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
BinarySearch.java
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 $\Sigma$ consists of

- a set $\textit{Func}$ of function symbols with specified number of arguments
- a set $\textit{Pred}$ of predicate symbols with specified number of arguments
- a potentially infinite set $\textit{Var}$ of variables.
Untyped First-Order Logic

Vocabulary
A vocabulary $\Sigma$ consists of
- a set $\text{Func}$ of function symbols with specified number of arguments
- a set $\text{Pred}$ of predicate symbols with specified number of arguments
- a potentially infinite set $\text{Var}$ of variables.

Inductive Definition of Terms
If $f \in \text{Func}$ 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 $\Sigma$ consists of
- a set $\text{Func}$ of function symbols with specified number of arguments
- a set $\text{Pred}$ of predicate symbols with specified number of arguments
- a potentially infinite set $\text{Var}$ of variables.

Inductive Definition of Terms
If $f \in \text{Func}$ with arity $n$ and $t_1, \ldots, t_n$ are terms so is $f(t_1, \ldots, t_n)$.

Inductive Definition of Formulas
If $p \in \text{Pred}$ with arity $n$ and $t_1, \ldots, t_n$ are terms then $p(t_1, \ldots, t_n)$ is an (atomic) formula.
If $x \in \text{Var}$ and $\varphi_1, \varphi_2$ 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 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

\[ \Gamma \Rightarrow \Delta \]

▶ \(\Gamma\), \(\Delta\) finite sets of first-order formulas

▶ Positive formulation (prove validity)

▶ Structural rules (ACI) implicit: classical validity, efficiency
Sequents: Syntax & Semantics

Syntax

\[ \psi_1, \ldots, \psi_m \implies \varphi_1, \ldots, \varphi_n \]

where the \( \varphi_i, \psi_i \) are formulae
Sequents: Syntax & Semantics

Syntax

\[ \psi_1, \ldots, \psi_m \Rightarrow \varphi_1, \ldots, \varphi_n \]

where the \( \varphi_i, \psi_i \) are formulae

Semantics

Same as the formula

\[(\forall \bar{x})(\left(\psi_1 \land \cdots \land \psi_m\right) \rightarrow \left(\varphi_1 \lor \cdots \lor \varphi_n\right))\]

where \( \bar{x} = \text{Free}(\{\psi_1, \ldots, \psi_m, \varphi_1, \ldots, \varphi_n\}) \)
Sequent Rule Schemata I

**Rule schemata** where $\Gamma$, $\Delta$ are metavariables for sets of formulae, $\varphi$, $\psi$ for formulae

**andRight**

\[
\frac{
\Gamma \Rightarrow \varphi_1, \Delta \quad \cdots 
\Gamma \Rightarrow \varphi_n, \Delta
}{
\Gamma \Rightarrow \varphi_1 \land \cdots \land \varphi_n, \Delta
}
\]

**andLeft**

\[
\frac{
\Gamma, \varphi_1, \ldots, \varphi_n \Rightarrow \Delta
}{
\Gamma, \varphi_1 \land \cdots \land \varphi_n \Rightarrow \Delta
}
\]

**orLeft**

\[
\frac{
\Gamma, \varphi_1 \Rightarrow \Delta \quad \cdots 
\Gamma, \varphi_n \Rightarrow \Delta
}{
\Gamma, \varphi_1 \lor \cdots \lor \varphi_n \Rightarrow \Delta
}
\]

**orRight**

\[
\frac{
\Gamma \Rightarrow \varphi_1, \ldots, \varphi_n, \Delta
}{
\Gamma \Rightarrow \varphi_1 \lor \cdots \lor \varphi_n, \Delta
}
\]
## Sequent Rule Schemata II

