

Automatisierte Logik und Programmierung

Einheit 14

Beweisautomatisierung für die Logik erster Stufe



1. Taktische Beweismethoden
2. Maschinennahe Beweistechniken
3. **JProver**: Hybride Beweiskonstruktion

ARTEN DER BEWEISFÜHRUNG IN LOGIK ERSTER STUFE

- **Interaktive Anwendung logischer Regeln**

- Benutzer gibt Regeln des Sequenzenkalküls und Parameter an
- System führt Regeln aus und liefert Teilziele

Mühsam, aber sicher

- **Taktikbasierte Beweissuche**

- Taktik sucht nach anwendbaren Regeln
- Analyse von Konklusion & Hypothesen zur Parameterbestimmung

Hilfreich in der Praxis, aber unvollständig wegen begrenzter Vorausschau

- **Vollautomatische Beweisverfahren** $\mapsto$ CADE, TABLEAUX, ...

- Transformation einer Sequenz in effiziente Datenstruktur
- Charakterisierung von Gültigkeit durch Eigenschaften dieser Struktur
- Beweiser benutzt Standardverfahren zur Prüfung der Eigenschaften
- Viele “Standalone” Methoden für klassische Logik
- Nur wenige Verfahren erweiterbar auf konstruktive Logik

Integration aufwendig, da Konsistenzcheck oder Beweisterme erforderlich

ENTWICKLUNG EINES TAKTIKBASIERTEN BEWEISERS: SCHRITTWEISE STEIGERUNG DES AUTOMATISIERUNGSGRADES

1. Interaktion mit Regeln der Logik erster Stufe

- Verstecke zugrundeliegende Basiskonstrukte der Typentheorie

2. Automatische Bestimmung anwendbarer Regeln

- Regel ergibt sich oft unmittelbar aus Beweiskontext

3. Verkettung von Implikationen & Äquivalenzen

- Aufbau einer kurzen Serie von Argumenten, die zum Ziel führen

4. Metalevel Analyse zur Instantiierung von Quantoren

- Bestimme einzusetzende Terme durch (partiell) Matching

5. Sortiere Beweistechniken nach Aufwand

- Beschleunigt Beweissuche, wenn mehrere Möglichkeiten bestehen

Verhältnismäßig leicht zu implementieren und erweitern

TAKTIKBASIERTE BEWEISFÜHRUNG I:

EINBETTUNG DER REGELN DER LOGIK ERSTER STUFE

- **Logik ist eingebettet über Curry-Howard Isomorphie**

- Implementierung der Regeln ist Dekomposition + Wohlgeformtheitstest

```
let allR = D 0 THENW Auto
```

- **Beschränke Anwendung der Regeln auf geeignete Ziele**

```
let andR = TryOn D 0 is_and_term
```

```
and orR1 = TryOn (SEL 1 D) is_or_term THENW Auto
```

```
and orR2 = TryOn (SEL 2 D) is_or_term THENW Auto
```

```
and impR = TryOn D 0 is_imp_term THENW Auto
```

```
⋮
```

- **Erzeuge gekapselte Varianten der Regeln**

```
andI, orI1, orI2, impI, ..., andEi, orEi, impEi, ...,
```

- Regeln werden bei Inspektion interner Beweise nicht aufgefaltet
- Tactical Run konvertiert Taktik in (eingebettete) Elementarregel

TAKTIKBASIERTE BEWEISFÜHRUNG II: BESTIMMUNG ANWENDBARER REGELN

- **Wende Taktik auf erste passende Hypothese an**

- Tactical `TryAllHyps` durchsucht Hypothesen auf Anwendbarkeit

```
let contradiction = TryAllHyps falseE is_false_term
let conjunctionE  = TryAllHyps andE is_and_term
let disjunctionE  = TryAllHyps orE is_or_term
let existentialE   = TryAllHyps exE is_ex_term
```

- **Bestimme Namen einer Einführungsregel**

```
let nondangerousI pf =
  let kind = operator_id_of_term (conclusion pf)
  in
    if mem kind ['all'; 'not'; 'implies'; 'iff'; 'and']
    then Run (kind ^ 'R') pf
    else failwith 'tactic inappropriate'
```

TAKTIKBASIERTE BEWEISFÜHRUNG III:

VERKETTUNG VON IMPLIKATIONEN & ÄQUIVALENZEN

- **Chaining: Wiederholte Anwendung einer Taktik**

- Iteriere Taktik t auf ausgewählten Hypothesen
- Letzter Schritt ist Anwendung einer Basistaktik

```
let Chain t hyps groundtac =  
  letrec chainfor i =  
    groundtac  
    ORELSE if i=0 then Id  
            else TryOn hyps (\h.t h THEN Complete (chainfor(i-1)))  
  in chain_for chain_limit
```

