CADE Tutorial:
The Sequent Calculus of the KeY Tool

Reiner Hähnle & Peter H. Schmitt

Department of Computer Science
Technische Universität Darmstadt / Karlsruhe Institute of Technology

3 August 2015

Material

http://www.key-project.org/cade25-tutorial

Account name: `cade25member`
Password: `deduction4ever`

Introduction

Basic Notions

The Design Space of Sequent/Tableau Calculi

From Calculus to Proof Procedure

Properties of Sequent Calculi

A Classification of Sequent Calculi

Scope of this Tutorial

KeY is a state-of-art semi-automated formal verification tool

Here, we concentrate on first-order reasoning in KeY for Java

Scope of this Tutorial

KeY is a state-of-art semi-automated formal verification tool

Here, we concentrate on first-order reasoning in KeY for Java

How KeY works in a nutshell

- A program logic formalizes a symbolic interpreter for Java
    - Proof nodes correspond to execution stage under a path condition
    - Understanding proof situation essential for interactive paradigm
- Symbolic states represented as first-order expressions
- Loops handled by invariant rule
- Method calls can be (precisely) approximated by contracts
- Symbolic execution interleaved with first-order simplification

Source of interaction: annotations (invariants, contracts), first-order VCs

Brief KeY Demo

BinarySearch.java

A Case Study

The TimSort Bug [De Gouw et al., 2015], CAV 2015

- Java's default sorting algorithm (TimSort) throws uncaught `ArrayIndexOutOfBoundsException` for certain inputs
- Affected Open JDK, Apache products, Haskell, Python, Android
- Bug found during (failed) verification attempt with KeY
  - performed on unaltered JDK code
- Symbolic counter example generation & analysis lead to witness
- Interaction (understanding intermediate proof state) crucial
- Proven with KeY that fixed version throws no exception
  - 2,200,000 rule applications
  - 99.8 % automatic

Requirements on the KeY Calculus

- Full first-order logic (no normal form, nested quantifiers)
- Partially ordered types (reflecting type system of Java, etc)
- Proof state intelligible at interaction points
- No backtracking over interaction points
- Counter example generation
- Manual pruning of proofs possible
- Extensible: many theories
- Heuristic guidance
  - Triggers to instantiate quantifiers
  - Hierachical reasoning, many rules
- Large proofs, Save & Load whole proof

Introduction

Basic Notions

The Design Space of Sequent/Tableau Calculi

- From Calculus to Proof Procedure
- Properties of Sequent Calculi
- A Classification of Sequent Calculi

Untyped First-Order Logic

Vocabulary

A vocabulary Σ consists of

- a set *Func* of function symbols with specified number of arguments
- a set *Pred* of predicate symbols with specified number of arguments
- a potentially infinite set *Var* of variables.

Untyped First-Order Logic

Vocabulary

A vocabulary Σ consists of

- a set *Func* of function symbols with specified number of arguments
- a set *Pred* of predicate symbols with specified number of arguments
- a potentially infinite set *Var* of variables.

Inductive Definition of Terms

If $f \in \mathit{Func}$ with arity n and $t_1, \ldots, t_n$ are terms so is $f(t_1, \ldots, t_n)$.

Untyped First-Order Logic

Vocabulary

A vocabulary Σ consists of

- a set *Func* of function symbols with specified number of arguments
- a set *Pred* of predicate symbols with specified number of arguments
- a potentially infinite set *Var* of variables.

Inductive Definition of Terms

If $f \in \mathit{Func}$ with arity n and $t_1, \dots, t_n$ are terms so is $f(t_1, \dots, t_n)$.

Inductive Definition of Formulas

If $p \in \mathit{Pred}$ with arity n and $t_1, \dots, t_n$ are terms then $p(t_1, \dots, t_n)$ is an (atomic) formula.

If $x \in \mathit{Var}$ and φ_1, φ_2 are formulas, so are

$\neg\varphi_1$, $(\varphi_1 \land \varphi_2)$, $(\varphi_1 \lor \varphi_2)$, $(\varphi_1 \rightarrow \varphi_2)$, $(\varphi_1 \leftrightarrow \varphi_2)$, $(\exists x)\varphi_1$, $(\forall x)\varphi_1$

Sequents

Sequents vs. Tableaux

Differences governed by notation, data structures, polarity

- Could have taken either, but sequents more usual in formal verification systems

Sequents

- A sequent is an expression of the form

$$\Gamma \Longrightarrow \Delta$$

- Γ, Δ finite sets of first-order formulas
- Positive formulation (prove validity)
- Structural rules (ACI) implicit: classical validity, efficiency

Sequents: Syntax & Semantics

Syntax

$$\underbrace{\psi_1, \ldots, \psi_m}_{\textit{Antecedent}} \quad \Longrightarrow \quad \underbrace{\varphi_1, \ldots, \varphi_n}_{\textit{Succedent}}$$

where the φ_i, ψ_i are formulae

Sequents: Syntax & Semantics

Syntax

$$\underbrace{\psi_1, \ldots, \psi_m}_{\textit{Antecedent}} \quad \Longrightarrow \quad \underbrace{\varphi_1, \ldots, \varphi_n}_{\textit{Succedent}}$$

where the φ_i, ψ_i are formulae

Semantics

Same as the formula

$$(\forall \overline{x})\big((\psi_1 \ \wedge \ \cdots \ \wedge \ \psi_m) \quad \rightarrow \quad (\varphi_1 \ \vee \ \cdots \ \vee \ \varphi_n)\big)$$

where $\overline{x} = \text{Free}(\{\psi_1, \ldots, \psi_m, \varphi_1, \ldots, \varphi_n\})$

Sequent Rule Schemata I

Rule schemata where Γ, Δ are metavariables for sets of formulae, φ, ψ for formulae

$$\text{andRight}\ \frac{\Gamma \Longrightarrow \varphi_1, \Delta \qquad \cdots \qquad \Gamma \Longrightarrow \varphi_n, \Delta}{\Gamma \Longrightarrow \varphi_1 \land \cdots \land \varphi_n, \Delta}$$

$$\text{andLeft}\ \frac{\Gamma, \varphi_1, \ldots, \varphi_n \Longrightarrow \Delta}{\Gamma, \varphi_1 \land \cdots \land \varphi_n \Longrightarrow \Delta}$$

$$\text{orLeft}\ \frac{\Gamma, \varphi_1 \Longrightarrow \Delta \qquad \cdots \qquad \Gamma, \varphi_n \Longrightarrow \Delta}{\Gamma, \varphi_1 \lor \cdots \lor \varphi_n \Longrightarrow \Delta}$$

$$\text{orRight}\ \frac{\Gamma \Longrightarrow \varphi_1, \ldots, \varphi_n, \Delta}{\Gamma \Longrightarrow \varphi_1 \lor \cdots \lor \varphi_n, \Delta}$$

Sequent Rule Schemata II

$$\text{allRight}\ \frac{\Gamma \Longrightarrow [x/f(\overline{X})](\varphi), \Delta}{\Gamma \Longrightarrow (\forall\, x)\varphi, \Delta} \qquad \text{allLeft}\ \frac{\Gamma, (\forall\, x)\varphi, [x/X](\varphi) \Longrightarrow \Delta}{\Gamma, (\forall\, x)\varphi \Longrightarrow \Delta}$$

$$\text{exLeft}\ \frac{\Gamma, [x/f(\overline{X})](\varphi) \Longrightarrow \Delta}{\Gamma, (\exists\, x)\varphi \Longrightarrow \Delta} \qquad \text{exRight}\ \frac{\Gamma \Longrightarrow (\exists\, x)\varphi, [x/X](\varphi), \Delta}{\Gamma \Longrightarrow (\exists\, x)\varphi, \Delta}$$

f new function symbol of arity $|\overline{X}|$, where $\overline{X} = \mathsf{Free}((\forall x)\varphi)$

