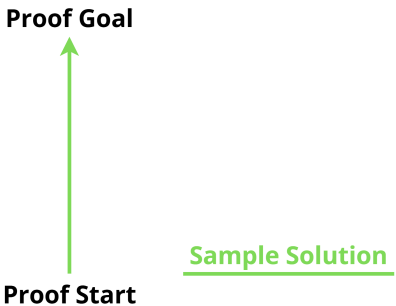
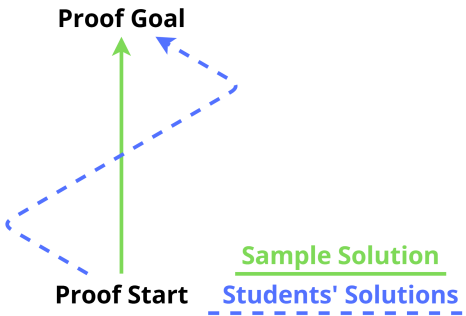
Automatic Assessment and Feedback on Undergraduates' Structural Induction Proofs

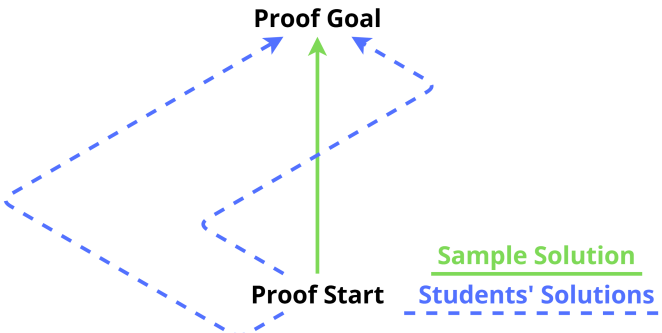
Edward Sabinus Thomas Kühn Wolf Zimmermann

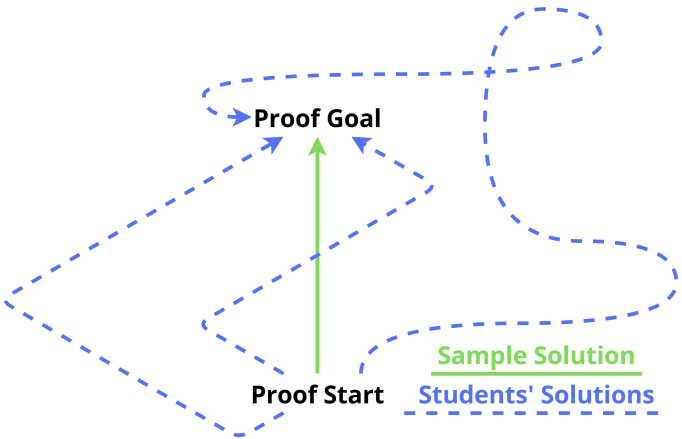
Martin-Luther-University Halle-Wittenberg
Institute for Computer Science

18. May 2026

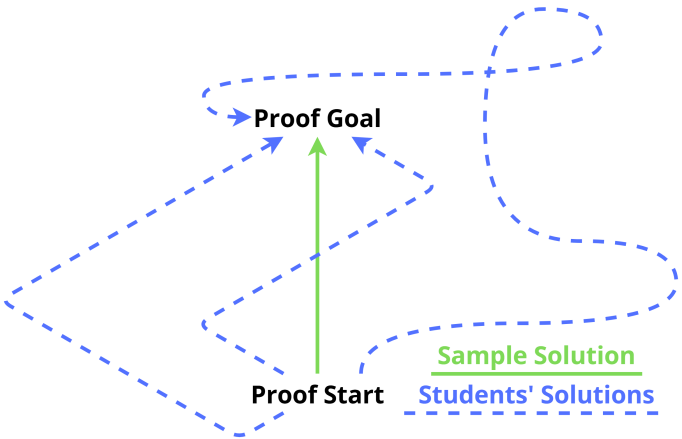

Proof Goal
Sample Solution
Proof Start


Proof Goal
Proof Start
Sample Solution
Students' Solutions


Proof Goal
Proof Start
Sample Solution
Students' Solutions

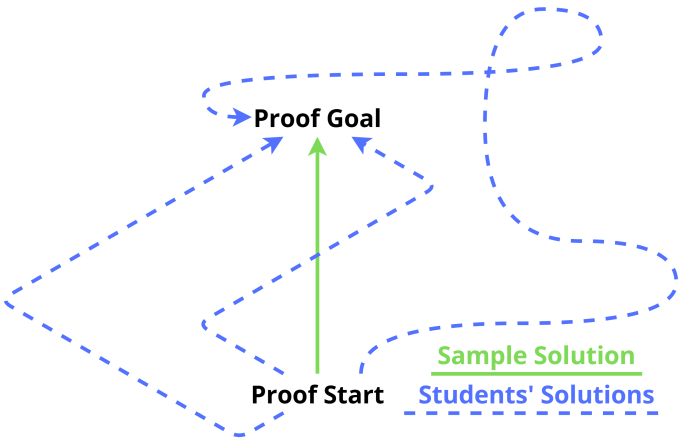

Proof Goal
Proof Start
Sample Solution
Students' Solutions

Assessment and Feedback on Undergraduates' Structural Induction Proofs



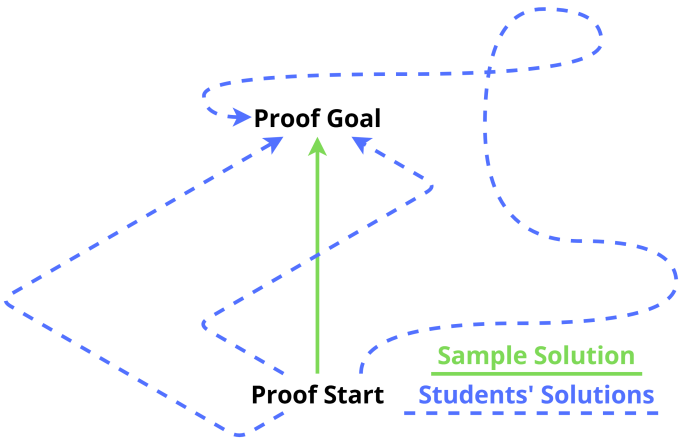
- Wide variaty of correct solutions
  $\rightarrow$ Difficult manual assessment

