Higher-order Unification (a personal perspective)

Tomer Libal

Inria, France

EMCL Workshop 2016
Vienna, February 12

History

History

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

History

How can we proceed?

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

History

How can we proceed?

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

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

History

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

How can we proceed?

$$\dfrac{\dfrac{\dfrac{\dfrac{\vdash Pa, \neg Pa \qquad \vdash Pfa, \neg Pfa \qquad \vdash Pffa, \neg Pffa}{\cdots}}{\vdash \neg Pa, Pffa, Pa \wedge \neg Pfa, Pfa \wedge \neg Pffa}\,\exists : r}{\vdash \neg Pa, Pffa, Pa \wedge \neg Pfa, \exists x.Px \wedge \neg Pfx}\,\exists : r}{\vdash \neg Pa, Pffa, \exists x.Px \wedge \neg Pfx, \exists x.Px \wedge \neg Pfx}\,\mathtt{con}}{\vdash \neg Pa, Pffa, \exists x.Px \wedge \neg Pfx}$$

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 mehcanizing logic

$$\dfrac{\dfrac{\dfrac{\dfrac{\vdash Pa, \neg Pa \qquad \vdash Pfa, \neg Pfa \qquad \vdash Pffa, \neg Pffa}{\cdots}}{\vdash \neg Pa, Pffa, Pa \wedge \neg Pfa, Pfa \wedge \neg Pffa}}{\vdash \neg Pa, Pffa, Pa \wedge \neg Pfa, \exists x.Px \wedge \neg Pfx}\ \exists : r}{\vdash \neg Pa, Pffa, \exists x.Px \wedge \neg Pfx, \exists x.Px \wedge \neg Pfx}\ \exists : r}{\vdash \neg Pa, Pffa, \exists x.Px \wedge \neg Pfx}\ \mathtt{con}$$

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 mehcanizing logic

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

$$\dfrac{\dfrac{\dfrac{\dfrac{\vdash Pa, \neg Pa \qquad \vdash Pfa, \neg Pfa \qquad \vdash Pffa, \neg Pffa}{\cdots}}{\vdash \neg Pa, Pffa, Pa \wedge \neg Pfa, Pfa \wedge \neg Pffa}}{\vdash \neg Pa, Pffa, Pa \wedge \neg Pfa, \exists x.Px \wedge \neg Pfx}\ \exists : r}{\vdash \neg Pa, Pffa, \exists x.Px \wedge \neg Pfx, \exists x.Px \wedge \neg Pfx}\ \exists : r}{\vdash \neg Pa, Pffa, \exists x.Px \wedge \neg Pfx}\ \mathtt{con}$$

History

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

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

How can we proceed?

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

But, the Herband Space is infinite!

$$\dfrac{\dfrac{\dfrac{\dfrac{\vdash Pa, \neg Pa \qquad \vdash Pfa, \neg Pfa \qquad \vdash Pffa, \neg Pffa}{\cdots}}{\vdash \neg Pa, Pffa, Pa \wedge \neg Pfa, Pfa \wedge \neg Pffa}}{\vdash \neg Pa, Pffa, Pa \wedge \neg Pfa, \exists x.Px \wedge \neg Pfx}\ \exists : r}{\vdash \neg Pa, Pffa, \exists x.Px \wedge \neg Pfx, \exists x.Px \wedge \neg Pfx}\ \exists : r}{\vdash \neg Pa, Pffa, \exists x.Px \wedge \neg Pfx}\ \mathtt{con}$$

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 mehcanizing logic

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

But, the Herband Space is infinite!

Prawitz (1960): try my Unification method!

$$\dfrac{\dfrac{\dfrac{\dfrac{\vdash Pa, \neg Pa \qquad \vdash Pfa, \neg Pfa \qquad \vdash Pffa, \neg Pffa}{\cdots}}{\vdash \neg Pa, Pffa, Pa \wedge \neg Pfa, Pfa \wedge \neg Pffa}}{\vdash \neg Pa, Pffa, Pa \wedge \neg Pfa, \exists x.Px \wedge \neg Pfx}\ \exists : r}{\vdash \neg Pa, Pffa, \exists x.Px \wedge \neg Pfx, \exists x.Px \wedge \neg Pfx}\ \exists : r}{\vdash \neg Pa, Pffa, \exists x.Px \wedge \neg Pfx}\ \mathrm{con}$$

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 mehcanizing logic

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

But, the Herband Space is infinite!

Prawitz (1960): try my Unification method!

Thanks! It is exactly what we looked for

$$\dfrac{\dfrac{\dfrac{\dfrac{\vdash Pa, \neg Pa \qquad \vdash Pfa, \neg Pfa \qquad \vdash Pffa, \neg Pffa}{\cdots}}{\vdash \neg Pa, Pffa, Pa \wedge \neg Pfa, Pfa \wedge \neg Pffa}}{\vdash \neg Pa, Pffa, Pa \wedge \neg Pfa, \exists x.Px \wedge \neg Pfx}\ \exists : r}{\vdash \neg Pa, Pffa, \exists x.Px \wedge \neg Pfx, \exists x.Px \wedge \neg Pfx}\ \exists : r}{\vdash \neg Pa, Pffa, \exists x.Px \wedge \neg Pfx}\ \mathrm{con}$$

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 mehcanizing logic

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

But, the Herband 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!

$$\dfrac{\dfrac{\dfrac{\dfrac{\vdash Pa, \neg Pa \qquad \vdash Pfa, \neg Pfa \qquad \vdash Pffa, \neg Pffa}{\cdots}}{\vdash \neg Pa, Pffa, Pa \wedge \neg Pfa, Pfa \wedge \neg Pffa}}{\vdash \neg Pa, Pffa, Pa \wedge \neg Pfa, \exists x.Px \wedge \neg Pfx}\ \exists : r}{\vdash \neg Pa, Pffa, \exists x.Px \wedge \neg Pfx, \exists x.Px \wedge \neg Pfx}\ \exists : r}{\vdash \neg Pa, Pffa, \exists x.Px \wedge \neg Pfx}\ \mathrm{con}$$

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 mehcanizing logic

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

But, the Herband 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 J.A. Robinson has read Prawitz's paper and
in January 1965 the Resolution method was published

$$\dfrac{\dfrac{\dfrac{\dfrac{\vdash Pa, \neg Pa \qquad \vdash Pfa, \neg Pfa \qquad \vdash Pffa, \neg Pffa}{\cdots}}{\vdash \neg Pa, Pffa, Pa \wedge \neg Pfa, Pfa \wedge \neg Pffa}}{\vdash \neg Pa, Pffa, Pa \wedge \neg Pfa, \exists x.Px \wedge \neg Pfx}\ \exists : r}{\vdash \neg Pa, Pffa, \exists x.Px \wedge \neg Pfx, \exists x.Px \wedge \neg Pfx}\ \exists : r}{\vdash \neg Pa, Pffa, \exists x.Px \wedge \neg Pfx}\ \mathrm{con}$$

First-order Unification

First-order Unification

- Applications:

First-order Unification

- Applications:
  - Automated deduction (Resolution, Tableau, . . .)

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:
  - $\{t_1 \doteq s_1, \ldots, t_n \doteq s_n\}$

First-order Unification - Basic Algorithm

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

First-order Unification - Basic Algorithm

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

$$\frac{\{u \doteq u\} \cup S}{S}\ \textit{delete} \qquad \frac{\{f(v_1,\ldots,v_n) \doteq f(u_1,\ldots,u_n)\} \cup S}{\{v_1 \doteq u_1, \ldots, v_n \doteq u_n\} \cup S}\ \textit{decomp} \qquad \frac{\{x \doteq v\} \cup S}{\sigma(S)}\ \textit{bind}$$

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 σ:
  - $\forall\theta\exists\delta.\sigma = \theta \circ \delta$

$$\frac{\{u \doteq u\} \cup S}{S}\ \textit{delete} \qquad \frac{\{f(v_1,\ldots,v_n) \doteq f(u_1,\ldots,u_n)\} \cup S}{\{v_1 \doteq u_1, \ldots, v_n \doteq u_n\} \cup S}\ \textit{decomp} \qquad \frac{\{x \doteq v\} \cup S}{\sigma(S)}\ \textit{bind}$$

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

$$\frac{\{u \doteq u\} \cup S}{S}\ \textit{delete} \qquad \frac{\{f(v_1,\ldots,v_n) \doteq f(u_1,\ldots,u_n)\} \cup S}{\{v_1 \doteq u_1, \ldots, v_n \doteq u_n\} \cup S}\ \textit{decomp} \qquad \frac{\{x \doteq v\} \cup S}{\sigma(S)}\ \textit{bind}$$

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

$$\{f\,x\,a \doteq f\,a\,a\} \rightarrow \{x \doteq a, a \doteq a\}$$ 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 σ:
  - $\forall\theta\exists\delta.\sigma = \theta \circ \delta$

$$\frac{\{u \doteq u\} \cup S}{S}\ \textit{delete} \qquad \frac{\{f(v_1,\ldots,v_n) \doteq f(u_1,\ldots,u_n)\} \cup S}{\{v_1 \doteq u_1, \ldots, v_n \doteq u_n\} \cup S}\ \textit{decomp} \qquad \frac{\{x \doteq v\} \cup S}{\sigma(S)}\ \textit{bind}$$

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

$$\{f\,x\,a \doteq f\,a\,a\} \rightarrow \{x \doteq a, a \doteq a\}$$

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 σ:
  - $\forall\theta\exists\delta.\sigma = \theta \circ \delta$

$$\frac{\{u \doteq u\} \cup S}{S}\ \textit{delete} \qquad \frac{\{f(v_1,\ldots,v_n) \doteq f(u_1,\ldots,u_n)\} \cup S}{\{v_1 \doteq u_1, \ldots, v_n \doteq u_n\} \cup S}\ \textit{decomp} \qquad \frac{\{x \doteq v\} \cup S}{\sigma(S)}\ \textit{bind}$$

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

$$\begin{array}{r} \{f\,x\,a \doteq f\,a\,a\} \\ \{x \doteq a, a \doteq a\} \\ \{a \doteq a\} \end{array}$$

decompose

bind

First-order Unification - Basic Algorithm

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

$$\frac{\{u \doteq u\} \cup S}{S}\ \textit{delete} \qquad \frac{\{f(v_1,\ldots,v_n) \doteq f(u_1,\ldots,u_n)\} \cup S}{\{v_1 \doteq u_1, \ldots, v_n \doteq u_n\} \cup S}\ \textit{decomp} \qquad \frac{\{x \doteq v\} \cup S}{\sigma(S)}\ \textit{bind}$$

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

$$\{f\,x\,a \doteq f\,a\,a\} \xrightarrow{\text{decompose}} \{x \doteq a, a \doteq a\} \xrightarrow{\text{bind}} \{a \doteq a\}$$

First-order Unification - Basic Algorithm

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

$$\frac{\{u \doteq u\} \cup S}{S}\ \textit{delete} \qquad \frac{\{f(v_1,\ldots,v_n) \doteq f(u_1,\ldots,u_n)\} \cup S}{\{v_1 \doteq u_1, \ldots, v_n \doteq u_n\} \cup S}\ \textit{decomp} \qquad \frac{\{x \doteq v\} \cup S}{\sigma(S)}\ \textit{bind}$$

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

$$\begin{aligned} \{f\,x\,a \doteq f\,a\,a\} &\xrightarrow{\textit{decompose}} \\ \{x \doteq a, a \doteq a\} &\xrightarrow{\textit{bind}} \\ \{a \doteq a\} &\xrightarrow{\textit{delete}} \\ \{\} & \end{aligned}$$

First-order Unification - Basic Algorithm

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

$$\frac{\{u \doteq u\} \cup S}{S}\ \textit{delete} \qquad \frac{\{f(v_1,\ldots,v_n) \doteq f(u_1,\ldots,u_n)\} \cup S}{\{v_1 \doteq u_1, \ldots, v_n \doteq u_n\} \cup S}\ \textit{decomp} \qquad \frac{\{x \doteq v\} \cup S}{\sigma(S)}\ \textit{bind}$$

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

$$\{f\,x\,a \doteq f\,a\,a\} \xrightarrow{\text{decompose}} \{x \doteq a, a \doteq a\} \xrightarrow{\text{bind}} \{a \doteq a\} \xrightarrow{\text{delete}} \{\}$$

Higher-order Unification

Higher-order Unification

- Applications:

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 (λ-Prolog)
  - Type inference (Coq, dependent types, ...)

Higher-order Unification

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

Higher-order Unification

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

Higher-order Unification

- Applications:
  - Automated deduction (Arithmetic, Meta-physics,...)
  - Programming languages (λ-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, ...

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,...)
  - 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".
  - $P(dan) \doteq \texttt{likes}(dan, \texttt{wife-of}(dan))$

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".
  - $P(dan) \doteq \texttt{likes}(dan, \texttt{wife-of}(dan))$
    - $P \mapsto \lambda z.\texttt{likes}(dan, \texttt{wife-of}(dan))$

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".
  - $P(dan) \doteq \texttt{likes}(dan, \texttt{wife-of}(dan))$
    - $P \mapsto \lambda z.\texttt{likes}(dan, \texttt{wife-of}(dan))$
    - $P \mapsto \lambda z.\texttt{likes}(dan, \texttt{wife-of}(z))$

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".
  - $P(dan) \doteq \texttt{likes}(dan, \texttt{wife-of}(dan))$
    - $P \mapsto \lambda z.\texttt{likes}(dan, \texttt{wife-of}(dan))$
    - $P \mapsto \lambda z.\texttt{likes}(dan, \texttt{wife-of}(z))$
    - $P \mapsto \lambda z.\texttt{likes}(z, \texttt{wife-of}(z))$

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".
  - $P(dan) \doteq \texttt{likes}(dan, \texttt{wife-of}(dan))$
    - $P \mapsto \lambda z.\texttt{likes}(dan, \texttt{wife-of}(dan))$
    - $P \mapsto \lambda z.\texttt{likes}(dan, \texttt{wife-of}(z))$
    - $P \mapsto \lambda z.\texttt{likes}(z, \texttt{wife-of}(z))$
    - $P \mapsto \lambda z.\texttt{likes}(z, \texttt{wife-of}(dan))$

Higher-order Unification

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

Higher-order Unification

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

decompose

Higher-order Unification

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

decompose

bind

Higher-order Unification

$$\{f\,x\,(y\,b)\,(y\,c) \doteq f\,a\,b\,c\}$$
$$\{x \doteq a, y\,b \doteq b, y\,c \doteq c\} \quad \text{decompose}$$
$$\{a \doteq a, y\,b \doteq b, y\,c \doteq c\} \quad \text{bind}$$
$$\{y\,b \doteq b, y\,c \doteq c\} \quad \text{delete}$$

Higher-order Unification

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

decompose

bind

delete

Higher-order Unification

$$\{f\,x\,(y\,b)\,(y\,c) \doteq f\,a\,b\,c\}$$
$$\{x \doteq a, y\,b \doteq b, y\,c \doteq c\} \quad \text{decompose}$$
$$\{a \doteq a, y\,b \doteq b, y\,c \doteq c\} \quad \text{bind}$$
$$\{y\,b \doteq b, y\,c \doteq c\} \quad \text{delete}$$

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

Higher-order Unification

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

decompose

$$\{x \doteq a,\ y\,b \doteq b,\ y\,c \doteq c\}$$

bind

$$\{a \doteq a,\ y\,b \doteq b,\ y\,c \doteq c\}$$

delete

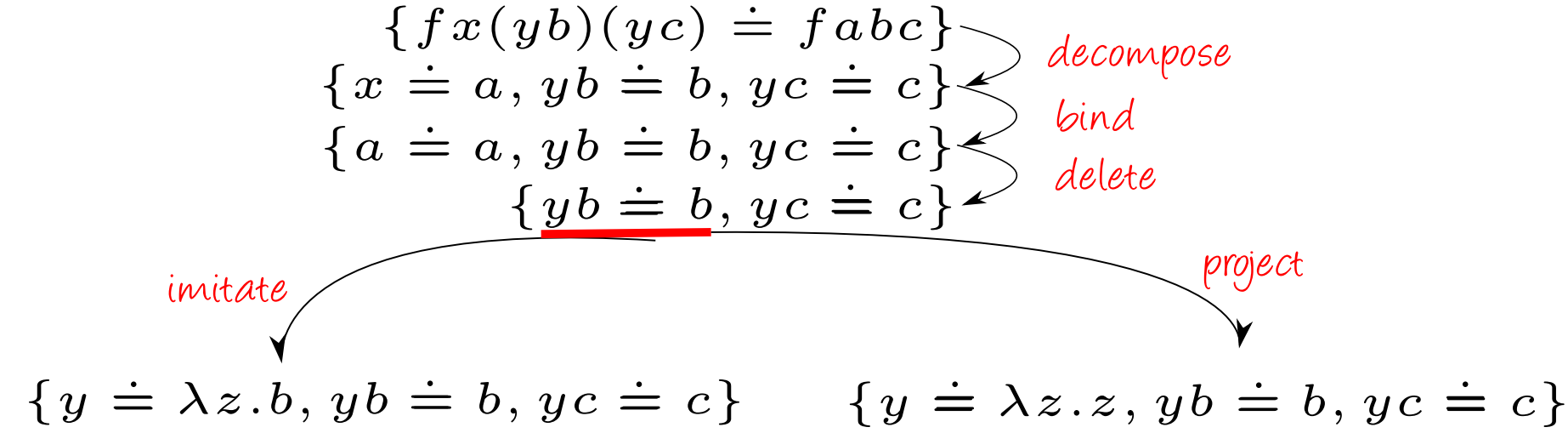
$$\{y\,b \doteq b,\ y\,c \doteq c\}$$

imitate

$$\{y \doteq \lambda z.\,b,\ y\,b \doteq b,\ y\,c \doteq c\}$$

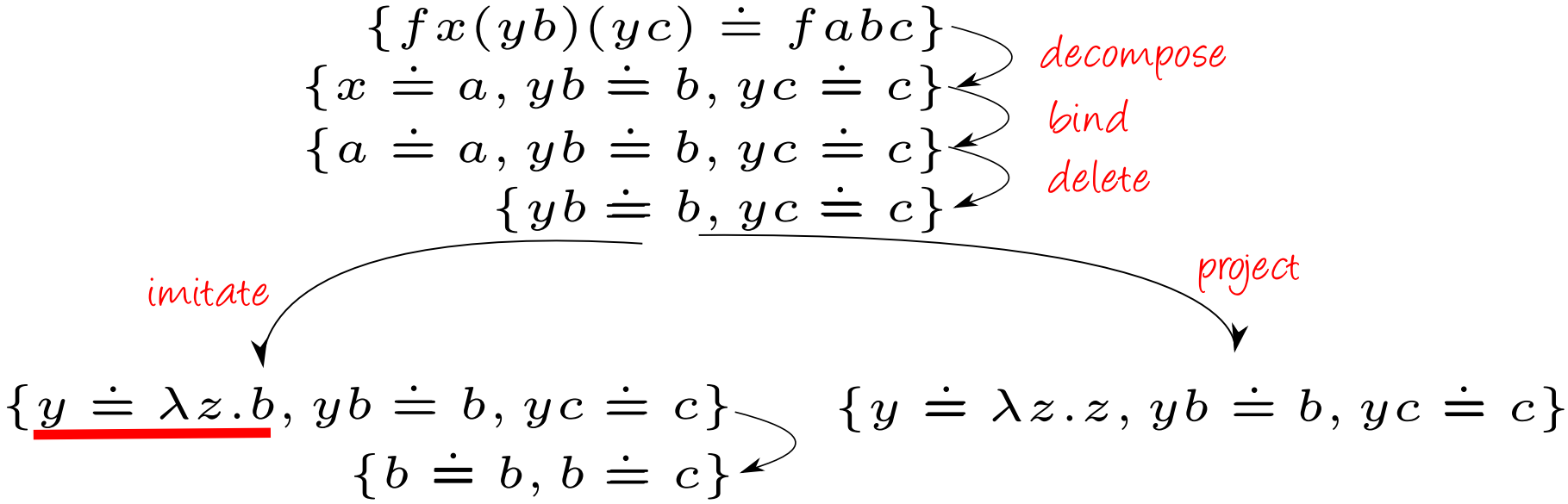
- Jensen and Pietrzykowski (1973): either
  - imitation $y \mapsto \lambda z.b$
  - projection $y \mapsto \lambda z.z$

Higher-order Unification



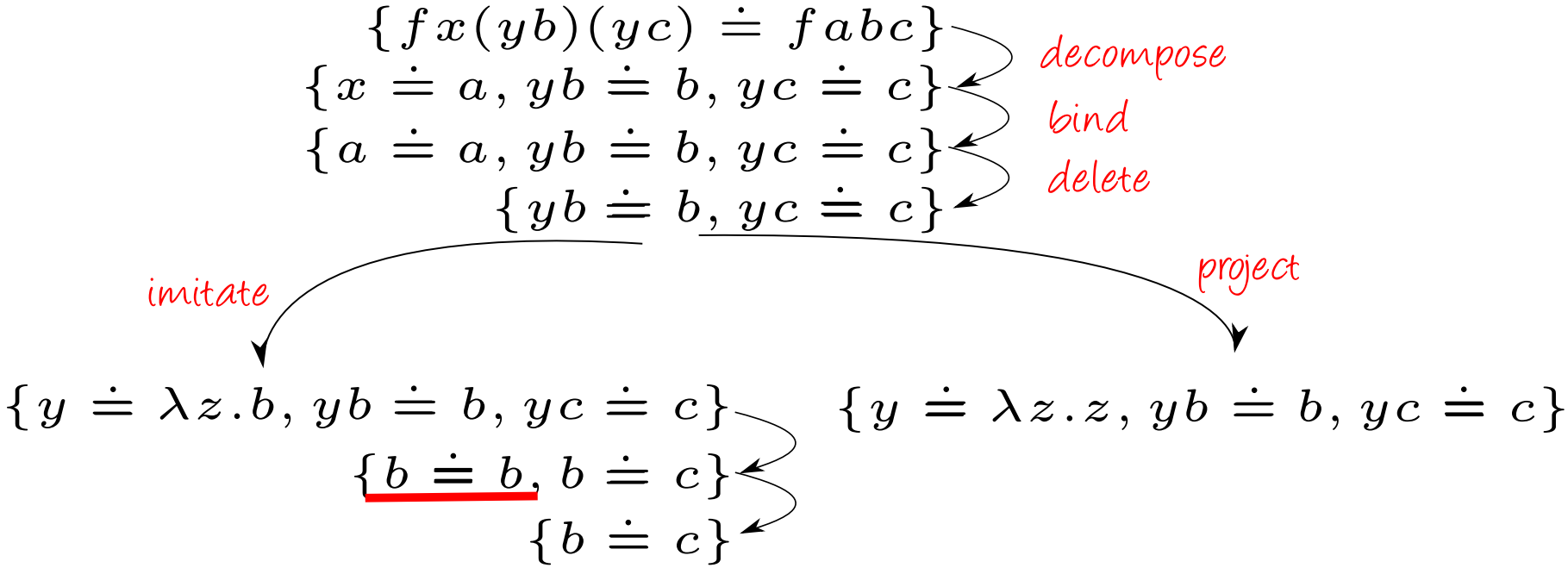
- Jensen and Pietrzykowski (1973): either
  - imitation $y \mapsto \lambda z.b$
  - projection $y \mapsto \lambda z.z$

