CS 671 Automated Reasoning

Proof Automation in First Order Logic

1. **Tactic-based proof search**
2. **Complete proof search with** JProver

Tactic-based proof search

Sort rule applications by cost of induced proof search

```
let simple_prover = Repeat
                     (      hypotheses
                      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))
                ;;
```

simple_prover: Component Tactics

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

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

let imp_chain pf  = Chain impE (select_hyps is_imp_term pf) hypotheses pf
                    ;;
let not_chain     = TryAllHyps (\pos. notE pos THEN imp_chain) is_not_term
                    ;;
let 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
                    ;;
```

simple_prover: Matching and instantiation

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

 let InstantiateEx =
   let InstEx_aux pos pf =
        let qterm = conclusion pf
        and hyp = type_of_hyp pos pf             in
           let sigma = match_subEx qterm hyp    in
              let terms = map snd sigma         in
                 (exIon terms THEN (hypothesis pos)) pf
  in
       TryAllHyps InstEx_aux (\h.true)
;;
```

See `/home/kreitz/nuprl/Nuprl5/ml/CS671/Prover-simple.ml` for further details

Integrating Complete Proof Search Procedures

- **Tactic-based proof search has limitations**
  - Many proofs require some "lookahead"
  - Proof search must perform meta-level analysis first
- **Complete proof search procedures are "unintuitive"**
  - Proof search tree represented in compact form
  - Link similar subformulas that may represent leafs of a sequent proof
  - Proof search checks if all leaves can be covered by connections
    and if parameters all connected subformulas can be unified
- JProver**: proof search for Nuprl**
  - Find machine proof of goal sequent and convert it into sequent proof

