ATPCL.mth: Automated Theorem Provers for Propositional Classical Logic with DERIVE

Aguilera Venegas, Gabriel gabri@ctima.uma.es
Galán García, José Luis jl_galan@uma.es
Gálvez Galiano, Antonio antonio@academiasanmillan.es
Rodríguez Cielos, Pedro prodriguez@uma.es

Department of Applied Mathematic
University of Málaga (Spain)

Abstract

In this paper the file *ATPCL.mth* is presented. This file has been developed for using DERIVE as an automated theorem prover for Propositional Classical Logic by means of different algorithms such as *Quine*, *Semantic Tableaux* and *Short Normal Form + Resolution*.

The use of this utility file is specially useful for teachers and students in subjects on Computational Logic in Computer Science degree. In particular, the file can be used as a didactical tool for helping in the teaching and learning of automated logic deduction process.

In order to develop these algorithms, another utility file has been created to deal with tree structure using the new features of programming in DERIVE. In this paper this utility file *(tree.mth)* will be also described.

The use of these utility files will be presented by means of some examples about validity, satisfiability and deduction using both utility files. Finally, conclusions and future work will be shown.

1 Introduction

The main aim that have guided the develop of the packages *tree.mth* and *ATPCL.mth* has been to obtain a practical tool for using automated theorem provers in Propositional Classical Logic. The initial idea was motivated by the necessity of introducing

some practical classes in the subject of *Computational Logic*. This subject belongs to the Computer Science Degree in the University of Malaga. Practical classes were necessary for helping students to work with automated theorem provers but no much time was available for these practical classes. DERIVE is a powerful tool that can be used in this subject and in others of the same degree, specially subjects involving mathematics. The previous time necessary for explaining basic characteristic of DERIVE is very short so the student can take practical classes spending a very little quantity of time.

Of course the methods included in the packages can be used not only for pedagogical reasons but for solving "real" reasoning problems. Moreover the commands included in *tree.mth* can be used in a very wide range of applications.

The methods included in the package *ATPCL.mth* are refutation systems, that is, for proving the formula A, the formula $\neg A$ is introduced to the system. If $\neg A$ is a contradiction (unsatisfiable) then A is valid and in other case A is non valid. Analogously, for proving the reasoning $\{H_1, H_2, \ldots, H_n\} \models C$ the formula $B = H_1 \wedge H_2 \wedge \ldots H_n \wedge \neg C$ is the input of the method and if the result is that B is satisfiable then the reasoning is not valid and in other case the reasoning is valid. Therefore, the methods in the package are methods for testing the satisfiability of a formula.

In section 2 a brief description of the methods for automated theorem proving *Quine*, *Semantic Tableaux* and *Short Normal Form + Resolution* are included for helping in the description of the package *ATPCL.mth*.

In section 3 the utility file *tree.mth* is presented, including the syntax of the commands and examples of using them.

In section 4 the package *ATPCL.mth* is presented, including the syntax of the commands and examples of using them.

Finally some conclussions and future work are shown in section 5.

2 Some methods for automated theorem proving in Propositional Classical Logic

In this section, some of the most well known automated theorem provers for Propositional Classical Logic will be briefly described. A more detailed description can be found in [4, 2, 3].

2.1 Quine

This method is a variant of the method of the truth tables but improving it using partial interpretations due to Quine. It can be seen in [2]. Let be A a propositional formula and let $P = \{p_1, p_2, \ldots, p_n\}$ the set of propositional symbols in A. Let be J

a set of integers such that if $j \in J$ then $j \in \{1, 2, \ldots, n\}$ or $-j \in \{1, 2, \ldots, n\}$. Let be I_J the interpretation that assigns $I_J(p_k) = 1$ if $k \in J$ and $I_J(p_k) = 0$ if $-k \in J$. $I_J(A)$ can be 0, 1 or unknown (?). The method begins with $Quine(A, \emptyset)$ and it is described below.

$Quine(f, J, i := 1, l, r, J+, J-, lq, rq) :=$
$Prog$
If $i > n$
$RETURN$ "$error$"
$J+ := J \cup \{i\}$
$J- := J \cup \{-i\}$
$l := I_{J+}(f)$
If $l = 1$
$RETURN\ J+$
$r := I_{J+}(f)$
If $r = 1$
$RETURN\ J-$
If $l =$ "?"
$Prog$
$lq := Quine(A, J+, i + 1)$
If $lq \neq$ "$UNSAT$"
$RETURN\ lq$
If $r =$ "?"
$Prog$
$rq := Quine(A, J-, i + 1)$
If $rq \neq$ "$UNSAT$"
$RETURN\ rq$
$RETURN$"$UNSAT$"

Since if $\mid J \mid = n$ then $I_J(A)$ can not be "?", the method ends.

