NuITP: An Inductive Theorem Prover for Maude

F. Durán[1], S. Escobar[3], J. Meseguer[2], J. Sapiña[3]
[1]Universidad de Málaga
[2]University of Illinois at Urbana-Champaign
[3]Universitat Politècnica de València

2nd Workshop on Logic, Algebra and Category Theory: LAC 2025
Fukuoka, September 29 – October 3, 2025

Outline

1. Introduction
2. Examples
3. Gilbreath’s card trick
4. Inference rules
5. Conclusions

Introduction

- Inductive theorem proving for equational programs has two problems:
  - Expressiveness. Types and subtypes, conditional equations, and rewriting modulo associativity and/or commutativity and/or identity axioms.
  - Scalability. The theorem prover should scale up to large proofs. Tactics, auxiliary lemmas, automatic reasoning.
- Maude has been endowed with new symbolic equational reasoning techniques during the last 15 years that tackle expressiveness but also scalability.
  - Equality predicates, order-sorted conditional narrowing, variant narrowing, variant unification, variant satisfiability, and order-sorted congruence closure.
- NuITP is a next-generation inductive theorem prover based on inductive order-sorted first-order logic. Theoretical foundations in [Meseguer-JLAMP2025].

Features

- Equational Theories in Maude $(\Omega, B_\Omega, \emptyset) \subseteq (\Sigma_1, B_1, E_1) \subseteq (\Sigma, B, E)$
- B any combination of associativity (A), commutativity (C) and identity (U)
- E a set of convergent conditional equations
- Constructor subtheory $(\Omega, B_\Omega, \emptyset)$ and Finite Variant subtheory (Σ_1, B_1, E_1)
- Formulas $(w_1 = w'_1 \wedge \ldots \wedge w_n = w'_n) \longrightarrow$

$$(u_1^1 = v_1^1 \vee \ldots \vee u_{m_1}^1 = v_{m_1}^1) \wedge \ldots \wedge (u_1^k = v_1^k \vee \ldots \vee u_{m_k}^k = v_{m_k}^k)$$

- Reduction path ordering (RPO) given by user via annotations
- Generator sets over constructors
- Proof tactics given by user
- Internalization of previously proved auxiliary lemmas
- Automatic Equality Predicate Simplification

Peano

```
fmod PEANO+ADD-NO-ORDER is
  sorts Nat NzNat .
  subsorts NzNat < Nat .

  op 0 : -> Nat  .
  op s_ : Nat -> NzNat .

  op _+_ : Nat Nat -> Nat [ assoc comm ] .
  eq N:Nat + 0 = N:Nat .
  eq N:Nat + s M:Nat = s(N:Nat + M:Nat) .
endfm
```

Peano

```
fmod PEANO+ADD-NO-ORDER is
  sorts Nat NzNat .
  subsorts NzNat < Nat .

  op 0 : -> Nat [ ctor ] .
  op s_ : Nat -> NzNat [ ctor ] .

  op _+_ : Nat Nat -> Nat [ assoc comm ] .
  eq N:Nat + 0 = N:Nat .
  eq N:Nat + s M:Nat = s(N:Nat + M:Nat) .
endfm
```

Peano

```
fmod PEANO+ADD-WITH-ORDER is
  sorts Nat NzNat .
  subsorts NzNat < Nat .

  op 0 : -> Nat [ ctor metadata "1" ] .
  op s_ : Nat -> NzNat [ ctor metadata "2" ] .

  op _+_ : Nat Nat -> Nat [ assoc comm metadata "3" ] .
  eq N:Nat + 0 = N:Nat .
  eq N:Nat + s M:Nat = s(N:Nat + M:Nat) .
endfm
```

Peano: associativity of addition

```
NuITP> set goal X:Nat + (Y:Nat + Z:Nat) = (X:Nat + Y:Nat) + Z:Nat .

  Initial goal set.

  Goal Id: 0
  Skolem Ops:
    None
  Executable Hypotheses:
    None
  Non-Executable Hypotheses:
    None
  Goal:
    ($1:Nat + ($2:Nat + $3:Nat)) = (($1:Nat + $2:Nat) + $3:Nat)
```

