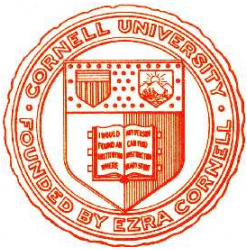


Automatisierte Logik und Programmierung

Einheit 4

Computergestützte Beweisführung



1. Aufgaben interaktiver Beweissysteme
2. Realisierungsvarianten
 - Wissensverwaltung
 - Benutzerinteraktion
 - Beweisautomatisierung
3. Arbeiten mit Nuprl

COMPUTERGESTÜTZTE BEWEISER – WOZU?

- **Formale Beweisführung ist sehr schematisch**
 - Vorgabe eines Beweisziels
 - Iteratives Anwenden von Beweisregeln und Erzeugen von Teilzielen
 - Kreativität liegt in Wahl geeigneter Regeln
- **Formale Beweisführung ist nichts für Menschen**
 - Zu viele Details: mühsame Schreiarbeit, fehleranfällig in Ausführung
 - Als Training für gründliche Vorgehensweise geeignet, aber ungeeignet für praktisches Arbeiten mit großen Beweisen
- **Anwendung formaler Regeln ist leicht zu programmieren**
 - Pattern Matching vergleicht Beweisziel mit Hauptziel der Regel
 - Teilziele können durch Instantiieren der Parameter generiert werden
- **Ein erster Schritt zur Beweisautomatisierung**
 - Benutzer **konstruieren Beweise interaktiv** durch Angabe der Regeln
 - Ermöglicht Experimente mit verschiedenen Beweisansätzen
 - Ermöglicht Programmierung von Beweisstrategien

DIE EINFACHSTE VARIANTE - KOMMANDOZEILE

- **System verwaltet Beweisbaum nur intern**
 - Benutzer gibt Theorem als initiales Beweisziel
 - Benutzer gibt Namen der Regel und ggf Parameter ein
 - System zeigt berechnete Teilziele an
 - Benutzer muß mit Kommandos durch Beweisbaum navigieren
- **Schnell zu implementieren**
 - Parser für Terme, Formeln, Regelnamen und Navigationskommandos
 - Datenstruktur für Beweisbäume
 - Regeln des Systems als Operationen auf Beweisbäumen
- **Nur für kleine Experimente geeignet**
 - Fokus liegt auf Programm zur Ausführung formaler Inferenzen
 - Term- und Formelsprache oft zu einfach (z.B `and(A, B)` statt $A \wedge B$)
 - Beweis wird nach Fertigstellung “vergessen”
 - Interaktion mit Benutzer sehr primitiv

THEOREMBEWEISEN IST MEHR ALS NUR INFERENZEN

Es geht um Computerassistenz bei der Beweisführung

- **Beweisassistenten müssen Wissen verwalten können**
 - “Echte Beweisführung” ist niemals isoliert, sondern in einem Kontext
 - **Mathematik:** Algebra, Analysis, Logik, Kategorientheorie, ...
 - **Programmierung:** Verifikation, Synthese, Optimierung, Security, ...
 - Kontext bestimmt Begriffswelt, Erkenntnisse, Methoden, ...
 - Beweisführung stützt sich massiv auf bereits bekanntes Wissen
- **Beweisassistenten benötigen “visuelles” Benutzerinterface**
 - Unterstützung für interaktive Bearbeitung von Statements/Theoremen, Beweisen, Definitionen, Theorien, informalen Kommentaren, ...
 - Präsentation formaler Konstrukte muß verständlich bleiben
- **Beweisassistenten sollten Automatisierung unterstützen**
 - Elementare Inferenzregeln sind ungeeignet für große Beweise
Beweise werden unlesbar und sind extrem mühsam zu führen
 - Automatisierung kann komplexe Folgen von Beweisschritten in Gruppen zusammenfassen und selbständig nach “trivialen” Beweisen suchen

KERNBESTANDTEILE EINES BEWEISASSISTENZSYSTEMS

- **Bibliothek**

- Verwaltung von des formalen Wissens
- Zugriff, Suche, Modifikation, globale Theorieoperationen

- **Benutzerinterface**

- Kommunikation mit der Bibliothek
- Visuelle Bearbeitung von Termen, Beweisen, Definitionen, ...

- **Inferenzmaschine**

- Anwendung von Inferenzregeln auf Beweisziele
- Automatisierung der Beweisführung

- **Optionale Komponenten**

- **Extraktion** von Programmen aus Beweisen
- **Evaluator**: Ausführung von Programmen
- **Konvertierung** in andere Formate (PDF, HTML, Fremdbeweiser, ...)

Es gibt unterschiedliche Arten der Gestaltung

BEWEISASSISTENTEN: AUFGABEN DER BIBLIOTHEK

- **Bibliothek muß Wissensverarbeitung unterstützen**
 - Erzeugung formaler Definitionen, Sätze, Beweise, Methoden, Texte
 - Strukturierung formalen Wissens in Theorien und Sub-Theorien
 - Umbenennen, Verschieben, Verlinken, Entfernen von Wissen
 - Verwendung formalen Wissens in Beweisen, Methoden und Texten
 - Durchsuchen gespeicherten Wissens, Suche nach “relevantem” Wissen
- **Wissensverarbeitung ist mehr als “Sammeln”**
 - Neue Erkenntnisse kommen hinzu, andere werden entfernt
 - Spezifische Beweise und Beweismethoden ändern sich
 - **Konsistenz** des gespeicherten Wissens muß sichergestellt sein
(Versions- / Abhängigkeitskontrolle, Vermeidung zirkulärer Beweise)
 - Vorhandensein gespeicherten Wissens benötigt eine **Rechtfertigung**
z.B. durch Verweise auf Inferenzregeln oder externe “Autoritäten”
- **Unterstützung für dezentrales Arbeiten**
 - Export, Import, Mischen und Prüfung von (Teil-)Theorien
 - Einschränkung von **Schreib- und Zugriffsrechten**

- **Einfachste und häufigste Form der Wissensverwaltung**

(Isabelle, Coq, MetaPRL, Agda, ACL2,...)