2.2 Semantic Tableaux

Semantic Tableaux is a refutation method so initially a set of propositional formulae $S = \{A_1, A_2, \ldots, A_n\}$ is given. The algorithm has to return *S SATISFIABLE* or *S UNSATISFIABLE*. For the description of the algorithm, uniform notation due to Smullyan [6] will be introduced. In this notation formulae are classified in α-formulae and β-formulae. Each α-formula is equivalent to the conjunction of two formulae α_1 and α_2. Analogously each β-formula is equivalent to the disjunction of

two formulae β_1 and β_2. Next table shows the different possibilities:

α	α_1	α_2	β	β_1	β_2
$A \wedge B$	A	B	$A \vee B$	A	B
$\neg(A \vee B)$	$\neg A$	$\neg B$	$\neg(A \wedge B)$	$\neg A$	$\neg B$
$\neg(A \rightarrow B)$	A	$\neg B$	$A \rightarrow B$	$\neg A$	B
$\neg\neg A$	A	A	$A \leftrightarrow B$	$A \wedge B$	$\neg A \wedge \neg B$
			$\neg(A \leftrightarrow B)$	$\neg A \wedge B$	$A \wedge \neg B$

Another important concept for the description of the algorithm is tableau or Jeffrey's tree [5]. A Jeffrey's tree is a tree which nodes are labelled with propositional formulae. A branch of a Jeffrey's tree is called *closed* if A and $\neg A$ occur in the branch for some formula A.

The initial step in the algorithm is to consider the one branch tree associated with

$S = \{A_1, A_2, \ldots, A_n\}$, that is:

A_1

A_2

$\vdots$

A_n

Let α be a not marked α-formula *to apply an α-rule to α* means to mark the formula α and to add the tree T to every leaf in the tree, not closed and descendant of α. Where $T = \alpha_1$ if $\alpha_1 = \alpha_2$ or $T = \begin{array}{c} \alpha_1 \\ \alpha_2 \end{array}$ in other case.

Analogously, let β be a not marked β-formula *to apply an β-rule to β* means to mark the formula β and to add the tree T to every leaf in the tree, not closed and descendant of α. Where $T = \beta_1$ if $\beta_1 = \beta_2$ or $T = \begin{array}{c} \beta_1 \\ \wedge \\ \beta_2 \end{array}$ in other case. A branch of a Jeffrey's tree is called *open* if is not closed and all the formulae except literals in the branch are marked.

The algorithm consists of the following steps:

1. Take the one branch of the Jeffrey's tree corresponding to $S = \{A_1, A_2, \ldots, A_n\}$.

2. If a branch of the Jeffrey's tree is open then S is unsatisfiable, literals in the branch correspond to a model for S and END.

3. If all the branches of the Jeffrey's tree are closed then S is unsatisfiable and END.

4. If a formula α without mark exists, apply an α-rule to the first in deep formula α, else got to step 6.

5. Review open and closed branches and go to step 2.

6. If a formula β without mark exists, apply a β-rule to the first in deep formula β.

7. Review open and closed branches and go to step 2.

2.3 Short Normal Forms + Resolution

2.3.1 Short Normal Forms

Translating a formula to clausal form by renaming is a method due to Boy de la Tour [3]. Form a propositional formula A, a set of clauses Ω (set of sets of literals) equisatisfiable with A is obtained. Let $P = \{p_1, p_2, \ldots, p_n\}$ the set of propositional symbols in A. Ω is obtained recursively as follows:

1. $contSNF := n + 1$

2. $\Omega := \{\{contSNF\}\} \cup SNF(A, contSNF)$

where operator SNF is defined recursively as follows:

- $SNF(p_i, cont) := i$

- $SNF(\neg B, cont) :=$
  $Prog(contSNF := contSNF{+}1, rd\_ := contSNF, rt\_ := SNF(B, contSNF),$
  $If(INTEGER?(rt\_), Prog(contSNF := contSNF{-}1, RETURN(-rt\_)),$
  $RETURN(\{\{cont, rd\_\}, \{-cont, -rd\_\}\} \cup rt\_)))$