X new variable symbol

Sequent Rule Schemata II

$$\text{allRight}\ \frac{\Gamma \Longrightarrow [x/f(\overline{X})](\varphi), \Delta}{\Gamma \Longrightarrow (\forall\, x)\varphi, \Delta} \qquad \text{allLeft}\ \frac{\Gamma, (\forall\, x)\varphi, [x/X](\varphi) \Longrightarrow \Delta}{\Gamma, (\forall\, x)\varphi \Longrightarrow \Delta}$$

$$\text{exLeft}\ \frac{\Gamma, [x/f(\overline{X})](\varphi) \Longrightarrow \Delta}{\Gamma, (\exists\, x)\varphi \Longrightarrow \Delta} \qquad \text{exRight}\ \frac{\Gamma \Longrightarrow (\exists\, x)\varphi, [x/X](\varphi), \Delta}{\Gamma \Longrightarrow (\exists\, x)\varphi, \Delta}$$

f new function symbol of arity $|\overline{X}|$, where $\overline{X} = \mathsf{Free}((\forall x)\varphi)$

X new variable symbol

$$\text{closeU}\ \frac{\sigma}{\Gamma, \psi \Longrightarrow \varphi, \Delta}$$

σ is MGU of ψ, φ and is applied to whole sequent proof

$$\text{closeFalse}\ \frac{\{\}}{\Gamma, \mathrm{false} \Longrightarrow \Delta} \qquad \text{closeTrue}\ \frac{\{\}}{\Gamma \Longrightarrow \mathrm{true}, \Delta}$$

Sequent Proofs

Definition (Sequent Proof Tree, Sequent Proof)

A sequent proof tree is a tree whose nodes are either sequents or substitutions, inductively defined as follows:

Sequent Proofs

Definition (Sequent Proof Tree, Sequent Proof)

A sequent proof tree is a tree whose nodes are either sequents or substitutions, inductively defined as follows:

1. For any closed sequent S, the tree having S as its single node is a sequent proof tree.

Sequent Proofs

Definition (Sequent Proof Tree, Sequent Proof)

A sequent proof tree is a tree whose nodes are either sequents or substitutions, inductively defined as follows:

1. For any closed sequent S, the tree having S as its single node is a sequent proof tree.
2. If P is a sequent proof tree, S a sequent leaf node in it, and R is an instance of a sequent rule with conclusion S, then a new sequent proof tree P' is obtained by extending S with children whose nodes are exactly the premisses of R. If the premise of R is a substitution σ, then P' is obtained from $\sigma(P)$.

   Notation: $P \preceq P'$

Sequent Proofs

Definition (Sequent Proof Tree, Sequent Proof)

A sequent proof tree is a tree whose nodes are either sequents or substitutions, inductively defined as follows:

1. For any closed sequent S, the tree having S as its single node is a sequent proof tree.
2. If P is a sequent proof tree, S a sequent leaf node in it, and R is an instance of a sequent rule with conclusion S, then a new sequent proof tree P' is obtained by extending S with children whose nodes are exactly the premisses of R. If the premise of R is a substitution σ, then P' is obtained from $\sigma(P)$.

   Notation: $P \preceq P'$

A sequent proof tree (with root node S) whose leaves are all substitutions ("closed") is called sequent proof (for S).

Soundness, Completeness

Theorem (Soundness)

The free variable sequent calculus is sound: If there exists a sequent proof for the closed sequent S, then S is valid.

Theorem (Completeness)

The free variable sequent calculus is complete: If the closed sequent S is valid, then there exists a sequent proof for S.

Introduction

Basic Notions

The Design Space of Sequent/Tableau Calculi
- From Calculus to Proof Procedure
- Properties of Sequent Calculi
- A Classification of Sequent Calculi

A Simplification

Flat sequents

To keep the following technically simple, assume w.l.o.g. flat sequents:

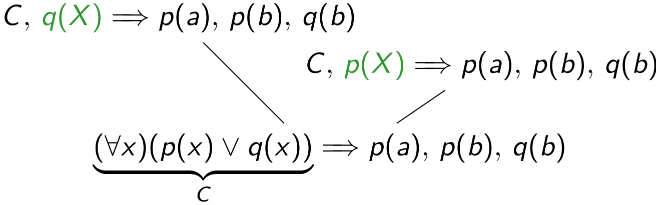
- $(\forall \overline{x})(P_1 \vee \cdots \vee P_m) \in \Gamma$
- $(\exists \overline{y})(Q_1 \wedge \cdots \wedge Q_n) \in \Delta$

where P_i and Q_j are literals.

Dynamic Free Variable Sequent Proof Construction

$$\underbrace{(\forall x)(p(x) \lor q(x))}_{C} \Longrightarrow p(a),\ p(b),\ q(b)$$

Dynamic Free Variable Sequent Proof Construction

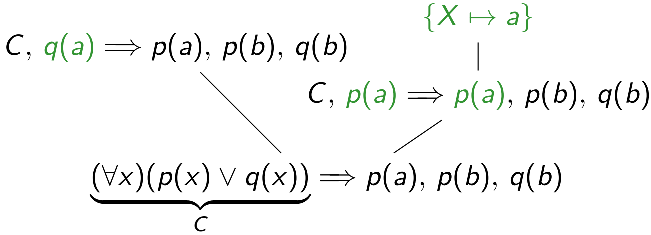


$C,\ q(X) \Longrightarrow p(a),\ p(b),\ q(b)$

$C,\ p(X) \Longrightarrow p(a),\ p(b),\ q(b)$

$$\underbrace{(\forall x)(p(x) \lor q(x))}_{C} \Longrightarrow p(a),\ p(b),\ q(b)$$

Dynamic Free Variable Sequent Proof Construction



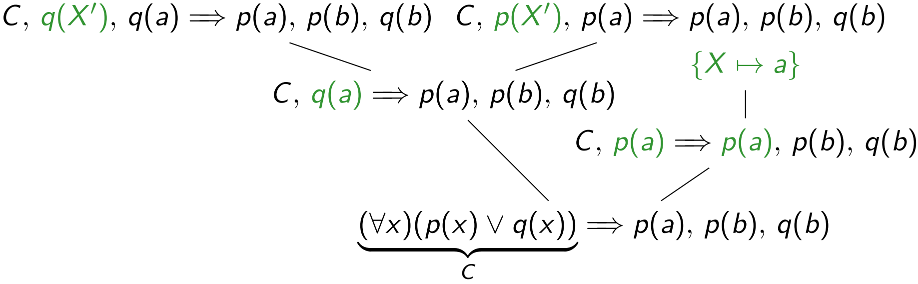
$\{X \mapsto a\}$

$C,\ q(a) \Longrightarrow p(a),\ p(b),\ q(b)$

$C,\ p(a) \Longrightarrow p(a),\ p(b),\ q(b)$

$$\underbrace{(\forall x)(p(x) \lor q(x))}_{C} \Longrightarrow p(a),\ p(b),\ q(b)$$

Dynamic Free Variable Sequent Proof Construction



$C,\ q(X'),\ q(a) \Longrightarrow p(a),\ p(b),\ q(b)$

$C,\ p(X'),\ p(a) \Longrightarrow p(a),\ p(b),\ q(b)$

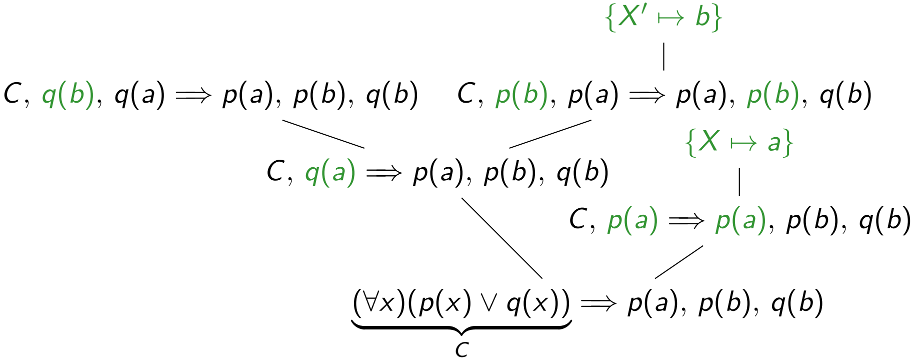
$\{X \mapsto a\}$

$C,\ q(a) \Longrightarrow p(a),\ p(b),\ q(b)$

$C,\ p(a) \Longrightarrow p(a),\ p(b),\ q(b)$

$$\underbrace{(\forall x)(p(x) \lor q(x))}_{C} \Longrightarrow p(a),\ p(b),\ q(b)$$