- Objekte werden konventionell in Textdatei gelagert
- Strukturierung durch Schlüsselworte (Definition, Theorem, Proof, ...)
- Dateien werden wie Programmcode sequentiell gelesen und kompiliert
- Spezielle Suchwerkzeuge operieren auf geladenen internen Daten

- **Vorteile**

- + Konventionell editierbar, vertraute Textsuche mit grep oder Emacs
- + Informationen leicht austauschbar, geringer Platzbedarf

- **Nachteile**

- Konsistenz nur durch strikt lineare Verarbeitung gesichert
- Nur ein Benutzer, nur ein aktuell sichtbares Objekt
- Keine Zugriffskontrolle: jeder kann alles überschreiben / löschen
- Lokale Bibliotheken verschiedener Nutzer nicht leicht zu integrieren
- Systemupdates können existierende Nutzerbibliotheken ungültig machen

BIBLIOTHEK: REALISIERUNG IN ISABELLE UND COQ

Library ist Textdatei, wird angezeigt in passendem Emacs mode

```
theory Num
  imports Datatype BNF_LFP
  begin

  subsection {* The @{text num} type *}

  datatype num = One | Bit0 num | Bit1 num

  text {* Increment function for type @{typ num} *}

  induct inc :: "num  $\Rightarrow$  num" where
    "inc One = Bit0 One" |
    "inc (Bit0 x) = Bit1 x" |
    "inc (Bit1 x) = Bit0 (inc x)"

  text {* Converting between type @{typ num} and type @{typ nat} *}

  induct nat_of_num :: "num  $\Rightarrow$  nat" where
    "nat_of_num One = Suc 0" |
    "nat_of_num (Bit0 x) = nat_of_num x + nat_of_num x" |
    "nat_of_num (Bit1 x) = Suc (nat_of_num x + nat_of_num x)"

  induct num_of_nat :: "nat  $\Rightarrow$  num" where
    "num_of_nat 0 = One" |
    "num_of_nat (Suc n) = (if 0 < n then inc (num_of_nat n) else One)"

  lemma nat_of_num_pos: "0 < nat_of_num x"
  by (induct x) simp_all

  lemma nat_of_num_neq_0: "nat_of_num x  $\neq$  0"
  by (induct x) simp_all

  lemma nat_of_num_inc: "nat_of_num (inc x) = Suc (nat_of_num x)"
  by (induct x) simp_all

  lemma num_of_nat_double:
    "0 < n  $\implies$  num_of_nat (n + n) = Bit0 (num_of_nat n)"
  by (induct n) simp_all
```

```
(* Interpretation of booleans as propositions *)

Definition Is_true (b:bool) :=
  match b with
  | true => True
  | false => False
  end.

(*****
** * Decidability *
*****)

Lemma bool_dec : forall b1 b2 : bool, {b1 = b2} + {b1  $\neq$  b2}.
Proof.
  decide equality.
Defined.

(*****
** * Discrimination *
*****)

Lemma diff_true_false : true  $\neq$  false.
Proof.
  discriminate.
Qed.
Hint Resolve diff_true_false : bool v62.

Lemma diff_false_true : false  $\neq$  true.
Proof.
  discriminate.
Qed.
Hint Resolve diff_false_true : bool v62.
Hint Extern 1 (false  $\neq$  true) => exact diff_false_true.
```


BIBLIOTHEK: REALISIERUNG IN ACL2

Library ist einfache Textdatei, keine optische Unterstützung

```
(in-package "ACL2")

(include-book "inequalities")

; theorems about natp, posp

(defthm natp-fc-1
  (implies (natp x)
            (<= 0 x))
  :rule-classes :forward-chaining)

(defthm natp-fc-2
  (implies (natp x)
            (integerp x))
  :rule-classes :forward-chaining)

(defthm posp-fc-1
  (implies (posp x)
            (< 0 x))
  :rule-classes :forward-chaining)

(defthm posp-fc-2
  (implies (posp x)
            (integerp x))
  :rule-classes :forward-chaining)
```

```
(defthm natp-rw
  (implies (and (integerp x)
                (<= 0 x))
            (natp x)))

(defthm posp-rw
  (implies (and (integerp x)
                (< 0 x))
            (posp x)))

(defthm |(natp a) <=> (posp a+1)|
  (implies (integerp a)
            (equal (posp (+ 1 a))
                   (natp a))))

(encapsulate
  ()
  (local
   (defthm posp-natp-l1
     (implies (posp (+ -1 x))
              (natp (+ -1 (+ -1 x))))))

  (defthm posp-natp
    (implies (posp (+ -1 x))
              (natp (+ -2 x)))
    :hints (("goal" :use posp-natp-l1))))
```

● **Globales Verständnis formalen Wissens**

(Nuprl)

- “*Mathematisches Wissen ist universell und nicht für jeden anders*”
- Wissen wird nicht lokal sondern uniform verwaltet
- Zugriffe über Datenbankmanagementsystem (vgl. Buchungssysteme)
Datenbanksprache codiert in Menüs und Buttons
- DBMS verwaltet Namensgebung, Strukturierung und Zugriffsrechte

● **Nachteile**

- Komplexeres System, kein einfaches Editieren von Text möglich
- Synchronisation, Import/Export von Theorien nur über das DBMS

