COMPUTATIONAL LOGIC

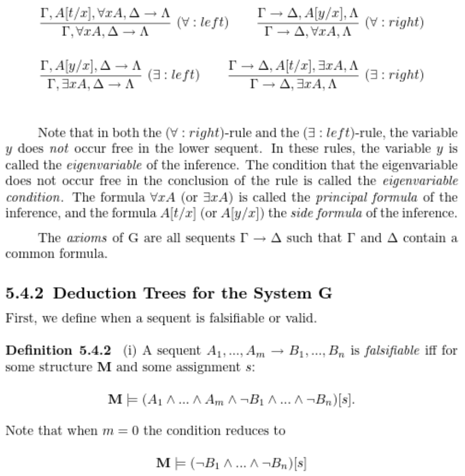
Course Introduction and Organization

Wolfgang Schreiner
Research Institute for Symbolic Computation (RISC)
Wolfgang.Schreiner@risc.jku.at

JOHANNES KEPLER
UNIVERSITY LINZ

Logic

Can be this…



$$\frac{\Gamma, A[t/x], \forall x A, \Delta \rightarrow \Lambda}{\Gamma, \forall x A, \Delta \rightarrow \Lambda} (\forall : left) \qquad \frac{\Gamma \rightarrow \Delta, A[y/x], \Lambda}{\Gamma \rightarrow \Delta, \forall x A, \Lambda} (\forall : right)$$

$$\frac{\Gamma, A[y/x], \Delta \rightarrow \Lambda}{\Gamma, \exists x A, \Delta \rightarrow \Lambda} (\exists : left) \qquad \frac{\Gamma \rightarrow \Delta, A[t/x], \exists x A, \Lambda}{\Gamma \rightarrow \Delta, \exists x A, \Lambda} (\exists : right)$$

Note that in both the $(\forall : right)$-rule and the $(\exists : left)$-rule, the variable y does *not* occur free in the lower sequent. In these rules, the variable y is called the *eigenvariable* of the inference. The condition that the eigenvariable does not occur free in the conclusion of the rule is called the *eigenvariable condition*. The formula $\forall x A$ (or $\exists x A$) is called the *principal formula* of the inference, and the formula $A[t/x]$ (or $A[y/x]$) the *side formula* of the inference.

The *axioms* of G are all sequents $\Gamma \rightarrow \Delta$ such that Γ and Δ contain a common formula.

5.4.2 Deduction Trees for the System G

First, we define when a sequent is falsifiable or valid.

Definition 5.4.2 (i) A sequent $A_1, ..., A_m \rightarrow B_1, ..., B_n$ is *falsifiable* iff for some structure **M** and some assignment s:

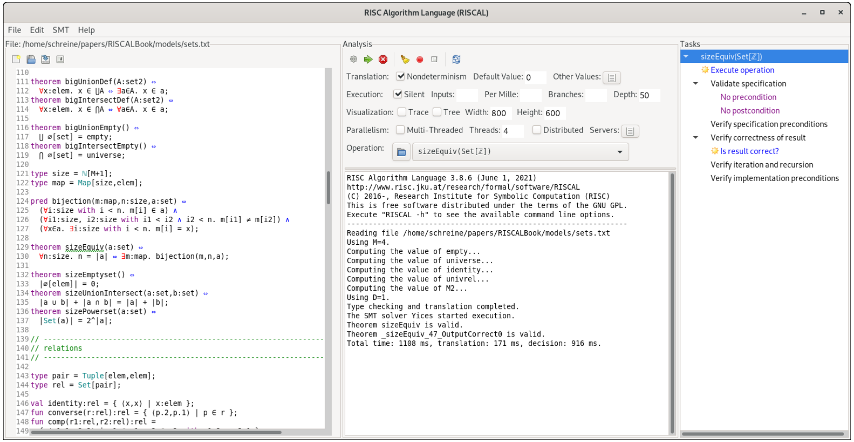
$$\mathbf{M} \models (A_1 \wedge ... \wedge A_m \wedge \neg B_1 \wedge ... \wedge \neg B_n)[s].$$

Note that when $m = 0$ the condition reduces to

$$\mathbf{M} \models (\neg B_1 \wedge ... \wedge \neg B_n)[s]$$

Logic

but also this:



RISC Algorithm Language (RISCAL)

