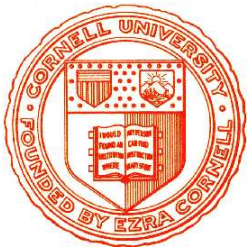
Automatisierte Logik und Programmierung

Einheit 15



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 für nächste Schritte

  Sehr sicher und kontrollierbar, aber mühsam
- **Taktikbasierte Beweissuche**
  - Taktik sucht nach anwendbaren Regeln und analysiert Konklusion und Hypothesen zur Parameterbestimmung

  Für Benutzer leicht programmierbar, praktisch sehr nützlich, aber unvollständig
- **Vollautomatische Beweisverfahren** (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
    aber nur wenige Verfahren erweiterbar auf konstruktive Logik

  Gründliche theoretische Vorarbeiten führen zu sehr hoher Effizienz
  Integration erfordert Konsistenzcheck oder Erzeugung von Beweistermen

Entwicklung eines taktikbasierten Beweisers: Schrittweise Steigerung des Automatisierungsgrades

1. **Interaktion: Verwende Regeln der Logik erster Stufe**
   – Verstecke zugrundeliegende Basiskonstrukte der Typentheorie
2. **Zielgerichtete Auswahl 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 (partielles) Matching
5. **Sortiere Beweistechniken nach Aufwand**
   – Beschleunigt Beweissuche, wenn mehrere Möglichkeiten bestehen

Einfache Techniken liefern erstaunlich gute Ergebnisse

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**
  - Tactical `TryOn` wendet Taktik auf Klausel an, wenn Bedingung erfüllt

```
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** (Kosmetik ist wichtig)

  `andI`, `orI1`, `orI2`, `impI`, ..., `andE`*i*, `orE`*i*, `impE`*i*, ...,

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

Taktikbasierte Beweisführung II: Automatische 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 aus Kontext**

```
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 IIa

Primitiver Beweiser auf Basis einfacher Ideen

```
let prim_prover = Repeat
                  (       Hypothesis
                   ORELSE contradiction
                   ORELSE conjunctionE
                   ORELSE existentialE
                   ORELSE nondangerousI
                   ORELSE disjunctionE
                   )
```

– Erledigt einfache Teilaufgaben in Beweisen (Zerlegen, Strukturieren, . . .)

– Ermöglicht Benutzern, sich auf die eigentlichen Probleme zu konzentrieren

Noch kein “Beweiser”, da viele leichte Aufgaben unerledigt bleiben

Taktikbasierte Beweisführung III: Verkettung von Implikationen & Äquivalenzen

- **Chaining: Wiederholte Anwendung einer Taktik**
  - Iteriere Taktik t auf ausgewählten Hypothesen
  - Letzter Schritt ist Anwendung einer Basistaktik
  - Anzahl der Wiederholungen muß begrenzt werden

```
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 für $\Rightarrow, \neg, \Leftrightarrow$**

```
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 ∀-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 ∀-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 ∀-Hypothese durch Subterm-Matching**

(vgl. Implementierung von `InstantiateEx` *in §13, Folie 21)*

```
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

Implementiere Leitlinien für Beweise aus §3 (Folie 42)

```
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, wenn Suche auf naheliegende Methoden begrenzt bleibt

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 fokussiert werden

- **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

  Beweissuchverfahren 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 — nicht konstruktiv
  - **Matrixmethoden**: Kompakte Repräsentation von Suchbäumen
    - Matrix repräsentiert Verzweigungsstruktur von Beweisbäumen
    - Teste, ob alle Pfade komplementäre Literale enthalten

  Gut geeignet zur Steuerung von Sequenzenbeweisen

Matrixmethoden: Verdichtung von Inferenzen

- **Viele Sequenzenregeln haben ähnliche Struktur**

$$\texttt{orL}\ i \quad \frac{H, \boldsymbol{A} \vee \boldsymbol{B}, H' \vdash C}{H, \boldsymbol{A}, H' \vdash C \quad H, \boldsymbol{B}, H' \vdash C} \qquad \frac{H \vdash \boldsymbol{A} \wedge \boldsymbol{B}}{H \vdash \boldsymbol{A} \quad H \vdash \boldsymbol{B}}\ \texttt{andR}$$

$$\texttt{andL}\ i \quad \frac{H, \boldsymbol{A} \wedge \boldsymbol{B}, H' \vdash C}{H, \boldsymbol{A}, \boldsymbol{B}, H' \vdash C} \qquad \frac{H \vdash \boldsymbol{A} \vee \boldsymbol{B}}{H \vdash \boldsymbol{A}}\ \texttt{orR1}$$

$$\frac{H \vdash \boldsymbol{A} \vee \boldsymbol{B}}{H \vdash \boldsymbol{B}}\ \texttt{orR2}$$