● **Vorteile**

- + Multiuser-Kooperationen möglich, viele Objekte simultan sichtbar
- + Selektive Sichten und Kombinationen von Theorien möglich
- + Zugriffskontrolle und Transaktionskonzept mit vielfachem Undo/Redo
sichert Konsistenz, ermöglicht Versionskontrolle,
erhöhte Sicherheit gegenüber Mißbrauch, Irrtümern, Absturz

BIBLIOTHEK: REALISIERUNG IN NUPRL

- TERM: Navigator

```
MkTHY*  OpenThy*  CloseThy*  ExportThy*  ChkThy*  ChkAllThys*  ChkOpenThy*
CheckMinTHY*  MinTHY*  EphTHY*  ExTHY*
```

```
Mill*  ObidCollector*  NameSearch*  PathStack*  RaiseTopLoops*
PrintObjTerm*  PrintObj*  MkThyDocObj*  ProofHelp*  FixRefEnvs*
CpObj*  reNameObj*  EditProperty*  SaveObj*  RmLink*  MkLink*  RmGroup*
```

```
MkTHM*  MkML*  AddDef*  AddRecDef*  AddRecMod*  AddDefDisp*  AbReduce*
Act*  DeAct*  MkThyDir*  RmThyObj*  MvThyObj*
```

```
↑↑↑↑  ↑↑↑  ↓↓↓↓  ↓↓↓  <>  ><
```

```
Navigator: [num_thy_1; standard; theories]
```

```
List Scroll : Total 159, Point 5, Visible : 10
```

```
-----
CODE  TTF  RE_init_num_thy_1
COM   TTF  num_thy_1_begin
COM   TTF  num_thy_1_summary
COM   TTF  num_thy_1_intro
DISP  TTF  divides_df
-> ABS  TTF  divides
STM   TTF  divides_wf
STM   TTF  comb_for_divides_wf
STM   TTF  zero_divs_only_zero
STM   TTF  one_divs_any
-----
```

- Navigieren durch Bibliothek mit Maus und Pfeiltasten
- Öffnen von Objekten mit passendem Editor (Mausclick oder Pfeil)
- Ausführen von Bibliothekskommandos mit Buttons

Visuelle Unterstützung zur Bearbeitung von Wissen

- **Benutzer muß Theorien interaktiv entwickeln können**
 - Erzeugung von Definitionen, Statements, Führen von Beweisen, Strukturierung in Theorien, informale Dokumentation, ...
 - System muß (Zwischen-)Ergebnisse der Bearbeitung anzeigen
 - System sollte Unterstützung für typische Arbeitstechniken anbieten z.B. Rückblättern, gleichzeitige Sicht auf viele Daten, alternative Beweisansätze, ...
- **Layoutfragen sind wichtig**
 - Verständnis formaler Texte ist of abhängig von gewählter Notation
Computernahe Formalisierung erschwert Entwurf “guter” Theorien
 - Formales System sollte gängige mathematische Notation unterstützen

- **Einfache Erweiterung der Kommandozeile**

(Isabelle, Coq, MetaPRL, ACL2, SpecWare)

- Definitionen, Sätze, Beweisskripte, etc. entstehen durch Eingabe von Schlüsselworten, Formeln und Kommandos in Textdatei
- Interface (e.g. *ProofGeneral*, *CoqIDE*, *jEdit*) zwischen Textdatei und Kommandozeile des “eigentlichen” Systems unterstützt serielle (ggf. auch parallele) Verarbeitung von Theorien und Beweisskripten
- Ausgaben des Systems werden separaten Fenstern angezeigt

- **Vorteile**

- + Für Anfänger leichter zu erlernen, Einsatz vertrauter Editoren möglich
- + Geringer Implementierungsaufwand

- **Nachteile**

- Textbasiertes Vorgehen, nur aktuelles Beweisziel sichtbar
- Formale Notation eingeschränkt durch Fähigkeiten des Parsers

BENUTZERINTERFACE: REALISIERUNG IN ISABELLE

Interface zeigt Beweisknoten entsprechend der Cursorposition

Isabelle2014 - Logic.thy (modified)

```
File Edit Search Markers Folding View Utilities Macros Plugins Help
Logic.thy (~/Desktop/)
theory Logic
imports Main
begin

lemma not_over_and:
  "  $\forall A B. (\neg A \vee \neg B) \longrightarrow \neg(A \wedge B)$  "
  apply (rule allI)
  apply (rule allI)
  apply (rule impI)
  apply (rule notI)
  apply (rule_tac P="A" and Q="B" in conjE)
  apply (simp)
  apply (rule_tac P="¬A" and Q="¬B" in disjE)
  apply (simp)
  apply (rule_tac P="A" in notE)
  apply (simp)
  apply (simp)
  apply (rule_tac P="B" in notE)
  apply (simp)
  apply (simp)
done
```

Documentation Sidekick Theories

Auto update Update Search: 100%

proof (prove): depth 0

goal (1 subgoal):

1. $\bigwedge A B. \neg A \vee \neg B \implies A \wedge B \implies \text{False}$

Output Query Sledgehammer Symbols

10,19 (160/476) (isabelle,sidekick,UTF-8-Isabelle)Nm r o UG 337/549MB 11:53 AM

Isabelle2014 - Logic.thy (modified)

```
File Edit Search Markers Folding View Utilities Macros Plugins Help
Logic.thy (~/Desktop/)
theory Logic
imports Main
begin

lemma not_over_and:
  "  $\forall A B. (\neg A \vee \neg B) \longrightarrow \neg(A \wedge B)$  "
  apply (rule allI)
  apply (rule allI)
  apply (rule impI)
  apply (rule notI)
  apply (rule_tac P="A" and Q="B" in conjE)
  apply (simp)
  apply (rule_tac P="¬A" and Q="¬B" in disjE)
  apply (simp)
  apply (rule_tac P="A" in notE)
  apply (simp)
  apply (simp)
  apply (rule_tac P="B" in notE)
  apply (simp)
  apply (simp)
done
```

