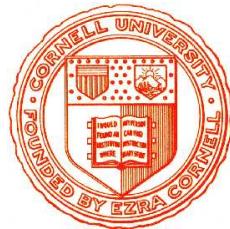


# Automatisierte Logik und Programmierung II

## Teil IV

### Beweisautomatisierung



1. Taktische Beweisführung
2. Entscheidungsprozeduren
3. Integration externer Systeme

# Automatisierte Logik und Programmierung



## Lektion 13

### Taktiken



## Benutzerdefinierbare Beweisstrategien

1. Grundkonzept und Arbeitsweise
2. Programmierung von Taktiken
3. Rewriting als Taktiken
4. Erfahrungen im praktischen Umgang

- **Refiner akzeptiert Metalevel-Programme**

- Programme enthalten Aufrufe von Beweisregeln, die Refiner ausführt
- Programme analysieren Beweiskomponenten, um Regeln zu bestimmen
- Programme dürfen beliebig komplex sein

- **Flexibel: Benutzer programmiert Beweisstrategien**

- Vorausplanung von Beweisen
- Suche nach Beweisen
- Strukturierung von Beweisen (Verstecken überflüssiger Details)
- Abgeleitete Inferenzregeln für benutzerdefinierte Theorien
- Austesten komplexer Beweis-/Syntheseverfahren in sicherer Umgebung

- **Sicher: Taktiken sind immer korrekt**

- Taktiken können erfolglos sein oder nicht terminieren
- *Das Resultat einer Taktik-Anwendung ist immer ein gültiger Beweis des zugrundeliegenden logischen Kalküls*

- Benutzer gibt Taktik als Inferenzschritt
- Beweiseditor ergänzt notwendige Daten
  - Aktuelle Beweissequenz wird zum Beweisziel
  - Beweiseditor übergibt Beweisziel und Taktik an Refiner
  - Beweisbaum (falls vorhanden) unterhalb des Knotens wird ignoriert
- Refiner wendet Taktik auf Beweisziel an
  - Ergibt ungelöste Teilziele und Validierung
  - Anwendung der Validierung auf Teilziele erzeugt Beweisbaum
  - Fehlermeldung, falls Taktik nicht anwendbar
- Library speichert Beweisbaum
  - Rechtfertigung für den durchgeführten Inferenzschritt
  - Beweiseditor zeigt Taktik und offene Teilziele
  - Beweisbaum wird nur auf expliziten Wunsch sichtbar gemacht

**Taktik wirkt wie abgeleitete Inferenzregel**

- **Benutzer gibt Taktik als Kommando**
- **Beweiseditor ergänzt notwendige Daten**
  - Gesamter aktueller Beweisbaum wird zum Beweisziel
  - Beweiseditor über gibt Beweisziel und Taktik an Refiner
- **Refiner wendet Taktik auf Beweisziel an**
  - Ergibt ungelöste Teilziele und Validierung
  - Anwendung der Validierung auf Teilziele erzeugt Beweisbaum
  - Beweisziel bleibt unverändert, falls Taktik nicht anwendbar
- **Library speichert Beweisbaum**
  - Rechtfertigung für die durchgeführten Inferenzschritte
  - Beweiseditor zeigt ausgeführte Einzelschritte und offene Teilziele
  - Taktikname wird nicht gespeichert
- **Ziel: Modifikation existierender Beweise**
  - Kopieren, Expandieren, Komprimieren, Analogie, ...
  - **In Nuprl 5 nur als vordefinierte Operationen des Beweiseditors**

# VERFEINERUNGS VS. TRANSFORMATIONSTAKTIK

## Effekte der Anwendung der Taktik Cases

THM cases

# top  
|- T

BY Cases [A;B]