Dynamic Free Variable Sequent Proof Construction



$\{X' \mapsto b\}$

$C,\ q(b),\ q(a) \Longrightarrow p(a),\ p(b),\ q(b)$ $\qquad$ $C,\ p(b),\ p(a) \Longrightarrow p(a),\ p(b),\ q(b)$

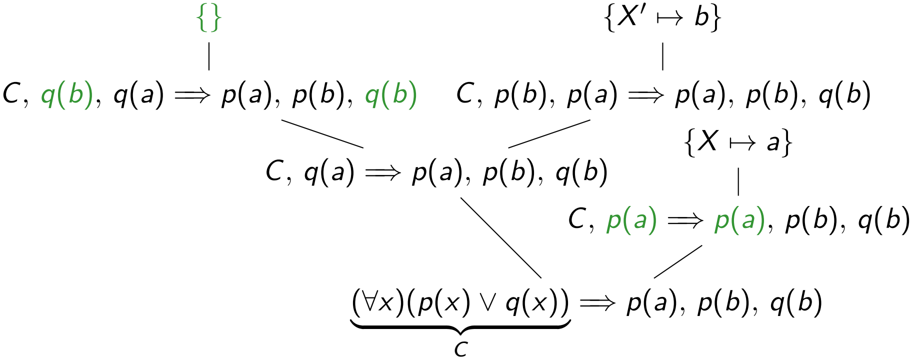
$\{X \mapsto a\}$

$C,\ q(a) \Longrightarrow p(a),\ p(b),\ q(b)$

$C,\ p(a) \Longrightarrow p(a),\ p(b),\ q(b)$

$$\underbrace{(\forall x)(p(x) \vee q(x))}_{C} \Longrightarrow p(a),\ p(b),\ q(b)$$

Dynamic Free Variable Sequent Proof Construction



$\{\}$

$C,\ q(b),\ q(a) \Longrightarrow p(a),\ p(b),\ q(b)$

$\{X' \mapsto b\}$

$C,\ p(b),\ p(a) \Longrightarrow p(a),\ p(b),\ q(b)$

$C,\ q(a) \Longrightarrow p(a),\ p(b),\ q(b)$

$\{X \mapsto a\}$

$C,\ p(a) \Longrightarrow p(a),\ p(b),\ q(b)$

$$\underbrace{(\forall x)(p(x) \vee q(x))}_{C} \Longrightarrow p(a),\ p(b),\ q(b)$$

From Calculus to Proof Procedure

Completeness merely guarantees **existence** of sequent proof:
Proof (search) procedure needed to find it!

From Calculus to Proof Procedure

Completeness merely guarantees **existence** of sequent proof:
Proof (search) procedure needed to find it!

Choice Points of Non-Deterministic Sequent Proof Search

1. Next open goal a rule is applied to?
2. Close the goal or extend it?
3. Extension: with which main formula?
4. closeU: with which literals (which MGU)?

From Calculus to Proof Procedure

Completeness merely guarantees **existence** of sequent proof:
Proof (search) procedure needed to find it!

Choice Points of Non-Deterministic Sequent Proof Search

1. Next open goal a rule is applied to?
2. Close the goal or extend it?
3. Extension: with which main formula?
4. closeU: with which literals (which MGU)?

Bad choice can prevent finding a sequent proof for unsatisfiable formula

From Calculus to Proof Procedure Cont'd

Definition (Sequent Proof Procedure)

A **sequent proof procedure** consists of

1. a sequent calculus (a set of sequent rule schemata);
2. a function computing for given sequent proof tree P
   in deterministic polynomial time (in size of P)
   the kind, instance and position of the next rule to be applied on P.

This function is called **(sequent) computation rule**.

From Calculus to Proof Procedure Cont'd

Definition (Sequent Proof Procedure)

A **sequent proof procedure** consists of

1. a sequent calculus (a set of sequent rule schemata);
2. a function computing for given sequent proof tree P
   in deterministic polynomial time (in size of P)
   the kind, instance and position of the next rule to be applied on P.

This function is called **(sequent) computation rule**.

Definition (Strongly Complete)

A sequent proof procedure that preserves completeness of the underlying calculus (i.e., computes a proof for any given valid root sequent) is called **strongly complete**.

From Calculus to Proof Procedure Cont'd

Subgoal Selection

Observations

- All subgoals of a sequent tree must be closed
- Consequence of lifting construction in completeness theorem: sequence of closure rule applications is irrelevant
- Consequence of proof of ground completeness: No need to work on closed subgoals

From Calculus to Proof Procedure Cont'd

Subgoal Selection

Observations

- All subgoals of a sequent tree must be closed
- Consequence of lifting construction in completeness theorem: sequence of closure rule applications is irrelevant
- Consequence of proof of ground completeness: No need to work on closed subgoals

Any deterministic computation rule selecting open subgoals will do

From Calculus to Proof Procedure Cont'd

Subgoal Selection

Observations

- All subgoals of a sequent tree must be closed
- Consequence of lifting construction in completeness theorem: sequence of closure rule applications is irrelevant
- Consequence of proof of ground completeness: No need to work on closed subgoals

Any deterministic computation rule selecting open subgoals will do

Common choices of computation rule for subgoal selection

Typically driven by effiency in implementation

- leftmost-open-first
- rightmost-open-first

From Calculus to Proof Procedure Cont'd

Closure vs. Extension

Select Kind of Sequent Rule: (closeU)/(Extension)

Bad news: greedy closure can destroy completeness

From Calculus to Proof Procedure Cont'd

Closure vs. Extension

Select Kind of Sequent Rule: (closeU)/(Extension)

Bad news: greedy closure can destroy completeness

Example

Right-open-first subgoal computation rule, main formulas selected round-robin C_1, C_2, C_3, C_4, ...

$$\overbrace{(\forall u)p(u,a)}^{C_1},\ \overbrace{(\forall y)(p(y,b) \vee r(y))}^{C_4},\ q(b),\ r(a) \Longrightarrow \overbrace{(\exists x)(p(a,x) \wedge q(x))}^{C_2},\ \overbrace{(\exists w)p(b,w)}^{C_3}$$

is valid, but ...

From Calculus to Proof Procedure Cont'd

Closure vs. Extension

$$\overbrace{(\forall u)p(u,a)}^{C_1},\ \overbrace{(\forall y)(p(y,b) \vee r(y))}^{C_4},\ q(b),\ r(a) \Longrightarrow \overbrace{(\exists x)(p(a,x) \wedge q(x))}^{C_2},\ \overbrace{(\exists w)p(b,w)}^{C_3}$$

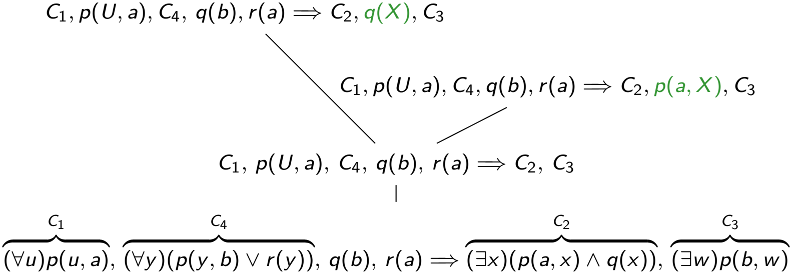
From Calculus to Proof Procedure Cont'd

Closure vs. Extension

$$
\begin{array}{c}
C_1,\ p(U,a),\ C_4,\ q(b),\ r(a) \Longrightarrow C_2,\ C_3 \\
| \\
\overbrace{(\forall u)p(u,a)}^{C_1},\ \overbrace{(\forall y)(p(y,b) \vee r(y))}^{C_4},\ q(b),\ r(a) \Longrightarrow \overbrace{(\exists x)(p(a,x) \wedge q(x))}^{C_2},\ \overbrace{(\exists w)p(b,w)}^{C_3}
\end{array}
$$