Documentation Sidekick Theories

Auto update Update Search: 100%

proof (prove): depth 0

goal (2 subgoals):

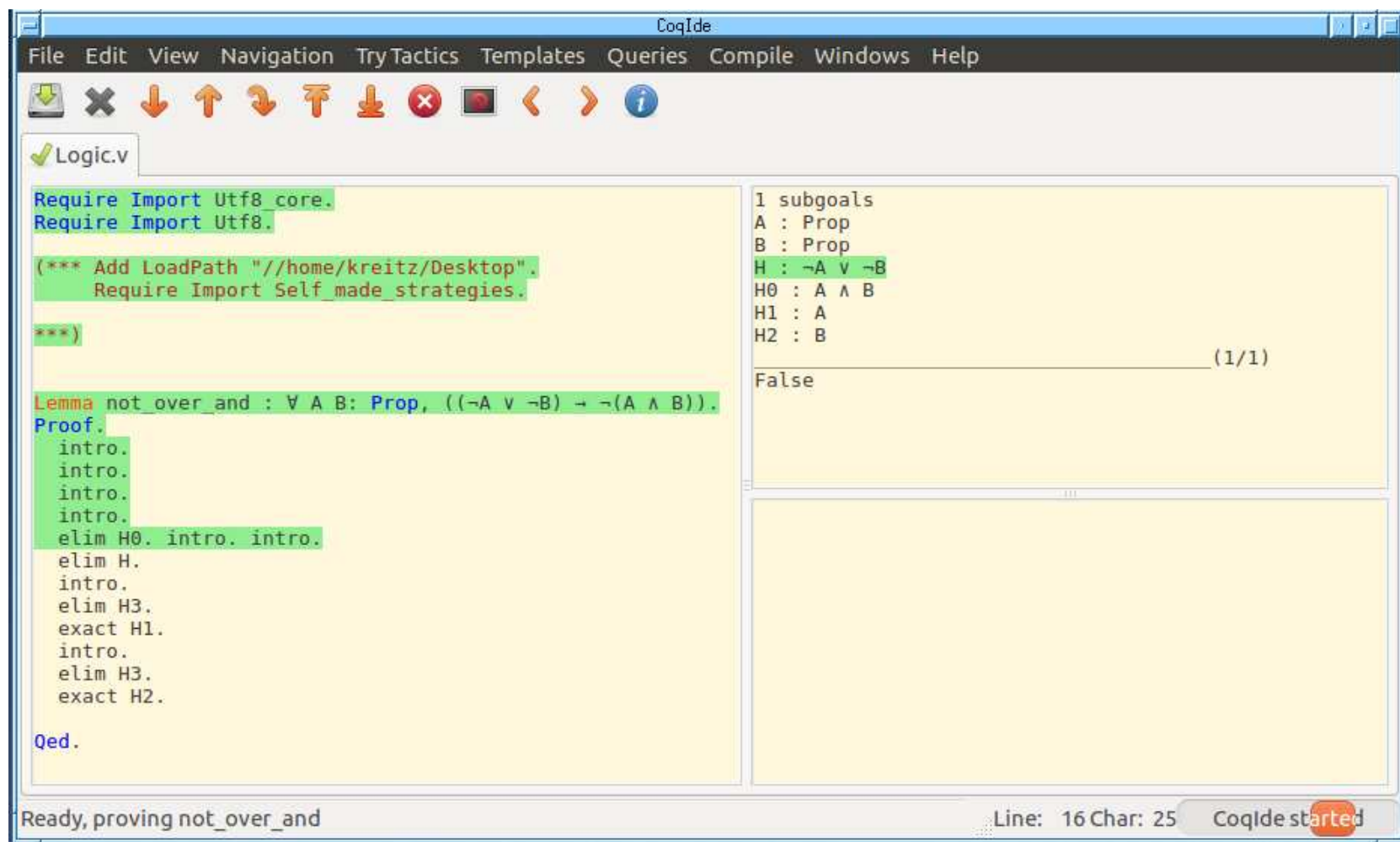
1. $\bigwedge A B. \neg A \vee \neg B \implies A \wedge B \implies A \wedge B$
2. $\bigwedge A B. \neg A \vee \neg B \implies A \wedge B \implies A \implies B \implies \text{False}$

Output Query Sledgehammer Symbols

11,43 (203/476) (isabelle,sidekick,UTF-8-Isabelle)Nm r o UG 345/549MB 11:53 AM

BENUTZERINTERFACE: REALISIERUNG IN COQ

Interface reicht Skript-Kommandos and Coq-Interpreter und zeigt Beweisknoten entsprechend der Cursorposition



The screenshot shows the CoqIDE window with the following content:

```
File Edit View Navigation TryTactics Templates Queries Compile Windows Help
Logic.v
Require Import Utf8_core.
Require Import Utf8.

(** Add LoadPath "//home/kreitz/Desktop".
    Require Import Self_made_strategies.
***)

Lemma not_over_and : ∀ A B: Prop, ((¬A ∨ ¬B) → ¬(A ∧ B)).
Proof.
  intro.
  intro.
  intro.
  intro.
  elim H0. intro. intro.
  elim H.
  intro.
  elim H3.
  exact H1.
  intro.
  elim H3.
  exact H2.
Qed.
```