File: /home/schreine/papers/RISCALBook/models/sets.txt

```
theorem bigUnionDef(A:set2) ⇔
  ∀x:elem. x ∈ ⋃A ⇔ ∃a∈A. x ∈ a;
theorem bigIntersectDef(A:set2) ⇔
  ∀x:elem. x ∈ ⋂A ⇔ ∀a∈A. x ∈ a;

theorem bigUnionEmpty() ⇔
  ⋃ ∅[set] = empty;
theorem bigIntersectEmpty() ⇔
  ⋂ ∅[set] = universe;

type size = ℕ[M+1];
type map = Map[size,elem];

pred bijection(m:map,n:size,a:set) ⇔
  (∀i:size with i < n. m[i] ∈ a) ∧
  (∀i1:size, i2:size with i1 < i2 ∧ i2 < n. m[i1] ≠ m[i2]) ∧
  (∀x∈a. ∃i:size with i < n. m[i] = x);

theorem sizeEquiv(a:set) ⇔
  ∀n:size. n = |a| ⇔ ∃m:map. bijection(m,n,a);

theorem sizeEmptyset() ⇔
  |∅[elem]| = 0;
theorem sizeUnionIntersect(a:set,b:set) ⇔
  |a ∪ b| + |a ∩ b| = |a| + |b|;
theorem sizePowerset(a:set) ⇔
  |Set(a)| = 2^|a|;

// -----------------------------------------------------------------
// relations
// -----------------------------------------------------------------

type pair = Tuple[elem,elem];
type rel = Set[pair];

val identity:rel = { ⟨x,x⟩ | x:elem };
fun converse(r:rel):rel = { ⟨p.2,p.1⟩ | p ∈ r };
fun comp(r1:rel,r2:rel):rel =
```

Analysis

Translation: ✓ Nondeterminism Default Value: 0 Other Values:
Execution: ✓ Silent Inputs: Per Mille: Branches: Depth: 50
Visualization: Trace Tree Width: 800 Height: 600
Parallelism: Multi-Threaded Threads: 4 Distributed Servers:
Operation: sizeEquiv(Set[ℤ])

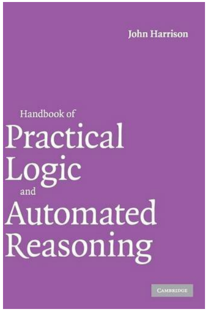
```
RISC Algorithm Language 3.8.6 (June 1, 2021)
http://www.risc.jku.at/research/formal/software/RISCAL
(C) 2016-, Research Institute for Symbolic Computation (RISC)
This is free software distributed under the terms of the GNU GPL.
Execute "RISCAL -h" to see the available command line options.
-----------------------------------------------------------------
Reading file /home/schreine/papers/RISCALBook/models/sets.txt
Using M=4.
Computing the value of empty...
Computing the value of universe...
Computing the value of identity...
Computing the value of univrel...
Computing the value of M2...
Using D=1.
Type checking and translation completed.
The SMT solver Yices started execution.
Theorem sizeEquiv is valid.
Theorem _sizeEquiv_47_OutputCorrect0 is valid.
Total time: 1108 ms, translation: 171 ms, decision: 916 ms.
```

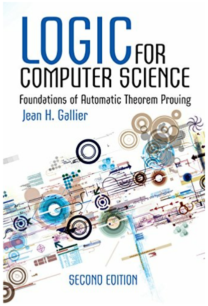
Contents

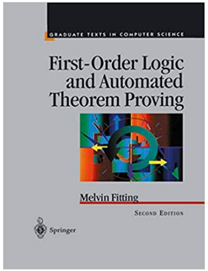
- Goal: an introduction to the computational aspects of formal logic.
  - Generally: abstract syntax, formal semantics, proving calculus.
  - Propositional logic: automatic decisions by “SAT solvers”.
  - First-order logic: model checking, automated proving, interactive proving.
  - Quantifier-free logic: automatic decisions over certain theories by “SMT solvers”.
  - Throughout the course: application examples.
- Prerequisite: already a practical understanding of formal logic.
  - Mathematics: course “Logic as a Working Language” (W. Windsteiger).
  - Computer science: course “Logic” (M. Seidl, W. Schreiner, W. Windsteiger).
  - …
- Prepares for: more in-depth courses on selected topics.
  - Propositional logic: course “SAT Solving” (FMV/Seidl).
  - First-order logic: course “Automated Resoning” (RISC/Jebelean, Kutsia).
  - …

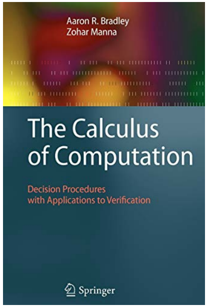
A combination of theoretical and software presentations.