# 1  
|- A ∨ B

# 2  
1. A  
|- T

# 3  
1. B  
|- T

THM cases

# top  
|- T

BY cut 1 A ∨ B

# 1  
|- A ∨ B

# 2  
1. A ∨ B  
|- T

BY orE 1

# 2 1  
1. A  
|- T

# 2 2  
1. B  
|- T

- **Explizite Umwandlung von Regeln mittels `refine`**

- Nur für ausfallenere Regeln oder stärkere Kontrolle erforderlich
  - Fast alle Regeln sind durch vordefinierte Taktiken abgedeckt

- **Vordefinierte Standardtaktiken**

- Gespeichert in Code-Objekten der Library

- **Komposition existierender Taktiken durch `Tacticals`**

- Operationen, die Taktiken zu neuen Taktiken zusammensetzen
  - Viele vordefinierte `Tacticals` in der Library

- **Metalevel-Steuerung von Taktikanwendungen**

- ML-Programme analysieren Beweisziel und Kontext

- Umwandlung der Regel `hypothesis`

```
let get_pos_hyp_num i proof =
  if i<0 then length (hypotheses proof)+1+i
  else i
;;
let NthHyp i proof =
  let i' = get_pos_hyp_num i proof
  in
    Refine 'hypothesis' [mk_int_arg i'] proof
;;
```

- Ausdünnen überflüssiger Hypothesen

```
let Thin i proof =
  let i' = get_pos_hyp_num i p roof
  in
    if i=0 then failwith 'Thin: cannot thin conclusion'
    else Refine 'thin' [mk_int_arg i'] proof
;;
```

## ● Vereinheitlichung der Dekompositionsregeln

- **D *i***: Top-Level Dekomposition einer Hypothese oder der Konklusion
- **EqD *i*, MemD *i***: Dekomposition einer Gleichheit bzw. Typzugehörigkeit
- **EqTypeD *i*, MemTypeD *i***: Dekomposition des Typs einer Gleichheit

## ● Strukturelle Regeln

- **Hypothesis, Declaration**: Konklusion ist in Hypothesen enthalten
- **Assert *t***: Einführen von Zwischenbehauptungen (Schnittregel)

## ● Berechnungsregeln

- **Reduce *i***: Top-Level Reduktion einer Hypothese oder der Konklusion

## ● Einfügen von Steuerungsparametern in Taktiken

- Name ***x*** einer neuen Variablen **New *[x]* (D 0)**
- Typ ***T*** eines Teilterms im Beweisziel **With *[x:S→T]* (MemD 0)**
- Term, ***s***, der für eine Variable einzusetzen ist **With *[s]* (D 0)**
- Level ***j*** eines Universums **At *[j]* (D 0)**
- Abhängigkeit eines Terms ***C*** von Variable ***x*** **Using *[z,C]* (D 0)**

- **Autotaktik** (triviale Schlüsse) Auto
- **Regeln der Logik erster Stufe:** andR, orR, ..., exL *i*
- **Datentypspezifische Induktionen**
  - Standardinduktion: NatInd *i*, NSubsetInd *i*
  - IntInd *i*, ListInd *i*
  - Vollständige Induktion auf natürlichen Zahlen: CompNatInd *i*
- **Fallanalysen**
  - Analyse Boolescher Variablen: BoolCases *i*
  - Allgemeine Fallunterscheidung: Cases  $[t_1; \dots; t_n]$
  - Analyse entscheidbarer Aussagen (Fälle  $P$  und  $\neg P$ ) Decide  $P$
- **Chaining (Verkettung von Argumenten)**
  - Instantiierung quantifizierter Aussagen: InstHyp  $[t_1; \dots; t_n]$  *i*
  - Vorwärtsverkettung von Hypothesen: FHyp *i*  $[h_1; \dots; h_n]$ ,
  - Rückwärtsverkettung im Beweisknoten: BHyp *i*,
  - " " durch Hypothesen & Lemmata: Backchain  $bc\_names$

## ● Operationen einer kommandoartigen Taktiksprache

- Setze Taktiken zu neuen Taktiken zusammen
- Funktionen höherer Ordnung, oft in Infixschreibweise