Peano: associativity of addition

```
NuITP> apply gsi to 0 on $3 with 0 ;; s(K:Nat) .

  Generator Set Induction (GSI) applied to goal 0.

  Goal Id: 0.1
  Skolem Ops:
    None
  Executable Hypotheses:
    None
  Non-Executable Hypotheses:
    None
  Goal:
    ($1:Nat + ($2:Nat + 0)) = (($1:Nat + $2:Nat) + 0)

  Goal Id: 0.2
  Skolem Ops:
    $4.Nat
  Executable Hypotheses:
    (($1:Nat + $2:Nat) + $4) => ($1:Nat +($2:Nat + $4))
  Non-Executable Hypotheses:
    None
  Goal:
    ($1:Nat + ($2:Nat + s $4)) = (($1:Nat + $2:Nat) + s $4)
```

Peano: associativity of addition

```
NuITP> apply eps to 0.1 .

  Equality Predicate Simplification (EPS) applied to goal 0.1.

  Goal 0.1.1 has been proved.

  Unproved goals:

  Goal Id: 0.2
  Skolem Ops:
    $4.Nat
  Executable Hypotheses:
    (($1:Nat + $2:Nat) + $4) => ($1:Nat + ($2:Nat + $4))
  Non-Executable Hypotheses:
    None
  Goal:
    ($1:Nat + ($2:Nat + s $4)) = (($1:Nat + $2:Nat) + s $4)

  Total unproved goals: 1
```

Peano: associativity of addition

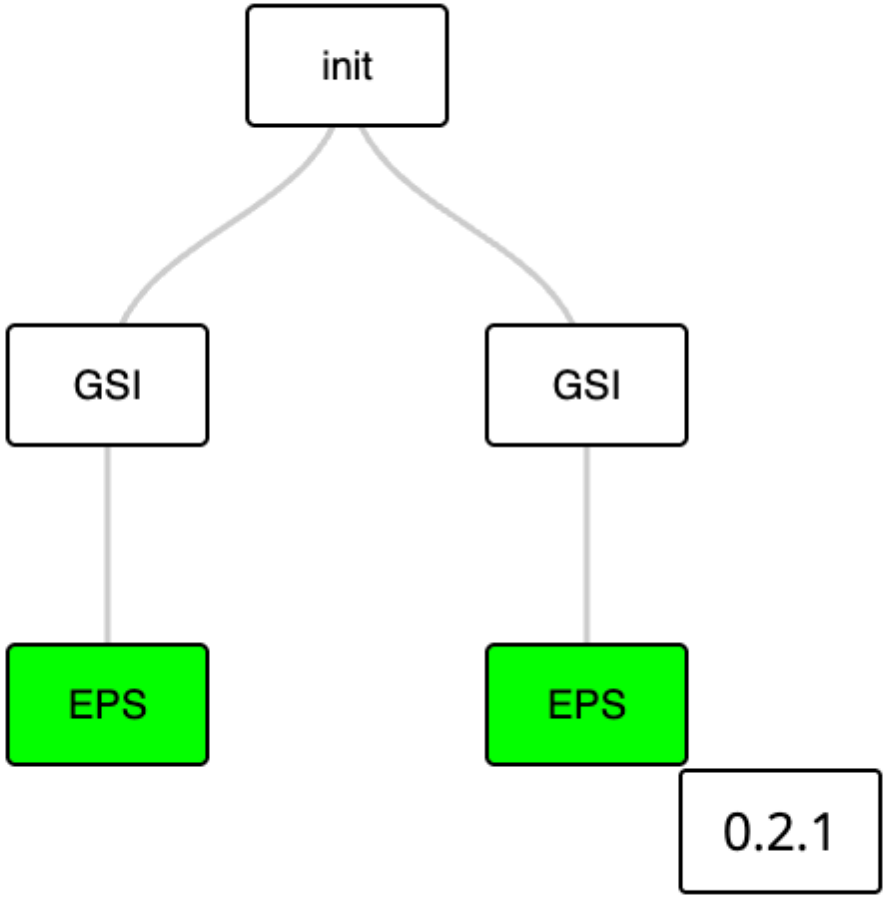
```
NuITP> apply eps to 0.2 .

  Equality Predicate Simplification (EPS) applied to goal 0.2.

  Goal 0.2.1 has been proved.

  qed
```

