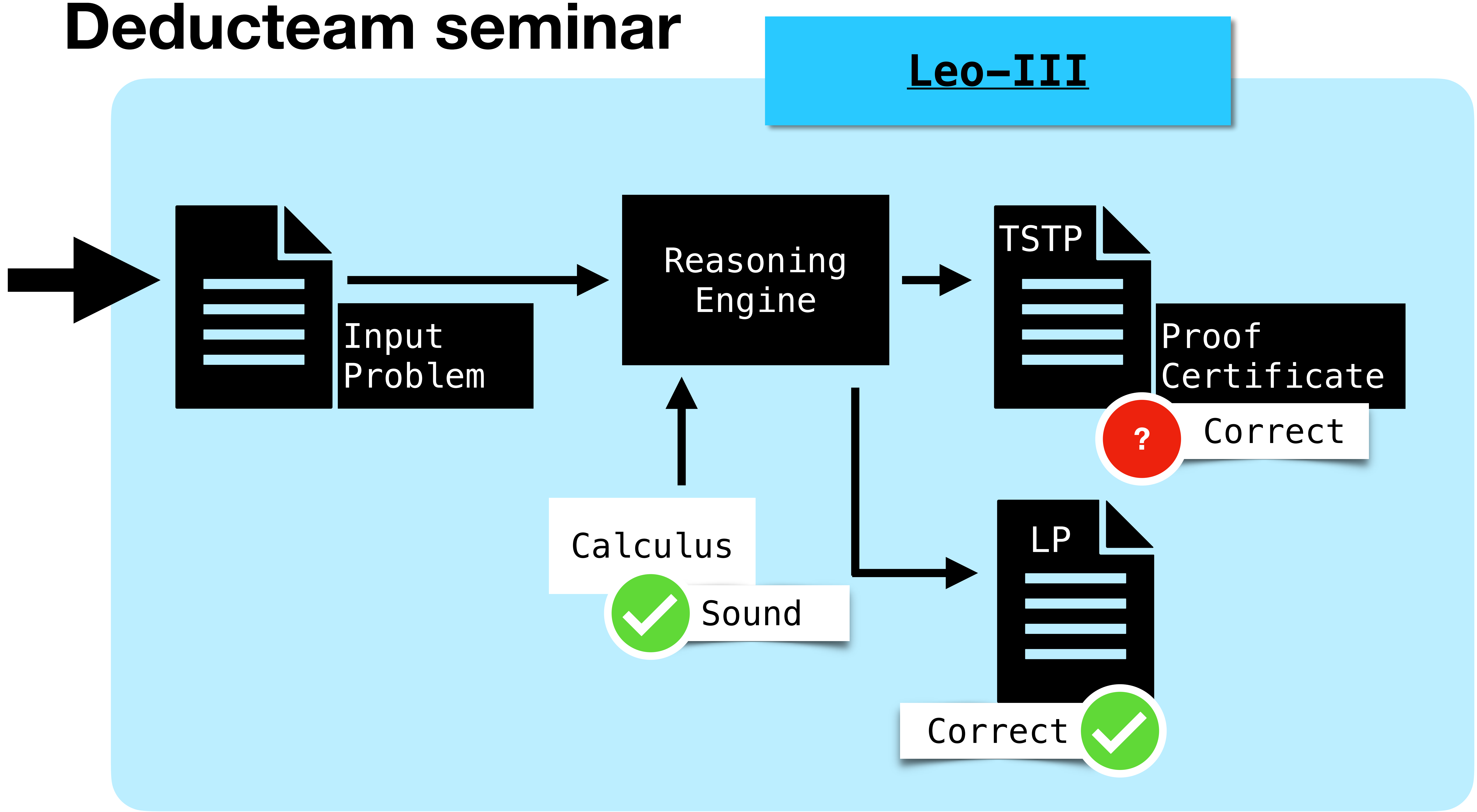
Certification of LEO-III Proofs

Deducteam seminar

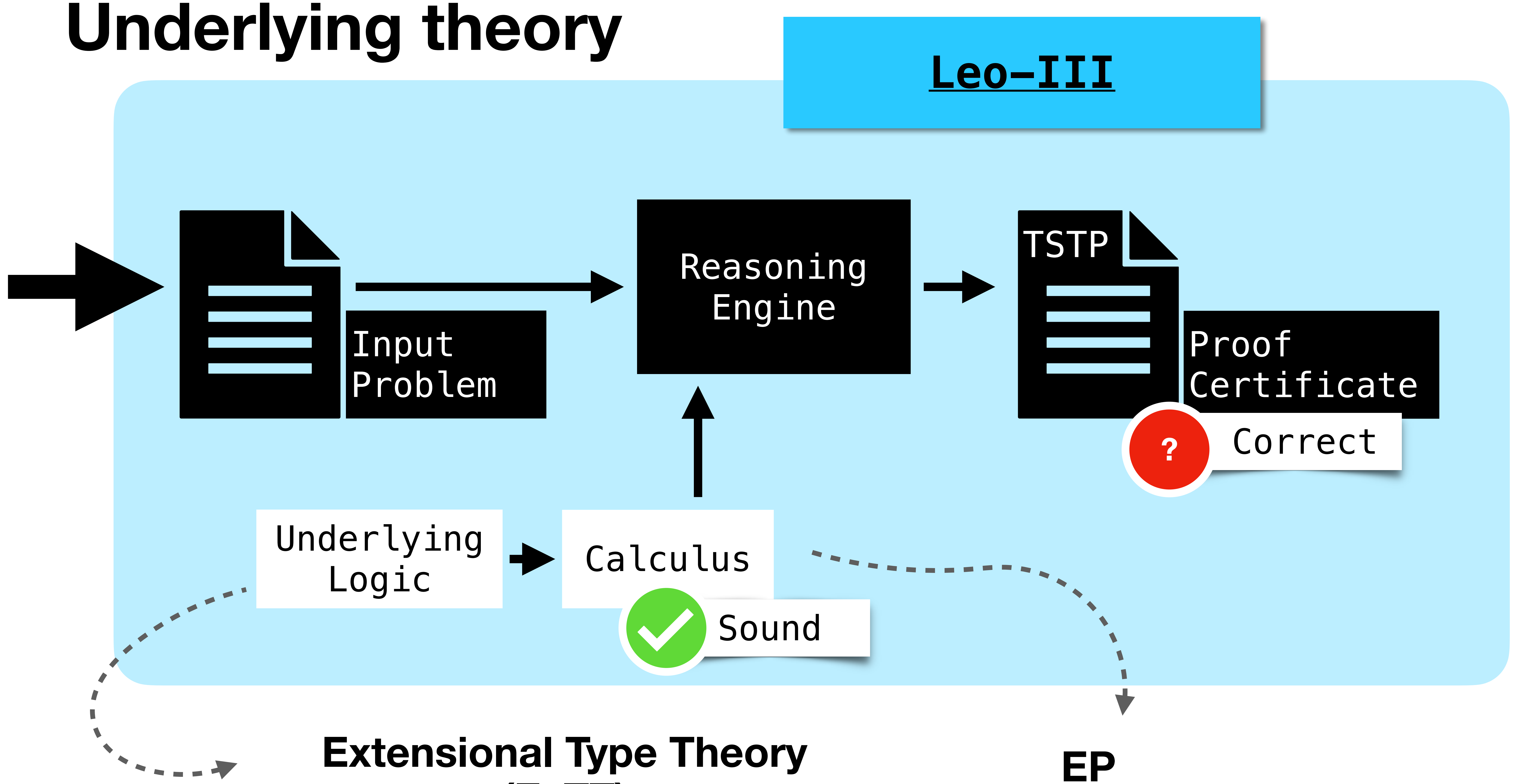


1. Leo-III
2. Proof Checking using Lambdapi (LP)
3. Definition of a LP-Theory
4. Encoding of the Calculus
5. Conclusion and Outlook

Melanie Taprogge, 26.09.2024

Leo-III

Underlying theory



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