Assessment and Feedback on Undergraduates' Structural Induction Proofs



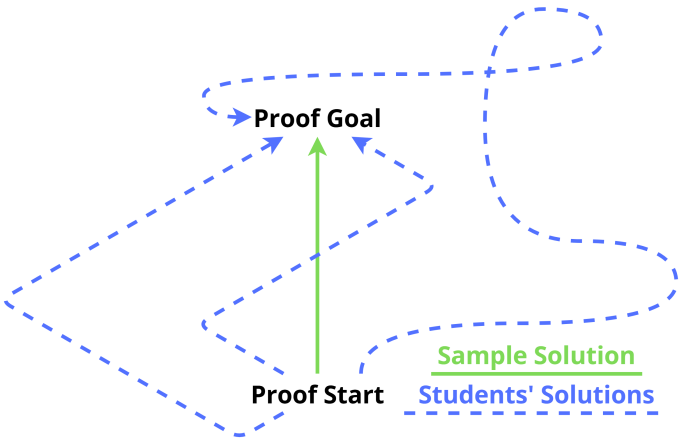
- Wide variaty of correct solutions
  $\rightarrow$ Difficult manual assessment
- Time consuming, potentially unfair, and biased assessment and feedback

Assessment and Feedback on Undergraduates' Structural Induction Proofs



- Wide variaty of correct solutions
  $\rightarrow$ Difficult manual assessment
- Time consuming, potentially unfair, and biased assessment and feedback
- Students need much training and feedback

Assessment and Feedback on Undergraduates' Structural Induction Proofs



- Wide variaty of correct solutions
  $\rightarrow$ Difficult manual assessment
- Time consuming, potentially unfair, and biased assessment and feedback
- Students need much training and feedback
- Idea: automatic proof checking and assessment

Research Questions

RQ1 How must proof checking be performed to generate comprehensible feedback for students?

Research Questions

RQ1 How must proof checking be performed to generate comprehensible feedback for students?
RQ2 How can erroneous solutions be automatically assessed, such that the assessment is correct and fair, as well as graded according to errors?

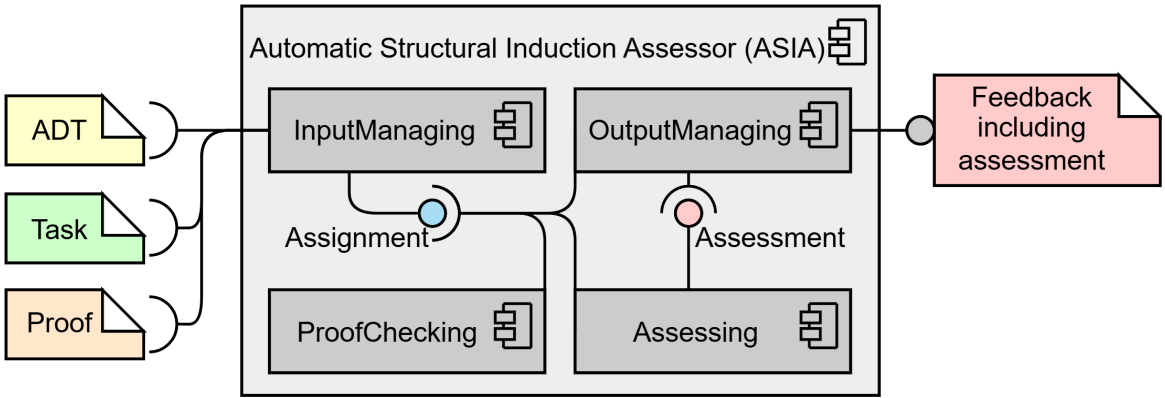
Research Questions

RQ1 How must proof checking be performed to generate comprehensible feedback for students?
RQ2 How can erroneous solutions be automatically assessed, such that the assessment is correct and fair, as well as graded according to errors?
RQ3 How suitable is automatic assessment in an undergraduate course setting?

Research Questions

RQ1 How must proof checking be performed to generate comprehensible feedback for students?
RQ2 How can erroneous solutions be automatically assessed, such that the assessment is correct and fair, as well as graded according to errors?
RQ3 How suitable is automatic assessment in an undergraduate course setting?
RQ4 How usable is the resulting tool for undergraduate students?

Architecture



Structural Induction Proofs on Abstract Data Types

```
adt Peano
sorts Nat
constructors zero: Nat
             inc: Nat -> Nat
operations add: Nat >< Nat -> Nat
vars n:Nat, m:Nat
axioms A1: add(n,zero)   = n
       A2: add(n,inc(m)) = inc(add(n,m))
```

Structural Induction Proofs on Abstract Data Types

```
adt Peano
sorts Nat
constructors zero: Nat
             inc: Nat -> Nat
operations add: Nat >< Nat -> Nat
vars n:Nat, m:Nat
axioms A1: add(n,zero)   = n
       A2: add(n,inc(m)) = inc(add(n,m))
```

```
task forall n:Nat : add(zero,n) = n induction n
  case zero maxpt 1 minsteps 1 maxsteps 1
  case inc  maxpt 2 minsteps 2 maxsteps 2
  IH        maxpt 1
```

Structural Induction Proofs on Abstract Data Types

```
adt Peano
sorts Nat
constructors zero: Nat
             inc: Nat -> Nat
operations add: Nat >< Nat -> Nat
vars n:Nat, m:Nat
axioms A1: add(n,zero)   = n
       A2: add(n,inc(m)) = inc(add(n,m))
```

```
task forall n:Nat : add(zero,n) = n induction n
  case zero maxpt 1 minsteps 1 maxsteps 1
  case inc  maxpt 2 minsteps 2 maxsteps 2
  IH        maxpt 1
```

```
proof /* of a student */
BC: show: fixed n:Nat : add(zero,inc(n))=inc(n)
add(zero,inc(n))
  {A2, lr, add(zero,inc(n)), [zero/n,n/m]}
= inc(add(zero,n))
  {A1, lr, add(zero,n), []}
= inc(n)
IH: fixed n:Nat : add(zero,n)=n
IS: show: add(zero,zero)=zero
add(zero,zero)
  {IH, lr, add(zero,zero), [zero/n]}
= zero
```

Proof Checking - Example

```
proof /* of a student */
BC: show: fixed n:Nat : add(zero,inc(n))=inc(n)
add(zero,inc(n))
  {A2, lr, add(zero,inc(n)), [zero/n,n/m]}
= inc(add(zero,n))
  {A1, lr, add(zero,n), []}
= inc(n)
IH: fixed n:Nat : add(zero,n)=n
IS: show: add(zero,zero)=zero
add(zero,zero)
  {IH, lr, add(zero,zero), [zero/n]}
= zero
```

- Wrong constructor inc for base case (BC)
- Incorrect application of Axiom A1
- Wrong constructor zero for induction step (IS)
- Induction hypothesis (IH) in base case zero
- Substitution of fixed variable n