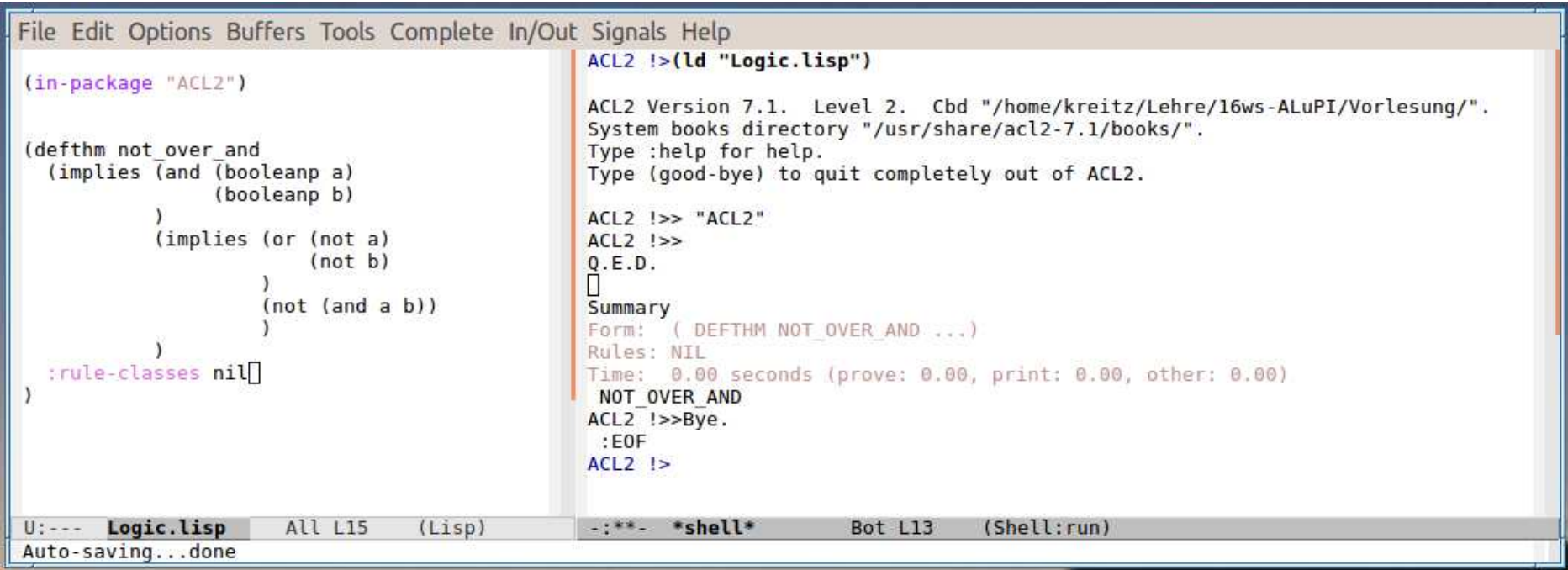
The right-hand pane displays the current proof state:

```
I subgoals
A : Prop
B : Prop
H : ¬A ∨ ¬B
H0 : A ∧ B
H1 : A
H2 : B
(1/1)
False
```

The status bar at the bottom indicates: Ready, proving not_over_and Line: 16 Char: 25 Coqide started

BENUTZERINTERFACE: REALISIERUNG IN ACL2

**Shell Kommando lädt Theoriedatei in ACL2-System
System versucht automatischen Beweis und zeigt Ergebnis
oder ausführliche Fehlermeldung**



```
File Edit Options Buffers Tools Complete In/Out Signals Help

(in-package "ACL2")

(defthm not_over_and
  (implies (and (booleanp a)
                (booleanp b))
            (implies (or (not a)
                        (not b))
                    (not (and a b)))))
  :rule-classes nil

ACL2 !>(ld "Logic.lisp")

ACL2 Version 7.1. Level 2. Cbd "/home/kreitz/Lehre/16ws-ALuPI/Vorlesung/".
System books directory "/usr/share/acl2-7.1/books/".
Type :help for help.
Type (good-bye) to quit completely out of ACL2.

ACL2 !>> "ACL2"
ACL2 !>>
Q.E.D.
□
Summary
Form: ( DEFTHM NOT_OVER_AND ...)
Rules: NIL
Time: 0.00 seconds (prove: 0.00, print: 0.00, other: 0.00)
NOT_OVER_AND
ACL2 !>>Bye.
:EOF
ACL2 !>

U:--- Logic.lisp All L15 (Lisp) -:***- *shell* Bot L13 (Shell:run)
Auto-saving...done
```

- **Spezialeditoren für Objekte der Wissensbank** (Nuprl)
 - Benutzer navigiert visuell durch Bibliothek, Beweisbaum, ...
 - Notation für formale Objekte unabhängig von interner Darstellung
 - Struktureditoren unterstützen Eingabe und Manipulation verschiedenartiger Objekte
- **Nachteile**
 - Steilere Lernkurve für Anfänger (es gibt mehr zu erlernen)
 - Aufwendigere Implementierung
- **Vorteile**
 - + Beliebig paralleler Zugriff erlaubt gleichzeitig sichtbare Information
 - + Parallele Bearbeitung mehrerer Beweisziele
 - + Parallele Bearbeitung mehrerer Beweisversuche für dasselbe Ziel
 - + Flexible Syntax ohne Notwendigkeit für aufwendige Parser
 - + Trennung zwischen Notation und Bedeutung ermöglicht Anpassung der Präsentation an Papierform und nachträgliche(!) Veränderungen

BENUTZERINTERFACE: REALISIERUNG IN NUPRL

- TERM: Navigator

```
....  
MkTHM*  MkML*  AddDef*  AddRecDef*  ...  
↑↑↑↑  ↑↑  ↓↓↓↓  ↓↓  <>  >>  
Navigator: [kreitz; user; theories]  
List Scroll : Total 1, Point 0, Visible : 1  
-----  
-> STM  FFF  not_over_and  
-----
```

