation can be brought up.

After the user input the rule checker is called and either the appropriate changes are made to the tree or an error message comes up. The performance of the rule checker depends on the teaching mode. In Supervisor mode anything goes, in Pedagogue and Assistant mode only correct applications are allowed. If rule application is chosen to be analytic in the cases of *PB* and γ only such analytic rule applications are approved.

5.2.6 The Options Menu

Under Mode the desired teaching mode can be chosen. Bookkeeping brings up a dialogue where the features that should be visible on the proof tree can be selected. The features provided are the number of a formula, its derivation rule, its parents, and the information whether it has been used on all open branches (not available for γ formulas).

Further Settings enters another dialogue. There first of all analytic rule application may be switched on or off. Secondly it can be chosen whether warnings should be brought up when either of the *delete* or the *undo* tool is used before nodes are eliminated from the tree. Finally using that dialogue it can be determined by the user, if the tree should be renumbered in preorder after any change of it immediately without an explicit call of Renumber. Save Settings causes the current settings to be saved to a file, so that they can be used at the next program start.

5.2.7 Help

The help system is intended to be a program separate from WinKE. It could be set up using the standard Windows 95 Help environment. Most of the non-trivial dialogues of WinKE have *Help* buttons, which enter the help system at the appropriate page. The Contents item of the Help menu enters it at the main page. The About option under Help brings up the WinKE about message.

5.3 Summary

In this chapter we described the interface and the functionality of WinKE as it has been designed. The evaluation of MacKE and the design process reflected by chapter 4 lead to this particular design.

Next some remarks on the implementation of WinKE will be made.

Chapter 6

The Implementation of WinKE

6.1 WinProlog

The implementation has been carried out in LPA Prolog for Windows, version 3.0, a product of the Logic Programming Association, London. It proved to be particularly useful for setting up windows, menus, and dialogues. Unfortunately the support for graphics is less practical than in the corresponding version for Macintosh, in which for example MacKE has been implemented.

6.2 Modules and Global Values

The most important values are the ones mentioned in the section on data structure in chapter 4. In addition during the computation of the graphic tree some temporary global values to represent the x- and the y-coordinates of nodes and the contours of subtrees respectively are used.

The settings that can be manipulated via the Options menu are stored using the prolog clauses listed in figure 6.1. `Mode` has one of the values `Pedagogue`, `Supervisor`, or `Assistant`. The elements of the list kept by `bookkeeping_settings/1` are either 1 or 0 depending on what information should or should not appear on the graphic proof tree. `N` stands for

```
mode( Mode)
bookkeeping_settings( [N,R,P,U])
analytic/0
warnings/0
always_renumber/0
```

Figure 6.1: Prolog Clauses for Settings

numbers, `R` for derivation rule, `P` for parents, and `U` for the information stating whether the particular formula has been used on all open branches. According to the user inputs the predicates `analytic/0`, `warnings/0`, and `always_renumber/0` are either asserted or retracted. For their meaning consult chapter 5.

The modules as described in 4.8 are made transparent in the implementation through the division of the code in different files. Just the parts associated with the Dialogue Manager have been spread throughout the whole program for practical reasons.

6.3 WinKE Version 1.2

The present version WinKE v1.2 (June 1996) follows the specification given in the preceding chapter apart from a few omissions due to a lack of time. The only menu option that hasn't been implemented at all is Print. Apart from that the help facilities are not as detailed as they should. Also no little graphic window has been included so far. But as the work with MacKE showed, such a facility is only useful for rather large problems (which arise quite seldom) anyway. As under Windows there seem to be no fonts available that combine logic symbols with the standard alphanumeric ones so far rather unaesthetic operators needed to be defined (like for example **`v`** for $\vee$, **`<=>`** for $\leftrightarrow$, or **`@`** instead of $\forall$).

Apart from those deficiencies it has been managed to set up a working application under Windows 95 that can be used for the purposes lined out in the introduction.

6.4 Demonstration

In chapter 5 the functionality of WinKE is described in detail. To further illustrate its working, in particular of the actual implementation of version v1.2, some screen dumps are shown on the following pages.