Assessment Schema

Assessment with Error Kinds

Error Kind	**Assessment of the Context**
Formal Errors F_i	0%
Content Errors C_i	Configurable Weight α_i (0%-100%)
Warnings	100%

Assessment Schema

Assessment with Error Kinds

Error Kind	**Assessment of the Context**
Formal Errors F_i	0%
Content Errors C_i	Configurable Weight α_i (0%-100%)
Warnings	100%

Basic Concepts of Aggregation of Assessments

$$A_c = \begin{cases} 0 & , if \sum_{F_i \in F(c)} F_i > 0 \\ maxPt_c - maxPt_c \cdot \dfrac{e_c}{maxE_c}, else \end{cases}$$

$$e_c = \begin{cases} maxE_c & , if \sum_{F_i \in F(c)} F_i > 0 \\ maxE_c \cdot \sum_{C_i \in C(c)} \alpha_i C_i, else \end{cases}$$

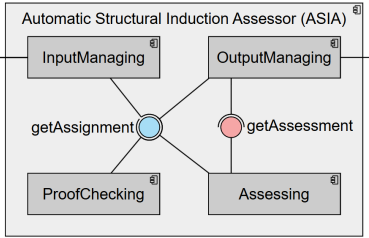
$$G_c = \sum\nolimits_{\forall c_i \in c} G_{c_i}, \ where \ c = \{c_1, ..., c_n\}, \ G \in \{A, e\}$$

Assessment and Feedback

```
adt Peano
sorts Nat
constructors zero: Nat
             inc: Nat -> Nat
operations add: Nat >< Nat -> Nat
vars n:Nat, m:Nat
axioms A1: add(n,zero)    = n
       A2: add(n,inc(m)) = inc(add(n,m))
```

```
task forall n:Nat : add(zero,n) = n induction n
  case zero maxpt 1 minsteps 1 maxsteps 1
  case inc  maxpt 2 minsteps 2 maxsteps 2
  IH        maxpt 1
```

```
proof /* of a student */
BC: show: fixed n:Nat : add(zero,inc(n))=inc(n)
add(zero,inc(n))
  {A2, lr, add(zero,inc(n)), [zero/n,n/m]}
= inc(add(zero,n))
  {A1, lr, add(zero,n), []}
= inc(n)
IH: fixed n:Nat : add(zero,n)=n
IS: show: add(zero,zero)=zero
add(zero,zero)
  {IH, lr, add(zero,zero), [zero/n]}
= zero
```



Wrong constructor inc for base case (BC)

Incorrect application of Axiom A1

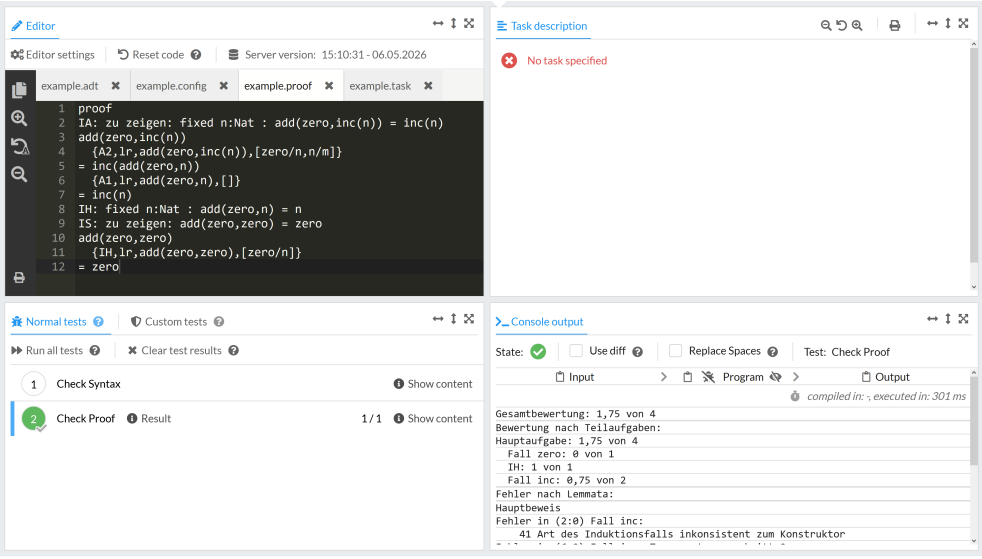
Wrong constructor zero for induction step (IS)

Induction hypothesis (IH) in base case zero

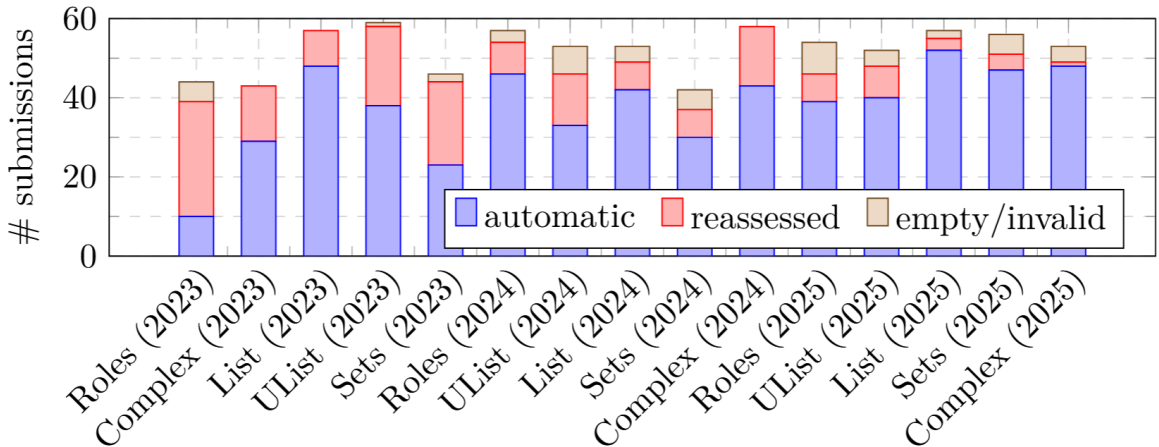
Substitution of fixed variable n

```
Overall Assessment: 1.75 of 4
Case zero: 0 of 1
IH: 1 of 1
Case inc: 0.75 of 2


Errors in (2:0) case inc:
  41
Errors in (6:2) case inc, term rewriting 2:
  8, 10 [ add( zero,n ) / n ]
Errors in (9:0) case zero:
  41
Errors in (11:2) case zero, term rewriting 1:
  36, 37 n


Error codes:
 8-Rule's application direction wrong
10-Wrong substitution for inverse direction
   of rule's application
36-Induction hypothesis is applied in base case
37-Fixed variable was substituted
41-Given induction case is
   inconsistent with constructor
```