EP
(a calculus for Extensional Higher-Order Paramodulation)

Leo-III

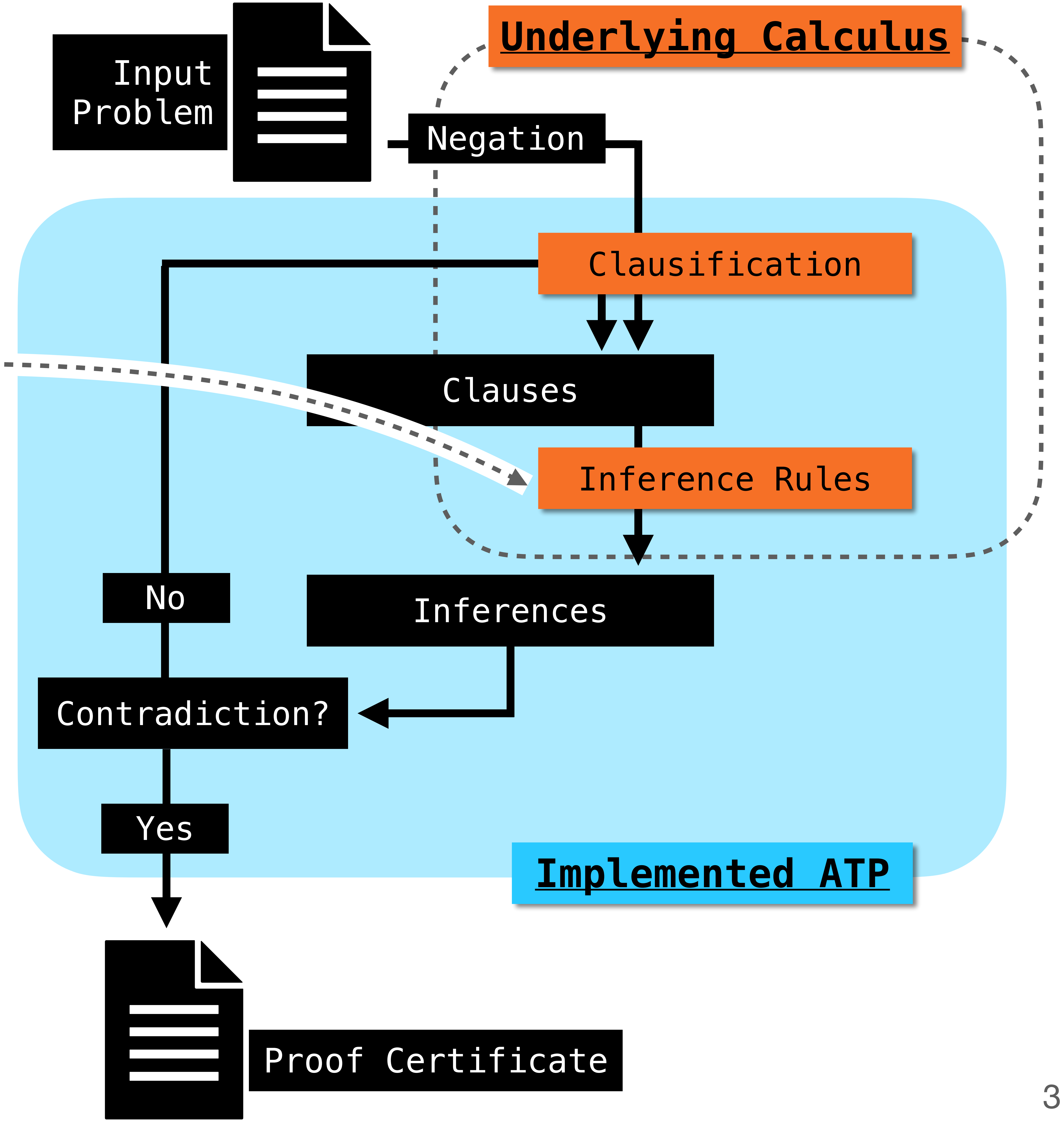
HOL-ATP Workflow

Inference Rules

e.g. functional extensionality:

$$\frac{C \vee [s_{\tau\to\nu} \simeq t_{\tau\to\nu}]^{tt}}{C \vee [s_{\tau\to\nu} X_\tau \simeq t_{\tau\to\nu} X_\tau]^{tt}} \; (FunExtPos)^\dagger$$