- **Erzeugung konkreter Beweisketten**

```
let imp_chain pf =  
  Chain impE (select_hyps is_imp_term pf) Hypothesis pf  
and not_chain =  
  TryAllHyps (\pos. notE pos THEN imp_chain) is_not_term  
and iff_chain =  
  TryAllHyps (\pos. (iffE pos THEN (imp_chain ORELSE not_chain))  
               ORELSE (iffE_b pos THEN (imp_chain ORELSE not_chain))  
               ) is_iff_term
```

TAKTIKBASIERTE BEWEISFÜHRUNG IV:

METALEVEL ANALYSE ZUR INSTANTIIERUNG VON QUANTOREN

- **Matche Konklusion gegen Subterm der $\forall$ -Formel**

```
let match_subAll quantified_term conclusion =  
  letrec match_sub_aux vars allterm =  
    map (\var.assoc var (match vars allterm conclusion)) (rev vars)  
    ? let var,T,prop = dest_all allterm in match_sub_aux (var.vars) prop  
  in match_sub_aux [] quantified_term
```

- **Instantiiere $\forall$ -Quantoren mit Liste von Termen**

```
letrec allE_last_and_thin terms =  
  let t.rest = terms  
  in OnLastHyp (\h.allE h t) THEN Thin(-2) THEN allE_last_and_thin rest  
  ? Id  
letrec allEon pos terms =  
  let t.rest = terms  
  in allE pos t THEN allE_last_and_thin rest) ? Id
```

- **Instantiiere $\forall$ -Hypothese durch Subterm-Matching**

```
let InstantiateAll =  
  let InstAll_aux pos pf =  
    let sigma = match_subAll (type_of_hyp pos pf) (conclusion pf)  
    in (allEon pos (map snd sigma) THEN (OnLastHyp hypothesis)) pf  
  in TryAllHyps InstAll_aux is_all_term;;
```

TAKTIKBASIERTE BEWEISFÜHRUNG V:

SORTIERE BEWEISTECHNIKEN 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))
```

MÖGLICHE VERBESSERUNGEN DES BEWEISERS

- **Aufwendigeres Matching**

- Matching mit Teiltermen von Konjunktionen in Hypothesen
- Matching mit Teiltermen von Disjunktionen in der Konklusion
- Gleichzeitige Analyse von Quantoren in Hypothesen und Konklusion
- Behandlung verschachtelt wechselnder Quantoren

⋮

- **Zielgerichtete Verkettung**

- Auswahl relevanter Implikationen & Äquivalenzen durch Matching
- Analyse von Teilen der Prämissen von Implikationen

⋮

Beweisverfahren wird dennoch unvollständig bleiben

SEQUENZENBASIERTER BEWEISSUCHE HAT GRENZEN

- **Effiziente Beweissuche erfordert Vorausschau**

- Ziel der Suche ist Anwendbarkeit der Regel **hypothesis**
- Alle **logischen Regeln zerlegen Formeln** in geeignete Teile
 - Welche Hypothese/Konklusion soll zerlegt werden?
 - Welcher Teil einer Disjunktion soll gewählt werden? (**orR1?** **orR2?**)
 - Welcher Term soll einen Quantor instantiieren? (**exR** / **allL**)

Beweisuche sollte auf möglichen Abschluß von Beweisästen fokussieren

- **Sequenzenbeweise enthalten zu viel Redundanz**

- Jeder Knoten enthält **alle gültigen Annahmen** und die Konklusion
- Regeln **zerlegen Syntaxbaum** von Formeln und **kopieren Teilformeln** in die Nachfolgerknoten des Beweises
- Konstruktion der Beweisbäume ist **zu aufwendig** für Beweissuche

Beweisverfahren sollte direkt auf Syntaxbaum operieren

MASCHINENNAHE BEWEISTECHNIKEN SIND EFFIZIENTER

- **Ziel: einfache und schnelle Suchtechnik**

- Verzicht auf intuitives Verständnis im Beweissuchverfahren
- Stattdessen maschinennahe Charakterisierung logischer Gültigkeit