- PRF: not_over_and

```
# top  
∀A,B:ℙ. (((¬A) ∨ (¬B)) ⇒ (¬(A ∧ B)))  
BY allI  
  
# 1  
1. A:ℙ  
┆ ∀B:ℙ. (((¬A) ∨ (¬B)) ⇒ (¬(A ∧ B)))  
BY |
```

● Erzeugung von Theoremen

- Benutzer generiert Theorem-Objekt in Wissensbank (MkTHM)
- Benutzer öffnet Beweiseditor (Maus-Click) und gibt Beweisziel ein
- Beweisziel wird unmittelbar in Bibliothek gesichert

● Beweisführung

- Benutzer gibt Beweisregel (oder Taktik) im **Regelslot** an
- Inferenzmaschine kann **synchron** oder **asynchron** aufgerufen werden
- Teilziele werden unmittelbar in Bibliothek gespeichert und angezeigt
- Sichtbarer Anteil des Beweisbaums nur abhängig von Fenstergröße
Benutzer navigiert ggf. visuell in Unterziele

AUFGABEN DER INFERENZMASCHINE

- **Verarbeitung von Regeln und Beweisstrategien**
 - Prüfen der Anwendbarkeit und Instantiierung von Parametern
 - Berechnen (und Abspeichern) der zu erzeugenden Unterziele
- **Unterstützung von Beweisautomatisierung**
 - Ergänzung des Regelsystems durch Laden neuer Beweisprogramme
 - Programmierte Anwendung von Inferenzregeln, Simplifikation, Entscheidungsprozeduren, automatische Beweissuche, ...
 - Spezialstrategien für bestimmte Anwendungsbereiche
 - Erweiterung existierender Beweisprogramme
 - Integration von Modulen, Erweiterung von Parameterlisten, ...
 - Optimierung komplexer Beweisprogramme
 - Compilierung, Caching, ...

MÖGLICHE ARTEN VON BEWEISAUTOMATISIERUNG

● Abgeleitete Inferenzregeln

(Nuprl, Coq, Isabelle)

- Anwendung bewiesener Theoreme der Form $(\forall \hat{x}) A[\hat{x}] \Rightarrow B[\hat{x}]$ als Regel
Matching von $B[\hat{x}]$ gegen Beweisziel, Instantiiertes $A[\hat{x}]$ wird Unterziel
- Mechanismus implementierbar als leicht anzuwendende **Lemma**-Regel
- Ermöglicht massive Erweiterung der Menge “primitiver” Regeln

● Beweistaktiken

(Nuprl, Coq)

- Programmierte Steuerung der Anwendung elementarer Inferenzregeln
z.B. Zusammensetzung von Regeln durch Verkettung, Iteration, etc.
- Komplexe Taktiken dürfen für die Analyse auf alle Teile des Systems zugreifen um passende Regeln auszuwählen
- **Benutzerdefinierbare Erweiterung** des Inferenzsystems
ohne das Risiko inkorrekte Beweise zu generieren
- Einfach zu implementierender Mechanismus

MÖGLICHE ARTEN VON BEWEISAUTOMATISIERUNG II

- **Komplexe Beweisuchprogramme für spezifische Aufgaben**
(Nuprl, Coq, Isabelle, ACL2, Agda, ...)
 - Entscheidungsprozeduren, Simplifikatoren, **Auto**-Prozeduren
 - Oft sehr mächtig und vielseitig verwendbar
 - Implementierung durch Systementwickler als komplexe Taktik oder als (meist unverifizierte) Systemprozedur
- **Benutzerdefinierbare Erweiterung von Systemprozeduren**
(Nuprl, Coq, Isabelle)
 - Theoreme oder Taktiken können in “hooks” eingefügt werden
z.B. neue beweisene Gleichheiten für Simplifikation
oder neue Schlußformen für Auto
 - Naiver Einsatz führt schnell zu Endlosschleifen in der Prozedur
- **Steuerungsparameter / Hints** (Coq, Isabelle, ACL2)
 - Benutzer kann Suchtiefe, Abarbeitungsreihenfolge, etc. beeinflussen
 - Braucht Verständnis der konkreten Arbeitsweise von Simplifikator/Auto

DAS INTERAKTIVE BEWEISSYSTEM NUPRL

“PROOF REFINEMENT LANGUAGE, VERSION ν ”

- **Anfänge liegen mehr als 30 Jahre zurück** (1984)
 - Nuprl 1 (Symbolics): Beweissystem für intuitionistische Typentheorie
 - Nuprl 2: Unix Version
 - Nuprl 3: Strategische Beweisführung (komplexe Taktiken) (1987)
 - Nuprl 4: Logisches Framework (Kalkül in Bibliothek enthalten) (1992)
 - Nuprl 5: Systemkomponenten als kommunizierende Prozesse (1999)
- **Anwendungen in Mathematik und Programmierung**
 - Girard’s Paradox, Higman’s Lemma (1987)
 - **Verifikation & Optimierung von Software**
 - Logikschaltungen, SCI cache coherency protocol (1993)
 - Ensemble Kommunikationssystem (100000 Zeilen ML Code) (1998)
 - Paxos und Consensus Protokolle (2015)
 - Cubical Type Theory (2016)

ARBEITEN MIT NUPRL

- **Basiskomponenten des Systems sind separate Prozesse**

- **Datenbank**: Managementsystem für alle Datenobjekte
- **Bibliothek**: Verwaltung des Wissens, Gateway zur Außenwelt
- **Refiner**: Ausführung der Inferenzmechanismen (mehrfache Prozesse möglich)
- **Editor**: Benutzerinterface (mehrfache Prozesse möglich)

