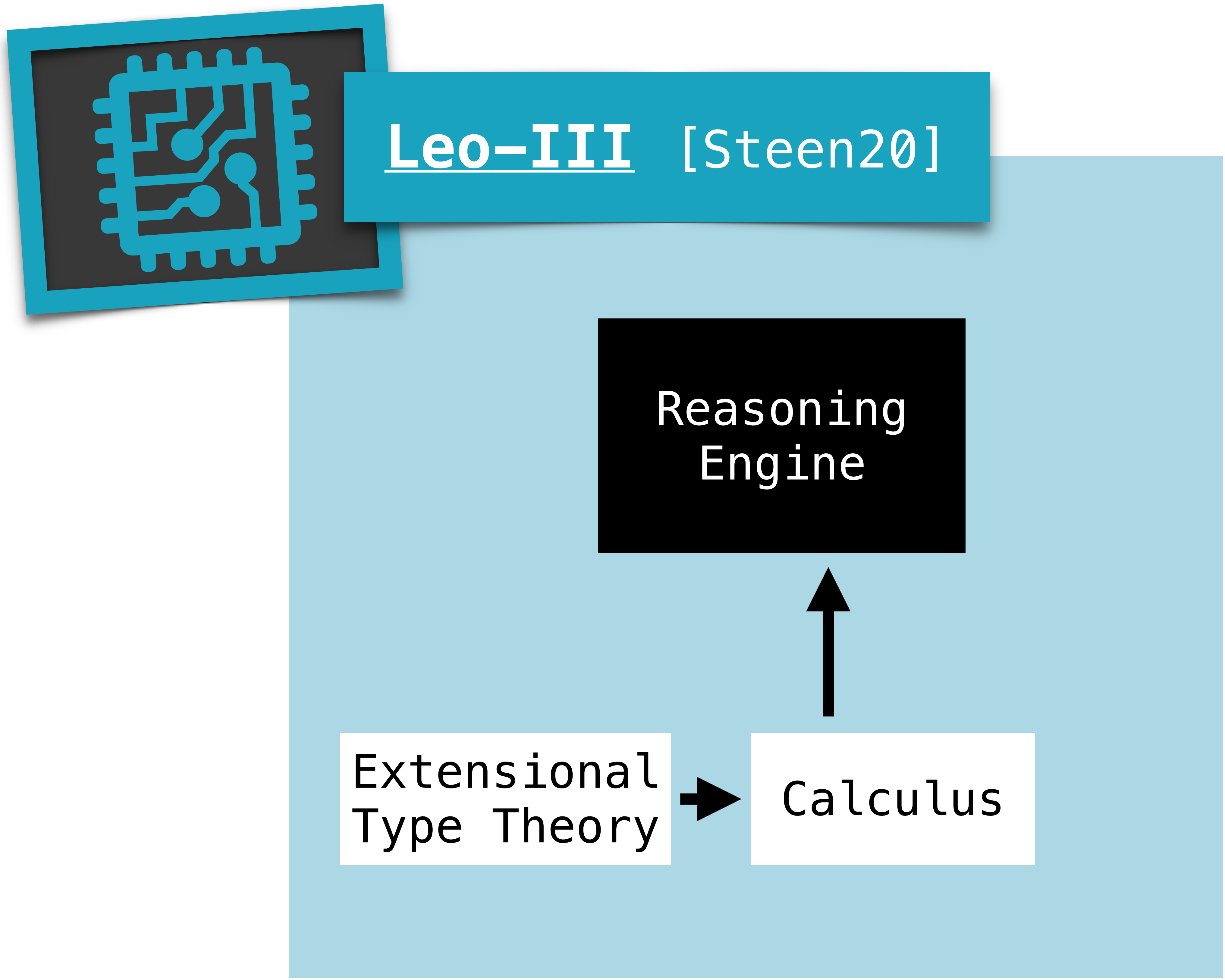
Encoding and Verifying Leo-III Proofs in the Dedukti Framework



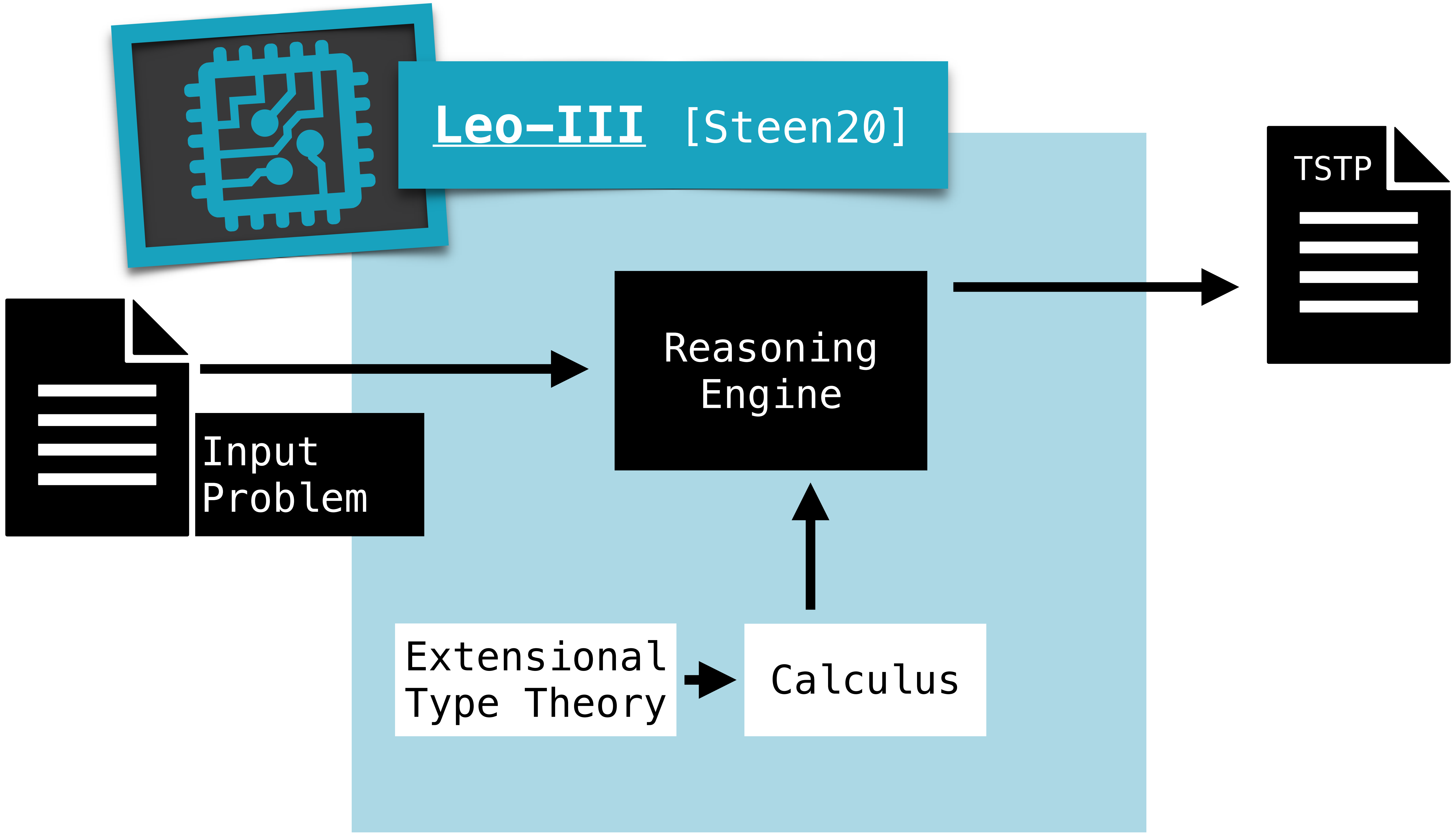
UNIVERSITÄT GREIFSWALD
Wissen lockt. Seit 1456

Melanie Taprogge

Supervised by:
Frédéric Blanqui
Alexander Steen

école normale supérieure paris–saclay

Encoding and Verifying Leo-III Proofs in the Dedukti Framework



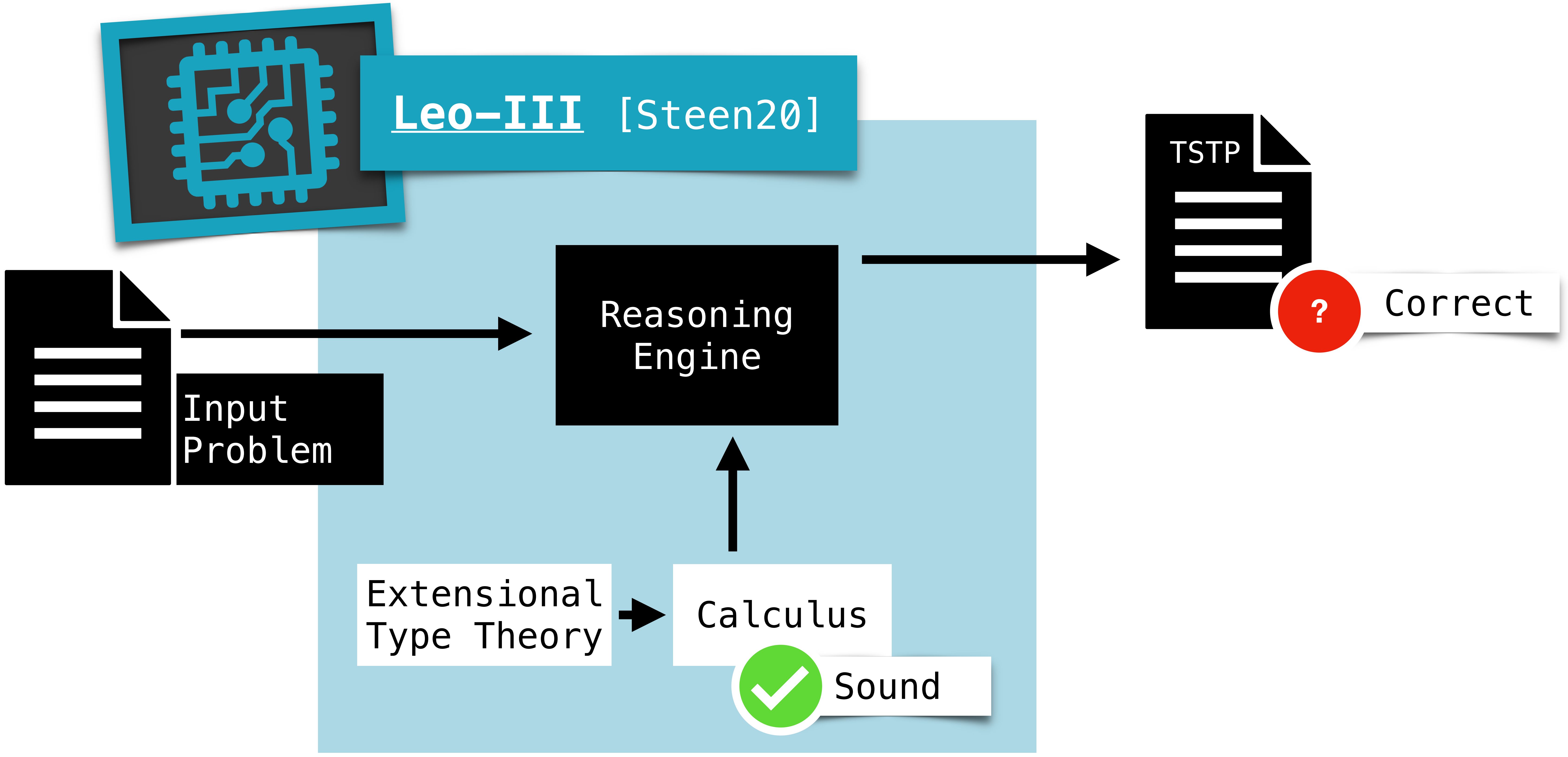
UNIVERSITÄT GREIFSWALD
Wissen lockt. Seit 1456

Melanie Taprogge

Supervised by:
Frédéric Blanqui
Alexander Steen

école normale supérieure paris–saclay

Encoding and Verifying Leo-III Proofs in the Dedukti Framework



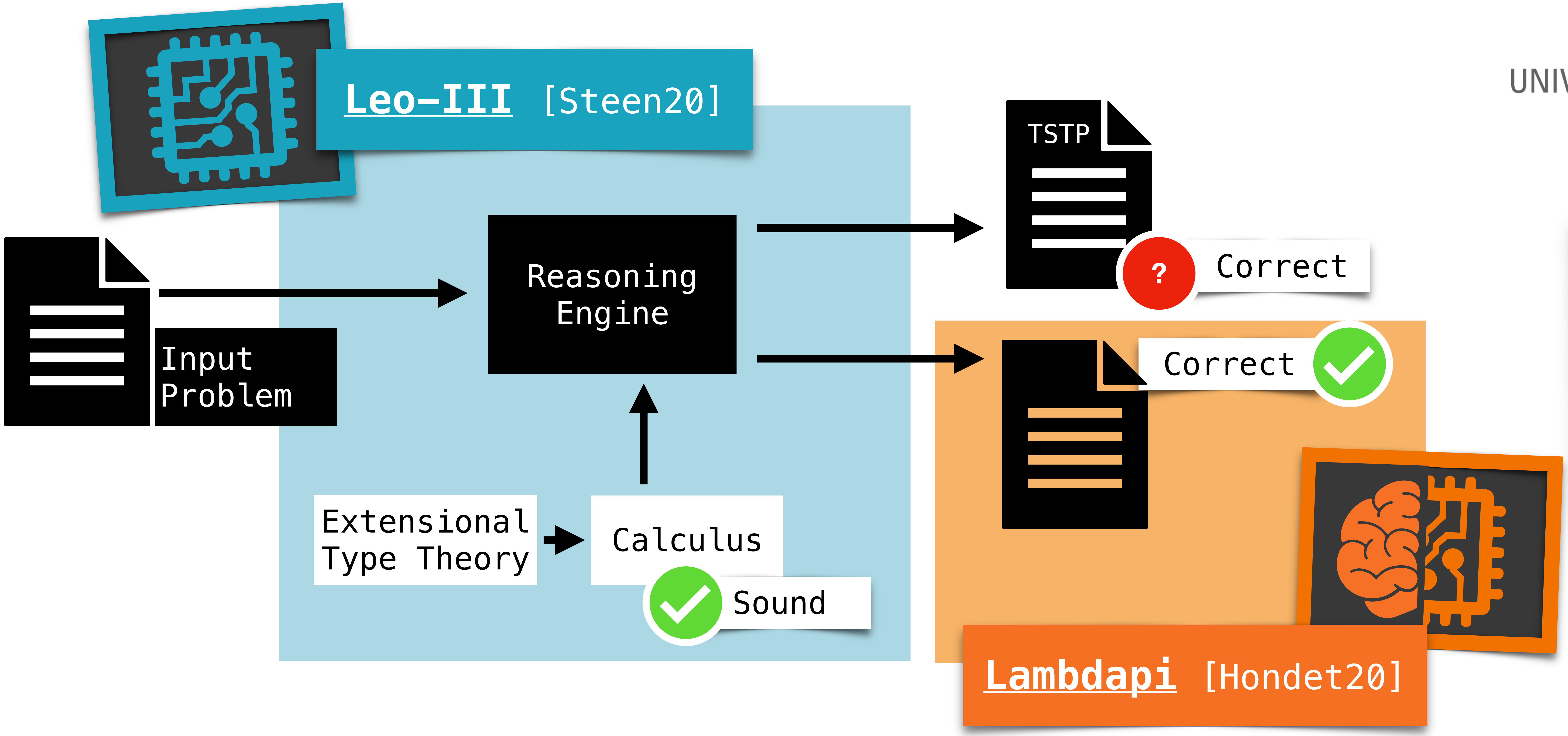
UNIVERSITÄT GREIFSWALD
Wissen lockt. Seit 1456