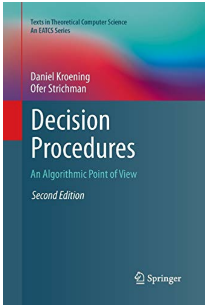
Literature











We will variously present OCaml software from John Harrison's book.

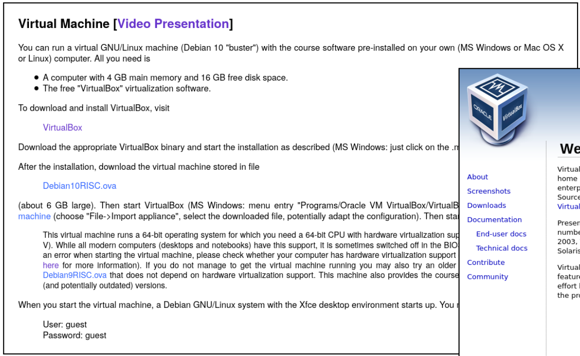
Software

- Code and resources for “Handbook of Practical Logic and Automated Reasoning”: `https://www.cl.cam.ac.uk/~jrh13/atp`
- Sequent Calculus Trainer: `https://www.uni-kassel.de/eecs/fmv/software/sequent-calculus-trainer`
- The MiniSat Page: `http://minisat.se/`
- Limboole: `http://fmv.jku.at/limboole`
- RISC Algorithm Language (RISCAL): `https://www.risc.jku.at/research/formal/software/RISCAL`
- RISC ProofNavigator: `https://www.risc.jku.at/research/formal/software/ProofNavigator/`
- Tree Proof Generator: `https://www.umsu.de/trees`
- SWI Prolog: `https://www.swi-prolog.org`
- Vampire: `https://vprover.github.io`
- Isabelle: `https://isabelle.in.tum.de`
- Z3 Theorem Prover: `https://github.com/Z3Prover`

The Course Virtual Machine

No need for self-installation, a x64 virtual machine provides all the software.

https://www.risc.jku.at/people/schreine/courses/software/#virtual



Virtual Machine [Video Presentation]

You can run a virtual GNU/Linux machine (Debian 10 "buster") with the course software pre-installed on your own (MS Windows or Mac OS X or Linux) computer. All you need is

- A computer with 4 GB main memory and 16 GB free disk space.
- The free "VirtualBox" virtualization software.

To download and install VirtualBox, visit

VirtualBox

Download the appropriate VirtualBox binary and start the installation as described (MS Windows: just click on the .

After the installation, download the virtual machine stored in file

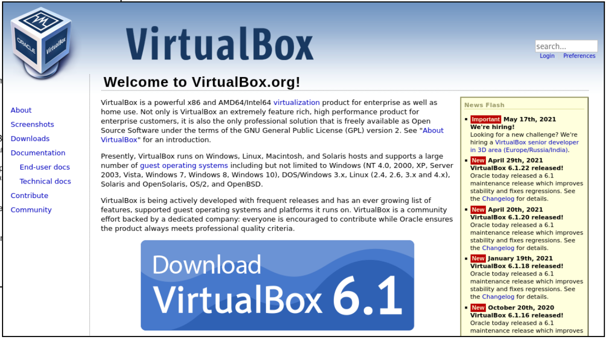
Debian10RISC.ova

(about 6 GB large). Then start VirtualBox (MS Windows: menu entry "Programs/Oracle VM VirtualBox/VirtualB machine (choose "File->Import appliance", select the downloaded file, potentially adapt the configuration). Then sta

This virtual machine runs a 64-bit operating system for which you need a 64-bit CPU with hardware virtualization sup V). While all modern computers (desktops and notebooks) have this support, it is sometimes switched off in the BIO an error when starting the virtual machine, please check whether your computer has hardware virtualization support here for more information). If you do not manage to get the virtual machine running you may also try an older Debian9RISC.ova that does not depend on hardware virtualization support. This machine also provides the course (and potentially outdated) versions.

