When using `rule`, `erule`, `drule` Isabelle uses a process known as \textit{higher-order unification}.
Isabelle’s use of unification

When using `rule`, `erule`, `drule` Isabelle uses a process known as *higher-order unification*

What is:

- Unification
- ...and specifically, higher-order unification?
Unification

Given two terms \( t, u \) defined over:

- Set of variables
- Constant symbols

can we find a substitution \( \theta \) such that:

\[ t\theta = u\theta \]

for some notion of equivalence or equality?
In first-order unification terms are first-order terms:

\[ t, u, v ::= X | c | f(t_1, \ldots, t_n) \]
In first-order unification terms are first-order terms:

\[ t, u, v ::= X \mid c \mid f(t_1, \ldots, t_n) \]

Equality is syntactic identity

Substitutions are finite functions from variables to terms
In *first-order unification* terms are *first-order terms*:

\[ t, u, v ::= X \mid c \mid f(t_1, \ldots, t_n) \]

Equality is syntactic identity

Substitutions are finite functions from variables to terms

This process may be familiar:

- Part of operational semantics of logic programming (e.g. Prolog)
- Used widely in first-order theorem proving
Example

Suppose + is a function symbol and 5 and 6 are constants

Suppose X and Y are variables
Example

Suppose $+$ is a function symbol and 5 and 6 are constants
Suppose $X$ and $Y$ are variables
Unify:

$$+(+(5, 6), Y) \text{ with } + (X, +(5, 5))$$

Solution

$X \mapsto +((5, 6), +((5, 5)))$ and $Y \mapsto +(5, 5)$
Example

Suppose $+$ is a function symbol and 5 and 6 are constants

Suppose $X$ and $Y$ are variables

Unify:

$$+(+(5, 6), Y) \text{ with } +(X, +(5, 5))$$

Solution $X \mapsto +(5, 6)$ and $Y \mapsto +(5, 5)$
Has many nice properties:

- Decidable
- Most general unifiers exist
- Linear-time algorithm via Martelli and Montanori
Terms in Isabelle are typed $\lambda$-terms
Isabelle’s terms

Terms in Isabelle are typed $\lambda$-terms

First-order unification inappropriate:

- Notion of equality is $\beta(\eta)$-equivalence
- Substitutions are *capture avoiding substitutions* from $\lambda$-calculus
Terms in Isabelle are typed λ-terms

First-order unification inappropriate:

- Notion of equality is $\beta(\eta)$-equivalence
- Substitutions are *capture avoiding substitutions* from λ-calculus

Need higher-order unification...
Properties of higher-order unification

- Unifiability test is undecidable (Goldfarb and Huet)
- When unifiers do exist, most general unifiers need not
- Unifier set may be infinite

Example:

Unify (where \( F \) is a variable of function type and \( c \) is a constant):

\[ F \mapsto \lambda x, c \]

Consider two different solutions:

\[ F \mapsto \lambda x, \lambda x, c \]

Note \((\lambda x, c)\) and \((\lambda x, \lambda x, c)\) are both equivalent to \( c \) (in equational theory of simply-typed \( \lambda \)-calculus)
Properties of higher-order unification

• Unifiability test is undecidable (Goldfarb and Huet)
• When unifiers do exist, most general unifiers need not
• Unifier set may be infinite

Example:

Unify (where $F$ is a variable of function type and $c$ is a constant):

$$F \ c \ \text{and} \ c$$

Consider two different solutions:

$$F \mapsto \lambda x. \ c \ \text{and} \ F \mapsto \lambda x. \ x$$

Note $(\lambda x. \ c)c$ and $(\lambda x. \ x)c$ are both equivalent to $c$ (in equational theory of simply-typed $\lambda$-calculus)
Gerard Huet discovered a semi-decision procedure for higher-order unification in 1970s

Huet’s algorithm:

- Finds unifiers when they exist
- May not terminate if unifiers do not exist
- Generally works well in practice
Gerard Huet discovered a semi-decision procedure for higher-order unification in 1970s

Huet’s algorithm:

- Finds unifiers when they exist
- May not terminate if unifiers do not exist
- Generally works well in practice

Most Isabelle unification problems are pattern unification problems:

- Decidable subfragment
- Discovered by Miller whilst working on λProlog
- Most general unifiers exist
- Efficient algorithms exist for pattern unification (Qian: linear time/space)

Isabelle uses pattern unification to reduce calls to Huet’s algorithm