From Calculus to Proof Procedure Cont'd

Closure vs. Extension



$$C_1, p(U,a), C_4,\ q(b), r(a) \Longrightarrow C_2, q(X), C_3$$

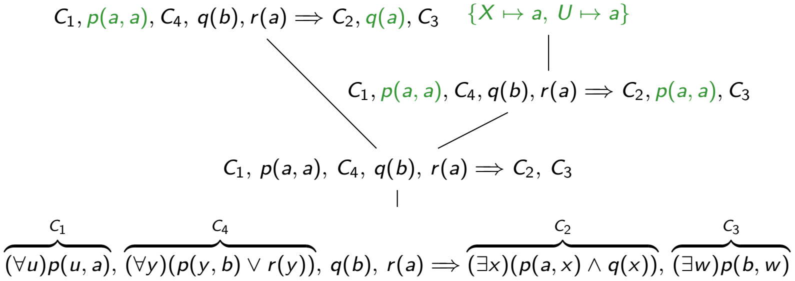
$$C_1, p(U,a), C_4, q(b), r(a) \Longrightarrow C_2, p(a,X), C_3$$

$$C_1,\ p(U,a),\ C_4,\ q(b),\ r(a) \Longrightarrow C_2,\ C_3$$

$$\overbrace{(\forall u)p(u,a)}^{C_1},\ \overbrace{(\forall y)(p(y,b) \lor r(y))}^{C_4},\ q(b),\ r(a) \Longrightarrow \overbrace{(\exists x)(p(a,x) \land q(x))}^{C_2},\ \overbrace{(\exists w)p(b,w)}^{C_3}$$

From Calculus to Proof Procedure Cont'd

Closure vs. Extension



$$C_1, p(a,a), C_4,\ q(b), r(a) \Longrightarrow C_2, q(a), C_3 \qquad \{X \mapsto a,\ U \mapsto a\}$$

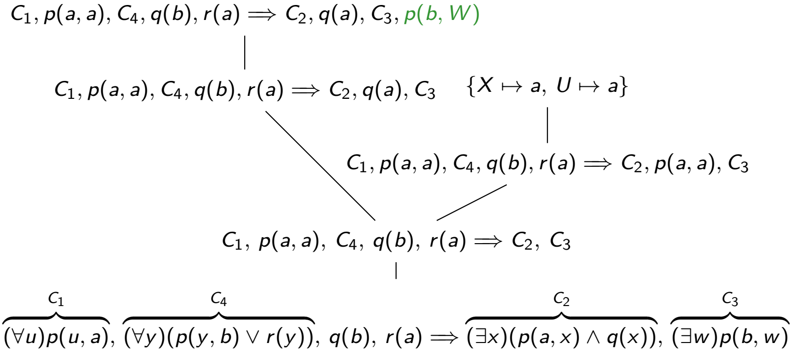
$$C_1, p(a,a), C_4, q(b), r(a) \Longrightarrow C_2, p(a,a), C_3$$

$$C_1,\ p(a,a),\ C_4,\ q(b),\ r(a) \Longrightarrow C_2,\ C_3$$

$$\overbrace{(\forall u)p(u,a)}^{C_1},\ \overbrace{(\forall y)(p(y,b) \vee r(y))}^{C_4},\ q(b),\ r(a) \Longrightarrow \overbrace{(\exists x)(p(a,x) \wedge q(x))}^{C_2},\ \overbrace{(\exists w)p(b,w)}^{C_3}$$

From Calculus to Proof Procedure Cont'd

Closure vs. Extension



$$C_1, p(a,a), C_4, q(b), r(a) \Longrightarrow C_2, q(a), C_3, p(b, W)$$

$$C_1, p(a,a), C_4, q(b), r(a) \Longrightarrow C_2, q(a), C_3 \qquad \{X \mapsto a,\ U \mapsto a\}$$

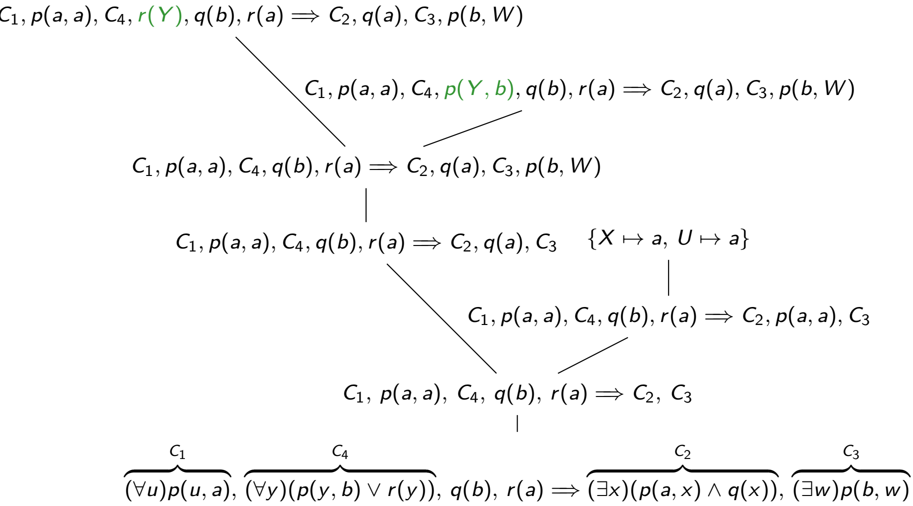
$$C_1, p(a,a), C_4, q(b), r(a) \Longrightarrow C_2, p(a,a), C_3$$

$$C_1,\ p(a,a),\ C_4,\ q(b),\ r(a) \Longrightarrow C_2,\ C_3$$

$$\overbrace{(\forall u)p(u,a)}^{C_1},\ \overbrace{(\forall y)(p(y,b) \vee r(y))}^{C_4},\ q(b),\ r(a) \Longrightarrow \overbrace{(\exists x)(p(a,x) \wedge q(x))}^{C_2},\ \overbrace{(\exists w)p(b,w)}^{C_3}$$

From Calculus to Proof Procedure Cont'd

Closure vs. Extension



$C_1, p(a,a), C_4, r(Y), q(b), r(a) \Longrightarrow C_2, q(a), C_3, p(b,W)$

$C_1, p(a,a), C_4, p(Y,b), q(b), r(a) \Longrightarrow C_2, q(a), C_3, p(b,W)$

$C_1, p(a,a), C_4, q(b), r(a) \Longrightarrow C_2, q(a), C_3, p(b,W)$

$C_1, p(a,a), C_4, q(b), r(a) \Longrightarrow C_2, q(a), C_3$

$\{X \mapsto a,\ U \mapsto a\}$

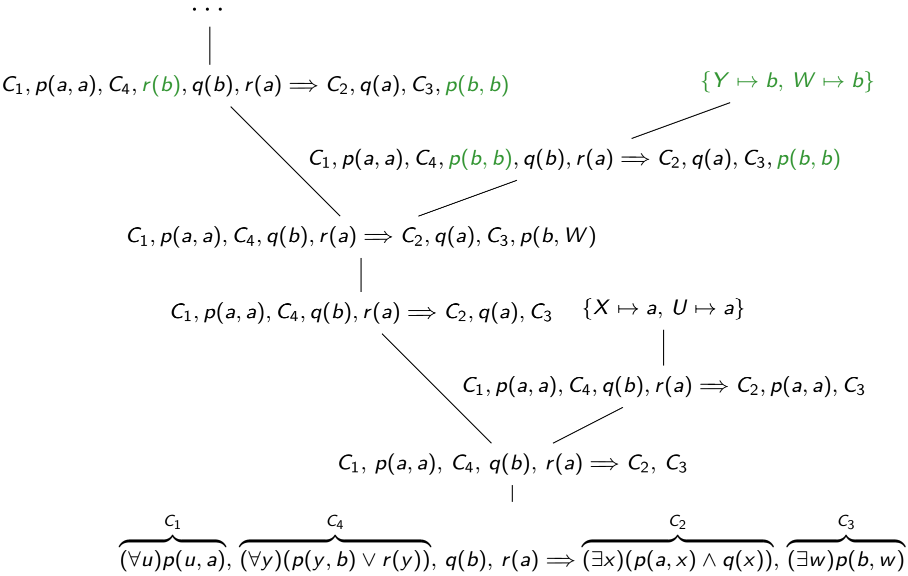
$C_1, p(a,a), C_4, q(b), r(a) \Longrightarrow C_2, p(a,a), C_3$