Peano: associativity of addition



Peano: commutativity of addition, a new proof

```
NuITP> apply gsi! to 0 on $3 with 0 ;; s(K:Nat) .

  Generator Set Induction with Equality Predicate Simplification (GSI!)
  applied to goal 0.

  Goals 0.1.1 and 0.2.1 have been proved.

  qed
```

Peano: commutativity of addition

```
NuITP> set goal (X:Nat + Y:Nat = Y:Nat + X:Nat) .

  Initial goal set.

  Goal Id: 0
  Skolem Ops:
    None
  Executable Hypotheses:
    None
  Non-Executable Hypotheses:
    None
  Goal:
    ($1:Nat + $2:Nat) =($2:Nat + $1:Nat)
```

Peano: commutativity of addition

```
NuITP> apply gsi! to 0 on $1 with 0 ;; s K:Nat .

  Generator Set Induction with Equality Predicate Simplification (GSI!)
  applied to goal 0.

  Goal Id: 0.1.1
  Skolem Ops:
    None
  Executable Hypotheses:
    None
  Non-Executable Hypotheses:
    None
  Goal:
    $2:Nat =(0 + $2:Nat)

  Goal Id: 0.2.1
  Skolem Ops:
    $3.Nat
  Executable Hypotheses:
    None
  Non-Executable Hypotheses:
    ($3 + $2:Nat) = ($2:Nat + $3)
  Goal:
    s($2:Nat + $3) = (s $3 + $2:Nat)
```

Peano: commutativity of addition

```
NuITP> apply gsi! to 0.1.1 on $2 with 0 ;; s K:Nat .

  Generator Set Induction with Equality Predicate Simplification (GSI!)
  applied to goal 0.1.1.

  Goals 0.1.1.1.1 and 0.1.1.2.1 have been proved.

  Unproved goals:

  Goal Id: 0.2.1
  Skolem Ops:
    $3.Nat
  Executable Hypotheses:
    None
  Non-Executable Hypotheses:
    ($3 + $2:Nat) =($2:Nat + $3)
  Goal:
    s($2:Nat + $3) =(s $3 + $2:Nat)

  Total unproved goals: 1
```

Peano: commutativity of addition

```
NuITP> apply gsi! to 0.2.1 on $2 with 0 ;; s K:Nat .

  Generator Set Induction with Equality Predicate Simplification (GSI!)
  applied to goal 0.2.1.

  Goals 0.2.1.1.1 and 0.2.1.2.1 have been proved.

  qed
```

Commutativity and associativity of addition in one shot

```
NuITP> genset GEN-NAT for Nat is 0 ;; s N:Nat .

  Generator set GEN-NAT for sort Nat added.

  GEN-NAT (default):
    0
    s N:Nat
```

Commutativity and associativity of addition in one shot

```
NuITP> set goal (X:Nat + Y:Nat = Y:Nat + X:Nat) /\ X1:Nat + (Y1:Nat + Z:Nat) = (X1:Nat + Y1:Nat) + Z:Nat .

  Initial goal set.

  Goal Id: 0
  Generated By: init
  Skolem Ops:
    None
  Executable Hypotheses:
    None
  Non-Executable Hypotheses:
    None
  Goal:
    (($2:Nat + $4:Nat) = ($4:Nat + $2:Nat)) /\
    ($1:Nat + ($3:Nat + $5:Nat)) = (($1:Nat + $3:Nat) + $5:Nat)
```

Commutativity and associativity of addition in one shot

```
NuITP> apply gsi! to 0 .

  Generator Set Induction (GSI) with Equality Predicate Simplification applied to goal 0.

  Goals 0.1.1, 0.10.1, 0.11.1, 0.12.1, 0.13.1, 0.14.1, 0.15.1, 0.16.1, 0.17.1, 0.18.1,
        0.19.1, 0.2.1, 0.20.1, 0.21.1, 0.22.1, 0.23.1, 0.24.1, 0.27.1, 0.28.1, 0.29.1,
        0.3.1, 0.30.1, 0.31.1, 0.32.1, 0.5.1, 0.6.1, 0.7.1 and 0.8.1 have been
    proved.

  qed
```

