Higher-order Unification
(a personal perspective)

Tomer Libal

Inria, France

EMCL Workshop 2016
Vienna, February 12
History
History

\[ \vdash \neg Pa, \forall a, \exists x. Px \land \neg Pfx \]
History

How can we proceed?

\[ \vdash \neg P_a, P_{fa}, \exists x.Px \land \neg Pfx \]
History

Herbrand (1930): just apply a finite number of contractions and instantiations

$$\vdash \neg Pa, Pff a, \exists x. Px \land \neg Pfx$$
How can we proceed?

Herbrand (1930): just apply a finite number of contractions and instantiations

\[
\begin{align*}
\vdash P_a, & \neg P_a & \vdash P_{fa}, & \neg P_{fa} & \vdash P_{ffa}, & \neg P_{ffa} \\
& & & & \vdots
\end{align*}
\]

\[
\begin{align*}
\vdash \neg P_a, & P_{ffa}, P_a \wedge \neg P_{fa}, & \neg P_{fa} & \vdash \exists x. P_x \wedge \neg P_{fx} \\
& \vdash \neg P_a, & P_{ffa}, & \neg P_{fa}, \exists x. P_x \wedge \neg P_{fx} \\
& \vdash \neg P_a, & P_{ffa}, \exists x. P_x \wedge \neg P_{fx}, & \exists x. P_x \wedge \neg P_{fx} \\
& \vdash \neg P_a, & P_{ffa}, & \exists x. P_x \wedge \neg P_{fx} \\
& \vdash \neg P_a, & P_{ffa}, & \exists x. P_x \wedge \neg P_{fx}
\end{align*}
\]
History

Herbrand (1930): just apply a finite number of contractions and instantiations

How can we proceed?

Can we automate this? After all, computational logic is about mechanizing logic

⊢ Pa, ¬Pa
⊢ Pfa, ¬Pfa
⊢ Pffa, ¬Pffa

... 

⊢ ¬Pa, Pffa, Pa ∧ ¬Pfa, Pfa ∧ ¬Pffa

⊢ ¬Pa, Pffa, Pa ∧ ¬Pfa, ∃x.Px ∧ ¬Pfx

⊢ ¬Pa, Pffa, ∃x.Px ∧ ¬Pfx, ∃x.Px ∧ ¬Pfx

⊢ ¬Pa, Pffa, ∃x.Px ∧ ¬Pfx

⊢ ¬Pa, Pffa, ∃x.Px ∧ ¬Pfx

∃ : r

∃ : r

con
History

How can we proceed?

Herbrand (1930): just apply a finite number of contractions and instantiations

Can we automate this? After all, computational logic is about mechanizing logic

30 years: Presburger arithmetic, tableaux, heuristics, Davis-Putnam

\[
\begin{align*}
\vdash Pa, \neg Pa & \quad \vdash Pfa, \neg Pfa & \quad \vdash Pffa, \neg Pffa \\
\vdash \neg Pfa, Pffa, Pa \land \neg Pfa, Pfa \land \neg Pffa & \\
\vdash \neg Pa, Pffa, Pa \land \neg Pfa, \exists x.Px \land \neg Pfx & \quad \exists : r \\
\vdash \neg Pa, Pffa, \exists x.Px \land \neg Pfx, \exists x.Px \land \neg Pfx & \quad \exists : r \\
\vdash \neg Pa, Pffa, \exists x.Px \land \neg Pfx & \quad \text{con}
\end{align*}
\]
History

How can we proceed?

Herbrand (1930): just apply a finite number of contractions and instantiations

Can we automate this? After all, computational logic is about mechanizing logic

30 years: Presburger arithmetic, tableaux, heuristics, Davis-Putnam

But, the Herbrand Space is infinite!

\[ \vdash Pa, \neg Pa \quad \vdash Pfa, \neg Pfa \quad \vdash Pffa, \neg Pffa \]

\[ \vdash \neg Pa, Pffa, Pa \wedge \neg Pfa, Pfa \wedge \neg Pffa \]

\[ \vdash \neg Pa, Pffa, Pa \wedge \neg Pfa, \exists x. Px \wedge \neg Pfx \]

\[ \vdash \neg Pa, Pffa, \exists x. Px \wedge \neg Pfx, \exists x. Px \wedge \neg Pfx \]

\[ \vdash \neg Pa, Pffa, \exists x. Px \wedge \neg Pfx \]

\[ \exists : r \]

\[ \exists : r \]
History

How can we proceed?

Herbrand (1930): just apply a finite number of contractions and instantiations

Can we automate this? After all, computational logic is about mechanizing logic

30 years: Presburger arithmetic, tableaux, heuristics, Davis-Putnam

But, the Herbrand Space is infinite!

Prawitz (1960): try my Unification method!

\[
\begin{align*}
\vdash & Pa, \neg Pa \\
\vdash & Pfa, \neg Pfa \\
\vdash & Pffa, \neg Pffa \\
\vdash & \neg Pa, Pffa, Pa \land \neg Pfa, Pfa \land \neg Pffa \\
\vdash & \neg Pa, Pffa, Pa \land \neg Pfa, \exists x. Px \land \neg Pfx \\
\vdash & \neg Pa, Pffa, \exists x. Px \land \neg Pfx, \exists x. Px \land \neg Pfx \\
\vdash & \neg Pa, Pffa, \exists x. Px \land \neg Pfx \\
\end{align*}
\]

\[
\begin{align*}
\exists : r \\
\exists : r \\
\text{con}
\end{align*}
\]
History

How can we proceed?

Herbrand (1930): just apply a finite number of contractions and instantiations

Can we automate this? After all, computational logic is about mechanizing logic

30 years: Presburger arithmetic, tableaux, heuristics, Davis-Putnam

But, the Herbrand Space is infinite!

Prawitz (1960): try my Unification method!

Thanks! It is exactly what we looked for

\[
\begin{align*}
\vdash Pa, \neg Pa & \quad \vdash Pfa, \neg Pfa & \quad \vdash Pffa, \neg Pffa \\
\cdots \\
\vdash \neg Pa, Pffa, Pa \land \neg Pfa, Pfa \land \neg Pffa & \\
\vdash \neg Pa, Pffa, Pa \land \neg Pfa, \exists x. Px \land \neg Pfx & \quad \exists : r \\
\vdash \neg Pa, Pffa, \exists x. Px \land \neg Pfx, \exists x. Px \land \neg Pfx & \quad \exists : r \\
\vdash \neg Pa, Pffa, \exists x. Px \land \neg Pfx & \\
\vdash \neg Pa, Pffa, \exists x. Px \land \neg Pfx & \quad \text{con}
\end{align*}
\]
History

Herbrand (1930): just apply a finite number of contractions and instantiations

Can we automate this? After all, computational logic is about mechanizing logic

30 years: Preßburger arithmetic, tableaux, heuristics, Davis-Putnam

But, the Herbrand Space is infinite!

Prawitz (1960): try my Unification method!

Thanks! It is exactly what we looked for

Herbrand (1930): so why have you waited 30 years? Look at my Property A!

\[
\begin{align*}
\vdash Pa, \neg Pa & \quad \vdash Pfa, \neg Pfa & \quad \vdash Pffa, \neg Pffa \\
\vdash \neg Pa, Pffa, Pa \land \neg Pfa, Pfa \land \neg Pffa & \\
\vdash \neg Pa, Pffa, Pa \land \neg Pfa, \exists x. Px \land \neg Pfx & \\
\vdash \neg Pa, Pffa, \exists x. Px \land \neg Pfx, \exists x. Px \land \neg Pfx & \\
\vdash \neg Pa, Pffa, \exists x. Px \land \neg Pfx & \\
\end{align*}
\]
History

Herbrand (1930): just apply a finite number of contractions and instantiations

Can we automate this? After all, computational logic is about mechanizing logic

30 years: Presburger arithmetic, tableaux, heuristics, Davis-Putnam

But, the Herbrand Space is infinite!

Prawitz (1960): try my Unification method!

Thanks! It is exactly what we looked for

Herbrand (1930): so why have you waited 30 years? Look at my Property A!

In 1962 JA Robinson has read Prawitz’s paper and in January 1965 the Resolution method was published

\[
\begin{align*}
\vdash Pa, \neg Pa & \quad \vdash Pfa, \neg Pfa & \quad \vdash Pffa, \neg Pffa \\
\vdash \vdash \vdash & \quad \vdash \vdash \vdash & \quad \vdash \vdash \vdash \\
\vdash \neg Pa, Pffa, Pa \land \neg Pfa, Pfa \land \neg Pffa \\
\vdash \neg Pa, Pffa, Pa \land \neg Pfa, \exists x. Px \land \neg Pfx \\
\vdash \neg Pa, Pffa, \exists x. Px \land \neg Pfx \\
\vdash \neg Pa, Pffa, \exists x. Px \land \neg Pfx \\
\vdash \neg Pa, Pffa, \exists x. Px \land \neg Pfx \\
\vdash \neg Pa, Pffa, \exists x. Px \land \neg Pfx
\end{align*}
\]
First-order Unification

Applications:
- Automated deduction (Resolution, Tableau, ...)
- Programming languages (Prolog, Constraint-based)
- Type inference (ML, Haskell, ...)
- Linguistics (Unification-based grammars, ...)
- Term rewriting, pattern matching, ...