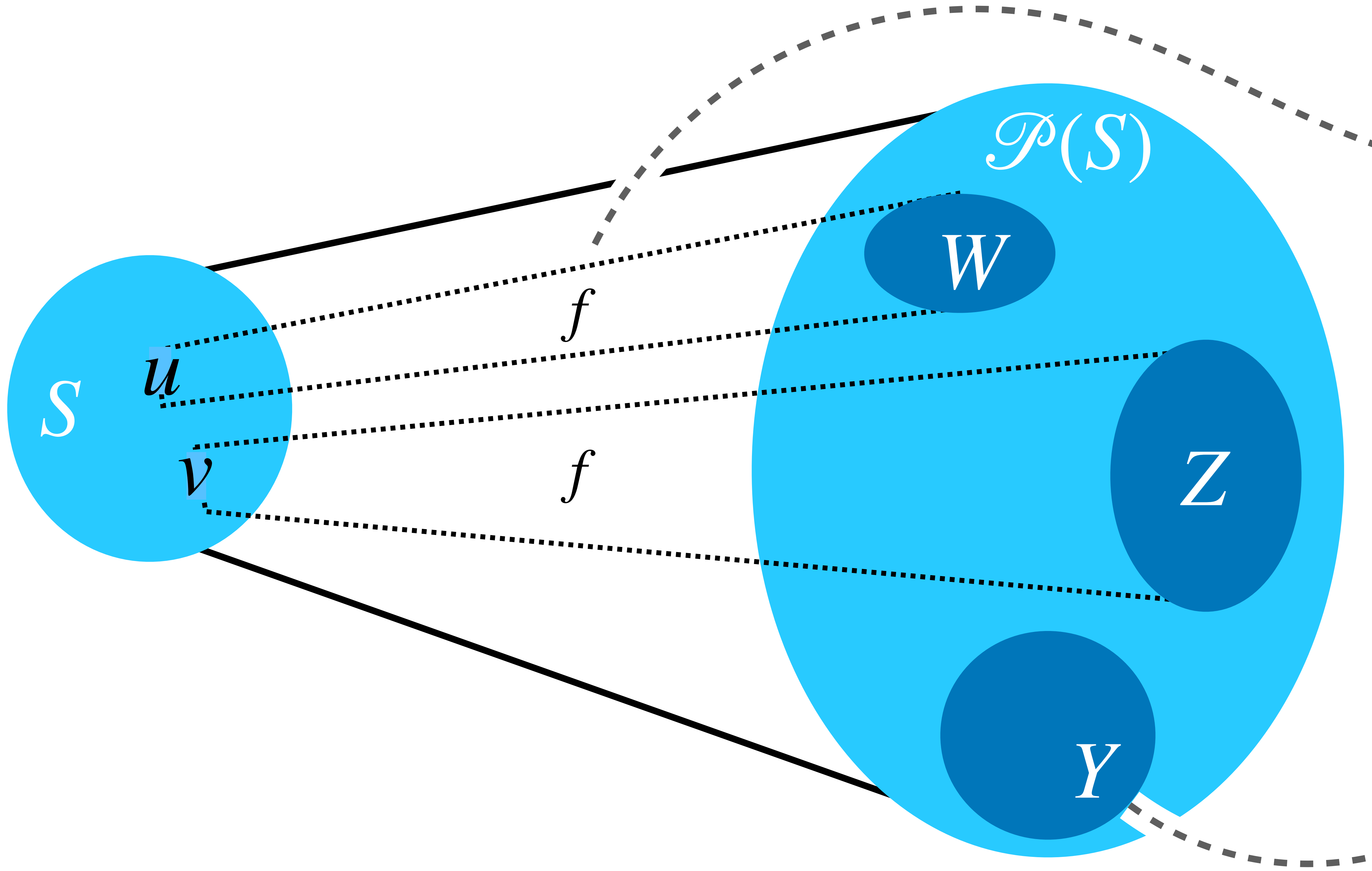
† : *where X is a fresh variable*



Leo-III

Example: Cantor's Theorem

There is no surjective function f form a set S to its powerset $\mathscr{P}(S)$ [Cantor 1932]



$$\neg \exists f_{\iota\to(\iota\to o)} \,.\, \forall y_{\iota\to o} \,.\, \exists x_\iota \,.\, f\,x = y$$

Representation of sets:

$$y(x) = \begin{cases} \textit{true} & \textit{if } x \in Y \\ \textit{false} & \textit{else} \end{cases}$$