Lists: associativity of concatenation

```
fmod LIST-APPEND is
  sorts Nat List .

  op 0 : -> Nat [ ctor metadata "1" ] .
  op s : Nat -> Nat [ ctor metadata "2" ] .

  op nil : -> List [ ctor metadata "3" ] .
  op _;_ : Nat List -> List [ ctor metadata "4" ] .

  op _@_ : List List -> List [ metadata "5" ] .
  eq nil @ L:List = L:List .
  eq (N:Nat ; L:List) @ Q:List = N:Nat ; (L:List @ Q:List) .
endfm
```

```
NuITP> set goal (L:List @ P:List) @ Q:List = L:List @ (P:List @ Q:List) .
```

```
NuITP> apply gsi! to 0 on $1 with nil ;; (m:Nat ; R:List) .

  Generator Set Induction with Equality Predicate Simplification (GSI!) applied to
  goal 0.

  Goals 0.1.1 and 0.2.1 have been proved.

  qed
```

Lists: reverse of non-empty lists

```
NuITP> genset GLIST for List is X:Elt L:List .

  Generator set GLIST for sort List added.

  GLIST (default):
    X:Elt L:List

NuITP> set goal rev(Q:List Y:Elt) = Y:Elt rev(Q:List) .


  Initial goal set.

  Goal Id: 0
  Generated By: init
  Skolem Ops:
    None
  Executable Hypotheses:
    None
  Non-Executable Hypotheses:
    None
  Goal:
    rev($1:List $2:Elt) = $2:Elt rev($1:List)

NuITP> apply gsi! to 0 on $1 .

  Generator Set Induction (GSI) with Equality Predicate Simplification applied to goal 0.

  Goal 0.1.1 has been proved.

  qed
```

```
fmod REVERSING-LISTS is
  sorts Elt List .
  subsort Elt < List .

  op __ : List List -> List [ ctor assoc metadata "1" ] .

  op rev : List -> List [ metadata "2" ] .
  eq rev(X:Elt) = X:Elt .
  eq rev(X:Elt L:List) = rev(L:List) X:Elt .
endfm
```

Lists: reverse of non-empty lists with ni

```
fmod REVERSING-LISTS is
  sorts Elt List .
  subsort Elt < List .

  op __ : List List -> List [ ctor assoc metadata "1" ] .

  op rev : List -> List [ metadata "2" ] .
  eq rev(X:Elt) = X:Elt .
  eq rev(X:Elt L:List) = rev(L:List) X:Elt .
endfm
```

```
NuITP> set goal rev(Q:List Y:Elt) = Y:Elt rev(Q:List) .
```

```
NuITP> apply ni! to 0 on rev($1:List $2:Elt) .

  Narrowing Induction (NI) with Equality Predicate Simplification applied to goal 0.

  Goals 0.1.1 and 0.2.1 have been proved.

  qed
```

Lists: reverse of non-empty lists with ni

```
fmod REVERSING-LISTS is
  sorts Elt List .
  subsort Elt < List .

  op __ : List List -> List [ ctor assoc metadata "1" ] .

  op rev : List -> List [ metadata "2" ] .
  eq rev(X:Elt) = X:Elt .
  eq rev(X:Elt L:List) = rev(L:List) X:Elt .
endfm
```

```
NuITP> show goal 0.1 .

  Goal Id: 0.1
  Generated By: NI
  Skolem Ops:
    $3.Elt
    $4.Elt
    $5.List
  Executable Hypotheses:
    rev($5 $4) => $4 rev($5)
  Non-Executable Hypotheses:
    None
  Goal:
    ($4 rev($3 $5)) = rev($5 $4) $3

NuITP> show goal 0.2 .

  Goal Id: 0.2
  Generated By: NI
  Skolem Ops:
    None
  Executable Hypotheses:
    None
  Non-Executable Hypotheses:
    None
  Goal:
    ($4:Elt rev($3:Elt)) = rev($4:Elt) $3:Elt
```

Gilbreath’s card trick (1/5)

- Norman L. Gilbreath’s principle
- Given an initial deck of cards with some adequate properties, after a random shuffle the resulting deck will preserve some of those properties.
- In the Gilbreath’s card trick, given an initial deck of cards with alternating colors (e.g., red and black), after shuffling it once, if we deal the resulting deck in pairs, each pair will always contain one card of each color.