Melanie Taprogge

Supervised by:
Frédéric Blanqui
Alexander Steen

école normale supérieure paris-saclay

Encoding and Verifying Leo-III Proofs in the Dedukti Framework



UNIVERSITÄT GREIFSWALD
Wissen lockt. Seit 1456

Melanie Taprogge

Supervised by:
Frédéric Blanqui
Alexander Steen

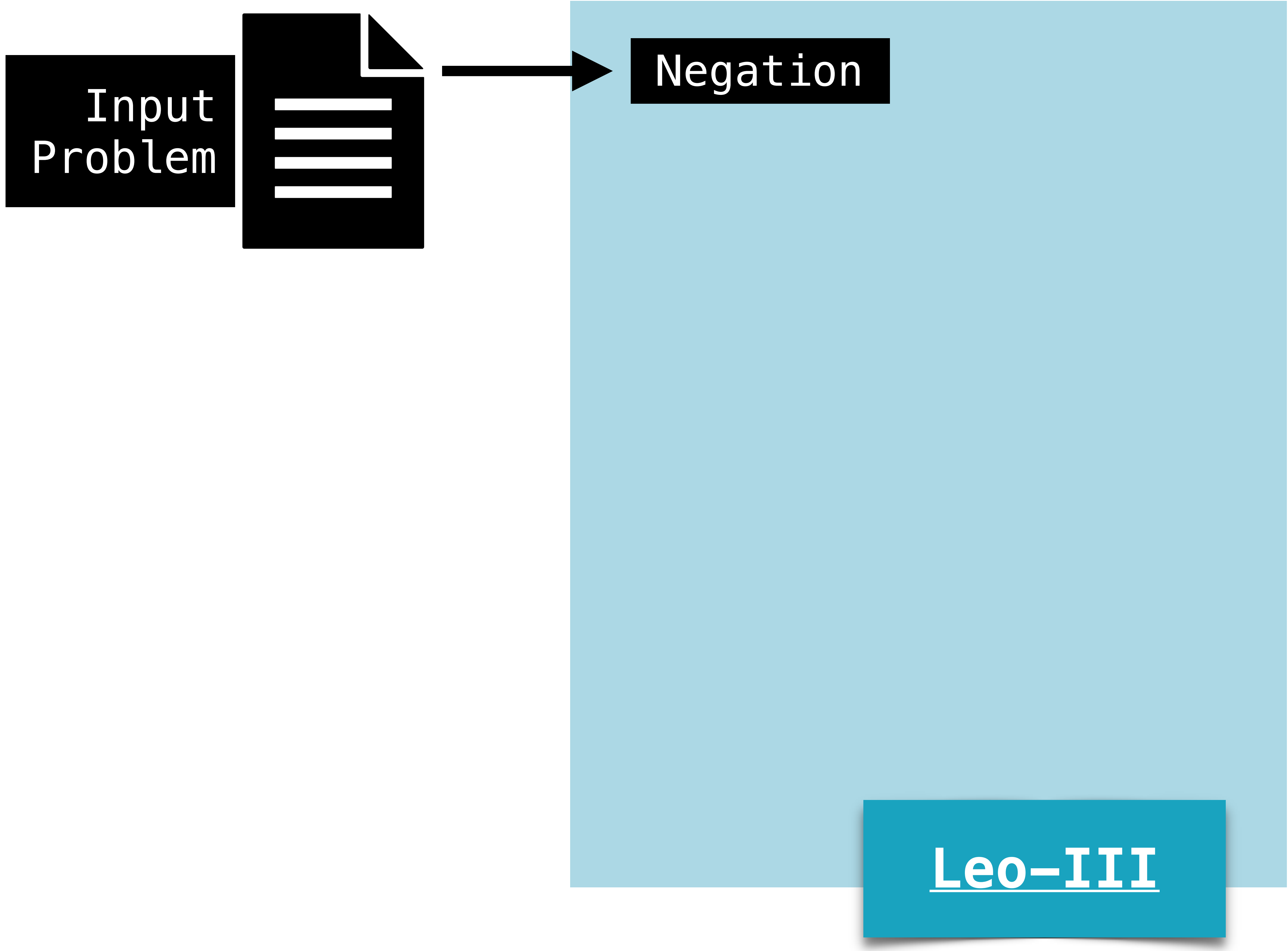
école normale supérieure paris-saclay

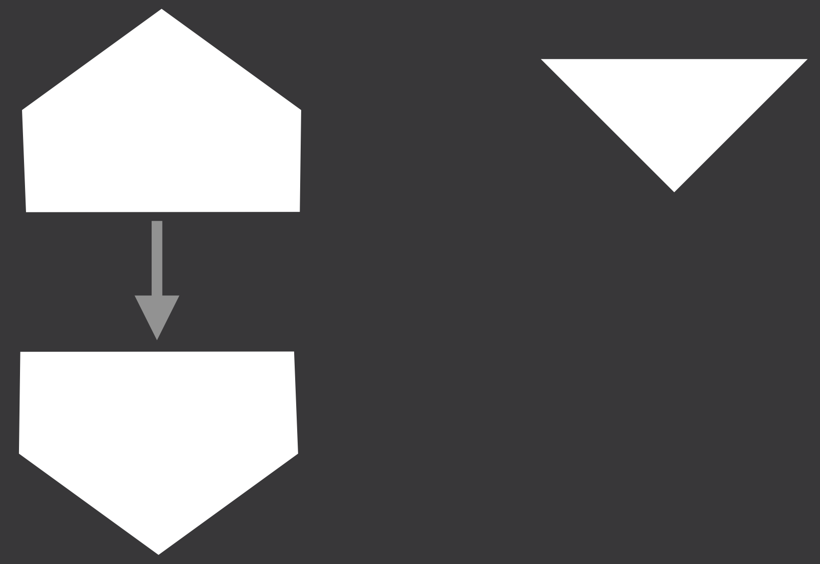
ATP Workflow

Found Proof

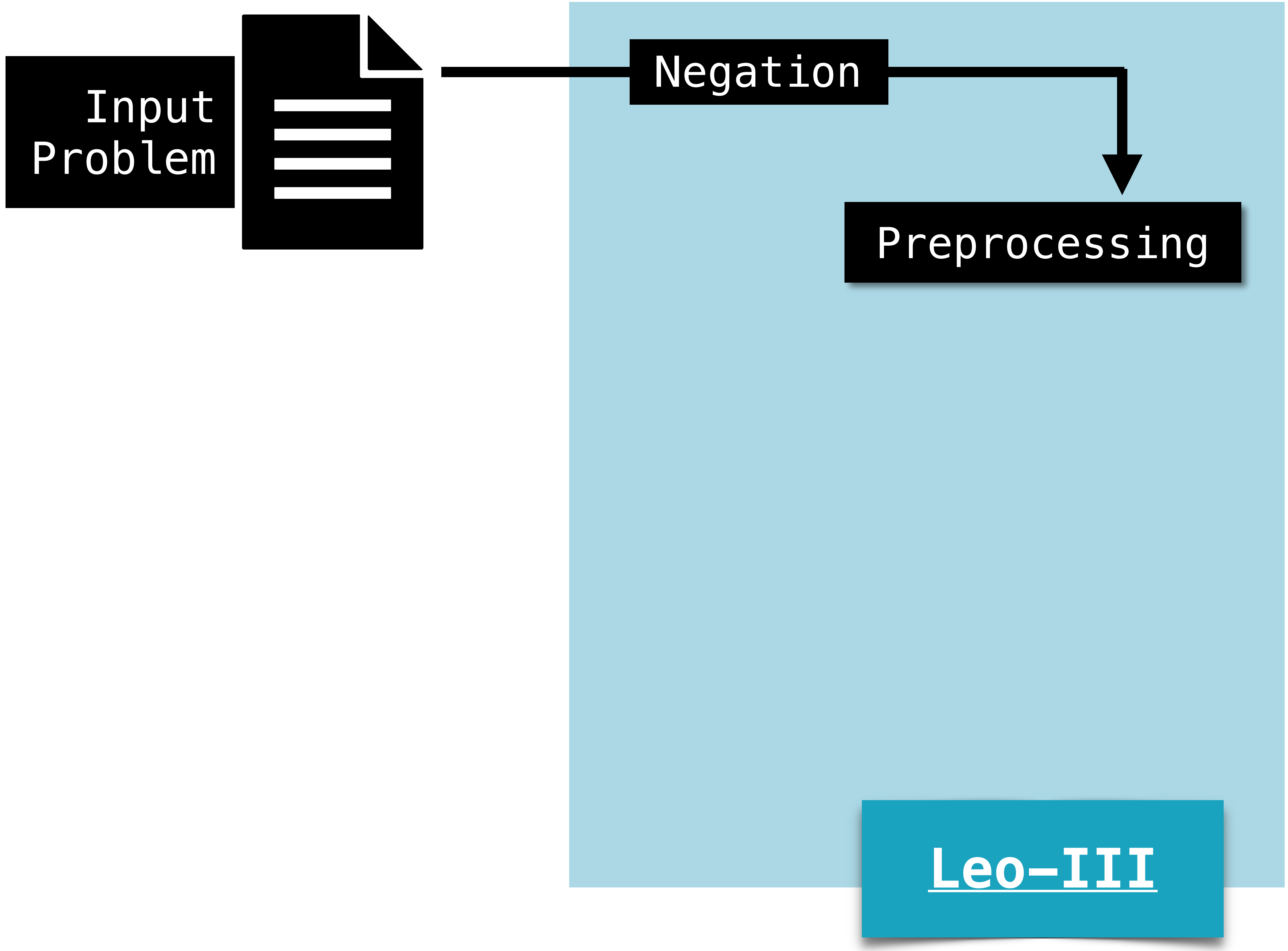
ATP Workflow

Found Proof

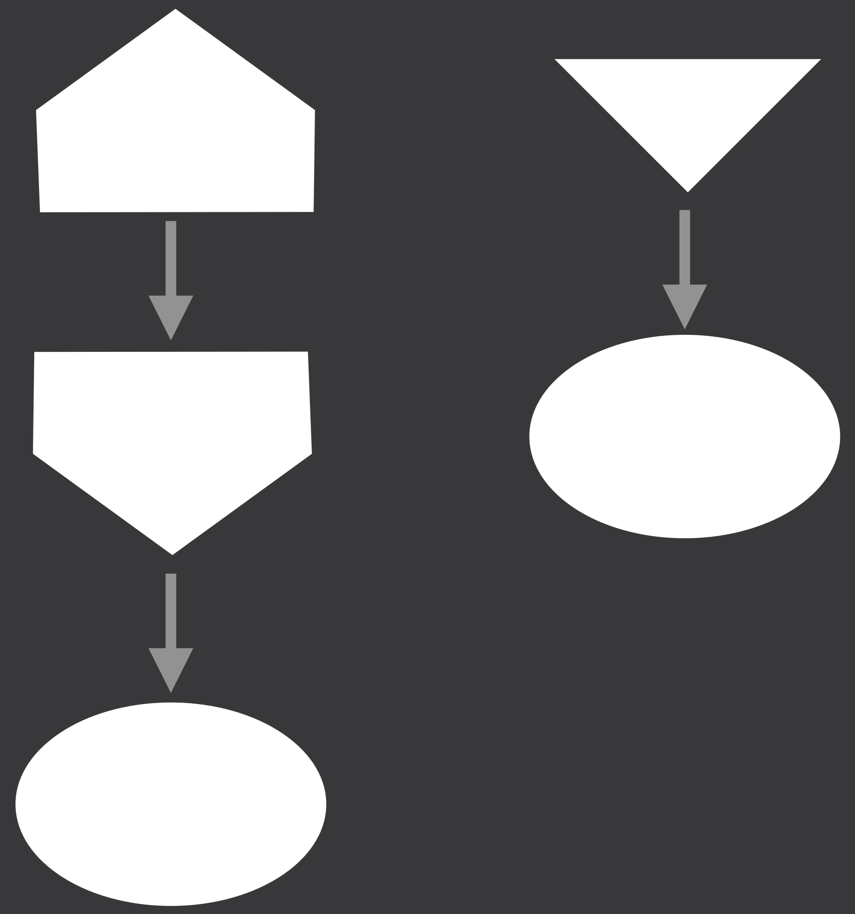




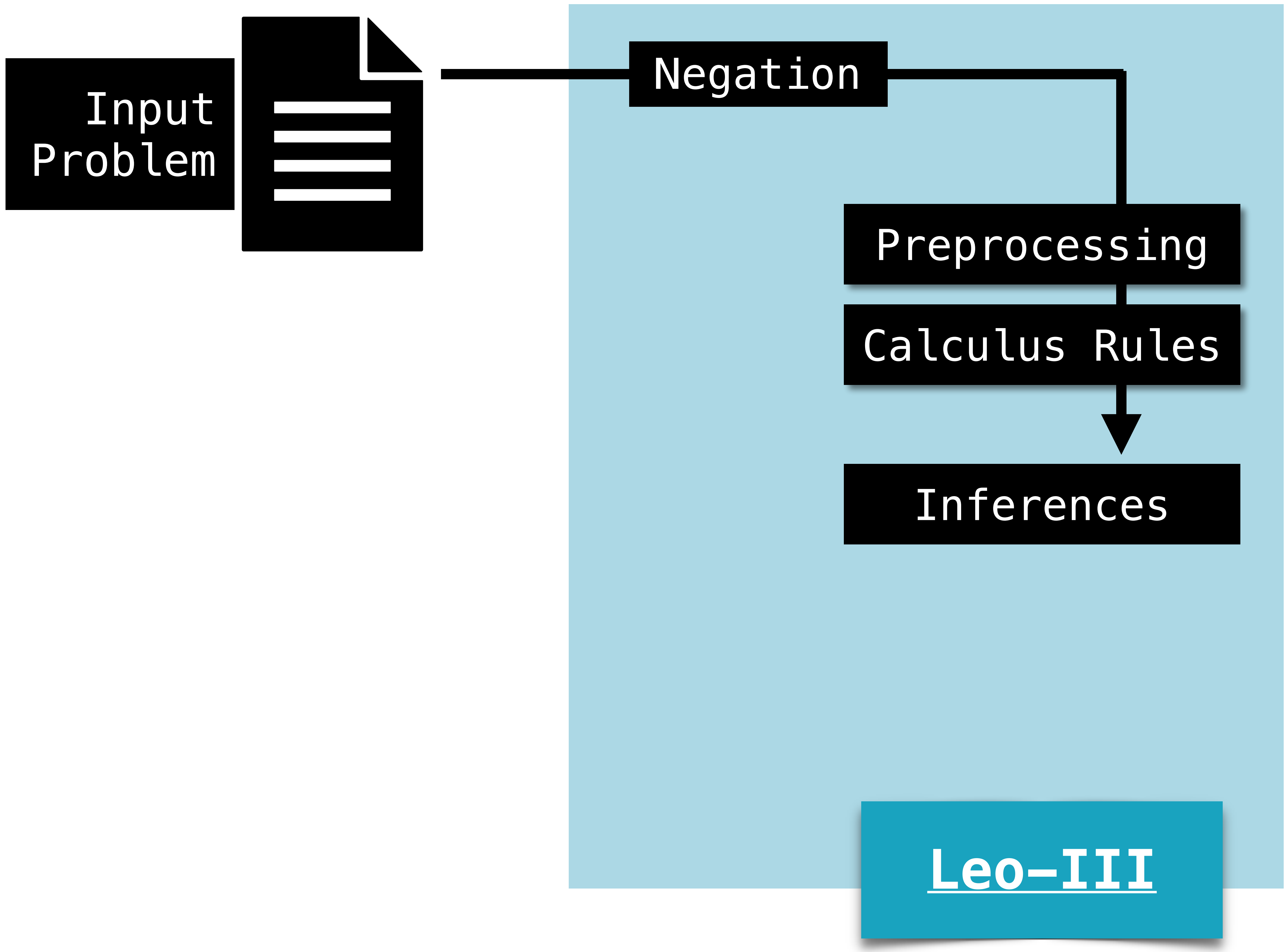
ATP Workflow



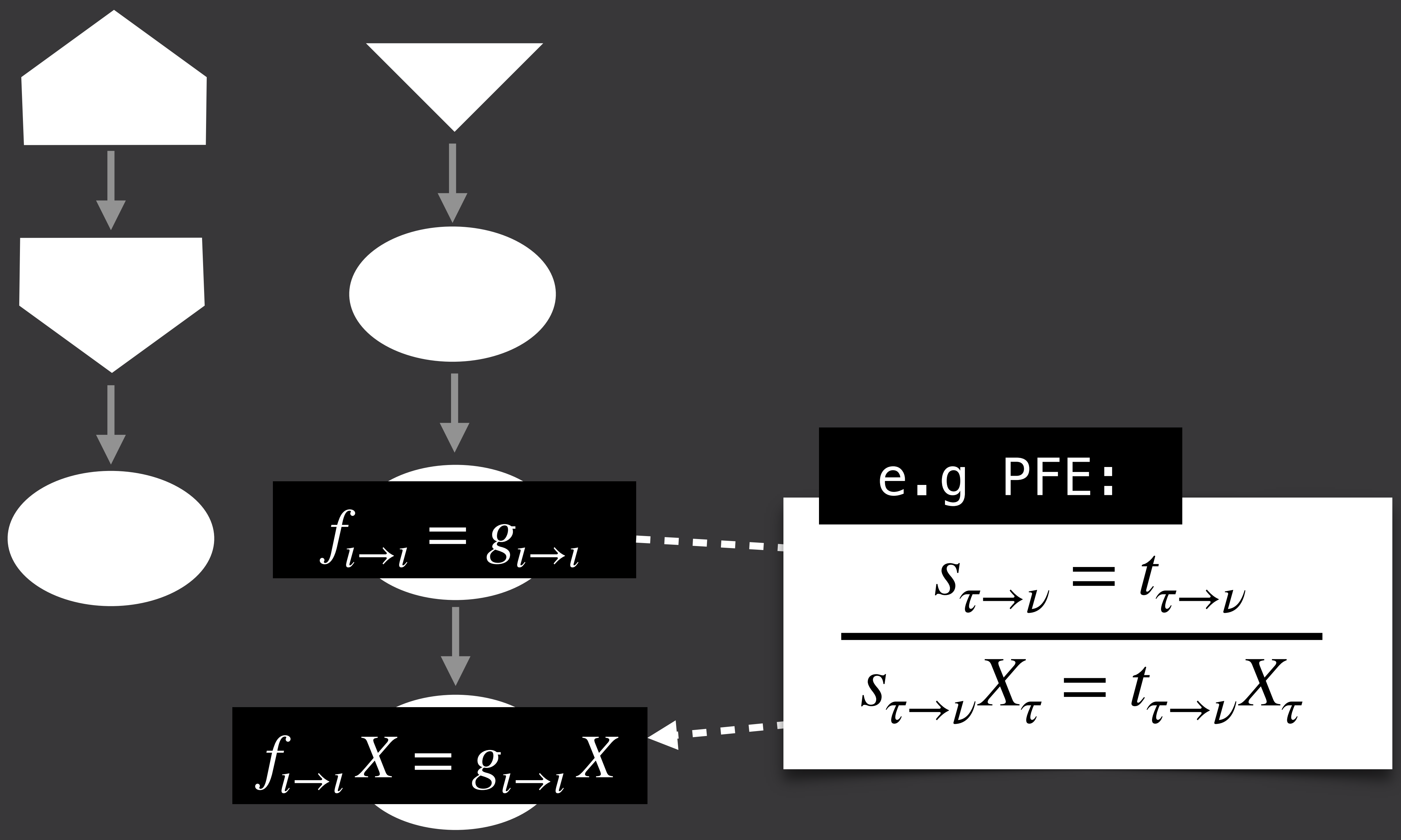
Found Proof



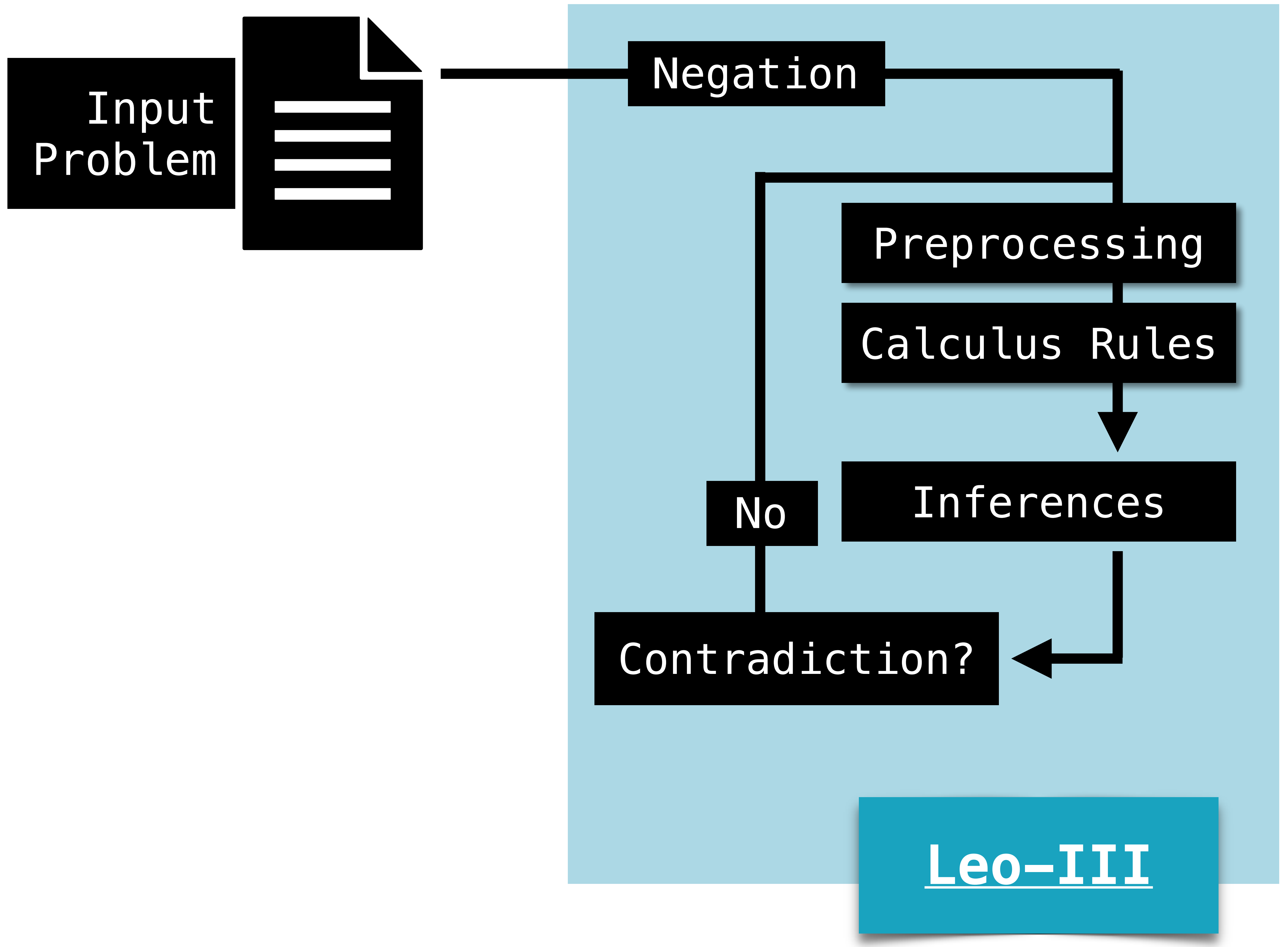
ATP Workflow



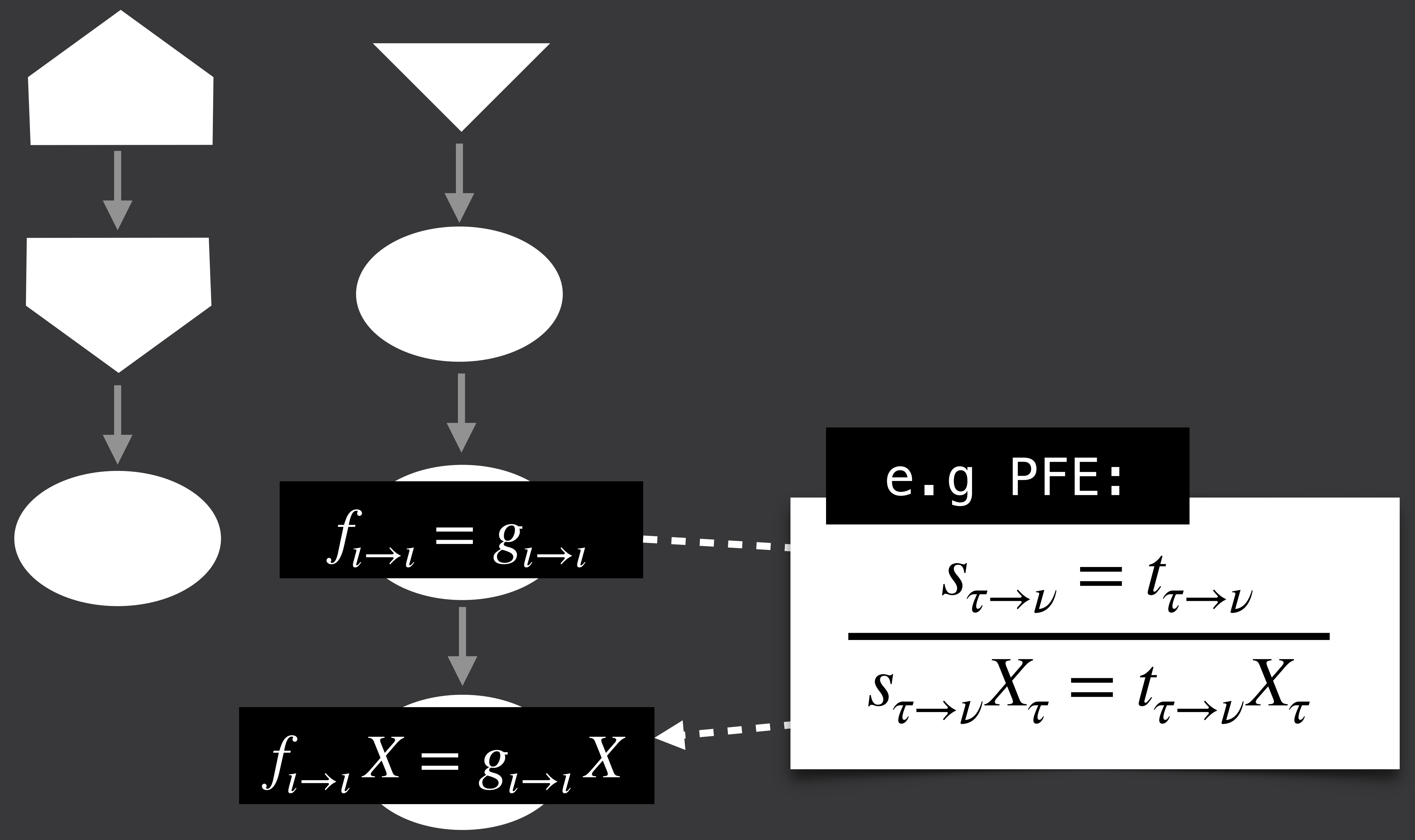
Found Proof



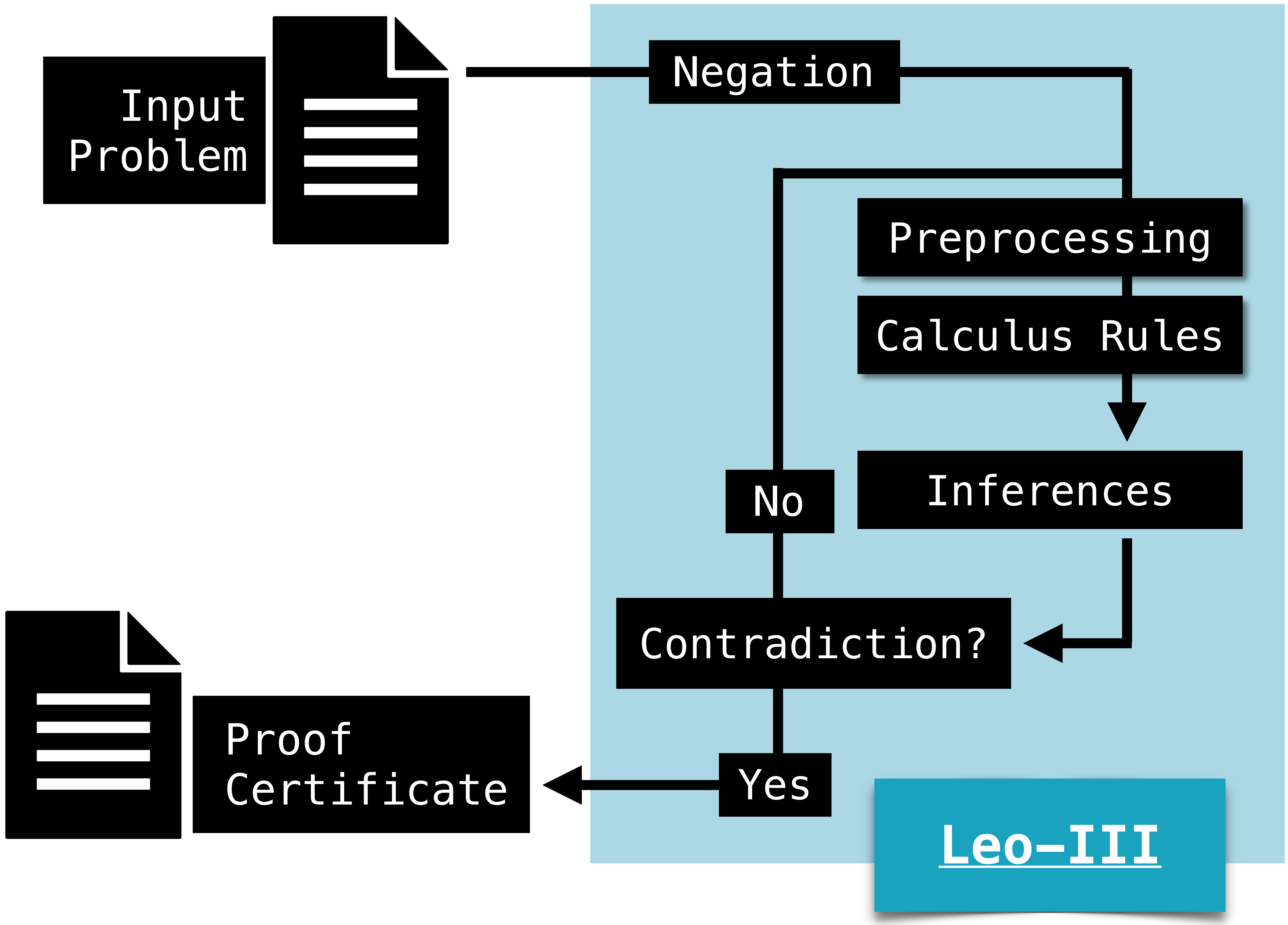
ATP Workflow



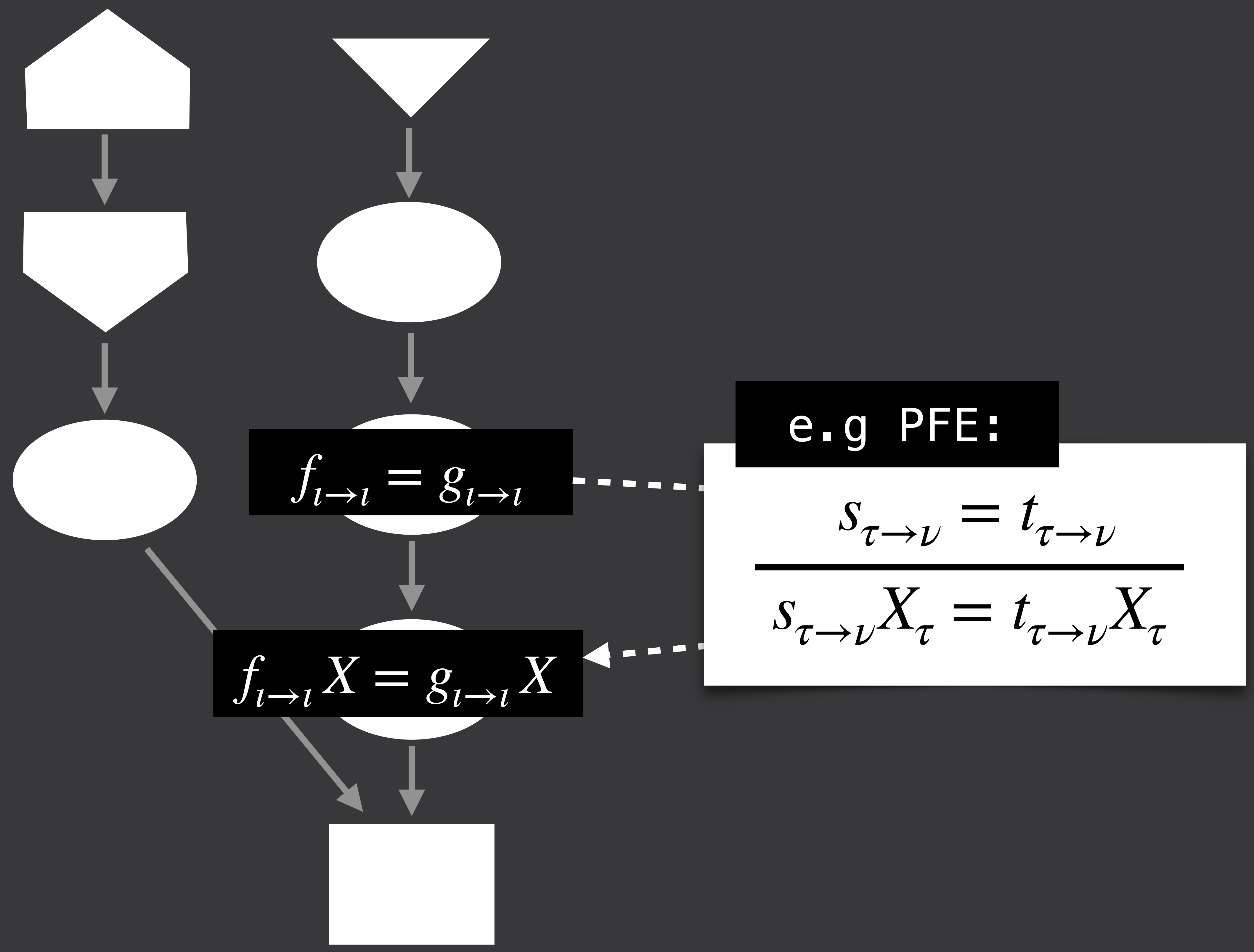
Found Proof



ATP Workflow

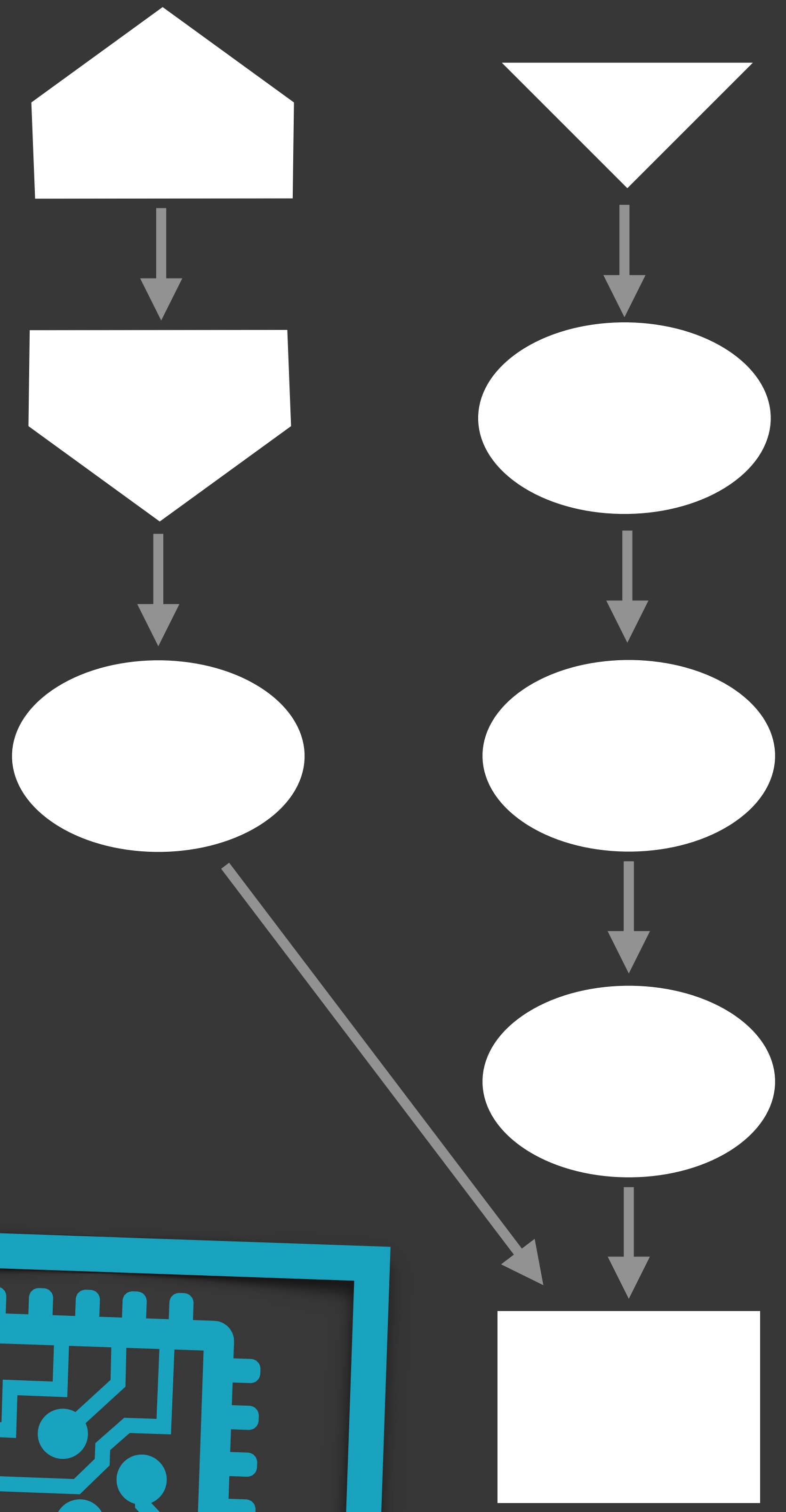


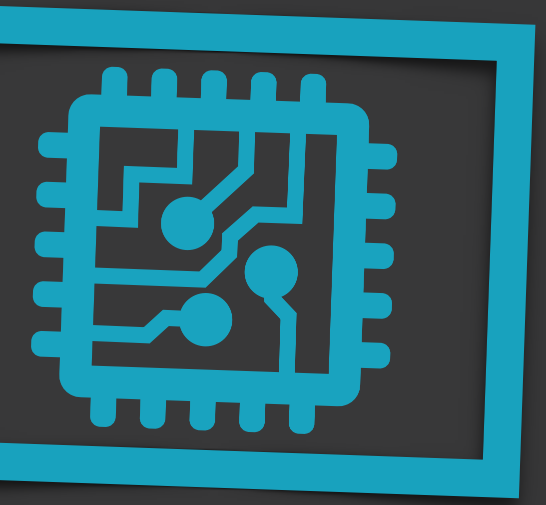
Found Proof



Found Proof

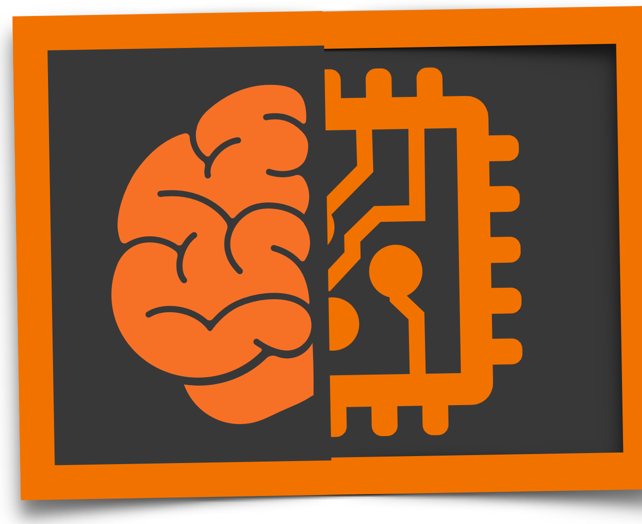
Leo-III





Encoded Proof

Lambdapi

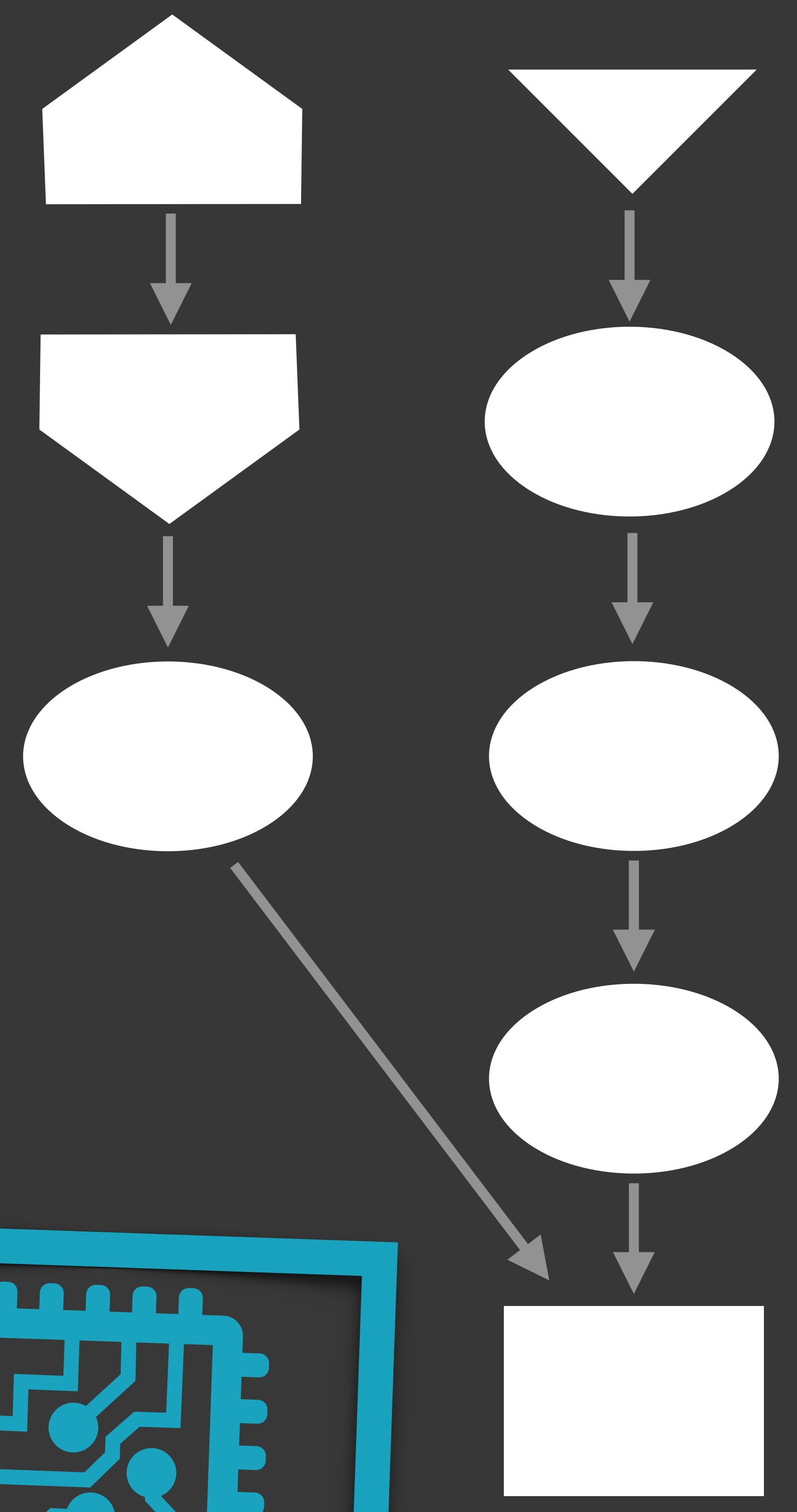


Found Proof

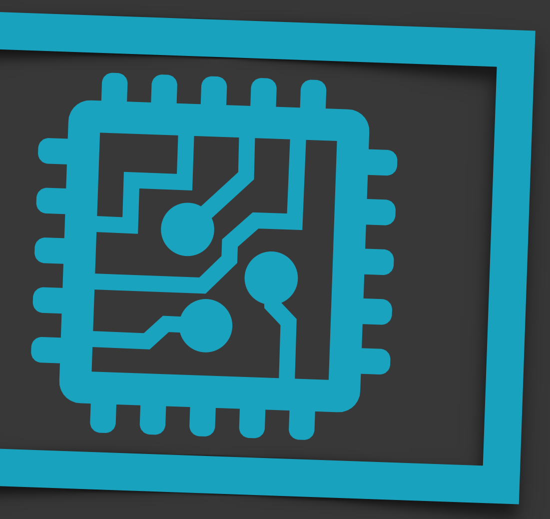
Encoded Proof

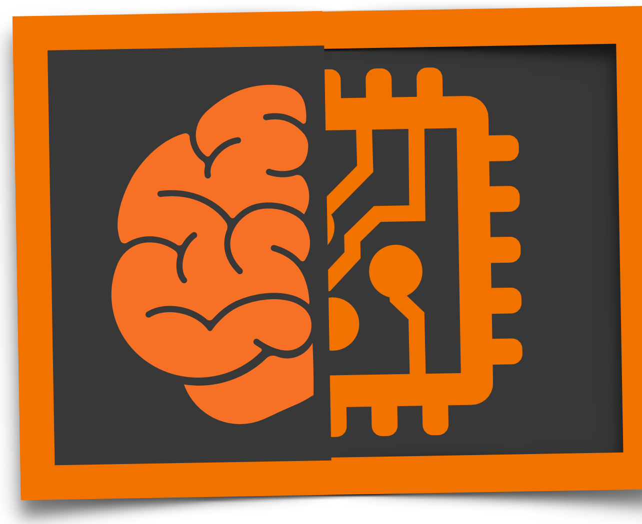
Leo-III

Lambdapi



1 Definition of a **Lambdapi Theory**



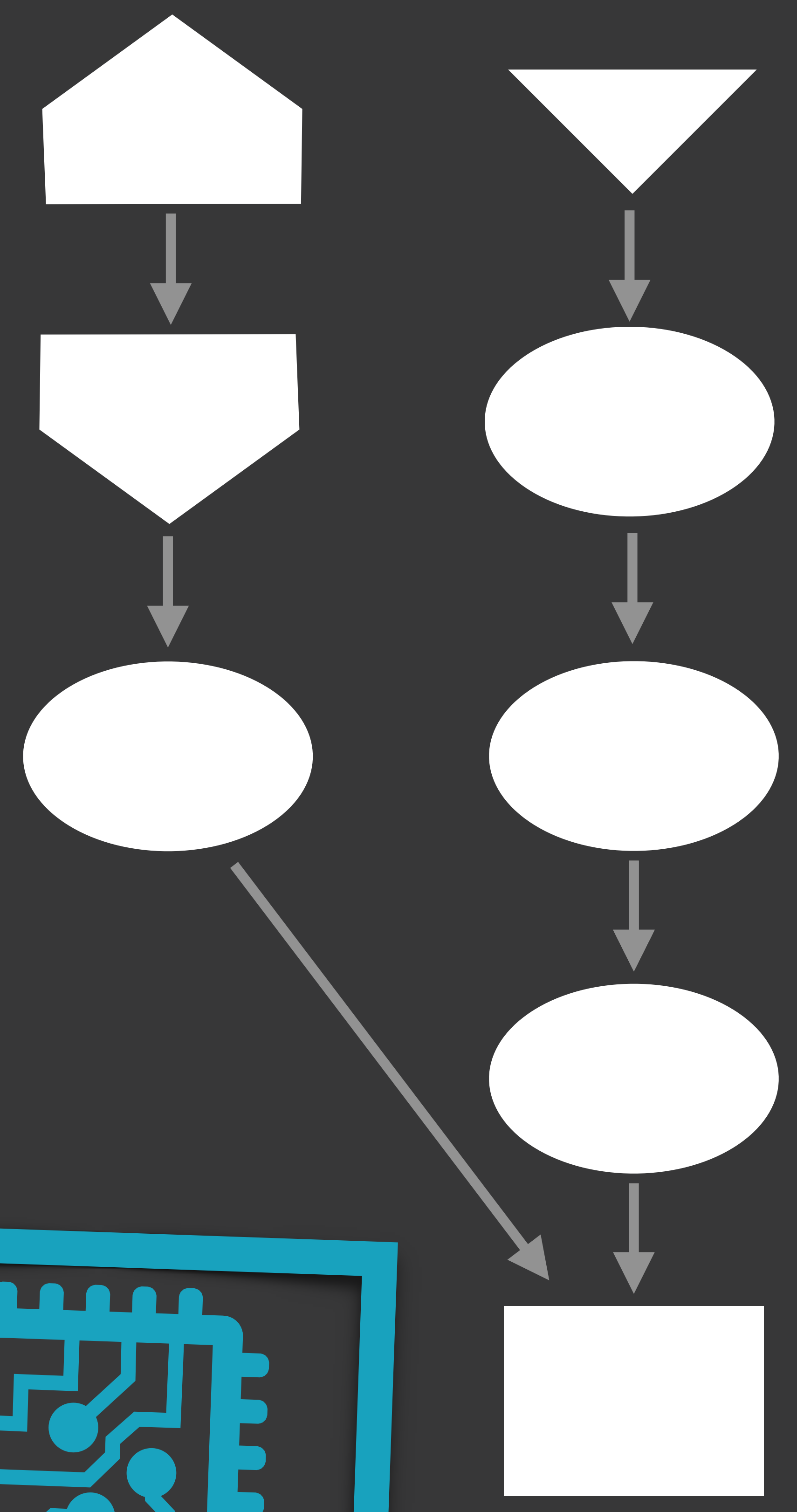


Found Proof

Encoded Proof

Leo-III

Lambdapi

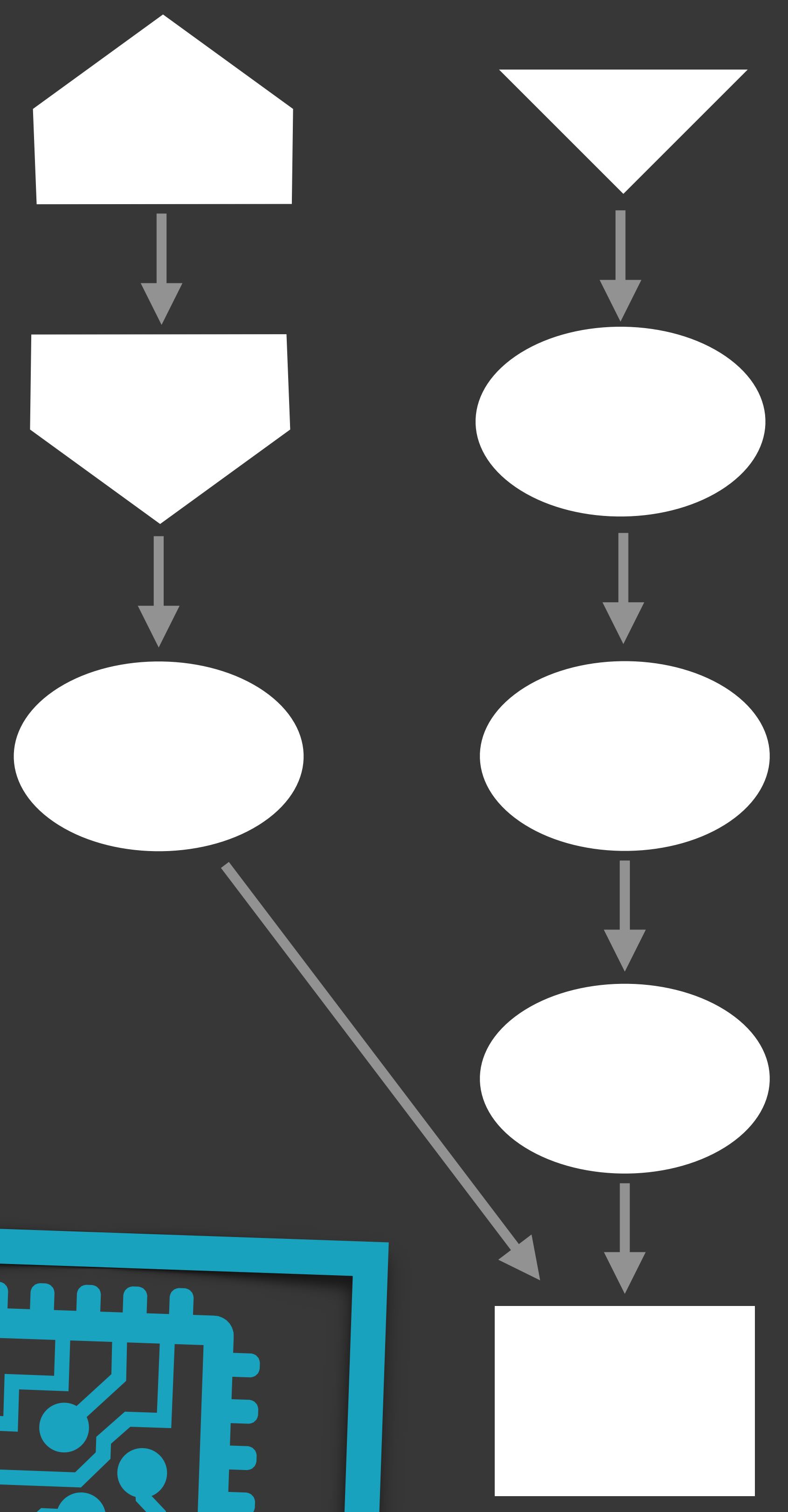


1 Definition of a **Lambdapi Theory**

2 Encoding of **Problems and Proof** Steps

Found Proof

Leo-III



1 Definition of a **Lambdapi Theory**

2 Encoding of **Problems and Proof** Steps

3 Encoding of the **Calculus Rules**

Encoded Proof

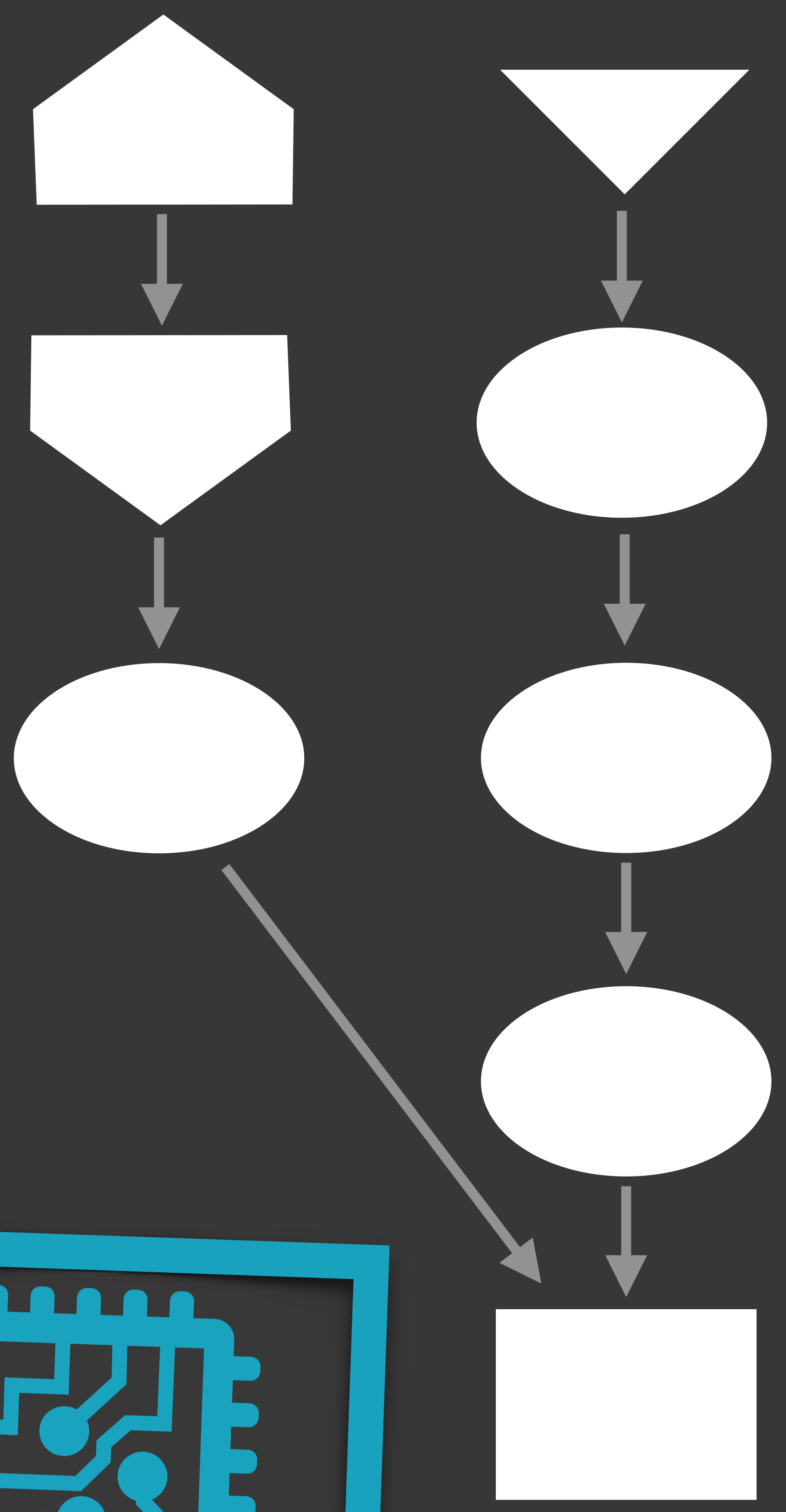
Lambdapi

Found Proof

Encoded Proof

Leo-III

Lambdapi



1 Definition of a **Lambdapi Theory**

2 Encoding of **Problems and Proof** Steps

3 Encoding of the **Calculus Rules**

4 Verification of **generated Proofs**

$\lambda\Pi$**-calculus modulo**

-> HOL + $\boxed{\text{dependant types}}$

\+ $\boxed{\text{rewriting}}$

Written $\Pi x : T . S$,
e.g. $\Pi x : Nat . array(x)$

Written $l \hookrightarrow r$, replace occurrences of l with r

\+ Meta-logical type of types (called $TYPE$) e.g: $array_{Nat \to TYPE}$

ExTT

1

$\lambda\Pi$-calculus modulo

Leo-III

Lambdapi

$\lambda\Pi$-calculus modulo

-> HOL + dependant types

+ rewriting

Written $\Pi x : T . S$,
e.g. $\Pi x : Nat . array(x)$