- Effiziente low-level Suchstrategien mit speziellen Datenstrukturen

- **Viele unabhängig entstandene Verfahren**

- **Davis Putnam**: Iterative Anwendung von Aufspaltung und Reduktion
 - Schnellstes Verfahren für Aussagenlogik, nicht erweiterbar
- **Resolution**: Widerlegungsverfahren für Formeln in DNF $\mapsto$ PROLOG
 - Verschmelze Klauseln mit “komplementären” Literalen
 - Komplementaritätstest erster Stufe benötigt **Unifikation**
 - Ziel ist Herleitung der leeren Klausel
- **Matrixmethoden**: Kompakte Repräsentation von Suchbäumen
 - Matrix repräsentiert Verzweigungsstruktur von Beweisbäumen
 - Teste, ob alle Pfade komplementäre Literale enthalten
 - **Geeignet zur Steuerung von Sequenzbeweisen**

MATRIXMETHODEN: VERDICHTUNG VON INFERENZEN

• Viele Sequenzenregeln haben ähnliche Struktur

orL i	$\Gamma, A \vee B, \Delta \vdash C$	$\Gamma \vdash A \wedge B$	andR
	$\Gamma, A, \Delta \vdash C$	$\Gamma \vdash A$	
	$\Gamma, B, \Delta \vdash C$	$\Gamma \vdash B$	
andL i	$\Gamma, A \wedge B, \Delta \vdash C$	$\Gamma \vdash A \vee B$	orR1
	$\Gamma, A, B, \Delta \vdash C$	$\Gamma \vdash A$	
		$\Gamma \vdash A \vee B$	orR2
		$\Gamma \vdash B$	

• Gleichartige Regeln werden zusammengefaßt

- **andL** und **orR**: Dekomposition liefert ein Teilziel **Typ α**
- **andR** und **orL**: Dekomposition verzweigt Beweis **Typ β**
- **allL** und **exR**: Dekomposition instantiiert Variable mit Term **Typ γ**
- **allR** und **exL**: Dekomposition deklariert neue Variable **Typ δ**
- Annahmen und Konklusion werden durch **Polarität** gekennzeichnet

• **Komplementarität** ersetzt **hypothesis** Regel