F. Durán, S. Escobar, J. Meseguer, J. Sapiña: NuITP: An Inductive Theorem Prover for Equational Program Verification. PPDP 2024: 6:1-11

Gilbreath's card trick (2/5)

1. Begin with an even deck of cards so that they are sorted alternating colors (red and black);
2. Split the deck in two, not necessarily equal piles;
3. If the bottom cards of each pile are equal, take one of the cards and move (rotate) it to the top of that pile;
4. Riffle both piles (the shuffle does not need to be perfect); and
5. Deal the resulting deck in pairs. All pairs will always have a card of each color (e.g., either red-black or black-red cards).

Gilbreath's card trick (3/5)

```
op paired : Card Card -> Boolean [metadata "6"] .
eq paired(red, black) = True [variant] .
eq paired(black, red) = True [variant] .
eq paired(C, C) = False [variant] .

op opposite : List List -> Boolean [metadata "7"] .
eq opposite(nil, L) = False [variant] .
eq opposite(L, nil) = False [variant] .
eq opposite(C1 L1, C2 L2) = paired(C1, C2) [variant] .

op alter : List -> Boolean [metadata "8"] .
eq alter(nil) = True .
eq alter(C) = True .
ceq alter(C1 C2 L) = alter(C2 L) if paired(C1, C2) = True .
ceq alter(C1 C2 L) = False if paired(C1, C2) = False .

op pairedList : List -> Boolean [metadata "9"] .
eq pairedList(nil) = True .
eq pairedList(C) = False .
eq pairedList(C C L) = False .
ceq pairedList(C1 C2 L) = pairedList(L) if paired(C1, C2) = True .
```

```
op even : List -> Boolean [metadata "11"] .
eq even(nil) = True .
eq even(C) = False .
eq even(L1 C1 L2 C2 L3) = even(L1 L2 L3) .

op rotate : List -> List [metadata "13"] .
eq rotate(nil) = nil [variant] .
eq rotate(C L) = L C [variant] .
```

Gilbreath's card trick (4/5)

```
eq shuffle(nil, nil, nil) = True .
eq shuffle(nil, nil, C3 L3) = False .
eq shuffle(C1 L1, L2, nil) = False .
ceq shuffle(C1 L1, nil, C3 L3) = False if paired(C1, C3) = True .
ceq shuffle(C1 L1, nil, C3 L3) = shuffle(L1, nil, L3)
  if paired(C1, C3) = False .
eq shuffle(L1, C2 L2, nil) = False .
ceq shuffle(nil, C2 L2, C3 L3) = False if paired(C2, C3) = True .
ceq shuffle(nil, C2 L2, C3 L3) = shuffle(nil, L2, L3)
  if paired(C2, C3) = False .
```

```
ceq shuffle(C1 L1, C2 L2, C3 L3) = True
  if paired(C1, C3) = False
  /\ shuffle(L1, C2 L2, L3) = True .
ceq shuffle(C1 L1, C2 L2, C3 L3) = True
  if paired(C2, C3) = False
  /\ shuffle(C1 L1, L2, L3) = True .
ceq shuffle(C1 L1, C2 L2, C3 L3) = False
  if paired(C1, C3) = True
  /\ paired(C2, C3) = True .
ceq shuffle(C1 L1, C2 L2, C3 L3) = False
  if shuffle(L1, C2 L2, L3) = False
  /\ shuffle(C1 L1, L2, L3) = False .
ceq shuffle(C1 L1, C2 L2, C3 L3) = False
  if paired(C1, C3) = True
  /\ shuffle(C1 L1, L2, L3) = False .
ceq shuffle(C1 L1, C2 L2, C3 L3) = False
  if paired(C2, C3) = True
  /\ shuffle(L1, C2 L2, L3) = False .
```

Gilbreath's card trick (5/5)

```
> set goal ((alter(L1:List L2:List) = True)
             /\ (even((L1:List L2:List)) = True)
             /\ (opposite(L1:List, L2:List) = True)
             /\ (shuffle(L1:List, L2:List, L3:List) = True))
            -> (pairedList(L3:List) = True) .
```

```
> set goal ((alter(L1:List L2:List) = True)
             /\ (even(L1:List L2:List) = True)
             /\ (opposite(L1:List, L2:List) = False)
             /\ (shuffle(L1:List, L2:List, L3:List) = True))
            -> (pairedList(rotate(L3:List)) = True) .
```