$C_1,\ p(a,a),\ C_4,\ q(b),\ r(a) \Longrightarrow C_2,\ C_3$

$$\overbrace{(\forall u)p(u,a)}^{C_1},\ \overbrace{(\forall y)(p(y,b) \vee r(y))}^{C_4},\ q(b),\ r(a) \Longrightarrow \overbrace{(\exists x)(p(a,x) \wedge q(x))}^{C_2},\ \overbrace{(\exists w)p(b,w)}^{C_3}$$

From Calculus to Proof Procedure Cont'd

Closure vs. Extension



$\ldots$

$C_1, p(a,a), C_4, r(b), q(b), r(a) \Longrightarrow C_2, q(a), C_3, p(b,b)$

$\{Y \mapsto b,\ W \mapsto b\}$

$C_1, p(a,a), C_4, p(b,b), q(b), r(a) \Longrightarrow C_2, q(a), C_3, p(b,b)$

$C_1, p(a,a), C_4, q(b), r(a) \Longrightarrow C_2, q(a), C_3, p(b,W)$

$C_1, p(a,a), C_4, q(b), r(a) \Longrightarrow C_2, q(a), C_3$

$\{X \mapsto a,\ U \mapsto a\}$

$C_1, p(a,a), C_4, q(b), r(a) \Longrightarrow C_2, p(a,a), C_3$

$C_1,\ p(a,a),\ C_4,\ q(b),\ r(a) \Longrightarrow C_2,\ C_3$

$$\overbrace{(\forall u)p(u,a)}^{C_1},\ \overbrace{(\forall y)(p(y,b) \vee r(y))}^{C_4},\ q(b),\ r(a) \Longrightarrow \overbrace{(\exists x)(p(a,x) \wedge q(x))}^{C_2},\ \overbrace{(\exists w)p(b,w)}^{C_3}$$

From Calculus to Proof Procedure Cont'd

Main Formula Selection

Select Main Formula Used for Extension

Unfair choice can prevent subgoal closure

Example

$$\begin{array}{c}
\vdots \\
p(X''),\ p(X'),\ p(X),\ (\forall x)p(x),\ q \Longrightarrow q \\
| \\
p(X'),\ p(X),\ (\forall x)p(x),\ q \Longrightarrow q \\
| \\
p(X),\ (\forall x)p(x),\ q \Longrightarrow q \\
| \\
(\forall x)p(x),\ q \Longrightarrow q
\end{array}$$

Fairness

Fair Computation Rule

Defined in the usual manner

- No incompleteness due to main formula selection
- Easy to implement (queue allLeft, exRight at end after application)
- Even fair computation rule doesn't prevent incompleteness from greedy closure

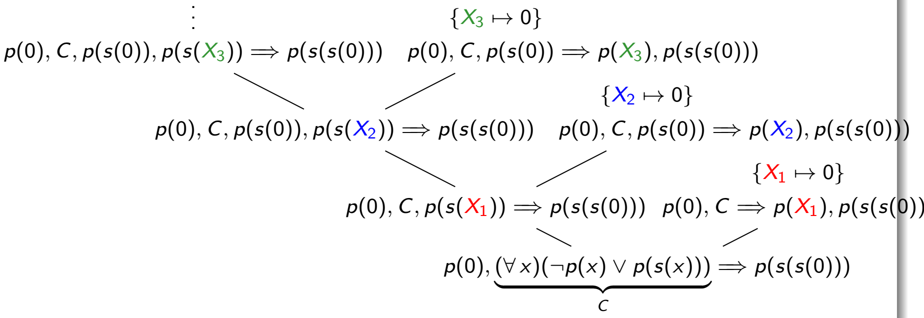
From Calculus to Proof Procedure Cont'd

MGU Selection

Select MGU Used for closeU

Unfair choice among several possible MGUs can prevent closure

Example



$$\vdots$$

$$\{X_3 \mapsto 0\}$$

$$p(0), C, p(s(0)), p(s(X_3)) \Longrightarrow p(s(s(0))) \quad p(0), C, p(s(0)) \Longrightarrow p(X_3), p(s(s(0)))$$

$$\{X_2 \mapsto 0\}$$

$$p(0), C, p(s(0)), p(s(X_2)) \Longrightarrow p(s(s(0))) \quad p(0), C, p(s(0)) \Longrightarrow p(X_2), p(s(s(0)))$$

$$\{X_1 \mapsto 0\}$$

$$p(0), C, p(s(X_1)) \Longrightarrow p(s(s(0))) \quad p(0), C \Longrightarrow p(X_1), p(s(s(0))$$

$$p(0), \underbrace{(\forall x)(\neg p(x) \vee p(s(x)))}_{C} \Longrightarrow p(s(s(0)))$$

From Calculus to Proof Procedure

Summary

- A computation rule turns the non-deterministic sequent calculus into an implementable search procedure
- Selection of (open) subgoals is uncritical
- Fair selection of main formulas required for completeness
  - Deals effectively with that choice point
- How to deal with choice Closure vs. Extension and choice of MGU?
  - Greedy closure causes incompleteness even for fair computation rule
  - No obvious fairness notion for different possible MGUs in closure

Introduction

Basic Notions

The Design Space of Sequent/Tableau Calculi

- From Calculus to Proof Procedure
- Properties of Sequent Calculi
- A Classification of Sequent Calculi

Two Central Properties of Sequent/Tableau Calculi

closeU and main formula selection can interact subtly

Two Central Properties of Sequent/Tableau Calculi

closeU and main formula selection can interact subtly

Definition (Destructive Sequent Calculus)

A sequent calculus is **non-destructive** if all sequent proof trees P' such that $P \preceq P'$ contain P as an initial subtree.

Two Central Properties of Sequent/Tableau Calculi

closeU and main formula selection can interact subtly

Definition (Destructive Sequent Calculus)

A sequent calculus is **non-destructive** if all sequent proof trees P' such that $P \preceq P'$ contain P as an initial subtree.

closeU rule renders free variable sequent calculus destructive

Two Central Properties of Sequent/Tableau Calculi

closeU and main formula selection can interact subtly

Definition (Destructive Sequent Calculus)

A sequent calculus is **non-destructive** if all sequent proof trees P' such that $P \preceq P'$ contain P as an initial subtree.

closeU rule renders free variable sequent calculus destructive

Definition (Proof Confluent Sequent Calculus)

A sequent calculus is **proof confluent** if every sequent proof tree with a valid root sequent S can be extended to a sequent proof for S.

Proof confluence: “no need to backtrack”

Trade-Offs for the Design of Proof Procedures

Proof Confluence is Highly Desirable

1. Proof confluence avoids necessity for proof enumeration (implicit via backtracking or explicit via breadth-first search).
2. In a proof confluent framework, open subgoals where rules were exhaustively applied indicate satisfiability and allow construction of counter models (+ simplify completeness proof).

Trade-Offs for the Design of Proof Procedures

Proof Confluence is Highly Desirable

1. Proof confluence avoids necessity for proof enumeration (implicit via backtracking or explicit via breadth-first search).
2. In a proof confluent framework, open subgoals where rules were exhaustively applied indicate satisfiability and allow construction of counter models (+ simplify completeness proof).

Main problem: How to deal with destructive closeU rule?

Allow it A strongly complete, destructive sequent proof procedure
Does it even exist? Must deal with fairness of MGUs in closeU!