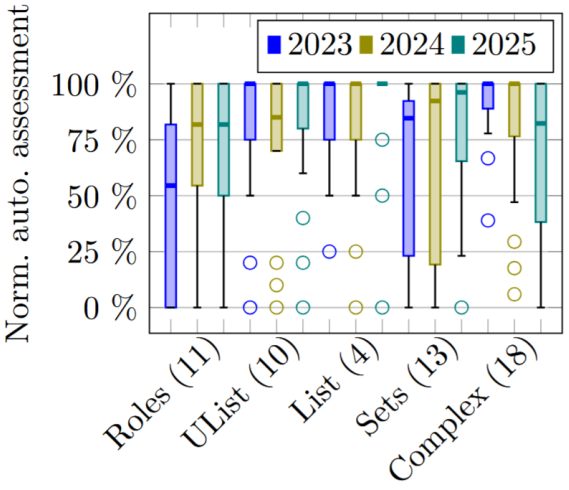
(Live Demo on) YAPEX



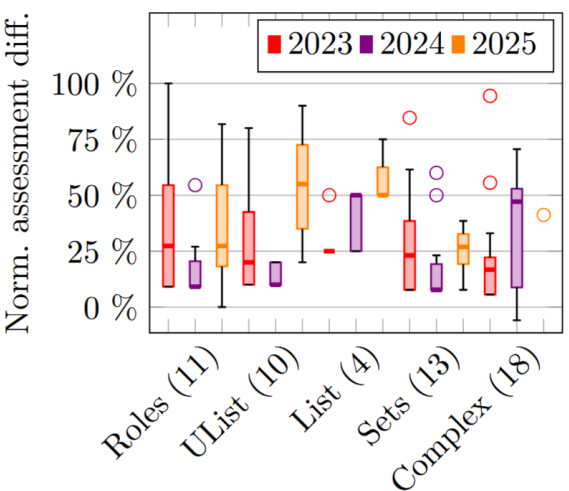
Evaluation - Submissions (192 Participants)



Evaluation - Distribution of Points

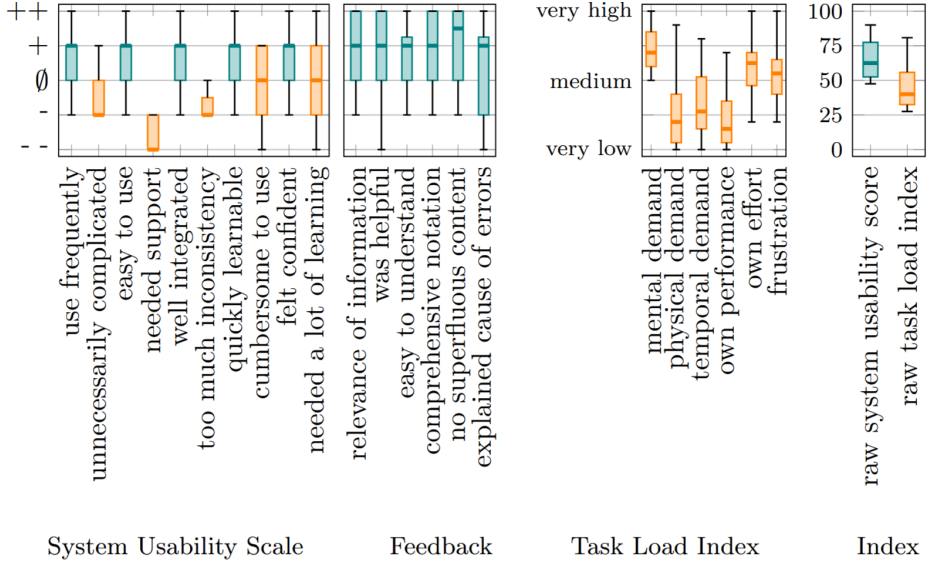


Automatic assessment



Manual reassessment

Evaluation - Survey (17 Participants - only 2025)



Conclusion

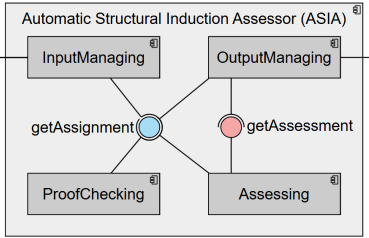
- Tool: Automatic Structural Induction Assessor (ASIA)
  - DSL for ADT, task, proof
  - Proof checking with follow-up error consideration
  - Fair assessment formula based on found errors
  - Integrated in YAPEX Webapp
- Evaluation - Case study (192 Participants):
  - Low amount of reassessment required
- Evaluation - Survey (17 Participants):
  - System Usability Score 62.5 (above average)
  - NASA Task Load Index 40 (reduced load)
  - Helpful and comprehensible Feedback

Thank You!

```
adt Peano
sorts Nat
constructors zero: Nat
             inc: Nat -> Nat
operations add: Nat >< Nat -> Nat
vars n:Nat, m:Nat
axioms A1: add(n,zero)    = n
       A2: add(n,inc(m)) = inc(add(n,m))
```

```
task forall n:Nat : add(zero,n) = n induction n
  case zero maxpt 1 minsteps 1 maxsteps 1
  case inc  maxpt 2 minsteps 2 maxsteps 2
  IH        maxpt 1
```

```
proof /* of a student */
BC: show: fixed n:Nat : add(zero,inc(n))=inc(n)
add(zero,inc(n))
  {A2, lr, add(zero,inc(n)), [zero/n,n/m]}
= inc(add(zero,n))
  {A1, lr, add(zero,n), []}
= inc(n)
IH: fixed n:Nat : add(zero,n)=n
IS: show: add(zero,zero)=zero
add(zero,zero)
  {IH, lr, add(zero,zero), [zero/n]}
= zero
```



Wrong constructor inc for base case (BC)