<table>
<thead>
<tr>
<th>Rule</th>
<th>Premise</th>
<th>Conclusion</th>
</tr>
</thead>
<tbody>
<tr>
<td>allRight</td>
<td>( \Gamma \models <a href="%5Cvarphi">x/f(\overline{X})</a>, \Delta )</td>
<td>( \Gamma \models (\forall x)\varphi, \Delta )</td>
</tr>
<tr>
<td>exLeft</td>
<td>( \Gamma, <a href="%5Cvarphi">x/f(\overline{X})</a> \models \Delta )</td>
<td>( \Gamma, (\exists x)\varphi \models \Delta )</td>
</tr>
<tr>
<td>allLeft</td>
<td>( \Gamma, (\forall x)\varphi, <a href="%5Cvarphi">x/X</a> \models \Delta )</td>
<td>( \Gamma, (\forall x)\varphi \models \Delta )</td>
</tr>
<tr>
<td>exRight</td>
<td>( \Gamma \models (\exists x)\varphi, <a href="%5Cvarphi">x/X</a>, \Delta )</td>
<td>( \Gamma \models (\exists x)\varphi, \Delta )</td>
</tr>
</tbody>
</table>

- \( f \) new function symbol of arity \( |\overline{X}| \), where \( \overline{X} = \text{Free}((\forall x)\varphi) \)
- \( X \) new variable symbol
Sequent Rule Schemata II

allRight \[
\frac{\Gamma \Rightarrow [x/f(\overline{X})](\varphi), \Delta}{\Gamma \Rightarrow (\forall x)\varphi, \Delta}
\]

allLeft \[
\frac{\Gamma, (\forall x)\varphi \Rightarrow \Delta}{\Gamma \Rightarrow \Delta}
\]

exLeft \[
\frac{\Gamma, [x/f(\overline{X})](\varphi) \Rightarrow \Delta}{\Gamma, (\exists x)\varphi \Rightarrow \Delta}
\]

exRight \[
\frac{\Gamma \Rightarrow (\exists x)\varphi, [x/X](\varphi), \Delta}{\Gamma \Rightarrow (\exists x)\varphi, \Delta}
\]

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

\( X \) new variable symbol

\[ closeU \quad \frac{\sigma}{\Gamma, \psi \Rightarrow \varphi, \Delta} \]

\( \sigma \) is MGU of \( \psi, \varphi \) and is applied to whole sequent proof

\[ closeFalse \quad \frac{\{}\}{\Gamma, \text{false} \Rightarrow \Delta} \]

\[ closeTrue \quad \frac{\{}\}{\Gamma \Rightarrow \text{true}, \Delta} \]
**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 $\sigma$, then $P'$ is obtained from $\sigma(P)$.
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.

Notation: $P \preceq P'$

A sequent proof tree (with root node $S$) whose leaves are all substitutions (“closed”) is called sequent proof (for $S$).
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 $\sigma$, then $P'$ is obtained from $\sigma(P)$.

Notation: $P \preceq P'$
**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 $\sigma$, 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:

1. $$(\forall x)(P_1 \lor \cdots \lor P_m) \in \Gamma$$
2. $$(\exists y)(Q_1 \land \cdots \land Q_n) \in \Delta$$

where $P_i$ and $Q_j$ are literals.
\((\forall x)(p(x) \lor q(x)) \Rightarrow p(a), p(b), q(b)\)
\( C, q(X) \implies p(a), p(b), q(b) \)

\( C, p(X) \implies p(a), p(b), q(b) \)

\( (\forall x)(p(x) \lor q(x)) \implies p(a), p(b), q(b) \)

\( C \)
Dynamic Free Variable Sequent Proof Construction

\[
\forall x (p(x) \lor q(x)) \Rightarrow p(a), p(b), q(b)
\]

\[
C, q(a) \Rightarrow p(a), p(b), q(b)
\]

\[
\{X \mapsto a\}
\]

\[
C, p(a) \Rightarrow p(a), p(b), q(b)
\]

\[
(\forall x)(p(x) \lor q(x)) \Rightarrow p(a), p(b), q(b)
\]

\[
C
\]
\( \forall x ) ( p(x) \lor q(x)) \Rightarrow p(a), p(b), q(b) \\
C, q(a) \Rightarrow p(a), p(b), q(b) \\
C, q(X'), q(a) \Rightarrow p(a), p(b), q(b) \\
C, p(X'), p(a) \Rightarrow p(a), p(b), q(b) \\
\{ X \mapsto a \} \\
C, p(a) \Rightarrow p(a), p(b), q(b) \\
C, q(a) \Rightarrow p(a), p(b), q(b) \\
C, q(a) \Rightarrow p(a), p(b), q(b) \\
(\forall x ) ( p(x) \lor q(x)) \Rightarrow p(a), p(b), q(b) \\
C
Dynamic Free Variable Sequent Proof Construction