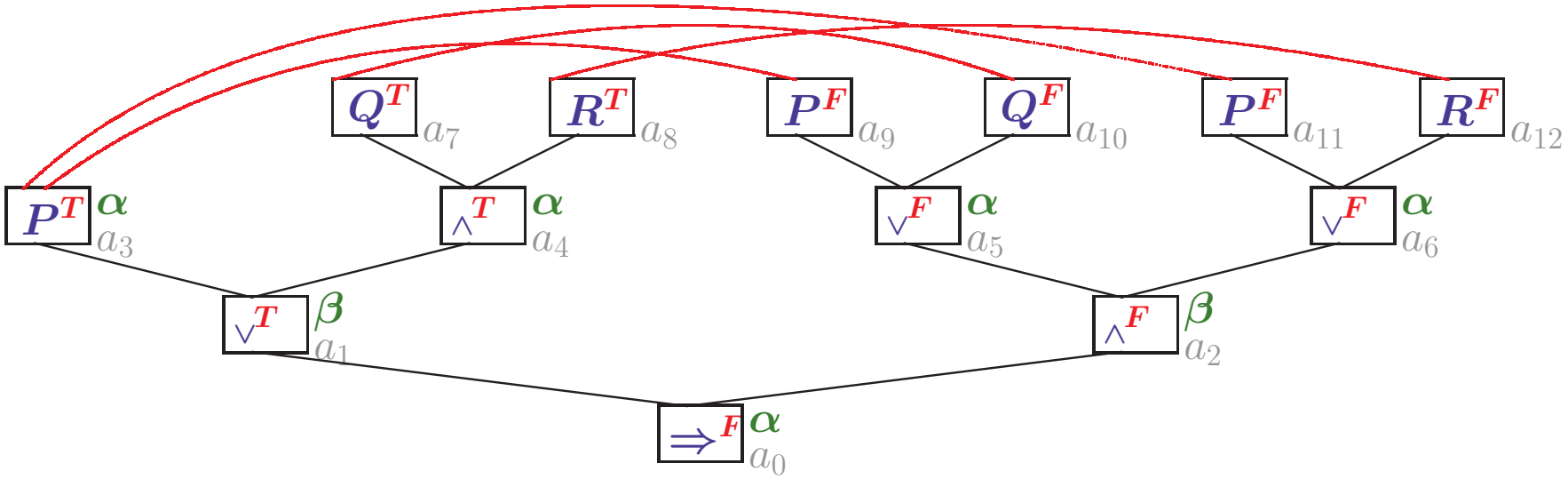
- **Gleichartige Regeln werden zusammengefaßt**
  - `andL` und `orR`: Dekomposition liefert ein Teilziel — **Typ α**
  - `andR` und `orL`: Dekomposition verzweigt Beweis — **Typ β**
  - `allL` und `exR`: Dekomposition instantiiert Variable mit Term — **Typ γ**
  - `allR` und `exL`: Dekomposition deklariert neue Variable — **Typ δ**
  - Annahmen und Konklusion werden durch **Polarität** gekennzeichnet
- **Komplementaritätskriterium ersetzt `hypothesis` Regel**
  - Gleiche Formeln mit verschiedener Polarität schließen Beweisast ab

Matrixmethoden: Verdichtung der Beweise

- **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 Sequenzenbeweises sind durch β-Knoten definiert
  - Teilformeln mit α-Knoten als gemeinsamen Vorgänger erscheinen im gleichen Ast eines Sequenzenbeweises
  - `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