- $SNF(B \wedge C, cont) := Prog(contSNF := contSNF + 1,$
  $ld\_ := contSNF, lt\_ := SNF(B, contSNF),$
  $If(INTEGER?(lt\_), Prog(rd\_ := contSNF, rt\_ := SNF(C, contSNF)),$
  $Prog(contSNF := contSNF{+}1, rd\_ := contSNF, rt\_ := SNF(C, contSNF)))$
  $If(INTEGER?(rt\_), contSNF := contSNF - 1),$
  $If(\neg INTEGER?(lt\_) \wedge \neg INTEGER?(rt\_),$
  $RETURN((lt\_ \cup rt\_) \cup \{\{-cont, ld\_\}, \{-cont, rd\_\}, \{cont, -ld\_, -rd\_\}\})),$
  $If(INTEGER?(lt\_) \wedge \neg INTEGER?(rt\_),$
  $RETURN(rt\_ \cup \{\{-cont, lt\_\}, \{-cont, rd\_\}, \{cont, -lt\_, -rd\_\}\})),$
  $If(\neg INTEGER?(lt\_) \wedge INTEGER?(rt\_),$
  $RETURN(lt\_ \cup \{\{-cont, ld\_\}, \{-cont, rt\_\}, \{cont, -ld, -rt\_\}\})),$
  $If(INTEGER?(lt\_) \wedge INTEGER?(rt\_),$
  $RETURN(\{\{-cont, lt\_\}, \{-cont, rt\_\}, \{cont, -lt\_, -rt\_\}\})))$

- $SNF(B \vee C, cont) := Prog(contSNF := contSNF + 1,$
  $ld\_ := contSNF, lt\_ := SNF(B, contSNF),$
  $If(INTEGER?(lt\_), Prog(rd\_ := contSNF, rt\_ := SNF(C, contSNF)),$
  $Prog(contSNF := contSNF{+}1, rd\_ := contSNF, rt\_ := SNF(C, contSNF)))$
  $If(INTEGER?(rt\_), contSNF := contSNF - 1),$
  $If(\neg INTEGER?(lt\_) \wedge \neg INTEGER?(rt\_),$
  $RETURN((lt\_ \cup rt\_) \cup \{\{cont, -ld\_\}, \{cont, -rd\_\}, \{-cont, ld\_, rd\_\}\})),$
  $If(INTEGER?(lt\_) \wedge \neg INTEGER?(rt\_),$
  $RETURN(rt\_ \cup \{\{cont, -lt\_\}, \{cont, -rd\_\}, \{-cont, lt\_, rd\_\}\})),$
  $If(\neg INTEGER?(lt\_) \wedge INTEGER?(rt\_),$
  $RETURN(lt\_ \cup \{\{cont, -ld\_\}, \{cont, -rt\_\}, \{-cont, ld\_, rt\_\}\})),$
  $If(INTEGER?(lt\_) \wedge INTEGER?(rt\_),$
  $RETURN(\{\{cont, -lt\_\}, \{cont, -rt\_\}, \{-cont, lt\_, rt\_\}\})))$

- $SNF(B \rightarrow C, cont) := Prog(contSNF := contSNF + 1,$
  $ld\_ := contSNF, lt\_ := SNF(B, contSNF),$
  $If(INTEGER?(lt\_), Prog(rd\_ := contSNF, rt\_ := SNF(C, contSNF)),$
  $Prog(contSNF := contSNF{+}1, rd\_ := contSNF, rt\_ := SNF(C, contSNF)))$
  $If(INTEGER?(rt\_), contSNF := contSNF - 1),$
  $If(\neg INTEGER?(lt\_) \wedge \neg INTEGER?(rt\_),$
  $RETURN((lt\_ \cup rt\_) \cup \{\{cont, ld\_\}, \{cont, -rd\_\}, \{-cont, -ld\_, -rd\_\}\})),$
  $If(INTEGER?(lt\_) \wedge \neg INTEGER?(rt\_),$
  $RETURN(rt\_ \cup \{\{cont, lt\_\}, \{cont, -rd\_\}, \{-cont, -lt\_, -rd\_\}\})),$
  $If(\neg INTEGER?(lt\_) \wedge INTEGER?(rt\_),$
  $RETURN(lt\_ \cup \{\{cont, ld\_\}, \{cont, -rt\_\}, \{-cont, -ld_, - rt\_\}\})),$
  $If(INTEGER?(lt\_) \wedge INTEGER?(rt\_),$
  $RETURN(\{\{cont, lt\_\}, \{cont, -rt\_\}, \{-cont, -lt\_, -rt\_\}\})))$