\[
\begin{align*}
C, q(b), q(a) & \Rightarrow p(a), p(b), q(b) & C, p(b), p(a) & \Rightarrow p(a), p(b), q(b) \\
C, q(a) & \Rightarrow p(a), p(b), q(b) & \{X \mapsto a\} & \Rightarrow \{X' \mapsto b\} \\
C, p(a) & \Rightarrow p(a), p(b), q(b) \\
(\forall x)(p(x) \lor q(x)) & \Rightarrow p(a), p(b), q(b) & C
\end{align*}
\]
Dynamic Free Variable Sequent Proof Construction

\[
\begin{align*}
\forall x \left( p(x) \lor q(x) \right) \\
C, q(b), q(a) \Rightarrow p(a), p(b), q(b) \quad & \quad C, p(b), p(a) \Rightarrow p(a), p(b), q(b) \\
C, q(a) \Rightarrow p(a), p(b), q(b) \quad & \quad \{X \mapsto a\} \\
C, p(a) \Rightarrow p(a), p(b), q(b) \quad & \quad \{X' \mapsto b\} \\
(\forall x)(p(x) \lor q(x)) \Rightarrow p(a), p(b), q(b) \quad & \quad C
\end{align*}
\]
Completeness merely guarantees **existence** of sequent proof: Proof (search) procedure needed to find it!
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)?
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
### 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**.

---

**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**.

---

**CADE Tutorial: The KeY Calculus:**

TU Darmstadt, KIT
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.
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
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 efficiency in implementation

- leftmost-open-first
- rightmost-open-first
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 $C_1$, $C_2$, $C_3$, $C_4$, ...

$$(\forall u)p(u, a), (\forall y)(p(y, b) \lor r(y)), q(b), r(a) \Rightarrow (\exists x)(p(a, x) \land q(x)), (\exists w)p(b, w)$$

is valid, but ...
From Calculus to Proof Procedure Cont’d
Closure vs. Extension

\[
\begin{align*}
\text{C}_1 \quad & (\forall u) p(u, a), \\
\text{C}_4 \quad & (\forall y)(p(y, b) \lor r(y)), \\
& q(b), \\
& r(a) \implies (\exists x)(p(a, x) \land q(x)), \\
\text{C}_2 \quad & (\exists w)p(b, w)
\end{align*}
\]
From Calculus to Proof Procedure Cont’d
Closure vs. Extension

\[ C_1, \ p(U, a), \ C_4, \ q(b), \ r(a) \Rightarrow C_2, C_3 \]
From Calculus to Proof Procedure Cont’d
Closure vs. Extension

\[ C_1, p(U, a), C_4, q(b), r(a) \Rightarrow C_2, q(X), C_3 \]
\[ C_1, p(U, a), C_4, q(b), r(a) \Rightarrow C_2, p(a, X), C_3 \]
\[ C_1, p(U, a), C_4, q(b), r(a) \Rightarrow C_2, C_3 \]
\[ (\forall u)p(u, a), (\forall y)(p(y, b) \lor r(y)), q(b), r(a) \Rightarrow (\exists x)(p(a, x) \land q(x)), (\exists w)p(b, w) \]
From Calculus to Proof Procedure Cont’d
Closure vs. Extension

\[
\begin{align*}
&C_1, p(a, a), C_4, q(b), r(a) \Rightarrow C_2, q(a), C_3 \quad \{X \mapsto a, U \mapsto a\} \\
&C_1, p(a, a), C_4, q(b), r(a) \Rightarrow C_2, p(a, a), C_3 \\
&C_1, p(a, a), C_4, q(b), r(a) \Rightarrow C_2, C_3 \\
&C_1 \\
&(\forall u) p(u, a), (\forall y) (p(y, b) \lor r(y)), q(b), r(a) \Rightarrow (\exists x) (p(a, x) \land q(x)), (\exists w) p(b, w)
\end{align*}
\]
From Calculus to Proof Procedure Cont’d
Closure vs. Extension