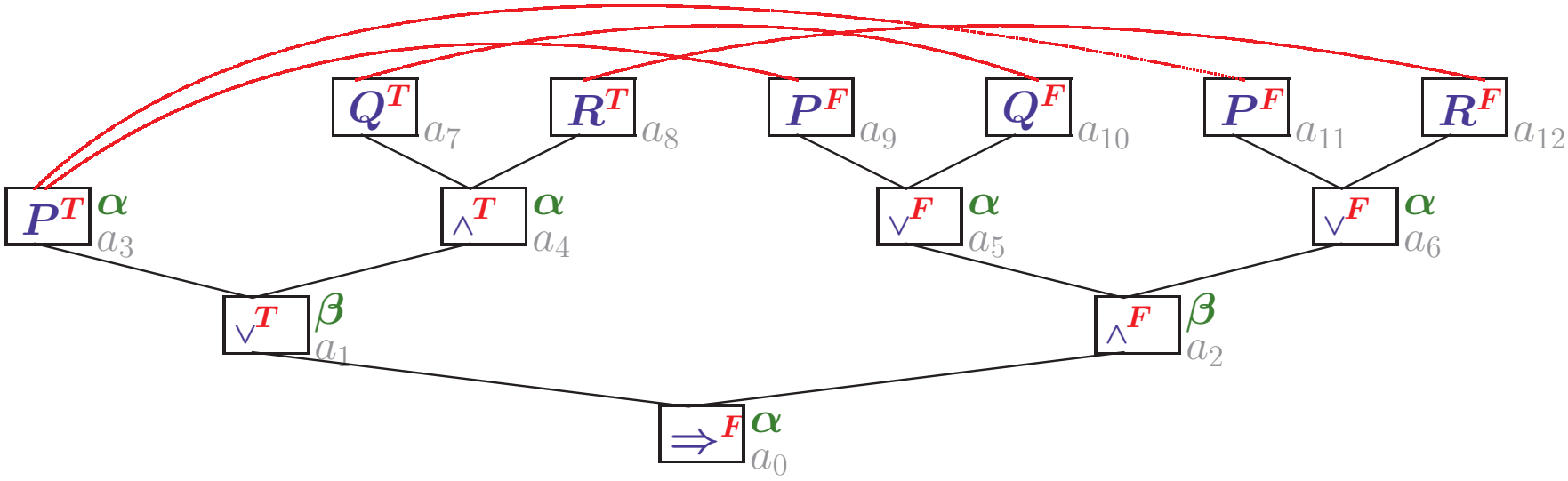
$$(P \vee (Q \wedge R)) \;\Rightarrow\; ((P \vee Q) \wedge (P \vee R))$$



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

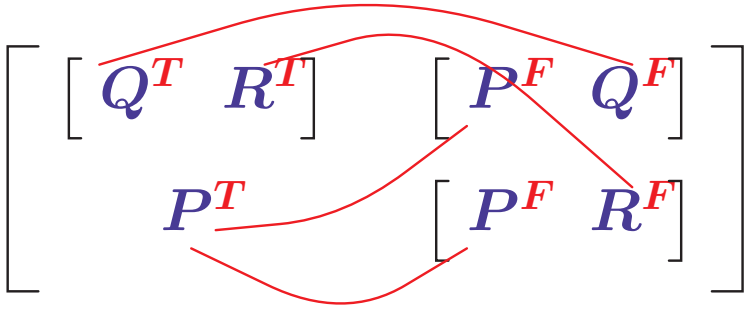
Beweisführung: Analyse der Pfade



- **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**

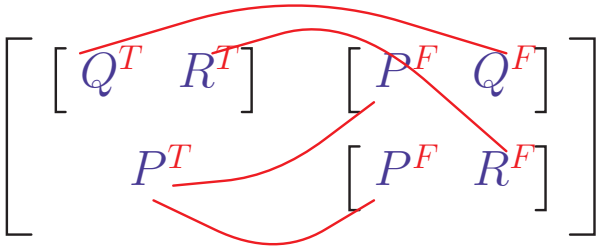
  Formel $(P \vee (Q \wedge R)) \Rightarrow ((P \vee Q) \wedge (P \vee R))$ **ist gültig**
- **Zweidimensionale Repräsentation**