Written $l \hookrightarrow r$, replace occurrences of l with r

+ Meta-logical type of types (called $TYPE$) e.g: $array_{Nat \to TYPE}$

➡ **Encode and check proofs following the propositions as types principle**

Proofs are encoded as terms, their types are the propositions they proof and logical connectives are identified with type constructors [Brouwer75,Curry34,Heyting30,Howard80,Kolmogoroff32]

ExTT

Leo-III

Extensional Type Theory (ExTT)

-> HOL + extensionality
+ choice
+ Rank-1 Polymorphism

Underlying Logic

1

$\lambda\Pi$-calculus modulo

Lambdapi

$\lambda\Pi$-calculus modulo

-> HOL + $\boxed{\text{dependant types}}$
+ $\boxed{\text{rewriting}}$

Written $\Pi x : T \,.\, S$,
e.g. $\Pi x : Nat \,.\, array(x)$

Written $l \hookrightarrow r$, replace occurrences of l with r

+ Meta-logical type of types (called $TYPE$) e.g: $array_{Nat \rightarrow TYPE}$

➡ **Encode and check proofs following the propositions as types principle**

Proofs are encoded as terms, their types are the propositions they proof and logical connectives are identified with type constructors [Brouwer75,Curry34,Heyting30,Howard80,Kolmogoroff32]

ExTT

1

$\lambda\Pi$-calculus modulo

Leo-III

Extensional Type Theory (ExTT)
-> HOL + extensionality
+ choice
+ Rank-1 Polymorphism

Underlying Logic

Encodings of the Standard Library [Blanqui23]

$\lambda\Pi$-calculus modulo
-> HOL + dependant types
+ rewriting

Lambdapi

$$Set \ : \ TYPE$$

ExTT

Leo-III

1

λΠ-calculus modulo

Lambdapi

Encodings of the Standard Library [Blanqui23]

λΠ-calculus modulo
-> HOL + dependant types
+ rewriting

Extensional Type Theory (ExTT)
-> HOL + extensionality
+ choice
+ Rank-1 Polymorphism

Underlying Logic

$$Set \; : \; TYPE$$

$$o \; : \; Set \qquad \iota \; : \; Set$$

$$\rightsquigarrow \; : \; Set \to Set \to Set$$

ExTT, Problems

1

2

$\lambda\Pi$-calculus modulo

Leo-III

Encodings of the Standard Library [Blanqui23]

$\lambda\Pi$-calculus modulo
-> HOL + dependant types
+ rewriting

Lambdapi

Extensional Type Theory (ExTT)
-> HOL + extensionality
+ choice
+ Rank-1 Polymorphism

Underlying Logic

$$Set : TYPE \qquad \tau : Set \to TYPE$$
$$o : Set \qquad \iota : Set$$
$$\rightsquigarrow : Set \to Set \to Set$$

$f : \tau\ (\iota \rightsquigarrow \iota)$

$$\tau\ (\mu \rightsquigarrow \nu) \hookrightarrow \tau\ \mu \to \tau\ \nu$$

Rewrite rules

Problem

$f_{\iota\to\iota}$

ExTT, Problems

1

2

λΠ-calculus modulo