Algorithms:
- Robinson (1965) - exponential
- Huet (1976) - "almost" linear, infinite terms
- Martelli and Montanari (1982) - linear, relatively efficient
First-order Unification

Applications:
First-order Unification

- Applications:
  - Automated deduction (Resolution, Tableau, …)
  - Term rewriting, pattern matching, …

- Algorithms:
  - Robinson (1965) - exponential
  - Huet (1976) - "almost" linear, infinite terms
  - Martelli and Montanari (1982) - linear, relatively efficient
First-order Unification

- Applications:
  - Automated deduction (Resolution, Tableau, …)
  - Programming languages (Prolog, Constrained-based)
First-order Unification

- Applications:
  - Automated deduction (Resolution, Tableau, …)
  - Programming languages (Prolog, Constrained-based)
  - Type inference (ML, Haskel, …)
First-order Unification

Applications:

- Automated deduction (Resolution, Tableau, …)
- Programming languages (Prolog, Constrained-based)
- Type inference (ML, Haskel, …)
- Linguistics (Unification-based grammars, …)
First-order Unification

- Applications:
  - Automated deduction (Resolution, Tableau, …)
  - Programming languages (Prolog, Constrained-based)
  - Type inference (ML, Haskel, …)
  - Linguistics (Unification-based grammars, …)
  - Term rewriting,
First-order Unification

- Applications:
  - Automated deduction (Resolution, Tableau, ...)
  - Programming languages (Prolog, Constrained-based)
  - Type inference (ML, Haskel, ...)
  - Linguistics (Unification-based grammars, ...)
  - Term rewriting, pattern matching,
First-order Unification

Applications:

- Automated deduction (Resolution, Tableau, ...)
- Programming languages (Prolog, Constrained-based)
- Type inference (ML, Haskel, ...)
- Linguistics (Unification-based grammars, ...)
- Term rewriting, pattern matching, ...
First-order Unification

▶ Applications:
  ▶ Automated deduction (Resolution, Tableau, …)
  ▶ Programming languages (Prolog, Constrained-based)
  ▶ Type inference (ML, Haskel, …)
  ▶ Linguistics (Unification-based grammars, …)
  ▶ Term rewriting, pattern matching, …

▶ Algorithms:
First-order Unification

Applications:
- Automated deduction (Resolution, Tableau, …)
- Programming languages (Prolog, Constrained-based)
- Type inference (ML, Haskel, …)
- Linguistics (Unification-based grammars, …)
- Term rewriting, pattern matching, …

Algorithms:
- Robinson (1965) - exponential
First-order Unification

- Applications:
  - Automated deduction (Resolution, Tableau, …)
  - Programming languages (Prolog, Constrained-based)
  - Type inference (ML, Haskel, …)
  - Linguistics (Unification-based grammars, …)
  - Term rewriting, pattern matching, …

- Algorithms:
  - Robinson (1965) - exponential
  - Huet (1976) - "almost" linear, infinite terms
First-order Unification

- Applications:
  - Automated deduction (Resolution, Tableau, …)
  - Programming languages (Prolog, Constrained-based)
  - Type inference (ML, Haskel, …)
  - Linguistics (Unification-based grammars, …)
  - Term rewriting, pattern matching, …

- Algorithms:
  - Robinson (1965) - exponential
  - Huet (1976) - "almost" linear, infinite terms
  - Martelli and Montanari (1982) - linear, relatively efficient
First-order Unification - Basic Algorithm

Unification problems and their solutions:

\{ t_1 \equiv s_1, \ldots, t_n \equiv s_n \}
First-order Unification - Basic Algorithm

- Unification problems and their solutions:
  - \{t_1 \doteq s_1, \ldots, t_n \doteq s_n\}

- Most general unifier \(\sigma\):
  - \(\forall \theta \exists \delta. \sigma = \theta \circ \delta\)
First-order Unification - Basic Algorithm

- Unification problems and their solutions:
  - $\{t_1 \doteq s_1, \ldots, t_n \doteq s_n\}$
- Most general unifier $\sigma$:
  - $\forall \theta \exists \delta. \sigma = \theta \circ \delta$

\[
\begin{align*}
\{u \doteq u\} \cup S & \quad \text{delete} \quad \{f(v_1, \ldots, v_n) \doteq f(u_1, \ldots, u_n)\} \cup S \\
\{v_1 \doteq u_1, \ldots, v_n \doteq u_n\} \cup S & \quad \text{decomp} \quad \{x \doteq v\} \cup S \\
\sigma(S) & \quad \text{bind}
\end{align*}
\]

Where $x$ does not occur in $v$ and $\sigma = [v/x]$. 
First-order Unification - Basic Algorithm

- Unification problems and their solutions:
  - \( \{ t_1 \doteq s_1, \ldots, t_n \doteq s_n \} \)

- Most general unifier \( \sigma \):
  - \( \forall \theta \exists \delta. \sigma = \theta \circ \delta \)

\[
\begin{align*}
\{ u \doteq u \} \cup S & \quad \text{delete} \\
\{ f(v_1, \ldots, v_n) \doteq f(u_1, \ldots, u_n) \} \cup S & \quad \text{decomp} \\
\{ v_1 \doteq u_1, \ldots, v_n \doteq u_n \} \cup S & \quad \text{bind}
\end{align*}
\]

Where \( x \) does not occur in \( v \) and \( \sigma = [v/x] \).

\[
\{ f x a \doteq f a a \}
\]
First-order Unification - Basic Algorithm

- Unification problems and their solutions:
  - \( \{ t_1 \equiv s_1, \ldots, t_n \equiv s_n \} \)

- Most general unifier \( \sigma \):
  - \( \forall \theta \exists \delta. \sigma = \theta \circ \delta \)

\[
\begin{align*}
\{ u \equiv u \} \cup S & \quad \text{delete} \\
\{ f(v_1, \ldots, v_n) \equiv f(u_1, \ldots, u_n) \} \cup S & \quad \text{decomp} \\
\{ v_1 \equiv u_1, \ldots, v_n \equiv u_n \} \cup S & \quad \text{bind} \\
\end{align*}
\]

Where \( x \) does not occur in \( v \) and \( \sigma = [v/x] \).

\[
\begin{align*}
\{ f x a \equiv f a a \} \\
\{ x \equiv a, a \equiv a \} & \quad \text{decompose}
\end{align*}
\]
First-order Unification - Basic Algorithm

- Unification problems and their solutions:
  - $\{t_1 \doteq s_1, \ldots, t_n \doteq s_n\}$

- Most general unifier $\sigma$:
  - $\forall \theta \exists \delta. \sigma = \theta \circ \delta$

\[
\begin{align*}
\{u \doteq u\} \cup S & \quad \text{delete} \quad \{f(v_1, \ldots, v_n) \doteq f(u_1, \ldots, u_n)\} \cup S & \quad \text{decomp} \quad \{x \doteq v\} \cup S \quad \text{bind} \\
\{v_1 \doteq u_1, \ldots, v_n \doteq u_n\} \cup S & \\
\sigma(S) & \\
\end{align*}
\]

Where $x$ does not occur in $v$ and $\sigma = [v/x]$.

\[
\begin{align*}
\{f x a \doteq f a a\} \\
\{x \doteq a, a \doteq a\} \\
\end{align*}
\]

\textit{decompose}
First-order Unification - Basic Algorithm

- Unification problems and their solutions:
  - \{t_1 \doteq s_1, \ldots, t_n \doteq s_n\}

- Most general unifier \(\sigma\):
  - \(\forall \theta \exists \delta. \sigma = \theta \circ \delta\)

\[
\frac{\{u \doteq u\} \cup S}{S \text{ delete}} \quad \frac{\{f(v_1, \ldots, v_n) \doteq f(u_1, \ldots, u_n)\} \cup S}{\{v_1 \doteq u_1, \ldots, v_n \doteq u_n\} \cup S \text{ decomp}} \quad \frac{\{x \doteq v\} \cup S}{\sigma(S) \text{ bind}}
\]

Where \(x\) does not occur in \(v\) and \(\sigma = [v/x]\).

\[
\begin{align*}
\{f x a \doteq f a a\} \\
\{x \doteq a, a \doteq a\} \quad \text{decompose} \\
\{a \doteq a\} \quad \text{bind}
\end{align*}
\]
First-order Unification - Basic Algorithm

- Unification problems and their solutions:
  - \( \{ t_1 \doteq s_1, \ldots, t_n \doteq s_n \} \)

- Most general unifier \( \sigma \):
  - \( \forall \theta \exists \delta. \sigma = \theta \circ \delta \)

\[
\begin{align*}
\{ u \doteq u \} \cup S & \quad \text{delete} \quad \{ f(v_1, \ldots, v_n) \doteq f(u_1, \ldots, u_n) \} \cup S & \quad \text{decomp} \quad \{ x \doteq v \} \cup S & \quad \text{bind} \\
\{ v_1 \doteq u_1, \ldots, v_n \doteq u_n \} \cup S & & \sigma(S) & \end{align*}
\]

Where \( x \) does not occur in \( v \) and \( \sigma = [v/x] \).

\[
\begin{align*}
\{ f(x) a \doteq f(a a) \} & \quad \text{decompose} \\
\{ x \doteq a, a \doteq a \} & \quad \text{bind} \\
\{ a \doteq a \} & \end{align*}
\]
First-order Unification - Basic Algorithm

- Unification problems and their solutions:
  - \( \{ t_1 \doteq s_1, \ldots, t_n \doteq s_n \} \)

- Most general unifier \( \sigma \):
  - \( \forall \theta \exists \delta. \sigma = \theta \circ \delta \)

\[
\begin{align*}
\{ u \doteq u \} \cup S & \quad \text{delete} \\
\{ f(v_1, \ldots, v_n) \doteq f(u_1, \ldots, u_n) \} \cup S & \quad \text{decomp} \\
\{ v_1 \doteq u_1, \ldots, v_n \doteq u_n \} \cup S & \quad \text{bind} \\
\sigma(S) & \quad \text{bind}
\end{align*}
\]

Where \( x \) does not occur in \( v \) and \( \sigma = [v/x] \).

\[
\{ f x a \doteq f a a \} \\
\{ x \doteq a, a \doteq a \} \\
\{ a \doteq a \} \\
\{ \} \\
\]

\{ decompose, bind, delete \}
First-order Unification - Basic Algorithm

- Unification problems and their solutions:
  - \( \{ t_1 \doteq s_1, \ldots, t_n \doteq s_n \} \)

- Most general unifier \( \sigma \):
  - \( \forall \theta \exists \delta. \sigma = \theta \circ \delta \)

\[
\begin{align*}
\frac{\{u \doteq u\} \cup S}{S} & \quad \text{delete} \\
\frac{\{f(v_1, \ldots, v_n) \doteq f(u_1, \ldots, u_n)\} \cup S}{\{v_1 \doteq u_1, \ldots, v_n \doteq u_n\} \cup S} & \quad \text{decomp} \\
\frac{\{x \doteq v\} \cup S}{\sigma(S)} & \quad \text{bind}
\end{align*}
\]

Where \( x \) does not occur in \( v \) and \( \sigma = [v/x] \).

\[
\begin{align*}
\{ f \ x \ a \ & \doteq f \ a \ a \} \\
\{ x \ & \doteq a, \ a \ & \doteq a \} \\
\{ a \ & \doteq a \} \\
\{ \} \quad \text{delete}
\end{align*}
\]
Higher-order Unification

Applications:
- Automated deduction (Arithmetic, Meta-physics, . . . )
- Programming languages (\(\lambda\)-Prolog)
- Type inference (Coq, dependent types, . . . )
- Linguistics (Ellipsis, . . . )

Term rewriting, Meta-logic, . . .

