COMP 4161
NICTA Advanced Course

Advanced Topics in Software Verification

Simon Winwood, Toby Murray, June Andronick, Gerwin Klein

\[ \lambda \longrightarrow \text{and HOL} \]
Types: 
\[ \tau ::= b \mid \nu \mid \nu :: C \mid \tau \Rightarrow \tau \mid (\tau, \ldots, \tau) K \]
- \( b \in \{\text{bool, int, \ldots}\} \) base types
- \( \nu \in \{\alpha, \beta, \ldots\} \) type variables
- \( K \in \{\text{set, list, \ldots}\} \) type constructors
- \( C \in \{\text{order, linord, \ldots}\} \) type classes

Terms: 
\[ t ::= v \mid c \mid ?v \mid (t \ t) \mid (\lambda x. \ t) \]
- \( v, x \in V, \ c \in C, \ V, C \) sets of names
Types and Terms in Isabelle

Types: \[ \tau ::= b \mid \nu \mid \nu :: C \mid \tau \Rightarrow \tau \mid (\tau, \ldots, \tau) K \]
- \(b \in \{\text{bool, int,} \ldots\}\) base types
- \(\nu \in \{\alpha, \beta, \ldots\}\) type variables
- \(K \in \{\text{set, list,} \ldots\}\) type constructors
- \(C \in \{\text{order, linord,} \ldots\}\) type classes

Terms: \[ t ::= v \mid c \mid ?v \mid (t t) \mid (\lambda x. t) \]
- \(v, x \in V, \quad c \in C, \quad V, C\) sets of names

→ type constructors: construct a new type out of a parameter type.
Example: \(\text{int list}\)
Types and Terms in Isabelle

**Types:**
\[ \tau ::= b \mid '\nu \mid '\nu :: C \mid \tau \Rightarrow \tau \mid (\tau, \ldots, \tau) K \]
- \(b \in \{\text{bool, int, \ldots}\}\) base types
- \(\nu \in \{\alpha, \beta, \ldots\}\) type variables
- \(K \in \{\text{set, list, \ldots}\}\) type constructors
- \(C \in \{\text{order, linord, \ldots}\}\) type classes

**Terms:**
\[ t ::= v \mid c \mid {?v} \mid (t t) \mid (\lambda x. t) \]
- \(v, x \in V, \ c \in C, \ V, C\) sets of names

- **type constructors:** construct a new type out of a parameter type.
  Example: \(\text{int list}\)

- **type classes:** restrict type variables to a class defined by axioms.
  Example: \(\alpha :: \text{order}\)
Types and Terms in Isabelle

Types: \[ \tau ::= b \mid \nu \mid \nu :: C \mid \tau \Rightarrow \tau \mid (\tau, \ldots, \tau) K \]
- \( b \in \{\text{bool, int, } \ldots\} \) base types
- \( \nu \in \{\alpha, \beta, \ldots\} \) type variables
- \( K \in \{\text{set, list, } \ldots\} \) type constructors
- \( C \in \{\text{order, linord, } \ldots\} \) type classes

Terms: \[ t ::= v \mid c \mid ?v \mid (t \ t) \mid (\lambda x. \ t) \]
- \( v, x \in V, \ c \in C, \ V, C \) sets of names

- **type constructors**: construct a new type out of a parameter type.
  - Example: \( \text{int list} \)

- **type classes**: restrict type variables to a class defined by axioms.
  - Example: \( \alpha :: \text{order} \)

- **schematic variables**: variables that can be instantiated.
Type Classes

- similar to Haskell’s type classes, but with semantic properties

\textbf{axclass} order \textless ord

- order\_refl: "\(x \leq x\)"
- order\_trans: "\([x \leq y; y \leq z] \implies x \leq z\)"

\ldots
Type Classes

→ similar to Haskell’s type classes, but with semantic properties
  
  **axclass** order < ord
  
  order_refl: ""x ≤ x"
  
  order_trans: ""[x ≤ y; y ≤ z] → x ≤ z"
  
  ...