Higher-order Unification



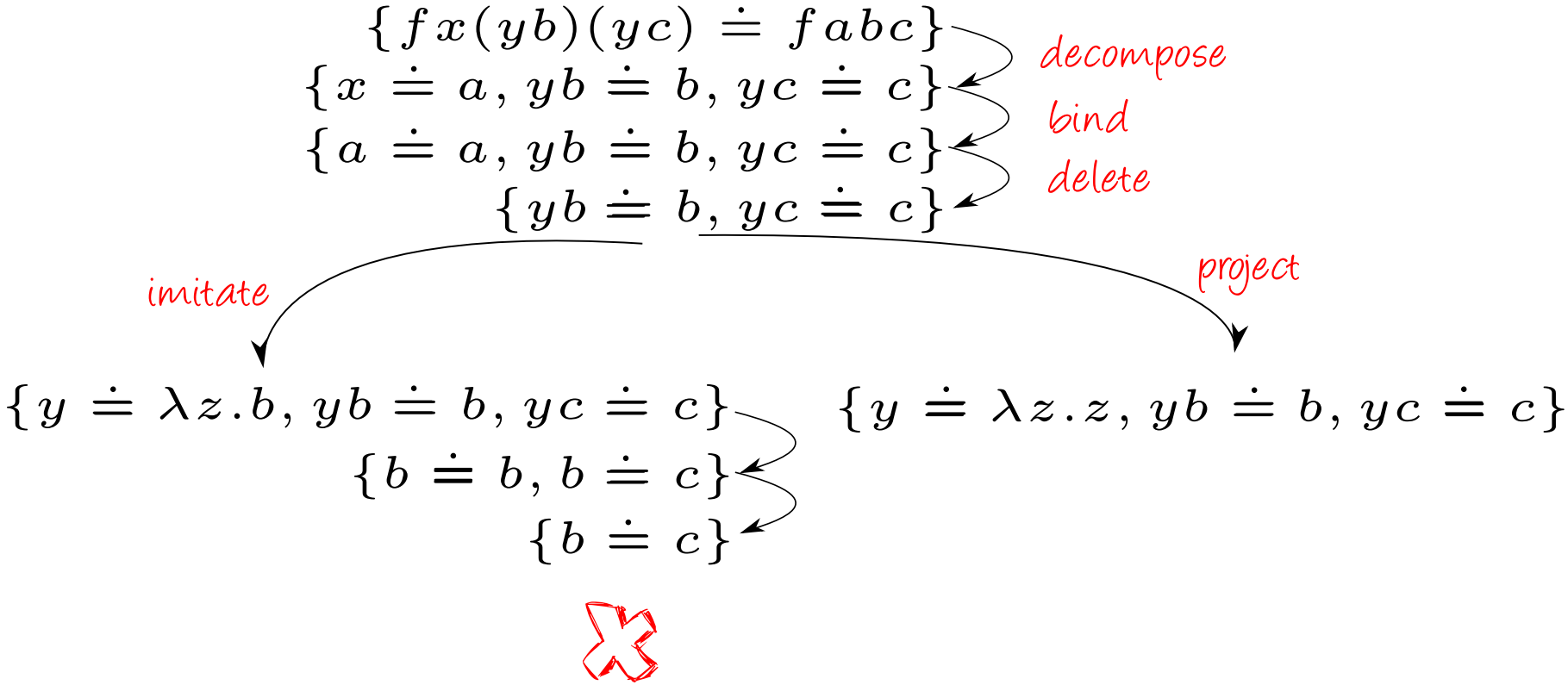
- Jensen and Pietrzykowski (1973): either
  - imitation $y \mapsto \lambda z.b$
  - projection $y \mapsto \lambda z.z$

Higher-order Unification



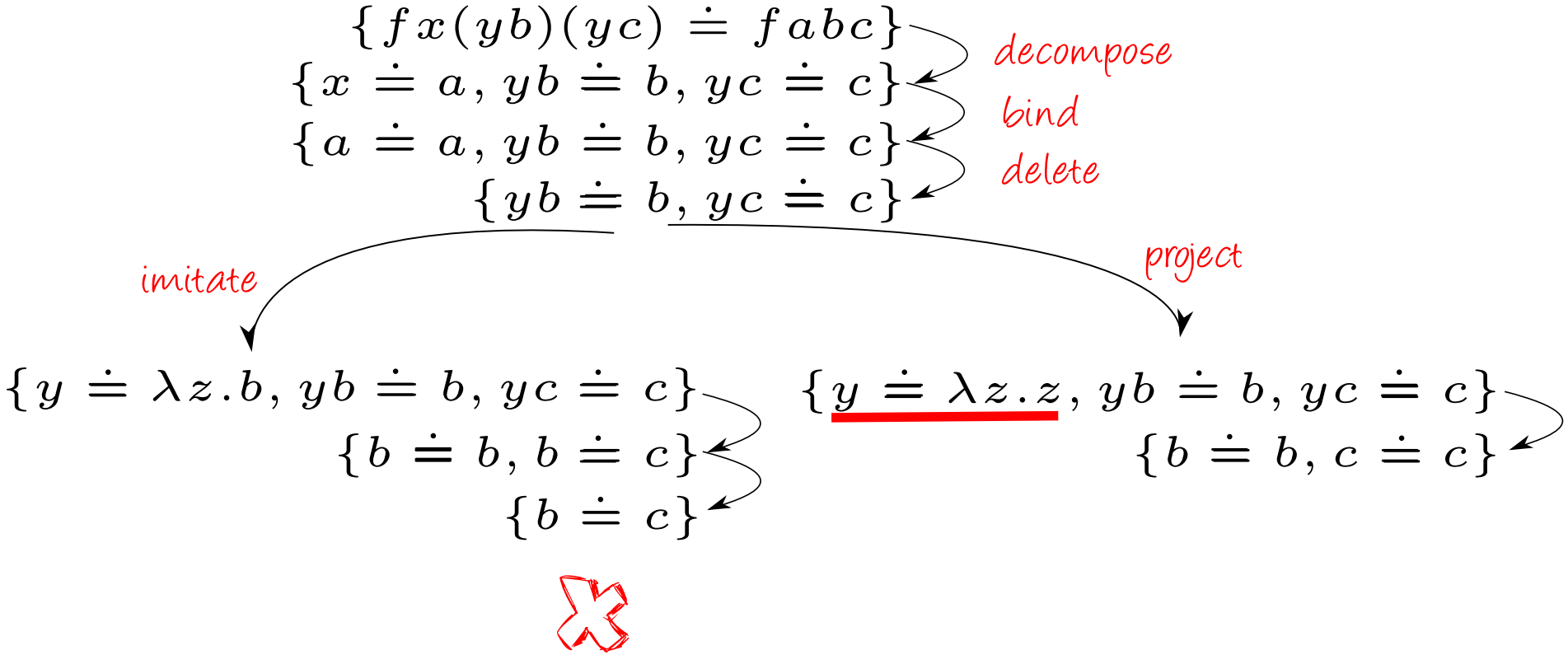
- Jensen and Pietrzykowski (1973): either
  - imitation $y \mapsto \lambda z.b$
  - projection $y \mapsto \lambda z.z$

Higher-order Unification



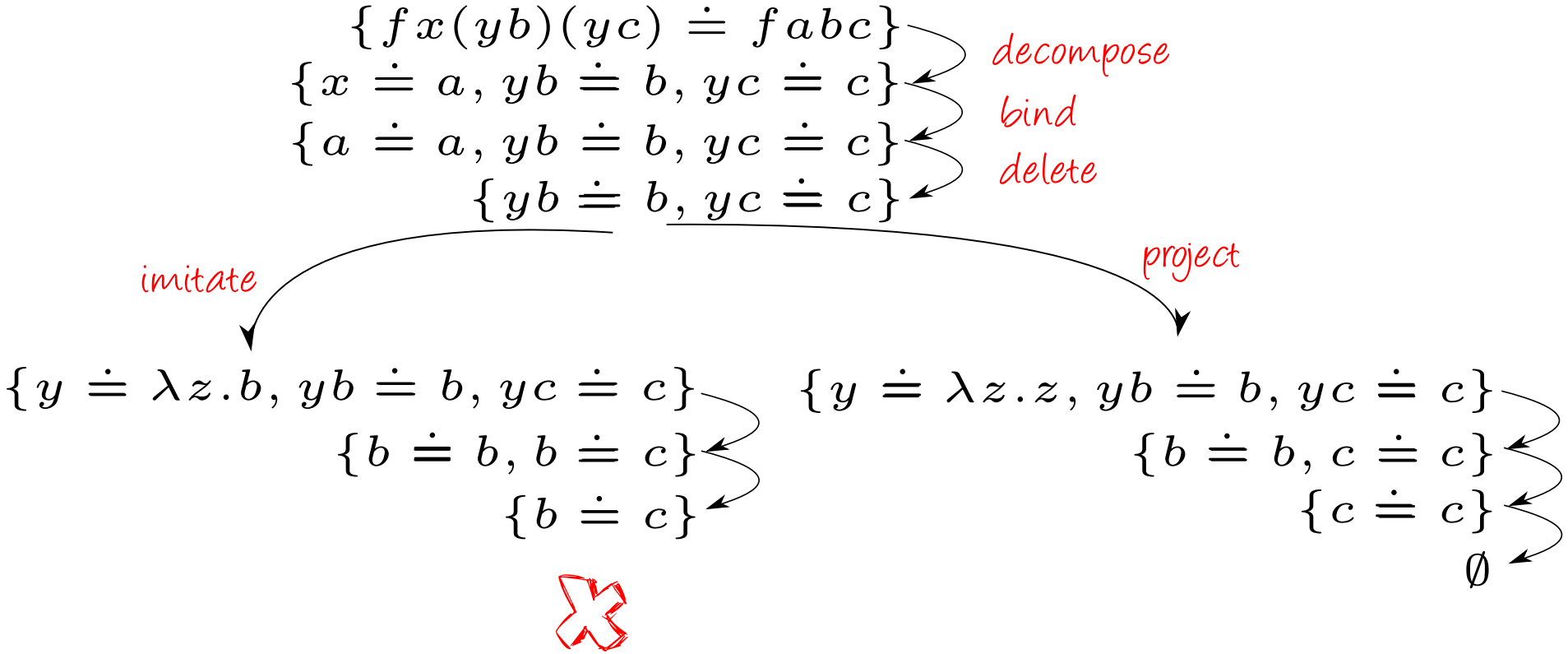
- Jensen and Pietrzykowski (1973): either
  - imitation $y \mapsto \lambda z.b$
  - projection $y \mapsto \lambda z.z$

Higher-order Unification



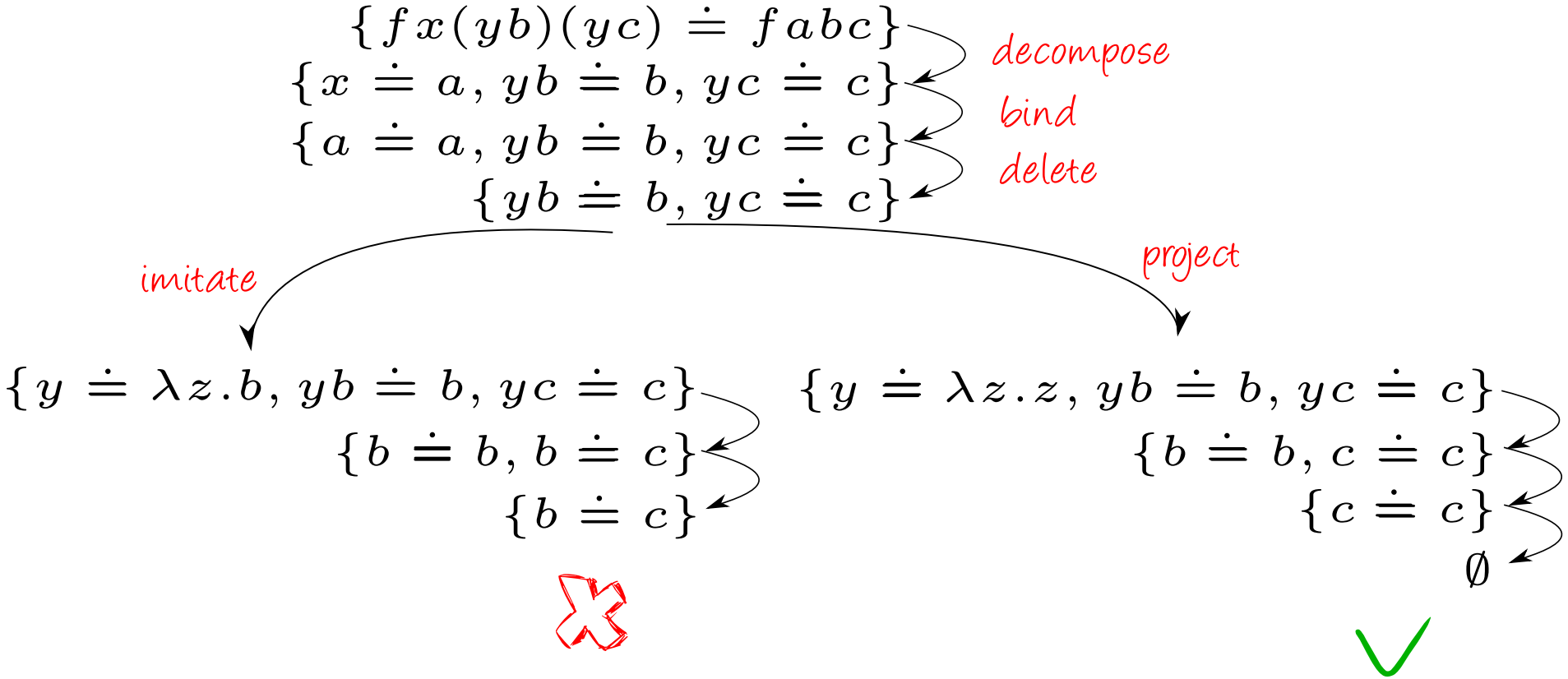
- Jensen and Pietrzykowski (1973): either
  - imitation $y \mapsto \lambda z.b$
  - projection $y \mapsto \lambda z.z$

Higher-order Unification



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

Higher-order Unification



- 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

- $xab \doteq y(fa)a$

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

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

Higher-order Unification - flexible pairs

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

$$\frac{\{u \doteq u\} \cup S}{S}\ \textit{delete} \qquad \frac{\{\lambda\overline{x_k}.z(\overline{x_k}) \doteq \lambda\overline{x_k}.v\} \cup S}{\{\langle\langle z, \lambda\overline{x_k}.v\rangle\rangle\} \cup \sigma(S)\downarrow_\beta}\ \textit{bind}$$

$$\frac{\{\lambda\overline{x_k}.a(\overline{v_n}) \doteq \lambda\overline{x_k}.a(\overline{u_n}\} \cup S}{\{\lambda\overline{x_k}.v_1 \doteq \lambda\overline{x_k}.u_1, \ldots, \lambda\overline{x_k}.v_n \doteq \lambda\overline{x_k}.u_n\} \cup S}\ \textit{decomp}$$

$$\frac{\{\lambda\overline{x_k}.y(\overline{u_n}) \doteq \lambda\overline{x_k}.b(\overline{v_m})\} \cup S}{\{y \uparrow_\eta \doteq t \uparrow_\eta, \lambda\overline{x_k}.y(\overline{u_n}) \doteq \lambda\overline{x_k}.b(\overline{v_m})\} \cup S}\ \textit{imitate} \qquad \frac{\{\lambda\overline{x_k}.y(\overline{u_n}) \doteq \lambda\overline{x_k}.a(\overline{v_m})\} \cup S}{\{y \uparrow_\eta \doteq s \uparrow_\eta, \lambda\overline{x_k}.y(\overline{u_n}) \doteq \lambda\overline{x_k}.a(\overline{v_m})\} \cup S}\ \textit{project}$$

Where $a \in \Sigma$ or $a \in \overline{x_k}$; $b \in \Sigma$; z does not occur in v;
$\sigma = [\lambda\overline{x_k}.v/x]$; $t = \lambda\overline{x_n}.b(\overline{y_m(\overline{x_n})})$;
$s = \lambda\overline{x_n}.x_i(\overline{y_l(\overline{x_n})})$ for $0 < i \leq n$ and $l = \mathtt{ty}(x_i)$.

Higher-order Unification: non-termination

Higher-order Unification: non-termination

- Enumerates a minimal complete set of unifiers.

Higher-order Unification: non-termination

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

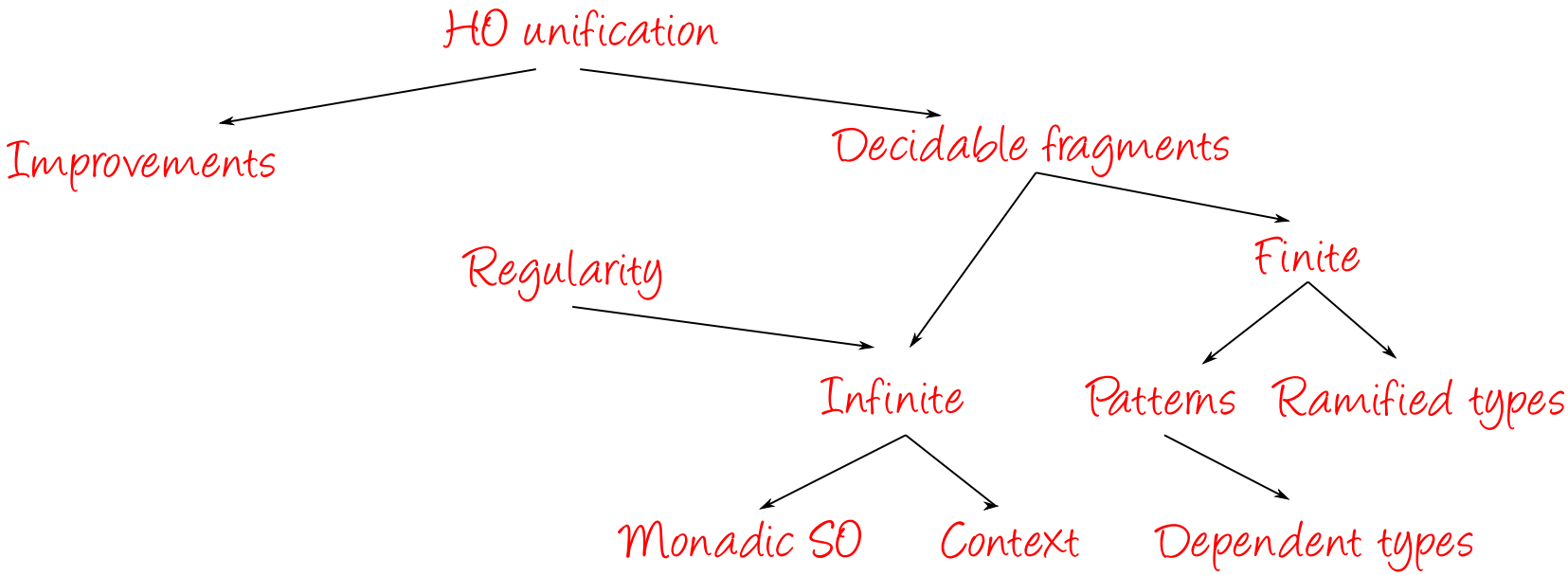
Higher-order Unification: non-termination

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

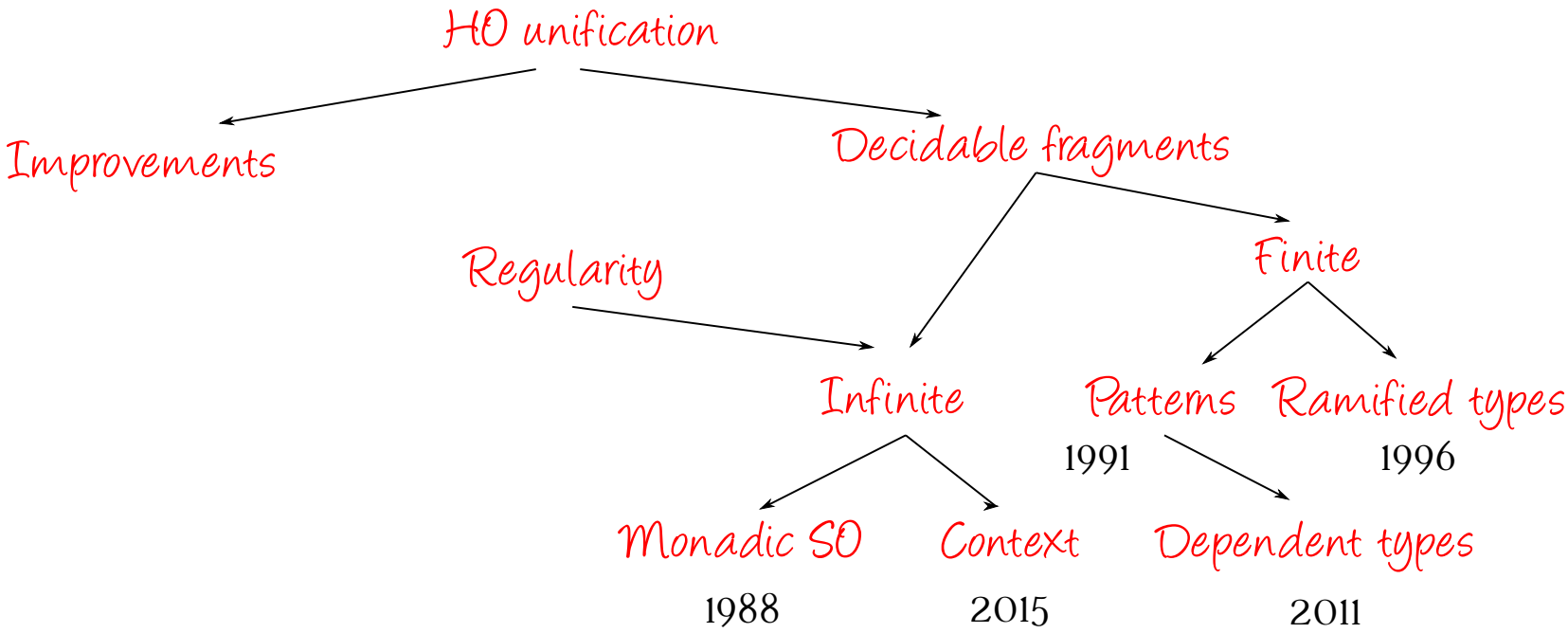
Higher-order Unification: non-termination