  Mit Konnektionen



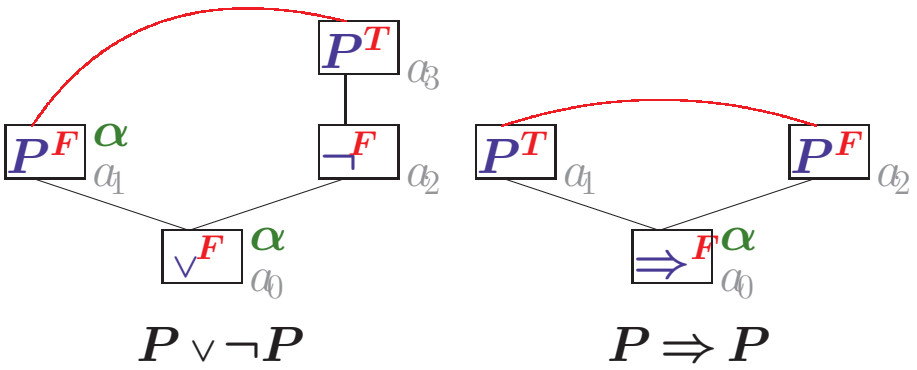
Matrixmethoden: zusätzliche Aspekte

- **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

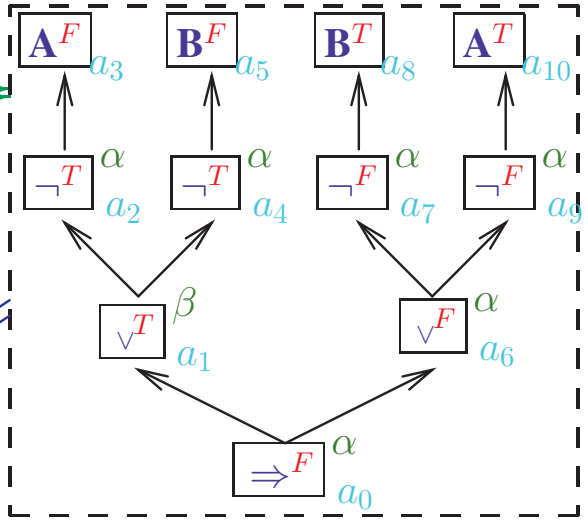
Formel

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

Annotierung

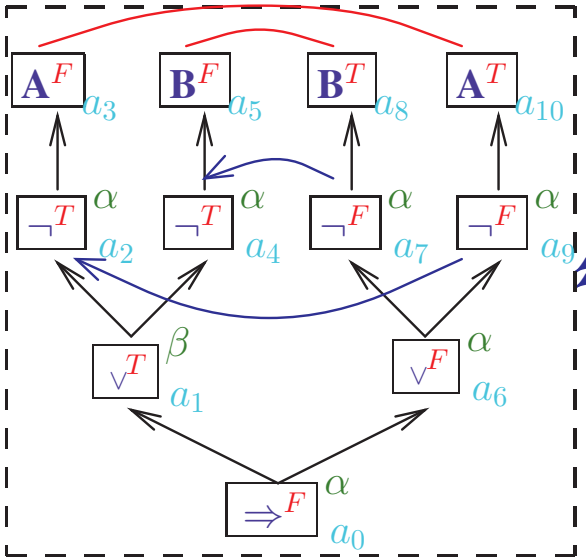
Typen, Polaritäten, Präfixe

Annotierter Formelbaum



Matrixbeweiser

Pfadchecking + Unifikation
Substitutionen induzieren Ordnung $\lhd$



Reduktionsordnung $\lhd$

Beweistransformation

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