Incorrect application of Axiom A1

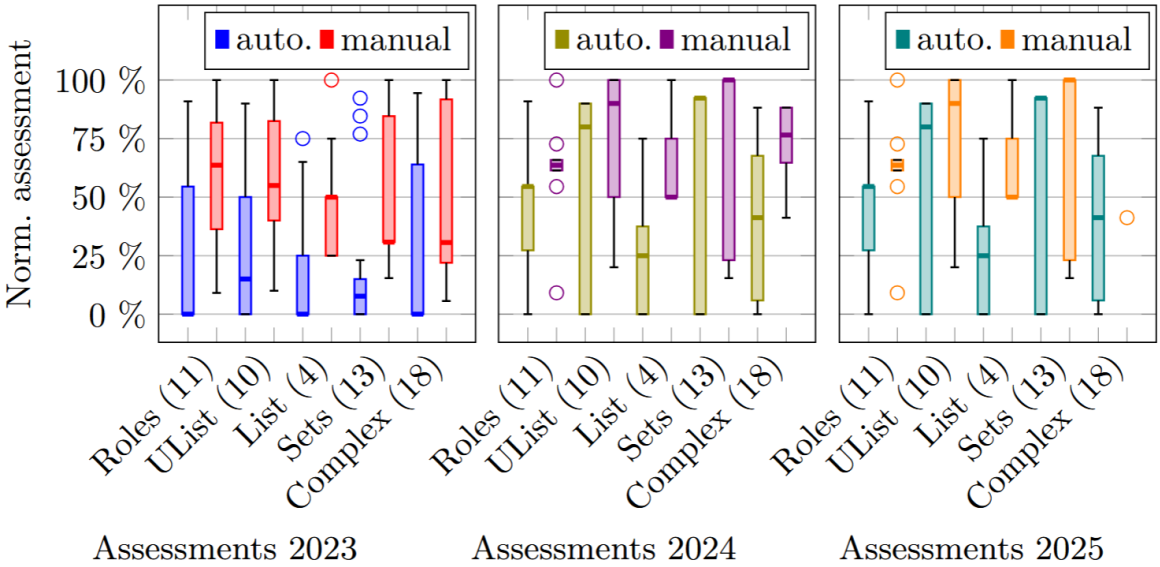
Wrong constructor zero for induction step (IS)

Induction hypothesis (IH) in base case zero

Substitution of fixed variable n

```
Overall Assessment: 1.75 of 4
Case zero: 0 of 1
IH: 1 of 1
Case inc: 0.75 of 2


Errors in (2:0) case inc:
  41
Errors in (6:2) case inc, term rewriting 2:
  8, 10 [ add( zero,n ) / n ]
Errors in (9:0) case zero:
  41
Errors in (11:2) case zero, term rewriting 1:
  36, 37 n


Error codes:
 8-Rule's application direction wrong
10-Wrong substitution for inverse direction
   of rule's application
36-Induction hypothesis is applied in base case
37-Fixed variable was substituted
41-Given induction case is
   inconsistent with constructor
```

Evaluation - Reassessed Submissions



Representation

ADT	::=	Sorts Constructors? Operations Vars Axioms?
Sorts	::=	$'sorts'$ Sort $(',' \text{ Sort})^*$
Constructors	::=	$'constructors'$ Operation$^+$
Operations	::=	$'operations'$ Operation$^+$
Operation	::=	OperationName $':'$ $('><' \text{ Sort})^*$ $'->'?$ Sort
Vars	::=	Var $':'$ Sort $(',' \text{ Var } ':' \text{ Sort})^*$
Axioms	::=	$'axioms'$ (AxiomName $':'$ Term $'='$ Term)$^+$
Term	::=	Var | (OperationName $('(' \text{ Term } (',' \text{ Term})^* ')')?$)

Representation

ADT	::=	Sorts Constructors? Operations Vars Axioms?
Sorts	::=	$'sorts'$ Sort $(',' \text{ Sort})^*$
Constructors	::=	$'constructors'$ Operation$^+$
Operations	::=	$'operations'$ Operation$^+$
Operation	::=	OperationName $':'$ $(\text{Sort } ('><' \text{ Sort})^* '->')?$ Sort
Vars	::=	Var $':'$ Sort $(',' \text{ Var } ':' \text{ Sort})^*$
Axioms	::=	$'axioms'$ (AxiomName $':'$ Term $'='$ Term)$^+$
Term	::=	Var | (OperationName $('(' \text{ Term } (',' \text{ Term})^* ')')?$)

Task	::=	$'task'$ Eq $('induction' \text{ Var})?$ (TaskPt | SubTasks)
Eq	::=	(Fixed $':')?$ (Forall $':')?$ Term $'='$ Term
Fixed	::=	$'fixed'$ Var $':'$ Sort $(',' \text{ Var } ':' \text{ Sort})^*$
Forall	::=	$'forall'$ Var $':'$ Sort $(',' \text{ Var } ':' \text{ Sort})^*$
TaskPt	::=	$'maxpt'$ Nat $'minsteps'$ Nat $'maxsteps'$ Nat
SubTasks	::=	$('case'$ ConstructorName TaskPt$)^+$ $'IH'$ $'maxpt'$ Nat

Representation

ADT	::=	Sorts Constructors? Operations Vars Axioms?
Sorts	::=	$'sorts'$ Sort $(',$ Sort$)^*$
Constructors	::=	$'constructors'$ Operation$^+$
Operations	::=	$'operations'$ Operation$^+$
Operation	::=	OperationName $' :'$ (Sort $('><'$ Sort$)^*$ $'->')?$ Sort
Vars	::=	Var $' :'$ Sort $(',$ Var $' :'$ Sort$)^*$
Axioms	::=	$'axioms'$ (AxiomName $' :'$ Term $' ='$ Term$)^+$
Term	::=	Var | (OperationName $('('$ Term $(',$ Term$)^*$ $')')?$)

Task	::=	$'task'$ Eq $('induction'$ Var$)?$ (TaskPt | SubTasks)
Eq	::=	(Fixed $' :')?$ (Forall $' :')?$ Term $' ='$ Term
Fixed	::=	$'fixed'$ Var $' :'$ Sort $(',$ Var $' :'$ Sort$)^*$
Forall	::=	$'forall'$ Var $' :'$ Sort $(',$ Var $' :'$ Sort$)^*$
TaskPt	::=	$'maxpt'$ Nat $'minsteps'$ Nat $'maxsteps'$ Nat
SubTasks	::=	$('case'$ ConstructorName TaskPt$)^+$ $'IH'$ $'maxpt'$ Nat