Example (linguistics):
Assume that "dan likes his wife and george does too".
\[ P(dan) = \text{likes}(\text{dan}, \text{wife-of}(\text{dan})) \]
\[ P \mapsto \lambda z. \text{likes}(\text{dan}, \text{wife-of}(z)) \]
\[ P \mapsto \lambda z. \text{likes}(z, \text{wife-of}(z)) \]
\[ P \mapsto \lambda z. \text{likes}(z, \text{wife-of}(\text{dan})) \]
Higher-order Unification

Applications:

Example (linguistics):
Assume that “dan likes his wife and george does too”.

\[
P(dan) = \text{likes}(dan, \text{wife-of}(dan))
\]

\[
P \mapsto \lambda z. \text{likes}(z, \text{wife-of}(z))
\]

\[
P \mapsto \lambda z. \text{likes}(z, \text{wife-of}(dan))
\]
Higher-order Unification

- Applications:
  - Automated deduction (Arithmetic, Meta-physics,…)
Higher-order Unification

Applications:
- Automated deduction (Arithmetic, Meta-physics,...)
- Programming languages (λ-Prolog)
Higher-order Unification

Applications:

- Automated deduction (Arithmetic, Meta-physics, ...)
- Programming languages ($\lambda$-Prolog)
- Type inference (Coq, dependent types, ...)

Example (linguistics):

Assume that "dan likes his wife and george does too".

\[ P(dan) = \text{likes}(dan, \text{wife-of}(\text{dan})) \]
Higher-order Unification

- Applications:
  - Automated deduction (Arithmetic, Meta-physics, …)
  - Programming languages ($\lambda$-Prolog)
  - Type inference (Coq, dependent types, …)
  - Linguistics (Ellipsis, …)
Higher-order Unification

- Applications:
  - Automated deduction (Arithmetic, Meta-physics, ...)
  - Programming languages ($\lambda$-Prolog)
  - Type inference (Coq, dependent types, ...)
  - Linguistics (Ellipsis, ...)
  - Term rewriting,
Higher-order Unification

Applications:
- Automated deduction (Arithmetic, Meta-physics, ...)
- Programming languages ($\lambda$-Prolog)
- Type inference (Coq, dependent types, ...)
- Linguistics (Ellipsis, ...)
- Term rewriting, Meta-logic,
Higher-order Unification

Applications:
- Automated deduction (Arithmetic, Meta-physics, ...)
- Programming languages ($\lambda$-Prolog)
- Type inference (Coq, dependent types, ...)
- Linguistics (Ellipsis, ...)
- Term rewriting, Meta-logic, ...
Higher-order Unification

- Applications:
  - Automated deduction (Arithmetic, Meta-physics, ...)
  - Programming languages (λ-Prolog)
  - Type inference (Coq, dependent types, ...)
  - Linguistics (Ellipsis, ...)
  - Term rewriting, Meta-logic, ...

- Example (linguistics):
  - Assume that "dan likes his wife and george does too".
Higher-order Unification

- Applications:
  - Automated deduction (Arithmetic, Meta-physics, \ldots)
  - Programming languages (\lambda-Prolog)
  - Type inference (Coq, dependent types, \ldots)
  - Linguistics (Ellipsis, \ldots)
  - Term rewriting, Meta-logic, \ldots

- Example (linguistics):
  - Assume that "dan likes his wife and george does too".
  - \( P(dan) \models \text{likes}(dan, \text{wife-of}(dan)) \)
Higher-order Unification

Applications:
- Automated deduction (Arithmetic, Meta-physics, ...)
- Programming languages (λ-Prolog)
- Type inference (Coq, dependent types, ...)
- Linguistics (Ellipsis, ...)
- Term rewriting, Meta-logic, ...

Example (linguistics):
- Assume that "dan likes his wife and george does too".
- \( P(dan) \overset{\cdot}{=} \text{likes}(dan, \text{wife-of}(dan)) \)
- \( P \mapsto \lambda z. \text{likes}(dan, \text{wife-of}(dan)) \)
Higher-order Unification

▶ Applications:
  ▶ Automated deduction (Arithmetic, Meta-physics, ...)
  ▶ Programming languages (λ-Prolog)
  ▶ Type inference (Coq, dependent types, ...)
  ▶ Linguistics (Ellipsis, ...)
  ▶ Term rewriting, Meta-logic, ...

▶ Example (linguistics):
  ▶ Assume that "dan likes his wife and george does too".
  ▶ \( P(dan) \equiv \text{likes}(dan, \text{wife-of}(dan)) \)
    ▶ \( P \mapsto \lambda z.\text{likes}(dan, \text{wife-of}(dan)) \)
    ▶ \( P \mapsto \lambda z.\text{likes}(dan, \text{wife-of}(z)) \)
Higher-order Unification

- **Applications:**
  - Automated deduction (Arithmetic, Meta-physics, ...)
  - Programming languages ($\lambda$-Prolog)
  - Type inference (Coq, dependent types, ...)
  - Linguistics (Ellipsis, ...)
  - Term rewriting, Meta-logic, ...

- **Example (linguistics):**
  - Assume that "dan likes his wife and george does too".
  - $P(dan) \equiv \text{likes}(dan, \text{wife-of}(dan))$
    - $P \mapsto \lambda z. \text{likes}(dan, \text{wife-of}(\text{dan}))$
    - $P \mapsto \lambda z. \text{likes}(\text{dan}, \text{wife-of}(z))$
    - $P \mapsto \lambda z. \text{likes}(z, \text{wife-of}(z))$
Higher-order Unification

- Applications:
  - Automated deduction (Arithmetic, Meta-physics, ...)
  - Programming languages ($\lambda$-Prolog)
  - Type inference (Coq, dependent types, ...)
  - Linguistics (Ellipsis, ...)
  - Term rewriting, Meta-logic, ...

- Example (linguistics):
  - Assume that "dan likes his wife and george does too".
  - \( P(dan) \stackrel{\cdot}{=} \text{likes}(dan, \text{wife-of}(dan)) \)
    - \( P \mapsto \lambda z. \text{likes}(dan, \text{wife-of}(dan)) \)
    - \( P \mapsto \lambda z. \text{likes}(dan, \text{wife-of}(z)) \)
    - \( P \mapsto \lambda z. \text{likes}(z, \text{wife-of}(z)) \)
    - \( P \mapsto \lambda z. \text{likes}(z, \text{wife-of}(dan)) \)
Higher-order Unification

\{ f x(yb)(yc) \doteq f ab \}
Higher-order Unification

\[
\begin{align*}
\{ f x (y b) (y c) & \doteq f a b c \} \\
\{ x & \doteq a, y b \doteq b, y c \doteq c \}
\end{align*}
\]
Higher-order Unification

\{ fx(yb)(yc) ≡ fabc \}
\{ x ≡ a, yb ≡ b, yc ≡ c \}
\{ a ≡ a, yb ≡ b, yc ≡ c \}

de decompose
bind
Higher-order Unification

\[
\begin{align*}
\{ f x (yb)(yc) &\doteq f abc \} \\
\{ x &\doteq a, yb \doteq b, yc \doteq c \} \\
\{ a &\doteq a, yb \doteq b, yc \doteq c \} \\
\{ yb &\doteq b, yc \doteq c \}
\end{align*}
\]
Higher-order Unification

\{ \text{decompose} \}
\{ f(x(yb)(yc)) \doteq f\ a\ b\ c \}
\{ x \doteq a,\ yb \doteq b,\ yc \doteq c \}
\{ a \doteq a,\ yb \doteq b,\ yc \doteq c \}
\{ yb \doteq b,\ yc \doteq c \}
\{ \text{bind} \}
\{ \text{delete} \}
Higher-order Unification

\{ f x (y b) (y c) \doteq f a b c \} \\
\{ x \doteq a, y b \doteq b, y c \doteq c \} \\
\{ a \doteq a, y b \doteq b, y c \doteq c \} \\
\{ y b \doteq b, y c \doteq c \}

- Jensen and Pietrzykowski (1973): either
  - imitation \( y \mapsto \lambda z. b \)
  - projection \( y \mapsto \lambda z. z \)
Higher-order Unification

\[
\{ f x(yb)(yc) \equiv f abc \}
\]

\[
\{ x \equiv a,yb \equiv b, yc \equiv c \}
\]

\[
\{ a \equiv a,yb \equiv b, yc \equiv c \}
\]

\[
\{ yb \equiv b, yc \equiv c \}
\]

\[
\{ y \equiv \lambda z.b, yb \equiv b, yc \equiv c \}
\]

- Jensen and Pietrzykowski (1973): either
  - imitation \( y \mapsto \lambda z.b \)
  - projection \( y \mapsto \lambda z.z \)
Higher-order Unification

\[
\{ f x(yb)(yc) \doteq f abc \} \\
\{ x \doteq a, yb \doteq b, yc \doteq c \} \\
\{ a \doteq a, yb \doteq b, yc \doteq c \} \\
\{ yb \doteq b, yc \doteq c \}
\]

- decompose
- bind
- delete

\[
\{ y \doteq \lambda z.b, yb \doteq b, yc \doteq c \} \\
\{ y \doteq \lambda z.z, yb \doteq b, yc \doteq c \}
\]

- Jensen and Pietrzykowski (1973): either
  - imitation \( y \mapsto \lambda z.b \)
  - projection \( y \mapsto \lambda z.z \)
Higher-order Unification

\[
\begin{align*}
\{ f \, x \, (y \, b) \, (y \, c) \} & \succ f \, a \, b \, c \\
\{ x \ succ a, \ y \, b \ succ b, \ y \, c \ succ c \} & \succ bind \\
\{ a \ succ a, \ y \, b \ succ b, \ y \, c \ succ c \} & \succ decompose \\
\{ y \, b \ succ b, \ y \, c \ succ c \} & \succ delete \\
\{ y \ succ \lambda \, z \, . \, b, \ y \, b \ succ b, \ y \, c \ succ c \} & \succ imitate \\
\{ b \ succ b, \ b \ succ c \} & \succ project
\end{align*}
\]