Leo-III

Lambdapi

Encodings of the Standard Library [Blanqui23]

λΠ-calculus modulo
-> HOL + dependant types
+ rewriting

Extensional Type Theory (ExTT)
-> HOL + extensionality
+ choice
+ Rank-1 Polymorphism

Underlying Logic

Problem

$f_{\iota \to \iota}$

$$Set : TYPE \qquad \tau : Set \to TYPE$$

$$o : Set \qquad \iota : Set$$

$$\rightsquigarrow : Set \to Set \to Set$$

$$f : \tau\,(\iota \rightsquigarrow \iota)$$

$$\tau\,(\mu \rightsquigarrow \nu) \hookrightarrow \tau\,\mu \to \tau\,\nu$$

Rewrite rules

Dependant types

$$= : \Pi\, t : Set\,.\,\tau\,(t \rightsquigarrow t \rightsquigarrow o)$$

ExTT, Problems

1

2

$\lambda\Pi$-calculus modulo

Leo-III

Encodings of the Standard Library [Blanqui23]

$\lambda\Pi$-calculus modulo
-> HOL + dependant types
+ rewriting

Lambdapi

Propositions as types!

Extensional Type Theory (ExTT)
-> HOL + extensionality
+ choice
+ Rank-1 Polymorphism

Underlying Logic

$$Set \ : \ TYPE \qquad \tau \ : \ Set \to TYPE$$

$$o \ : \ Set \qquad \iota \ : \ Set$$

$$\rightsquigarrow \ : \ Set \to Set \to Set$$

$$f : \tau\,(\iota \rightsquigarrow \iota)$$

$$\tau\,(\mu \rightsquigarrow \nu) \ \hookrightarrow \ \tau\,\mu \to \tau\,\nu$$

Problem

$f_{\iota \to \iota}$

Rewrite rules

Dependant types

Proof Step

$f_{\iota\to\iota} = g_{\iota\to\iota}$

$$= \ : \Pi\, t : Set\,.\ \tau\,(t \rightsquigarrow t \rightsquigarrow o)$$

$$\pi \ : \tau\, o \to TYPE$$

$$step_m : \pi(f = g)$$

EP

1 2 3

Encoded Rules

Leo-III

Calculus

EP
(a calculus for Extensional Higher-Order Paramodulation)

$f_{\iota\to\iota} = g_{\iota\to\iota}$

$f_{\iota\to\iota} X = g_{\iota\to\iota} X$

e.g: PFE

$$\frac{s_{\tau\to\nu} = t_{\tau\to\nu}}{s_{\tau\to\nu} X_\tau = t_{\tau\to\nu} X_\tau}$$

Lambdapi

$\lambda\Pi$-calculus modulo
-> HOL + dependant types
+ rewriting

Propositions
as types!

$step_m : \pi(f = g)$

?

$step_n : \Pi\, x : \tau\, \iota\, .\ \pi(f\, x = g\, x)$

EP

Encoded Rules

1 2 3

Leo-III

Calculus

EP
(a calculus for Extensional Higher-Order Paramodulation)

$f_{\iota\to\iota} = g_{\iota\to\iota}$

$f_{\iota\to\iota}X = g_{\iota\to\iota}X$

e.g: PFE

$$\frac{s_{\tau\to\nu} = t_{\tau\to\nu}}{s_{\tau\to\nu}X_\tau = t_{\tau\to\nu}X_\tau}$$

Lambdapi

$\lambda\Pi$-calculus modulo
-> HOL + dependant types
+ rewriting

Propositions as types!

$step_m : \pi(f = g)$

$PFE \ : \ \pi(s = t) \to \pi(s\,x = t\,x)$

$step_n : \Pi\, x : \tau\,\iota\,.\,\pi(f\,x = g\,x)$