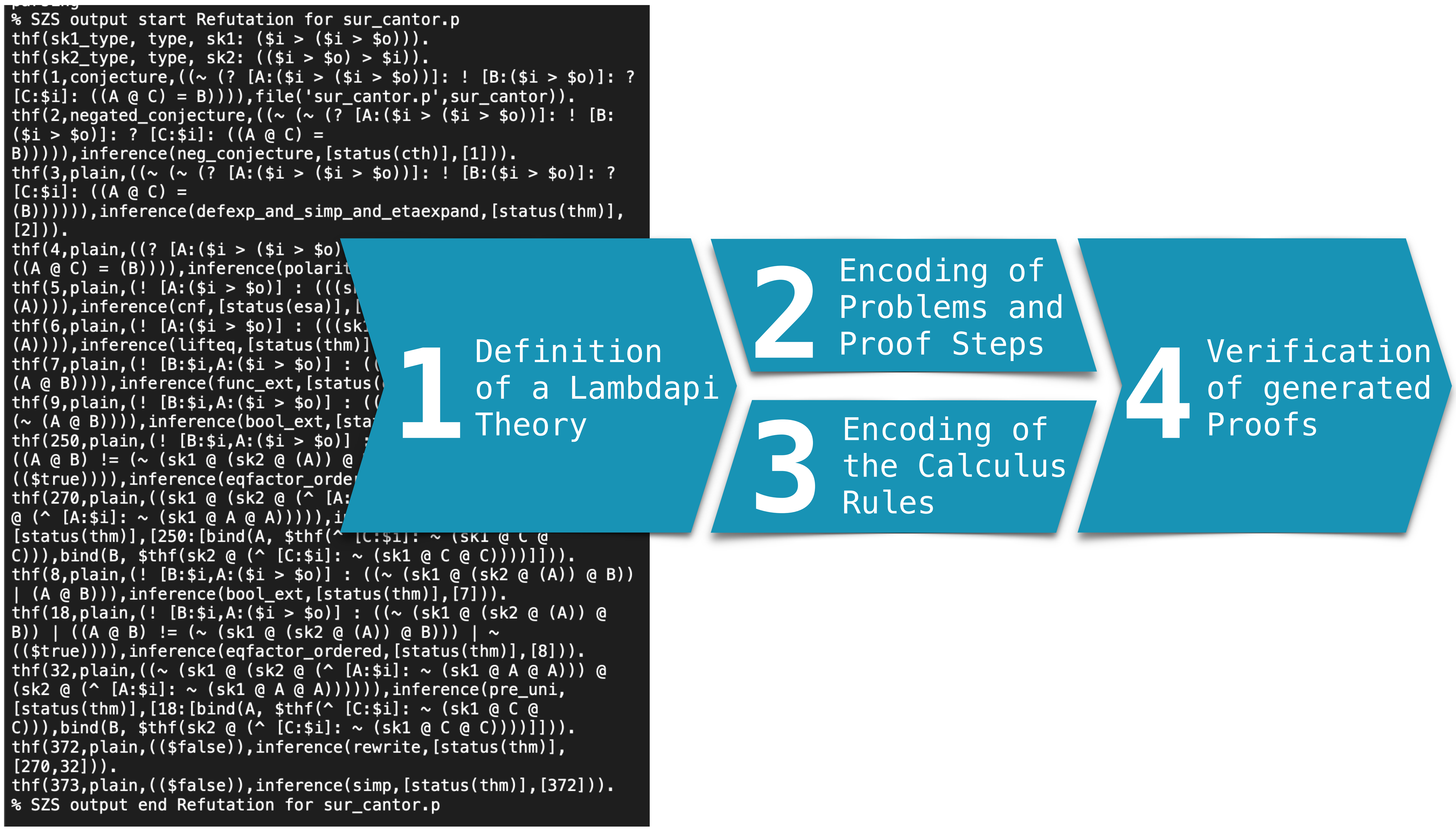
TPTP Encoding

```
thf(sur_cantor, conjecture,
  (~ ( ? [F: $i > ($i > $o)] : (
        ! [Y: $i > $o] :
        ? [X: $i] : (
        (F @ X) = Y))))).
```

Proof Checking using Lampdapi

- Goal: Encode proofs in a way that allows us to check their correctness
- The Dedukti framework implements the λΠ-modulo-Theory [Cousineau and Dowek 2007] and enables an encoding of proofs following the propositions as types principle [Curry 1934, Howard 1980]
    - Dependant types $\Pi x : T \,.\, S$ parameterise types with terms
    - Rewrite rules $l \hookrightarrow r$ replace occurrences of l with the term r
- Proof checking is reduced to type checking
- Lambdapi offers interactive proof scripts and a user-friendly syntax

Proof Checking using Lampdapi



Definition of a LP-Theory

Encoding ExTT

```
symbol Prop : TYPE;
symbol ⇒ : Prop → Prop → Prop;
symbol Prf : Prop → TYPE;
...
```

$$\neg\neg\exists f_{\iota\to(\iota\to o)} \,.\, \forall y_{\iota\to o} \,.\, \exists x_\iota \,.\, f\,x = y$$

```
symbol negatedConjecture:
  Prf (¬ ¬  ∃(λ (f : El(ι ⤳ (ι ⤳ o))),
   ∀(λ (y : El(ι ⤳ o)),
   ∃(λ (x : El ι),
   f x = y)))))
```

Propositions as Types

```
rule Prf ($x ⇒ $y)
     ↪ Prf $x → Prf $y;
...
```

Definition of a LP-Theory

Encoding ExTT

```
symbol Prop : TYPE;
symbol ⇒ : Prop → Prop → Prop;
symbol Prf : Prop → TYPE;
...
```

extt.lp

Propositions as Types

```
rule Prf ($x ⇒ $y)
     ↪ Prf $x → Prf $y;
...
```

rwr.lp

Sub-theory of Theory U
[Blanqui et al. 2023]

+ New symbol „=" defined as Leibniz-equality
+ Axioms for functional and propositional extensionality
+ Axiom for excluded middle