- $SNF(B \leftrightarrow C, cont) := Prog(contSNF := contSNF + 1,$
  $ld\_ := contSNF, lt\_ := SNF(B, contSNF),$
  $If(INTEGER?(lt\_), Prog(rd\_ := contSNF, rt\_ := SNF(C, contSNF)),$
  $Prog(contSNF := contSNF{+}1, rd\_ := contSNF, rt\_ := SNF(C, contSNF)))$
  $If(INTEGER?(rt\_), contSNF := contSNF - 1),$
  $If(\neg INTEGER?(lt\_) \wedge \neg INTEGER?(rt\_),$
  $RETURN((lt\_ \cup rt\_) \cup$
  $\{\{-cont, -ld\_, rd\_\}, \{-cont, ld\_, -rd\_\}, \{cont, -ld\_, -rd\_\}, \{cont, ld\_, rd\_\}\})),$
  $If(INTEGER?(lt\_) \wedge \neg INTEGER?(rt\_),$
  $RETURN(rt\_ \cup$
  $\{\{-cont, -lt\_, rd\_\}, \{-cont, lt\_, rd\_\}, \{cont, -lt\_, -rd\_\}, \{cont, lt\_, rd\_\}\})),$
  $If(\neg INTEGER?(lt\_) \wedge INTEGER?(rt\_),$
  $RETURN(lt\_ \cup$
  $\{\{-cont, -ld\_, rt\_\}, \{-cont, ld\_, rt\_\}, \{cont, -ld_, -rt\_\}, \{cont, ld_, rt\_\}\})),$

$$If(INTEGER?(lt\_) \wedge INTEGER?(rt\_),$$
$$RETURN(\{\{-cont, lt\_, rt\_\}, \{-cont, lt\_, rt\_\}, \{cont, -lt\_, -rt\_\}, \{cont, lt\_, rt\_\}\}))$$

2.3.2 Ground Resolution

Resolution is based on the following inference: $\{A \vee B, \neg A \vee C\} \models B \vee C$, where A, B, C are formulae. In the particular case in which l is a literal (implemented as an integer) and C_1 and C_2 are clauses (implemented as sets of integers) and $l \in C_1$ and $-l \in C_2$ the following inference is obtained: $\{C_1, C_2\} \models R_l(C_1, C_2)$ where $R_l(C_1, C_2) = (C_1 \backslash \{l\}) \cup (C_2 \backslash \{-l\}$. $R_l(C_1, C_2)$ is called *resolvent* of C_1 and C_2 with regard to literal l. In the particular case that $C_1 = \{l\}$ and $C_2 = \{-l\}$, $R_l(C_1, C_2) = \Box$, that is the empty clause.

Let Ω be a set of clauses (implemented as a set of sets of integers). Ω is unsatisfiable if and only if $\Box$ can be obtained form Ω adding resolvents of clauses in Ω to Ω.

Ground resolution is a basic (and inefficient) method described below.

1. $\Omega_0 = \Omega$
2. $i := 1$
3. $\Omega_i := \Omega_{i-1} \cup \{R_l(C_1, C_2) \mid l \in C_1; -l \in C_2; C_1, C_2 \in \Omega_{i-1}\}$
4. If ($\Box \in \Omega_i$, RETURN (Ω UNSATISFIABLE))
5. If ($\Omega_i = \Omega_{i-1}$, RETURN (Ω SATISFIABLE))
6. Go to step 3.

2.3.3 Chaining both methods

Let be A a propositional formula. Let be Ω the set of clauses applying Short Normal Forms to A. Now ground resolution is applied to Ω. Since Ω is equisatisfiable with A, if the result is of resolution is Ω UNSATISFIABLE then A is unsatisfiable and A is satisfiable in the other case.

3 The utility file *tree.mth*

The following functions have been developed in the utility/demo dfw file *tree.dfw* to manage binary trees, specially syntactic trees of propositional formulae.