EP

Encoded Rules

1 2 3

Leo-III

Calculus

EP
(a calculus for Extensional Higher-Order Paramodulation)

$f_{\iota\to\iota} = g_{\iota\to\iota}$

$f_{\iota\to\iota} X = g_{\iota\to\iota} X$

e.g: PFE

$$\frac{s_{\tau\to\nu} = t_{\tau\to\nu}}{s_{\tau\to\nu} X_\tau = t_{\tau\to\nu} X_\tau}$$

Lambdapi

$\lambda\Pi$-calculus modulo
-> HOL + dependant types + rewriting

Propositions as types!

$step_m : \pi(f = g)$

Dependant types

$$PFE \;:\; \Pi s : \tau\,(\mu \rightsquigarrow \nu)\,.\,\Pi t : \tau\,(\mu \rightsquigarrow \nu)\,.\,\Pi x : \tau\,\mu\,.\; \pi(s = t) \to \pi(s\,x = t\,x)$$

$step_n : \Pi\, x : \tau\,\iota\,.\,\pi(f\,x = g\,x)$

EP

1 2 3

Encoded Rules

Leo-III

Calculus

EP
(a calculus for Extensional Higher-Order Paramodulation)

$f_{\iota\to\iota} = g_{\iota\to\iota}$

$f_{\iota\to\iota}X = g_{\iota\to\iota}X$

e.g: PFE

$$\frac{s_{\tau\to\nu} = t_{\tau\to\nu}}{s_{\tau\to\nu}X_\tau = t_{\tau\to\nu}X_\tau}$$

Lambdapi

$\lambda\Pi$-calculus modulo
-> HOL + dependant types
+ rewriting

Propositions as types!

$step_m : \pi(f = g)$

$$PFE \ : \ \Pi\mu : Set\,.\,\Pi\nu : Set\,.\ \Pi s : \tau\,(\mu \rightsquigarrow \nu)\,.\,\Pi t : \tau\,(\mu \rightsquigarrow \nu)\,.\,\Pi x : \tau\,\mu\,.\ \pi(s = t) \to \pi(s\,x = t\,x)$$

$step_n : \Pi\,x : \tau\,\iota\,.\,\pi(f\,x = g\,x)$

EP

1 2 3

Encoded Rules

Leo-III

Lambdapi

$\lambda\Pi$-calculus modulo
-> HOL + dependant types
+ rewriting

Propositions as types!

Calculus

EP
(a calculus for Extensional Higher-Order Paramodulation)

$f_{\iota\to\iota} = g_{\iota\to\iota}$

$f_{\iota\to\iota} X = g_{\iota\to\iota} X$

e.g: PFE

$$\frac{s_{\tau\to\nu} = t_{\tau\to\nu}}{s_{\tau\to\nu} X_\tau = t_{\tau\to\nu} X_\tau}$$

$step_m : \pi(f = g)$

$$\begin{aligned} PFE \ : \ & \Pi\mu : Set \, . \, \Pi\nu : Set \, . \\ & \Pi s : \tau\,(\mu \rightsquigarrow \nu) \, . \, \Pi t : \tau\,(\mu \rightsquigarrow \nu) \, . \, \Pi x : \tau\,\mu \, . \\ & \pi(s = t) \to \pi(s\,x = t\,x) := \ldots \end{aligned}$$

$step_n : \Pi\, x : \tau\,\iota \, . \, \pi(f\,x = g\,x)$

EP

1 2 3

Encoded Rules

Leo-III

Calculus

EP
(a calculus for Extensional Higher-Order Paramodulation)

$f_{\iota\to\iota} = g_{\iota\to\iota}$

$f_{\iota\to\iota}X = g_{\iota\to\iota}X$

e.g: PFE

$$\frac{s_{\tau\to\nu} = t_{\tau\to\nu}}{s_{\tau\to\nu}X_\tau = t_{\tau\to\nu}X_\tau}$$

Lambdapi

$\lambda\Pi$-calculus modulo
-> HOL + dependant types
+ rewriting

Propositions as types!