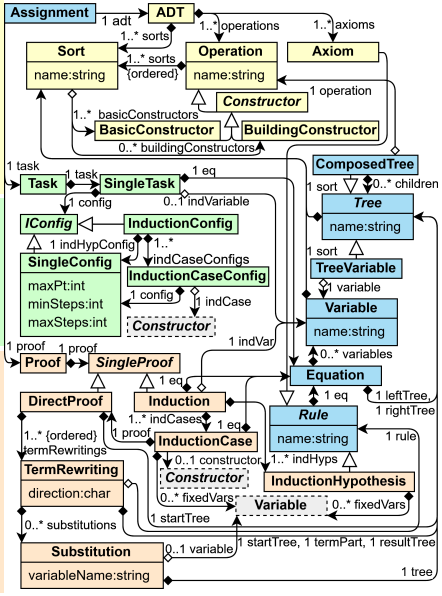
Proof	::=	$'proof'$ DirectProof | InductionProof
InductionProof	::=	$('induction'$ Var$)?$ Induction
Induction	::=	Basis$^+$ Hypothesis$^+$ Step*
Basis	::=	$'IB'$ Goal DirectProof
Hypothesis	::=	$'IH'$ IHName? Eq
Step	::=	$'IS'$ Goal DirectProof
Goal	::=	$'goal :'$ Eq
DirectProof	::=	Term TermRewriting$^+$
TermRewriting	::=	$'\{'$Rule$',('lr'|'rl')',$Term$',$Subst$'\}'$ $' ='$ Term
Subst	::=	$'['$ (Term $'/'$ Var $(',$ Term $'/'$ Var$)^*)?$ $']'$

Representation

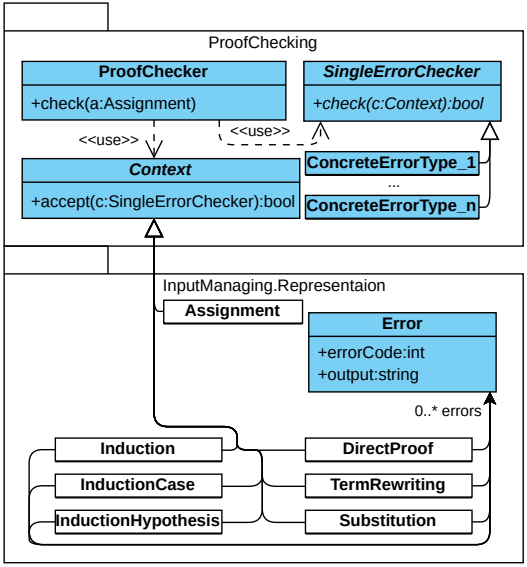
ADT	::=	Sorts Constructors? Operations Vars Axioms?
Sorts	::=	$'sorts'$ Sort $(',' \text{ Sort})^*$
Constructors	::=	$'constructors'$ Operation$^+$
Operations	::=	$'operations'$ Operation$^+$
Operation	::=	OperationName $':'$ (Sort $('><'$ Sort$)^*$ $'->')$? Sort
Vars	::=	Var $':'$ Sort $(',' \text{ Var } ':' \text{ Sort})^*$
Axioms	::=	$'axioms'$ (AxiomName $':'$ Term $'='$ Term)$^+$
Term	::=	Var | (OperationName $('('$ Term $(',' \text{ Term})^* ')')$?)

Task	::=	$'task'$ Eq $('induction'$ Var$)$? (TaskPt | SubTasks)
Eq	::=	(Fixed $':')$? (Forall $':')$? Term $'='$ Term
Fixed	::=	$'fixed'$ Var $':'$ Sort $(',' \text{ Var } ':' \text{ Sort})^*$
Forall	::=	$'forall'$ Var $':'$ Sort $(',' \text{ Var } ':' \text{ Sort})^*$
TaskPt	::=	$'maxpt'$ Nat $'minsteps'$ Nat $'maxsteps'$ Nat
SubTasks	::=	$('case'$ ConstructorName TaskPt$)^+$ $'IH'$ $'maxpt'$ Nat

Proof	::=	$'proof'$ DirectProof | InductionProof
InductionProof	::=	$('induction'$ Var$)$? Induction
Induction	::=	Basis$^+$ Hypothesis$^+$ Step*
Basis	::=	$'IB'$ Goal DirectProof
Hypothesis	::=	$'IH'$ IHName? Eq
Step	::=	$'IS'$ Goal DirectProof
Goal	::=	$'goal :'$ Eq
DirectProof	::=	Term TermRewriting$^+$
TermRewriting	::=	$'\{'$Rule$','$ $('lr'|'rl')'$$','$ Term$','$ Subst$'\}' ' ='$ Term
Subst	::=	$'['$ (Term $'/'$ Var $(',' \text{ Term } '/' \text{ Var})^*)$? $']'$



Proof Checking - Metamodel



Related Work

Table 1: Comparison of related approaches for assessing formal proofs

Criterion	Rocq	Isabelle/HOL	PVS
Proof checking	■	■	■
Mathematical induction	■	■	■
Structural induction	■	■	■
Any abstract data types	■	■	■
Assignment checking	□	□	□
Follow-up error consideration	□	□	□
Automatic assessment	□	□	□
Reliable Correctness	■	■	■

■ supported □ not-supported

Related Work

Table 1: Comparison of related approaches for assessing formal proofs

Criterion	Rocq	Isabelle/HOL	PVS	LogInd
Proof checking	■	■	■	■
Mathematical induction	■	■	■	■
Structural induction	■	■	■	■
Any abstract data types	■	■	■	□
Assignment checking	□	□	□	■
Follow-up error consideration	□	□	□	■
Automatic assessment	□	□	□	□
Reliable Correctness	■	■	■	■

■ supported □ not-supported

Related Work

Table 1: Comparison of related approaches for assessing formal proofs

Criterion	Rocq	Isabelle/HOL	PVS	LogInd	CalcCheck
Proof checking	■	■	■	■	■
Mathematical induction	■	■	■	■	■
Structural induction	■	■	■	■	■
Any abstract data types	■	■	■	□	□
Assignment checking	□	□	□	■	■
Follow-up error consideration	□	□	□	■	■
Automatic assessment	□	□	□	□	■
Reliable Correctness	■	■	■	■	■

■ supported □ not-supported

Related Work

Table 1: Comparison of related approaches for assessing formal proofs

Criterion	**Rocq**	**Isabelle/HOL**	**PVS**	**LogInd**	**CalcCheck**	**Ωmega**	**eMathChecker**	**ComIn-M**	**Yalep**	**LeanTutor**
Proof checking	■	■	■	■	■	■	■	■	■	■
Mathematical induction	■	■	■	■	■	■	■	■	■	■
Structural induction	■	■	■	■	■	□	□	□	□	□
Any abstract data types	■	■	■	□	□	□	□	□	□	□
Assignment checking	□	□	□	■	■	■	■	■	■	■
Follow-up error consideration	□	□	□	■	■	■	■	■	■	■
Automatic assessment	□	□	□	□	■	□	□	□	□	□
Reliable Correctness	■	■	■	■	■	■	■	■	■	■