- Jensen and Pietrzykowski (1973): either
  - imitation \( y \mapsto \lambda \, z \, . \, b \)
  - projection \( y \mapsto \lambda \, z \, . \, z \)
Higher-order Unification

\{ f x (y b) (y c) \doteq f a b c \} \\
\{ x \doteq a, y b \doteq b, y c \doteq c \} \\
\{ a \doteq a, y b \doteq b, y c \doteq c \} \\
\{ y b \doteq b, y c \doteq c \} \\
\{ y \doteq \lambda z . b, y b \doteq b, y c \doteq c \} \\
\{ b \doteq b, b \doteq c \} \\
\{ b \doteq c \}

Jensen and Pietrzykowski (1973): either

- imitation $y \mapsto \lambda z . b$
- projection $y \mapsto \lambda z . z$
Higher-order Unification

\[
\{ f \ x \ (y \ b) \ (y \ c) \ \vdash \ f \ a \ b \ c \} \\
\{ x \ \vdash \ a, \ y \ b \ \vdash \ b, \ y \ c \ \vdash \ c \} \\
\{ a \ \vdash \ a, \ y \ b \ \vdash \ b, \ y \ c \ \vdash \ c \} \\
\{ y \ b \ \vdash \ b, \ y \ c \ \vdash \ c \} \\
\{ y \ \vdash \ \lambda \ z . \ b, \ y \ b \ \vdash \ b, \ y \ c \ \vdash \ c \} \\
\{ y \ \vdash \ \lambda \ z . \ z, \ y \ b \ \vdash \ b, \ y \ c \ \vdash \ c \} \\
\{ b \ \vdash \ b, \ b \ \vdash \ c \} \\
\{ b \ \vdash \ c \} \\
\]

\[\times\]

- Jensen and Pietrzykowski (1973): either
  - imitation \( y \mapsto \lambda z . b \)
  - projection \( y \mapsto \lambda z . z \)
Higher-order Unification

\[\{ f \cdot x(yb)(yc) \doteq f \cdot abc \} \]
\[\{ x \doteq a, \ yb \doteq b, \ yc \doteq c \} \]
\[\{ a \doteq a, \ yb \doteq b, \ yc \doteq c \} \]
\[\{ yb \doteq b, \ yc \doteq c \} \]

 decompose
 bind
 delete

 imitate

\[\{ y \doteq \lambda z.b, \ yb \doteq b, \ yc \doteq c \} \]
\[\{ b \doteq b, \ b \doteq c \} \]
\[\{ b \doteq c \} \]

 project

\[\{ y \doteq \lambda z.z, \ yb \doteq b, \ yc \doteq c \} \]
\[\{ b \doteq b, \ c \doteq c \} \]

\[\times\]

- Jensen and Pietrzykowski (1973): either
  - imitation \( y \mapsto \lambda z.b \)
  - projection \( y \mapsto \lambda z.z \)
Higher-order Unification

- Jensen and Pietrzykowski (1973): either
  - imitation $y \mapsto \lambda z.b$
  - projection $y \mapsto \lambda z.z$
Higher-order Unification

\{ f x(yb)(yc) \equiv f abc \}
\{ x \equiv a, yb \equiv b, yc \equiv c \}
\{ a \equiv a, yb \equiv b, yc \equiv c \}
\{ yb \equiv b, yc \equiv c \}

imitate

\{ y \equiv \lambda z.b, yb \equiv b, yc \equiv c \}
\{ b \equiv b, b \equiv c \}
\{ b \equiv c \}

\xmark

Jensen and Pietrzykowski (1973): either

- imitation \( y \mapsto \lambda z.b \)
- projection \( y \mapsto \lambda z.z \)
Higher-order Unification - flexible pairs
Higher-order Unification - flexible pairs

- \( xab \equiv y(fa)a \)
Higher-order Unification - flexible pairs

- $xab \doteq y(fa)a$
- $x \mapsto \lambda z_1, z_2. r(fz_1)c; y \mapsto \lambda z_1, z_2. rz_1c$
Higher-order Unification - flexible pairs

- $xab = y(fa)a$
- $x \mapsto \lambda z_1, z_2.r(fz_1)c; y \mapsto \lambda z_1, z_2.rz_1c$
- $x \mapsto \lambda z_1, z_2.c; y \mapsto \lambda z_1, z_2.c$
Higher-order Unification - flexible pairs

- \( xab \doteq y(fa)a \)
- \( x \mapsto \lambda z_1, z_2.r(fz_1)c; y \mapsto \lambda z_1, z_2.rz_1c \)
- \( x \mapsto \lambda z_1, z_2.c; y \mapsto \lambda z_1, z_2.c \)

\[
\begin{align*}
\{ u \doteq u \} \cup S & \quad \text{delete} \\
\{ \lambda x_k.z(x_k) \doteq \lambda x_k.v \} \cup S & \quad \text{bind} \\
\{ \lambda x_k.a(v_n) \doteq \lambda x_k.a(u_n) \} \cup S & \quad \text{decomp} \\
\{ \lambda x_k.y(u_n) \doteq \lambda x_k.b(v_m) \} \cup S & \quad \text{imitate} \\
\{ y \uparrow \eta \doteq t \uparrow \eta; \lambda x_k.y(u_n) \doteq \lambda x_k.b(v_m) \} \cup S & \quad \text{project}
\end{align*}
\]

Where \( a \in \Sigma \) or \( a \in x_k; b \in \Sigma; z \) does not occur in \( v \);
\( \sigma = [\lambda x_k.v/x]; t = \lambda x_n.b(y_m(x_n)); \)
\( s = \lambda x_n.x_i(y_l(x_n)) \) for \( 0 < i \leq n \) and \( l = \tau y(x_i) \).
Higher-order Unification: non-termination

- Enumerates a minimal complete set of unifiers.
- Can be infinite: $x(fa) = f(xa)$:
  - $x \mapsto \lambda z. fnz$ for all $n \geq 0$.
- Second-order unification problem is undecidable (Goldfarb 1981)
- Higher-order unification problem is semi-decidable
Higher-order Unification: non-termination

- Enumerates a minimal complete set of unifiers.

- Second-order unification problem is undecidable (Goldfarb 1981)

- Higher-order unification problem is semi-decidable
Higher-order Unification: non-termination

- Enumerates a minimal complete set of unifiers.
- Can be infinite:
  - \( x(fa) \doteq f(xa) \):
    - \( x \mapsto \lambda z.f^nz \) for all \( n \geq 0 \).
Higher-order Unification: non-termination

- Enumerates a minimal complete set of unifiers.
- Can be infinite:
  - $x(fa) = f(xa)$:
    - $x \mapsto \lambda z.f^nz$ for all $n \geq 0$.
- Second-order unification problem is undecidable (Goldfarb 1981)
Higher-order Unification: non-termination

- Enumerates a minimal complete set of unifiers.
- Can be infinite:
  - $x(fa) \vdash f(xa)$:
    - $x \mapsto \lambda z.f^nz$ for all $n \geq 0$.
- Second-order unification problem is undecidable (Goldfarb 1981)
- Higher-order unification problem is semi-decidable
Higher-order Unification - Overview

- HO unification
  - Improvements
  - Decidable fragments
    - Regularity
      - Infinite
      - Finite
    - Monadic SO
      - Context
      - Dependent types
    - Patterns
      - Ramified types
Higher-order Unification - Overview

- Improvements
- Decidable fragments
  - Regularity
    - Infinite
      - Monadic SO
        - 1988
    - Patterns
      - Ramified types
        - 1991
    - Finite
      - 1996
- Context
  - 2015
- Dependent types
  - 2011
Higher-order Unification - Overview

- HO unification
  - Decidable fragments
    - Regularity
      - Infinite
      - Finite
      - Patterns
      - Ramified types
        - Monadic SO
          - Improved termination
            - Regular tree automata
        - Context
          - 1988
          - 1991
          - 1996
        - Dependent types
          - 2015
          - 2011
        - Extended patterns
Higher-order Unification - Overview

- HO unification
  - Improvements
  - Decidable fragments
    - Regularity
      - Improved termination
        - Regular tree automata
    - Finite
      - Patterns
        - Ramified types
          - Monadic SO
            - 1988
          - Context
            - 2015
          - Dependent types
            - 2011
        - Extended patterns
          - 1991
          - 1996
Cantor’s Theorem
Cantor’s Theorem
Cantor's Theorem

\{ x : x \not\in f(x) \}  

Cantor's diagonal set
Cantor’s Theorem

\[ \{ x : x \not\in f(x) \} \]

Andrews-Miller-Cohen-Pfenning ('84)

\[ \neg \exists f \ x \rightarrow i \rightarrow o \ \forall b \ x \rightarrow o \ \exists a \ x : f(a) = b \]
Cantor's Theorem

\[ \{ x : x \not\in f(x) \} \]

Andrews-Miller-Cohen-Pfenning ('84)

\[ \neg \exists f_i \rightarrow i \rightarrow_o \forall b_i \rightarrow_o \exists a_i : f(a) = b \]

LEO-III HOL Theorem Prover
Cantor's Theorem

\[ \{ x : x \notin f(x) \} \]

Cantor's diagonal set

Andrews-Miller-Cohen-Pfenning ('84)
\[
\neg \exists f \rightarrow i \rightarrow o \forall b \rightarrow o \exists a \rightarrow o : f(a) = b
\]

\[ \lambda z. \neg f(z, z) \]

LEO-III HOL Theorem Prover
Cantor's Theorem

\[ \{ x : x \not\in f(x) \} \]

Andrews-Miller-Cohen-Pfenning ('84)

\[ \neg \exists f \circ i \rightarrow o \forall b \circ i \rightarrow o \exists a : f(a) = b \]

\[ \lambda z. \neg f(z, z) \]

LEO-III HOL Theorem Prover
Cantor's Theorem

Andrews-Miller-Cohen-Pfenning ('84)
\[ \neg \exists f_i \rightarrow i \rightarrow_o \forall b_i \rightarrow_o \exists a_i : f(a) = b \]

\begin{align*}
\lambda z. \neg f(z, z) & \quad \text{LEO-III HOL Theorem Prover} \\
\text{Huet's pre-unification procedure ('75)} & \\
\end{align*}

Cantor's diagonal set

\{ x : x \not\in f(x) \}
Cantor's Theorem

\[
\neg \exists f_i : i \rightarrow o \forall b_i : o \exists a_i : f(a) = b
\]

Andrews-Miller-Cohen-Pfenning ('84)

Cantor's diagonal set

\[
\lambda z. \neg f(z, z)
\]

Huet's pre-unification procedure ('75)

Semi-decidable
Forcing termination

- Search space is pruned to be finite.