$step_m : \pi(f = g)$

$$\begin{aligned} PFE \ : \ & \Pi\mu : Set \,.\, \Pi\nu : Set \,. \\ & \Pi s : \tau\,(\mu \rightsquigarrow \nu)\,.\,\Pi t : \tau\,(\mu \rightsquigarrow \nu)\,.\,\Pi x : \tau\,\mu\,. \\ & \pi(s = t) \to \pi(s\,x = t\,x) := \ldots \end{aligned}$$

$PFE\ \iota\ \iota\ f\ g$

$\Pi\, x : \tau\,\iota\,.\,\pi(f = g) \to \pi(f\,x = g\,x)$

$step_n : \Pi\, x : \tau\,\iota\,.\,\pi(f\,x = g\,x)$

EP

1 2 3

Encoded Rules

Leo-III

Calculus

EP
(a calculus for Extensional Higher-Order Paramodulation)

$f_{\iota\to\iota} = g_{\iota\to\iota}$

$f_{\iota\to\iota}X = g_{\iota\to\iota}X$

e.g PFE:

$$\frac{s_{\tau\to\nu} = t_{\tau\to\nu} \lor C}{s_{\tau\to\nu}X_\tau = t_{\tau\to\nu}X_\tau \lor C}$$

Lambdapi

$\lambda\Pi$-calculus modulo
-> HOL + dependant types + rewriting

Propositions as types!

EP

Encoded Rules

1 2 3

Leo-III

Calculus

EP
(a calculus for Extensional Higher-Order Paramodulation)

$(f_{\iota\to\iota} = g_{\iota\to\iota}) \vee l_o$

$(f_{\iota\to\iota} X = g_{\iota\to\iota} X) \vee l_o$

e.g PFE:

$$\frac{s_{\tau\to\nu} = t_{\tau\to\nu} \vee C}{s_{\tau\to\nu} X_\tau = t_{\tau\to\nu} X_\tau \vee C}$$

Lambdapi

$\lambda\Pi$-calculus modulo
-> HOL + dependant types
+ rewriting

Propositions as types!

We need a function of type…

$$(\pi(f = g) \to \pi(f\,x = g\,x))$$
$$\to \pi((f = g) \vee l)$$
$$\to \pi((f\,x = g\,x) \vee l)$$

EP

Leo-III

Calculus

EP
(a calculus for Extensional Higher-Order Paramodulation)

$(f_{\iota\to\iota} = g_{\iota\to\iota}) \vee l_o$

$(f_{\iota\to\iota} X = g_{\iota\to\iota} X) \vee l_o$

e.g PFE:

$$\frac{s_{\tau\to\nu} = t_{\tau\to\nu} \quad \vee C}{s_{\tau\to\nu} X_\tau = t_{\tau\to\nu} X_\tau \quad \vee C}$$

1 2 3

Encoded Rules

Lambdapi

$\lambda\Pi$-calculus modulo
-> HOL + dependant types
+ rewriting

Propositions as types!

Rule modifying the literal

We need a function of type…

$$(\pi(f = g) \to \pi(f\,x = g\,x))$$
$$\to \pi((f = g) \vee l)$$
$$\to \pi((f\,x = g\,x) \vee l)$$

EP

Encoded Rules

1 2 3

Leo-III

Lambdapi

Calculus

EP
(a calculus for Extensional Higher-Order Paramodulation)

$\lambda\Pi$-calculus modulo
-> HOL + dependant types
+ rewriting

Propositions as types!

Rule modifying the literal

Orignal clause

We need a function of type…

$(\pi(f = g) \to \pi(f\,x = g\,x))$

$\to \pi((f = g) \vee l)$

$\to \pi((f\,x = g\,x) \vee l)$

$(f_{\iota\to\iota} = g_{\iota\to\iota}) \vee l_o$

$(f_{\iota\to\iota}X = g_{\iota\to\iota}X) \vee l_o$

e.g PFE:

$$\frac{s_{\tau\to\nu} = t_{\tau\to\nu} \vee C}{s_{\tau\to\nu}X_\tau = t_{\tau\to\nu}X_\tau \vee C}$$

EP

1 2 3

Encoded Rules

Leo-III

Lambdapi

$\lambda\Pi$-calculus modulo
-> HOL + dependant types + rewriting

Propositions as types!

Calculus

EP
(a calculus for Extensional Higher-Order Paramodulation)

Rule modifying the literal

We need a function of type...

$$(\pi(f = g) \to \pi(f\,x = g\,x))$$

Orignal clause

$$\to \pi((f = g) \vee l)$$

Modified clause

$$\to \pi((f\,x = g\,x) \vee l)$$

$(f_{\iota\to\iota} = g_{\iota\to\iota}) \vee l_o$

$(f_{\iota\to\iota}X = g_{\iota\to\iota}X) \vee l_o$

e.g PFE:

$$\frac{s_{\tau\to\nu} = t_{\tau\to\nu} \quad \vee C}{s_{\tau\to\nu}X_\tau = t_{\tau\to\nu}X_\tau \quad \vee C}$$

EP

Encoded Rules

1 2 3

Leo-III

Lambdapi

Calculus

EP
(a calculus for Extensional Higher-Order Paramodulation)

$\lambda\Pi$-calculus modulo
-> HOL + dependant types
+ rewriting

Propositions as types!

e.g PFE:

$$\frac{s_{\tau\to\nu} = t_{\tau\to\nu} \vee C}{s_{\tau\to\nu}X_\tau = t_{\tau\to\nu}X_\tau \vee C}$$

$(f_{\iota\to\iota} = g_{\iota\to\iota}) \vee l_o$

$(f_{\iota\to\iota}X = g_{\iota\to\iota}X) \vee l_o$

We need a function of type...

Rule modifying the literal: $(\pi(f = g) \to \pi(f\,x = g\,x))$

Orignal clause: $\to \pi((f = g) \vee l)$

Modified clause: $\to \pi((f\,x = g\,x) \vee l)$

This can be proven as the following theorem:

$$transform : \Pi\ l' : \tau o\ .$$

```
transform (f X = g X)
```

EP

1 2 3

Encoded Rules

Leo-III

Calculus

EP
(a calculus for Extensional Higher-Order Paramodulation)

$(f_{\iota\to\iota} = g_{\iota\to\iota}) \vee l_o$

$(f_{\iota\to\iota} X = g_{\iota\to\iota} X) \vee l_o$

e.g PFE:

$$\frac{s_{\tau\to\nu} = t_{\tau\to\nu} \vee C}{s_{\tau\to\nu} X_\tau = t_{\tau\to\nu} X_\tau \vee C}$$

Lambdapi

$\lambda\Pi$-calculus modulo
-> HOL + dependant types + rewriting

Propositions as types!

Rule modifying the literal

Orignal clause

Modified clause

We need a function of type...

$(\pi(f = g) \to \pi(f\,x = g\,x))$

$\to \pi((f = g) \vee l)$

$\to \pi((f\,x = g\,x) \vee l)$

This can be proven as the following theorem:

List of terms of type o
-> original clause

$$transform : \Pi\, l' : \tau o\,.\,\Pi\, c : L\, o\,.$$

```
transform (f X = g X) ((f = g) :: l :: □)
```

EP

1 2 3

Encoded Rules

Leo-III

Calculus

EP
(a calculus for Extensional Higher-Order Paramodulation)

$(f_{\iota\to\iota} = g_{\iota\to\iota}) \vee l_o$

$(f_{\iota\to\iota} X = g_{\iota\to\iota} X) \vee l_o$

e.g PFE:

$$\frac{s_{\tau\to\nu} = t_{\tau\to\nu} \quad \vee C}{s_{\tau\to\nu} X_\tau = t_{\tau\to\nu} X_\tau \quad \vee C}$$

Lambdapi

$\lambda\Pi$-calculus modulo
-> HOL + dependant types + rewriting

Propositions as types!

Rule modifying the literal

Orignal clause

Modified clause

We need a function of type…

$(\pi(f = g) \to \pi(f\,x = g\,x))$

$\to \pi((f = g) \vee l)$

$\to \pi((f\,x = g\,x) \vee l)$

This can be proven as the following theorem:

List of terms of type o
-> original clause

Natural numbers
-> Index of literal

$$transform : \Pi\, l' : \tau o\,.\,\Pi\, c : L\,o\,.\,\Pi\, n : N\,.$$

```
transform (f X = g X) ((f = g) :: l :: □) 0
```

EP

Encoded Rules

1 2 3

Leo-III

Lambdapi

Calculus

EP
(a calculus for Extensional Higher-Order Paramodulation)

$(f_{\iota\to\iota} = g_{\iota\to\iota}) \vee l_o$

$(f_{\iota\to\iota}X = g_{\iota\to\iota}X) \vee l_o$

e.g PFE:

$$\frac{s_{\tau\to\nu} = t_{\tau\to\nu} \vee C}{s_{\tau\to\nu}X_\tau = t_{\tau\to\nu}X_\tau \vee C}$$

$\lambda\Pi$-calculus modulo
-> HOL + dependant types + rewriting

Propositions as types!

Rule modifying the literal

Orignal clause

Modified clause

We need a function of type…

$(\pi(f = g) \to \pi(f\,x = g\,x))$

$\to \pi((f = g) \vee l)$

$\to \pi((f\,x = g\,x) \vee l)$

This can be proven as the following theorem:

List of terms of type o
-> original clause

Natural numbers
-> Index of literal

$$transform : \Pi\, l' : \tau o\,.\,\Pi\, c : L\,o\,.\,\Pi\, n : N\,.\, \pi((nth\,\bot\,c\,n) \Rightarrow l')$$

```
transform (f X = g X) ((f = g) :: l :: □) 0
```

EP

Leo-III

`Calculus`

EP
(a calculus for Extensional Higher-Order Paramodulation)

$(f_{\iota\to\iota} = g_{\iota\to\iota}) \vee l_o$

$(f_{\iota\to\iota} X = g_{\iota\to\iota} X) \vee l_o$

`e.g PFE:`

$$\frac{s_{\tau\to\nu} = t_{\tau\to\nu} \quad \vee\ C}{s_{\tau\to\nu} X_\tau = t_{\tau\to\nu} X_\tau \quad \vee\ C}$$

1 2 3

Encoded Rules

Lambdapi

$\lambda\Pi$-calculus modulo
-> HOL + dependant types
+ rewriting

Propositions as types!

Rule modifying the literal

Orignal clause

Modified clause

We need a function of type…

$(\pi(f = g) \to \pi(f\,x = g\,x))$
$\to \pi((f = g) \vee l)$
$\to \pi((f\,x = g\,x) \vee l)$

This can be proven as the following theorem:

List of terms of type o
-> original clause

Natural numbers
-> Index of literal

$$transform : \Pi\, l' : \tau o\,.\, \Pi\, c : L\, o\,.\, \Pi\, n : N\,.$$
$$\pi((nth \perp c\ n) \Rightarrow l')$$
$$\to \pi(disj\ c)$$

```
transform (f X = g X) ((f = g) :: l :: □) 0
```

EP

Encoded Rules

1 2 3

Leo-III

Lambdapi

Calculus

EP
(a calculus for Extensional Higher-Order Paramodulation)

e.g PFE:

$$\frac{s_{\tau\to\nu} = t_{\tau\to\nu} \lor C}{s_{\tau\to\nu}X_\tau = t_{\tau\to\nu}X_\tau \lor C}$$

$(f_{\iota\to\iota} = g_{\iota\to\iota}) \lor l_o$

$(f_{\iota\to\iota}X = g_{\iota\to\iota}X) \lor l_o$

$\lambda\Pi$-calculus modulo
-> HOL + dependant types
+ rewriting

Propositions as types!

Rule modifying the literal

Orignal clause

Modified clause

We need a function of type...

$$(\pi(f = g) \to \pi(f\,x = g\,x))$$
$$\to \pi((f = g) \lor l)$$
$$\to \pi((f\,x = g\,x) \lor l)$$

This can be proven as the following theorem:

List of terms of type o
-> original clause

Natural numbers
-> Index of literal

$$transform : \Pi\, l' : \tau o \,.\, \Pi\, c : L\, o \,.\, \Pi\, n : N \,.$$
$$\pi((nth \perp c\ n) \Rightarrow l')$$
$$\to \pi(disj\ c)$$
$$\to \pi(disj(set\_nth \perp c\ n\ l')) := \ldots$$

```
transform (f X = g X) ((f = g) :: l :: □) 0
```

Further Meta-Theorems:

https://github.com/melanie-taprogge/Leo-III-lambdapi-lib/blob/main/MetaTheorems.lp

EP

Encoded Rules

```
transform (f X = g X) ((f = g) :: ι :: □) 0
```

Further Meta-Theorems:

$$\frac{C_0 \wedge \ldots \wedge C_i \wedge \ldots \wedge C_n}{C_i}\ \texttt{select}$$

Given a proof of a conjunction of clauses, proof an arbitrary one of the clauses

https://github.com/melanie-taprogge/Leo-III-lambdapi-lib/blob/main/MetaTheorems.lp

```
transform (f X = g X) ((f = g) :: ι :: □) 0
```

Further Meta-Theorems:

$$\frac{C_0 \wedge \ldots \wedge C_i \wedge \ldots \wedge C_n}{C_i}\ \texttt{select}$$

Given a proof of a conjunction of clauses, proof an arbitrary one of the clauses

$$\frac{(l_0 \vee l_1 \vee \ldots \vee l_n)}{(l_{\sigma(0)} \vee l_{\sigma(1)} \vee \ldots \vee l_{\sigma(n)})}\ \texttt{permute}$$

Given a proof of a disjunction of literals, proof an arbitrary permutation of them

https://github.com/melanie-taprogge/Leo-III-lambdapi-lib/blob/main/MetaTheorems.lp

```
transform (f X = g X) ((f = g) :: ι :: □) 0
```

Further Meta-Theorems:

$$\frac{C_0 \wedge \ldots \wedge C_i \wedge \ldots \wedge C_n}{C_i}\ \texttt{select}$$

Given a proof of a conjunction of clauses, proof an arbitrary one of the clauses

$$\frac{(l_0 \vee l_1 \vee \ldots \vee l_n)}{(l_{\sigma(0)} \vee l_{\sigma(1)} \vee \ldots \vee l_{\sigma(n)})}\ \texttt{permute}$$

Given a proof of a disjunction of literals, proof an arbitrary permutation of them

$$\frac{l_0 \vee \ldots \vee l_i \vee \ldots \vee l_i \vee \ldots \vee l_n}{l_0 \vee \ldots \vee l_i \vee \ldots \vee l_n}\ \texttt{delete}$$

Given a proof of a disjunction of literals, proof that any double occurrences can be omitted

https://github.com/melanie-taprogge/Leo-III-lambdapi-lib/blob/main/MetaTheorems.lp

```
transform (f X = g X) ((f = g) :: l :: □) 0
```

Found Proof

1 2 3 4

Encoded Proof

Leo-III

Lambdapi

Calculus

EP
(a calculus for Extensional Higher-Order Paramodulation)

$(f_{\iota\to\iota} = g_{\iota\to\iota}) \vee l_o$

$(f_{\iota\to\iota}X = g_{\iota\to\iota}X) \vee l_o$

e.g PFE:

$$\frac{s_{\tau\to\nu} = t_{\tau\to\nu} \quad \vee C}{s_{\tau\to\nu}X_\tau = t_{\tau\to\nu}X_\tau \quad \vee C}$$

$step_m : \pi\,((f = g) \vee l)$

PFE

$step_n : \Pi x : El\,\iota\,.\,\pi\,((f\,x = g\,x) \vee l)$

```
symbol step_m :
  π ((f = g ) ∨ l) ≔ …
```

```
symbol step_n : Π (x : τ ι),
  π ((f x = g x) ∨ l) ≔
begin
  assume x;
  refine transform [f x = g x] ((f = g) :: l :: □) 0
        (PFE ι ι f g x) step_m;
end;
```

Found Proof

Encoded Proof

1 2 3 4

Leo-III

Lambdapi

e.g: Simplification

$$\frac{(s = t) \vee \ldots}{(simp(s) = simp(t)) \vee \ldots}$$

Tactics make constructing proof-terms more convenient!

Found Proof

1 2 3 4

Encoded Proof

Leo-III

Lambdapi

Tactics make constructing proof-terms more convenient!

e.g: Simplification

$$\frac{(s = t) \vee \ldots}{(simp(s) = simp(t)) \vee \ldots}$$

$a \wedge a \rightarrow a$

$a = a \rightarrow \top$

$\neg(\top) \rightarrow \bot$

$\ldots$

simp: Exhaustively apply boolean identities

Found Proof

1 2 3 4

Encoded Proof

Leo-III

Lambdapi

Tactics make constructing proof-terms more convenient!

`e.g: Simplification`

$$\frac{(s = t) \vee \ldots}{(simp(s) = simp(t)) \vee \ldots}$$

$a \wedge a \rightarrow a$
$a = a \rightarrow \top$
$\neg(\top) \rightarrow \bot$
$\ldots$

simp: Exhaustively apply boolean identities

$$\neg((a_o \wedge a_o) = a_o)$$

$$\bot$$

Found Proof

1 | 2 | 3 | 4

Encoded Proof

Leo-III

Lambdapi

`e.g: Simplification`

$$\frac{(s = t) \lor \ldots}{(simp(s) = simp(t)) \lor \ldots}$$

$a \land a \to a$
$a = a \to \top$
$\neg(\top) \to \bot$
$\ldots$

simp: Exhaustively apply boolean identities

$\neg((a_o \land a_o) = a_o)$

$\bot$

Tactics make constructing proof-terms more convenient!