## ● Die wichtigsten vordefinierten **Tacticals**

$t_1 \text{ THEN } t_2$ :	Wende $t_2$ auf alle von $t_1$ erzeugten Teilziele an
$t \text{ THENL } [t_1; \dots; t_n]$ :	Wende $t_i$ auf das $i$ -te von $t$ erzeugte Teilziel an
$t_1 \text{ ORELSE } t_2$ :	Wende $t_1$ an. Falls dies fehlschlägt, wende $t_2$ an
$\text{Repeat } t$ :	Wiederhole Taktik $t$ bis sie fehlschlägt

## ● Häufig benutzte

$t_1 \text{ THENA } t_2$ :	Wende $t_2$ auf alle von $t_1$ erzeugten Hilfsziele an
$t_1 \text{ THENW } t_2$ :	Wende $t_2$ auf alle von $t_1$ erzeugten wf-ziele an
$\text{Try } t$ :	Wende $t$ an, Bei Fehlschlag lasse den Beweis unverändert
$\text{Complete } t$ :	Wende $t$ nur an, wenn der Beweis vollständig wird
$\text{Progress } t$ :	Wende $t$ nur an, wenn ein “Fortschritt” erzielt wird
$\text{RepeatFor } i \ t$ :	Wiederhole $t$ genau $i$ mal
$\text{AllHyps } t$ :	Wende $t$ auf alle möglichen Hypothesen an
$\text{OnSomHyp } t$ :	Wende $t$ auf die erstmögliche Hypothese an

# PROGRAMMIERUNG EINER TAKTIK FÜR TRIVIALE SCHLÜSSE

```
let Equality = Refine `equality` [] ;;
and Arith      = Refine `arith`
  [mk_term_arg
   (mk_universe_term
    (mk_level_exp ['i',0])) ]
;;
let Immediate =
  Declaration
  ORELSE Hypothesis
  ORELSE OnSomeHyp falseL
  ORELSE Contradiction
  ORELSE Equality
  ORELSE Arith
  ORELSE D 0   ORELSE OnSomeHyp D
  ORELSE EqD 0 ORELSE OnSomeHyp EqD
;;

```

Ausschließlich bekannte Taktiken und Tactics

# IMPLEMENTIERUNG VON TACTICALS

```
ml_curried_infix 'THENL' ;;
ml_curried_infix 'ORELSE' ;;

let $THENL (tac: tactic) (tac_list : tactic list) (pf:proof) =
  let subgoals, val = tac pf
  in
    if not length tac_list = length subgoals then fail
    else let subgoalLists, valList = map_apply tac_list subgoals
         in
           (flatten subgoalLists),
           \prfs. val ( (mapshape (map length subgoalLists) valList) prfs)
;;
let $ORELSE (t1:tactic) (t2:tactic) pf =  t1 pf ? t2 pf ;;

let Complete (tac:tactic) (pf:proof) = let subgoals, val = tac pf
                                         in
                                           if null subgoals
                                           then subgoals, val
                                           else fail
;;

```

# PROGRAMMIERUNG DER TAKTIK Hypothesis

## Anwendung von NthHyp auf alle Hypothesen

```
let OnHyp i (T: int->tactic) p =
  T (get_pos_hyp_num i p) p
;;

let OnSomeHyp T p =
  letrec Aux i p' =
    if i = 0 then failwith 'OnSomeHyp'
    else (OnHyp i T ORELSE Aux (i-1)) p'
  in
    Aux (length (hypotheses proof)) p
;;
let Hypothesis = OnSomeHyp NthHyp ;;
```