→ theorems can be proved in the abstract

  **lemma** order_less_trans: ""∀ x :: order. [x < y; y < z] → x < z""
similar to Haskell’s type classes, but with semantic properties

**axclass** order < ord

- order_refl: ”$x \leq x$”
- order_trans: ”$[x \leq y; y \leq z] \implies x \leq z$”

theorems can be proved in the abstract

**lemma** order_less_trans: ”$\bigwedge x ::'a :: order. [x < y; y < z] \implies x < z$”

can be used for subtyping

**axclass** linorder < order

- linorder_linear: ”$x \leq y \lor y \leq x$”
Type Classes

→ similar to Haskell’s type classes, but with semantic properties

  **axclass** order < ord
  
  order_refl: ”$x \leq x$”
  
  order_trans: ”$[x \leq y; y \leq z] \implies x \leq z$”
  
  …

→ theorems can be proved in the abstract

  **lemma** order_less_trans: ”$\forall x ::'a :: order. [x < y; y < z] \implies x < z$”

→ can be used for subtyping

  **axclass** linorder < order
  
  linorder_linear: ”$x \leq y \lor y \leq x$”

→ can be instantiated

  **instance** nat :: ”$\{\text{order, linorder}\}$” by …
Schematic Variables

\[
\frac{X \quad Y}{X \land Y}
\]

→ \( X \) and \( Y \) must be **instantiated** to apply the rule
Schematic Variables

\[ \frac{X \quad Y}{X \land Y} \]

→ \( X \) and \( Y \) must be **instantiated** to apply the rule

**But:** lemma “\( x + 0 = 0 + x \)”

→ \( x \) is free
→ convention: lemma must be true for all \( x \)
→ **during the proof**, \( x \) must **not** be instantiated
Schematic Variables

\[
\frac{X \quad Y}{X \land Y}
\]

⇒ $X$ and $Y$ must be instantiated to apply the rule

But: lemma “$x + 0 = 0 + x$”

⇒ $x$ is free
⇒ convention: lemma must be true for all $x$
⇒ during the proof, $x$ must not be instantiated

Solution:
Isabelle has free ($x$), bound ($x$), and schematic (?X) variables.

Only schematic variables can be instantiated.

Free converted into schematic after proof is finished.
Higher Order Unification

Unification:
Find substitution $\sigma$ on variables for terms $s, t$ such that $\sigma(s) = \sigma(t)$
Higher Order Unification

Unification:
Find substitution $\sigma$ on variables for terms $s, t$ such that $\sigma(s) = \sigma(t)$

In Isabelle:
Find substitution $\sigma$ on schematic variables such that $\sigma(s) =_{\alpha\beta\eta} \sigma(t)$
Higher Order Unification

Unification:
Find substitution $\sigma$ on variables for terms $s, t$ such that $\sigma(s) = \sigma(t)$

In Isabelle:
Find substitution $\sigma$ on schematic variables such that $\sigma(s) = \alpha\beta\eta \sigma(t)$

Examples:

$\ ?X \land ?Y =_{\alpha\beta\eta} x \land x$

$\ ?P \ x =_{\alpha\beta\eta} x \land x$

$P \ (?f \ x) =_{\alpha\beta\eta} ?Y \ x$
Higher Order Unification

Unification:
Find substitution $\sigma$ on variables for terms $s, t$ such that $\sigma(s) = \sigma(t)$

In Isabelle:
Find substitution $\sigma$ on schematic variables such that $\sigma(s) =_{\alpha\beta\eta} \sigma(t)$

Examples:

\[
\begin{align*}
?X \land ?Y &=_{\alpha\beta\eta} x \land x \quad [?X \leftarrow x, ?Y \leftarrow x] \\
?P \; x &=_{\alpha\beta\eta} x \land x \quad [?P \leftarrow \lambda x. \; x \land x] \\
P \; (?f \; x) &=_{\alpha\beta\eta} ?Y \; x \quad [?f \leftarrow \lambda x. \; x, ?Y \leftarrow P]
\end{align*}
\]

Higher Order: schematic variables can be functions.
Higher Order Unification

→ Unification modulo $\alpha \beta$ (Higher Order Unification) is semi-decidable
Higher Order Unification

- Unification modulo $\alpha\beta$ (Higher Order Unification) is semi-decidable
- Unification modulo $\alpha\beta\eta$ is undecidable
Higher Order Unification

- Unification modulo $\alpha \beta$ (Higher Order Unification) is semi-decidable
- Unification modulo $\alpha \beta \eta$ is undecidable
- Higher Order Unification has possibly infinitely many solutions
Higher Order Unification

- Unification modulo $\alpha\beta$ (Higher Order Unification) is semi-decidable
- Unification modulo $\alpha\beta\eta$ is undecidable
- Higher Order Unification has possibly infinitely many solutions