- **Benutzerstart**

- **Empfohlen**: Zugriff auf laufende Version über VNC
 - Datenbank, Bibliothek, Refiner und Editoren sind bereits gestartet
 - Benutzer greift auf laufenden Editorprozess zu oder startet ihn neu
 - System zeigt Navigator und Kommandoloop
- **Virtuelle Maschine (Standalone)**:
 - System mit Extras der Vorlesung ist vorinstalliert und gestartet
- **Eigene Installation (Standalone)**:
 - Download der Binaries und Bibliothek als komprimiertes Archiv
 - **Derzeit nicht zu empfehlen**, da nur veraltete Versionen verfügbar

ARBEITEN MIT NUPRL II

- TERM: Navigator

```
CpObj*  reNameObj*  SaveObj*  RmLink*  MkLink*
MkTHM*  MkML*      AddDef*   AddRecDef* AddDefDisp*
Act*    DeAct*    MkTHY*  MkThyDir*  RmThyObj*  MvThyObj*
↑↑↑↑  ↑↑↑  ↓↓↓↓  ↓↓↓  <>  ><
Navigator: [num_thy_1; standard; theories]
List Scroll : Total 159, Point 4, Visible : 5
-----
CODE  TTF  RE_init_num_thy_1
COM   TTF  num_thy_1_begin
DISP  TTF  divides_df
-> ABS  TTF  divides
STM   TTF  divides_wf
-----
```

● Erzeuge Objekte im Navigator

- Theorien zur Strukturierung
- Definitionen: Abstraktionen und separate Displayform
- Theoreme (für Beweise),
- Code Objekte (für Taktiken), ...

● Öffnen des Objektes startet geeigneten Editor

- **Termeditor**: Strukturelles Editieren von Termen in Präsentationsform
- **Beweiseditor**: Beweisführung und Navigation durch Beweisbäume
- weitere **Objekteditoren** für spezifische Objekte

● Graphische Interaktion bewußt einfach gestaltet

- Textterminal-Version erlaubt Fernzugriff bei geringer Bandbreite
- Für Standalone wären moderne GUIs möglich

(aufwendig)

DIENSTLEISTUNGEN DES NAVIGATORS

- **Visuelle Navigation durch Bibliothek**
 - Keyboard- oder Maus-gesteuertes **Durchlaufen**
 - Patterngesteuerte **Namensuche**
 - **Springen** zu gespeicherten Positionen
- **Ausführung von Bibliothekskommandos**
 - **Vorbereitete “Buttons”** für die wichtigsten Operationen
 - Erzeugung von Objekten, Theorien, Definitionen, Modulen
 - Löschen, Kopieren, Verschieben, Umbenennen, Drucken, ...
 - Import, Export, Drucken und Dokumentation von Theorien
 - Aufruf der Operationen öffnet **Kommandomenü**
- **Undo und Redo für jede Operation**
- **Anpassbar**
 - Buttons und Erscheinungsbild **durch Bibliotheksobjekte definiert**

EIGENSCHAFTEN DES TERMEDITORORS

- **Mathematische Notation erlaubt keine Parser**
 - Zu reichhaltig (nicht kontextfrei) und nicht einheitlich geregelt
 - Notation ist keine gute Repräsentationform für logische Konzepte
- **Nuprl trennt Notation von Struktur**
 - Logische Struktur leichter zu verarbeiten
 - Separate Darstellungsform sorgt für verständliche Notation
- **Termeditor ist Struktureditor**
 - **Editiere logische Struktur** von Termen bei gleichzeitiger Präsentation der Darstellungsform auf dem Bildschirm
 - Erzeugung des Termbaums durch **Eintrag in Slots der Darstellungsform**
 - Benutzer kann **Layout** von Termen separat modifizieren
 - Kenntnis der genauen Syntax nicht erforderlich
 - **Umdenken** erforderlich: keine lineare Eingabe von Text

Benutzer kann mit verständlicher Notation arbeiten

Aktiv, wenn Cursor in einem Termslot ist

● Eingabe neuer Terme

- Eingabe des Termnamens erzeugt
Template mit Subtermslots

- z.B. Eingabe von `all` \leftarrow erzeugt

\forall [var] : [type] . [prop]

Typisierte Prädikatenlogik mit geringfügig anderer Syntax

- Unbekannte Namen werden als Variablen interpretiert

not	$\neg A$
and	$A \wedge B$
or	$A \vee B$
implies	$A \Rightarrow B$
all	$\forall x:T.A$
exists	$\exists x:T.A$

● Modifikation von Termen

- Navigation im Syntaxbaum mit Emacs-ähnlichen Kommandos
- Variablennamen und Zahlen können wie Text verändert werden
- Teilterme müssen gelöscht und neu erzeugt oder kopiert werden

EIGENSCHAFTEN DES BEWEISEDITORS

- **Visuelle Entwicklung von Top-Down Beweisen**

- Strukturierte Bearbeitung von Beweisbäumen
- Navigation durch Beweisbaum mit Maus und Keyboard
- Lokales Arbeiten im einzelnen Beweisknoten möglich
- Kontrolliertes Interface zum Refiner (via Library Gateway)
- Graphische Interaktion einfach gestaltet

- **Operationen auf Beweisen**

(vgl. Folie 10)

- Erzeugung von Beweiszielen mit Term-Editor
Beweisziel wird nach Eingabe in Bibliothek gesichert
- Synchroner oder asynchroner Ausführung von Beweisregeln
Anzeige der erzeugten Teilziele
- Komprimierung und Expansion bis zu elementaren Schritten
- Verarbeitung von Backup-Beweisen und ‘Schmierblatt’-Beweisen
- Erzeugung von Evidenz-Termen