- Gleiche Formeln mit verschiedener Polarität schließen Beweisast ab

- **Kompakte Repräsentation von Beweisbäumen**

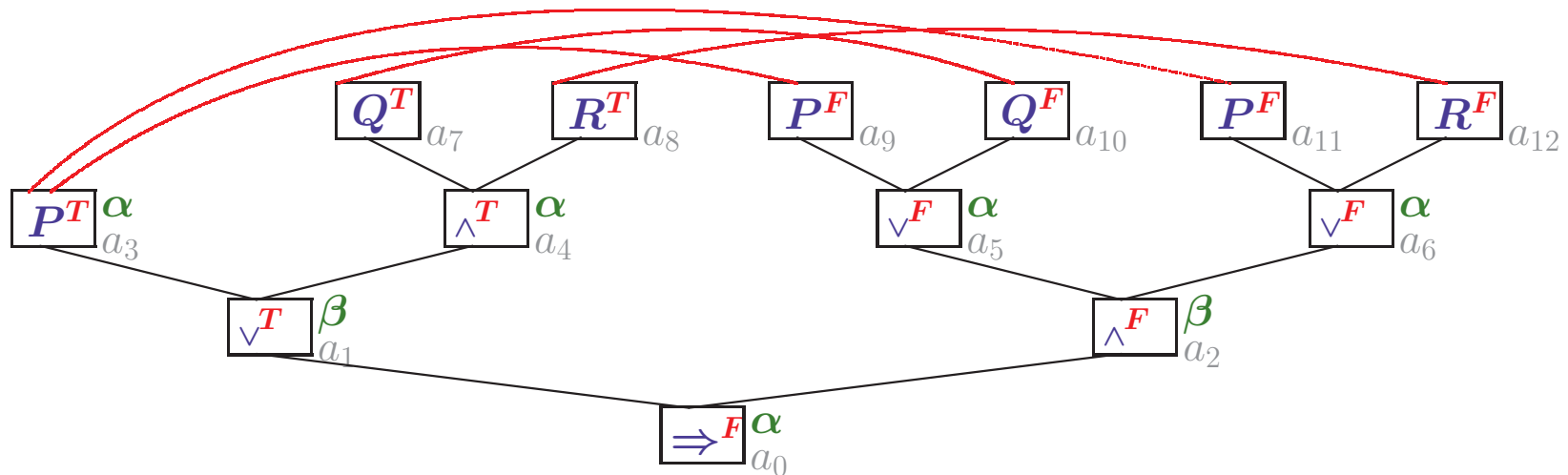
- Formelbaum enthält bereits alle Teilformeln
- Polaritäten und Formeltypen können top-down ergänzt werden
 - Beide hängen nur vom Konnektiv und bisheriger Polarität ab
- Äste eines Sequenzbeweises sind durch β -Knoten definiert
- Teilformeln mit α -Knoten als gemeinsamen Vorgänger erscheinen im gleichen Ast eines Sequenzbeweises
- `hypothesis` $\hat{=}$ komplementäre atomare Formeln in α -Beziehung

- **Einfache Beweismethode**

- Ordne **Literale** (atomare Formeln) in zweidimensionaler Matrix an
 - Nebeneinander $\hat{=}$ α -Beziehung
 - Übereinander $\hat{=}$ β -Beziehung
- Teste alle Pfade auf Existenz komplementärer Literale

ERZEUGUNG ANNOTierter FORMELBÄUME

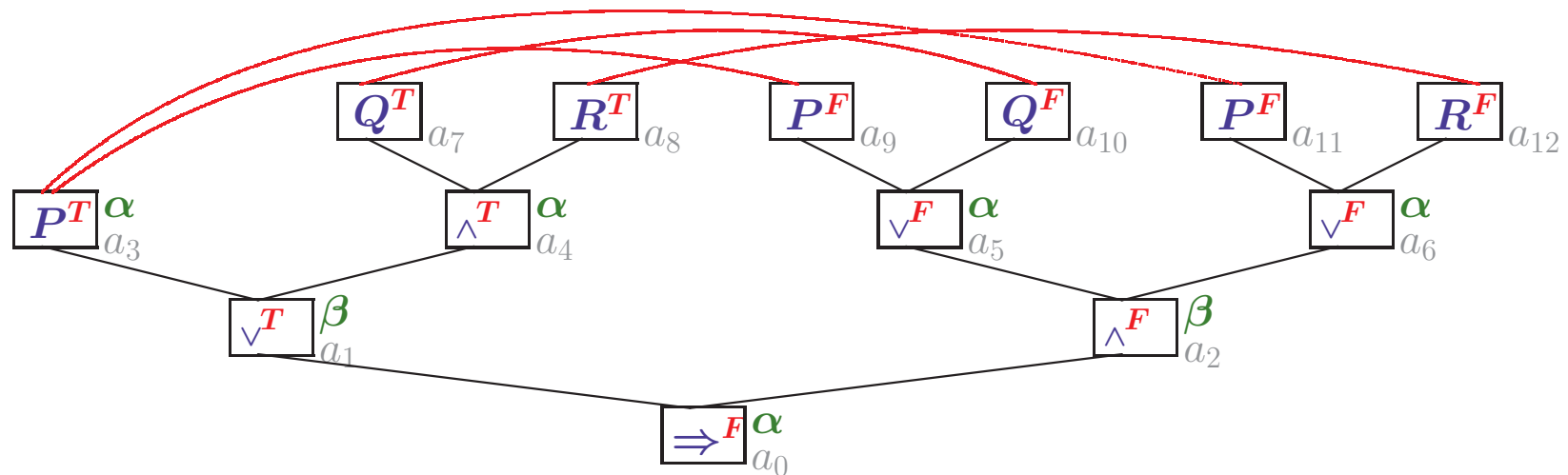
$$(P \vee (Q \wedge R)) \Rightarrow ((P \vee Q) \wedge (P \vee R))$$



Parsen der Formel erzeugt Formelbaum

- Zuweisung von Polaritäten: $T \hat{=}$ Hypothese, $F \hat{=}$ Konklusion
- Bestimmung des Typs: $\alpha \hat{=}$ linear, $\beta \hat{=}$ Verzweigung
- Zuweisung von Polaritäten an Unterformeln
- Bestimmung des Typs der Unterformeln
- Erzeuge **Konnektionen** zwischen komplementären Literalen

BEWEISFÜHRUNG: ANALYSE DER PFADE



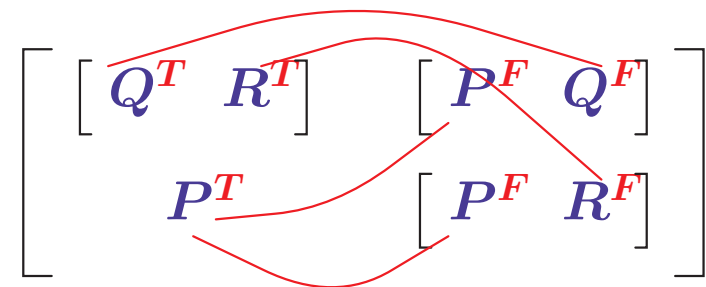
- 4 atomare Pfade $a_3a_9a_{10}$, $a_3a_{11}a_{12}$, $a_7a_8a_9a_{10}$, $a_7a_8a_{11}a_{12}$

- Alle Pfade enthalten komplementäre Literale

Formel $(P \vee (Q \wedge R)) \Rightarrow ((P \vee Q) \wedge (P \vee R))$ ist gültig

- Zweidimensionale Repräsentation

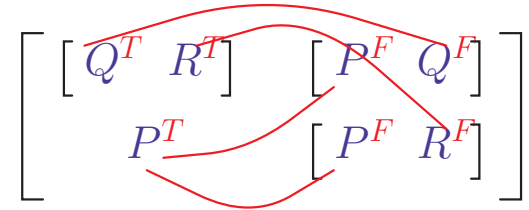