But:
- Most cases are well-behaved
Higher Order Unification

- Unification modulo $\alpha\beta$ (Higher Order Unification) is semi-decidable
- Unification modulo $\alpha\beta\eta$ is undecidable
- Higher Order Unification has possibly infinitely many solutions

But:
- Most cases are well-behaved
- Important fragments (like Higher Order Patterns) are decidable
Higher Order Unification

→ Unification modulo $\alpha \beta$ (Higher Order Unification) is semi-decidable
→ Unification modulo $\alpha \beta \eta$ is undecidable
→ Higher Order Unification has possibly infinitely many solutions

But:
→ Most cases are well-behaved
→ Important fragments (like Higher Order Patterns) are decidable

Higher Order Pattern:
→ is a term in $\beta$ normal form where
→ each occurrence of a schematic variable is of the form $?f t_1 \ldots t_n$
→ and the $t_1 \ldots t_n$ are $\eta$-convertible into $n$ distinct bound variables
We have learned so far...

- Simply typed lambda calculus: $\lambda \rightarrow$
We have learned so far...

- Simply typed lambda calculus: $\lambda \rightarrow$
- Typing rules for $\lambda \rightarrow$, type variables, type contexts
We have learned so far...

- Simply typed lambda calculus: $\lambda \rightarrow$
- Typing rules for $\lambda \rightarrow$, type variables, type contexts
- $\beta$-reduction in $\lambda \rightarrow$ satisfies subject reduction
We have learned so far...

- Simply typed lambda calculus: $\lambda \rightarrow$
- Typing rules for $\lambda \rightarrow$, type variables, type contexts
- $\beta$-reduction in $\lambda \rightarrow$ satisfies subject reduction
- $\beta$-reduction in $\lambda \rightarrow$ always terminates
We have learned so far...

- Simply typed lambda calculus: $\lambda \to$
- Typing rules for $\lambda \to$, type variables, type contexts
- $\beta$-reduction in $\lambda \to$ satisfies subject reduction
- $\beta$-reduction in $\lambda \to$ always terminates
- Types and terms in Isabelle
PREVIEW: PROOFS IN ISABELLE
General schema:

\textbf{lemma} name: "<goal>"
\textbf{apply} <method>
\textbf{apply} <method>
\ldots
\textbf{done}
Proofs in Isabelle

General schema:

```
lemma name: "<goal>"
apply <method>
apply <method>
...
done
```

→ Sequential application of methods until all **subgoals** are solved.
The Proof State

1. \(\bigwedge x_1 \ldots x_p.\left[ A_1; \ldots; A_n \right] \implies B\)

2. \(\bigwedge y_1 \ldots y_q.\left[ C_1; \ldots; C_m \right] \implies D\)
The Proof State

1. $\forall x_1 \ldots x_p.\left[ A_1; \ldots; A_n \right] \implies B$
2. $\forall y_1 \ldots y_q.\left[ C_1; \ldots; C_m \right] \implies D$

$x_1 \ldots x_p$ Parameters

$A_1 \ldots A_n$ Local assumptions

$B$ Actual (sub)goal
Syntax:

theory MyTh
imports ImpTh₁ ... ImpThₙ
begin
(declarations, definitions, theorems, proofs, ...)*
end

- MyTh: name of theory. Must live in file MyTh.thy
- ImpThᵢ: name of imported theories. Import transitive.
Syntax:

theory MyTh
imports ImpTh₁ ... ImpThₙ
begin
(declarations, definitions, theorems, proofs, ...)*
end

→ MyTh: name of theory. Must live in file MyTh.thy
→ ImpThᵢ: name of imported theories. Import transitive.

Unless you need something special:

theory MyTh imports Main begin ... end
Natural Deduction Rules

\[
\begin{align*}
&\frac{}{A \land B} \text{ conjI} \\
&\frac{}{A \lor B} \frac{}{A \lor B} \text{ disjI1/2} \\
&\frac{}{A \rightarrow B} \text{ impl} \\
&\frac{}{A \land B} \frac{}{C} \text{ conjE} \\
&\frac{}{A \lor B} \frac{}{C} \text{ disjE} \\
&\frac{}{A \rightarrow B} \frac{}{C} \text{ impE}
\end{align*}
\]

For each connective (\&, \lor, etc):