Figure 6.2: The Menu Window of WinKE

Figure 6.3: The Tool Box Window

Figure 6.4: The Graphic Window of WinKE

Figure 6.5: Using the Analysis Menu

Figure 6.6: Sample Rule Dialogue

Figure 6.7: Rule Dialogue in Assistant Mode

Figure 6.8: Using the Bookkeeping Tool

Chapter 7

Conclusion

7.1 Results

We believe that with the specification of WinKE as given in chapter 5 the main aim of the project, the design of a practical teaching tool for automated theorem proving using KE has been achieved. To defend this claim a reevaluation involving users of different backgrounds should take place. A working implementation of WinKE has been set up.

7.2 Further Work

First of all the deficiencies of the present implementation as pointed out in section 6.3 should be eliminated from the program. An evaluation like the one applied to MacKE should follow to spot further possible problems.

At this stage the main idea of improvement is to extend the KE Manager in such a way, that at any state the "ideal" rule to be applied next can be found. For how this is done in theory (as far as it is possible at all) consult the section on fairness in chapter 2. Then an option Suggestion could be added to the Analysis menu, which (possibly only in Assistant mode) suggests the next rule application when chosen. Also a new teaching mode (it might be called the Trainer mode) could be added, which works similarly to the Pedagogue mode, but only accepts a rule application, if it is considered to be "ideal" at the particular state of the proof.

The next task would be to integrate modal logics into WinKE. To be able to do this in a satisfactory way for arbitrary distinct normal modal logics, it would be very useful to find a general specification of Modal KE, without exceptions as those that still have to be made for **K4** and **K45**. In general the present specification of WinKE is open towards expanding the data structure to handle labels, i.e. the program is open towards an extension to prefixed propositional modal logics.

7.3 Acknowledgements

First of all I wish to thank my project supervisor Jeremy Pitt for his invaluable help during the last eight months.

Also I'm grateful to my lecturers Krysia Broda, Jim Cunningham, and Peter Schmitt for introducing me to the broad fields of Automated Deduction and Modal Logic respectively. Krysia Broda also made WinProlog available to me.

Without the Erasmus Programme and the people involved in it at Imperial College and at the University of Karlsruhe I wouldn't have had the chance to work on this project nor to study at Imperial College.

Bibliography

[1] Marcello D'Agostino. *Investigations into the Complexity of some Propositional Calculi.* PhD thesis, Oxford University Computing Laboratory Programming Research Group, Technical monograph PRG-88, 1990.

[2] J. Barwise and J. Etchemendy. *Hyperproof.* CSLI Publications, 1994

[3] Krysia Broda, Marcello D'Agostino and Marco Mondadori. *A Solution to a Problem of Popper.* In The Epistemology of Karl Popper, Kluwer, to appear

[4] Jim Cunningham and Jeremy Pitt. *Towards Model Building in Multi-Modal Logics using KE and Constraint Satisfaction.* Esprit Medlar Project, 1995.

[5] Roy Dyckhoff. *MacLogic: A Proof Assistant for First Order Logic on the Macintosh.* Computational Science Division, University of St. Andrews, 1989

[6] Melvin Fitting. *Basic Modal Logic.* In D. Gabbay, C. Hogger, and J. Robinson, editors, Handbook of Logic in Artificial Intelligence and Logic Programming, Vol. 1, OUP, 1993.

[7] Jeremy Pitt and Jim Cunningham. *Theorem Proving and Model Building with the Calculus KE.* In the Bulletin of the IGPL, 1995.

[8] Jeremy Pitt. *User Interface Design for an Automated Pedagogic Tool.*

[9] Jeremy Pitt. *MacKE: Yet Another Proof Assistant & Automated Pedagogic Tool.* In P. Baumgärtner, R. Hähnle, J. Possegga, editors, Theorem Proving with Analytic Tableaux and Related Methods, Springer-Verlag, 1995

[10] M. Potter and D. Watt. *Tableau II: A Logic Teaching Program.* Oxford University Computing Services, Learning and Resources Centre, 1988

[11] Edward Reingold and John Tilford. *Tidier Drawings of Trees.* in IEE transactions on Software Engineering, Vol. SE-7, No. 2, March 1981