Avoid it Replace closeU with something non-destructive

Introduction

Basic Notions

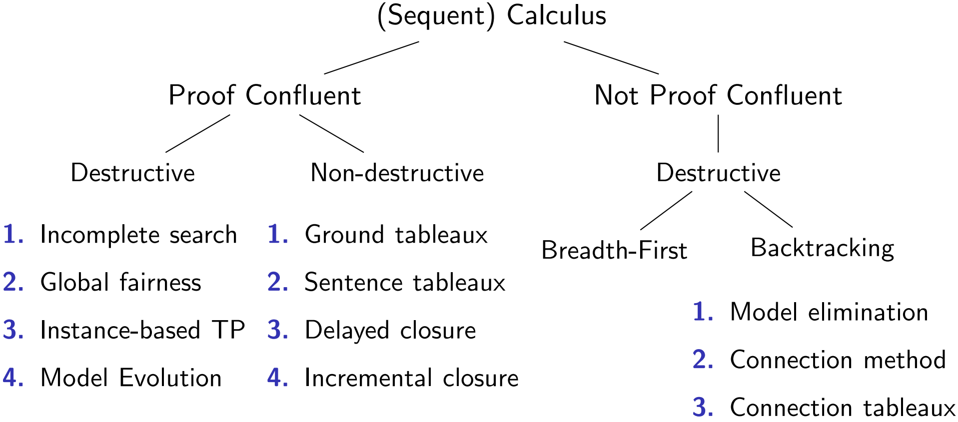
The Design Space of Sequent/Tableau Calculi

From Calculus to Proof Procedure

Properties of Sequent Calculi

A Classification of Sequent Calculi

A Classification of Sequent-Like Calculi



(Sequent) Calculus

Proof Confluent

Destructive

1. Incomplete search
2. Global fairness
3. Instance-based TP
4. Model Evolution

Non-destructive

1. Ground tableaux
2. Sentence tableaux
3. Delayed closure
4. Incremental closure

Not Proof Confluent

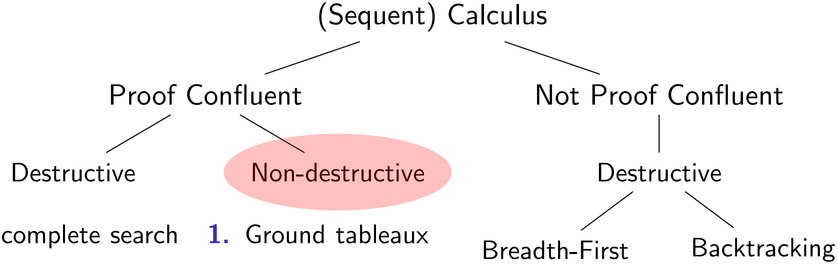
Destructive

Breadth-First

Backtracking

1. Model elimination
2. Connection method
3. Connection tableaux

A Classification of Sequent-Like Calculi



(Sequent) Calculus

Proof Confluent

Not Proof Confluent

Destructive

Non-destructive

Destructive

Breadth-First

Backtracking

Proof Confluent – Destructive:

1. Incomplete search
2. Global fairness
3. Instance-based TP
4. Model Evolution

Proof Confluent – Non-destructive:

1. Ground tableaux
2. Sentence tableaux
3. Delayed closure
4. Incremental closure

Not Proof Confluent – Destructive – Backtracking:

1. Model elimination
2. Connection method
3. Connection tableaux

The Proof Confluent, Non-Destructive Case

Avoid destructiveness

Assuming a fair computation rule for main formula selection

1 Ground/Propositional Calculi

Sequents quantifier-free: all MGUs empty ⇒ closeU is non-destructive

- Not available for general FOL
- Works also for bounded/range-restricted formulas

The Proof Confluent, Non-Destructive Case

Avoid destructiveness

Assuming a fair computation rule for main formula selection

1 Ground/Propositional Calculi

Sequents quantifier-free: all MGUs empty ⇒ closeU is non-destructive

- Not available for general FOL
- Works also for bounded/range-restricted formulas

2 Smullyan or Sentence Calculi [Smullyan, 1968]

- In allLeft, exRight, instead of fresh variables, use ground instances
- Combine enumeration of ground instances and fair main formula selection

Discussion:

- Unguided enumeration of ground terms very inefficient search
- Incomplete, heuristic "triggers" can work well in specific situations (used as instantiation patterns in SMT solvers and KeY)

The Proof Confluent, Non-Destructive Case Cont'd

Delay destructiveness

Assuming a fair computation rule for main formula selection

3 Delayed Closure Rule

Apply closeU only if all open subgoals can be closed simultaneously

- Cannot discard closable subgoals: possible space problem
- Repeated closure test of same branches

The Proof Confluent, Non-Destructive Case Cont'd

Delay destructiveness

Assuming a fair computation rule for main formula selection

3 Delayed Closure Rule

Apply closeU only if all open subgoals can be closed simultaneously

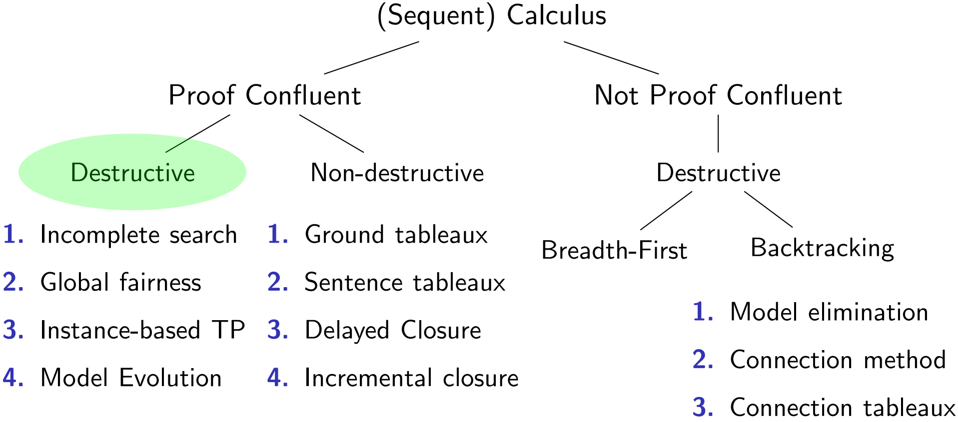
- Cannot discard closable subgoals: possible space problem
- Repeated closure test of same branches

4 Calculi with Incremental Closure [Giese, 2001]

At each proof node maintain constraint system characterizing all possible closures of the subtree above it without applying them

- Many tricky implementation issues, system PRINS
- Several faulty implementation attempts exist in literature
- System PRINCESS FOL+LIA won TFA division of CASC 2012

A Classification of Sequent-Like Calculi



(Sequent) Calculus

Proof Confluent

Not Proof Confluent

Destructive

1. Incomplete search
2. Global fairness
3. Instance-based TP
4. Model Evolution

Non-destructive

1. Ground tableaux
2. Sentence tableaux
3. Delayed Closure
4. Incremental closure

Destructive

Breadth-First

Backtracking

1. Model elimination
2. Connection method
3. Connection tableaux

The Proof Confluent, Destructive Case

1 Accept Incompleteness (Bounded Reasoning)

Limit number of instances or size of MGUs to achieve finiteness

- Nature of incompleteness also practical problem (just as in bounded MC)
- Hard to find natural bounds, explosive growth

The Proof Confluent, Destructive Case

1 Accept Incompleteness (Bounded Reasoning)

Limit number of instances or size of MGUs to achieve finiteness

- Nature of incompleteness also practical problem (just as in bounded MC)
- Hard to find natural bounds, explosive growth

2 Global Fairness [Beckert, 2001]

Fairness takes main formula selection and closeU into account

A strongly complete, destructive proof procedure

- Fair computation rule requires to keep closed subgoals
- Was never properly implemented due to its complexity