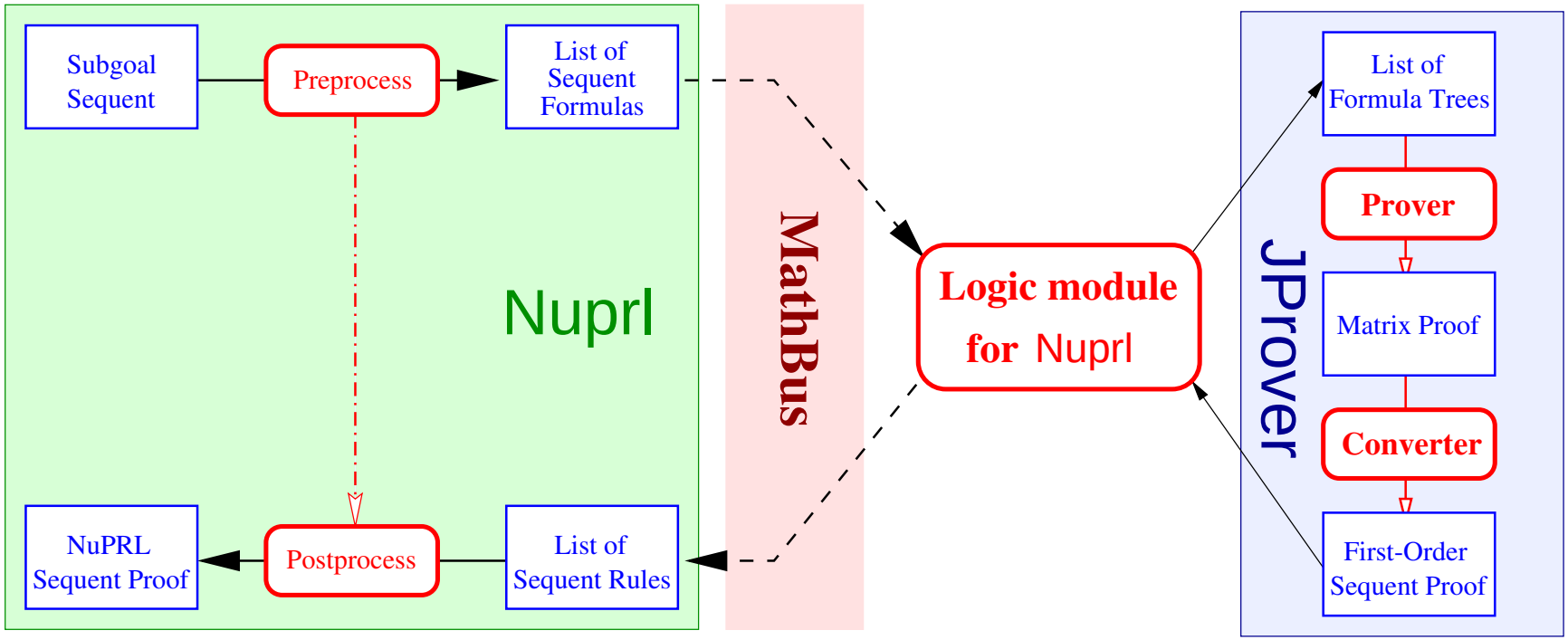
$$\dfrac{\dfrac{\dfrac{\dfrac{}{A \vdash A}\,\text{hyp}}{\neg A, A \vdash}\,\neg\text{l}}{\neg A \vdash \neg B, \neg A}\,\neg\text{r} \qquad \dfrac{\dfrac{\dfrac{}{B \vdash B}\,\text{hyp}}{\neg B, B \vdash}\,\neg\text{l}}{\neg B \vdash \neg B, \neg A}\,\neg\text{r}}{\dfrac{\neg A \vee \neg B \vdash \neg B, \neg A}{\dfrac{\neg A \vee \neg B \vdash \neg B \vee \neg A}{\vdash \neg A \vee \neg B \Rightarrow \neg B \vee \neg A}\,\Rightarrow\text{r}}\,\vee\text{r}}\,\vee\text{l}$$