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

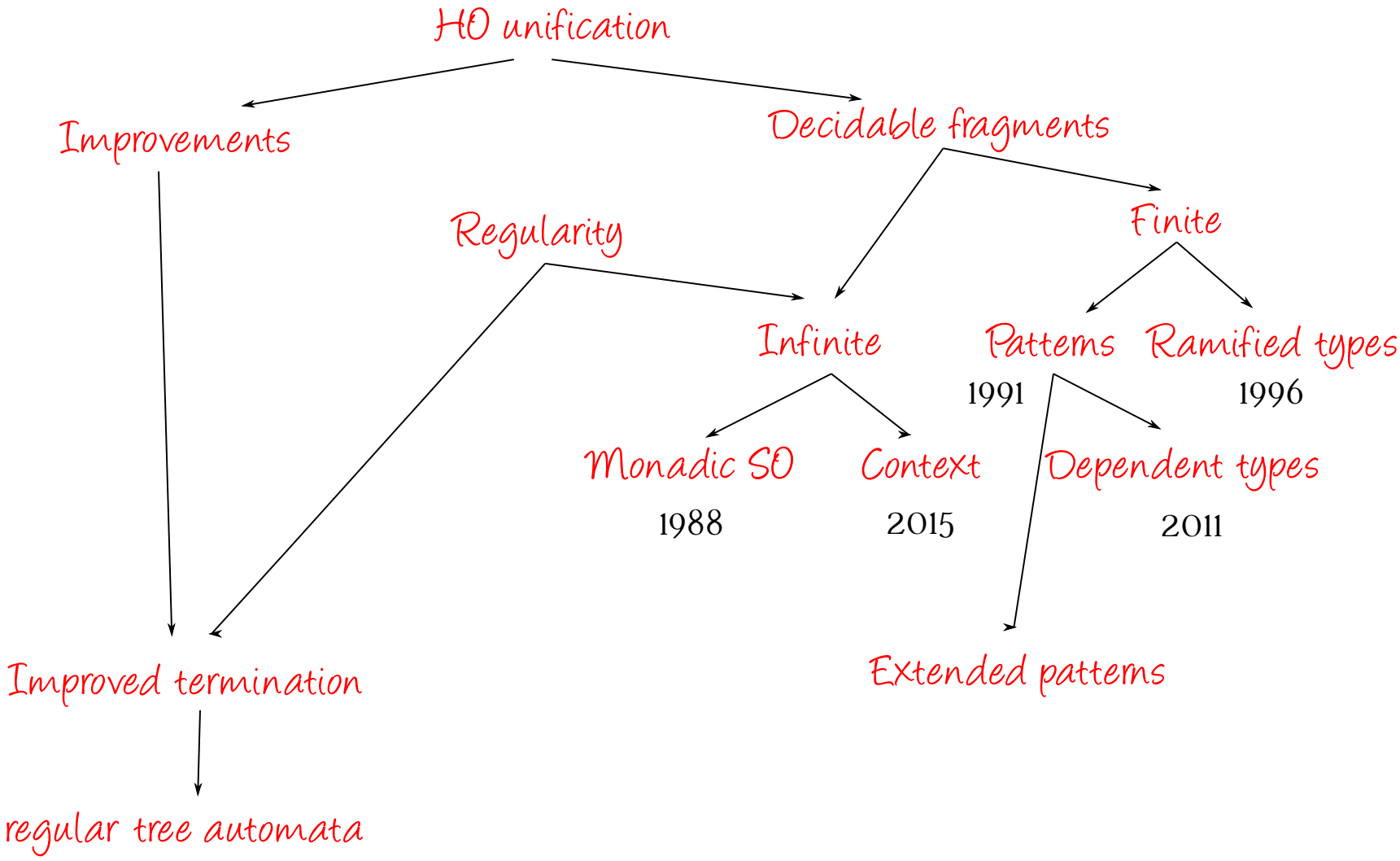
Higher-order Unification - Overview



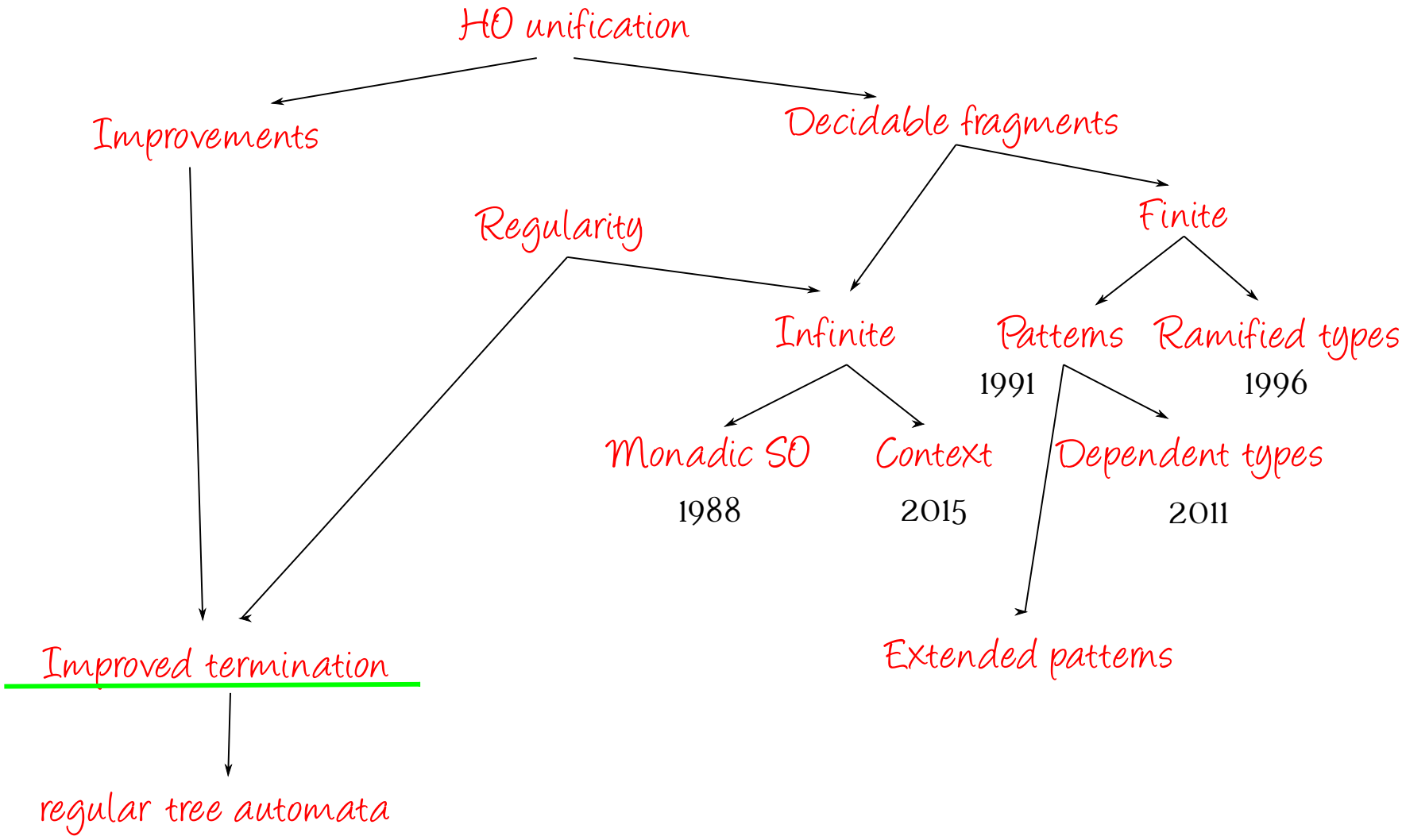
Higher-order Unification - Overview



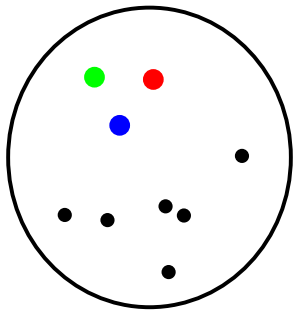
Higher-order Unification - Overview

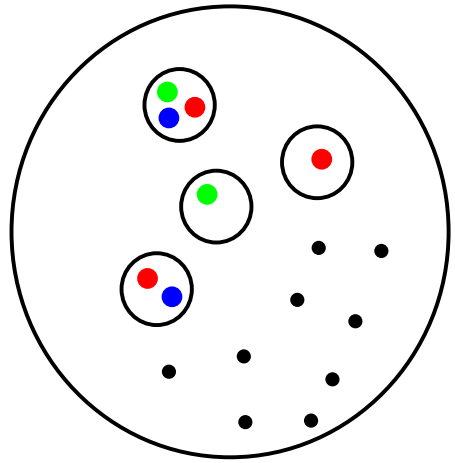


Higher-order Unification - Overview

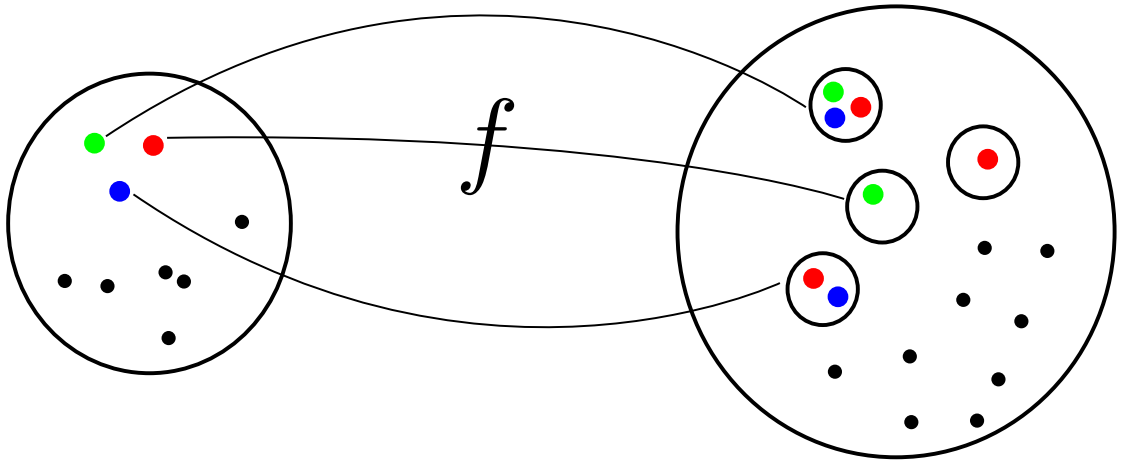


Cantor's Theorem

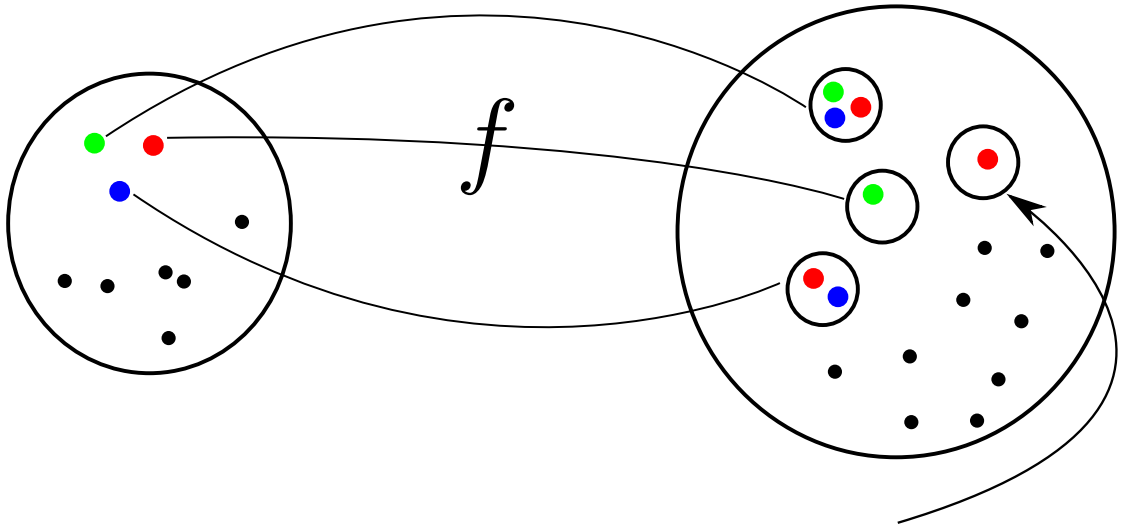




Cantor's Theorem



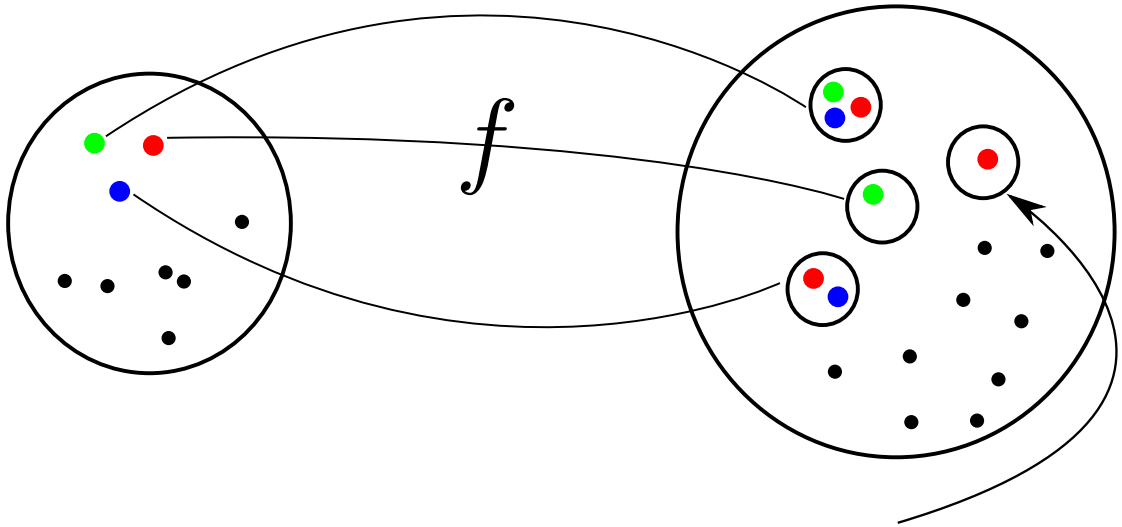
Cantor's Theorem



$\{x : x \notin f(x)\}$

Cantor's diagonal set

Cantor's Theorem

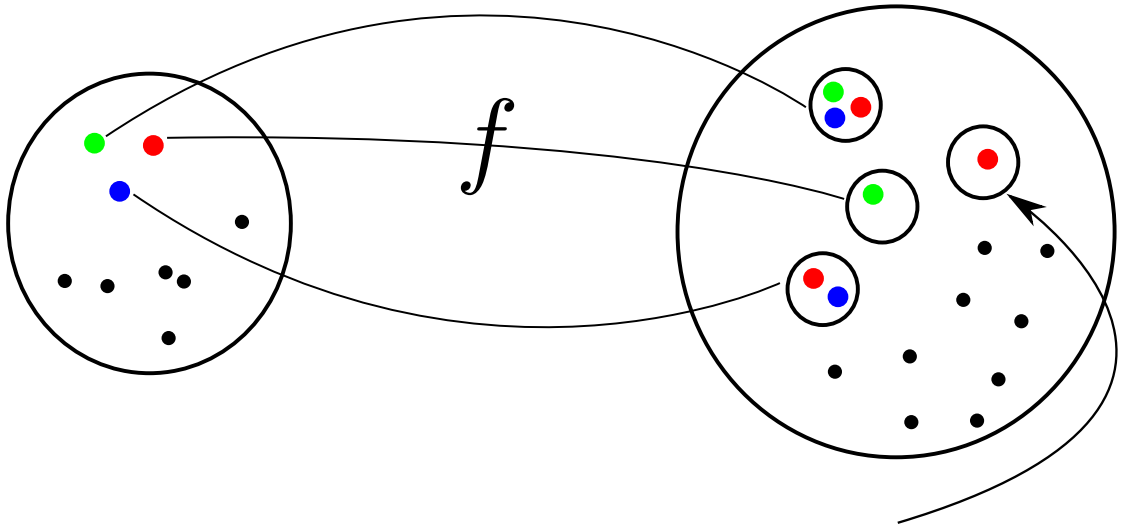


Cantor's diagonal set

Andrews-Miller-Cohen-Pfenning ('84)

$$\neg \exists f_{i \to i \to o} \forall b_{i \to o} \exists a_i : f(a) = b$$

Cantor's Theorem

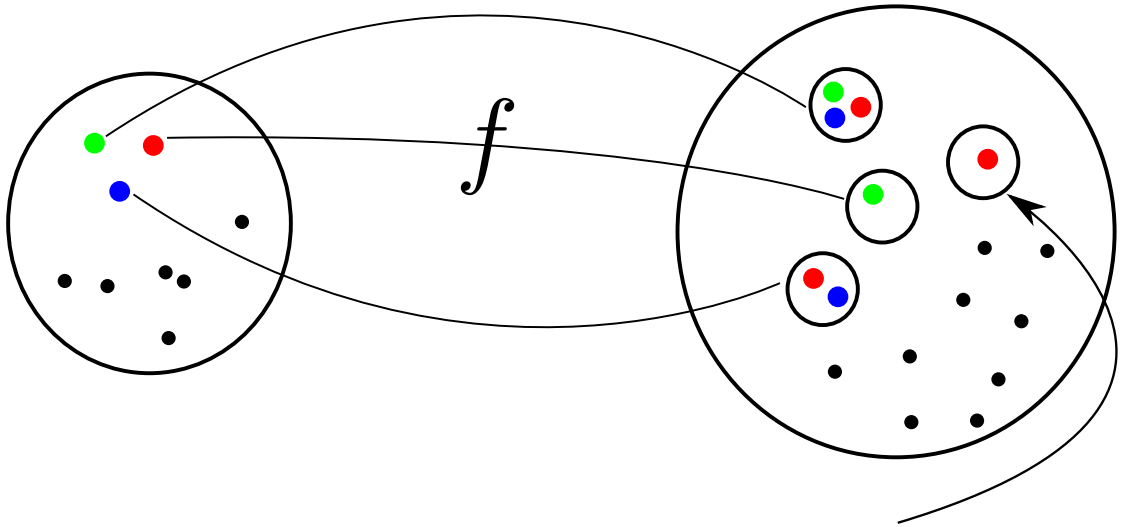


Andrews-Miller-Cohen-Pfenning ('84)

$$\neg \exists f_{i \to i \to o} \forall b_{i \to o} \exists a_i : f(a) = b$$

LEO-III HOL Theorem Prover

Cantor's Theorem



$\{x : x \notin f(x)\}$

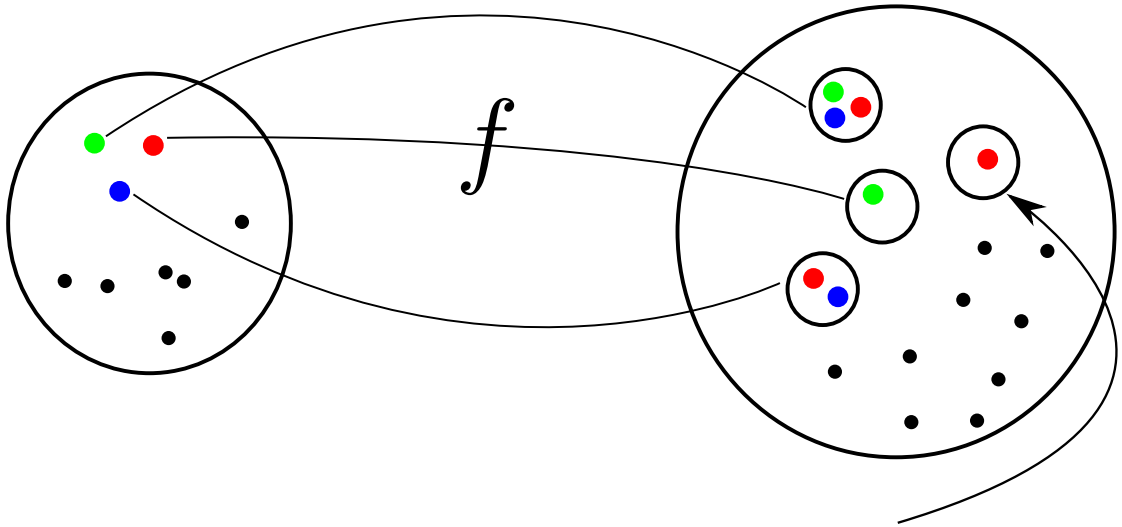
Cantor's diagonal set

Andrews-Miller-Cohen-Pfenning ('84)

$$\neg \exists f_{i \to i \to o} \forall b_{i \to o} \exists a_i : f(a) = b$$

$\lambda z . \neg f(z, z)$ ← LEO-III HOL Theorem Prover

Cantor's Theorem



$\{x : x \notin f(x)\}$

Cantor's diagonal set

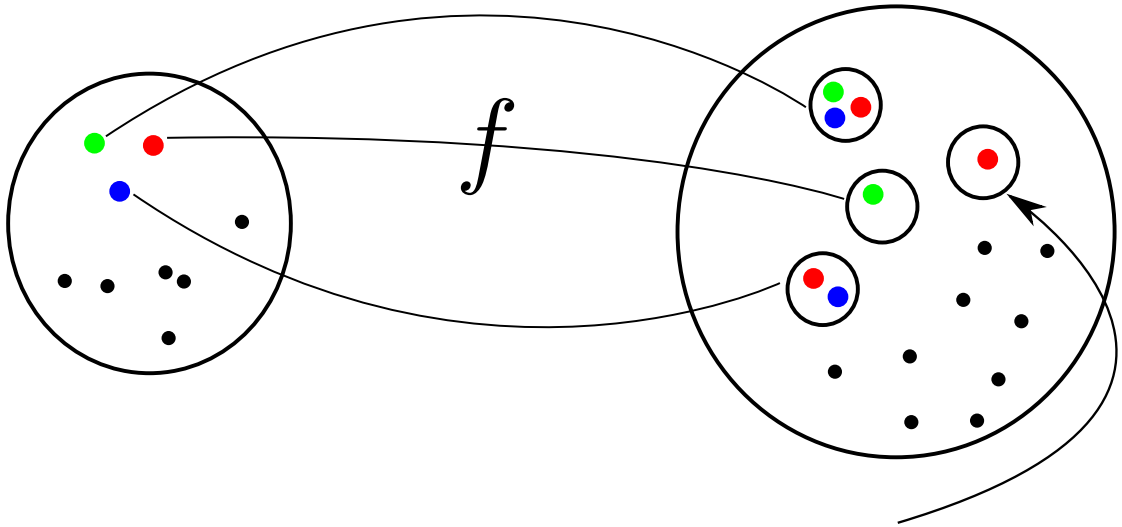
Andrews-Miller-Cohen-Pfenning ('84)

$$\neg \exists f_{i \to i \to o} \forall b_{i \to o} \exists a_i : f(a) = b$$

$$\downarrow ?$$

$\lambda z . \neg f(z, z)$ ← LEO-III HOL Theorem Prover

Cantor's Theorem



$\{x : x \notin f(x)\}$

Cantor's diagonal set

Andrews-Miller-Cohen-Pfenning ('84)

$$\neg \exists f_{i \to i \to o} \forall b_{i \to o} \exists a_i : f(a) = b$$

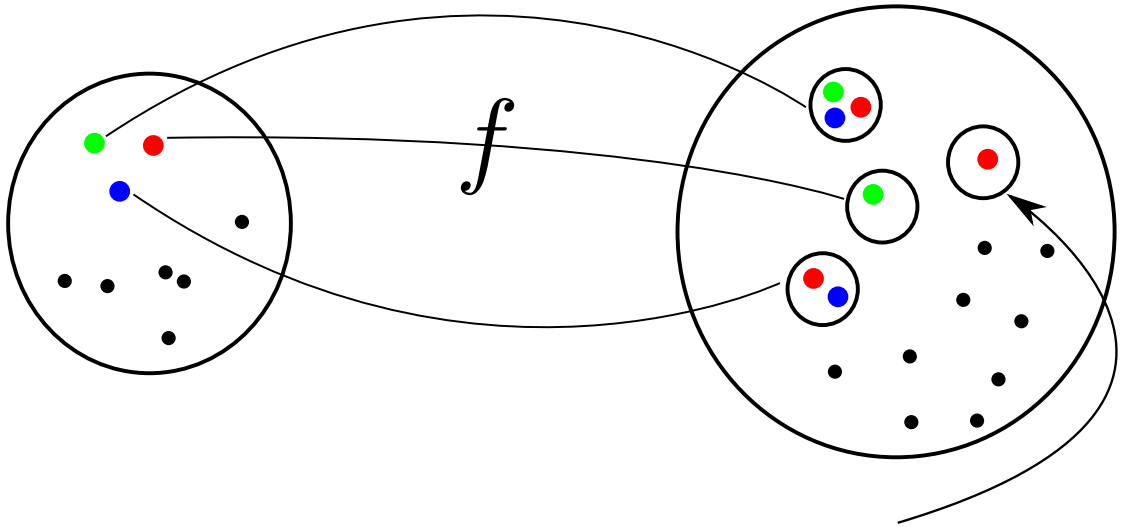
$$\downarrow ?$$

$$\lambda z . \neg f(z, z) \longleftarrow \text{LEO-III HOL Theorem Prover}$$

$$\uparrow \qquad \downarrow$$

Huet's pre-unification procedure ('75)

Cantor's Theorem



Andrews-Miller-Cohen-Pfenning ('84)

$$\neg \exists f_{i \to i \to o} \forall b_{i \to o} \exists a_i : f(a) = b$$

$\downarrow$?

$$\lambda z . \neg f(z, z)$$ ← LEO-III HOL Theorem Prover

↑ Huet's pre-unification procedure ('75)

- Semi-decidable

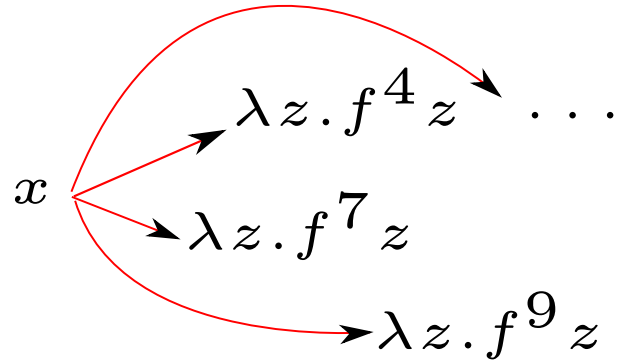
Forcing termination

Forcing termination

- Search space is prunned to be finite.