When you start the virtual machine, a Debian GNU/Linux system with the Xfce desktop environment starts up. You

User: guest
Password: guest



Welcome to VirtualBox.org!

VirtualBox is a powerful x86 and AMD64/Intel64 virtualization product for enterprise as well as home use. Not only is VirtualBox an extremely feature rich, high performance product for enterprise customers, it is also the only professional solution that is freely available as Open Source Software under the terms of the GNU General Public License (GPL) version 2. See "About VirtualBox" for an introduction.

Presently, VirtualBox runs on Windows, Linux, Macintosh, and Solaris hosts and supports a large number of guest operating systems including but not limited to Windows (NT 4.0, 2000, XP, Server 2003, Vista, Windows 7, Windows 8, Windows 10), DOS/Windows 3.x, Linux (2.4, 2.6, 3.x and 4.x), Solaris and OpenSolaris, OS/2, and OpenBSD.

VirtualBox is being actively developed with frequent releases and has an ever growing list of features, supported guest operating systems and platforms it runs on. VirtualBox is a community effort backed by a dedicated company: everyone is encouraged to contribute while Oracle ensures the product always meets professional quality criteria.

News Flash

- Important May 17th, 2021
  We're hiring!
  Looking for a new challenge? We're hiring a VirtualBox senior developer in 3D area (Europe/Russia/India).
- New April 29th, 2021
  VirtualBox 6.1.22 released!
  Oracle today released a 6.1 maintenance release which improves stability and fixes regressions. See the Changelog for details.
- New April 20th, 2021
  VirtualBox 6.1.20 released!
  Oracle today released a 6.1 maintenance release which improves stability and fixes regressions. See the Changelog for details.
- New January 19th, 2021
  VirtualBox 6.1.18 released!
  Oracle today released a 6.1 maintenance release which improves stability and fixes regressions. See the Changelog for details.
- New October 20th, 2020
  VirtualBox 6.1.16 released!
  Oracle today released a 6.1 maintenance release which improves

Just install VirtualBox and import the virtual machine.

John Harrison’s OCaml Software

https://www.cl.cam.ac.uk/~jrh13/atp

The course VM provides shell scripts `ocamlprop` (propositional logic) and `ocamlfol` (first-order logic) for simple interactive use.

```
debian10!1> ocamlprop
        OCaml version 4.05.0
Camlp5 parsing version 7.01
# tautology << p /\ (p ==> q) ==> q >> ;;
- : bool = true
# (* press CTRL-D to stop OCaml interpreter *)

debian10!2> cat >> example.ml
tautology << p /\ (p ==> q) ==> q >> ;;
debian10!3> ocamlprop < example.ml
        OCaml version 4.05.0
Camlp5 parsing version 7.01
# - : bool = true
```

OCaml source code in `/software/Harrison/OCaml/atp_interactive.ml`

Course Outline

- Propositional Logic: Syntax and Semantics.
- Propositional Logic: Proofs.
- Propositional Logic: Modern SAT Solving.
- Propositional Logic: Applications of SAT Solving.
- First-Order Logic: Syntax and Semantics.
- First-Order Logic: Proofs.
- First-Order Logic: Software for Proving.
- First-Order Logic: The Method of Analytic Tableaux.
- First-Order Logic: The Resolution Method.
- First-Order Logic: Reasoning about Equality.
- SMT Solving: Decidable Theories.
- SMT Solving: Combining Decision Procedures.

Course Organization

13 units consisting of lectures and exercises.

- Lectures: Wolfgang Schreiner.
  - Theoretical and software presentations.
  - Graded by a written exam at the end of the semester.
- Exercises: Nikolaj Popov.
  - 10 exercise sheets to be elaborated within two weeks (paper&pencil/software).
  - Grading scheme will be explained in the first exercise unit.
- Moodle Course: `https://www.risc.jku.at/people/schreine/courses/ws2023/complogic`
  - Requires self-registration in Moodle and self-enrolment in course.
  - Questions per messages in the "Questions and Answers" forum.
  - Upload exercises as single PDF files (may include photos of handwritten sheets).
    - Possibly an archive with additional "formal" files for use by software.