- Basic tree management functions

- NewBTree() returns the empty (binary) tree [].
  Example:
  # NewBTree()
  # []
- MakeBtree(ro,le,ri) returns the binary tree which root is ro, his left son is le and his right son is ri.
  Example:
  # MakeBTree("O", 1, -2)
  # $[1, O, -2]$
- Root(t) returns the root node of the tree t.
  Example:
  # Root($[1, "O", -2]$)
  # O
- Left(t) returns the left son of the root node of the tree t.
  Example:
  # Left($[1, "O", -2]$)
  # -2
- Right(t) returns the right son of the root node of the tree t.
  Example:
  # Right($[1, "O", -2]$)
  # -2

- Strings and Trees
  - StringtoTree(f) makes the syntax analysis of f (f is a propositional formula delimited by quotation marks), returning the corresponding syntactic tree.
    Example:
    # StringtoTree("$(p \to q) \leftrightarrow (q \wedge \neg p)$")
    # $[[1, I, 2], F, [2, O, -1]]$
  - TreetoString(t) returns the string of the propositional formula which syntactic tree is t.
    Example:
    # TreetoString($[[1, "I", 2], F, [2, "O", -1]]$)
    # $(p \to q) \leftrightarrow (q \wedge \neg p)$

4 The package *ATPCL.mth*

The following functions have been developed in the utility/demo dfw file *ATPCL.dfw* to use DERIVE as an Automated Theorem Prover for Propositional Classical Logic. The algorithms which have been implemented are Quine, Semantic Tableaux and Short Normal Forms plus Resolution.

- Quine
  - Quine(t) to check the satisfiability of the syntactic tree t of a formula A using Quine method. If A is unsatisfiable then "UNSAT" is return. If A is satisfiable then a list of literals corresponding to a model of A is returned.

    Examples:

    # Quine(StringtoTree("$(p \to q) IFF (\neg p \vee q)$"))

    # [-1]

    # Quine(StringtoTree("$\neg((p \to q) IFF (\neg p \vee q))$"))

    # UNSAT
  - InferenceQuine(h,f) to check if the formula f (conclusion) can be deduced from the list of formulae h (hypothesis) using Quine method. In case that the inference were non-valid a countermodel is returned.

    Examples:

    # InferenceQuine(["$p \to q$","$p \vee r$","$r \to t$"],"$q \vee t$")

    # VALID INFERENCE

    # InferenceQuine(["$p \to q$","$p \vee r$","$r \to t$"],"$q \wedge t$")

    # NON VALID INFERENCE. COUNTERMODEL: $I(t) = 0, I(r) = 0, I(q) = 1, I(p) = 1$
  - TAUTQuine(f) to check if the formula f is a valid formula using Quine method. In case that the formula were non-valid a countermodel is returned.

    Examples:

    # TAUTQuine("$(p \to q) \vee (q \to p)$")

    # VALID FORMULA

    # TAUTQuine("$(p \to q) \to (q \to p)$")

    # NON VALID FORMULA. COUNTERMODEL: $I(q) = 1, I(p) = 0$
  - SATQuine(f) to check if the formula f is a satisfiable formula using Quine method. In case that the formula were satisfiable a model is returned.

    Examples:

    # SATQuine("$(p \to q) \to (q \to p)$")

SATISFIABLE FORMULA. MODEL: $I(p) = 1$
SATQuine("$(p \rightarrow q) \wedge \neg(p \rightarrow q)$")
UNSATISFIABLE FORMULA

- SemanticTableaux
  - SemanticTableaux(f) to check the satisfiability of the formula f using Semantic Tableaux method. If A is unsatisfiable then [] is return. If A is satisfiable then a list of literal corresponding to a model of f is returned.

    Examples:

    # SemanticTableaux("$\neg(p \rightarrow p)$")
    # []
    # SemanticTableaux("$p \vee q$")
    # [1]
  - InferenceSemanticTableaux(h,f) to check if the formula f (conclusion) can be deduced from the list of formulae h (hypothesis) using Semantic Tableaux method. In case that the inference were non-valid a counter-model is returned.