The Proof Confluent, Destructive Case Cont'd

3 Instance-Based Theorem Proving "Third Stream"

Compute from MGU in closeU formula instances that are added to sequents

Moves fairness issue from closeU to formula selection: easier to handle

Disconnection Method [Billon, 1996] not properly implemented

Hyper Tableaux [Baumgartner, 1998] used/maintained until 2010

Disconnection Tableaux [Letz & Stenz, 2001] DCTP until 2007

Related, but not tableau-based: system IPROVER by K. Korovin

The Proof Confluent, Destructive Case Cont'd

3 Instance-Based Theorem Proving "Third Stream"

Compute from MGU in closeU formula instances that are added to sequents

Moves fairness issue from closeU to formula selection: easier to handle

Disconnection Method [Billon, 1996] not properly implemented

Hyper Tableaux [Baumgartner, 1998] used/maintained until 2010

Disconnection Tableaux [Letz & Stenz, 2001] DCTP until 2007

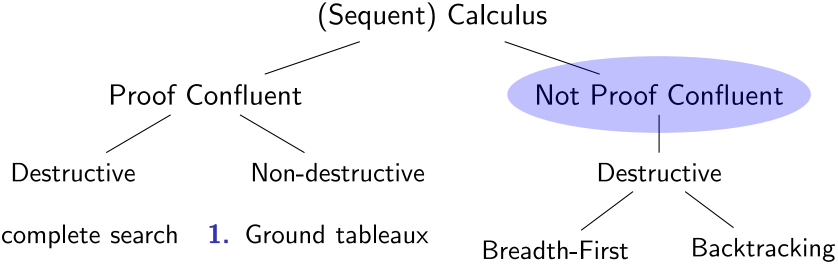
Related, but not tableau-based: system IPROVER by K. Korovin

4 Model Evolution [Baumgartner & Tinelli, 2003]

Use MGUs to maintain partial Herbrand model as **non-ground** literal set

- Atoms in model are **universal literals** wrt their variables
- Systems DARWIN, E-DARWIN, until 2012?

A Classification of Sequent-Like Calculi



(Sequent) Calculus

Proof Confluent

Not Proof Confluent

Destructive

Non-destructive

Destructive

Breadth-First

Backtracking

Proof Confluent – Destructive:

1. Incomplete search
2. Global fairness
3. Instance-based TP
4. Model Evolution

Proof Confluent – Non-destructive:

1. Ground tableaux
2. Sentence tableaux
3. Delayed Closure
4. Incremental closure

Not Proof Confluent – Destructive – Backtracking:

1. Model elimination
2. Connection method
3. Connection tableaux

The Non-Proof Confluent Case

For each choice of closure vs. extension and each MGU in closeU explore all possible sequent proofs

Breadth-First Search

- Node in search tree is a sequent proof tree, proofs are success nodes
- Root sequent finite, # premisses finite, only MGUs: branching degree finite
- Success nodes (i.e., finite proofs) must occur at finite depth
- Space inefficiency

The Non-Proof Confluent Case

For each choice of closure vs. extension and each MGU in closeU explore all possible sequent proofs

Depth-First Iterative Deepening Search (DFID)

Space-efficient implementation of breadth-first search

- Enumerate sequent trees until finite limit via backtracking + increment
- Used in practice for non-confluent proof procedures

The Non-Proof Confluent Case

For each choice of closure vs. extension and each MGU in closeU explore all possible sequent proofs

Depth-First Iterative Deepening Search (DFID)

Space-efficient implementation of breadth-first search

- Enumerate sequent trees until finite limit via backtracking + increment
- Used in practice for non-confluent proof procedures

Some sequent-like calculi are non-proof confluent already at ground level

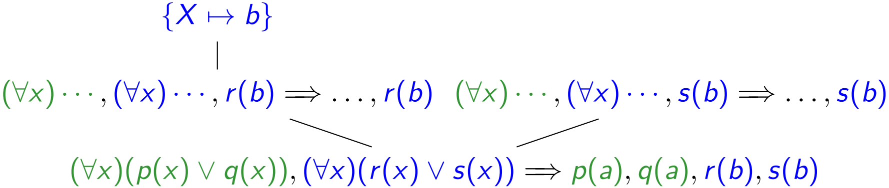
Connection Conditions: Motivation

Example

$$\dfrac{(\forall x)\cdots,(\forall x)\cdots,r(X) \Longrightarrow \ldots,r(b) \qquad (\forall x)\cdots,(\forall x)\cdots,s(X) \Longrightarrow \ldots,s(b)}{(\forall x)(p(x) \lor q(x)),(\forall x)(r(x) \lor s(x)) \Longrightarrow p(a),q(a),r(b),s(b)}$$

Connection Conditions: Motivation

Example



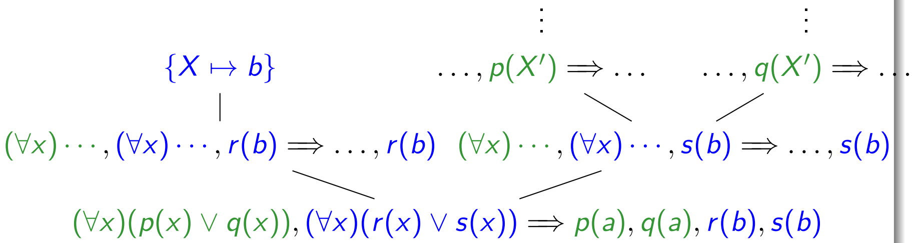
$\{X \mapsto b\}$

$(\forall x)\cdots, (\forall x)\cdots, r(b) \Longrightarrow \ldots, r(b)$ $\quad$ $(\forall x)\cdots, (\forall x)\cdots, s(b) \Longrightarrow \ldots, s(b)$

$(\forall x)(p(x) \vee q(x)), (\forall x)(r(x) \vee s(x)) \Longrightarrow p(a), q(a), r(b), s(b)$

Connection Conditions: Motivation

Example



$\vdots \qquad \vdots$

$\{X \mapsto b\} \qquad \ldots, p(X') \Longrightarrow \ldots \qquad \ldots, q(X') \Longrightarrow \ldots$

$(\forall x)\cdots,(\forall x)\cdots,r(b) \Longrightarrow \ldots,r(b) \qquad (\forall x)\cdots,(\forall x)\cdots,s(b) \Longrightarrow \ldots,s(b)$

$(\forall x)(p(x) \lor q(x)),(\forall x)(r(x) \lor s(x)) \Longrightarrow p(a),q(a),r(b),s(b)$

Connection Condition for Sequents

Definition (Connection Condition)

A sequent proof tree satisfies the **connection condition** if in each orLeft/andRight rule application at least one of the new literals in the premisses is complementary to the literal introduced in the most recent orLeft/andRight rule application.

- At least one new subgoal is immediately closeable
- Technically realized by combining orLeft/andRight with closeU
- Doesn't restrict the first orLeft/andRight rule application
- Can be generalized to non-flat formulas, but (even more) messy
- Matrix or semantic path characterizations more adequate

Properties of Connection Conditions

Lemma

Ground sequents with connection condition are not proof confluent.

Proof.

$$p \vee q,\ r \vee s \Longrightarrow p, q$$

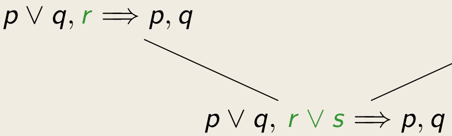
□

Properties of Connection Conditions

Lemma

Ground sequents with connection condition are not proof confluent.

Proof.



$$p \vee q, r \Longrightarrow p, q \qquad\qquad p \vee q, s \Longrightarrow p, q$$

$$p \vee q,\ r \vee s \Longrightarrow p, q$$

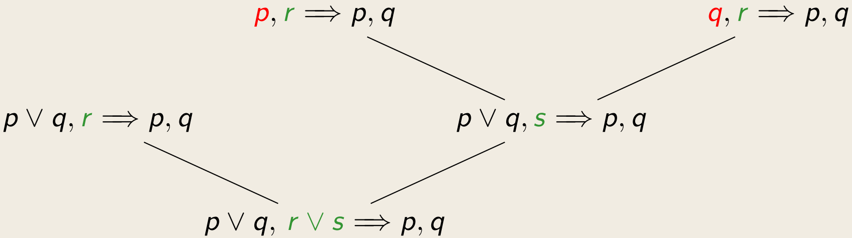
□

Properties of Connection Conditions

Lemma

Ground sequents with connection condition are not proof confluent.

Proof.



