CTP-based programming languages
Considerations about an experimental design

F. Haftmann\textsuperscript{1}  C. Kaliszyk\textsuperscript{1}  W. Neuper\textsuperscript{2}

\textsuperscript{1}Software & Systems Engineering, TU München
\textsuperscript{2}Institute for Software Technology, TU Graz

PLMMS 2010 Paris
Jun.2, 2010
Outline

1 Introduction
   Plans for integrating three lines of work
   Motivation for the language: tutoring

2 Design of the programming language
   Syntax of the language
   Specification is a language element
   Contexts = environments ∪ predicates
   We run into “patterns of specifications”

3 Preview and Summary
1 **Introduction**

Plans for integrating three lines of work

Motivation for the language: tutoring

2 **Design of the programming language**

Syntax of the language

Specification is a language element

Contexts = environments $\cup$ predicates

We run into “patterns of specifications”

3 **Preview and Summary**
We plan to integrate 3 different lines of work:

1. Code generation in Isabelle (Florian Haftmann)
2. CAS in HOL Light (Cezary Kaliszyk)
3. Language/interpreter for tutoring (Walther Neuper) based on work by Peter Lucas (“Lucas-interpreter”)
Three lines of work

We plan to integrate 3 different lines of work:

1. Code generation in Isabelle (Florian Haftmann)
2. CAS in HOL Light (Cezary Kaliszyk)
3. Language/interpreter for tutoring (Walther Neuper) based on work by Peter Lucas (“Lucas-interpreter”)
Introduction

Plans for integrating three lines of work
Motivation for the language: tutoring

Design of the programming language
Syntax of the language
Specification is a language element
Contexts = environments $\cup$ predicates
We run into “patterns of specifications”

Preview and Summary
Topic of the talk (= paper)

proving and programming

specification

program

interpreter

Introduction
Three lines of work
Motivation: tutoring

Language
Syntax
Specification
Contexts
Spec. patterns

Preview and Summary
Topic of the talk (= paper)

Isabelle/Isar
logical operating system
(contexts etc)

Interpreter

Proving and Programming
Topic of the paper: the final goal

Isabelle/Isar
logical operating system
(contexts etc)

production
specification
program
output

Introduction
Three lines of work
Motivation: tutoring

Language
Syntax
Specification
Contexts
Spec. patterns

Preview and Summary
(My) motivation: Tutoring

Tutoring

Authoring

worksheet

dialog module

specification

program

output

Tutoring Authoring

interpret (P.L)

Isabelle/Isar
logical operating system
(contexts etc)

worksheet
dialog module
Example from the paper

CTP-based languages
Florian Haftmann, Cezary Kaliszyk, Walther Neuper

Introduction
Three lines of work
Motivation: tutoring

Language
Syntax
Specification
Contexts
Spec. patterns

Preview and Summary

Dialog module
worksheet
Isabelle/Isar
logical operating system
(contexts etc)

Interpretation (P.L)

Specification

In: function q, Length L
pre: q is_integrable
Δ L > 0
Post: y(0)=0 Δ y'(0)=0
Δ V(0)=q.L
Δ M_b(L)=0

Program

Script B (q, L, v, Cs) =
LET
  funs = Subproblem
    (thy, pbl, met) q, L, v
equs = Subproblem ...
sols = Subproblem ...
B = Take (LAST funs)
B = ((Substitute sols)@
  (Rewrite_Set poly)) B
IN B

output

In: function q, Length L
pre: q is_integrable
Δ L > 0
Post: y(0)=0 Δ y'(0)=0
Δ V(0)=q.L
Δ M_b(L)=0

interpretation (P.L)
Example: start tutoring

Worksheet

Dialog module

Tutor

Isabelle/Isar
logical operating system
(contexts etc)

Interpreter (P.L)

Specification

In: function q, Length L
pre: q is integrable
Δ L > 0
out: function y(x)
Post: y(0)=0 Δ y'(0)=0
Δ V(0)=q.L
Δ M_b(L)=0