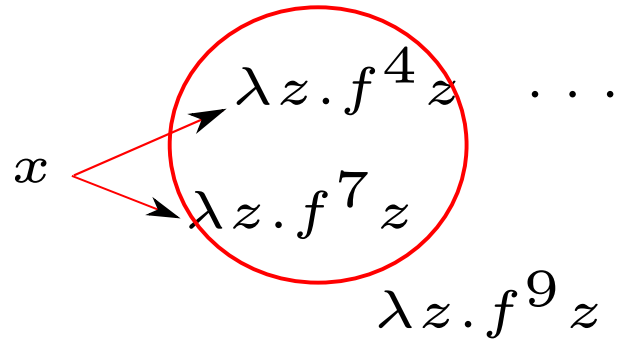
Forcing termination

- Search space is prunned to be finite.



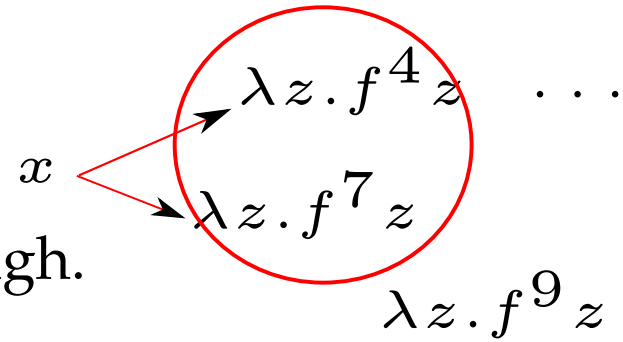
Forcing termination

- Search space is prunned to be finite.



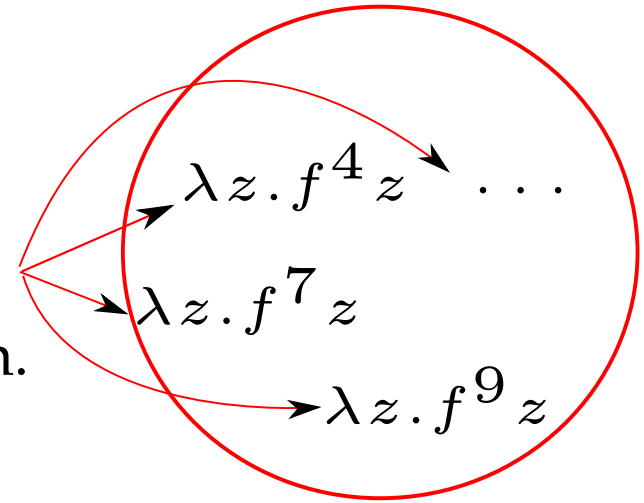
Forcing termination

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



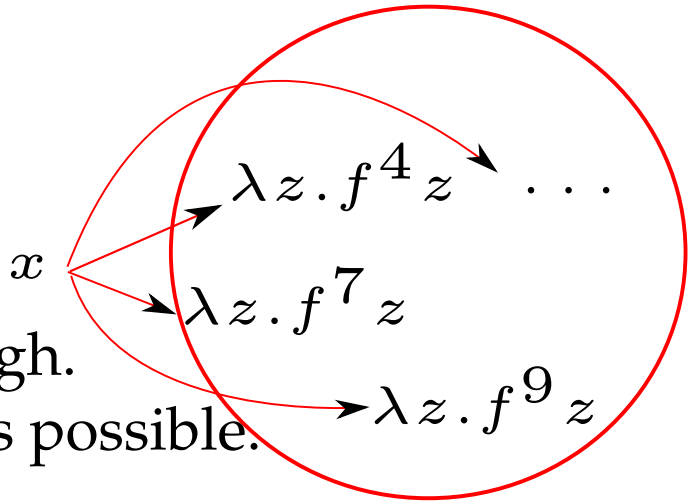
Forcing termination

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



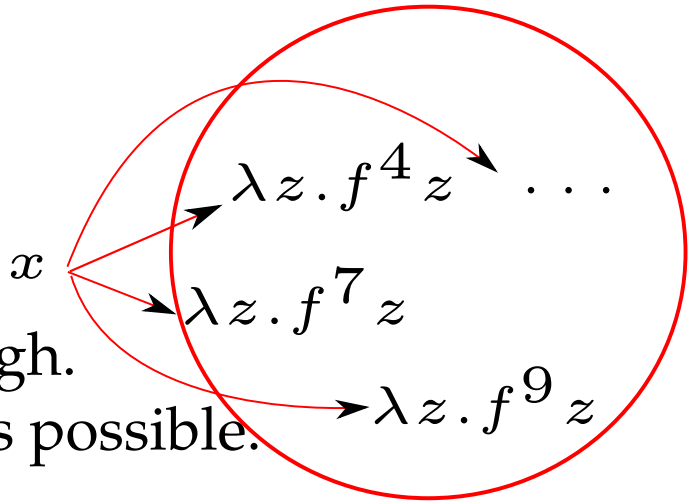
Forcing termination

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



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.



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

$$x\,0\,y \doteq \neg\, x\,0\,y$$

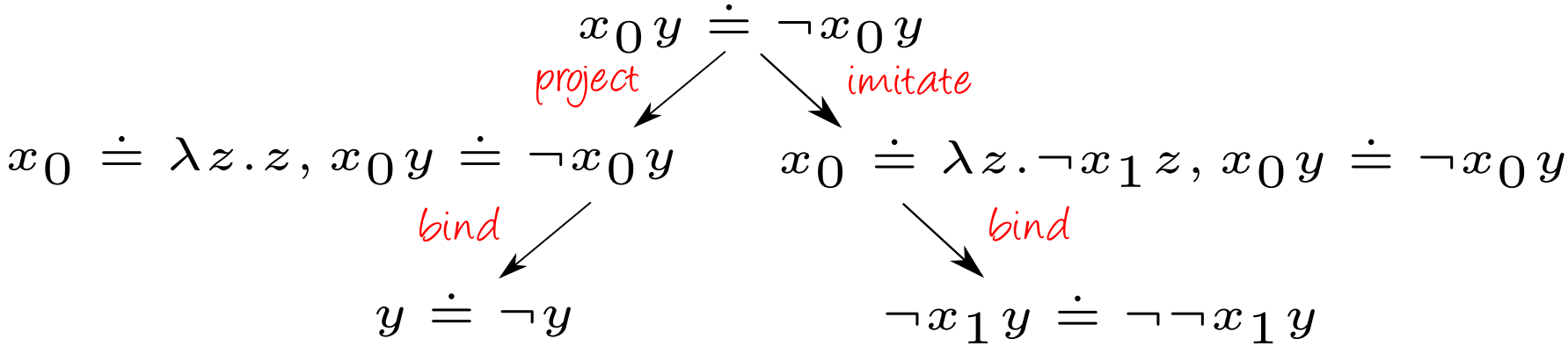
Non-termination

$$x_0\, y \doteq \neg x_0\, y$$

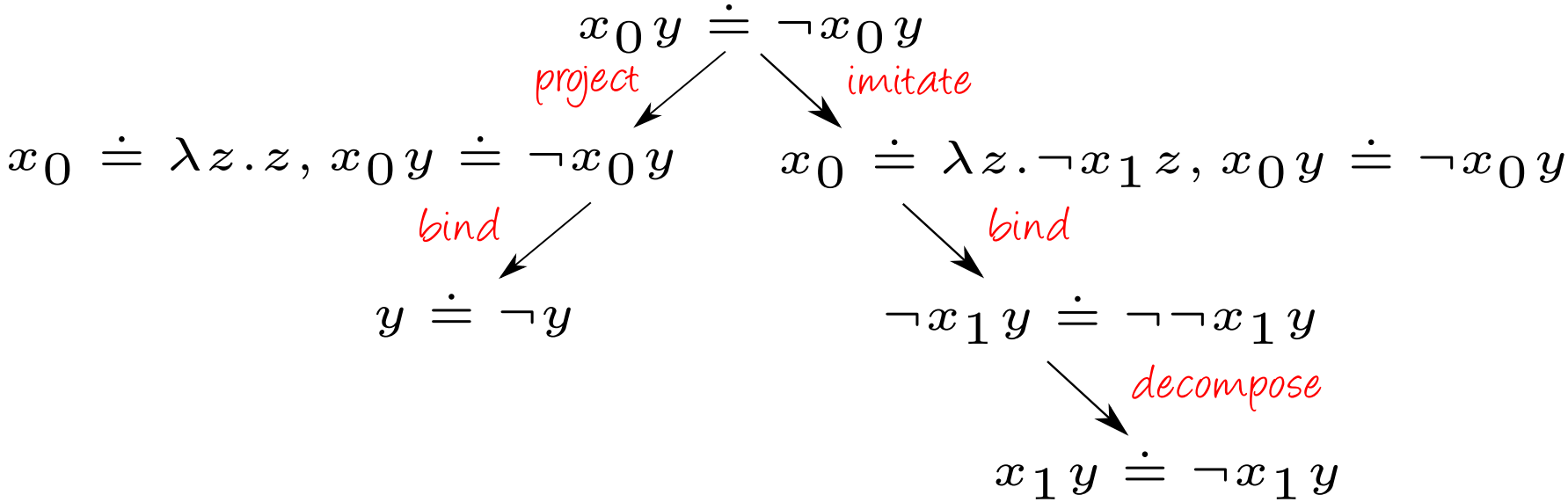
project: $x_0 \doteq \lambda z.\, z,\ x_0\, y \doteq \neg x_0\, y$

imitate: $x_0 \doteq \lambda z.\, \neg x_1\, z,\ x_0\, y \doteq \neg x_0\, y$

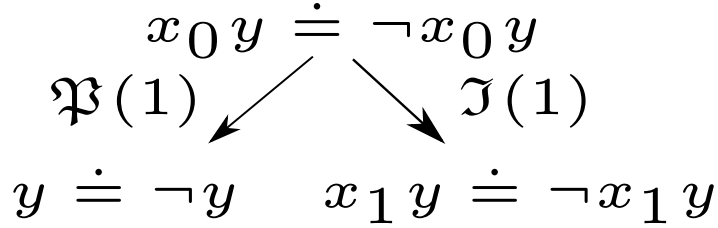
Non-termination



Non-termination



Non-termination



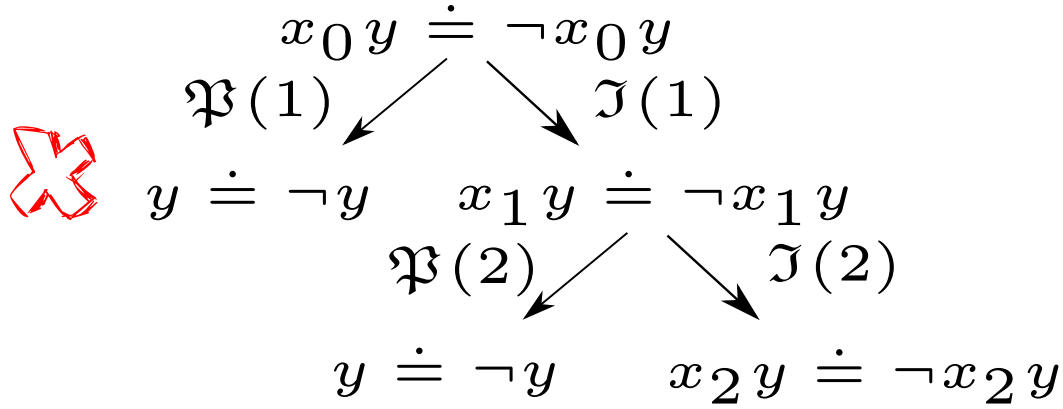
Non-termination

$$x_0\, y \doteq \neg x_0\, y$$

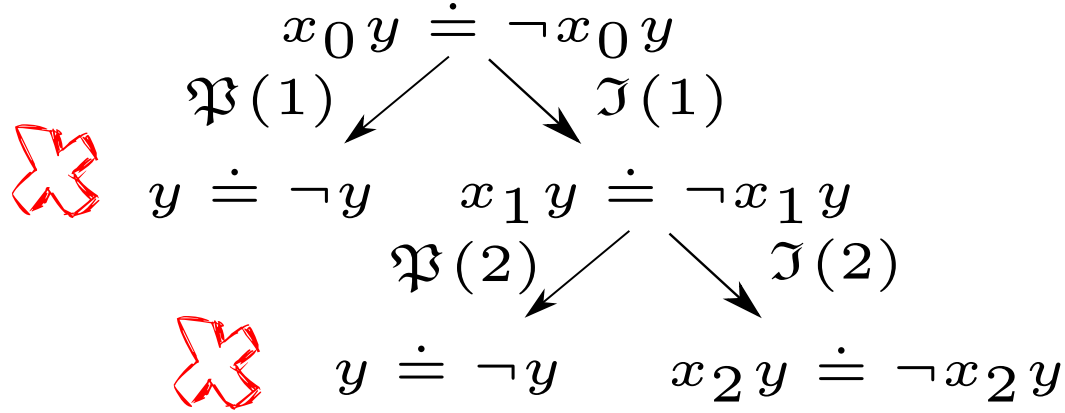
$\mathfrak{P}(1)$ ↙ ↘ $\mathfrak{I}(1)$

$$y \doteq \neg y \qquad x_1\, y \doteq \neg x_1\, y$$

Non-termination

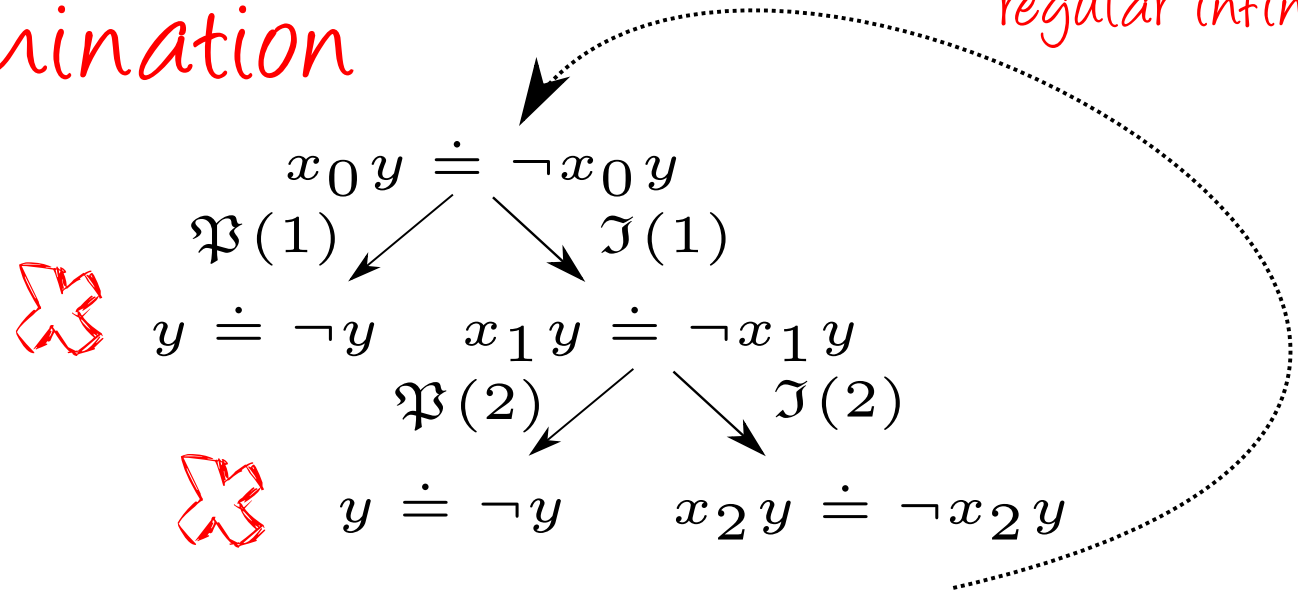


Non-termination

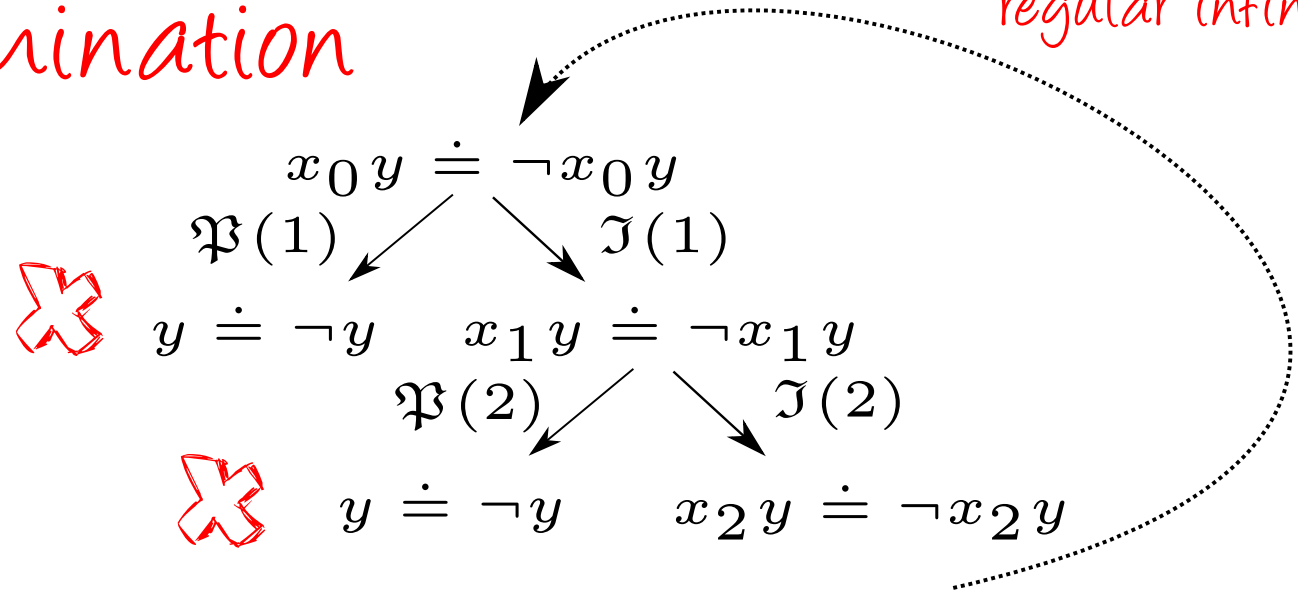


Non-termination

regular infinite tree

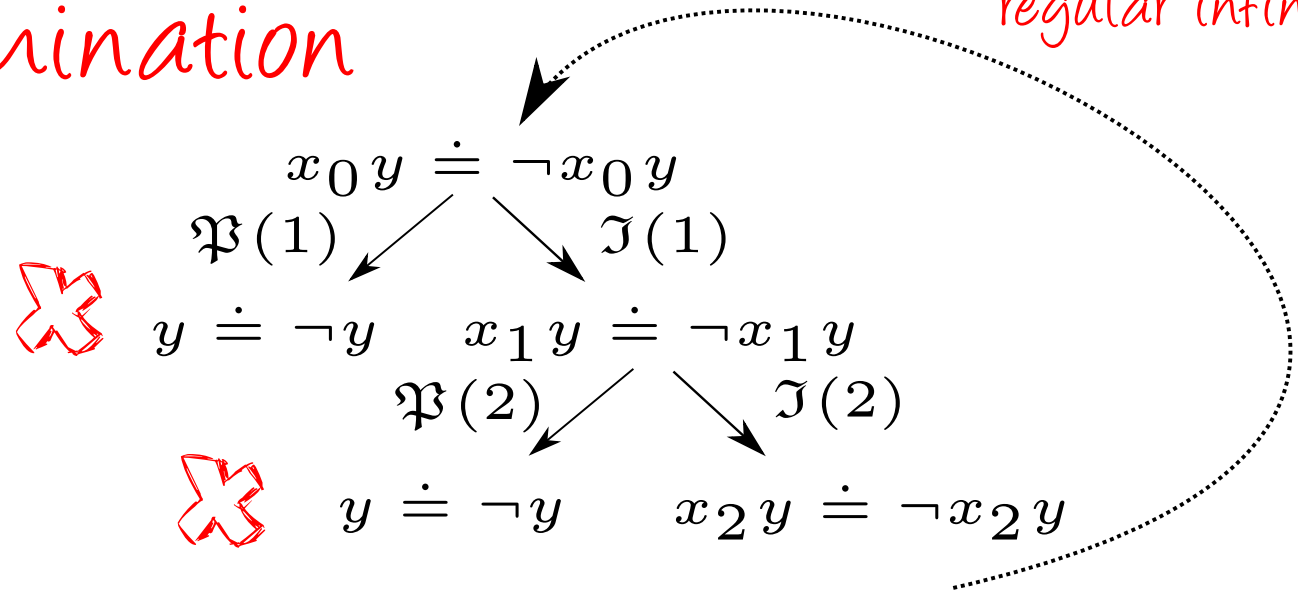


Non-termination



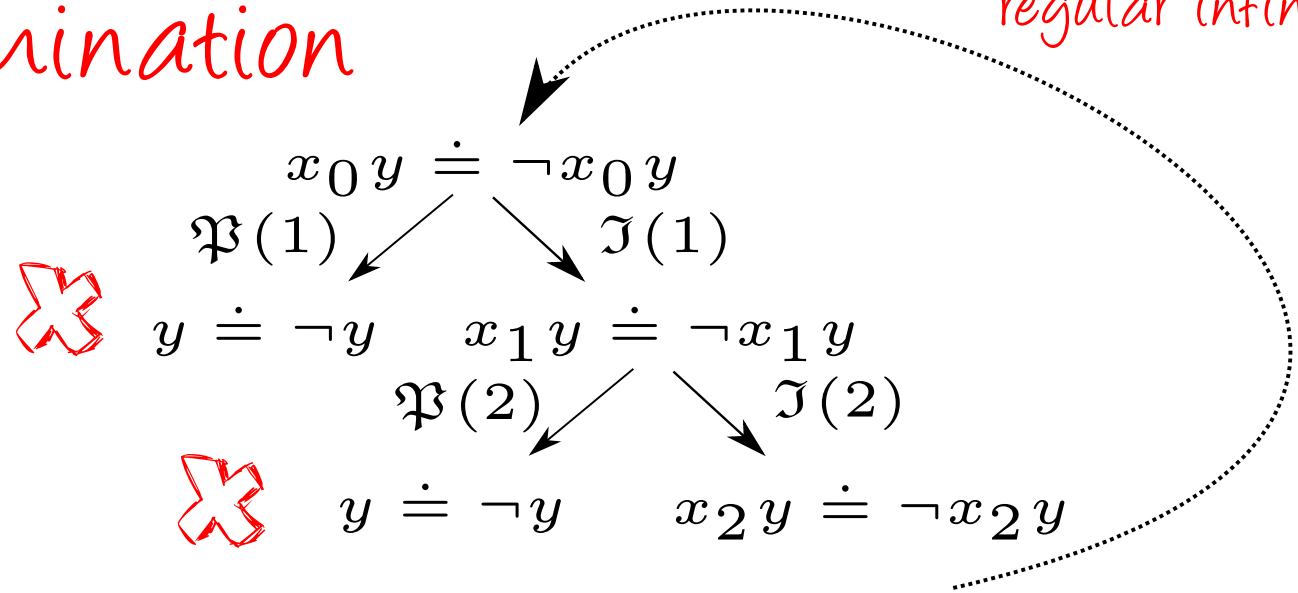
- Semi-decidable: possible non-termination only if not unifiable.