TYPISCHER BEWEISKNOTEN

```
- PRF: intsqrt
① # top 1
②
③ 1. x:N
   ⊢ ∃y:N. y2≤x ∧ x<(y+1)2
④ BY NatInd 1
⑤ # 1 1
   .....basecase.....
   ⊢ ∃y:N. y2≤0 ∧ 0<(y+1)2
⑥ BY exR 「0」
   There is 1 hidden subgoal
⑤ # 1 2
   .....upcase.....
   1. x:ℤ
   2. 0<x
   3. ∃y:N. y2≤x-1 ∧ x-1<(y+1)2
   ⊢ ∃y:N. y2≤x ∧ x<(y+1)2
⑤ BY
```

```
- PRF: intsqrt
① # top 1 2
② .....upcase.....
③ 1. x:ℤ
   2. 0<x
   3. ∃y:N. y2≤x-1 ∧ x-1<(y+1)2
   ⊢ ∃y:N. y2≤x ∧ x<(y+1)2
④ BY |
```

- ① Status und Adresse im Beweisbaum
- ② Annotation des Beweisknotens
- ③ Beweisziel (Sequenz)
- ④ Angewandte Beweistaktik
- ⑤ Teilziele mit Status, Adresse, Sequenz (neue Hypothesen)
- ⑥ Beweise der Teilziele, sofern vorhanden

BEWEISTAKTIKEN IN NUPRL

- **Benutzerprogrammierbare Beweisstrategien**
 - Planung und Suche von Beweisen
 - Strukturierung von Beweisen (Verstecken überflüssiger Details)
 - Abgeleitete Inferenzregeln für benutzerdefinierte Theorien
 - Austesten komplexer Beweis-/Syntheseverfahren in sicherer Umgebung
- **Formalisiert als Metalevel-Programme in Classic-ML**
 - Formale metasprachliche Ausdrücke steuern Regelanwendungen
 - Einfache Komposition von Regeln und Taktiken durch **Tacticals**

t_1 THEN t_2 :	Wende t_2 auf alle von t_1 erzeugten Teilziele an
t THENL [$t_1; \dots; t_n$]:	Wende t_i auf das i -te von t erzeugte Teilziel an
t_1 ORELSE t_2 :	Wende t_1 an. Falls dies fehlschlägt, wende t_2 an
Repeat t :	Wiederhole Taktik t bis sie fehlschlägt
Try t :	Wende t an, Bei Fehlschlag lasse den Beweis unverändert
Complete t :	Wende t nur an, wenn der Beweis vollständig wird
Progress t :	Wende t nur an, wenn ein "Fortschritt" erzielt wird

- **Taktiken sind immer korrekt**
 - Taktikbeweis wird expandiert zu Beweis mit elementaren Regeln

PROGRAMMIERUNG EINES TAKTISCHEN BEWEISERS

Sortiere Regelanwendungen nach Aufwand für Beweissuche

```
let simple_prover = Repeat
  (
    Hypothesis
  ORELSE contradiction
  ORELSE InstantiateAll
  ORELSE InstantiateEx
  ORELSE conjunctionE
  ORELSE existentialE
  ORELSE nondangerousI
  ORELSE disjunctionE
  ORELSE not_chain
  ORELSE iff_chain
  ORELSE imp_chain
  );;

letrec prover = simple_prover
  THEN Try (
    Complete (orI1 THEN prover)
    ORELSE (Complete (orI2 THEN prover)))
  ;;
```

ANWENDUNG VON prover UND ERZEUGTER BEWEIS

```
dist_or_not 2016_09_05-PM-02_29_15 @edd,ck @vm-ubuntu-10
* top 1
┆ ((¬P) ∨ (¬Q)) ⇒ (¬(P ∧ Q))
* BY prover

dist_or_not 2016_09_05-PM-02_29_15 2016_09_05-PM-02_31_36 @edd,ck @vm-ubur
* top
┆ ((¬P) ∨ (¬Q)) ⇒ (¬(P ∧ Q))
* BY impliesR
* 1
4. (¬P) ∨ (¬Q)
┆ ¬(P ∧ Q)
* BY notR
* 1 1
4. (¬P) ∨ (¬Q)
5. P ∧ Q
┆ False
* BY andL (5)
* 1 1 1
4. (¬P) ∨ (¬Q)
5. P
6. Q
┆ False
* BY orL (4)
```

ARBEITEN MIT NUPRL IN POTSDAM

- **Nuprl 5 ist immer noch Experimentalsoftware**
 - Hohe Flexibilität führt zu steiler Lernkurve für Interface
 - Manual auf Webseite zu finden (Vorlesungsseite bzw. nuprl.org) müsste aber in einigen wichtigen Punkten überarbeitet werden
- **Installation auf eigenem Rechner zur Zeit schwierig**
 - Download Version ist einige Jahre alt und konzipiert für Ubuntu 10
 - Benötigt **CMU Common Lisp** oder **SBCL** für Linux
 - Systemanforderungen: ≥ 2 GB RAM, 10GB freier Plattenplatz
- **Verfügbare virtuelle Maschinen**
 - **Geschlossene Ubuntu-10 Installation:** mit Extras der Vorlesung Version für VMPlayer (ca. 8GB) von 2012, historisch gewachsen
 - **Cornell Version von 2015:** aktuelle Binaries und Bibliothek ohne eigenes X-Window System (Zugang über lokales VNC) (zur Zeit noch) ohne Extras der Vorlesung
 - **VNC Version auf Server der Theoriegruppe**
Cornell Version von 2015, vorgestartet, mit einigen Extras Anleitungen zum Aufruf werden demnächst bereitgestellt