$p, r \Longrightarrow p, q$ $\qquad$ $q, r \Longrightarrow p, q$

$p \vee q, r \Longrightarrow p, q$ $\qquad$ $p \vee q, s \Longrightarrow p, q$

$p \vee q, \ r \vee s \Longrightarrow p, q$

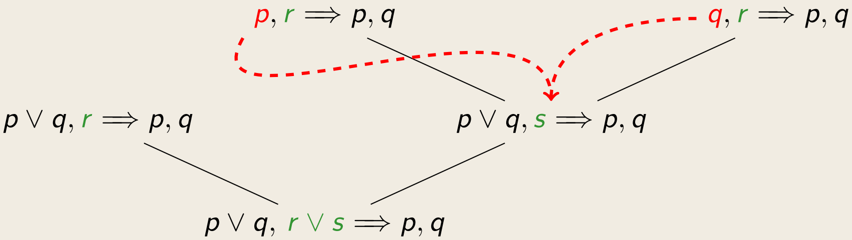
□

Properties of Connection Conditions

Lemma

Ground sequents with connection condition are not proof confluent.

Proof.



$p, r \Longrightarrow p, q$

$q, r \Longrightarrow p, q$

$p \vee q, r \Longrightarrow p, q$

$p \vee q, s \Longrightarrow p, q$

$p \vee q,\ r \vee s \Longrightarrow p, q$

□

Non-proof Confluent Calculi with Backtracking

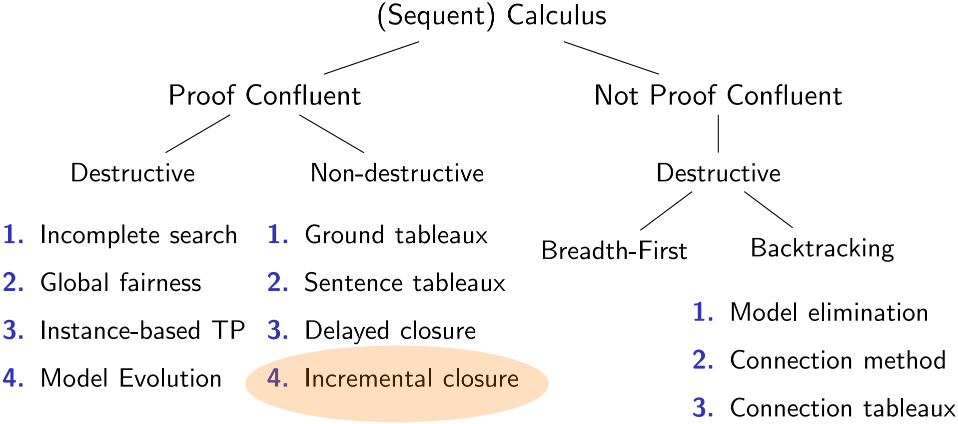
Summary

- Connection condition necessitates backtracking even for ground completeness
- Non-proof confluent refinements typically require syntactic completeness proof (really messy in non-clausal case)
- Implementations (for CNF)
  - SETHEO (1992–2002) regular connection tableaux
    - 1995–2002 a leading system
  - leanCoP 2.1: 6 PROLOG clauses, $<$ 1kB
    - surprisingly efficient, amazing PROLOG hack

Requirements on the KeY Calculus, Revisited

- Full first-order logic (no normal form, nested quantifiers)
- Partially ordered types (reflecting type system of Java, etc)
- Proof state intelligible at interaction points
- No backtracking over interaction points
- Counter example generation
- Manual pruning of proofs possible
- Extensible: many theories
- Heuristic guidance
  - Triggers to instantiate quantifiers
  - Hierachical reasoning, many rules
- Large proofs, Save & Load whole proof

KeY until Version 2.0



(Sequent) Calculus

Proof Confluent

Destructive

1. Incomplete search
2. Global fairness
3. Instance-based TP
4. Model Evolution

Non-destructive

1. Ground tableaux
2. Sentence tableaux
3. Delayed closure
4. Incremental closure

Not Proof Confluent

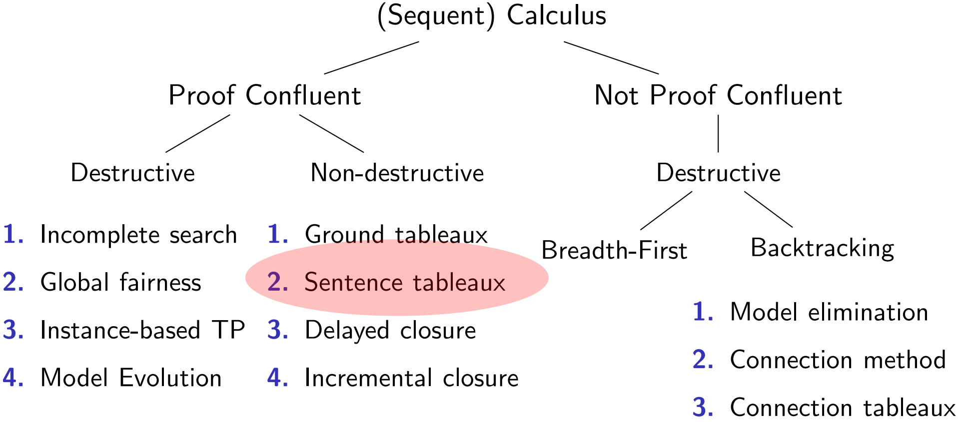
Destructive

Breadth-First

Backtracking

1. Model elimination
2. Connection method
3. Connection tableaux

KeY since Version 2.0



(Sequent) Calculus

Proof Confluent

Not Proof Confluent

Destructive

1. Incomplete search
2. Global fairness
3. Instance-based TP
4. Model Evolution

Non-destructive

1. Ground tableaux
2. Sentence tableaux
3. Delayed closure
4. Incremental closure

Destructive

Breadth-First

Backtracking

1. Model elimination
2. Connection method
3. Connection tableaux

Some References

Peter Baumgartner.

Hyper Tableaux — The Next Generation. In H. de Swart, ed., *Proc. Automated Reasoning with Analytic Tableaux and Related Methods, Oosterwijk, The Netherlands*, vol. 1397 in LNCS, pp. 60–76. Springer, 1998.

Peter Baumgartner and Cesare Tinelli.

The model evolution calculus. In F. Baader, ed., *19th Intl. Conf. on Automated Deduction, Miami Beach, FL, USA*, vol. 2741 of *LNCS*, pp. 350–364. Springer, 2003.

Bernhard Beckert.

Depth-first proof search without backtracking for free-variable clausal tableaux. *Journal of Symbolic Computation*, 36:117–138, 2003.

Jean-Paul Billon.

The disconnection method: a confluent integration of unification in the analytic framework. In P. Miglioli, U. Moscato, D. Mundici, and M. Ornaghi, ed., *Proc. Automated Reasoning with Analytic Tableaux and Related Methods, Terrassini, Italy*, vol. 1071 in LNCS, pp. 110–126. Springer, 1996.

S. de Gouw, J. Rot, F. de Boer, R. Bubel, R. Hähnle.

OpenJDK's `java.utils.Collection.sort()` is broken: The good, the bad and the worst case. In D. Kroening and C. Pasareanu, eds., *Proc. 27th Intl. Conf. on Computer Aided Verification (CAV), San Francisco, LNCS, Springer, 2015.*

Martin Giese.

Incremental closure of free variable tableaux. In R. Goré, A. Leitsch, & T. Nipkow, eds., *Proc. Intl. Joint Conf. on Automated Reasoning IJCAR, Siena, Italy*, vol. 2083 of *LNCS*, pp. 545–560. Springer, 2001.

Reinhold Letz and Gernot Stenz.

Automated theorem proving proof and model generation with disconnection tableaux. In R. Nieuwenhuis & A. Voronkov, eds., *Proc. Logic for Programming, Artificial Intelligence & Reasoning, Havana, Cuba*, vol. 2250 of *LNCS*, pp. 142–156. Springer, 2001.

Raymond M. Smullyan.

First-Order Logic. Springer-Verlag, New York, 1968.