Non-termination



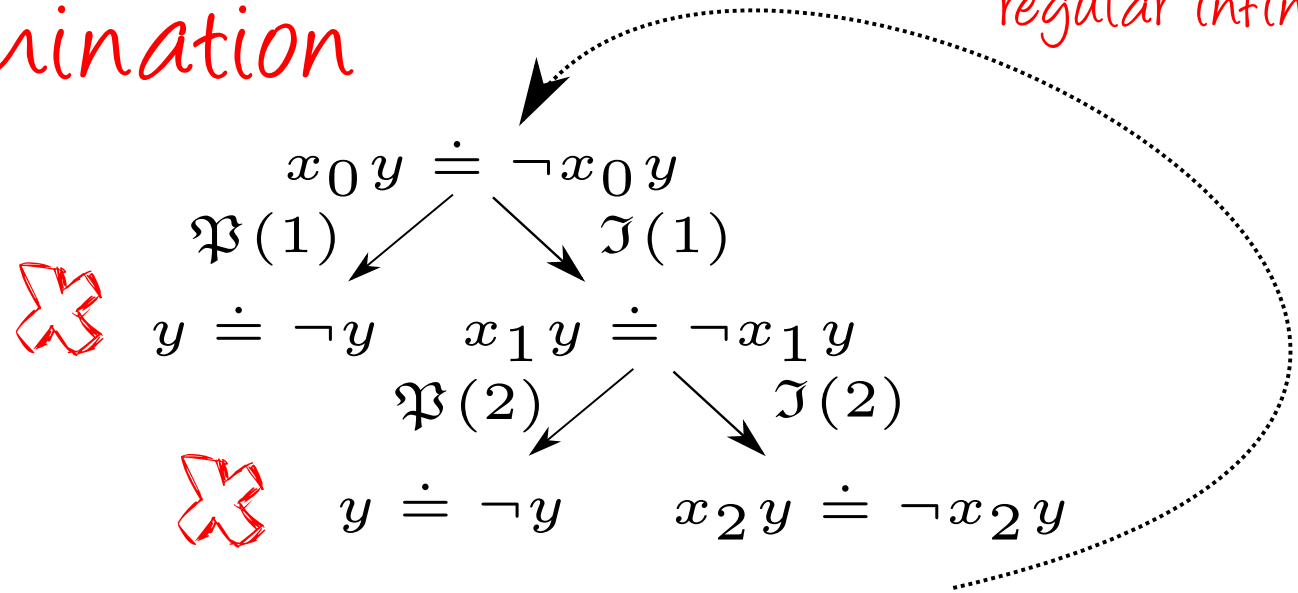
- Semi-decidable: possible non-termination only if not unifiable.
- Levy ('98): possible non-termination only if we can encounter cycles.

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 $\exists i > 0\, \mathfrak{P}(i)$ 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 $\exists i > 0\ \mathfrak{P}(i)$ is unifiable.
- Lemma 2: $\forall i, j > 0\ \mathfrak{P}(i)$ is unifiable if $\mathfrak{P}(j)$ is.

Cyclic equations - monadic signature

$$x_0\, t \doteq C(x_0\, s)$$

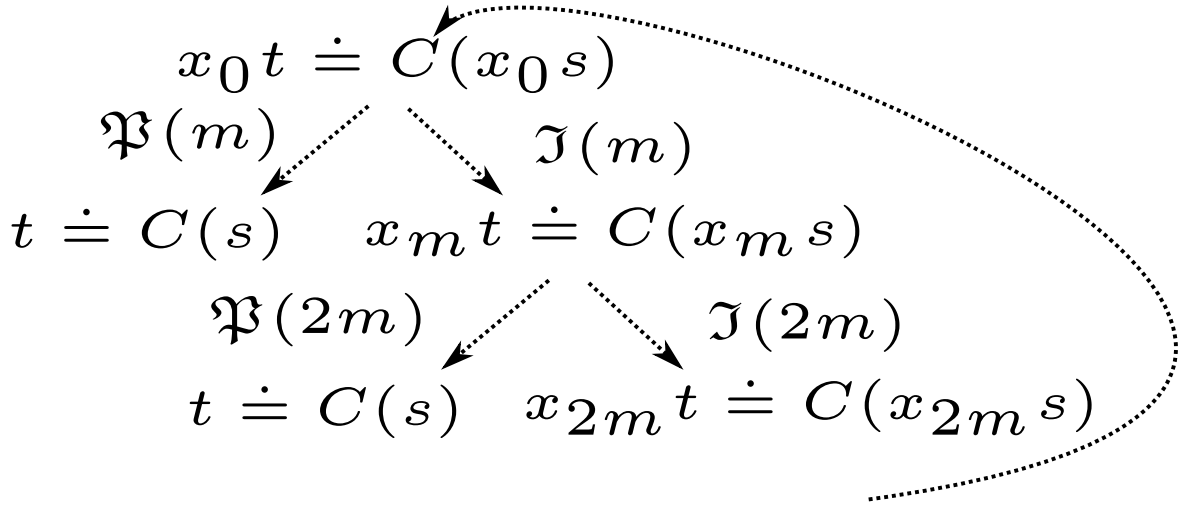
Cyclic equations - monadic signature

$$x_0\, t \doteq C(x_0\, s)$$

$$x_0 \notin \mathtt{FV}(t, C, s)$$

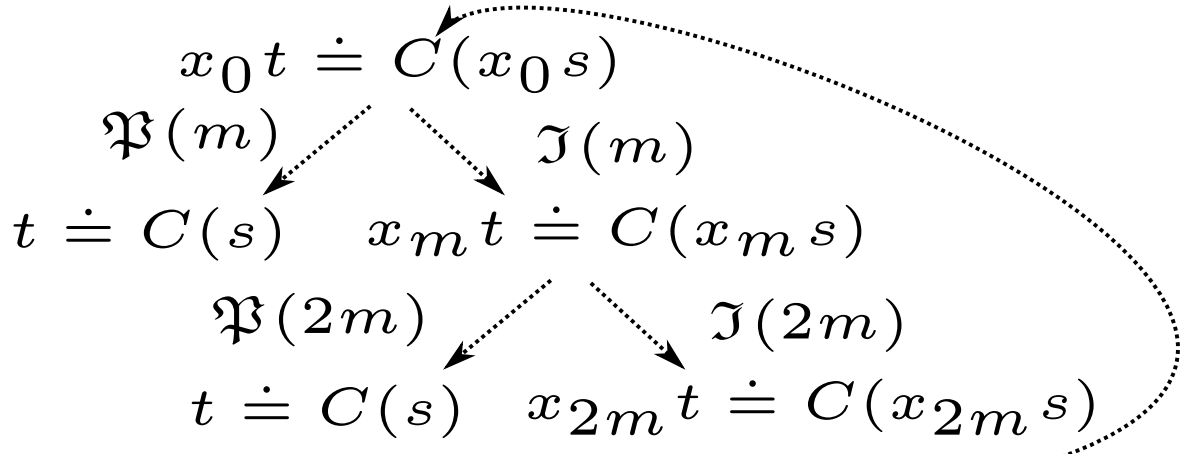
Cyclic equations - monadic signature

$$x_0 t \doteq C(x_0 s)$$



Cyclic equations - monadic signature

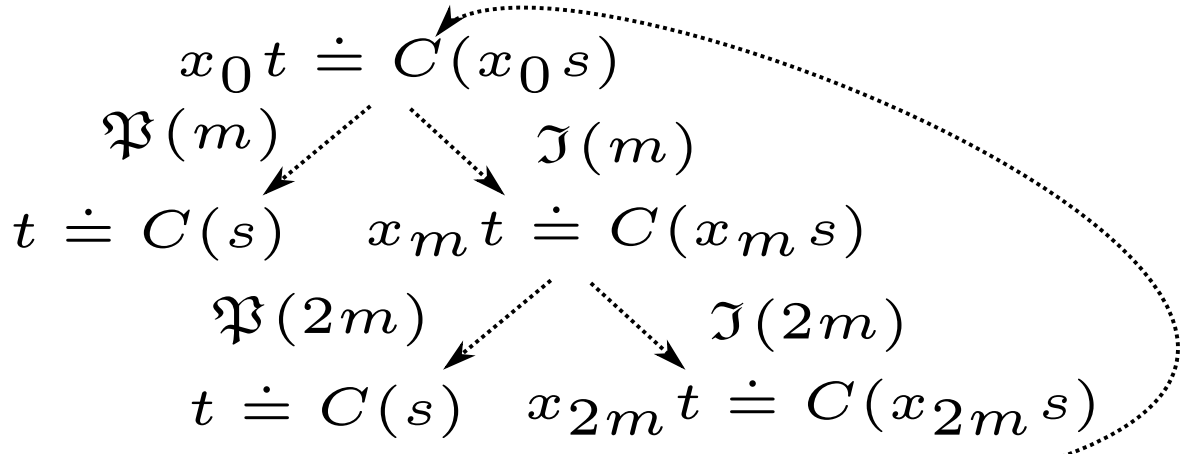
$$x_0 t \doteq C(x_0 s)$$



Lemma 1 Lemma 2

Cyclic equations - monadic signature

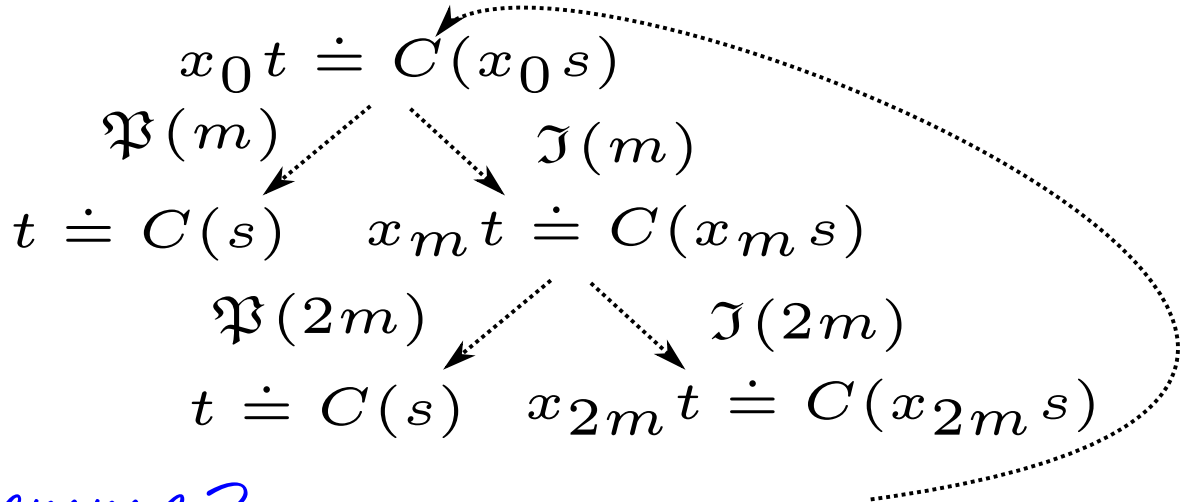
$$x_0 t \doteq C(x_0 s)$$



Lemma 1 Lemma 2

Cyclic equations - monadic signature

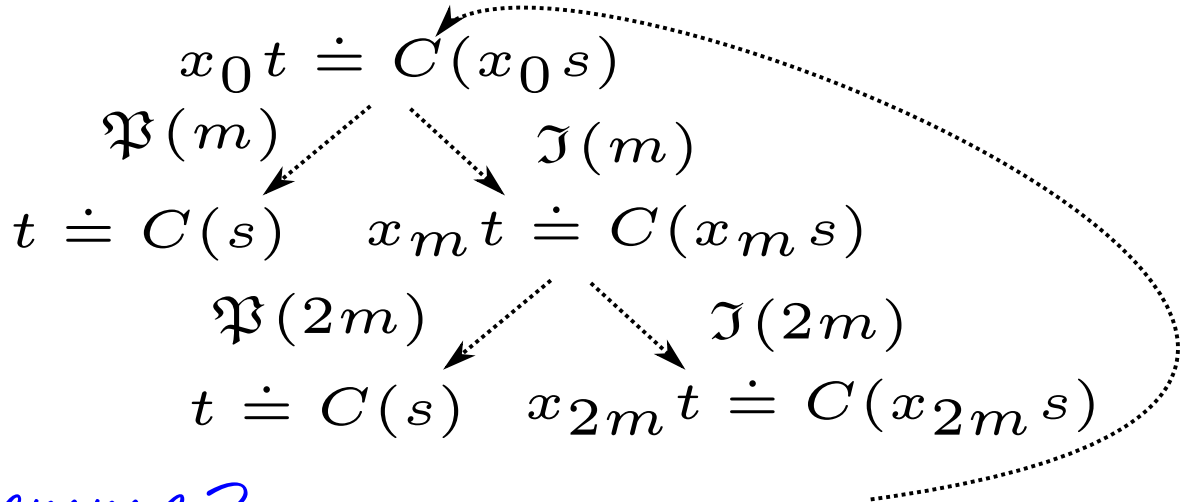
$$x_0 t \doteq C(x_0 s)$$



Lemma 1 ✓ Lemma 2 ✓

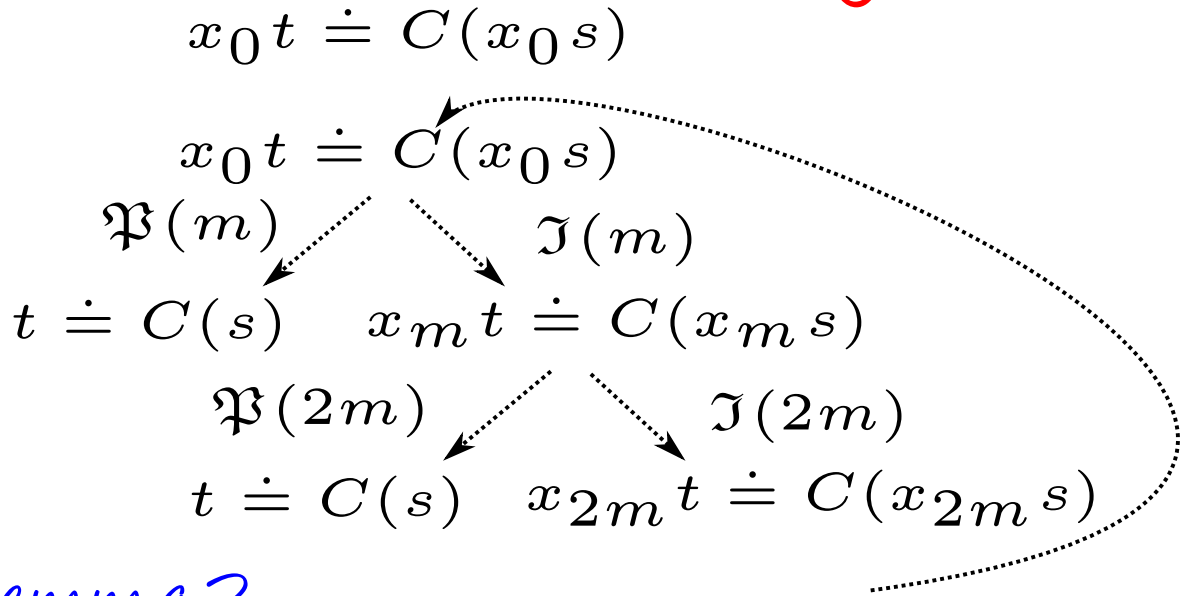
Cyclic equations - monadic signature

$$x_0 t \doteq C(x_0 s)$$



Lemma 1 ✓ Lemma 2 ✓

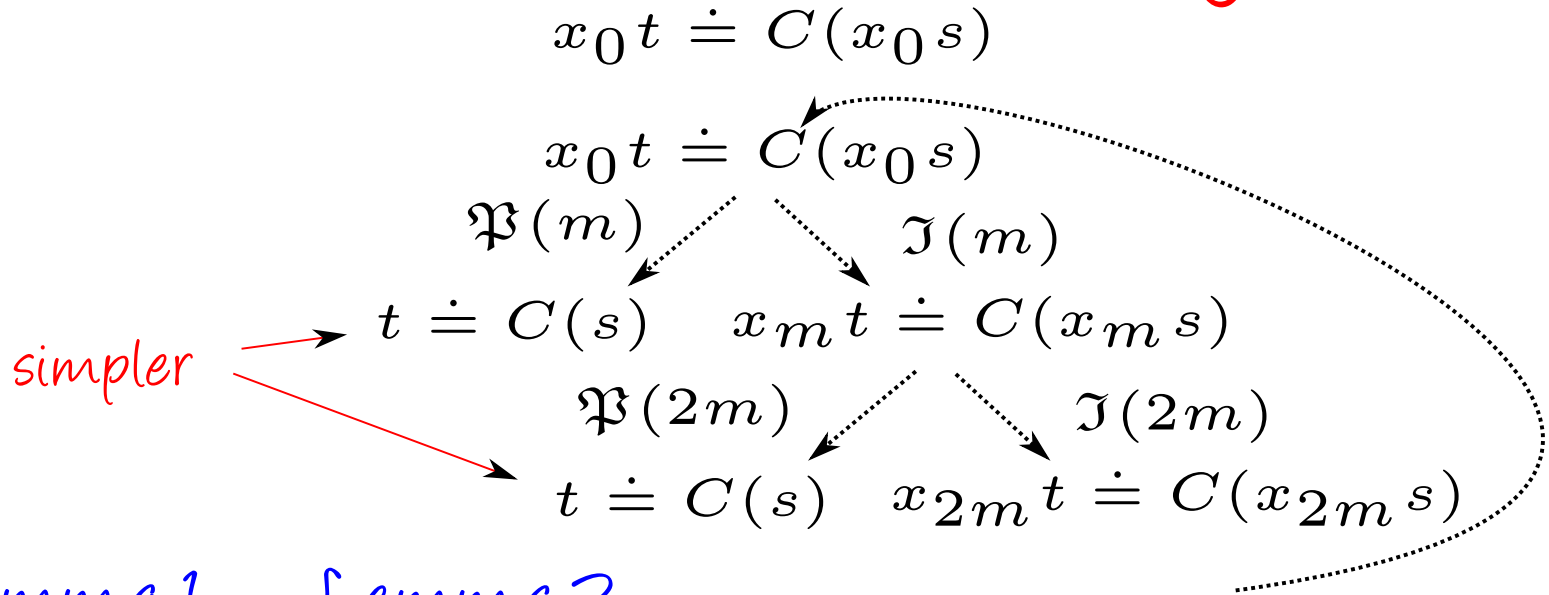
Cyclic equations - monadic signature



Lemma 1 ✓ Lemma 2 ✓

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

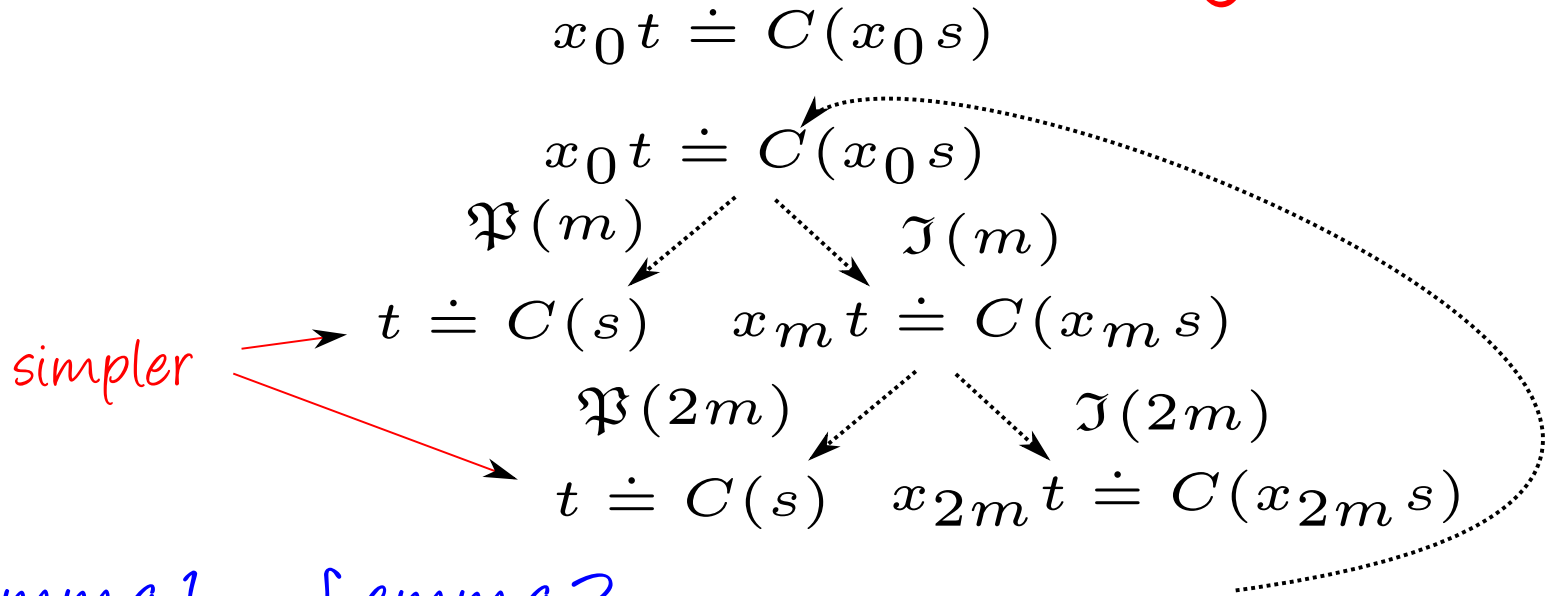
Cyclic equations - monadic signature



Lemma 1 ✓ Lemma 2 ✓

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

Cyclic equations - monadic signature



Lemma 1 ✓ Lemma 2 ✓

- Theorem: e is unifiable iff $\exists 0 \leq i \leq m$ s.t. $\mathfrak{P}(i)$ is unifiable.
- Corollary:Unification over monadic "cyclic equations" is decidable. (Farmer '88 for full monadic SOU)