JProver: Proof Methodology

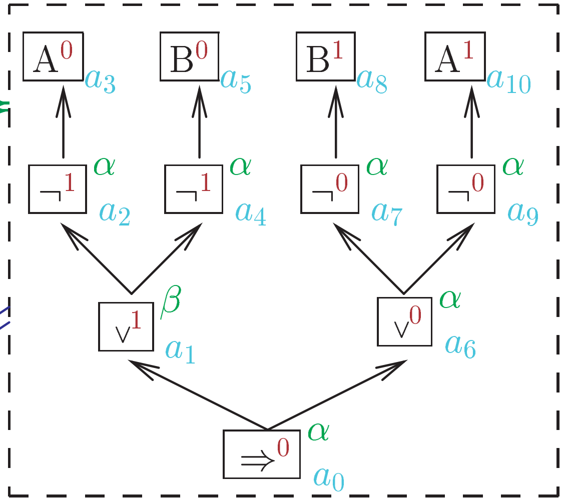
Formula

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

Annotation

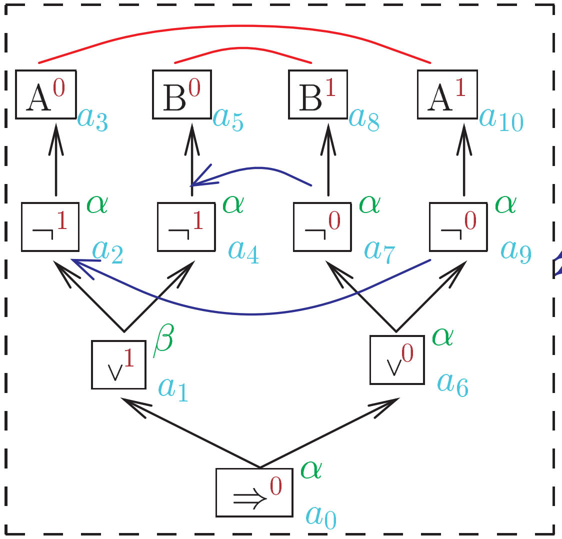
types, polarities, prefixes

Annotated Formula Tree



Matrix Prover

path checking + unification
Substitutions induce ordering $\lhd$



Reduction Ordering $\lhd$

Proof Transformation

Search-free traversal of $\lhd$
multiple $\longrightarrow$ single-conclusion

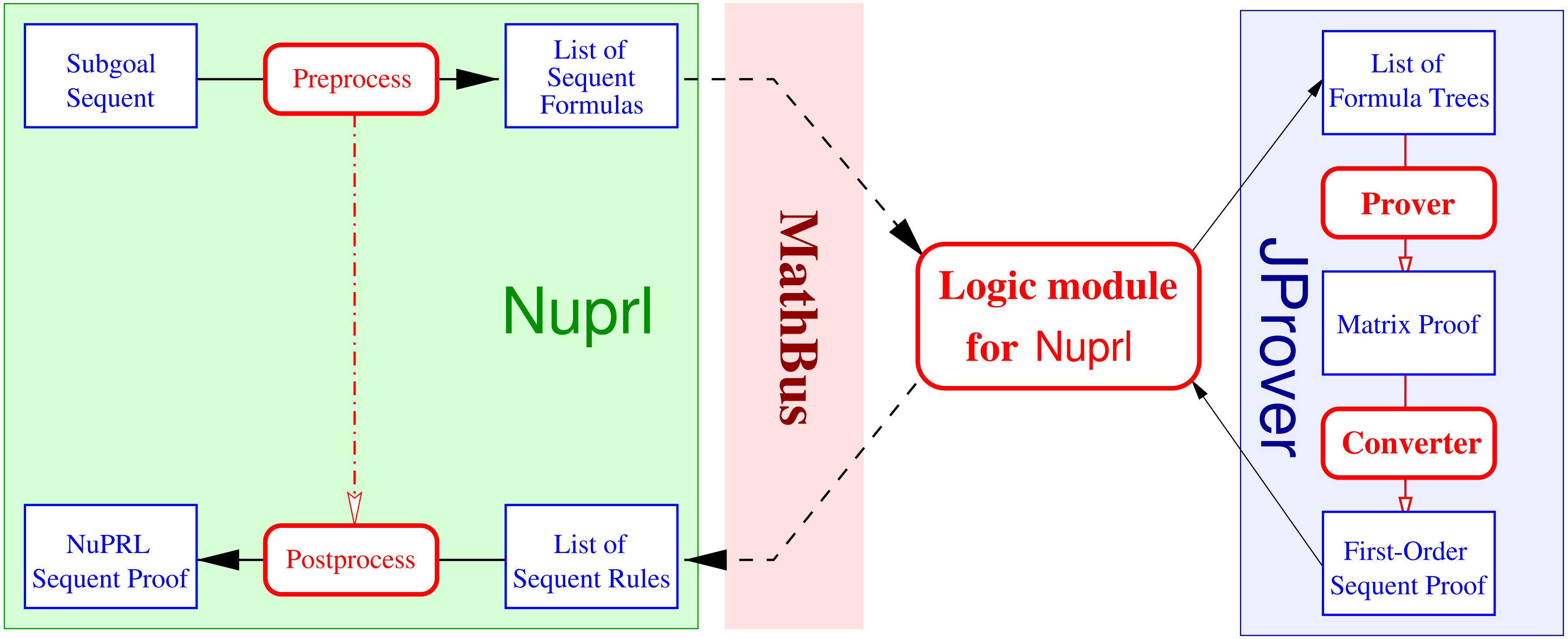
Sequent Proof

$$
\AxiomC{}
\RightLabel{$ax.$}
\UnaryInfC{$A \vdash A$}
\RightLabel{$\neg l$}
\UnaryInfC{$\neg A, A \vdash$}
\RightLabel{$\neg r$}
\UnaryInfC{$\neg A \vdash \neg B, \neg A$}
\AxiomC{}
\RightLabel{$ax.$}
\UnaryInfC{$B \vdash B$}
\RightLabel{$\neg l$}
\UnaryInfC{$\neg B, B \vdash$}
\RightLabel{$\neg r$}
\UnaryInfC{$\neg B \vdash \neg B, \neg A$}
\RightLabel{$\vee l$}
\BinaryInfC{$\neg A \vee \neg B \vdash \neg B, \neg A$}
\RightLabel{$\vee r$}
\UnaryInfC{$\neg A \vee \neg B \vdash \neg B \vee \neg A$}
\RightLabel{$\Rightarrow r$}
\UnaryInfC{$\vdash \neg A \vee \neg B \Rightarrow \neg B \vee \neg A$}
\DisplayProof
$$

The Automated Theorem Prover

- **Proof Search**
  - Matrix prover for first-order intuitionistic logic (connection-driven path checking + term unification) (Kreitz & Otten 1999)
  - Additional string unification for constructive part (Otten & Kreitz 1996)
  - Substitutions and formula tree induce reduction ordering
- **Proof Transformation**
  - Reconstructs first-order sequent proof from matrix proof (Kreitz & Schmitt 2000)
  - Traverses reduction ordering without search (Schmitt 2000)
  - Deals with multiple-/ single-conclusioned sequent calculi (Egly & Schmitt 1999)
- **Implementation** (Schmitt et. al 2001)
  - Stand-alone theorem prover implemented in OCaml
  - Embedded into MetaPRL environment providing basic functionality (term structure, quantifier unification, module system)

JProver: Integration Architecture



- Preprocess Nuprl sequent and semantical differences
- Send terms in MathBus format over an INET socket
- JLogic module: access semantical information from terms;
  convert sequent proof into Nuprl format
- Postproces result into Nuprl proof tree for original sequent

Logical Integration into Nuprl

- **Logic Module: Required Components**
  - OCaml code communicating with proof assistant
  - JLogic module representing the proof assistant's logic

- **The JLogic module**
  - Describes terms implementing Nuprl's logical connectives
  - Provides operations to access subterms
  - Decodes sequent received from communication code
  - Encodes JProver's sequent proof into format for communication code

```
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
```