Mit Konnektionen



MATRIXMETHODEN: ZUSÄTZLICHE ASPEKTE

- **Pfadüberprüfung folgt Konnektionen**

- Frühzeitiges Abschneiden zu prüfender Pfade
- Verringert Anzahl notwendiger Überprüfungen

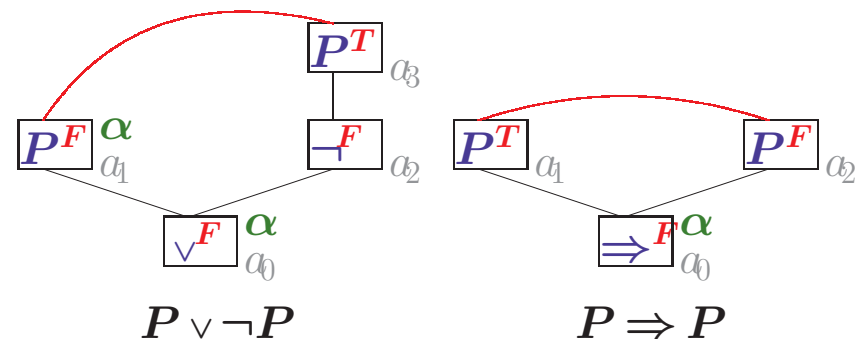


- **Logik erster Stufe braucht Term-Unifikation**

- Variablen von γ -Knoten können instantiiert werden
- Variablen von δ -Knoten gelten als Konstante
- Standard-Algorithmen von Robinson oder Martelli-Montanari

- **Konstruktive Logik braucht zusätzliche Methoden**

- Unterscheide $P \vee \neg P$ von $P \Rightarrow P$
- Regeln für $\Rightarrow$, $\neg$, $\forall$ sind irreversibel
- Bestimme Reihenfolge der $\Rightarrow$, $\neg$, $\forall$
- Hilfsmittel: Präfix(String)-Unifikation



Thema der Vorlesung “Inferenzmethoden”

INTEGRATION VON MATRIXMETHODEN IN Nuprl

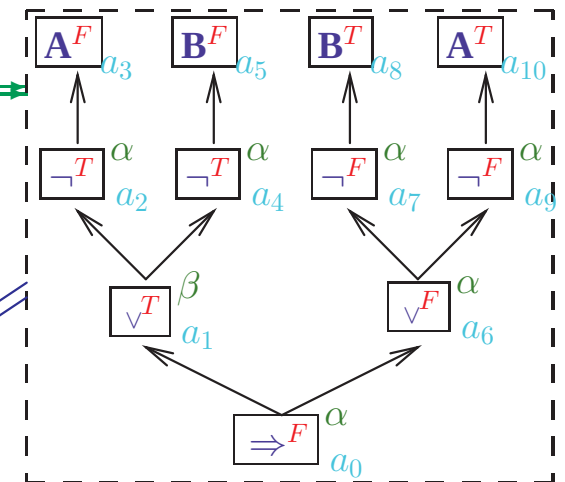
Formel

$$\neg A \vee \neg B \Rightarrow \neg B \vee \neg A$$

Annotierung

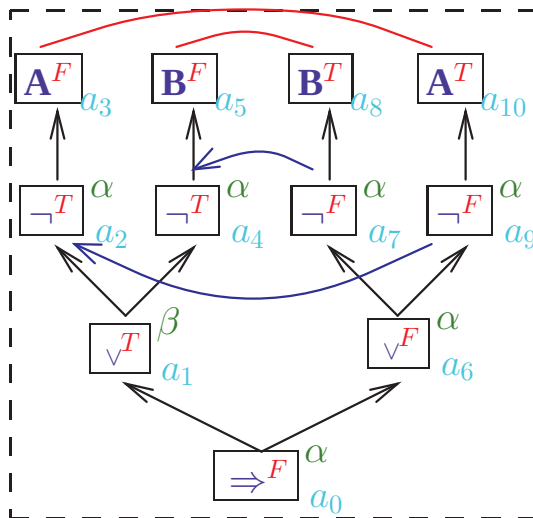
Typen, Polaritäten, Präfixe

Annotierter Formelbaum



Matrixbeweiser

Pfadchecking + Unifikation
Substitutionen induzieren Ordnung $\triangleleft$



Reduktionsordnung $\triangleleft$

Beweistransformation

Traversierung von $\triangleleft$
Vielfach $\rightarrow$ Einzel-Konklusion

$$\frac{\frac{\frac{\overline{A \vdash A} \text{ hyp}}{\neg A, A \vdash \neg I} \neg I}{\neg A \vdash \neg B, \neg A} \neg r \quad \frac{\frac{\frac{\overline{B \vdash B} \text{ hyp}}{\neg B, B \vdash \neg I} \neg I}{\neg B \vdash \neg B, \neg A} \neg r}{\neg A \vee \neg B \vdash \neg B, \neg A} \vee l}{\neg A \vee \neg B \vdash \neg B \vee \neg A} \vee r}{\vdash \neg A \vee \neg B \Rightarrow \neg B \vee \neg A} \Rightarrow r$$