- Several problems:
  - Incomplete - bound must be big enough.
  - Inefficient - bound must be as small as possible.
  - Some problems cannot have bounds.

- This part: a second-order pre-unification procedure.
  - Sound and complete.
  - Same complexity as Huet's pre-unification procedure.
  - Terminates on more problems than Huet's, including all problems generated by LEO-III for Cantor's theorem.
Forcing termination

- Search space is pruned to be finite.
Forcing termination

- Search space is pruned to be finite.

\[
x \rightarrow \lambda z. f^4 z \rightarrow \ldots \rightarrow \lambda z. f^7 z \rightarrow \lambda z. f^9 z
\]
Forcing termination

- Search space is pruned to be finite.

\[ x \leftarrow \lambda z. f^4 z \quad \ldots \]
\[ \lambda z. f^7 z \]
\[ \lambda z. f^9 z \]
Forcing termination

- Search space is pruned to be finite.

- Several problems:
  - Incomplete - bound must be big enough.

- Inefficient - bound must be as small as possible.

- Some problems cannot have bounds.

This part: a second-order pre-unification procedure.
- Sound and complete.
- Same complexity as Huet's pre-unification procedure.
- Terminates on more problems than Huet's, including all problems generated by LEO-III for Cantor's theorem.
Forcing termination

- Search space is prunned to be finite.
- Several problems:
  - Incomplete - bound must be big enough.

\[
\begin{align*}
\lambda z. f^4 z & \rightarrow \ldots \\
\lambda z. f^7 z & \rightarrow x \\
\lambda z. f^9 z & \rightarrow \lambda z. f^7 z
\end{align*}
\]
Forcing termination

- Search space is prunned to be finite.
- Several problems:
  - Incomplete - bound must be big enough.
  - Inefficient - bound must be as small as possible.

Some problems cannot have bounds.

This part: a second-order pre-unification procedure.

Sound and complete.

Same complexity as Huet's pre-unification procedure.

Terminates on more problems than Huet's,

including all problems generated by LEO-III

for Cantor's theorem.
Forcing termination

- Search space is prunned to be finite.
- Several problems:
  - Incomplete - bound must be big enough.
  - Inefficient - bound must be as small as possible.
  - Some problems cannot have bounds.

\[ \lambda z. f^4 z \quad \ldots \]
\[ \lambda z. f^7 z \]
\[ \lambda z. f^9 z \]
Forcing termination

- Search space is prunned to be finite.
- Several problems:
  - Incomplete - bound must be big enough.
  - Inefficient - bound must be as small as possible.
  - Some problems cannot have bounds.
- This part: a second-order pre-unification procedure.
  - Sound and complete.
  - Same complexity as Huet’s pre-unification procedure.
  - Terminates on more problems than Huet’s.
    - including all problems generated by LEO-III for Cantor’s theorem.
Non-termination

\[ x_0 y \vdash \neg x_0 y \]
Non-termination

\[ x_0 \doteq \lambda z. z, \quad x_0 y \doteq \neg x_0 y \quad x_0 \doteq \lambda z. \neg x_1 z, \quad x_0 y \doteq \neg x_0 y \]
Non-termination

\[ x_0 \equiv \lambda z. z, \ x_0 y \equiv \neg x_0 y \]

\[ x_0 \equiv \lambda z. \neg x_1 z, \ x_0 y \equiv \neg x_0 y \]

\[ \neg x_1 y \equiv \neg \neg x_1 y \]
Non-termination

\[ x_0 \doteq \lambda z.z, \quad x_0 y \doteq \neg x_0 y \]

\[ y \doteq \neg y \]

\[ x_0 \doteq \lambda z. \neg x_1 z, \quad x_0 y \doteq \neg x_0 y \]

\[ \neg x_1 y \doteq \neg \neg x_1 y \]

\[ x_1 y \doteq \neg x_1 y \]
Non-termination

\[ x_0 y \vdash \neg x_0 y \]
\[ y \vdash \neg y \]
\[ x_1 y \vdash \neg x_1 y \]
Non-termination

\[ x_0 y \stackrel{\vdash}{\Rightarrow} \neg x_0 y \]

\( \mathcal{P}(1) \xrightarrow{\Rightarrow} \mathcal{I}(1) \)

\( y \vdash \neg y \quad x_1 y \vdash \neg x_1 y \)
Non-termination

\[ x_0 y \doteq -x_0 y \]
\[ \Psi(1) \quad \Gamma(1) \]
\[ y \doteq -y \quad x_1 y \doteq -x_1 y \]
\[ \Psi(2) \quad \Gamma(2) \]
\[ y \doteq -y \quad x_2 y \doteq -x_2 y \]
Non-termination

\[ x_0 y \vdash \lnot x_0 y \]
\[ \mathcal{P}(1) \quad \mathcal{I}(1) \]
\[ y \vdash \lnot y \quad x_1 y \vdash \lnot x_1 y \]
\[ \mathcal{P}(2) \quad \mathcal{I}(2) \]
\[ x_2 y \vdash \lnot x_2 y \]
Non-termination

\[
x_0 y \vdash \neg x_0 y
\]

\[\mathfrak{P}(1) \quad \mathfrak{I}(1)\]

\[
y \vdash \neg y \\
x_1 y \vdash \neg x_1 y
\]

\[\mathfrak{P}(2) \quad \mathfrak{I}(2)\]

\[
y \vdash \neg y \\
x_2 y \vdash \neg x_2 y
\]
Non-termination

- Semi-decidable: possible non-termination only if not unifiable.
Non-termination

$\overline{x_0} y \equiv \neg x_0 y$

$\exists (1) \quad \exists (1)$

$y \equiv \neg y \quad x_1 y \equiv \neg x_1 y$

$\exists (2) \quad \exists (2)$

$y \equiv \neg y \quad x_2 y \equiv \neg x_2 y$

- Semi-decidable: possible non-termination only if not unifiable.
- Levy (’98): possible non-termination only if we can encounter cycles.
Non-termination

- Semi-decidable: possible non-termination only if not unifiable.
- Levy ('98): possible non-termination only if we can encounter cycles.
- Lemma 1: $e$ is unifiable iff $\exists i > 0 \mathcal{P}(i)$ is unifiable.
Non-termination

Semi-decidable: possible non-termination only if not unifiable.

Levy ('98): possible non-termination only if we can encounter cycles.

Lemma 1: $e$ is unifiable iff $\exists i > 0 \mathcal{P}(i)$ is unifiable.

Lemma 2: $\forall i, j > 0 \mathcal{P}(i)$ is unifiable if $\mathcal{P}(j)$ is.
Cyclic equations - monadic signature

\[ x_0 t \doteq C(x_0 s) \]
Cyclic equations - monadic signature

Theorem: $e$ is unifiable iff $\exists 0 \leq i \leq m$ s.t. $P(i)$ is unifiable.

Corollary: Unification over monadic "cyclic equations" is decidable. (Farmer '88 for full monadic SOU)
Cyclic equations - monadic signature

\[ x_0 t \doteq C(x_0 s) \]

\[ x_0 t \doteq C(x_0 s) \]

\[ \mathcal{P}(m) \]

\[ \mathcal{I}(m) \]

\[ t \doteq C(s) \]

\[ x_m t \doteq C(x_m s) \]

\[ \mathcal{P}(2m) \]

\[ \mathcal{I}(2m) \]

\[ t \doteq C(s) \]

\[ x_{2m} t \doteq C(x_{2m} s) \]
Cyclic equations - monadic signature

\[ x_0 t \overset{\cdot}{=} C(x_0 s) \]

\[ x_0 t \overset{\cdot}{=} C(x_0 s) \]
\[ \mathcal{P}(m) \overset{\cdot}{\longrightarrow} \mathcal{I}(m) \]
\[ t \overset{\cdot}{=} C(s) \]
\[ x_m t \overset{\cdot}{=} C(x_m s) \]
\[ \mathcal{P}(2m) \overset{\cdot}{\longrightarrow} \mathcal{I}(2m) \]
\[ t \overset{\cdot}{=} C(s) \]
\[ x_{2m} t \overset{\cdot}{=} C(x_{2m} s) \]

Lemma 1  Lemma 2
Cyclic equations - monadic signature

\[ \begin{align*}
  x_0 t & \doteq C(x_0 s) \\
  x_0 t & \doteq C(x_0 s) \\
  \mathcal{P}(m) & \xrightarrow{\mathcal{J}(m)} \\
  t & \doteq C(s) \\
  x_m t & \doteq C(x_m s) \\
  \mathcal{P}(2m) & \xrightarrow{\mathcal{J}(2m)} \\
  t & \doteq C(s) \\
  x_{2m} t & \doteq C(x_{2m} s)
\end{align*} \]

Lemma 1  Lemma 2

\[\checkmark\]
Theorem: \( e \) is unifiable iff \( \exists 0 \leq i \leq m \) s.t. \( P(i) \) is unifiable.