    Examples:

    # InferenceSemanticTableaux(["$p \leftrightarrow q$","$p \rightarrow r$","$\neg r$"],"$\neg q$")
    # VALID INFERENCE
    # InferenceSemanticTableaux(["$p \leftrightarrow q$","$p \rightarrow r$","$\neg r$"],"q")
    # NON VALID INFERENCE. COUNTERMODEL: $I(p) = 0, I(q) = 0, I(r) = 1$
  - TAUTSemanticTableaux(f) to check if the formula f is a valid formula using Semantic Tableaux method. In case that the formula were non-valid a countermodel is returned.

    Examples:

    # TAUTSemanticTableaux("$(pIMPq)or(qimpp)$")
    # VALID FORMULA
    # TAUTSemanticTableaux("$(pIMPq)AND(qimpp)$")
    # NON VALID FORMULA. COUNTERMODEL: $I(q) = 0, I(p) = 1$
  - SATSemanticTableaux(f) to check if the formula f is a satisfiable formula using Semantic Tableaux method.

    Examples:

    # SATSemanticTableaux("$(pimpq)and(qimpp)$")
    # SATISFIABLE FORMULA. MODEL: $I(q) = 0, I(p) = 0$
    # SATSemanticTableaux("$(porq)IFF(notpandNOTq)$")
    # UNSATISFIABLE FORMULA

- Short Normal forms plus Resolution
  - SNF(t). If t is the sintactic tree of A then SNF(t) returns a set of clauses (set of sets of integers) equisatisfiable with A.

    Example:

    # SNF("$(p \vee q) \wedge \neg p$")

    # {{-1, 5}, {-2, 5}, {4}, {-4, -1}, {-4, 5}, {-5, 1, 2}, {-5, 1, 4}}
  - GroundResolution(s) to check the satisfiability of the set of clauses s (set of sets of integers). It returns "SAT" or "UNSAT".

    Example:

    # GroundResolution($\{\{-1,5\},\{-2,5\},\{4\},\{-4,-1\},\{-4,5\},\{-5,1,2\},\{-5,1,4\}\}$)

    # SAT
  - InferenceResolution(h,f) to check if the formula f (conclusion) can be deduced from the list of formulae h (hypothesis) using short normal forms plus resolution method.

    Examples:

    # InferenceResolution(["$pimpq$","p"],"q")

    # VALID INFERENCE

    # InferenceResolution(["$porq$"],"$notp$"))

    # NON VALID INFERENCE
  - TAUTSResolution(f) to check if the formula f is a valid formula using short normal forms plus resolution method.

    Examples:

    # TAUTResolution("p or not p")

    # VALID FORMULA

    # TAUTResolution("p or q")

    # NON VALID FORMULA
  - SATResolution(f) to check if the formula f is a satisfiable formula using short normal forms plus resolution method.

    Examples:

    # SATResolution("p or q")

    # SATISFIABLE FORMULA

    # SATResolution("p and not p")

    # UNSATISFIABLE FORMULA

5 Conclusions and future work

5.1 Conclusions

The package *ATPCL.mth* has been developed in DERIVE. This package implement some Automated Theorem Provers for Propositional Classical Logic. The algorithms which have been implemented are: Quine, Semantic Tableaux and Short Normal Forms plus Resolution. The file *ATPCL.dfw* has a brief description of every user function and some examples. In the package, the structure of binary tree is widely used, so another package *tree.mth* has been added together with the file *tree.dfw* containing some explanations and examples too.

5.2 Future work

In the short term the implementation of the resolution method made in the package will be improved using ordered linear resolution instead of ground resolution. In the medium term, another method for automated theorem proving called TASD (see [1]) will be implemented in DERIVE. And in the long term methods for First Order Logic and for some non standard logics will be developed using DERIVE.

References

[1] Aguilera, G., P. de Guzmán, I., Ojeda-Aciego, M. y Valverde, A. Reductions for non-clausal theorem proving. *Theoretical Computer Science* 266 (1-2) (2001) 81–112.

[2] Aguilera, G., P. de Guzmán, I. Lógica para la Computación vol. I. *Ágora* (1993).

[3] Boy de la Tour, T. Minimizing the number of clauses by renaming. *Proc* 10^{th} *CADE, Kaiserslautern*, Springer, Heidelberg, (1990) 558–572.

[4] Fitting, Melvin First-Order Logic and Automated Theorem Proving *Springer-Verlag*, 1990.

[5] Jeffrey, R, Formal Logic: its scope and limits. *McGraw-Hill*, 1981.

[6] Smullyan, R. M First-Order Logic. *Springer-Verlag*, 1968.