\textit{introduction} and \textit{elimination} rules
### Natural Deduction Rules

<table>
<thead>
<tr>
<th>Rule</th>
<th>Formula</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>conjI</strong></td>
<td>$\frac{A \quad B}{A \land B}$</td>
</tr>
<tr>
<td><strong>conjE</strong></td>
<td>$\frac{A \land B}{C}$</td>
</tr>
<tr>
<td><strong>disjI1/2</strong></td>
<td>$\frac{A \lor B}{A \lor B}$</td>
</tr>
<tr>
<td><strong>disjE</strong></td>
<td>$\frac{A \lor B}{C}$</td>
</tr>
<tr>
<td><strong>impl</strong></td>
<td>$\frac{A \rightarrow B}{C}$</td>
</tr>
</tbody>
</table>

For each connective ($\land$, $\lor$, etc): **introduction** and **elimination** rules
Natural Deduction Rules

\[
\begin{align*}
\frac{A \quad B}{A \land B} & \quad \text{conjI} \\
\frac{A \lor B}{A \lor B} & \quad \text{disjI} \\
\frac{A \rightarrow B}{A \lor B} & \quad \text{disjI} \\
\frac{A \rightarrow B}{A \lor B} & \quad \text{disjI} \\
\frac{A \land B \quad [A ; B] \implies C}{C} & \quad \text{conjE} \\
\frac{A \lor B}{C} & \quad \text{disjE} \\
\frac{A \rightarrow B}{C} & \quad \text{implE}
\end{align*}
\]

For each connective (\(\land, \lor, \) etc):
- introduction and elimination rules
Natural Deduction Rules

\[
\frac{A}{A \land B} \quad \text{conjI} \qquad \frac{A \land B \quad [A; B] \implies C}{C} \quad \text{conjE}
\]

\[
\frac{A}{A \lor B} \quad \frac{B}{A \lor B} \quad \text{disjI1/2}
\]

\[
\frac{A \lor B}{C} \quad \text{disjE}
\]

\[
\frac{\quad \quad \quad \impl}{A \impl B}
\]

\[
\frac{A \impl B}{C} \quad \text{impE}
\]

For each connective (\(\land, \lor, \text{etc}\)): introduction and elimination rules
Natural Deduction Rules

\[
\frac{A \quad B}{A \land B} \quad \text{conjI} \\
\frac{A \quad \land B \quad B}{A \lor B} \quad \text{disjI1/2} \\
\frac{A}{A \rightarrow B} \quad \text{impl} \\
\]

\[
\frac{A \land B \quad [A; B] \rightarrow C}{C} \quad \text{conjE} \\
\frac{A \lor B \quad A \rightarrow C \quad B \rightarrow C}{C} \quad \text{disjE} \\
\frac{A \rightarrow B}{C} \quad \text{impE}
\]

For each connective (\(\land, \lor, \text{ etc})\): introduction and elimination rules
Natural Deduction Rules

\[
\frac{A \quad B}{A \land B} \quad \text{conjI} \\
\frac{A \quad B}{A \lor B} \quad \text{disjI1/2}
\]

\[
\frac{A \land B}{[A; B] \Rightarrow C} \quad \text{conjE}
\]

\[
\frac{A \lor B \quad A \Rightarrow C \quad B \Rightarrow C}{C} \quad \text{disjE}
\]

\[
\frac{A \Rightarrow B}{A \Rightarrow B} \quad \text{impl}
\]

\[
\frac{A \Rightarrow B}{C} \quad \text{impE}
\]

For each connective (\(\land, \lor, \text{etc}\): introduction and elimination rules
Natural Deduction Rules

\[
\begin{align*}
\frac{A \quad B}{A \land B} & \quad \text{conjI} \\
\frac{A \quad B}{A \lor B} & \quad \text{disjI1/2} \\
\frac{A \rightarrow B}{A \rightarrow B} & \quad \text{impl} \\
\frac{A \land B \quad [A; B] \implies C}{C} & \quad \text{conjE} \\
\frac{A \lor B \quad A \implies C \quad B \implies C}{C} & \quad \text{disjE} \\
\frac{A \rightarrow B \quad A \quad B \implies C}{C} & \quad \text{impE}
\end{align*}
\]

For each connective (\(\land\), \(\lor\), etc): introduction and elimination rules
Proof by assumption

**apply** assumption

proves

1. \([B_1; \ldots; B_m] \implies C\)

by unifying \(C\) with one of the \(B_i\)
Proof by assumption

**apply** assumption

proves

1. \([B_1; \ldots; B_m] \implies C\)

by unifying \(C\) with one of the \(B_i\)

There may be more than one matching \(B_i\) and multiple unifiers.