Corollary: Unification over monadic "cyclic equations" is decidable. (Farmer '88 for full monadic SOU)
Cyclic equations - monadic signature

\[ x_0 t \doteq C(x_0 s) \]
\[ x_0 t \doteq C(x_0 s) \]
\[ \mathcal{P}(m) \quad \mathcal{I}(m) \]
\[ t \doteq C(s) \quad x_m t \doteq C(x_m s) \]
\[ \mathcal{P}(2m) \quad \mathcal{I}(2m) \]
\[ t \doteq C(s) \quad x_{2m} t \doteq C(x_{2m} s) \]

Lemma 1     Lemma 2

✓     ✓
Cyclic equations - monadic signature

\[
x_0 t \doteq C(x_0 s)
\]
\[
x_0 t \doteq C(x_0 s)
\]
\[
\mathcal{P}(m) \quad \mathcal{I}(m)
\]
\[
t \doteq C(s) \quad x_m t \doteq C(x_m s)
\]
\[
\mathcal{P}(2m) \quad \mathcal{I}(2m)
\]
\[
t \doteq C(s) \quad x_{2m} t \doteq C(x_{2m} s)
\]

- **Lemma 1**
- **Lemma 2**

- **Theorem:** \( e \) is unifiable iff \( \exists 0 \leq i \leq m \) s.t. \( \mathcal{P}(i) \) is unifiable.
Theorem: $e$ is unifiable iff $\exists 0 \leq i \leq m \text{ s.t. } \mathcal{P}(i) \text{ is unifiable.}$
Cyclic equations - monadic signature

\[ x_0 t \doteq C(x_0 s) \]

\[ x_0 t \doteq C(x_0 s) \]

\[ \mathcal{P}(m) \rightarrow \mathcal{T}(m) \]

\[ t \doteq C(s) \quad x_m t \doteq C(x_m s) \]

\[ \mathcal{P}(2m) \rightarrow \mathcal{T}(2m) \]

\[ t \doteq C(s) \quad x_{2m} t \doteq C(x_{2m} s) \]

Lemma 1

Lemma 2

\[ \checkmark \quad \checkmark \]

Theorem: \( e \) is unifiable iff \( \exists 0 \leq i \leq m \) s.t. \( \mathcal{P}(i) \) is unifiable.

Corollary: Unification over monadic "cyclic equations" is decidable. (Farmer '88 for full monadic SOU)
Non-monadic cyclic equations

\[ x_0(-y_1) \doteq x_0 y_2 \lor y_3 \]
Non-monadic cyclic equations

\[ x_0(-y_1) \equiv x_0 y_2 \lor y_3 \]
\[ \neg y_1 \equiv y_2 \lor y_3 \]
\[ x_1(-y_1) \equiv x_1 y_2 \lor w_1 y_2 \]
\[ w_1 y_1 \equiv y_3 \]
Non-monadic cyclic equations

\[
x_0(-y_1) \equiv x_0 y_2 \lor y_3
\]

\[
\neg y_1 \equiv y_2 \lor y_3 \quad x_1(-y_1) \equiv x_1 y_2 \lor w_1 y_2 \quad w_1 y_1 \equiv y_3
\]
Non-monadic cyclic equations

\[ x_0(\neg y_1) \doteq x_0 y_2 \lor y_3 \]
\[ \exists 0 \leq i \leq 3 \text{ s.t. } P(i) \]
\[ \neg y_1 \doteq y_2 \lor y_3 \]
\[ x_1(\neg y_1) \doteq x_1 y_2 \lor w_1 y_2 \]
\[ w_1 y_1 \doteq y_3 \]
\[ x_2(\neg y_1) \doteq x_2 y_2 \lor w_2 y_2 \]
\[ w_1 y_1 \doteq y_3 + 1 \]
Non-monadic cyclic equations

\[ x_0(-y_1) \doteq x_0y_2 \lor y_3 \]

\[ \neg y_1 \doteq y_2 \lor y_3 \]

**\( \mathfrak{P}(1) \)\**

**\( \mathfrak{J}(1) \)\**

\[ x_1(-y_1) \doteq x_1y_2 \lor w_1y_2 \]

\[ w_1y_1 \doteq y_3 \]

\[ w_1y_1 \doteq y_3 + 1 \]

\[ x_2(-y_1) \doteq x_2y_2 \lor w_2y_2 \]

\[ x \]

\[ \neg y_1 \doteq y_2 \lor w_1y_2 \]
Non-monadic cyclic equations

\[ x_0(-y_1) \equiv x_0 y_2 \lor y_3 \]
\[ \neg y_1 \equiv y_2 \lor y_3 \]

\[ x_1(-y_1) \equiv x_1 y_2 \lor w_1 y_2 \quad w_1 y_1 \equiv y_3 \]
\[ \neg y_1 \equiv y_2 \lor w_1 y_2 \]

\[ x_2(-y_1) \equiv x_2 y_2 \lor w_2 y_2 \quad w_1 y_1 \equiv y_3 + 1 \]
\[ \neg y_1 \equiv y_2 \lor w_2 y_2 \]

\[ x_3(-y_1) \equiv x_3 y_2 \lor w_3 y_2 \quad w_1 y_1 \equiv y_3 + 2 \]
Non-monadic cyclic equations

\[ x_0(-y_1) \equiv x_0 y_2 \lor y_3 \]
\[ \neg y_1 \equiv y_2 \lor y_3 \]
\[ \forall (1) \]

\[ x_1(-y_1) \equiv x_1 y_2 \lor w_1 y_2 \]
\[ w_1 y_1 \equiv y_3 \]
\[ \forall (2) \]

\[ \exists y_2 \lor w_1 y_2 \]
\[ w_1 y_1 \equiv y_3 \]
\[ \forall (3) \]

\[ x_2(-y_1) \equiv x_2 y_2 \lor w_2 y_2 \]
\[ w_1 y_1 \equiv y_3 + 1 \]
\[ \forall (4) \]

\[ \exists y_2 \lor w_2 y_2 \]
\[ w_1 y_1 \equiv y_3 + 1 \]

\[ x_3(-y_1) \equiv x_3 y_2 \lor w_3 y_2 \]
\[ w_1 y_1 \equiv y_3 + 2 \]

**Theorem:** e is unifiable only if \[ \exists 0 \leq i \leq 3 \text{ s.t. } P_i \equiv (i) \] is unifiable.
Non-monadic cyclic equations

\[ x_0(-y_1) \doteq x_0 y_2 \lor y_3 \]

\[ \neg y_1 \doteq y_2 \lor y_3 \]

\[ x_1(-y_1) \doteq x_1 y_2 \lor w_1 y_2 \]

\[ w_1 y_1 \doteq y_3 \]

\[ \neg y_1 \doteq y_2 \lor w_1 y_2 \]

\[ w_1 y_1 \doteq y_3 + 1 \]

\[ x_2(-y_1) \doteq x_2 y_2 \lor w_2 y_2 \]

\[ w_1 y_1 \doteq y_3 + 1 \]

\[ x_3(-y_1) \doteq x_3 y_2 \lor w_3 y_2 \]

\[ w_1 y_1 \doteq y_3 + 2 \]

\[ \neg y_1 \doteq y_2 \lor w_3 y_2 \]

\[ w_1 y_1 \doteq y_3 + 2 \]
Theorem: $e$ is unifiable only if $\exists 0 \leq i \leq 3$ s.t. $P - (i)$ is unifiable.
Non-monadic cyclic equations

\[ x_0(-y_1) = x_0y_2 \lor y_3 \]
\[ \mathcal{P}(1) \]
\[ -y_1 = y_2 \lor y_3 \]
\[ \mathcal{I}(1) \]
\[ x_1(-y_1) = x_1y_2 \lor w_1y_2 \]
\[ \mathcal{P}(2) \]
\[ w_1y_1 = y_3 \]
\[ \mathcal{I}(2) \]
\[ -y_1 = y_2 \lor w_1y_2 \]
\[ \mathcal{P}(3) \]
\[ w_1y_1 = y_3 + 1 \]
\[ \mathcal{I}(3) \]
\[ x_2(-y_1) = x_2y_2 \lor w_2y_2 \]
\[ \mathcal{P}(4) \]
\[ x_3(-y_1) = x_3y_2 \lor w_3y_2 \]
\[ \mathcal{I}(4) \]
\[ w_1y_1 = y_3 + 2 \]

is it regular?
Non-monadic cyclic equations

\[ x_0(\neg y_1) = x_0 y_2 \lor y_3 \]
\[ \mathcal{P}(1) \]
\[ \neg y_1 = y_2 \lor y_3 \]
\[ x_1(\neg y_1) = x_1 y_2 \lor w_1 y_2 \]
\[ \mathcal{J}(1) \]
\[ w_1 y_1 = y_3 \]
\[ \mathcal{J}(2) \]
\[ \mathcal{P}(2) \]
\[ \neg y_1 = y_2 \lor w_1 y_2 \]
\[ x_2(\neg y_1) = x_2 y_2 \lor w_2 y_2 \]
\[ \mathcal{P}(3) \]
\[ w_1 y_1 = y_3 + 1 \]
\[ \mathcal{J}(3) \]
\[ \mathcal{J}(2) \]
\[ \neg y_1 = y_2 \lor w_2 y_2 \]
\[ w_1 y_1 = y_3 + 1 \]
\[ \mathcal{P}(4) \]
\[ x_3(\neg y_1) = x_3 y_2 \lor w_3 y_2 \]
\[ w_1 y_1 = y_3 + 2 \]
\[ \mathcal{J}(4) \]
\[ \neg y_1 = y_2 \lor w_3 y_2 \]
\[ w_1 y_1 = y_3 + 2 \]

Lemma 1  Lemma 2
Non-monadic cyclic equations

\[ x_0(-y_1) \doteq x_0 y_2 \lor y_3 \]
\[ \varphi(1) \]
\[ -y_1 \doteq y_2 \lor y_3 \]
\[ \varphi(2) \]
\[ w_1 y_1 \doteq y_3 \]
\[ \varphi(3) \]
\[ w_1 y_1 \doteq y_3 + 1 \]
\[ \varphi(4) \]
\[ w_1 y_1 \doteq y_3 + 2 \]