	User Cmnds.		**R. Applied**		**G. Generated**	
Rule	Goal 1	Goal 2	Goal 1	Goal 2	Goal 1	Goal 2
CAS	100	119	118	145	472	580
CS			70	81	70	81
CUT	9	14	9	14	18	28
CVUL			13	23	32	48
EPS			773	958	773	958
GND			24	23	48	46
LSB			112	104	112	104
NI	12	15	12	15	84	120
NS	18	23	18	23	256	288
RST			6	5	6	5
UFREE	1	1	1	1	8	8
Total	140	172	1156	1392	1879	2266

NuITP and its Inference Rules

- Simplification rules
  - Equality Predicate Simplification (EPS),
  - Constructor Variant Unification Left (CVUL),
  - Constructor Variant Unification Failure Right (CVUFR),
  - Substitution Left and Right (SUBL, SUBR),
  - Narrowing Simplification (NS),
  - Clause Subsumption (CS),
  - Equation Rewriting (EQ),
  - Inductive Congruence Closure (ICC), and
  - Variant Satisfiability (VARSAT).
- Inductive rules
  - Generator Set Induction (GSI),
  - Narrowing Induction (NI),
  - Lemma Enrichment (LE),
  - Split (SP),
  - Case (CAS),
  - Variable Abstraction (VA), and
  - Cut.

The inference rules of this system transform inductive goals of the form

$$[\overline{X}, \mathcal{E}, H] \Vdash \Gamma \rightarrow \Lambda$$

Generator Set Induction (GSI)

The **GSI** rule generalizes standard structural induction on constructors.

$$\frac{\{[\overline{X} \uplus \overline{Y}_u, \mathcal{E}, H \& H_u] \Vdash (\Gamma \to \bigwedge_{j \in J} \Delta_j)\{z \mapsto \overline{u}^{\bullet}\}\}_{u \in G}}{[\overline{X}, \mathcal{E}, H] \Vdash \Gamma \to \bigwedge_{j \in J} \Delta_j}$$

where $z \in vars(\Gamma \to \bigwedge_{j \in J} \Delta_j)$ has sort s, $\{u_1, \ldots, u_n\}$ is a B_0-generator set for s, with B_0 the non-unit axioms of the theory $\vec{\mathcal{E}}$,

Y_i are the variables of the u_i, $Y_i = vars(u_i)$, for $1 \leqslant i \leqslant n$, and

the induction hypotheses H_i are $H_i = \{(\Gamma \to \Delta_j)\{z \to \overline{v}\} \mid v \in PST_{B_0, \leqslant s}(u_i) \wedge j \in J\}$

where $PST_{B_0, \leqslant s}(u)$ are the proper B_0-subterms of u.

Generator Set Induction (GSI)

$$\frac{\{[\overline{X} \uplus \overline{Y}_{\vec{u}}, \mathcal{E}, H \& H_{\vec{u}}] \Vdash (\Gamma \rightarrow \bigwedge_{j \in J} \Delta_j)\{\vec{z} \mapsto \overline{\vec{u}}^{\bullet}\}\}_{\vec{u} \in G_1 \times \ldots \times G_n}}{[\overline{X}, \mathcal{E}, H] \Vdash \Gamma \rightarrow \bigwedge_{j \in J} \Delta_j}$$

where, (i) $\vec{z}$ (resp. $\vec{s}$) denotes the tuple $(z_1, \ldots, z_n)$ (resp. $(s_1, \ldots, s_n)$),

(ii) for $\vec{u} = (u_1, \ldots, u_n)$, $\{\vec{z} \mapsto \overline{\vec{u}}^{\bullet}\}$ denotes the substitution $\{z_1 \mapsto \overline{u_1}^{\bullet}, \ldots, z_n \mapsto \overline{u_n}^{\bullet}\}$,

(iii) $G_1 \times \ldots \times G_n$ is the cartesian product of B_0-generator sets G_i for sorts s_i, $1 \leq i \leq n$, all having *fresh* variables, and such that for each i, j, $1 \leq i < j \leq n$, $vars(G_i) \cap vars(G_j) = \emptyset$, and

(iv) $Y_{\vec{u}} = vars(\vec{u})$.