Sequenzenbeweis

JProver: DER AUTOMATISCHE BEWEISER

• Beweissuche

- Matrixbeweiser für Logik erster Stufe (Kreitz & Otten 1999)
(Konnektionsgetriebene Pfadüberprüfung + Termunifikation)
- Zusätzliche Stringunifikation für konstruktive Beweise (Otten & Kreitz 1996)
- Substitutionen und Formelbaum induzieren Reduktionsordnung

• Beweistransformation

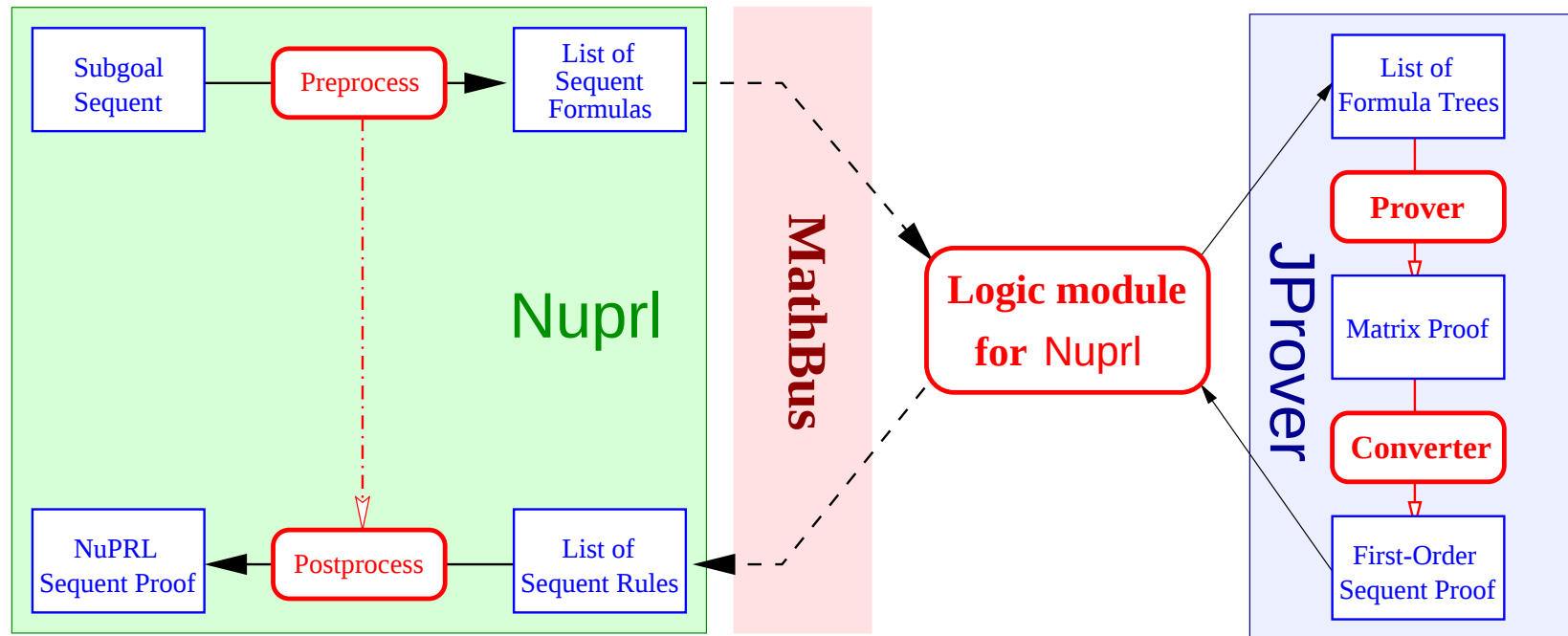
- Extrahiert Sequenzenbeweis aus Matrixbeweis (Kreitz & Schmitt 2000)
- Traversiert Reduktionsordnung ohne Suche (Schmitt 2000)
- Erzeugt Sequenzenkalküle mit mehreren/ einer Konklusion (Egly & Schmitt 1999)

• Implementierung

(Schmitt et. al 2001)

- Stand-alone Beweiser in OCaml
- Einbettung in MetaPRL-Umgebung liefert Basisfunktionalitäten
(Datentypen für Terme, Termunifikation, Modul System)

JProver: ANBINDUNG AN Nuprl



- Präprozessor für Nuprl Sequenzen und semantische Unterschiede
- Kommunikation von Termen im MathBus Format über INET socket
- JLogic Modul: extrahiert semantische Information aus Termen und konvertiert Sequenzenbeweis in das Format von Nuprl
- Postprozessor baut Nuprl Beweisbaum für Ausgangssequenz

LOGISCHE INTEGRATION VON JProver IN Nuprl

- **Logikmodul: Komponenten**

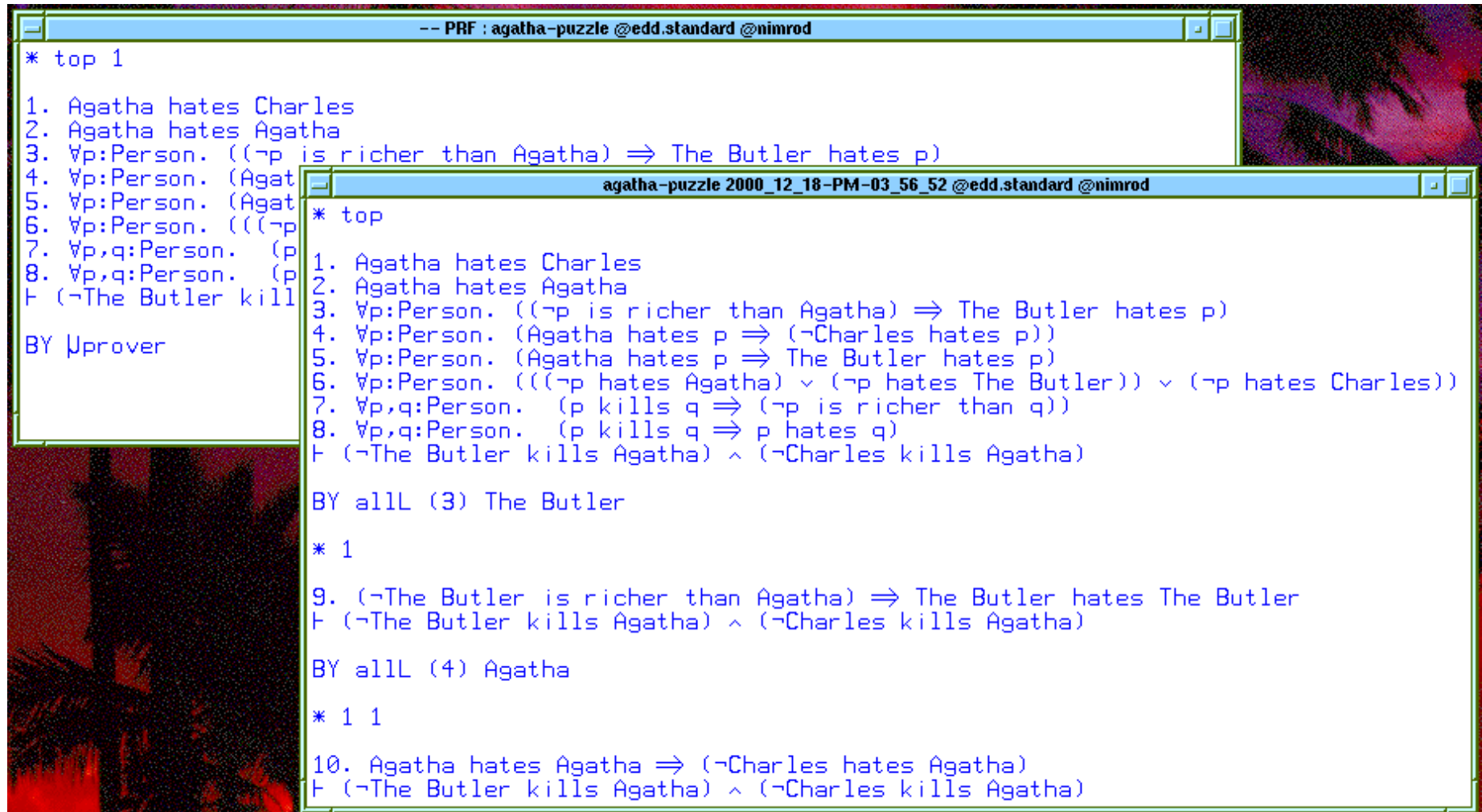
- OCaml code für Kommunikation mit interaktivem Beweiser
- JLogic Modul zur Darstellung Nuprl Logik

- **Das JLogic Modul**

- Beschreibt Terme, welche Nuprl's logische Konnektive implementieren
- Operationen für Zugriff auf Teilterme
- Decodiert Sequenzen, die in MathBus Format ankommen
- Codiert JProver's Sequenzenbeweis ins MathBus Format