\[ C_1, p(a, a), C_4, q(b), r(a) \rightarrow C_2, q(a), C_3, p(b, W) \]

\[ C_1, p(a, a), C_4, q(b), r(a) \rightarrow C_2, q(a), C_3 \quad \{X \mapsto a, U \mapsto a\} \]

\[ C_1, p(a, a), C_4, q(b), r(a) \rightarrow C_2, p(a, a), C_3 \]

\[ C_1, p(a, a), C_4, q(b), r(a) \rightarrow C_2, C_3 \]

\[ (\forall u)p(u, a), (\forall y)(p(y, b) \lor r(y)), q(b), r(a) \rightarrow (\exists x)(p(a, x) \land q(x)), (\exists w)p(b, w) \]
From Calculus to Proof Procedure Cont’d
Closure vs. Extension

\[ C_1, p(a, a), C_4, r(Y), q(b), r(a) \Rightarrow C_2, q(a), C_3, p(b, W) \]

\[ C_1, p(a, a), C_4, p(Y, b), q(b), r(a) \Rightarrow C_2, q(a), C_3, p(b, W) \]

\[ C_1, p(a, a), C_4, q(b), r(a) \Rightarrow C_2, q(a), C_3, p(b, W) \]

\[ C_1, p(a, a), C_4, q(b), r(a) \Rightarrow C_2, q(a), C_3 \quad \{X \mapsto a, U \mapsto a\} \]

\[ C_1, p(a, a), C_4, q(b), r(a) \Rightarrow C_2, p(a, a), C_3 \]

\[ C_1, p(a, a), C_4, q(b), r(a) \Rightarrow C_2, C_3 \]

\[ (\forall u)p(u, a), (\forall y)(p(y, b) \lor r(y)), q(b), r(a) \Rightarrow (\exists x)(p(a, x) \land q(x)), (\exists w)p(b, w) \]
From Calculus to Proof Procedure Cont’d

Closure vs. Extension

\[ C_1, p(a, a), C_4, r(b), q(b), r(a) \implies C_2, q(a), C_3, p(b, b) \]

\[ \{ Y \mapsto b, W \mapsto b \} \]

\[ C_1, p(a, a), C_4, p(b, b), q(b), r(a) \implies C_2, q(a), C_3, p(b, b) \]

\[ C_1, p(a, a), C_4, q(b), r(a) \implies C_2, q(a), C_3, p(b, W) \]

\[ \{ X \mapsto a, U \mapsto a \} \]

\[ C_1, p(a, a), C_4, q(b), r(a) \implies C_2, p(a, a), C_3 \]

\[ C_1, p(a, a), C_4, q(b), r(a) \implies C_2, C_3 \]

\[ (\forall u)p(u, a), (\forall y)(p(y, b) \lor r(y)), q(b), r(a) \implies (\exists x)(p(a, x) \land q(x)), (\exists w)p(b, w) \]
Select Main Formula Used for Extension

Unfair choice can prevent subgoal closure

Example

\[ \vdots \]
\[ p(X''), p(X'), p(X), (\forall x)p(x), q \Rightarrow q \]
\[ p(X'), p(X), (\forall x)p(x), q \Rightarrow q \]
\[ p(X), (\forall x)p(x), q \Rightarrow q \]
\[ (\forall x)p(x), q \Rightarrow q \]
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
Select MGU Used for closeU

Unfair choice among several possible MGUs can prevent closure

Example

\[
\begin{align*}
    p(0), C, p(s(0)), p(s(X_3)) &\Rightarrow p(s(s(0))) & p(0), C, p(s(0)) &\Rightarrow p(X_3), p(s(s(0))) \\
    p(0), C, p(s(0)), p(s(X_2)) &\Rightarrow p(s(s(0))) & p(0), C, p(s(0)) &\Rightarrow p(X_2), p(s(s(0))) \\
    p(0), C, p(s(X_1)) &\Rightarrow p(s(s(0))) & p(0), C &\Rightarrow p(X_1), p(s(s(0))) \\
    p(0), (\forall x)(\neg p(x) \lor p(s(x))) &\Rightarrow p(s(s(0))) & C
\end{align*}
\]
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
closeU and main formula selection can interact subtly
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 $P \leq P'$ 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
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”
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