`rewrite` tactic can use proofs of equalities to modify substructures of terms.

Found Proof

1 2 3 4

Encoded Proof

Leo-III

Lambdapi

`e.g: Simplification`

$$\frac{(s = t) \vee \dots}{(simp(s) = simp(t)) \vee \dots}$$

$a \wedge a \rightarrow a$
$a = a \rightarrow \top$
$\neg(\top) \rightarrow \bot$
$\dots$

simp: Exhaustively apply boolean identities

$\neg((a_o \wedge a_o) = a_o)$

$\bot$

Tactics make constructing proof-terms more convenient!

$\wedge_{idem} : \Pi x : \tau o \,.\, \pi((x \wedge x) = x)$

`rewrite` tactic can use proofs of equalities to modify substructures of terms.

Found Proof

1 2 3 4

Encoded Proof

Leo-III

Lambdapi

`e.g: Simplification`

$$\frac{(s = t) \vee \ldots}{(simp(s) = simp(t)) \vee \ldots}$$

$a \wedge a \rightarrow a$
$a = a \rightarrow \top$
$\neg(\top) \rightarrow \bot$
$\ldots$

simp: Exhaustively apply boolean identities

$\neg((a_o \wedge a_o) = a_o)$

$\bot$

Tactics make constructing proof-terms more convenient!

$\wedge_{idem} : \Pi x : \tau o \,.\, \pi((x \wedge x) = x)$

$=_{idem} : \Pi x : \tau o \,.\, \pi((x = x) = \top)$

$\top_{neg} : \pi(\neg \top = \bot)$

`rewrite` tactic can use proofs of equalities to modify substructures of terms.

Found Proof

1 2 3 4

Encoded Proof

Leo-III

Lambdapi

e.g: Simplification

$$\frac{(s = t) \vee \ldots}{(simp(s) = simp(t)) \vee \ldots}$$

$a \wedge a \rightarrow a$
$a = a \rightarrow \top$
$\neg(\top) \rightarrow \bot$
$\ldots$

simp: Exhaustively apply boolean identities

$\neg((a_o \wedge a_o) = a_o)$

$\bot$

Tactics make constructing proof-terms more convenient!

$\wedge_{idem} : \Pi x : \tau o \,.\, \pi((x \wedge x) = x)$

$=_{idem} : \Pi x : \tau o \,.\, \pi((x = x) = \top)$

$\top_{neg} : \pi(\neg \top = \bot)$

`rewrite` tactic can use proofs of equalities to modify substructures of terms.

```
have simpStep: π ((¬((a ∧ a) = a)) = ⊥) :=
begin


end;
```

Found Proof

1 | 2 3 | 4

Encoded Proof

Leo-III

Lambdapi

`e.g: Simplification`

$$\frac{(s = t) \vee \dots}{(simp(s) = simp(t)) \vee \dots}$$

$a \wedge a \rightarrow a$
$a = a \rightarrow \top$
$\neg(\top) \rightarrow \bot$
$\dots$

simp: Exhaustively apply boolean identities

$\neg((a_o \wedge a_o) = a_o)$

$\bot$

Tactics make constructing proof-terms more convenient!

$\wedge_{idem} : \Pi x : \tau o \, . \, \pi((x \wedge x) = x)$

$=_{idem} : \Pi x : \tau o \, . \, \pi((x = x) = \top)$

$\top_{neg} : \pi(\neg \top = \bot)$

`rewrite` tactic can use proofs of equalities to modify substructures of terms.

```
have simpStep: π ((¬((a ∧ a) = a)) = ⊥) :=
begin
  rewrite ∧_idem; rewrite =_idem; rewrite ¬⊤;

end;
```

Found Proof

1 | 2 | 3 | 4

Encoded Proof

Leo-III

Lambdapi

`e.g: Simplification`

$$\frac{(s = t) \lor \ldots}{(simp(s) = simp(t)) \lor \ldots}$$

$a \land a \to a$
$a = a \to \top$
$\neg(\top) \to \bot$
$\ldots$

simp: Exhaustively apply boolean identities

$\neg((a_o \land a_o) = a_o)$

$\bot$

Tactics make constructing proof-terms more convenient!

$\land_{idem} : \Pi x : \tau o \,.\, \pi((x \land x) = x)$

$=_{idem}: \Pi x : \tau o \,.\, \pi((x = x) = \top\,)$

$\top_{neg} : \pi(\neg \top = \bot\,)$

`rewrite` tactic can use proofs of equalities to modify substructures of terms:

```
have simpStep: π ((¬((a ∧ a) = a)) = ⊥) :=
begin
  rewrite ∧_idem; rewrite =_idem; rewrite ¬⊤;
  reflexivity
end;
```

Found Proof

Leo-III

e.g: Simplification

$$\frac{(s = t) \lor \ldots}{(simp(s) = simp(t)) \lor \ldots}$$

$a \land a \to a$
$a = a \to \top$
$\neg(\top) \to \bot$
$\ldots$

simp: Exhaustively apply boolean identities

$\neg((a_o \land a_o) = a_o)$

$\bot$

Encoded Proof

1 2 3 4

Lambdapi

Tactics make constructing proof-terms more convenient!

$\land_{idem} : \Pi x : \tau o \,.\, \pi((x \land x) = x)$

$=_{idem} : \Pi x : \tau o \,.\, \pi((x = x) = \top)$

$\top_{neg} : \pi(\neg \top = \bot)$

`rewrite` tactic can use proofs of equalities to modify substructures of terms:

```
have simpStep: π ((¬((a ∧ a) = a)) = ⊥) :=
begin
  rewrite ∧_idem; rewrite =_idem; rewrite ¬⊤;
  reflexivity
end;
```

Note: this can directly be used to encode Leo-III rule *RW*!

Found Proof

1 2 3 4

Encoded Proof

Leo-III

Lambdapi

`e.g: Simplification`

$$\frac{(s = t) \vee \ldots}{(simp(s) = simp(t)) \vee \ldots}$$

$a \wedge a \to a$

$a = a \to \top$

$\neg(\top) \to \bot$

$\ldots$

simp: Exhaustively apply boolean identities

$\neg((a_o \wedge a_o) = a_o)$

$\bot$

Tactics make constructing proof-terms more convenient!

$\wedge_{idem} : \Pi x : \tau o\, .\ \pi((x \wedge x) = x)$

$=_{idem} : \Pi x : \tau o\, .\ \pi((x = x) = \top)$

$\top_{neg} : \pi(\neg \top = \bot)$

User-defined tactics via `eval`, `orelse`, `repeat`, … :

Found Proof

1 2 3 4

Encoded Proof

Leo-III

Lambdapi

`e.g: Simplification`

$$\frac{(s = t) \lor \ldots}{(simp(s) = simp(t)) \lor \ldots}$$

$a \land a \to a$
$a = a \to \top$
$\neg(\top) \to \bot$
$\ldots$

simp: Exhaustively apply boolean identities

$\neg((a_o \land a_o) = a_o)$

$\bot$

Tactics make constructing proof-terms more convenient!

$\land_{idem} : \Pi x : \tau o \,.\, \pi((x \land x) = x)$

$=_{idem} : \Pi x : \tau o \,.\, \pi((x = x) = \top)$

$\top_{neg} : \pi(\neg \top = \bot)$

User-defined tactics via `eval`, `orelse`, `repeat`, … :

```
symbol simp_tac:= #repeat (#reflexivity #orelse
                  (#rewrite ∧_idem) #orelse …);
```

Found Proof

1 2 3 4

Encoded Proof

Leo-III

Lambdapi

e.g: Simplification

$$\frac{(s = t) \vee \ldots}{(simp(s) = simp(t)) \vee \ldots}$$

$a \wedge a \rightarrow a$
$a = a \rightarrow \top$
$\neg(\top) \rightarrow \bot$
$\ldots$

simp: Exhaustively apply boolean identities

$\neg((a_o \wedge a_o) = a_o)$

$\bot$

Tactics make constructing proof-terms more convenient!

$\wedge_{idem} : \Pi x : \tau o \,.\, \pi((x \wedge x) = x)$

$=_{idem} : \Pi x : \tau o \,.\, \pi((x = x) = \top)$

$\top_{neg} : \pi(\neg \top = \bot)$

User-defined tactics via `eval`, `orelse`, `repeat`, … :

```
symbol simp_tac:= #repeat (#reflexivity #orelse
                  (#rewrite ∧_idem) #orelse …);
```

```
have simpStep: π ((¬((a ∧ a) = a)) = ⊥) :=
begin
  eval simp_tac;
end;
```

Encoded Rules

Deciding how inference rule should be encoded

Corresponding Lambdapi operation?

yes

Encoding as …	-
Application to the clause	via corresponding tactic(s)
Example	RW

Encoded Rules

Deciding how inference rule should be encoded

Corresponding Lambdapi operation?

yes

no

Structure operated on

whole clause

substructure

Parent-child-relationship

consequence

Encoding as …	-	a function
Application to the clause	via corresponding tactic(s)	directly / using transform
Example	RW	PFE

Encoded Rules

Deciding how inference rule should be encoded

Corresponding Lambdapi operation?

yes

no

Structure operated on

whole clause

substructure

Parent-child-relationship

consequence

equivalent

	yes	whole clause / consequence	equivalent
Encoding as …	-	a function	an equality
Application to the clause	via corresponding tactic(s)	directly / using transform	using the rewrite tactic
Example	RW	PFE	Simp

Found Proof

1 2 3 **4**

Encoded Proof

Leo-III

Lambdapi

Calculus

EP
(a calculus for Extensional Higher-Order Paramodulation)

$step_m : \pi((f = g) \lor l)$ $\xrightarrow{PFE}$ $\Pi x : \tau\,\iota\,.\,\pi((f\,x = g\,x) \lor l)$

$(f_{\iota\to\iota} = g_{\iota\to\iota}) \lor l_o$

$(f_{\iota\to\iota}X = g_{\iota\to\iota}X) \lor l_o$

e.g PFE:

$$\frac{s_{\tau\to\nu} = t_{\tau\to\nu} \;\boxed{\lor\, C}}{s_{\tau\to\nu}X_\tau = t_{\tau\to\nu}X_\tau \;\boxed{\lor\, C}}$$

Found Proof

1

2

3

4

Encoded Proof

Leo-III

Calculus

EP
(a calculus for Extensional Higher-Order Paramodulation)

$(f_{\iota\to\iota} = g_{\iota\to\iota}) \vee l_o$

$(g_{\iota\to\iota}X = f_{\iota\to\iota}X) \vee l_o$

e.g PFE:

$$\frac{s_{\tau\to\nu} = t_{\tau\to\nu} \boxed{\vee\, C}}{s_{\tau\to\nu}X_\tau = t_{\tau\to\nu}X_\tau \boxed{\vee\, C}}$$

Lambdapi

$step_m : \pi\,((f = g) \vee l)$

PFE

$\Pi\, x : \tau\,\iota\,.\,\pi\,((f\,x = g\,x) \vee l)$

Found Proof

1 2 3 4

Encoded Proof

Leo-III

Lambdapi

Calculus

EP
(a calculus for Extensional Higher-Order Paramodulation)

$(f_{\iota\to\iota} = g_{\iota\to\iota}) \vee l_o$

$l_o \vee (g_{\iota\to\iota} X = f_{\iota\to\iota} X)$

e.g PFE:

$$\frac{s_{\tau\to\nu} = t_{\tau\to\nu} \ \boxed{\vee\, C}}{s_{\tau\to\nu} X_\tau = t_{\tau\to\nu} X_\tau \ \boxed{\vee\, C}}$$

$step_m : \pi((f = g) \vee l)$

PFE

$\Pi x : \tau\,\iota\,.\,\pi((f\,x = g\,x) \vee l)$

Found Proof

1 2 3 4

Encoded Proof

Leo-III

Lambdapi

Calculus

EP
(a calculus for Extensional Higher-Order Paramodulation)

$step_m : \pi((f = g) \vee l)$

PFE

$\Pi\, x : \tau\, \iota\, .\, \pi((f\, x = g\, x) \vee l)$

?

$step_n : \Pi\, x : \tau\, \iota\, .\, \pi(l \vee (g\, x = f\, x))$

$(f_{\iota\to\iota} = g_{\iota\to\iota}) \vee l_o$

$l_o \vee (g_{\iota\to\iota} X = f_{\iota\to\iota} X)$

e.g PFE:

$$\frac{s_{\tau\to\nu} = t_{\tau\to\nu} \quad \vee C}{s_{\tau\to\nu} X_\tau = t_{\tau\to\nu} X_\tau \quad \vee C}$$

Found Proof

1 2 3 4

Encoded Proof

Leo-III

Lambdapi

Calculus

EP
(a calculus for Extensional Higher-Order Paramodulation)

$(f_{\iota\to\iota} = g_{\iota\to\iota}) \vee l_o$

$l_o \vee (g_{\iota\to\iota}X = f_{\iota\to\iota}X)$

e.g PFE:

$$\frac{s_{\tau\to\nu} = t_{\tau\to\nu} \quad \vee C}{s_{\tau\to\nu}X_\tau = t_{\tau\to\nu}X_\tau \quad \vee C}$$

$step_m : \pi((f = g) \vee l)$

PFE

$\Pi\, x : \tau\, \iota\, .\, \pi((f\, x = g\, x) \vee l)$

permute

$=_{sym}: \Pi t : Set\, .\, \Pi x : \tau\, t\, .\, \Pi y : \tau\, t\, .\, \pi((x = y) = (y = x))$

$step_n : \Pi\, x : \tau\, \iota\, .\, \pi(l \vee (g\, x = f\, x))$

Identify possible additional modifications and encode them as inference rules

Found Proof

1 2 3 4

Encoded Proof

Leo-III

2. Define a modular encoding scheme for each individual calculus rule

Lambdapi

```
symbol step_m : π((f = g) ∨ l) ≔ …
```

```
symbol step_n : Π x, π(l ∨ (g x = f x)) ≔
begin
  assume x;

end;
```

Found Proof

1 2 3 4

Encoded Proof

Leo-III

Lambdapi

2. Define a modular encoding scheme for each individual calculus rule

1 . Apply the inference rule via transform/ rewrite/ application to function (e.g. PFE)

```
symbol step_m : π((f = g) ∨ l) ≔ …
```

```
symbol step_n : Π x, π(l ∨ (g x = f x)) ≔
begin
  assume x;

have FunExtApp: π((f x = g x) ∨ l)
   {refine transform [f x = g x]
    ((f = g) :: l :: □) 0 (PFE ι ι f g x) step_n};

end;
```

Found Proof

1 | 2 | 3 | 4

Encoded Proof

Leo-III

2. Define a modular encoding scheme for each individual calculus rule

1 . **Apply the inference rule via transform/ rewrite/ application to function** (e.g. PFE)

2. **Verify the other implicit transformations effecting single literals** (e.g. eqSym)

Lambdapi

```
symbol step_m : π((f = g) ∨ l) ≔ …
```

```
symbol step_n : Π x, π(l ∨ (g x = f x)) ≔
begin
  assume x;

  have FunExtApp: π((f x = g x) ∨ l)
    {refine transform [f x = g x]
     ((f = g) :: l :: □) 0 (PFE ι ι f g x) step_n};

  have ImplicitTransformations: π((g x = f x) ∨ l)
    {rewrite (=_sym (g x) (f x));
     refine FunExtApp};

end;
```

Found Proof

1 | 2 | 3 | 4

Encoded Proof

Leo-III

Lambdapi

2. Define a modular encoding scheme for each individual calculus rule

```
symbol step_m : π((f = g) ∨ l) ≔ …
```

```
symbol step_n : Π x, π(l ∨ (g x = f x)) ≔
begin
  assume x;

 have FunExtApp: π((f x = g x) ∨ l)
   {refine transform [f x = g x]
    ((f = g) :: l :: □) 0 (PFE ι ι f g x) step_n};

 have ImplicitTransformations: π((g x = f x) ∨ l)
   {rewrite (=_sym (g x) (f x));
    refine FunExtApp};

 refine permute (1 :: 0 :: □)
  ((g x = f x) :: l :: □) ⊤ᵢ ImplicidTransformations

end;
```

1. **Apply the inference rule via transform/ rewrite/ application to function** (e.g. PFE)

2. **Verify the other implicit transformations effecting single literals** (e.g. eqSym)

3. **Apply the implicit transformations effecting the whole-clause** (e.g. permutation)

Found Proof

1 | 2 | 3 | 4

Encoded Proof

Leo-III

Lambdapi

2. Define a modular encoding scheme for each individual calculus rule

0 . Apply any implicit transformations necessary to apply inference rules

1 . **Apply the inference rule via transform/ rewrite/ application to function** (e.g. PFE)

2. **Verify the other implicit transformations effecting single literals** (e.g. eqSym)

3. **Apply the implicit transformations effecting the whole-clause** (e.g. permutation)

```
symbol step_m : π((f = g) ∨ l) ≔ …
```

```
symbol step_n : Π x, π(l ∨ (g x = f x)) ≔
begin
  assume x;

 have FunExtApp: π((f x = g x) ∨ l)
   {refine transform [f x = g x]
    ((f = g) :: l :: □) 0 (PFE ι ι f g x) step_n};

 have ImplicitTransformations: π((g x = f x) ∨ l)
   {rewrite (=_sym (g x) (f x));
    refine FunExtApp};

 refine permute (1 :: 0 :: □)
  ((g x = f x) :: l :: □) ⊤ᵢ ImplicidTransformations

end;
```

Current State

(partial) implementation of proof steps encoded in Lambdapi for 14 of the 26 Leo-III rules relevant for the encoding!