Non-monadic cyclic equations

$$x_0(\neg y_1) \doteq x_0 y_2 \vee y_3$$

Non-monadic cyclic equations

$$x_0(\neg y_1) \doteq x_0 y_2 \vee y_3$$

$\mathfrak{P}(1)$ $\qquad$ $\mathfrak{I}(1)$

$$\neg y_1 \doteq y_2 \vee y_3 \qquad x_1(\neg y_1) \doteq x_1 y_2 \vee w_1 y2 \;\Big|\; w_1 y_1 \doteq y_3$$

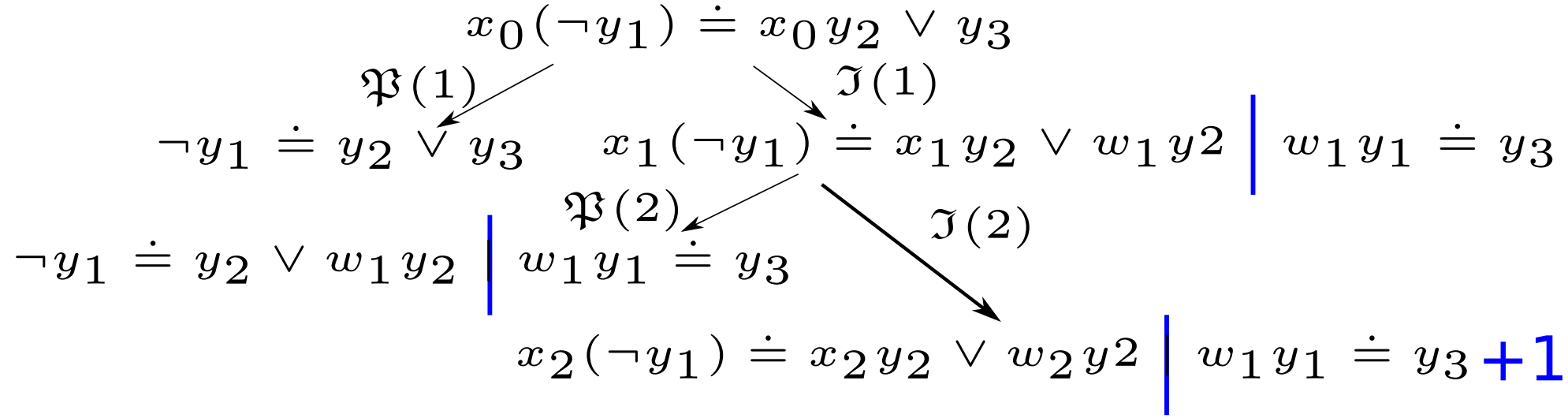
Non-monadic cyclic equations

$$x_0(\neg y_1) \doteq x_0 y_2 \vee y_3$$

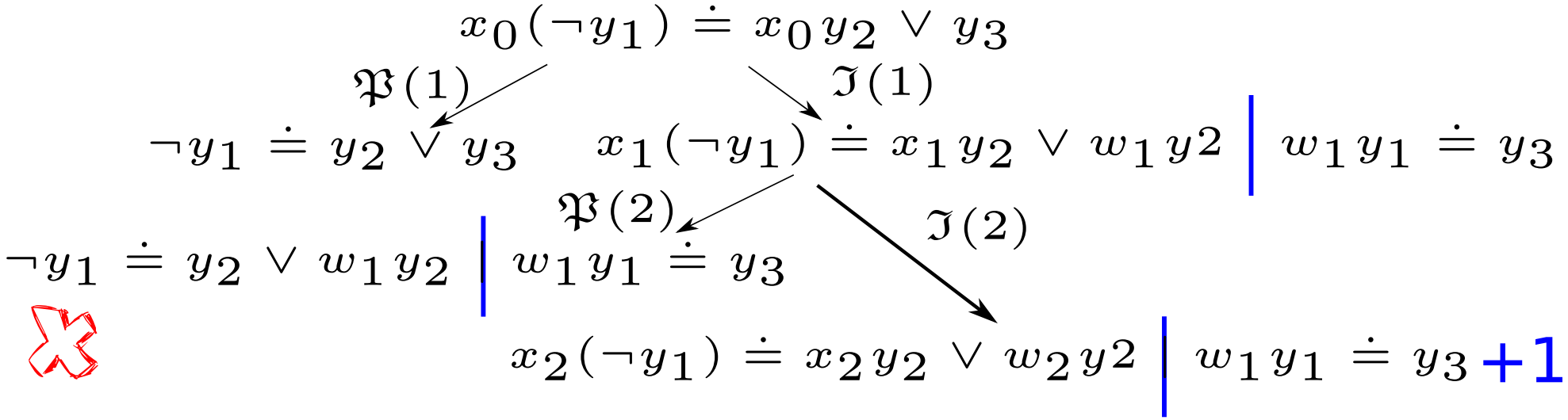
$\mathfrak{P}(1)$ ↙ ↘ $\mathfrak{I}(1)$

$\neg y_1 \doteq y_2 \vee y_3$ $\quad x_1(\neg y_1) \doteq x_1 y_2 \vee w_1 y2$ | $w_1 y_1 \doteq y_3$