Lemma 1  Lemma 2

Theorem: \( \exists 0 \leq i \leq 3 \) s.t. \( P_i - (i) \) is unifiable.
Non-monadic cyclic equations

\[ x_0(y_1) = x_0 y_2 \lor y_3 \]
\[ y_1 = y_2 \lor y_3 \]
\[ x_1(y_1) = x_1 y_2 \lor w_1 y_2 \]
\[ w_1 y_1 = y_3 \]

\[ y_1 = y_2 \lor w_1 y_2 \]
\[ w_1 y_1 = y_3 + 1 \]

\[ x_2(y_1) = x_2 y_2 \lor w_2 y_2 \]
\[ w_1 y_1 = y_3 + 1 \]

\[ y_1 = y_2 \lor w_2 y_2 \]
\[ w_1 y_1 = y_3 + 1 \]

\[ x_3(y_1) = x_3 y_2 \lor w_3 y_2 \]
\[ w_1 y_1 = y_3 + 2 \]

\[ y_1 = y_2 \lor w_3 y_2 \]
\[ w_1 y_1 = y_3 + 2 \]
Non-monadic cyclic equations

\[ x_0(\neg y_1) \doteq x_0 y_2 \lor y_3 \]

\[ \neg y_1 \doteq y_2 \lor y_3 \]

\[ x_1(\neg y_1) \doteq x_1 y_2 \lor w_1 y_2 \]

\[ w_1 y_1 \doteq y_3 \]

\[ \neg y_1 \doteq y_2 \lor w_1 y_2 \]

\[ w_1 y_1 \doteq y_3 + 1 \]

\[ \neg y_1 \doteq y_2 \lor w_2 y_2 \]

\[ w_1 y_1 \doteq y_3 + 1 \]

\[ x_2(\neg y_1) \doteq x_2 y_2 \lor w_2 y_2 \]

\[ \neg y_1 \doteq y_2 \lor w_3 y_2 \]

\[ w_1 y_1 \doteq y_3 + 2 \]

**Lemma 1**

\[ \neg y_1 \doteq y_2 \lor w_3 y_2 \]

\[ w_1 y_1 \doteq y_3 + 2 \]

**Lemma 2**

\[ \exists 0 \leq i \leq 3 \text{ s.t. } P(i) \] is unifiable only if

\[ w_1 y_1 \doteq y_3 + 1 \]
Non-monadic cyclic equations

\[ x_0(-y_1) \doteq x_0 y_2 \lor y_3 \]

\[ \neg y_1 \doteq y_2 \lor y_3 \]

\[ x_1(-y_1) \doteq x_1 y_2 \lor w_1 y_2 \quad \text{w}_1 y_1 \doteq y_3 \]

\[ \neg y_1 \doteq y_2 \lor w_1 y_2 \quad \text{w}_1 y_1 \doteq y_3 \]

\[ x_2(-y_1) \doteq x_2 y_2 \lor w_2 y_2 \quad \text{w}_1 y_1 \doteq y_3 + 1 \]

\[ \neg y_1 \doteq y_2 \lor w_2 y_2 \quad \text{w}_1 y_1 \doteq y_3 + 1 \]

\[ x_3(-y_1) \doteq x_3 y_2 \lor w_3 y_2 \quad \text{w}_1 y_1 \doteq y_3 + 2 \]

\[ \neg y_1 \doteq y_2 \lor w_3 y_2 \quad \text{w}_1 y_1 \doteq y_3 + 2 \]

Lemma 1  Lemma 2

\[ \lor \quad \lor \]
Non-monadic cyclic equations

\[ x_0(\neg y_1) \equiv x_0y_2 \lor y_3 \]

\[ \mathcal{P}(1) \quad \mathcal{J}(1) \]

\[ \neg y_1 \equiv y_2 \lor y_3 \quad x_1(\neg y_1) \equiv x_1y_2 \lor w_1y_2 \quad w_1y_1 \equiv y_3 \]

\[ \mathcal{P}(2) \quad \mathcal{J}(2) \]

\[ \neg y_1 \equiv y_2 \lor w_1y_2 \quad w_1y_1 \equiv y_3 \]

\[ \mathcal{P}^{-}(3) \quad \mathcal{P}(3) \quad \mathcal{J}(3) \]

\[ x_2(\neg y_1) \equiv x_2y_2 \lor w_2y_2 \quad w_1y_1 \equiv y_3 + 1 \]

\[ \mathcal{P}^{-}(4) \quad \mathcal{P}(4) \]

\[ \neg y_1 \equiv y_2 \lor w_2y_2 \quad w_1y_1 \equiv y_3 + 1 \]

\[ x_3(\neg y_1) \equiv x_3y_2 \lor w_3y_2 \quad w_1y_1 \equiv y_3 + 2 \]

Lemma 1  Lemma 2

\[ \checkmark \quad ? \]
Non-monadic cyclic equations

\[ x_0(\neg y_1) \equiv x_0 y_2 \lor y_3 \]
\[ \neg y_1 \equiv y_2 \lor y_3 \]
\[ x_1(\neg y_1) \equiv x_1 y_2 \lor w_1 y_2 \]
\[ w_1 y_1 \equiv y_3 \]
\[ \neg y_1 \equiv y_2 \lor w_1 y_2 \]
\[ w_1 y_1 \equiv y_3 \]
\[ x_2(\neg y_1) \equiv x_2 y_2 \lor w_2 y_2 \]
\[ w_1 y_1 \equiv y_3 + 1 \]
\[ \neg y_1 \equiv y_2 \lor w_2 y_2 \]
\[ w_1 y_1 \equiv y_3 + 1 \]
\[ x_3(\neg y_1) \equiv x_3 y_2 \lor w_3 y_2 \]
\[ w_1 y_1 \equiv y_3 + 2 \]
\[ \neg y_1 \equiv y_2 \lor w_3 y_2 \]
\[ w_1 y_1 \equiv y_3 + 2 \]
Non-monadic cyclic equations

\[ x_0(-y_1) \doteq x_0 y_2 \lor y_3 \]
\[ \mathcal{P}(1) \]
\[ \neg y_1 \doteq y_2 \lor y_3 \]
\[ \mathcal{I}(1) \]
\[ x_1(-y_1) \doteq x_1 y_2 \lor w_1 y_2 \]
\[ \mathcal{P}(2) \]
\[ w_1 y_1 \doteq y_3 \]
\[ \mathcal{I}(2) \]
\[ x_2(-y_1) \doteq x_2 y_2 \lor w_2 y_2 \]
\[ \mathcal{P}^{-}(3) \]
\[ w_1 y_1 \doteq y_3 + 1 \]
\[ \mathcal{I}(3) \]
\[ x_3(-y_1) \doteq x_3 y_2 \lor w_3 y_2 \]
\[ \mathcal{P}^{-}(4) \]
\[ w_1 y_1 \doteq y_3 + 2 \]
\[ \mathcal{I}(4) \]

Lemma 1

Lemma 2

Lemma 2*

\[ \lor \]

\[ ? \]

\[ \lor \]
Non-monadic cyclic equations

$\neg y_1 \equiv y_2 \lor y_3$

$\neg y_1 \equiv y_2 \lor w_1 y_2$

$x_2(\neg y_1) \equiv x_2 y_2 \lor w_2 y_2$

$x_3(\neg y_1) \equiv x_3 y_2 \lor w_3 y_2$

Lemma 1

Lemma 2

Lemma 2*

$\neg y_1 \equiv y_2 \lor w_1 y_2$

$\neg y_1 \equiv y_2 \lor w_1 y_2$

$\neg y_1 \equiv y_2 \lor w_1 y_2$

$\neg y_1 \equiv y_2 \lor w_1 y_2$

Lemma 1

Lemma 2

Lemma 2*

Theorem: $e$ is unifiable only if $\exists 0 \leq i \leq 3$ s.t. $P - (i)$ is unifiable.

$\mathcal{P} - (i) \subseteq \mathcal{P}(i)$

$\mathcal{U}(\mathcal{P}(i)) \subseteq \mathcal{U}(\mathcal{P} - (i))$
Non-monadic cyclic equations

\[ x_0(-y_1) \equiv x_0 y_2 \lor y_3 \]
\[ \mathcal{P}(1) \]
\[ -y_1 \equiv y_2 \lor y_3 \]
\[ \mathcal{I}(1) \]
\[ x_1(-y_1) \equiv x_1 y_2 \lor w_1 y_2 \]
\[ \mathcal{P}(2) \]
\[ w_1 y_1 \equiv y_3 \]
\[ \mathcal{I}(2) \]
\[ -y_1 \equiv y_2 \lor w_1 y_2 \]
\[ \mathcal{P}^{-}(3) \]
\[ -y_1 \equiv y_2 \lor w_2 y_2 \]
\[ \mathcal{P}(3) \]
\[ w_1 y_1 \equiv y_3 + 1 \]
\[ \mathcal{I}(3) \]
\[ x_2(-y_1) \equiv x_2 y_2 \lor w_2 y_2 \]
\[ \mathcal{P}(4) \]
\[ w_1 y_1 \equiv y_3 + 1 \]
\[ \mathcal{P}^{-}(4) \]
\[ x_3(-y_1) \equiv x_3 y_2 \lor w_3 y_2 \]
\[ \mathcal{P}^{-}(i) \subseteq \mathcal{P}(i) \]
\[ w_1 y_1 \equiv y_3 + 2 \]
\[ \mathcal{P}^{-}(i) \subseteq \mathcal{P}(i) \]
\[ \mathcal{U}(\mathcal{P}(i)) \subseteq \mathcal{U}(\mathcal{P}^{-}(i)) \]

Lemma 1

Lemma 2

Lemma 2*
Non-monadic cyclic equations

\[ x_0(\neg y_1) \equiv x_0 y_2 \lor y_3 \]
\[ \mathcal{P}(1) \quad \mathcal{J}(1) \]
\[ \neg y_1 \equiv y_2 \lor y_3 \]
\[ x_1(\neg y_1) \equiv x_1 y_2 \lor w_1 y_2 \]
\[ w_1 y_1 \equiv y_3 \]
\[ \mathcal{P}(2) \quad \mathcal{J}(2) \]
\[ \neg y_1 \equiv y_2 \lor w_1 y_2 \]
\[ w_1 y_1 \equiv y_3 \]
\[ \mathcal{P}(3) \quad \mathcal{J}(3) \]
\[ \neg y_1 \equiv y_2 \lor w_2 y_2 \]
\[ w_1 y_1 \equiv y_3 + 1 \]
\[ \mathcal{P}(4) \quad \mathcal{J}(4) \]
\[ \neg y_1 \equiv y_2 \lor w_3 y_2 \]
\[ w_1 y_1 \equiv y_3 + 2 \]

Lemma 1 \quad Lemma 2 \quad Lemma 2*

\[ \mathcal{P}^{-}(i) \subseteq \mathcal{P}(i) \]
\[ \mathcal{U}(\mathcal{P}(i)) \subseteq \mathcal{U}(\mathcal{P}^{-}(i)) \]

\[ \quad \blacksquare \quad \checkmark \quad \blacksquare \quad \checkmark \]