Program

Script B (q, L, v, Cs) =
LET
funs = Subproblem (thy, pbl, met) q, L, v
equs = Subproblem …
sols = Subproblem …
B = Take (LAST funs)
B = (Substitute sols)@(Rewrite_Set poly) B
IN B

In: function q, Length L
pre: q is_integrable
Δ L > 0
out: function y(x)
Post: y(0)=0 Δ y'(0)=0
Δ V(0)=q.L
Δ M_b(L)=0

Tutoring
Example: Tutoring start

Introduction
Three lines of work
Motivation: tutoring

Language
Syntax
Specification
Contexts
Spec. patterns

Preview and Summary

worksheet
Problem (B, bendl)

dialog module

specification
In: function q, Length L
pre: q is_integrable
 Δ L > 0
out: function y(x)
Post: y(0)=0 Δ y'(0)=0
 Δ V(0)=q.L
 Δ M_b(L)=0

program
Script B (q, L, v, Cs) = LET
funs = Subproblem 
thy, pbl, met) q, L, v
equs = Subproblem …
sols = Subproblem …
B = Take (LAST funs)
B = ((Substitute sols)@
(Rewrite_Set poly)) B
IN B

output

Isabelle/Isar
logical operating system
(contexts etc)

Tutoring

Initial interpreter (P.L)
Example: Tutoring step 1

**Worksheet**

```
Problem (B, bendl)
Problem (B, load2bl)
Q(x) = c·q·x, M(x) = ...
```

**Dialog module**

**Specification**

```
In: function q, Length L
pre: q is_integrable
Δ L > 0
out: function y(x)
Post: y(0)=0 Δ y'(0)=0
Δ V(0)=q·L
Δ M_b(L)=0
```

**Isabelle/Isar**

```
logical operating system
(contexts etc)
```

**Program**

```
Script B (q, L, v, Cs) =
  LET
  funs = Subproblem (thy, pbl, met) q, L, v
  equus = Subproblem ...
  sols = Subproblem ...
  B = Take (LAST funs)
  B = ((Substitute sols)@ (Rewrite_Set poly)) B
IN B
```

**Output**
Example: Tutoring step 2

Tutoring

worksheet
Problem (B, bendl)
Problem (B, load2bl)
Q(x) = c·q·x, M(x) = …
Problem (B, sidecds)
L·q = x, 0 = c_2+L·c…

dialog module

Isabelle/Isar
logical operating system
(contexts etc)

Tutor

specification

In: function q,
Length L
pre: q is integrable
Δ L > 0
out: function y(x)
Post: y(0)=0 Δ y'(0)=0
Δ V(0)=q·L
Δ M_b(L)=0

program

Script B (q, L, v, Cs) =
LET
funs = Subproblem
(thy, pbl, met) q, L, v
equs = Subproblem …
sols = Subproblem …
B = Take (LAST funs)
B = ((Substitute sols)@
(Rewrite_Set poly)) B
IN B

output
Example: Tutoring step 3

**Tutoring**

**worksheet**

Problem (B, bendl)
Problem (B, load2bl)
Q(x) = c·q·x, M(x) = …
Problem (B, sidecds)
L.q = x, 0 = c_2+L.c…
solveSys [0=c_3, …
c = q·L, c_2 = -L^2·q/2.

**dialog module**

**specification**

In: function q, Length L
pre: q is_integrable
Δ L > 0
out: function y(x)
Post: y(0)=0 Δ y'(0)=0
Δ V(0)=q·L
Δ M_b(L)=0

**Isabelle/Isar**

logical operating system
(contexts etc)

**program**

Script B (q, L, v, Cs) =
LET
funs = Subproblem (thy, pbl, met) q, L, v
equs = Subproblem …
sols = Subproblem …
B = Take (LAST funs)
B = ((Substitute sols)@ (Rewrite_Set poly)) B
IN B

**output**

Interpreter (P.L)
Example: Tutoring step 4

Worksheet

Problem (B, bendl)
Problem (B, load2bl)
Q(x) = c·q·x, M(x) = ...
Problem (B, sidecds)
L·q = x, 0 = c·L+c... solveSys [0=c·3, ...
c = q·L, c·2 = -L^2·q/2.
y(x) = c·4+c·3·x-1/EI...