■ supported □ not-supported

Related Work

Table 1: Comparison of related approaches for assessing formal proofs

Criterion	**Rocq**	**Isabelle/HOL**	**PVS**	**LogInd**	**CalcCheck**	**Ωmega**	**eMathChecker**	**ComIn-M**	**Yalep**	**LeanTutor**	**SIETTE**
Proof checking	■	■	■	■	■	■	■	■	■	■	□
Mathematical induction	■	■	■	■	■	■	■	■	■	■	□
Structural induction	■	■	■	■	■	□	□	□	□	□	□
Any abstract data types	■	■	■	□	□	□	□	□	□	□	□
Assignment checking	□	□	□	■	■	■	■	■	■	■	■
Follow-up error consideration	□	□	□	■	■	■	■	■	■	■	□
Automatic assessment	□	□	□	□	■	□	□	□	□	□	■
Reliable Correctness	■	■	■	■	■	■	■	■	■	■	■

■ supported □ not-supported

Related Work

Table 1: Comparison of related approaches for assessing formal proofs

Criterion	Rocq	Isabelle/HOL	PVS	LogInd	CalcCheck	Ωmega	eMathChecker	ComIn-M	Yalep	LeanTutor	SIETTE	AI autograders
Proof checking	■	■	■	■	■	■	■	■	■	■	□	■
Mathematical induction	■	■	■	■	■	■	■	■	■	■	□	■
Structural induction	■	■	■	■	■	□	□	□	□	□	□	□
Any abstract data types	■	■	■	□	□	□	□	□	□	□	□	□
Assignment checking	□	□	□	■	■	■	■	■	■	■	■	■
Follow-up error consideration	□	□	□	■	■	■	■	■	■	■	□	■
Automatic assessment	□	□	□	□	■	□	□	□	□	□	■	■
Reliable Correctness	■	■	■	■	■	■	■	■	■	■	■	□

■ supported □ not-supported

Concrete Assessment Schema

$$
\varphi(p) = \begin{cases}
0 & \textbf{if } (p{\equiv}tr_i \vee p{\equiv}ih) \wedge F_p \neq \emptyset \quad (1) \\
1 - \sum_{c \in C_p} \alpha(c) & \textbf{if } (p{\equiv}tr_i \vee p{\equiv}ih) \wedge F_p = \emptyset \quad (2) \\
pt \cdot \mathsf{min}\left(1\,, \frac{n}{l}\right) \cdot \left(\sum_{i=1}^{n} \frac{\varphi(tr_i)}{\mathsf{min}(n,u)}\right) & \textbf{if } p{\equiv}dp \quad (3) \\
0 & \textbf{if } p{\equiv}case \wedge F_{goal} \neq \emptyset \quad (4) \\
pt \cdot \mathsf{min}\left(1\,, \frac{n}{l}\right) \cdot \frac{\sum_{i=1}^{n} \left(\varphi(tr_i) - \sum_{c \in C_{goal}} \alpha(c)\right)}{\mathsf{min}(n,u)} & \textbf{if } p{\equiv}case \wedge F_{goal} = \emptyset \quad (5) \\
pt_{ih} \cdot \varphi(ih) + \sum_{i=1}^{m} \varphi(case_i) & \textbf{if } p{\equiv}ind \quad (6)
\end{cases}
$$

Fig. 2: Formal definition of the assessment function $\varphi : P \to \mathbb{Q}_0^+$. Here, a direct proof $dp = (pt, l, u, \langle tr_1, \cdots, tr_n \rangle)$ consists of the maximal points pt, the lower l and upper u bound of steps, and the term rewritings tr_i (with $i \in \{1, \ldots, n\}$); a $case = (goal, pt, l, u, \langle tr_1, \cdots, tr_n \rangle)$ additionally includes the $goal$, whereas an induction proof $ind = (pt_{ih}, ih, \{case_1, \ldots, case_m\})$ encompasses all cases, the induction hypothesis ih and corresponding points pt_{ih}.

Example: Calculation of Assessment for case inc

$$
\begin{aligned}
\varphi(inc) &= 2 \cdot \min\left(1\ ,\ \frac{2}{2}\right) \cdot \frac{\sum_{i=1}^{2}\left(\varphi(tr_i) - \sum_{c \in C_{goal}} \alpha(c)\right)}{\min(2,2)} \\
&= \sum_{i=1}^{2}\left(\varphi(tr_i) - \sum_{c \in C_{goal}} \alpha(c)\right) \ = (\varphi(tr_1) - 0.25) + (\varphi(tr_2) - 0.25) \\
&= \left(1 - \left(\sum_{c \in C_{tr_1}} \alpha(c)\right) - 0.25\right) + \left(1 - \left(\sum_{c \in C_{tr_2}} \alpha(c)\right) - 0.25\right) \\
&= (1 - 0 - 0.25) + (1 - (0.25 + 0.5) - 0.25) \ = 0.75 + 0 = \underline{0.75}
\end{aligned}
$$

Exercise: Roles

```
name Roles
sorts P,L
constructors player: P
             attack: P -> P
             spell : P -> P
             one: L
             up: L -> L
operations   fusion: P >< P -> P
             level: P -> L
             both: L >< L -> L
vars p : P, q : P, r : P, x : L, y : L
axioms F0: fusion(p,player) = p
       F1: fusion(p,attack(q)) = attack(fusion(p,q))
       F2: fusion(p,spell(q)) = spell(fusion(p,q))
       L0: level(player) = one
       L1: level(attack(p)) = up(level(p))
       L2: level(spell(p)) = up(up(level(p)))
       B0: both(x,one) = x
       B1: both(x,up(y)) = up(both(x,y))
```

```
task forall p:P, q:P, r:P :
     fusion(fusion(p,q),r)=fusion(p,fusion(q,r))
 induction r
  case player maxpt 2 minsteps 2 maxsteps 10
  case attack maxpt 4 minsteps 4 maxsteps 10
  case spell maxpt 4 minsteps 4 maxsteps 10
  IH maxpt 1
```

```
proof
IA: zu zeigen:
    forall p:P, q:P : fusion(fusion(p,q),player)
                      =fusion(p,fusion(q,player))
fusion(fusion(p,q),player)
  {...}
= ...

IH: fixed r:P : forall p:P, q:P : ...

IS: zu zeigen: forall p:P, q:P : fixed r:P : ...

IS: zu zeigen: forall p:P, q:P : fixed r:P : ...
```