Case (CAS)

$$\frac{\left\{ [\overline{X}, \mathcal{E}, H] \Vdash (\Gamma \to \Lambda)\{z \mapsto u_i\} \right\}_{1 \leq i \leq n}}{[\overline{X}, \mathcal{E}, H] \Vdash \Gamma \to \Lambda}$$

Narrowing simplification (NS)

The **NS** rule performs one step of symbolic evaluation on a term $f(\vec{v})$ of an equality of the form $f(\vec{v})=u$ appearing anywhere in a goal $\Gamma \rightarrow \Lambda$.

(1) f is a non-constructor function symbol, so that its defining equations are sufficiently complete,

(2) the argument subterms $\vec{v}$ are constructor terms,

(3) term u belongs to the FVP subtheory $\mathcal{E}_1$ of the module's theory $\mathcal{E}$, and

(4) the equations defining f, oriented as rewrite rules, have the form $\{[i] : f(\vec{u_i}) \rightarrow r_i \text{ if } \Gamma_i\}_{i \in I}$, where the $\vec{u_i}$ are construtor terms and the r_i are terms in $\mathcal{E}_1$.

$$\frac{\{[\overline{X} \uplus \overline{Y}_{i,j}, \mathcal{E}, H \,\&\, \widehat{H'}] \Vdash (\Gamma_i, (\Gamma \rightarrow \Lambda)[r_i = u]_p)\overline{\alpha}_{i,j}\}_{i \in I_0}^{j \in J_i}}{[\overline{X}, \mathcal{E}, H] \Vdash (\Gamma \rightarrow \Lambda)[f(\vec{v}) = u]_p}$$

where $I_0 \subseteq I$ is the subset of rule labels of the rules defining f such that there is at least one B-unifier of $f(\vec{v})$ and $f(\vec{u_i})$, and

$\{\alpha_{i,j}\}_{j \in J_i}$ denotes a complete set of B-unifiers of the equation $f(\vec{v}) = f(\vec{u_i})$.

Narrowing induction (NI)

$$\frac{\{[\overline{X} \uplus \overline{Y}_{i,j}, \mathcal{E},\ H \& H_{i,j}] \Vdash (\Gamma_i, (\Gamma \to \bigwedge_{l \in L} \Delta_l)[r_i]_p)\overline{\alpha}_{i,j}\}_{i \in I_0}^{j \in J_i}}{[\overline{X}, \mathcal{E}, H] \Vdash (\Gamma \to \bigwedge_{l \in L} \Delta_l)[f(\vec{v})]_p}$$

Constructor Variant Unification Left (CVUL)

The **CVUL** rule unifies those equations in the condition that belong to $\mathcal{E}_1$.

$$\frac{\{[\overline{X} \uplus \overline{Y}_{\alpha},\ \mathcal{E},\ H \& \widehat{H'}] \Vdash (\Gamma' \to \Lambda)\overline{\alpha}\}_{\alpha \in Unif^{\Omega}_{\mathcal{E}_1}(\Gamma^{\circ})}}{[\overline{X}, \mathcal{E}, H] \Vdash \Gamma, \Gamma' \to \Lambda}$$

where Γ is a conjunction of $\mathcal{E}_1$-equalities,
Γ' does not contain any $\mathcal{E}_1$-equalities, and
$Unif^{\Omega}_{\mathcal{E}_1}(\Gamma)$ denotes the set of constructor $\mathcal{E}_1$-unifiers of Γ

Constructor Variant Unification Failure Right (CVUFR)

The **CVUFR** rule may be seen as a restricted version of the more general **CVUL** rule.

$$\frac{[\overline{X}, \mathcal{E}, H] \Vdash \Gamma \rightarrow \Lambda \wedge \Delta}{[\overline{X}, \mathcal{E}, H] \Vdash \Gamma \rightarrow \Lambda \wedge (u = v, \Delta)}$$

where $u = v$ is a $\mathcal{E}_{1_{\overline{X}}}$-equality and $Unif^{\Omega}_{\mathcal{E}_1}((u = v)^{\circ}) = \varnothing$.