- Not Proof Confluent
  - Destructive
    - Breadth-First
    - Backtracking
    - 1. Model elimination
    - 2. Connection method
    - 3. Connection tableaux
  - Non-destructive
    - 1. Ground tableaux
    - 2. Sentence tableaux
    - 3. Delayed closure
    - 4. Incremental closure
A Classification of Sequent-Like Calculi

(Sequent) Calculus

Proof Confluent

Non-destructive

Destructive

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

Not Proof Confluent

Destructive

Breadth-First

Backtracking

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

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

Assuming a fair computation rule for main formula selection

1 Ground/Propositional Calculi

Sequents quantifier-free: all MGUs empty \( \Rightarrow \) 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 \(\Rightarrow\) 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)
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
## 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

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

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

Breadth-First

Backtracking
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
### 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
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.

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
  - Destructive
  - Non-destructive

- Not Proof Confluent
  - Destructive
    - Breadth-First
    - Backtracking

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

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

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
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

\[(\forall x) \ldots, (\forall x) \ldots, r(X) \Rightarrow \ldots, r(b) \ (\forall x) \ldots, (\forall x) \ldots, s(X) \Rightarrow \ldots, s(b)\]

\[\Rightarrow (\forall x)(p(x) \lor q(x)), (\forall x)(r(x) \lor s(x)) \Rightarrow p(a), q(a), r(b), s(b)\]
Connection Conditions: Motivation

Example

$$\{ X \mapsto b \}$$

$$(\forall x) \cdots, (\forall x) \cdots, r(b) \Rightarrow \cdots, r(b)$$

$$(\forall x) \cdots, (\forall x) \cdots, s(b) \Rightarrow \cdots, s(b)$$

$$(\forall x)(p(x) \lor q(x)), (\forall x)(r(x) \lor s(x)) \Rightarrow p(a), q(a), r(b), s(b)$$
Connection Conditions: Motivation

Example

\[ \{ X \mapsto b \} \quad \vdash \quad (\forall x) \ldots, (\forall x) \ldots, r(b) \Rightarrow \ldots, r(b) \quad (\forall x) \ldots, (\forall x) \ldots, s(b) \Rightarrow \ldots, s(b) \]

\[ (\forall x)(p(x) \lor q(x)), (\forall x)(r(x) \lor s(x)) \Rightarrow p(a), q(a), r(b), s(b) \]
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.

\[ p \lor q, r \lor s \Rightarrow p, q \]
Properties of Connection Conditions

**Lemma**

*Ground sequents with connection condition are not proof confluent.*

**Proof.**

\[ p \lor q, r \Rightarrow p, q \]

\[ p \lor q, r \lor s \Rightarrow p, q \]

\[ p \lor q, s \Rightarrow p, q \]
Lemma

Ground sequents with connection condition are not proof confluent.

Proof.

\[ p, r \Rightarrow p, q \]

\[ q, r \Rightarrow p, q \]

\[ p \lor q, r \Rightarrow p, q \]

\[ p \lor q, s \Rightarrow p, q \]

\[ p \lor q, r \lor s \Rightarrow p, q \]
Properties of Connection Conditions

Lemma

Ground sequents with connection condition are not proof confluent.

Proof.

\[ p \lor q, r \implies p, q \]
\[ p \lor q, s \implies p, q \]
\[ q, r \implies p, q \]
\[ p \lor q, r \implies p, q \]
\[ p \lor q, r \lor s \implies p, q \]
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
  - Hierarchical reasoning, many rules
- Large proofs, Save & Load whole proof
KeY until Version 2.0

(Sequent) Calculus

Proof Confluent

- Destructive
- Non-destructive

Not Proof Confluent

- Destructive

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

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

Breadth-First

Backtracking

1. Model elimination
2. Connection method
3. Connection tableaux
KeY since Version 2.0

(Sequent) Calculus

Proof Confluent

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

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

Breadth-First

Backtracking
Some References

Peter Baumgartner.

Peter Baumgartner and Cesare Tinelli.

Bernhard Beckert.

Jean-Paul Billon.

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

Martin Giese.

Reinhold Letz and Gernot Stenz.

Raymond M. Smullyan.