```
module Nuprl_JLogic =  
  struct  
    let is_all_term = nuprl_is_all_term  
    let dest_all = nuprl_dest_all  
    let is_exists_term = nuprl_is_exists_term  
    let dest_exists = nuprl_dest_exists  
    let is_and_term = nuprl_is_and_term  
    let dest_and = nuprl_dest_and  
    let is_or_term = nuprl_is_or_term  
    let dest_or = nuprl_dest_or  
    let is_implies_term = nuprl_is_implies_term  
    let dest_implies = nuprl_dest_implies  
    let is_not_term = nuprl_is_not_term  
    let dest_not = nuprl_dest_not  
  
    type inference = '(string*term*term) list  
    let empty_inf = []  
    let append_inf inf t1 t2 r =  
      ((Jall.ruletable r), t1, t2) :: inf  
    end
```

DEMO-BEISPIEL: “AGATHA MURDER PUZZLE”



```

-- PRF : agatha-puzzle @edd.standard @nimrod

* top 1
1. Agatha hates Charles
2. Agatha hates Agatha
3.  $\forall p:\text{Person. } ((\neg p \text{ is richer than Agatha}) \Rightarrow \text{The Butler hates } p)$ 
4.  $\forall p:\text{Person. } (\text{Agatha hates } p \Rightarrow (\neg \text{Charles hates } p))$ 
5.  $\forall p:\text{Person. } (\text{Agatha hates } p \Rightarrow \text{The Butler hates } p)$ 
6.  $\forall p:\text{Person. } (((\neg p \text{ hates Agatha}) \vee (\neg p \text{ hates The Butler})) \vee (\neg p \text{ hates Charles}))$ 
7.  $\forall p,q:\text{Person. } (p \text{ kills } q \Rightarrow (\neg p \text{ is richer than } q))$ 
8.  $\forall p,q:\text{Person. } (p \text{ kills } q \Rightarrow p \text{ hates } q)$ 
 $\vdash (\neg \text{The Butler kills Agatha}) \wedge (\neg \text{Charles kills Agatha})$ 

BY  $\uparrow$ prover

agatha-puzzle 2000_12_18-PM-03_56_52 @edd.standard @nimrod

* top
1. Agatha hates Charles
2. Agatha hates Agatha
3.  $\forall p:\text{Person. } ((\neg p \text{ is richer than Agatha}) \Rightarrow \text{The Butler hates } p)$ 
4.  $\forall p:\text{Person. } (\text{Agatha hates } p \Rightarrow (\neg \text{Charles hates } p))$ 
5.  $\forall p:\text{Person. } (\text{Agatha hates } p \Rightarrow \text{The Butler hates } p)$ 
6.  $\forall p:\text{Person. } (((\neg p \text{ hates Agatha}) \vee (\neg p \text{ hates The Butler})) \vee (\neg p \text{ hates Charles}))$ 
7.  $\forall p,q:\text{Person. } (p \text{ kills } q \Rightarrow (\neg p \text{ is richer than } q))$ 
8.  $\forall p,q:\text{Person. } (p \text{ kills } q \Rightarrow p \text{ hates } q)$ 
 $\vdash (\neg \text{The Butler kills Agatha}) \wedge (\neg \text{Charles kills Agatha})$ 

BY allL (3) The Butler

* 1
9.  $(\neg \text{The Butler is richer than Agatha}) \Rightarrow \text{The Butler hates The Butler}$ 
 $\vdash (\neg \text{The Butler kills Agatha}) \wedge (\neg \text{Charles kills Agatha})$ 

BY allL (4) Agatha

* 1 1
10.  $\text{Agatha hates Agatha} \Rightarrow (\neg \text{Charles hates Agatha})$ 
 $\vdash (\neg \text{The Butler kills Agatha}) \wedge (\neg \text{Charles kills Agatha})$ 

```

- **Hybride Beweiser verbinden verschiedene Kalküle**
 - Ausdruckskraft interaktiver Beweisassistenten für komplexe Beweise
 - + Effiziente Beweissuche für Teilprobleme erster Stufe
- **Typinformation in Grenzen verwendbar**
 - Codiere als Prädikate ohne Querbezüge
- **Erweiterbar jenseits von Logik erster Stufe**
 - Behandlung spezieller Theorien / Gleichheit
 - Induktionsbehandlung durch Integration von Rewriting
 - Integration von Entscheidungsprozeduren in den Unifikationsprozess
 - Effizientere Implementierung durch bessere Datenstrukturen

Thema für forschungsbezogene Studien/Diplomarbeiten