Equality Simplification (EPS) and Proof Search (ES)

$$\frac{\{[\overline{X}, \mathcal{E}, H] \Vdash \Gamma'_i \to \Lambda'_i\}_{i \in I}}{[\overline{X}, \mathcal{E}, H] \Vdash \Gamma \to \Lambda}$$

$$\frac{[\overline{X}, \mathcal{E}, H] \Vdash (\Gamma \to \Lambda)[\top]_{\vec{p}}}{[\overline{X}, \mathcal{E}, H] \Vdash (\Gamma \to \Lambda)[u = v]^{B}_{\vec{p}}} \quad \text{if } E \cup eq(H) \cup B \vdash_{[L_1, L_2], \phi, mod, k} u = v$$

Lemma Enrichment (LE) and Split (SP)

$$\frac{[\overline{X}_0, \mathcal{E}, H_0] \Vdash \Gamma' \to \bigwedge_{j \in J} \Delta'_j \qquad [\overline{X}, \mathcal{E}, H] \Vdash H_0 \qquad [\overline{X}, \mathcal{E}, H \& \{\Gamma' \to \Delta'_j\}_{j \in J}] \Vdash \Gamma \to \Lambda}{[\overline{X}, \mathcal{E}, H] \Vdash \Gamma \to \Lambda}$$

$$\frac{\{[X, \mathcal{E}, H] \Vdash \Gamma_i \theta, \Gamma \to \Lambda\}_{i \in I} \qquad [X, \mathcal{E}, H] \Vdash H_0 \qquad [X_0, \mathcal{E}, H_0] \Vdash cnf(\bigvee_{i \in I} \Gamma_i)}{[\overline{X}, \mathcal{E}, H] \Vdash \Gamma \to \Lambda}$$

Cut and Variable Abstraction

$$\frac{[\overline{X}, \mathcal{E}, H] \Vdash \Gamma \to \Gamma' \qquad [\overline{X}, \mathcal{E}, H] \Vdash \Gamma, \Gamma' \to \Lambda}{[\overline{X}, \mathcal{E}, H] \Vdash \Gamma \to \Lambda}$$

$$\frac{[\overline{X}, \mathcal{E}, H] \Vdash z = u, \Gamma' \to \Lambda'}{[\overline{X}, \mathcal{E}, H] \Vdash (\Gamma \to \Lambda)[u]_p}$$

Induction Congruence Closure

The **ICC** rule is a *modus ponens* type of rule that tries to discharge a goal $[\overline{X},\mathcal{E},H] \Vdash \Gamma \rightarrow \Lambda$ by assuming its condition Γ to prove its conclusion Λ.

$$\frac{\{[\overline{X},\mathcal{E},H] \Vdash \Gamma_i^{\sharp} \rightarrow \Lambda_i^{\sharp}\}_{i \in I}}{[\overline{X},\mathcal{E},H] \Vdash \Gamma \rightarrow \Lambda}$$

where: (i) by definition, $\overline{\Lambda}_i^{\sharp} \in \overline{\Lambda}!_{\vec{\mathcal{E}}^{=}_{\overline{X \cup Y}_U} \cup \vec{H}^{+}_{e_U} \cup \vec{\overline{\Gamma}}_i^{\sharp}}$, and we always pick $\overline{\Lambda}_i^{\sharp} = \top$ if $\top \in \overline{\Lambda}!_{\vec{\mathcal{E}}^{=}_{\overline{X \cup Y}_U} \cup \vec{H}^{+}_{e_U} \cup \vec{\overline{\Gamma}}_i^{\sharp}}$;

(ii) $\Gamma_i^{\sharp} \rightarrow \Lambda_i^{\sharp}$ is obtained from $\overline{\Gamma}_i^{\sharp} \rightarrow \overline{\Lambda}_i^{\sharp}$ by converting back the Skolem constants associated to the variables of $\Gamma \rightarrow \Lambda$ into those same variables, and

(iii) the case $\overline{\Gamma}^{\sharp} = \bot$ is the case when there are no conjunctions in the disjunctive normal form $\overline{\Gamma}^{\sharp}$

Conclusions

- We have introduced the NuITP, explained its most commonly used inference rules, and illustrated their use in proving the card trick benchmark
- Main objective expressiveness & scalability

Future work

- Improve strategy language
- More expressiveness
- Improve user interface
- Parallelization
- Backend for other tools
- Proof certification