The rules of Natural Deduction can be derived

% SZS output start Refutation for sur_cantor.p
thf(sk1_type, type, sk1: ($i > ($i > $o))).
thf(sk2_type, type, sk2: (($i > $o) > $i)).
thf(1,conjecture,((~ (? [A:($i > ($i > $o))]: ! [B:($i > $o)]: ? [C:$i]: ((A @ C) = B)))),file('sur_cantor.p',sur_cantor)).
thf(2,negated_conjecture,((~ (~ (? [A:($i > ($i > $o))]: ! [B: ($i > $o)]: ? [C:$i]: ((A @ C) = B))))),inference(neg_conjecture,[status(cth)],[1])).
thf(3,plain,((~ (~ (? [A:($i > ($i > $o))]: ! [B:($i > $o)]: ? [C:$i]: ((A @ C) = (B)))))),inference(defexp_and_simp_and_etaexpand,[status(thm)],[2])).
thf(4,plain,((? [A:($i > ($i > $o)
((A @ C) = (B)))),inference(polarit
thf(5,plain,(! [A:($i > $o)] : (((s
(A)))),inference(cnf,[status(esa)],[
thf(6,plain,(! [A:($i > $o)] : (((sk
(A)))),inference(lifteq,[status(thm)]
thf(7,plain,(! [B:$i,A:($i > $o)] : ((
(A @ B)))),inference(func_ext,[status(
thf(9,plain,(! [B:$i,A:($i > $o)] : ((
(~ (A @ B)))),inference(bool_ext,[sta
thf(250,plain,(! [B:$i,A:($i > $o)] :
((A @ B) != (~ (sk1 @ (sk2 @ (A)) @
(($true)))),inference(eqfactor_orde
thf(270,plain,((sk1 @ (sk2 @ (^ [A:
@ (^ [A:$i]: ~ (sk1 @ A @ A))))),i
[status(thm)],[250:[bind(A, $thf(^ [C:$i]: ~ (sk1 @ C @ C)),bind(B, $thf(sk2 @ (^ [C:$i]: ~ (sk1 @ C @ C))))]])).
thf(8,plain,(! [B:$i,A:($i > $o)] : ((~ (sk1 @ (sk2 @ (A)) @ B)) | (A @ B))),inference(bool_ext,[status(thm)],[7])).
thf(18,plain,(! [B:$i,A:($i > $o)] : ((~ (sk1 @ (sk2 @ (A)) @ B)) | ((A @ B) != (~ (sk1 @ (sk2 @ (A)) @ B))) | ~ (($true)))),inference(eqfactor_ordered,[status(thm)],[8])).
thf(32,plain,((~ (sk1 @ (sk2 @ (^ [A:$i]: ~ (sk1 @ A @ A))) @ (sk2 @ (^ [A:$i]: ~ (sk1 @ A @ A)))))),inference(pre_uni, [status(thm)],[18:[bind(A, $thf(^ [C:$i]: ~ (sk1 @ C @ C))),bind(B, $thf(sk2 @ (^ [C:$i]: ~ (sk1 @ C @ C))))]])).
thf(372,plain,(($false)),inference(rewrite,[status(thm)], [270,32])).
thf(373,plain,(($false)),inference(simp,[status(thm)],[372])).
% SZS output end Refutation for sur_cantor.p
1 Definition of a Lambdapi Theory
2 Encoding of Problems and Proof Steps
3 Encoding of the Calculus Rules
4 Verification of generated Proofs
extt.lp
rwr.lp
encodedProblem.lp
...
symbol negatedConjecture:
Prf(¬ ¬ ∃(λ(f: El(ι ⇝ (ι ⇝ o))),
∀(λ(y: El(ι ⇝ o)),
∃(λ(x: El ι),
f x = y))))
symbol step3 : … :=
begin
…
end;
symbol step4 : … :=
begin
…
end;
...
symbol step373 : Prf ⊥ :=
begin
…
end;

Encoding of the Calculus

Functional Extensionality

$$\frac{C \vee (s_{\tau\to\nu} = t_{\tau\to\nu})}{C \vee (s_{\tau\to\nu}X_\tau = t_{\tau\to\nu}X_\tau)}\ (FunExtPos)^\dagger$$

† : *where X is a fresh variable*

Example:

How can proof `stepM` based on `stepN` ?

```
symbol stepN : Prf(f = g) ∨ c;
symbol stepM : Π x, Prf(f x = g x) ∨ c;
```

First idea: A function of type `Π x, Prf(f = g) → Prf(f x = g x)`

```
symbol PFE : Π s, Π t, Π x, Prf(s = t) → Prf(s x = t x) := …
```

`(PFE f g x) stepN` can be used to proof `stepM`

`Prf(f x = g x)`

But what happens if we have multiple literals?

Encoding of the Calculus

Functional Extensionality

$$\frac{C \lor (s_{\tau\to\nu} = t_{\tau\to\nu})}{C \lor (s_{\tau\to\nu}X_\tau = t_{\tau\to\nu}X_\tau)} \ (FunExtPos)^\dagger$$

† : *where X is a fresh variable*

Example:

How can proof `stepM` based on `stepN` ?

```
symbol stepN : Prf(f = g) ∨ c;
symbol stepM : Π x, Prf(f x = g x) ∨ c;
```

Second idea: A term of type `Prf((f = g) = (f x = g x))`

Lambdapi can use proofs of equalities to perform a rewrite-like operation [Coltellacci et al. 2023]

```
symbol PFE : Π s, Π t, Π x, Prf((s = t) = (s x = t x));
```

`(PFE f g x)` can be used to rewrite `stepN`

Encoding of the Calculus

Functional Extensionality

$$\frac{C \vee (s_{\tau \to \nu} = t_{\tau \to \nu})}{C \vee (s_{\tau \to \nu} X_\tau = t_{\tau \to \nu} X_\tau)} \; (FunExtPos)^\dagger$$

† : *where X is a fresh variable*

Example:

How can proof `stepM` based on `stepN` ?

```
symbol stepN : Prf(f = g) ∨ c;
symbol stepM : Π x, Prf(f x = g x) ∨ c;
```

Second idea: A term of type `Prf((f = g) = (f x = g x))`

Lambdapi can use proofs of equalities to perform a rewrite-like operation [Coltellacci et al. 2023]

```
symbol PFE : Π s, Π t, Π x, Prf((s = t) = (s x = t x)) :=
begin
…
end;
```

`rules.lp`

Encoding of the Calculus

Summary

Structure operated on		
clause	literal	term
encoding as a function	encoding as an equality -> use of the rewrite tactic	

Encoding the Calculus

Implicit Transformations

$$\frac{C \vee (s_{\tau\to\nu} = t_{\tau\to\nu})}{C \vee (s_{\tau\to\nu} X_\tau = t_{\tau\to\nu} X_\tau)} \; (FunExtPos)^\dagger$$

$\dagger$: *where X is a fresh variable*

Example: What would we receive when applying Leo-III to a clause $(f_{\tau\to\nu} = g_{\tau\to\nu}) \vee l$?