**Backtracking!**

Explicit backtracking command: **back**
Intro rules
decompose formulae to the right of \( \iff \).

apply (rule \textless intro-rule\textgreater )
Intro rules decompose formulae to the right of $\implies$.

apply (rule $\langle$intro-rule$\rangle$)

Intro rule $[A_1; \ldots; A_n] \implies A$ means

$\implies$ To prove $A$ it suffices to show $A_1 \ldots A_n$
Intro rules

Intro rules decompose formulae to the right of $\implies$.

apply (rule <intro-rule>)

Intro rule $[[A_1; \ldots; A_n] \implies A]$ means

$\to$ To prove $A$ it suffices to show $A_1 \ldots A_n$

Applying rule $[[A_1; \ldots; A_n] \implies A$ to subgoal $C$:

$\to$ unify $A$ and $C$

$\to$ replace $C$ with $n$ new subgoals $A_1 \ldots A_n$
Elim rules decompose formulae on the left of $\equiv$.

**apply** (erule `<elim-rule>`)
**Elim rules**

**Elim** rules decompose formulae on the left of $\implies$.

**apply** (erule <elim-rule>)

Elim rule $[A_1; \ldots; A_n] \implies A$ means

$\Rightarrow$ If I know $A_1$ and want to prove $A$ it suffices to show $A_2 \ldots A_n$
Elim rules decompose formulae on the left of $\implies$.

**apply** (erule <elim-rule>)

Elim rule $[A_1; \ldots; A_n] \implies A$ means

$\Rightarrow$ If I know $A_1$ and want to prove $A$ it suffices to show $A_2 \ldots A_n$

Applying rule $[A_1; \ldots; A_n] \implies A$ to subgoal $C$:

Like **rule** but also

$\Rightarrow$ unifies first premise of rule with an assumption

$\Rightarrow$ eliminates that assumption
DEMO
MORE PROOF RULES
Iff, Negation, True and False

\[ A = B \] iffI

\[ A = B \] iffD1

\[ \neg A \] notl

\[ A = B \] iffE

\[ A = B \] iffD2

\[ \neg A \] notE
Iff, Negation, True and False

\[
\frac{A \implies B \quad B \implies A}{A = B} \quad \text{iffl}
\]

\[
\frac{A = B}{C} \quad \text{iffE}
\]

\[
\frac{A = B}{\neg A} \quad \text{notl}
\]

\[
\frac{A = B}{\neg A} \quad \text{notE}
\]

\[
\]
Iff, Negation, True and False

$$A \implies B \quad B \implies A \quad \iff A = B$$  iffI

$$A = B \quad [A \implies B; B \implies A] \implies C \quad \iff E$$

$$A = B \quad \iff D1$$

$$A = B \quad \iff D2$$

$$\neg A \quad \text{notl}$$

$$\neg A \quad P \quad \text{notE}$$
Iff, Negation, True and False

\[
\frac{A \implies B}{A = B} \quad \frac{B \implies A}{A = B} \quad \text{iffl}
\]

\[
\frac{A = B}{A \implies B} \quad \text{iffD1}
\]

\[
\frac{A = B}{A \implies B} \quad \text{iffD2}
\]

\[
\frac{\neg A}{\neg A} \quad \text{notl}
\]

\[
\frac{A = B}{B \implies A} \quad \frac{\neg A}{P} \quad \text{notE}
\]
Iff, Negation, True and False

\[
\frac{A \implies B \quad B \implies A}{A = B} \quad \text{iffl}
\]

\[
\frac{A = B}{A \implies B} \quad \text{iffD1}
\]

\[
\frac{A = B}{A \implies False} \quad \text{notl}
\]

\[
\frac{A \implies B \quad B \implies A}{A = B} \quad \text{iffE}
\]

\[
\frac{A = B}{B \implies A} \quad \text{iffD2}
\]

\[
\frac{\neg A}{P} \quad \text{notE}
\]
Iff, Negation, True and False

\[
\frac{A \implies B \quad B \implies A}{A = B} \quad \text{iffl}
\]

\[
\frac{A = B \quad [A \implies B; B \implies A]}{C} \quad \text{iffE}
\]

\[
\frac{A = B}{A \implies B} \quad \text{iffD1}
\]

\[
\frac{A = B}{B \implies A} \quad \text{iffD2}
\]

\[
\frac{A \implies \text{False}}{\neg A} \quad \text{notl}
\]