Sequenzenbeweis

JProver: der automatische Beweiser

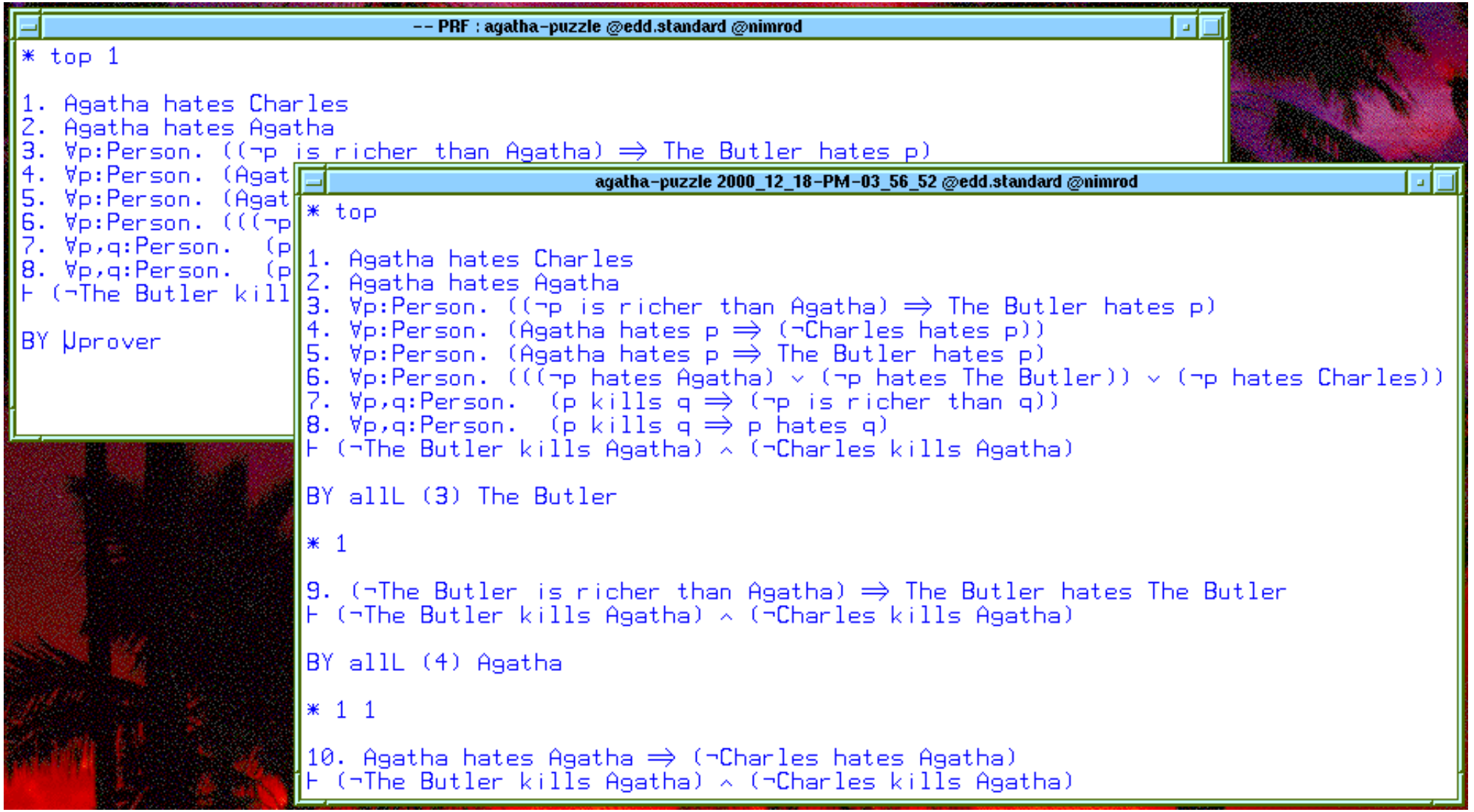
- **Beweissuche**
  - Matrixbeweiser für Logik erster Stufe (Kreitz & Otten 1999)
    (Konnektionsgetriebene Pfadüberprüfung + Termunifikation)
  - Zusätzliche Stringunifikation für konstruktive Beweise (Otten & Kreitz 1996)
  - Substitutionen und Formelbaum induzieren Reduktionsordnung
- **Beweistransformation**
  - 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)
  - Keine Optimierung der Geschwindigkeit auf Basis neuester Techniken
    Geschwindigkeit für Anwendungen ausreichend, Optimierungen nicht transformierbar

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

Demo-Beispiel: "Agatha Murder Puzzle"



JProver: Erkenntnisse

- **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-/Abschlussarbeiten

Beweisautomatisierung für logische Aussagen

- **Viele verschiedene Techniken kombinierbar**
  - Benutztersteuerung, Taktiken, vollständige Automatisierung
  - Im Endeffekt muß der Nutzer auswählen, was angebracht ist
- **Tools sind kein Ersatz für intelligente Nutzer**
  - Ziel ist Unterstützung bei "trivialen" Aufgaben
  - Nutzer kann sich auf die interessanten Ideen konzentrieren
- **Wir sind erst am Anfang**
  - Einfache Techniken machen schon vieles möglich
  - Viele Wege, Standardschlüsse zu automatisieren, sind noch unerforscht
  - Systementwickler können nur die allgemeinen Techniken bereitstellen

    Wo es hingeht, hängt vor allem an den Anwendern der Beweissysteme