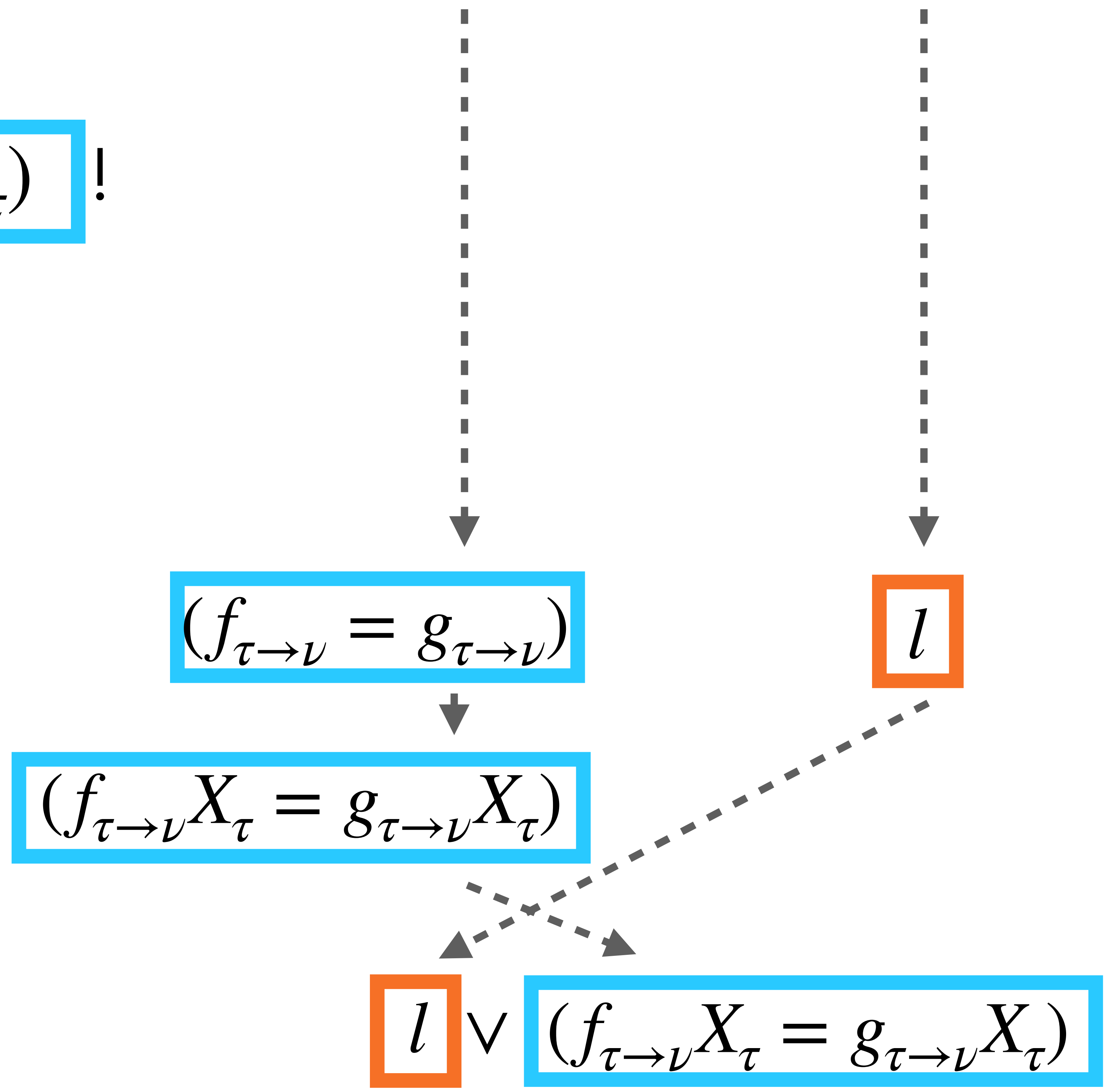
We would expect $(f_{\tau\to\nu} X_\tau = g_{\tau\to\nu} X_\tau) \vee l$.

But actually, Leo-III derives $l \vee (f_{\tau\to\nu} X_\tau = g_{\tau\to\nu} X_\tau)$!

Why does this happen?

(Simplified) implementation of *FunExtPos* in Leo-III:

1. Divide literals to those to wich *FunExtPos* can be applied and the rest
2. Apply *FunExtPos*
3. Form a new clause



Encoding the Calculus

Implicit Transformations

$$\frac{C \lor (s_{\tau\to\nu} = t_{\tau\to\nu})}{C \lor (s_{\tau\to\nu}X_\tau = t_{\tau\to\nu}X_\tau)}\ (FunExtPos)^\dagger$$

$\dagger$: *where X is a fresh variable*

Example: What would we receive when applying Leo-III to a clause $(f_{\tau\to\nu} = g_{\tau\to\nu}) \lor l$?

We would expect $(f_{\tau\to\nu}X_\tau = g_{\tau\to\nu}X_\tau) \lor C$.

But actually, Leo-III derives $C \lor (f_{\tau\to\nu}X_\tau = g_{\tau\to\nu}X_\tau)$!

Why is this relevant for our encoding?

Based on a clause such as `symbol stepN : Prf((f = g) ∨ l);`

We need to proof `symbol stepM : Π x, Prf(l ∨ (f x = g x));`

rather than `symbol stepM : Π x, Prf((f x = g x) ∨ l);`

➡ We need to verify two things:

- The permutation
- The application of the inference rule

Encoding the Calculus

Implicit Transformations: Permutation

Each rule of the calculus can perform a number of such implicit transformations.
In a verification they can be accounted for through additional steps in the verification using additional rules (called accessory rules)

In this example, we need a rule that permutes two literals:

```
symbol permute_1_0 : Π x, Π y, Prf(x ∨ y) → Prf(y ∨ x) :=
  …
```

rules.lp

Note that permute needs to mirror the structure of the clauses at hand and must thus be generated on-the-fly!