**Theorem:** \( e \) is unifiable only if \( \exists 0 \leq i \leq 3m \) s.t. \( \mathcal{P}^{-}(i) \) is unifiable.
Theorem: \( e \) is unifiable only if \( \exists 0 \leq i \leq 3m \) s.t. \( \mathcal{P}^{-}(i) \) is unifiable.
Decidable fragments of Projected Cycles

- $\mathcal{P}^-$ unifiable?
Decidable fragments of Projected Cycles

- $\mathfrak{P}^-$ unifiable?
- Simple such decidable classes
Decidable fragments of Projected Cycles

- $\mathcal{P}^-$ unifiable?
- Simple such decidable classes
- Stronger classes: regular tree automata
Decidable fragments of Projected Cycles

- $\mathcal{P}^-$ unifiable?
- Simple such decidable classes
- Stronger classes: regular tree automata
- Idea: $\mathcal{P} \setminus \mathcal{P}^-$ are freely generated
Decidable fragments of Projected Cycles

- $\mathcal{P}^{-}$ unifiable?
- Simple such decidable classes
- Stronger classes: regular tree automata
- Idea: $\mathcal{P} \setminus \mathcal{P}^{-}$ are freely generated
- Regular tree language + unifier for $\mathcal{P}^{-} = \text{decidability}$
Pattern unification

- Most useful subclass: higher-order unitary unification.
Pattern unification

- Most useful subclass: higher-order unitary unification.
- Applications:

- Proof assistants and Logical frameworks
- \( \lambda \text{Prolog} \ldots \)

- Variables are applied to a distinct list of bound variables:

- Pattern:
  \[
  \lambda z_1 z_2 (xz_1 z_2) = fyz_1 z_2
  \]

- Non-pattern:
  \[
  \lambda z_1 z_2 (xz_1 z_1) = a
  \]

- Idea: Determinism between (Project) and (Imitate)

- Higher-order patterns (Miller '91): same complexity as FOU
Pattern unification

- Most useful subclass: higher-order unitary unification.
- Applications:
  - Proof assistants and Logical frameworks
Pattern unification

- Most useful subclass: higher-order unitary unification.
- Applications:
  - Proof assistants and Logical frameworks
  - λProlog
Pattern unification

- Most usefull subclass: higher-order unitary unification.
- Applications:
  - Proof assistants and Logical frameworks
  - λProlog
  - ...

Pattern:

\[
\lambda z_1 z_2. x z_1 = f y z_1 z_2
\]

Non-pattern:

\[
\lambda z_1 z_2. x z_1 z_1 = a
\]

Idea: Determinism between (Project) and (Imitate)
Pattern unification

- Most useful subclass: higher-order unitary unification.
- Applications:
  - Proof assistants and Logical frameworks
  - λProlog
  - ...
- Variables are applied to a distinct list of bound variables:
Pattern unification

- Most useful subclass: higher-order unitary unification.
- Applications:
  - Proof assistants and Logical frameworks
  - λProlog
  - ...
- Variables are applied to a distinct list of bound variables:
- Pattern: \( \lambda z_1 z_2. x z_1 = f y z_1 z_2 \)
Pattern unification

- Most useful subclass: higher-order unitary unification.
- Applications:
  - Proof assistants and Logical frameworks
  - λProlog
  - ...
- Variables are applied to a distinct list of bound variables:
- Pattern: \( \lambda z_1 z_2. x z_1 = f y z_1 z_2 \)
- Non-pattern: \( \lambda z_1 z_2. x z_1 z_1 = a \)
Pattern unification

- Most useful subclass: higher-order unitary unification.
- Applications:
  - Proof assistants and Logical frameworks
  - \( \lambda \text{Prolog} \)
  - ...
- Variables are applied to a distinct list of bound variables:
  - Pattern: \( \lambda z_1 z_2 . x z_1 =^* fyz_1 z_2 \)
  - Non-pattern: \( \lambda z_1 z_2 . x z_1 z_1 = a \)
- Idea: Determinism between (Project) and (Imitate)
Pattern unification

- Most usefull subclass: higher-order unitary unification.
- Applications:
  - Proof assistants and Logical frameworks
  - $\lambda$Prolog
  - …
- variables are applied to a distinct list of bound variables:
- Pattern: $\lambda z_1 z_2. x z_1 \equiv f y z_1 z_2$
- Non-pattern: $\lambda z_1 z_2. x z_1 z_1 \equiv a$
- Idea: Determinism between (Project) and (Imitate)
- Higher-order patterns (Miller ’91): same complexity as FOU
Extending Pattern unification

- Many examples are unitary but are not patterns

\[ \lambda z. x (fz) = t \]

\[ \lambda \text{Prolog} \]

\[ \text{Remember: Determinism between (Project) and (Imitate)} \]

Class: restricting terms and subtle subterm relation

Examples:

Extended patterns:

\[ \lambda z_1, z_2. x (fz_1)(gz_1 z_2) = y (fz_1) \]

Non E-patterns:

\[ \lambda z_1, z_2. x (fz_1)(gz_1 z_2) = y z_1 \]
Extending Pattern unification

- Many examples are unitary but are not patterns
- Example:

Coq ssreflect:bigop (foldr)
\[ \lambda z. x (fz) = t \]

Prolog

Remember: Determinism between (Project) and (Imitate)

Class: restricting terms and subtle subterm relation

Examples:

Extended patterns:
\[ \lambda z_1, z_2. x (fz_1) (gz_1 z_2) = y (fz_1) \]

Non E-patterns:
\[ \lambda z_1, z_2. x (fz_1) (gz_1 z_2) = y z_1 \]
Extending Pattern unification

- Many examples are unitary but are not patterns
- Example:
  - Coq ssreflect:bigop (foldr)
    - $\lambda z.x(fz) \doteq t$
Extending Pattern unification

- Many examples are unitary but are not patterns

- Example:
  - Coq ssreflect:bigop (foldr)
    - $\lambda z.x(fz) \doteq t$
  - $\lambda$ Prolog
Extending Pattern unification

- Many examples are unitary but are not patterns
- Example:
  - Coq ssreflect:bigop (foldr)
    - $\lambda z. x(fz) \equiv t$
  - $\lambda$Prolog
  - $\ldots$
Extending Pattern unification

- Many examples are unitary but are not patterns
- Example:
  - Coq ssreflect:bigop (foldr)
    - $\lambda z.x(fz) \doteq t$
  - $\lambda$Prolog
  - ...
- Remember: Determinism between (Project) and (Imitate)
Extending Pattern unification

- Many examples are unitary but are not patterns
- Example:
  - Coq ssreflect:bigop (foldr)
    - \( \lambda z.x(fz) \equiv t \)
  - \( \lambda Prolog \)
  - ...
- Remember: Determinism between (Project) and (Imitate)
- Class: restricting terms and subtle subterm relation
Extending Pattern unification

- Many examples are unitary but are not patterns
- Example:
  - Coq ssreflect:bigop (foldr)
    - $\lambda z.x(fz) \doteq t$
  - $\lambda$Prolog
  - …
- Remember: Determinism between (Project) and (Imitate)
- Class: restricting terms and subtle subterm relation
- Examples:
Extending Pattern unification

- Many examples are unitary but are not patterns
- Example:
  - Coq ssreflect:bigop (foldr)
    - $\lambda z. x(fz) \doteq t$
  - $\lambda$ Prolog
  - ...
- Remember: Determinism between (Project) and (Imitate)
- Class: restricting terms and subtle subterm relation
- Examples:
  - Extended patterns: $\lambda z_1, z_2. x(fz_1)(gz_1z_2) \doteq y(fz_1)$
Extending Pattern unification

- Many examples are unitary but are not patterns
- Example:
  - Coq ssreflect:bigop (foldr)
    - $\lambda z. x(fz) \doteq t$
  - $\lambda$ Prolog
  - ...
- Remember: Determinism between (Project) and (Imitate)
- Class: restricting terms and subtle subterm relation
- Examples:
  - Extended patterns: $\lambda z_1, z_2. x(fz_1)(gz_1z_2) \doteq y(fz_1)$
  - Non E-patterns: $\lambda z_1, z_2. x(fz_1)(gz_1z_2) \doteq yz_1$
Conclusion

- Active research:
  - Define tree automata class
  - Add abstractions to extended patterns
  - Implementation:
    - Extended patterns: Coq, Matita, Isabelle, Abella, Twelf, $\lambda$Prolog, theorem provers.
  - Improved termination: theorem provers
  - Comparisons:
    - theorem provers with variations of the two unification procedures
Conclusion

- Active research:
  - Define tree automata class

- Implementation:
  - Extended patterns: Coq, Matita, Isabelle, Abella, Twelf, λ-Prolog, theorem provers.

- Improved termination: theorem provers

- Comparisons:
  - theorem provers with variations of the two unification procedures
Conclusion

- Active research:
  - Define tree automata class
  - Add abstractions to extended patterns
Conclusion

- Active research:
  - Define tree automata class
  - Add abstractions to extended patterns
- Implementation:
Conclusion

- Active research:
  - Define tree automata class
  - Add abstractions to extended patterns

- Implementation:
  - Extended patterns: Coq, Matita, Isabelle, Abella, Twelf, λProlog, theorem provers.
Conclusion

► Active research:
  ▶ Define tree automata class
  ▶ Add abstractions to extended patterns

► Implementation:
  ▶ Extended patterns: Coq, Matita, Isabelle, Abella, Twelf, λProlog, theorem provers.
  ▶ Improved termination: theorem provers
Conclusion

- Active research:
  - Define tree automata class
  - Add abstractions to extended patterns
- Implementation:
  - Extended patterns: Coq, Matita, Isabelle, Abella, Twelf, $\lambda$Prolog, theorem provers.
  - Improved termination: theorem provers
- Comparisons:
Conclusion

▶ Active research:
  ▶ Define tree automata class
  ▶ Add abstractions to extended patterns

▶ Implementation:
  ▶ Extended patterns: Coq, Matita, Isabelle, Abella, Twelf, λProlog, theorem provers.
  ▶ Improved termination: theorem provers

▶ Comparisons:
  ▶ theorem provers with variations of the two unification procedures