## Finde zwei einander widersprechende Hypothesen

```
let Contradiction proof =
  let facts      = map (fst o snd o dest_declaration)(hypotheses proof) in
  let facts_and_nums = (map2 o pair) facts (upto 1 (length facts))      in
  let negative_hyps = filter (\t,i. is_not_term t) facts_and_nums      in
  let positive_hyp, negative_hyp =
    (first_value
      (\t,i. let negated_term = dest_not t in
            let pos_hyp_num =
              first_value (\t,i. if t = negated_term then i else fail)
              facts_and_nums
            in
            pos_hyp_num, i
          )
      )
    negative_hyps
  ?
  failwith 'Contradiction'
)
in
if negative_hyp > positive_hyp
  then (notL negative_hyp THEN NthHyp positive_hyp ) proof
  else (notL negative_hyp THEN NthHyp (positive_hyp-1)) proof
;;

```

## Bestimme Werte für Variablen durch Matching

```
let match_subEx quantified_term assumption  =
  letrec match_sub_aux vars exprop =
    map (\var.assoc var (match vars exprop assumption)) (rev vars)
        % Terms must fit quantifier order%
    ?
    let var,type,prop = dest_exists exprop in match_sub_aux (var.vars) prop
  in
    match_sub_aux [] quantified_term
  ;;

letrec exIon terms pf =
  let t.rest = terms in (exI t THEN exIon rest) pf
  ?
  Id pf
;;
let InstantiateEx =
  let InstEx_aux pos pf =
    let sigma = match_subEx (conclusion pf) (type_of_hyp pos pf) in
      (exIon (map snd sigma) THEN (NthHyp pos)) pf
    in
      OnSomeHyp InstEx_aux
;;

```

## Markieren und Kopieren von Beweisen

(In Nuprl 5 vordefiniertes Kommando des Beweiseditors)

```
let Mark name pf = add_saved_proof name pf; Id pf;;  
  
letrec copy_pattern pattern =  
  if is_refined pattern  
  then Try (refine (refinement pattern)  
               THENL (map copy_pattern (children pattern)))  
            )  
  else Id  
;;  
let Copy name = copy_pattern (get_saved_proof name);;
```

---

**saved\_proofs**: globale Variable vom Typ **(tok#proof) list**

**add\_saved\_proof** speichert Beweise in **saved\_proofs**

**get\_saved\_proof** wählt Beweise aus **saved\_proofs**

## ● Einfache Rewrite Taktiken

- Substitution:  $\text{Subst } t_1=t_2 \in T \ c, \text{HypSubst } c_1 \ c_2$
- Falten und Auflösen von Definitionen:  $\text{Fold } \text{name } c, \text{Unfold } \text{name } c$
- **Reduce**  $c$ : wiederholte Auswertung von Redizes in Klausel  $c$

## ● Nuprl’s Rewrite Paket

- Funktionen zur Ersetzung von Termen in Ausdrücken (**conversions**)
- Programmierbar durch **vordefinierte conversions** und **conversionals**
- Rewrite Lemmas zur Rechtfertigung der Ersetzungen
- Unterstützung für **Vielfalt von Äquivalenzrelationen** (nicht nur Gleichheit)
- Taktiken zur **Anwendung** von **conversions** auf Klauseln im Beweis

Details im Nuprl Manual §9.9

- **Sehr hilfreich für Anwendungen**

- Keine Veränderung des eigentlichen Inferenzsystems erforderlich
- Benutzer können Inferenzsystem schnell auf eigene Bedürfnisse anpassen
- Benutzerdefinierte Strategien produzieren keine falschen Ergebnisse

- **Gut für Experimente**

- Ideen können unmittelbar ausprobiert werden

- **Sinnvoll für “kontrollierte” Inferenzen**

- Repräsentation von **Schlüssen** in speziellen Anwendungsbereichen
- **Macro-Inferenzen**: Schließen auf höherem Niveau
- Begrenzte Suche nach Beweisen

- **Nicht sinnvoll für “universelle” Beweissuche**

- Zu **langsam**: jeder Taktikschritt modifiziert den Beweisbaum
- Zu **unkontrolliert**: Taktik gibt unverständliche Teilziele zurück