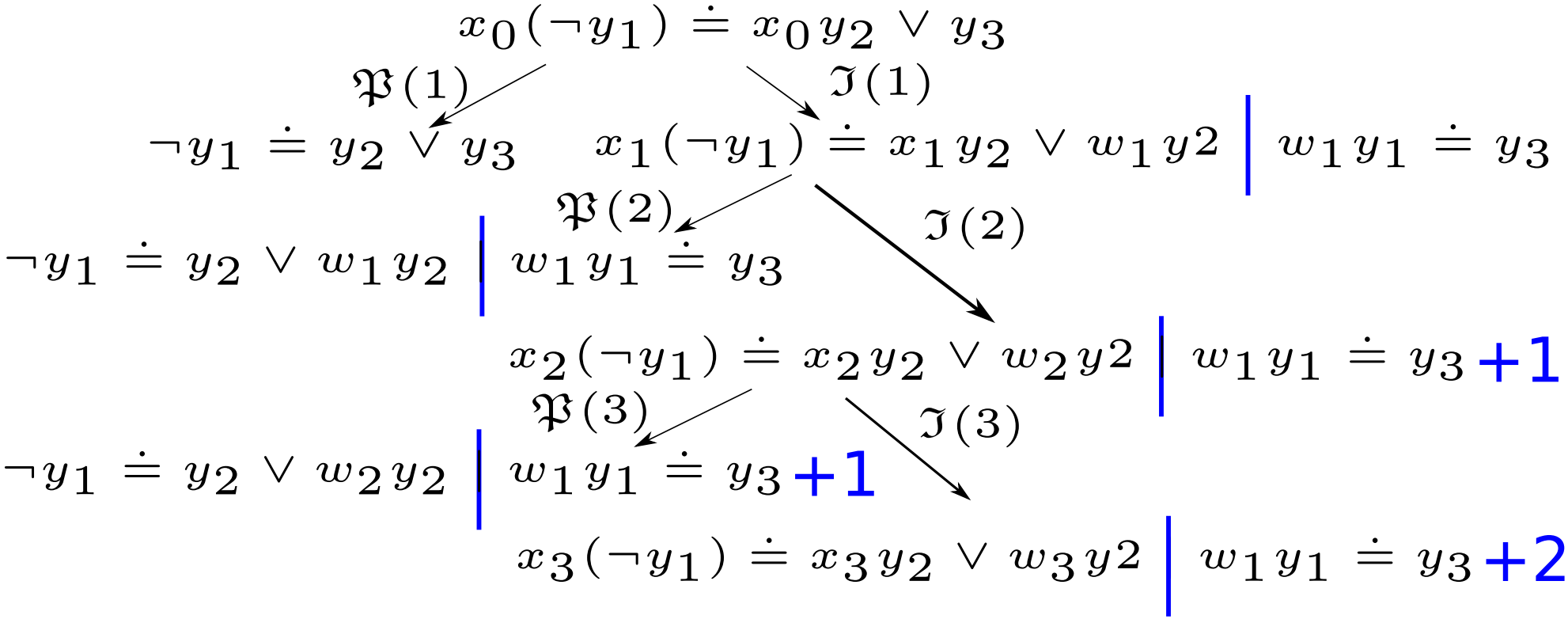
Non-monadic cyclic equations



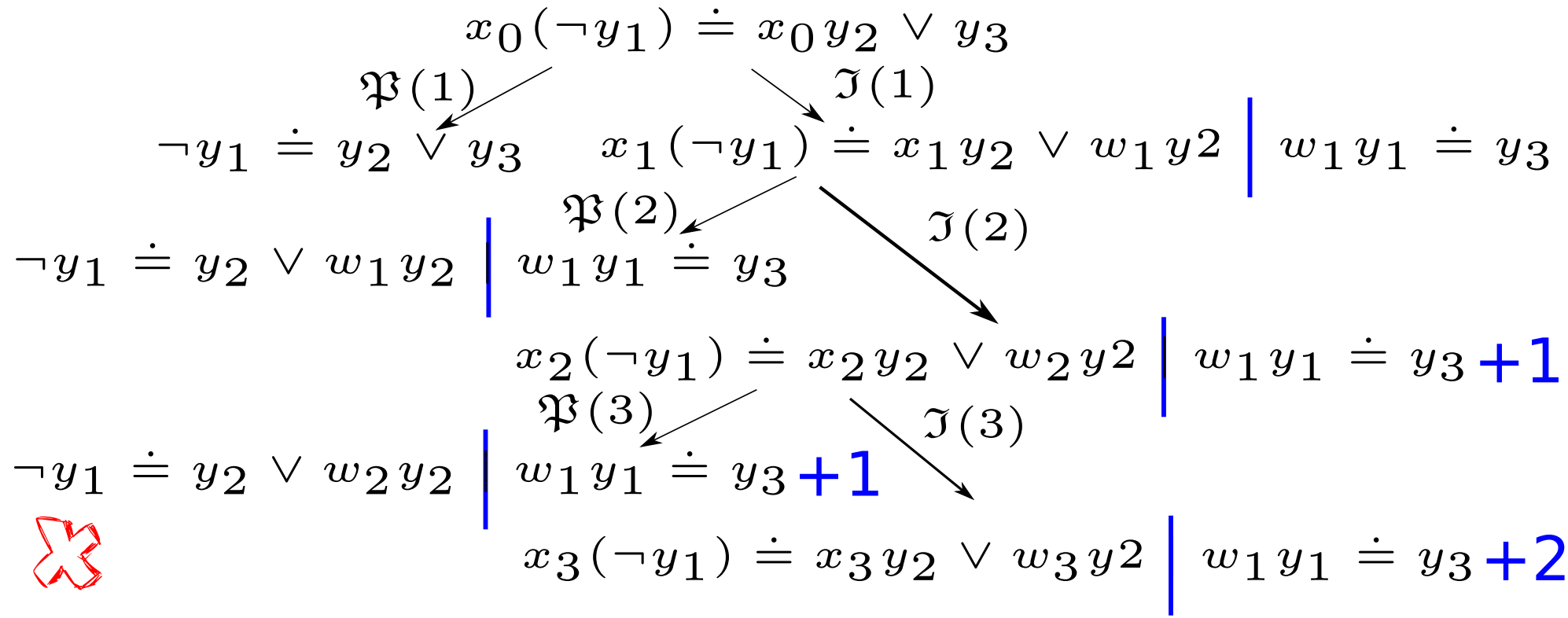
Non-monadic cyclic equations



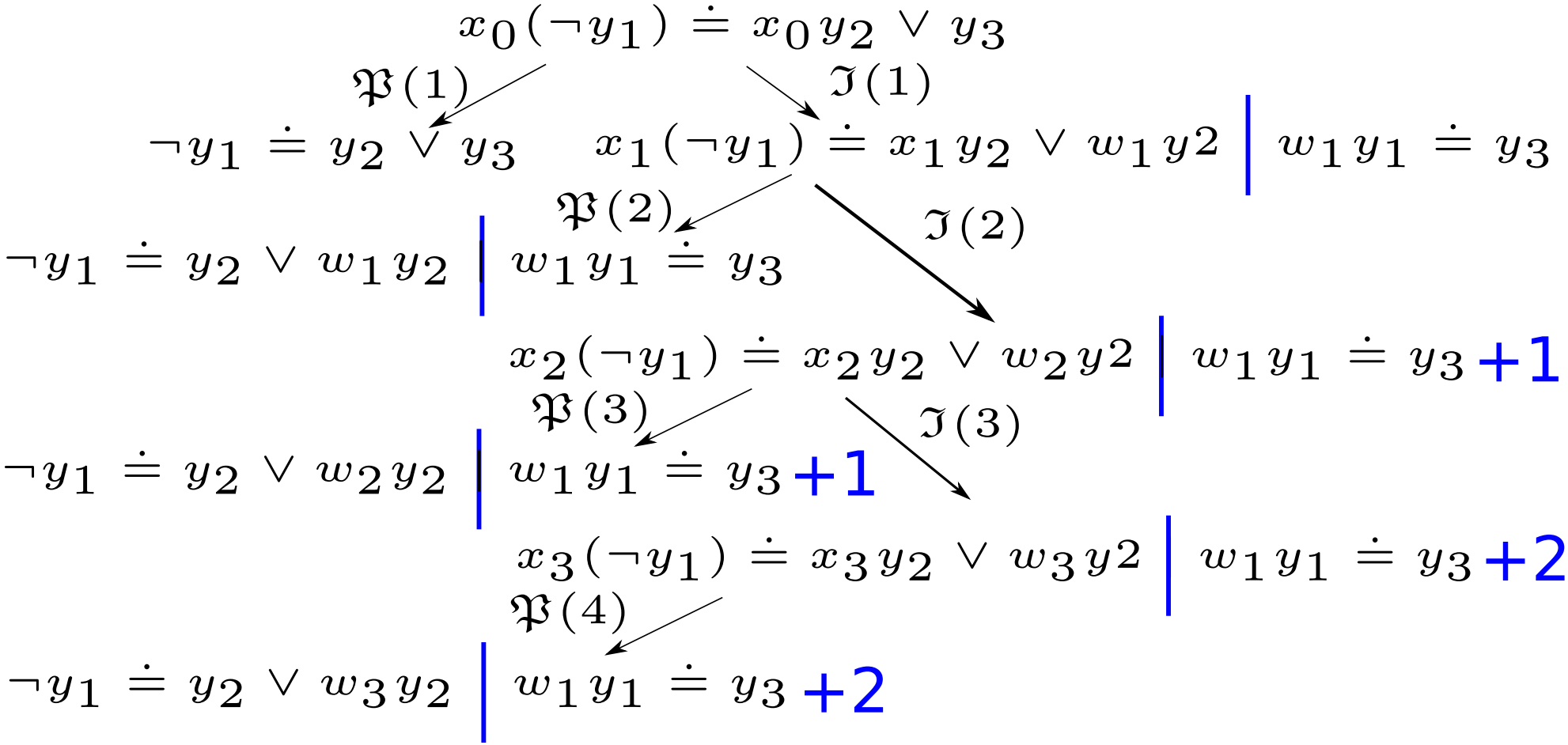
Non-monadic cyclic equations



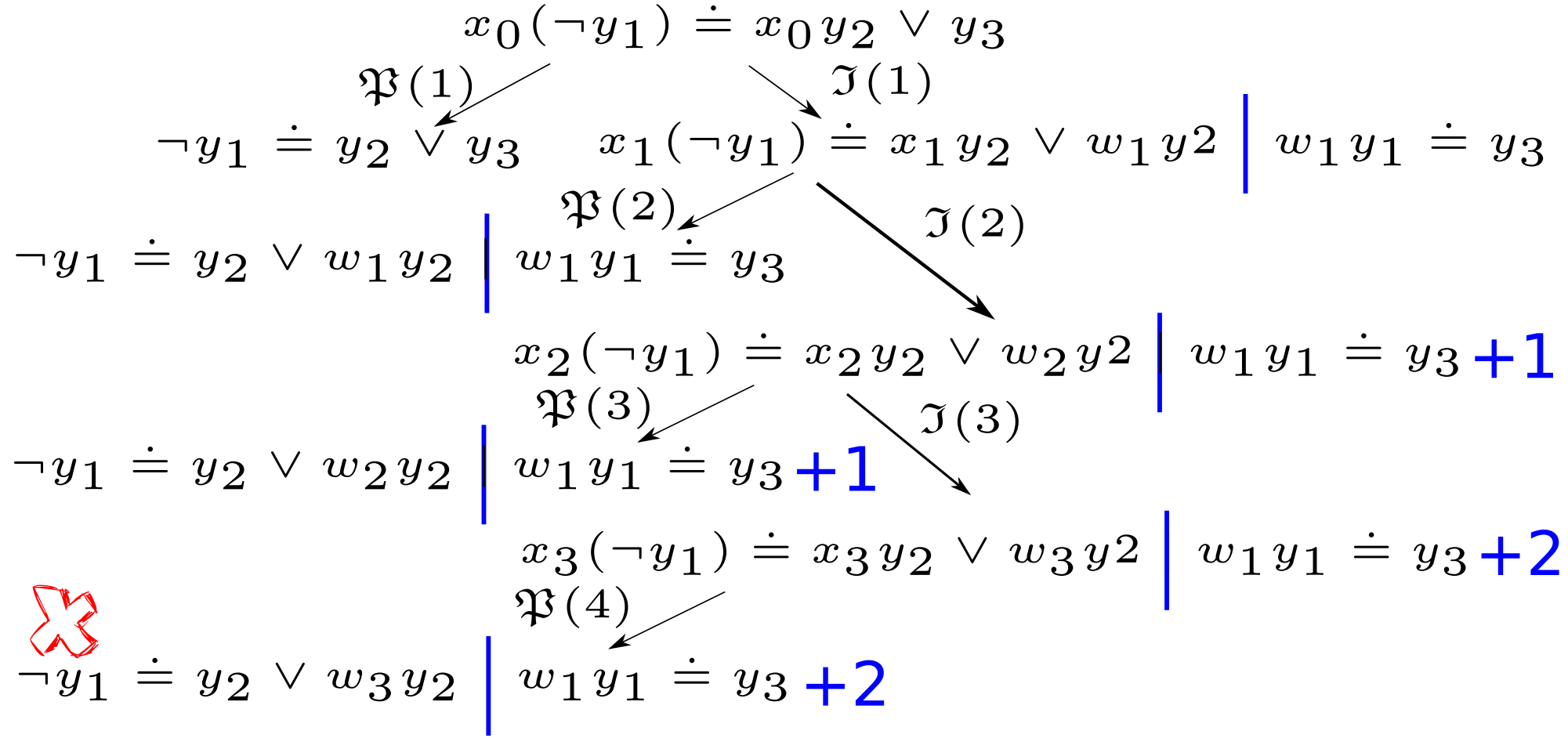
Non-monadic cyclic equations



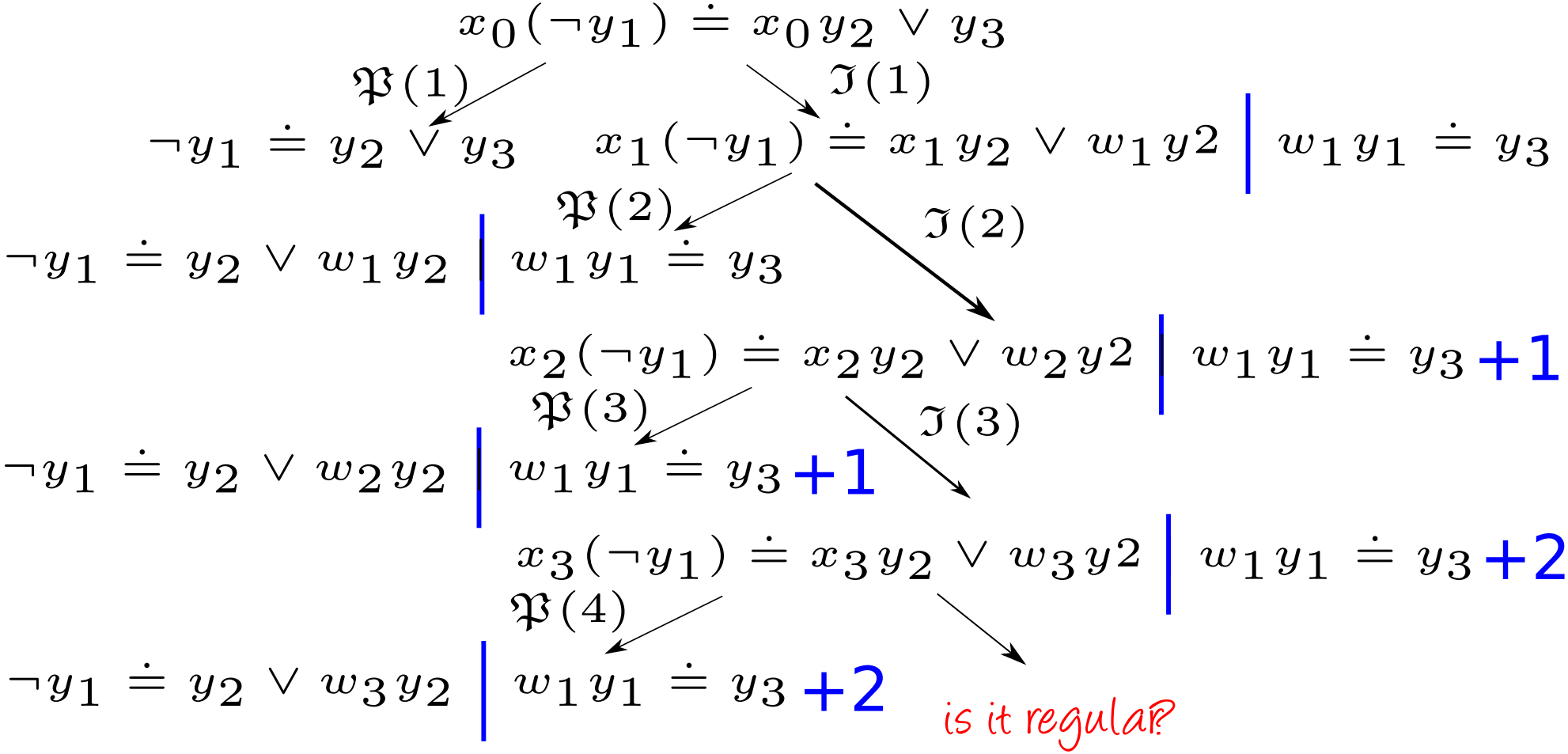
Non-monadic cyclic equations



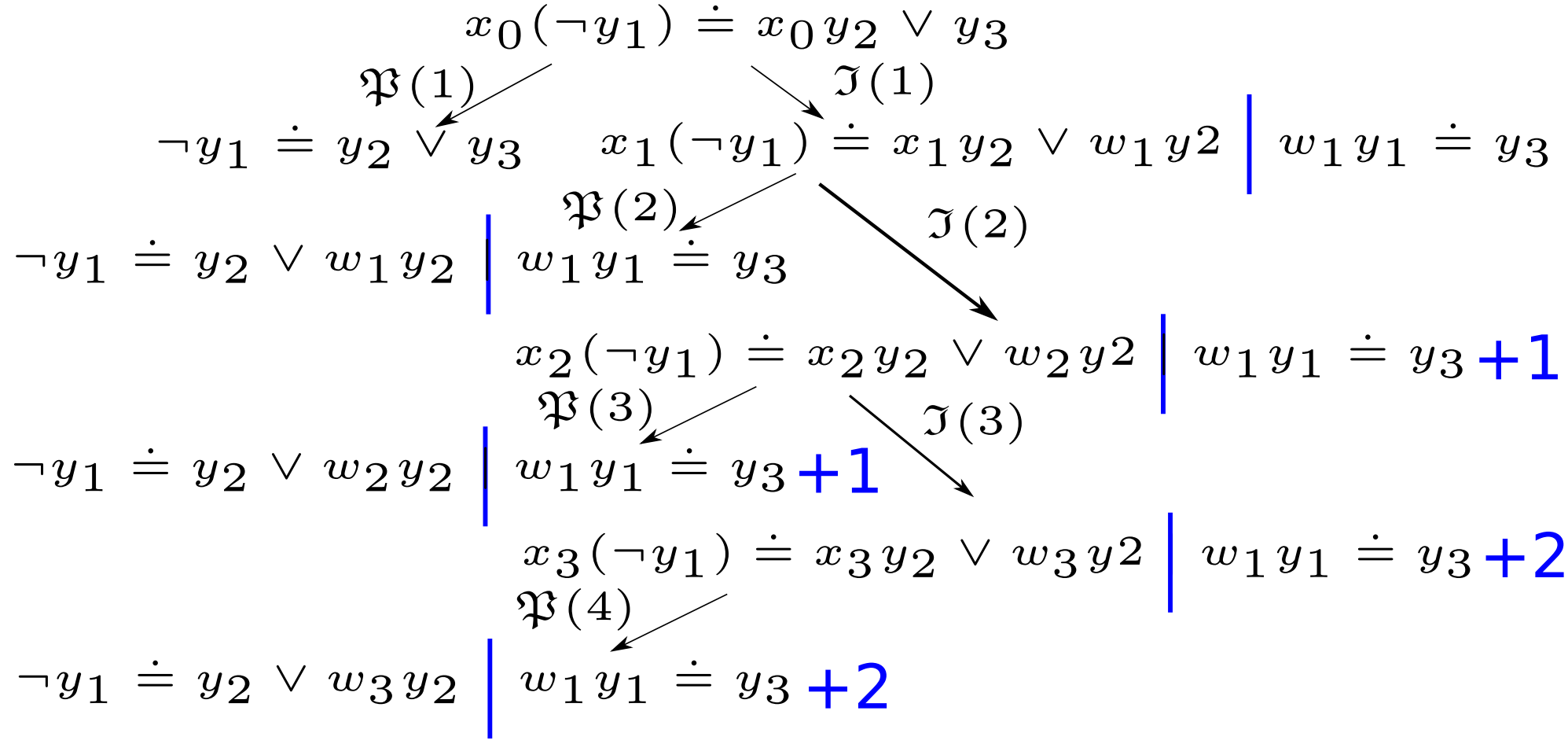
Non-monadic cyclic equations



Non-monadic cyclic equations

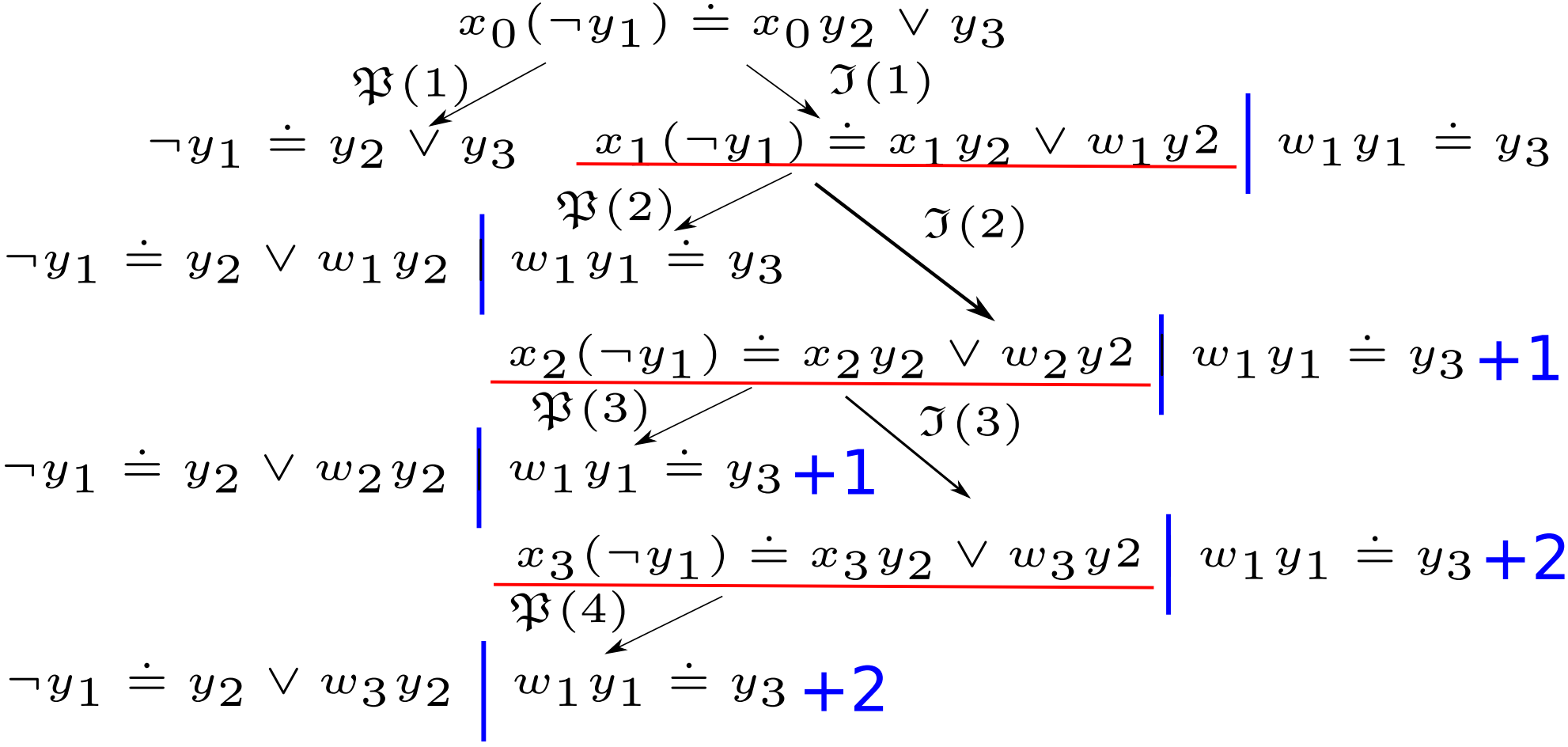


Non-monadic cyclic equations



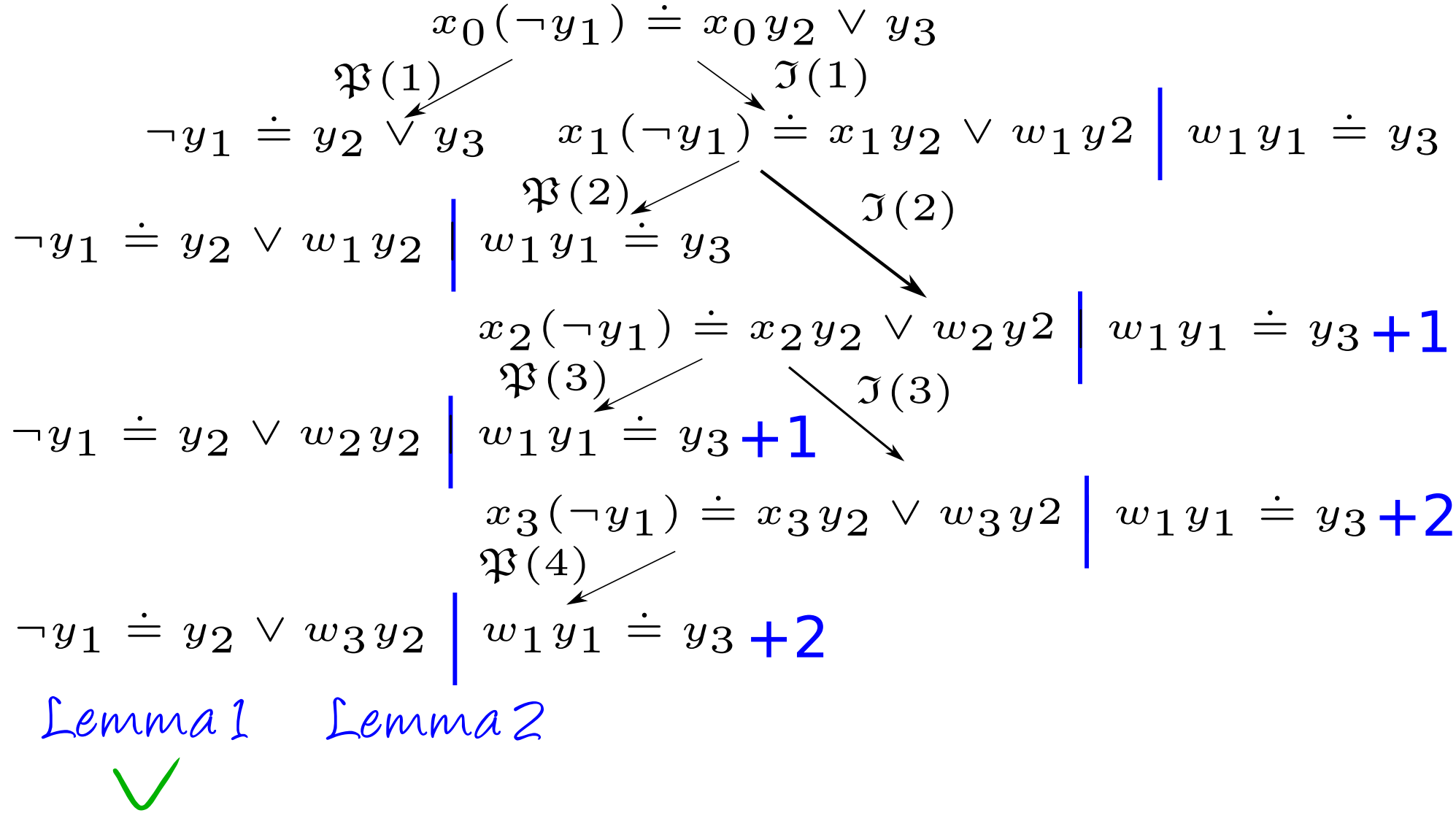
Lemma 1 Lemma 2

Non-monadic cyclic equations



Lemma 1 Lemma 2

Non-monadic cyclic equations



Lemma 1 Lemma 2

Non-monadic cyclic equations

$$x_0(\neg y_1) \doteq x_0 y_2 \vee y_3$$

$\mathfrak{P}(1)$ $\qquad$ $\mathfrak{I}(1)$

$$\neg y_1 \doteq y_2 \vee y_3 \qquad x_1(\neg y_1) \doteq x_1 y_2 \vee w_1 y2 \mid w_1 y_1 \doteq y_3$$

$\mathfrak{P}(2)$ $\qquad$ $\mathfrak{I}(2)$

$$\neg y_1 \doteq y_2 \vee w_1 y_2 \mid w_1 y_1 \doteq y_3 \qquad x_2(\neg y_1) \doteq x_2 y_2 \vee w_2 y2 \mid w_1 y_1 \doteq y_3 + 1$$

$\mathfrak{P}(3)$ $\qquad$ $\mathfrak{I}(3)$

$$\neg y_1 \doteq y_2 \vee w_2 y_2 \mid w_1 y_1 \doteq y_3 + 1 \qquad x_3(\neg y_1) \doteq x_3 y_2 \vee w_3 y2 \mid w_1 y_1 \doteq y_3 + 2$$

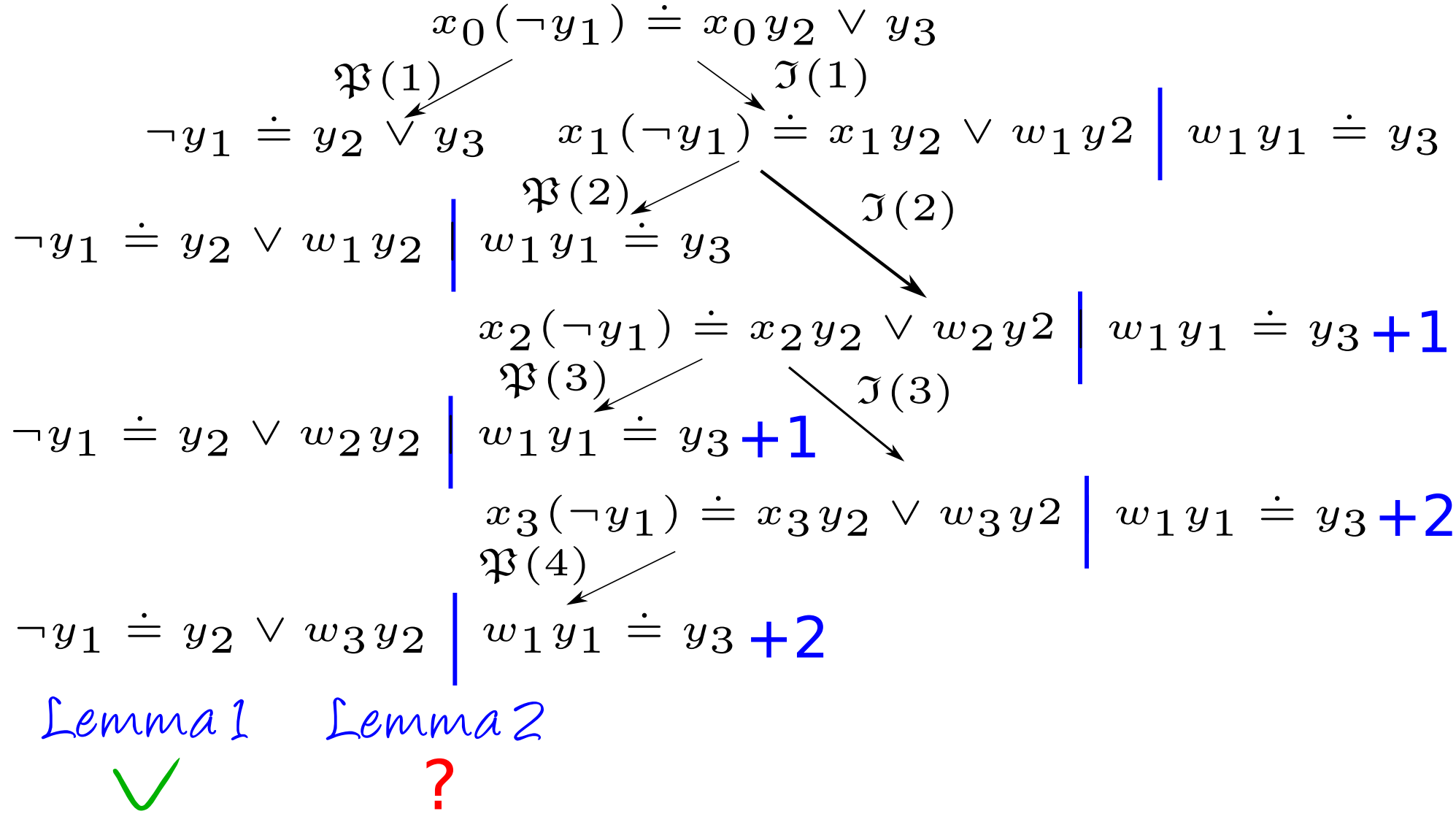
$\mathfrak{P}(4)$

$$\neg y_1 \doteq y_2 \vee w_3 y_2 \mid w_1 y_1 \doteq y_3 + 2$$

Lemma 1 $\qquad$ Lemma 2

$\vee$

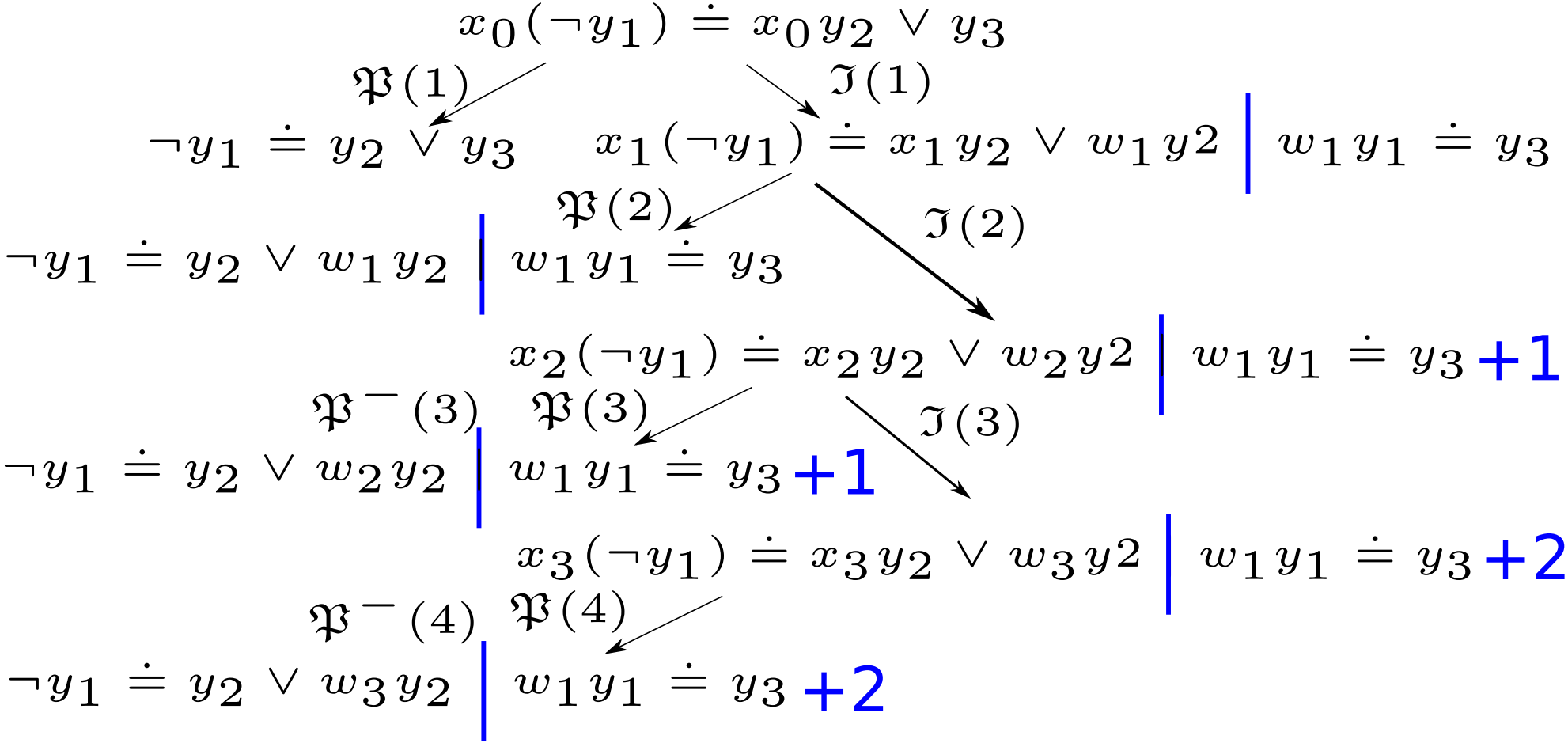
Non-monadic cyclic equations



Lemma 1 ✓

Lemma 2 ?