Exercise: Unordered List

```
name LIST
sorts L, N, T
constructors zero:              N
             inc:     N      -> N
             nil:            L
             cons:    T >< L -> L
operations   add:     N >< N -> N
             append:  L >< L -> L
             reverse: L      -> L
             length:  L      -> N
vars l:L, l1:L, l2:L, x:T, n:N, m:N
axioms N0: add(m,zero)           = m
       N1: add(m,inc(n))         = inc(add(m,n))
       A0: append(nil, l)        = l
       A1: append(cons(x,l1),l2) = cons(x,append(l1,l2))
       R0: reverse(nil)          = nil
       R1: reverse(cons(x,l))
             = append(reverse(l), cons(x, nil))
       L0: length(nil)           = zero
       L1: length(cons(x,l))     = inc(length(l))
       Z0: length(append(l1,l2))
             = add(length(l1),length(l2))
```

```
task forall l:L :length(reverse(l))=length(l)
 induction l
  case nil maxpt 1 minsteps 1 maxsteps 10
  case cons maxpt 8 minsteps 8 maxsteps 20
  IH maxpt 1
```

```
proof
IA: zu zeigen: ...
...

IH: fixed l:L : ...

IS: zu zeigen: fixed l:L : forall x:T: ...
...
```

Exercise: List

```
name LIST
sorts L, N, T
constructors zero:              N
             inc:     N      -> N
             nil:            L
             cons:    T >< L -> L
operations   add:     N >< N -> N
             append:  L >< L -> L
             reverse: L      -> L
             length:  L      -> N
vars l:L, l1:L, l2:L, x:T, n:N, m:N
axioms N0: add(zero,n)=n
       N1: add(inc(n),m)=inc(add(n,m))
       A0: append(nil, l)         = l
       A1: append(cons(x,l1),l2) = cons(x,append(l1,l2))
       R0: reverse(nil)           = nil
       R1: reverse(cons(x,l))
             = append(reverse(l), cons(x, nil))
       L0: length(nil)            = zero
       L1: length(cons(x,l))      = inc(length(l))
       Z0: length(append(l1,l2))
             = add(length(l1),length(l2))
```

```
task forall l:L :append(l,nil)=l
 induction l
  case nil maxpt 1 minsteps 1 maxsteps 10
  case cons maxpt 2 minsteps 2 maxsteps 10
  IH maxpt 1
```

```
proof
IA: zu zeigen: ...
...

IH: fixed l:L : ...

IS: zu zeigen: fixed l:L : forall x:T: ...
...
```

Exercise: Set

```
name SET
sorts S,B,N
constructors zero:                    N
             inc:      N            -> N
             emptySet:                S
             in:       N >< S       -> S
             true:                    B
             false:                   B
operations   equals:   N >< N       -> B
             single:   N            -> S
             find:     N >< S       -> S
             union:    S >< S       -> S
             inter:    S >< S       -> S
             isEmpty:  S            -> B
             if:       B >< S >< S -> S
vars x:N, y:N, s:S, s1:S, s2:S
axioms  Q0: equals(zero,zero)         = true
        Q1: equals(zero,inc(x))       = false
        Q2: equals(inc(x),zero)       = false
        Q3: equals(inc(x),inc(y))     = equals(x,y)
        Q4: equals(x,x)               = true
        A0: in(x,in(x,s))             = in(x,s)
        A1: in(y,in(x,s))             = in(x,in(y ,s))
        A2: single(x)                 = in(x,emptySet)
        F0: find(x,emptySet)          = emptySet
        F1: find(x,in(y,s))
              = if(equals(x,y),single(x),find(x,s))
```

```
U0: union(s,emptySet)         = s
U1: union(emptySet,s)         = s
U2: union(s1,in(x,s2))        = union(in(x,s1),s2)
U3: union(in(x,s1),s2)        = in(x,union(s1,s2))
I0: inter(s,emptySet)         = emptySet
I1: inter(emptySet,s)         = emptySet
I2: inter(s1,in(x,s2))        = union(find(x,s1)
                                      ,inter(s1,s2))
E0: isEmpty(emptySet)         = true
E1: isEmpty(in(x,s))          = false
C0: if(true,s1,s2)            = s1
C1: if(false,s1,s2)           = s2
Z0: in(x,inter(in(x,s1),s2)) = in(x,inter(s1,s2))
```

```
task forall s:S, t:S : inter(union(s,t),s)=s
 induction s
  case emptySet maxpt 2 minsteps 2 maxsteps 10
  case in maxpt 10 minsteps 8 maxsteps 20
  IH maxpt 1
```

```
proof
IA: zu zeigen: forall t:S : ...
  ...
IH: fixed s:S : forall t:S : inter(union(s,t),s)=s
IS: zu zeigen: fixed s:S : forall t:S, x:N : ...
  ...
```

Exercise: Complex

```
name Complex
sorts D,N
constructors alpha:          D
             omega:          D
             beta:      D -> D
             gamma:     D -> D
             zero:           N
             inc:       N -> N
operations   inv:       D -> D
             len:       D -> N
             con: D >< D -> D
vars x : D, y : D
axioms  A0: inv(alpha)=omega
        A1: inv(omega)=alpha
        A2: inv(beta(x))=gamma(inv(x))
        A3: inv(gamma(x))=beta(inv(x))
        L0: len(alpha)=inc(zero)
        L1: len(omega)=inc(zero)
        L2: len(beta(x))=inc(len(x))
        L3: len(gamma(x))=inc(len(x))
        C0: con(alpha,y)=y
        C1: con(omega,y)=y
        C2: con(beta(x),y)=beta(con(x,y))
        C3: con(gamma(x),y)=gamma(con(x,y))
```

```
task forall x:D, y:D : len(con(x,y))=len(con(inv(x),y))
 induction x
  case alpha maxpt 3 minsteps 3 maxsteps 10
  case omega maxpt 3 minsteps 3 maxsteps 10
  case beta maxpt 5 minsteps 5 maxsteps 10
  case gamma maxpt 5 minsteps 5 maxsteps 10
  IH maxpt 2
```

```
proof
IA: zu zeigen: forall y:D : len(con(alpha,y))
                              =len(con(inv(alpha),y))
len(con(alpha,y)) {...}
= ... {...}
= len(con(inv(alpha),y))
IA: zu zeigen: forall y:D : ...
...
IH: fixed x:D : forall y:D : ...
IS: zu zeigen: fixed x:D : forall y:D :
    len(con(beta(x),y))=len(con(inv(beta(x)),y))
len(con(beta(x),y)) {...}
= ... {...}
= len(con(inv(beta(x)),y))
IS: zu zeigen: fixed x:D : ...
...
```