\[
\frac{\neg A \quad A}{P} \quad \text{notE}
\]
Iff, Negation, True and False

\[ A \implies B \quad B \implies A \quad \text{iff} \]
\[ A = B \quad \text{iff} \]

\[ A = B \quad \text{iffD1} \]
\[ A \implies B \quad \text{iffD2} \]

\[ A \implies \text{False} \quad \text{notl} \]
\[ \neg A \quad \text{notE} \]

\[ \text{True} \quad \text{TrueEl} \]
\[ \text{False} \quad \text{FalseE} \]
Equality

\[
\begin{align*}
& t = t \quad \text{refl} \\
& s = t \\ \\
& t = s \quad \text{sym} \\ \\
& r = s \\ \\
& s = t \quad \text{trans}
\end{align*}
\]
Equality

\[
\begin{align*}
\text{refl} & : \quad t = t \\
\text{sym} & : \quad s = t \quad t = s \\
\text{trans} & : \quad r = s \quad s = t \quad r = t \\
\text{subst} & : \quad s = t \quad P \quad s \quad t \\
\end{align*}
\]
Equality

\[
\begin{align*}
\text{refl} & : t = t \\
\text{sym} & : \frac{s = t}{t = s} \\
\text{trans} & : \frac{r = s}{s = t}
\end{align*}
\]

\[
\frac{s = t}{P\ t} \quad \text{subst}
\]

Rarely needed explicitly — used implicitly by term rewriting
\[ P = \text{True} \lor P = \text{False} \]
Classical

\[ P = True \lor P = False \]  True-False

\[ P \lor \neg P \]  excluded-middle

\[ \neg A \implies False \]
\[ \neg A \implies A \]  ccontr  classical
\( P = \text{True} \lor P = \text{False} \)  

\[ P \lor \neg P \]  

excluded-middle

\[ \neg A \implies \text{False} \]  
\[ \frac{}{A} \]  

\( A \)  

\( \neg A \implies A \)  

\[ \frac{}{A} \]  

classical

→ excluded-middle, ccontr and classical

not derivable from the other rules.
Classical

\[ P = True \lor P = False \]

\[ P \lor \neg P \] excluded-middle

\[ \neg A \Rightarrow False \]
\[ A \quad \text{ccontr} \]
\[ \neg A \Rightarrow A \quad \text{classical} \]

→ excluded-middle, ccontr and classical
not derivable from the other rules.

→ if we include True-False, they are derivable

They make the logic “classical”, “non-constructive”
Cases

\[ P \lor \neg P \] excluded-middle

is a case distinction on type \textit{bool}
\( P \lor \neg P \) excluded-middle

is a case distinction on type \( \textit{bool} \)

Isabelle can do case distinctions on arbitrary terms:

\texttt{apply (case_tac \textit{term})}
Safe and not so safe

**Safe rules** preserve provability
Safe and not so safe

**Safe rules** preserve provability

conjI, impl, notI, iffI, refl, ccontr, classical, conjE, disjE

\[
\dfrac{A \quad B}{A \land B} \quad \text{conjI}
\]
Safe and not so safe

**Safe rules** preserve provability

- `conjI`, `impl`, `notI`, `iffI`, `refl`, `ccontr`, `classical`, `conjE`, `disjE`

\[
\frac{A \quad B}{A \land B} \quad \text{conjI}
\]

**Unsafe rules** can turn a provable goal into an unprovable one
Safe and not so safe

**Safe rules** preserve provability

- \( \text{conjI}, \text{impl}, \text{notI}, \text{iffI}, \text{refl}, \text{ccontr}, \text{classical}, \text{conjE}, \text{disjE} \)

\[
\frac{A \quad B}{A \land B} \quad \text{conjI}
\]

**Unsafe rules** can turn a provable goal into an unprovable one

- \( \text{disjI1}, \text{disjI2}, \text{impE}, \text{iffD1}, \text{iffD2}, \text{notE} \)

\[
\frac{A}{A \lor B} \quad \text{disjI1}
\]
Safe and not so safe

**Safe rules** preserve provability

conjI, impl, notI,iffI, refl, ccontr, classical, conjE, disjE

\[
\frac{A \quad B}{A \land B} \quad \text{conjI}
\]

**Unsafe rules** can turn a provable goal into an unprovable one

disjI1, disjI2, impE, iffD1, iffD2, notE

\[
\frac{A}{A \lor B} \quad \text{disjI1}
\]

Apply safe rules before unsafe ones