Encoding of the Calculus

Summary

Structure operated on		
clause	literal	term
encoding as a function	encoding as an equality -> use of the rewrite tactic	

Implicit Transformations
e.g. permutation of literals
generation of permutation rule in LP

Encoding of the Calculus

How many LP-terms are necessary to encode a calculus rule?

- Static: One rule is sufficient (e.g. funExt)

$$\frac{C \lor (s_{\tau\to\nu} = t_{\tau\to\nu})}{C \lor (s_{\tau\to\nu}X_\tau = t_{\tau\to\nu}X_\tau)}\ (FunExtPos)^\dagger$$

† : *where X is a fresh variable*

```
PFE : Π s, Π t, Π x,
 Prf((s = t) = (s x = t x))
```

Encoding of the Calculus

How many LP-terms are necessary to encode a calculus rule?

- Static: One rule is sufficient (e.g. funExt)
- Versatile: Multipel encodings for one rule (e.g. EqFact)

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

```
EqFact_p [T] x y z v:
  ((Prf ((x = y) ∨ (z = v))) →
  (Prf ((x = y) ∨ (¬(x = z)) ∨
  (¬ (y = v)))))
EqFact_n [T] x y z v:
  ((Prf ((¬(x = y)) ∨ (¬(z = v)))) →
  (Prf ((¬(x = y)) ∨ (¬(x = z)) ∨
  (¬ (y = v)))))
```

Encoding of the Calculus

How many LP-terms are necessary to encode a calculus rule?

- Static: One rule is sufficient (e.g. funExt)
- Versatile: Multipel encodings for one rule (e.g. EqFact)
- Flexible: needs to be generated on the fly (e.g. permute)
- Exception: Some rules can simply be translated through the corresponding Lambdapi operation (e.g. variable binding)

Encoding of the Calculus

Summary

Structure operated on		
clause	literal	term
encoding as a function	encoding as an equality -> use of the rewrite tactic	

Implicit Transformations
e.g. permutation of literals
generation of permutation rule in LP

Adaptability		
static	versatile	flexible
encoding as a single rule	encoding of multiple rules	on the fly generation

Encoding of the Calculus

Modular Encoding, e.g. (simplified) Functional Extensionality

Categorization of (PFE) Encoding Demands

Adaptability of Rule: Static
Structure operated on: Literals
Additional Transformations: Changing the order of literals, …

React to Implicit Transformations

Modular Encoding of (PFE)

…
-If the order of the literals was changed implicitly, …
-…
-Rewrite the proof-goal with PFE
-Refine with the (permuted) parent-formula

Apply actual calculus rule

Encoding of the Calculus

Functional Extensionality

$$\frac{C \ \vee \ (s_{\tau\to\nu} = t_{\tau\to\nu})}{C \ \vee \ (s_{\tau\to\nu}X_\tau = t_{\tau\to\nu}X_\tau)} \ (FunExtPos)^{\dagger}$$

$\dagger$: *where X is a fresh variable*

Example:

```
symbol stepN : Prf((f = g) ∨ l);
```

```
symbol stepM : Π x, Prf(l ∨ (f x = g x)):=
begin
  have Permutation:  Prf(l ∨ (f = g))
    {refine permute_1_0 (f = g) l step_N};



end;
```

1. Verify the permutation

   We generate the rule…

```
symbol permute_1_0 : Π x, Π y,
Prf(x ∨ y) → Prf(y ∨ x) :=
 …
```

We can then instantiate this term to fit our example:

```
 permute_1_0 (f = g) l
```

Resulting in:

```
 Prf((f = g) ∨ l) → Prf(l ∨ (f = g))
```

Encoding of the Calculus

Functional Extensionality

$$\frac{C \lor (s_{\tau\to\nu} = t_{\tau\to\nu})}{C \lor (s_{\tau\to\nu} X_\tau = t_{\tau\to\nu} X_\tau)} \ (FunExtPos)^{\dagger}$$

† : *where X is a fresh variable*

Example:

```
symbol stepN : Prf((f = g) ∨ l);
```

```
symbol stepM : Π x, Prf(l ∨ (f x = g x)):=
begin
  have Permutation:  Prf(l ∨ (f = g))
    {refine permute_1_0 (f = g) l step_N};
  assume x;
  have funExt: Prf(l ∨ (f x = g x))
    {rewrite .[x in _ ∨ x] (PFE f g);
     refine Permutation};
  refine funExt
end;
```

2. Verify the PFE application

We encode the rule as an equality …

```
symbol PFE : Π s, Π t, Π x,
    Prf((s x = t x) = (s = t)):=
…
```

We can thus instantiate this term to fit our example:

```
(PFE f g)
```

has type

```
Π x, Prf((f x = g x) = (f = g))
```

Expressing proofs in Lambdapi

Conclusion and Outlook

The current state of the encoding

Unification rules $\mathcal{UNI}$

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

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

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

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

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

Extensionality rules $\mathcal{EXT}$

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

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

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

Clausification rules $\mathcal{CNF}$

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

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

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

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

Primary inference rules $\mathcal{PI}$

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

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

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

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

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

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

$$\frac{\mathcal{C}' \vee [s[\mathcal{E}\ t]]^\alpha}{[t\ X]^{ff} \vee [t\ (\varepsilon\ t)]^{tt}}\ (\text{ACI}) \qquad \frac{\mathcal{C} \vee [s \simeq t]^\alpha \vee [s \simeq t]^\alpha}{\mathcal{C} \vee [s \simeq t]^\alpha}\ (\text{DD}) \qquad \frac{}{[P\ X]^{ff} \vee [P(f\ P)]^{tt}}\ (\text{ACD})$$

$$\frac{\mathcal{C} \vee [s[\forall X_\tau . u] \simeq t]^\alpha \qquad v \in \text{Heu}^\tau}{\mathcal{C} \vee [s[u\{v/X\}] \simeq t]^\alpha}\ (\text{HeuInst}) \qquad \frac{\mathcal{C} \vee [s \simeq t]^{ff} \qquad [l \simeq r]^{tt}}{\mathcal{C}}\ (\text{PSR})$$