Current State

(partial) implementation of proof steps encoded in Lambdapi for 14 of the 26 Leo-III rules relevant for the encoding!

Coverage & Proof Statistics

Metric	Value
Provable problems	1691
Total proof steps	32175
Encoded steps	20807
Encoding coverage	65 %

References

- Assaf, A., Burel, G., Cauderlier, R., Delahaye, D., Dowek, G., Dubois, C., ... & Saillard, R. (2016). Dedukti: a logical framework based on the λΠ-calculus modulo theory.
- Blanqui, Frédéric, et al. "A modular construction of type theories." *Logical Methods in Computer Science* 19 (2023).
- Cousineau, D., & Dowek, G. (2007). Embedding pure type systems in the lambda-pi-calculus modulo. In Typed Lambda Calculi and Applications: 8th International Conference, TLCA 2007, Paris, France, June 26-28, 2007. Proceedings 8 (pp. 102-117). Springer Berlin Heidelberg.
- Curry, H. B. (1934). Functionality in combinatory logic. *Proceedings of the National Academy of Sciences*, *20*(11), 584-590.
- Henkin, Leon. "Completeness in the theory of types1." *The Journal of Symbolic Logic* 15.2 (1950): 81-91.
- Heyting, Arend. "Die formalen Regeln der intuitionistischen Logik." Sitzungsbericht PreuBische Akademie der Wissenschaften Berlin, physikalisch-mathematische Klasse II (1930): 42-56.
- Hondet, Gabriel, and Frédéric Blanqui. "The new rewriting engine of dedukti." *arXiv preprint arXiv:2010.16115* (2020).
- Howard, W. A. (1980). The formulae-as-types notion of construction. To HB Curry: essays on combinatory logic, lambda calculus and formalism, 44, 479-490.
- Kolmogoroff, Andrej. "Zur deutung der intuitionistischen logik." Mathematische Zeitschrift 35.1 (1932): 58-65.
- Steen, A. (2020). Extensional paramodulation for higher-order logic and its effective implementation Leo-III. *KI-Künstliche Intelligenz*, *34*(1), 105-108.

Implementation Progress

Unification rules $\mathcal{UNI}$

$$\frac{\mathcal{C} \vee [s_\tau \simeq s_\tau]^{\mathrm{ff}}}{\mathcal{C}}\ (\text{Triv}) \qquad \frac{\mathcal{C} \vee [X_\tau \simeq s_\tau]^{\mathrm{ff}}}{\mathcal{C}\{s/X\}}\ (\text{Bind})^\dagger$$

$$\frac{\mathcal{C} \vee [c\ \overline{s^i} \simeq c\ \overline{t^i}]^{\mathrm{ff}}}{\mathcal{C} \vee [s^1 \simeq t^1]^{\mathrm{ff}} \vee \ldots \vee [s^n \simeq t^n]^{\mathrm{ff}}}\ (\text{Decomp})$$

$$\frac{\mathcal{C} \vee [X_{\nu\overline{\mu}}\ \overline{s^i} \simeq c_{\nu\overline{\tau}}\ \overline{t^j}]^{\mathrm{ff}} \qquad g_{\nu\overline{\mu}} \in \mathcal{GB}^{\{c\}}_{\nu\overline{\mu}}}{\mathcal{C} \vee [X_{\nu\overline{\mu}}\ \overline{s^i} \simeq c_{\nu\overline{\tau}}\ \overline{t^j}]^{\mathrm{ff}} \vee [X \simeq g]^{\mathrm{ff}}}\ (\text{FlexRigid})$$

$$\frac{\mathcal{C} \vee [X_{\nu\overline{\mu}}\ \overline{s^i} \simeq Y_{\nu\overline{\tau}}\ \overline{t^j}]^{\mathrm{ff}} \qquad g_{\nu\overline{\mu}} \in \mathcal{GB}^{\{h\}}_{\nu\overline{\mu}}}{\mathcal{C} \vee [X_{\nu\overline{\mu}}\ \overline{s^i} \simeq Y_{\nu\overline{\tau}}\ \overline{t^j}]^{\mathrm{ff}} \vee [X \simeq g]^{\mathrm{ff}}}\ (\text{FlexFlex})^\ddagger$$

†: where $X_\tau \notin \mathrm{fv}(s)$ ‡: where $h \in \Sigma$ is an appropriate constant

Extensionality rules $\mathcal{EXT}$

$$\frac{\mathcal{C} \vee [s_o \simeq t_o]^{\mathrm{tt}}}{\mathcal{C} \vee [s_o]^{\mathrm{tt}} \vee [t_o]^{\mathrm{ff}} \quad \mathcal{C} \vee [s_o]^{\mathrm{ff}} \vee [t_o]^{\mathrm{tt}}}\ (\text{PBE}) \qquad \frac{\mathcal{C} \vee [s_o \simeq t_o]^{\mathrm{ff}}}{\mathcal{C} \vee [s_o]^{\mathrm{tt}} \vee [t_o]^{\mathrm{tt}} \quad \mathcal{C} \vee [s_o]^{\mathrm{ff}} \vee [t_o]^{\mathrm{ff}}}\ (\text{NBE})$$

$$\frac{\mathcal{C} \vee [s_{\nu\tau} \simeq t_{\nu\tau}]^{\mathrm{tt}}}{\mathcal{C} \vee [s\ X_\tau \simeq t\ X_\tau]^{\mathrm{tt}}}\ (\text{PFE})^\dagger \qquad \frac{\mathcal{C} \vee [s_{\nu\tau} \simeq t_{\nu\tau}]^{\mathrm{ff}}}{\mathcal{C} \vee [s\ \mathrm{sk}_\tau \simeq t\ \mathrm{sk}_\tau]^{\mathrm{ff}}}\ (\text{NFE})^\ddagger$$

†: where X_τ is fresh for $\mathcal{C}$ ‡: where sk_τ is a Skolem term

Clausification rules $\mathcal{CNF}$

$$\frac{\mathcal{C} \vee [(l_\tau = r_\tau) \simeq \top]^\alpha}{\mathcal{C} \vee [l_\tau \simeq r_\tau]^\alpha}\ (\text{LiftEq}) \qquad \frac{\mathcal{C} \vee [\neg s_o]^\alpha}{\mathcal{C} \vee [s_o]^{\overline{\alpha}}}\ (\text{CNFNeg})$$

$$\frac{\mathcal{C} \vee [s_o \vee t_o]^{\mathrm{tt}}}{\mathcal{C} \vee [s_o]^{\mathrm{tt}} \vee [t_o]^{\mathrm{tt}}}\ (\text{CNFDisj}) \qquad \frac{\mathcal{C} \vee [s_o \vee t_o]^{\mathrm{ff}}}{\mathcal{C} \vee [s_o]^{\mathrm{ff}} \quad \mathcal{C} \vee [t_o]^{\mathrm{ff}}}\ (\text{CNFConj})$$

$$\frac{\mathcal{C} \vee [\forall X_\tau.\, s_o]^{\mathrm{tt}}}{\mathcal{C} \vee [s_o[Z/X]]^{\mathrm{tt}}}\ (\text{CNFAll})^\dagger \qquad \frac{\mathcal{C} \vee [\forall X_\tau.\, s_o]^{\mathrm{ff}}}{\mathcal{C} \vee [s_o[\mathrm{sk}\ \overline{\mathrm{fv}(\mathcal{C})}/X]]^{\mathrm{ff}}}\ (\text{CNFExists})^\ddagger$$

†: where Z_τ is a fresh variable for $\mathcal{C}$ ‡: where sk is a new Skolem constant of appropriate type

Primary inference rules $\mathcal{PI}$

$$\frac{\mathcal{C} \vee [s_\tau \simeq t_\tau]^\alpha \qquad \mathcal{D} \vee [l_\nu \simeq r_\nu]^{\mathrm{tt}}}{[s[r]_\pi \simeq t]^\alpha \vee \mathcal{C} \vee \mathcal{D} \vee [s|_\pi \simeq l]^{\mathrm{ff}}}\ (\text{Para})^\dagger$$

$$\frac{\mathcal{C} \vee [s_\tau \simeq t_\tau]^\alpha \vee [u_\tau \simeq v_\tau]^\alpha}{\mathcal{C} \vee [s_\tau \simeq t_\tau]^\alpha \vee [s_\tau \simeq u_\tau]^{\mathrm{ff}} \vee [t_\tau \simeq v_\tau]^{\mathrm{ff}}}\ (\text{Fac})$$

$$\frac{\mathcal{C} \vee [H_\tau\ \overline{s^i_{\tau^i}}]^\alpha \qquad G \in \mathcal{GB}^{\{\neg,\vee\} \cup \{\Pi^\nu, =^\nu \mid \nu \in \mathcal{T}\}}_\tau}{\mathcal{C} \vee [H_\tau\ \overline{s^i_{\tau^i}}]^\alpha \vee [H \simeq G]^{\mathrm{ff}}}\ (\text{Prim})$$

†: if $s|_\pi$ is of type ν and $\mathrm{fv}(s|_\pi) \subseteq \mathrm{fv}(s)$

$$\frac{[l^1 \simeq r^1]^{\alpha_1} \vee \cdots \vee [l^n \simeq r^n]^{\alpha_n}}{[\mathsf{simp}(l^1) \simeq \mathsf{simp}(r^1)]^{\alpha_1} \vee \cdots \vee [\mathsf{simp}(l^n) \simeq \mathsf{simp}(r^n)]^{\alpha_n}}\ (\text{Simp})$$

$s \vee s$	$\longrightarrow s$		$s \wedge s$	$\longrightarrow s$	
$\neg s \vee s$	$\longrightarrow \top$		$\neg s \wedge s$	$\longrightarrow \bot$	
$s \vee \top$	$\longrightarrow \top$		$s \wedge \top$	$\longrightarrow s$	
$s \vee \bot$	$\longrightarrow s$		$s \wedge \bot$	$\longrightarrow \bot$	
$t = t$	$\longrightarrow \top$		$t \neq t$	$\longrightarrow \bot$	
$s = \top$	$\longrightarrow s$		$s = \bot$	$\longrightarrow \neg s$	
$\forall X_\tau.\, s$	$\longrightarrow s$	if $X \notin \mathrm{fv}(s)$	$\exists X_\tau.\, s$	$\longrightarrow s$	if $X \notin \mathrm{fv}(s)$
$\neg\bot$	$\longrightarrow \top$		$\neg\top$	$\longrightarrow \bot$	
			$\neg\neg s$	$\longrightarrow s$	

$$\frac{\mathcal{C} \vee [s \simeq s]^{\mathrm{tt}}}{}\ (\text{TD1}) \qquad \frac{\mathcal{C} \vee [s \simeq t]^{\mathrm{tt}} \vee [s \simeq t]^{\mathrm{ff}}}{} \qquad \frac{\mathcal{C} \vee \mathcal{C}' \qquad \mathcal{D}}{}\ (\text{CS})$$

$$\frac{\mathcal{C}' \vee [s[E\ t]]^\alpha}{[t\ X]^{\mathrm{ff}} \vee [t\ (\varepsilon\ t)]^{\mathrm{tt}}}\ (\text{ACI}) \qquad \frac{\mathcal{C} \vee [s[\forall X_\tau.\, u] \simeq t]^\alpha \qquad v \in \mathrm{Heu}^\tau}{\mathcal{C} \vee [s[u\{v/X\}] \simeq t]^\alpha}\ (\text{HeuInst})$$

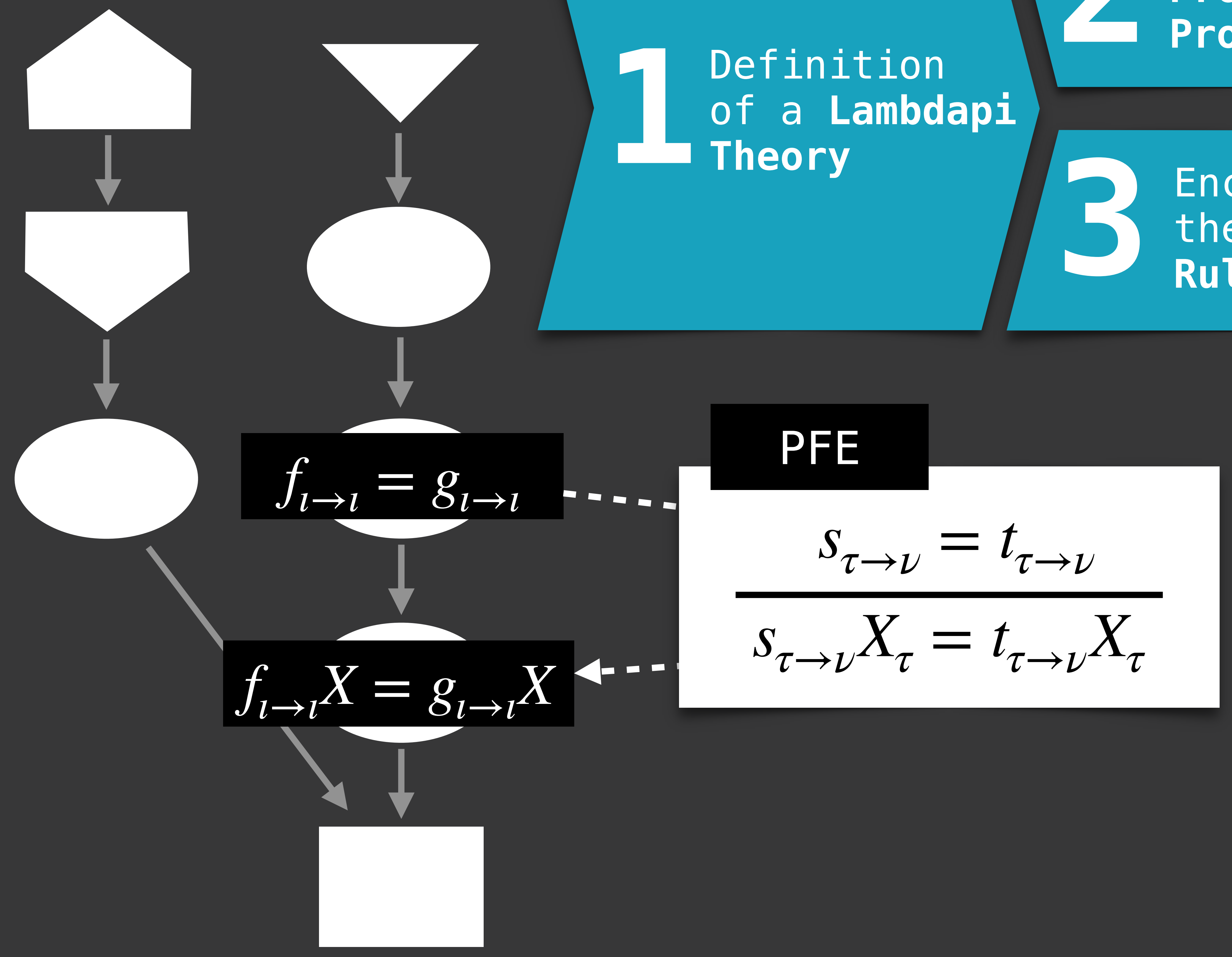
$$\frac{\mathcal{C} \vee [s \simeq t]^\alpha \vee [s \simeq t]^\alpha}{\mathcal{C} \vee [s \simeq t]^\alpha}\ (\text{DD}) \qquad \frac{\mathcal{C} \vee [s \simeq t]^{\mathrm{tt}} \qquad [l \simeq r]^{\mathrm{ff}}}{\mathcal{C}}\ (\text{NSR})$$

$$\frac{\mathcal{C} \vee [P\ s]^{\mathrm{ff}} \vee [P\ t]^{\mathrm{tt}}}{\mathcal{C}\{\lambda X.\, s = X/P\} \vee [s \simeq t]^{\mathrm{tt}}}\ (\text{LEQ}) \qquad \frac{\mathcal{C} \vee [P\ s\ s]^{\mathrm{ff}}}{\mathcal{C}\{\lambda X. \lambda Y.\, X = Y/P\}}\ (\text{AEQ})$$

$$\frac{\mathcal{C} \vee [s \simeq t]^{\mathrm{ff}} \qquad [l \simeq r]^{\mathrm{tt}}}{\mathcal{C}}\ (\text{PSR}) \qquad \frac{\mathcal{C} \vee [s \simeq t]^\alpha \qquad [l \simeq r]^{\mathrm{tt}}}{\mathcal{C} \vee [s[r\sigma]_p \simeq t]^\alpha}\ (\text{RW})$$

Found Proof

Leo-III



1 Definition of a **Lambdapi Theory**

2 Encoding of **Problems and Proof** Steps

3 Encoding of the **Calculus Rules**

4 Verification of **generated Proofs**

Encoded Proof

Lambdapi

```
require open StdLib.… ;
…

symbol axiom_i : … ;
…

symbol proof: conjecture ≔
begin
    …

    have step_m: …
      {…};

    have step_n: …
      {…};

      …

end;
```