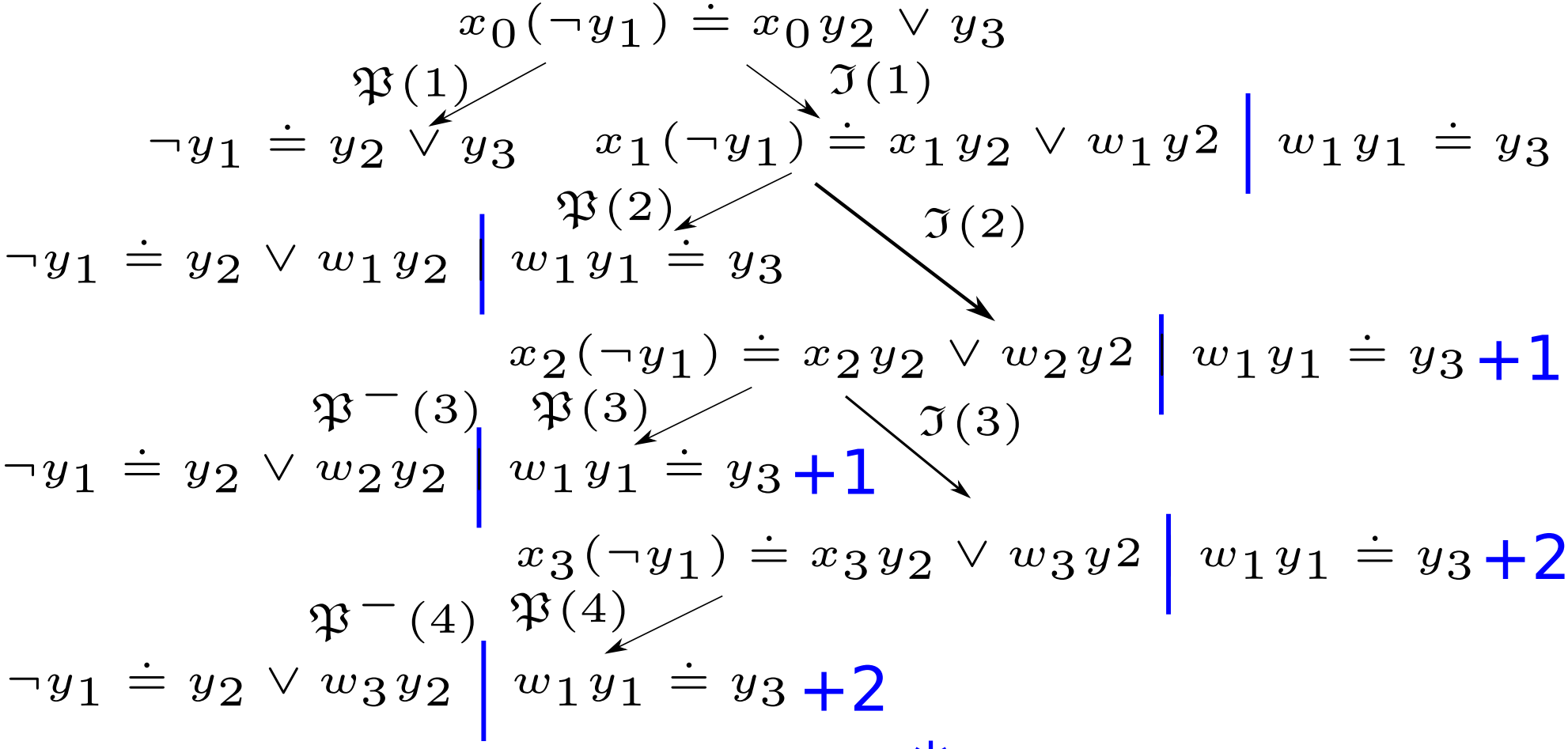
Non-monadic cyclic equations



Lemma 1 ✓

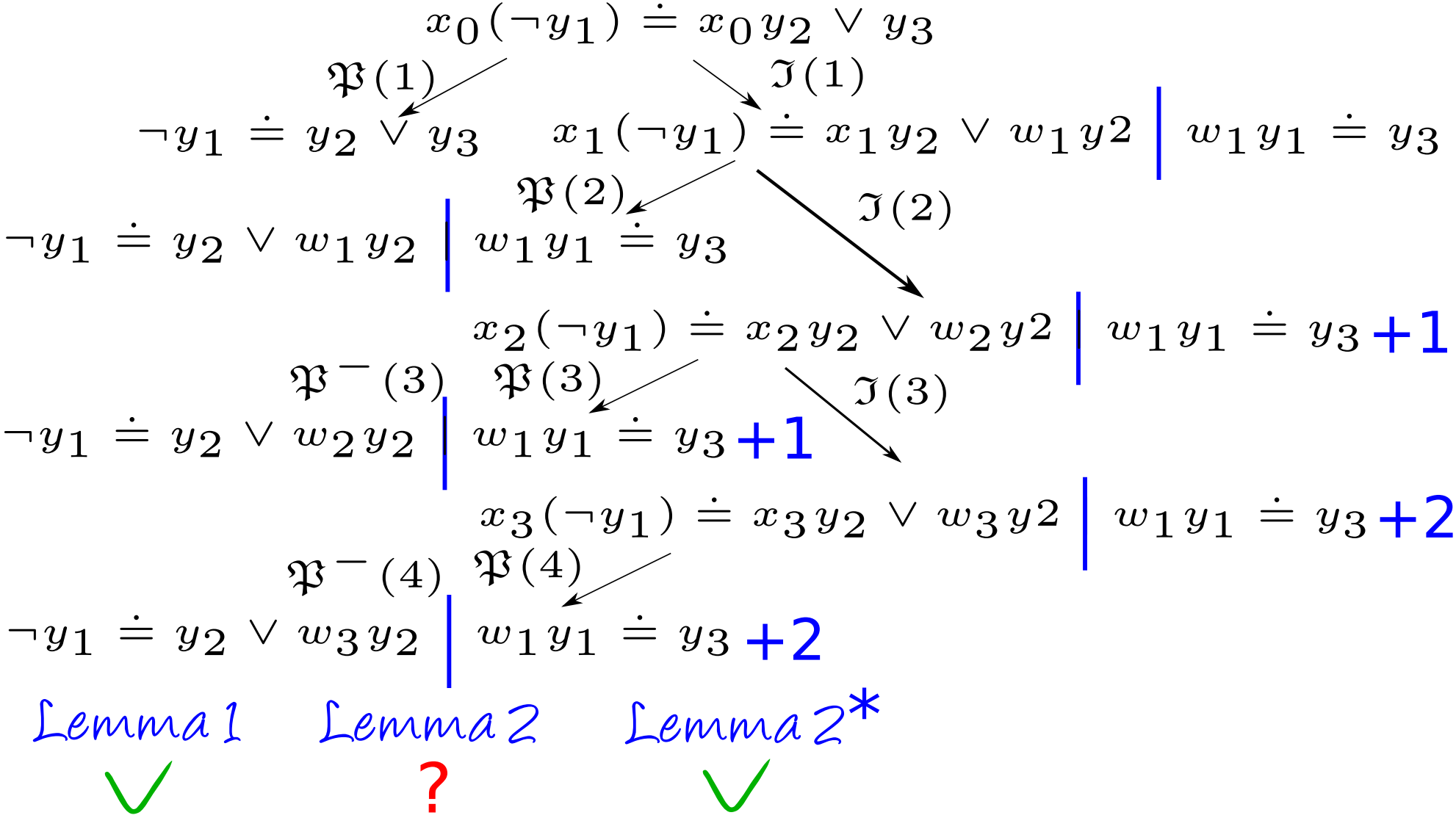
Lemma 2 ?

Non-monadic cyclic equations



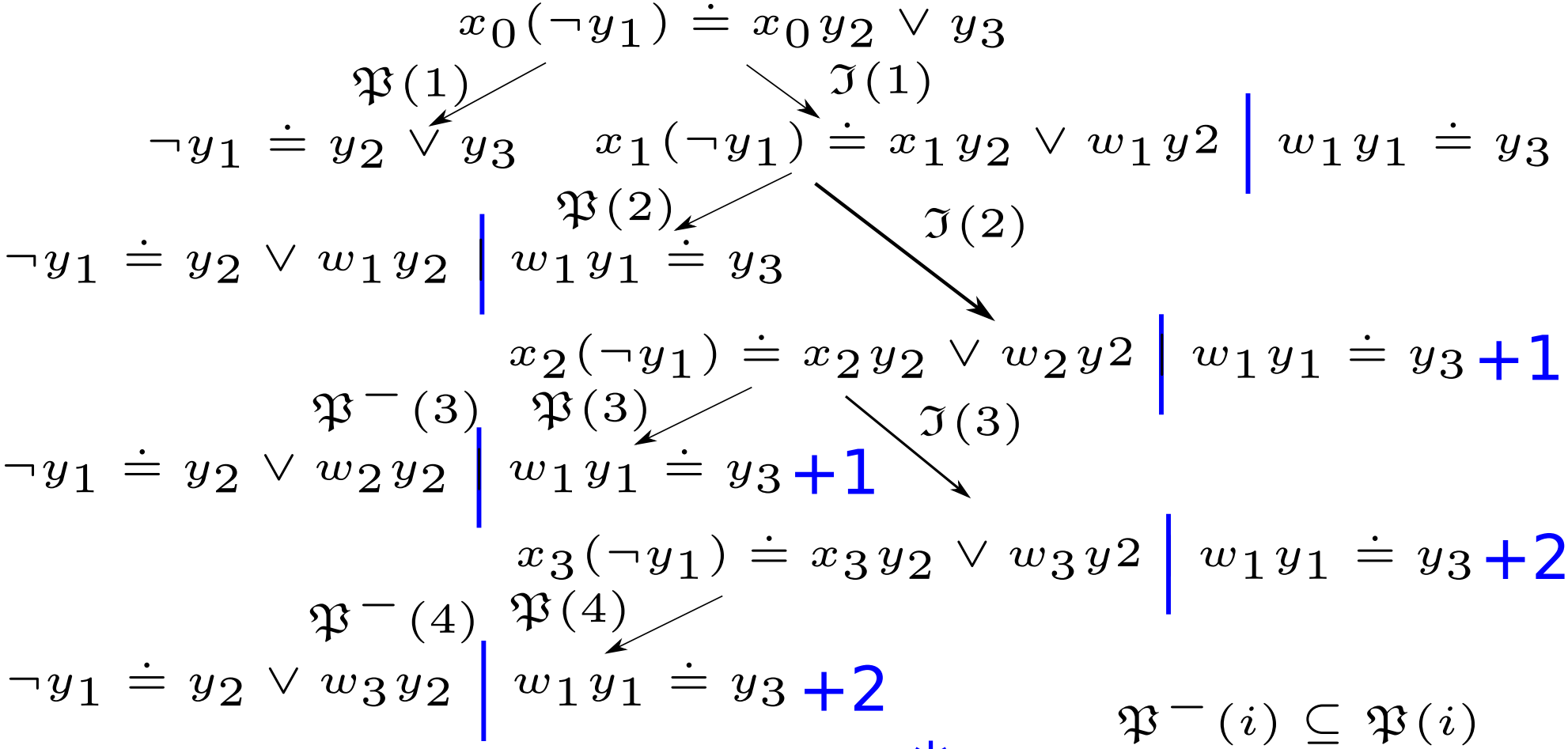
Lemma 1 ✓ Lemma 2 ? Lemma 2*

Non-monadic cyclic equations



Lemma 1 ✓ Lemma 2 ? Lemma 2* ✓

Non-monadic cyclic equations

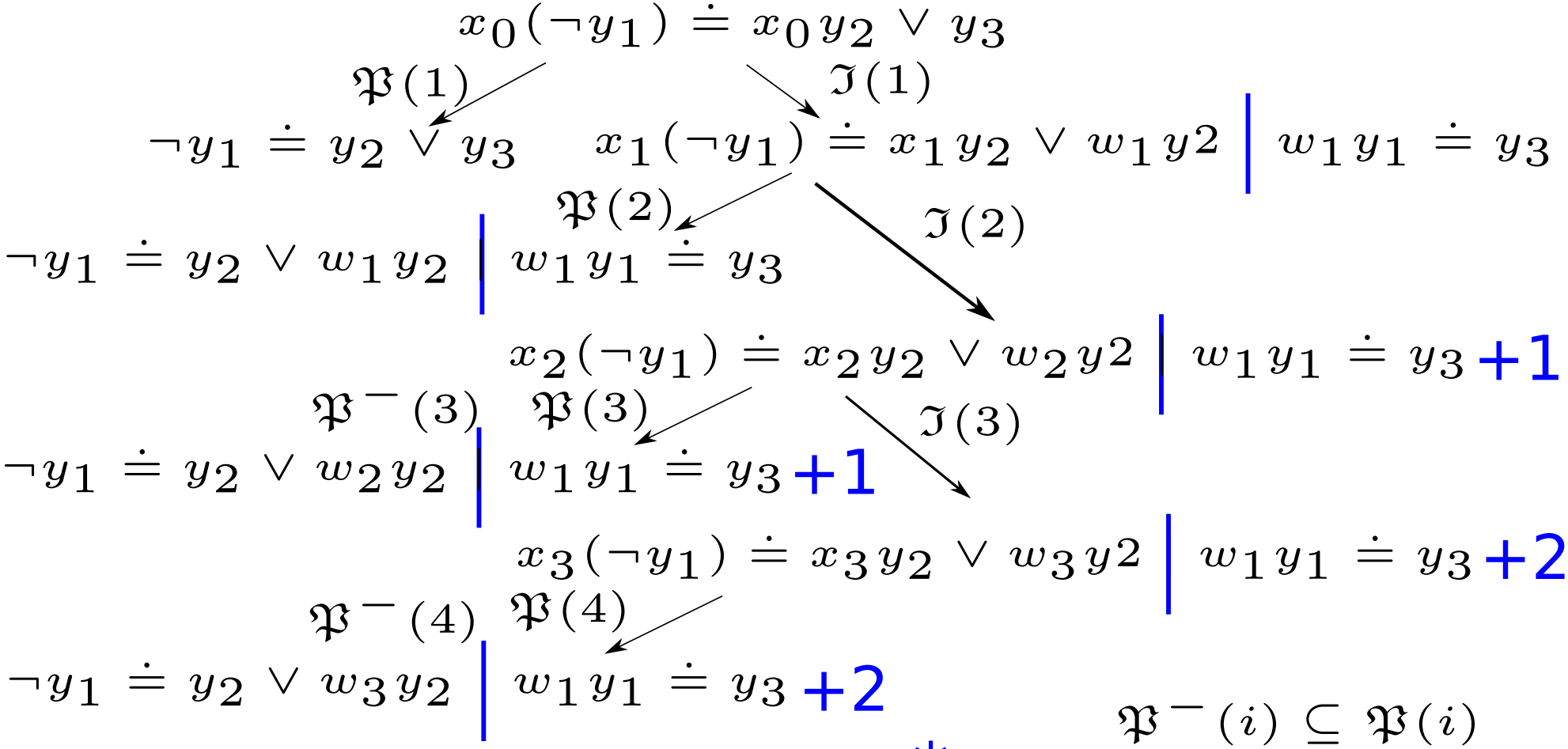


Lemma 1 ✓ Lemma 2 ? Lemma 2* ✓

$$\mathfrak{P}^-(i) \subseteq \mathfrak{P}(i)$$

$$\mathcal{U}(\mathfrak{P}(i)) \subseteq \mathcal{U}(\mathfrak{P}^-(i))$$

Non-monadic cyclic equations

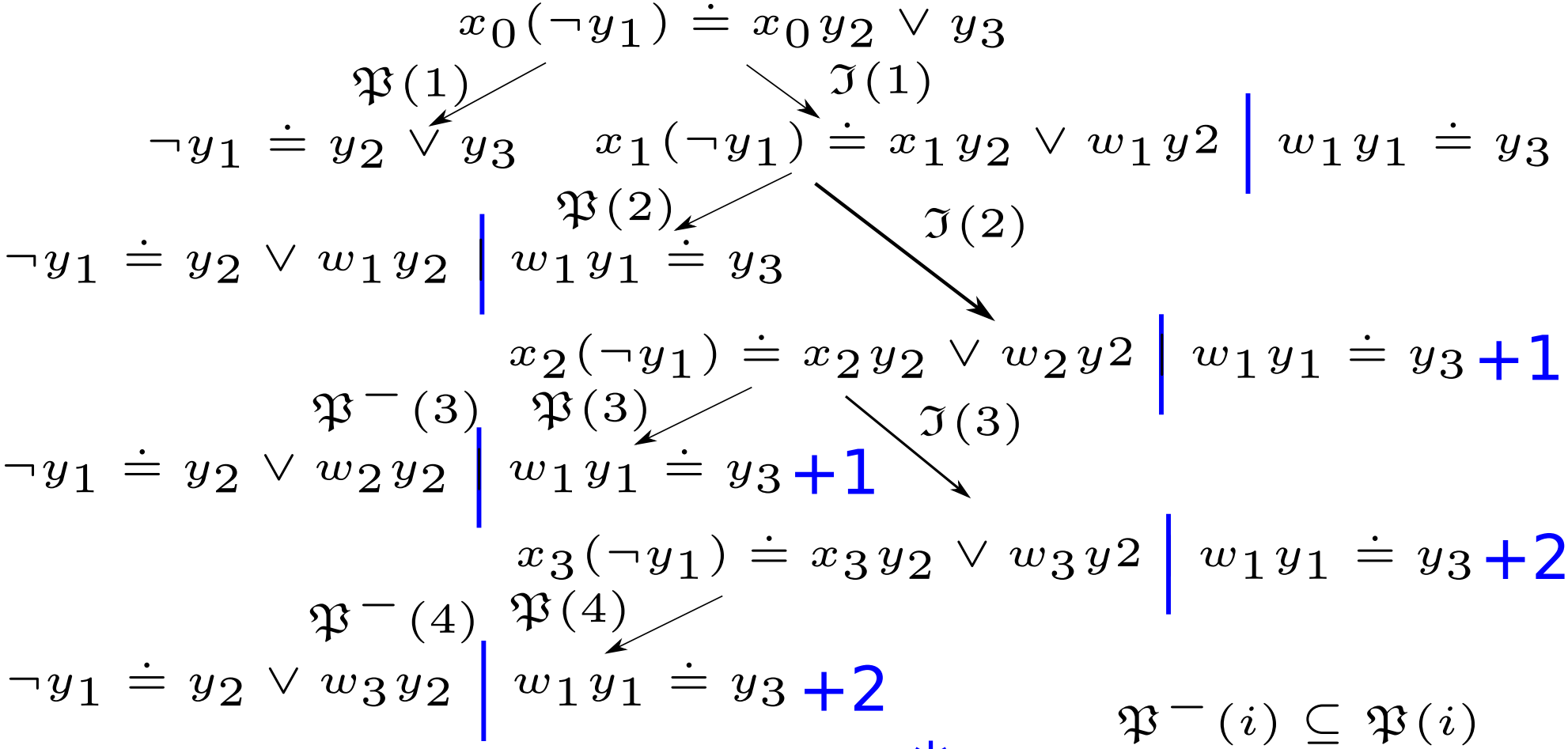


Lemma 1 ✓ Lemma 2 ✗ Lemma 2* ✓

$$\mathfrak{P}^-(i) \subseteq \mathfrak{P}(i)$$

$$\mathcal{U}(\mathfrak{P}(i)) \subseteq \mathcal{U}(\mathfrak{P}^-(i))$$

Non-monadic cyclic equations



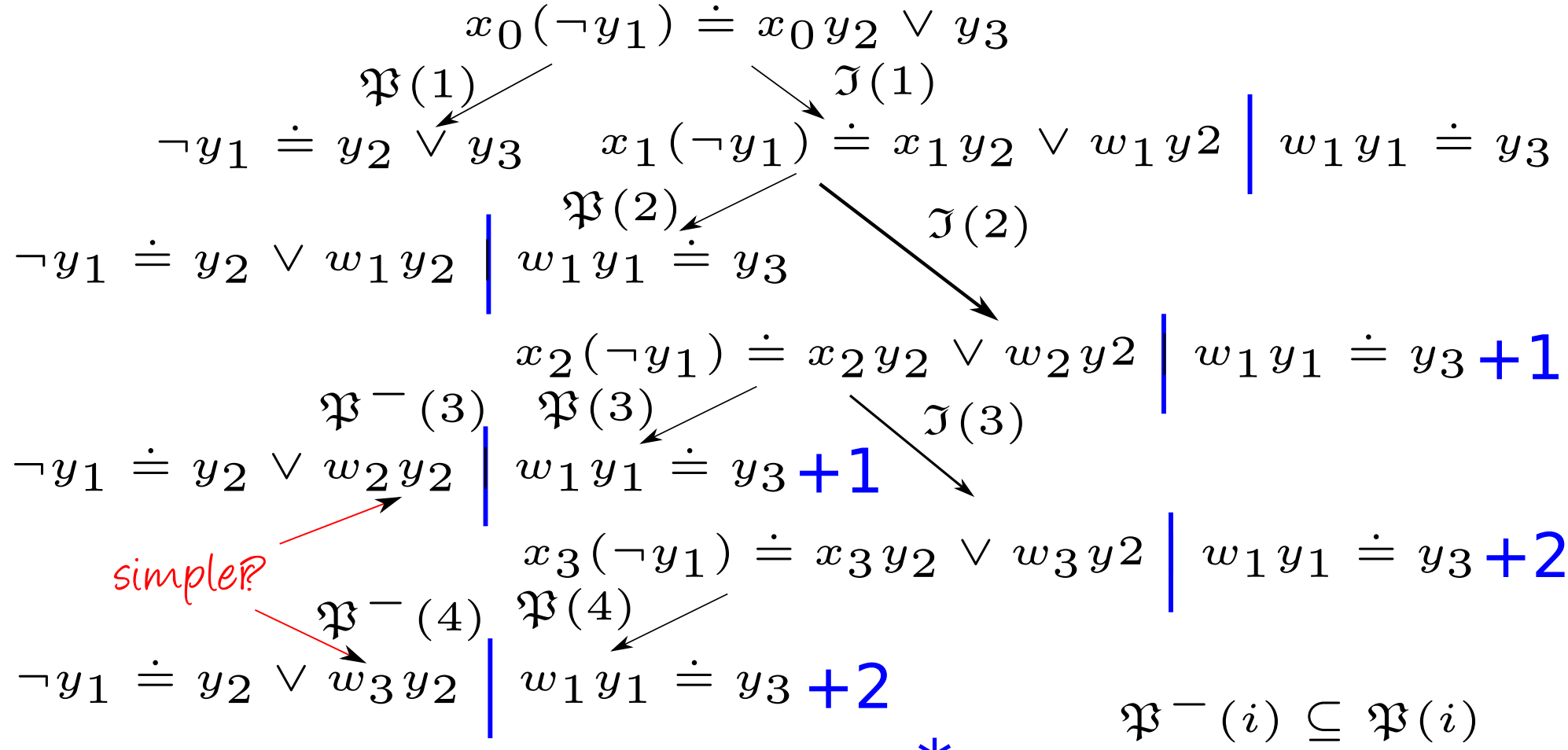
Lemma 1 ✓ Lemma 2 ✗ Lemma 2* ✓

$\mathfrak{P}^-(i) \subseteq \mathfrak{P}(i)$

$\mathcal{U}(\mathfrak{P}(i)) \subseteq \mathcal{U}(\mathfrak{P}^-(i))$

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

Non-monadic cyclic equations



Lemma 1 ✓ Lemma 2 ✗ Lemma 2* ✓

$$\mathfrak{P}^-(i) \subseteq \mathfrak{P}(i)$$

$$\mathcal{U}(\mathfrak{P}(i)) \subseteq \mathcal{U}(\mathfrak{P}^-(i))$$

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

Decidable fragements of Projected Cycles

- $\mathfrak{P}^-$ unifiable?

Decidable fragements of Projected Cycles

- $\mathfrak{P}^{-}$ unifiable?
- Simple such decidable classes

Decidable fragements of Projected Cycles

- $\mathfrak{P}^{-}$ unifiable?
- Simple such decidable classes
- Stronger classes: regular tree automata

Decidable fragements of Projected Cycles

- $\mathfrak{P}^-$ unifiable?
- Simple such decidable classes
- Stronger classes: regular tree automata
- Idea: $\mathfrak{P} \setminus \mathfrak{P}^-$ are freely generated

Decidable fragements of Projected Cycles

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

Pattern unification

- Most usefull subclass: higher-order unitary unification.

Pattern unification

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

Pattern unification

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

Pattern unification

- Most usefull 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 unification

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

Pattern unification

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

Pattern unification

- Most usefull subclass: higher-order unitary unification.
- Applications:
  - Proof assistants and Logical frameworks
  - λProlog
  - ...
- variables are applied to a distinct list of bound variables:
- Pattern: $\lambda z_1 z_2.x z_1 \doteq f y z_1 z_2$
- Non-pattern: $\lambda z_1 z_2.x z_1 z_1 \doteq a$
- Idea: Determinism between `(Project)` and `(Imitate)`

Pattern unification

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

Extending Pattern unification

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

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)
    - $\lambda z.x(fz) \doteq t$
  - λProlog

Extending Pattern unification

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

Extending Pattern unification

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

Extending Pattern unification

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

Conclusion

- Active research:
  - Define tree automata class

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, λProlog, 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