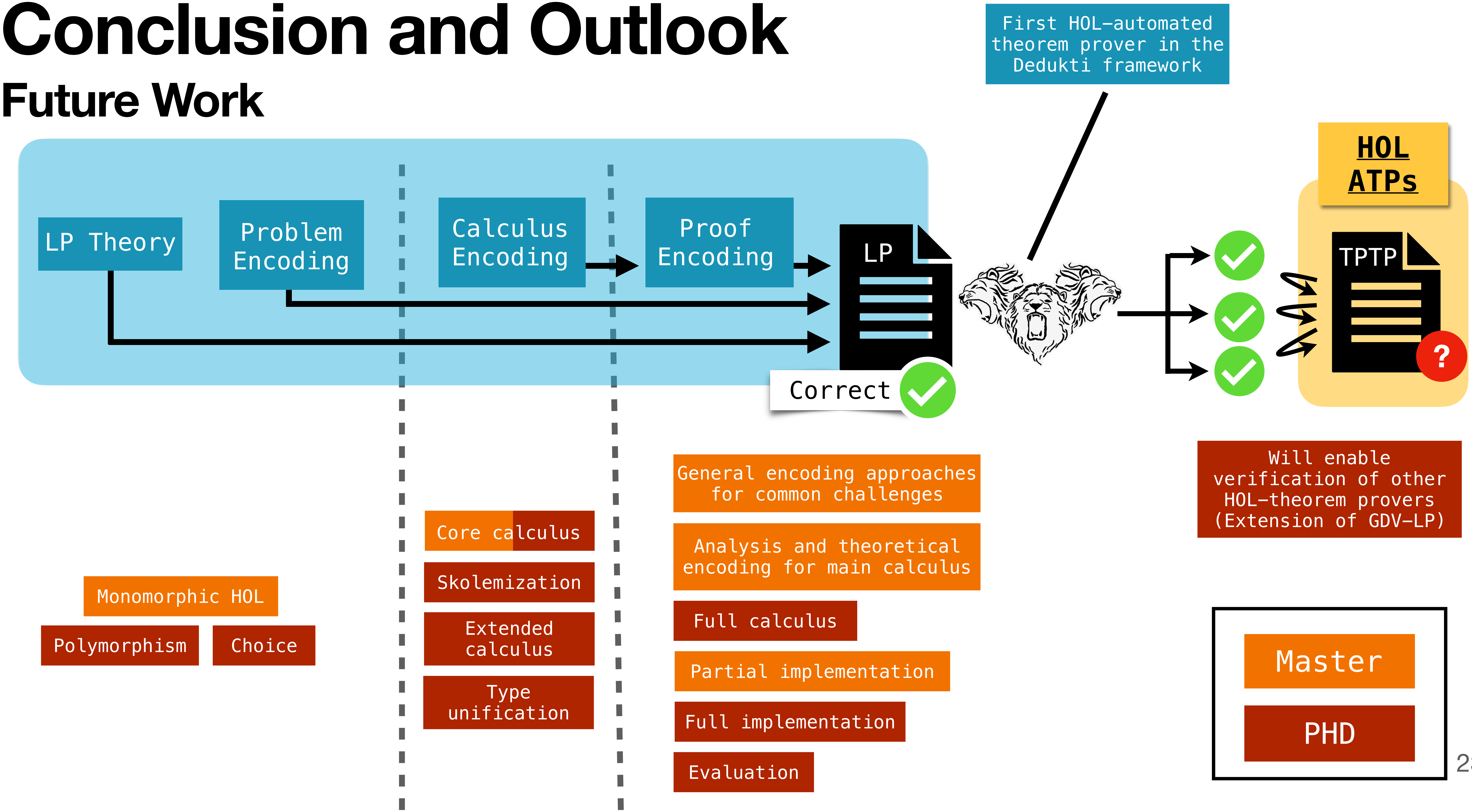
$$\frac{\mathcal{C} \vee [s \simeq t]^{tt} \qquad [l \simeq r]^{ff}}{\mathcal{C}}\ (\text{NSR}) \qquad \frac{\mathcal{C} \vee \mathcal{C}' \qquad \mathcal{D}}{}\ (\text{CS}) \qquad \frac{\mathcal{C} \vee [s \simeq t]^\alpha \qquad [l \simeq r]^{tt}}{\mathcal{C} \vee [s[r\sigma]_p \simeq t]^\alpha}\ (\text{RW})$$

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

$$\frac{\mathcal{C} \vee [s \simeq s]^{tt}}{}\ (\text{TD1}) \qquad \frac{\mathcal{C} \vee [s \simeq t]^{tt} \vee [s \simeq t]^{ff}}{}\ (\text{TD2})$$

Conclusion and Outlook

Future Work



References

- Blanqui, F., Dowek, G., Grienenberger, E., Hondet, G., & Thiré, F. (2023). A modular construction of type theories. *Logical Methods in Computer Science*, *19*.
- Cantor, G. Über eine elementare frage der mannigfaltigkeitslehre, jahresbericht der dmv (vol. 1, pp. 75–78). references to cantor (1932)
- Coltellacci A., Merz S., and Dowek G., “Reconstruction of smt proofs with lambdapi,” in Proceed- ings of the 21st International Workshop on Satisfiability Modulo Theories (SMT 2024), Montreal, Canada, July 22-23, 2024
- 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.
- 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.
- 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.
- Wadler, P. (2015). Propositions as types. *Communications of the ACM*, *58*(12), 75-84.
- Moschovakis, J. Intuitionistic Logic, *The Stanford Encyclopedia of Philosophy (Summer 2024 Edition)*, Edward N. Zalta & Uri Nodelman (eds.), URL = <https://plato.stanford.edu/archives/sum2024/entries/logic-intuitionistic/>.
- Steen, A. (2020). Extensional paramodulation for higher-order logic and its effective implementation Leo-III. *KI-Künstliche Intelligenz*, *34*(1), 105-108.