Tutoring

Script B (q, L, v, Cs) =
LET
  funs = Subproblem (thy, pbl, met) q, L, v
  equs = Subproblem ...
  sols = Subproblem ...
B = Take (LAST funs)
B = ((Substitute sols)@(Rewrite_Set poly)) B
IN B
Example: Tutoring step 5

worksheet
Problem (B, bendl)
Problem (B, load2bl)
Q(x) = c·q·x, M(x) = ...
Problem (B, sidescds)
L·q = x, 0 = c_2 + L·c...
solveSys [0=c_3, ... c = q·L, c_2 = -L^2·q/2.
y(x) = c_4 + c_3·x - 1/El...
y(x) = 0 + 0·x - 1/El ...

program
Script B (q, L, v, Cs) =
LET
funs = Subproblem (thy, pbl, met) q, L, v
equs = Subproblem ...
sols = Subproblem ...
B = Take (LAST funs)
B = (Substitute sols)@ (Rewrite_Set poly) B
IN B

Tutoring
Example: Tutoring finished

CTP-based languages
Florian Haftmann, Cezary Kaliszyk, Walther Neuper

Introduction
Three lines of work
Motivation: tutoring

Language
Syntax
Specification
Contexts
Spec. patterns

Preview and Summary

Tutoring

worksheet
Problem (B, bendl)
Problem (B, load2bl)
Q(x) = c-q.x, M(x) = …
Problem (B, sidecds)
L.q = x, 0 = c_2+L.c…
solveSys [0=c_3, …
c = q.L, c_2 = -L^2.q/2.
y(x) = c_4+c_3.x-1/El...
y(x) = 0 + 0.x – 1/El…
y(x) = (q.L^2)/(4.El) . X^2
- (q.L)/(6.El) . x^3
+ q/(24.El) . x^4

dialog module

Isabelle/Isar
logical operating system
(contexts etc)

interpreter (P.L)

specification
In: function q, Length L
pre: q is_integrale
Δ L > 0
out: function y(x)
Post: y(0)=0 Δ y'(0)=0
Δ V(0)=q.L
Δ M_b(L)=0

program
Script B (q, L, v, Cs) =
LET
funs = Subproblem
(thy, pbl, met) q, L, v
equs = Subproblem …
sols = Subproblem …
B = Take (LAST funs)
B = ((Substitute sols)@ (Rewrite_Set poly)) B
IN B

output
Same problem: just get result

Isabelle/Isar
logical operating system
(contexts etc)

specification
In: function q, Length L
pre: q is integrable
\[ \Delta L > 0 \]
out: function y(x)
Post: \[ y(0)=0, \Delta y'(0)=0 \]
\[ \Delta V(0)=q.L \]
\[ \Delta M_b(L)=0 \]

program
Script B (q, L, v, Cs) =
LET
funs = Subproblem (thy, pbl, met) q, L, v
equs = Subproblem …
sols = Subproblem …
B = Take (LAST funs)
B = ((Substitute sols)@(Rewrite_Set poly)) B
IN B

output
y(x) = \( \frac{(q.L^2)}{(4.EI)} \cdot x^2 \)
\[ - \frac{(q.L)}{(6.EI)} \cdot x^3 \]
\[ + \frac{q}{(24.EI)} \cdot x^4 \]
Topic of the talk

Isabelle/Isar
logical operating system (contexts etc)

```
<table>
<thead>
<tr>
<th>specification</th>
<th>program</th>
<th>output</th>
</tr>
</thead>
</table>
```

Proving and Programming
Design issues — focus of talk

We plan to integrate 3 different lines of work:

1. Code generation in Isabelle (Florian Haftmann)
2. CAS in HOL Light (Cezary Kaliszyk)
3. Language/interpreter for tutoring (Walther Neuper) based on work by Peter Lucas (“Lucas-interpreter”)

The plan raises design issues and open questions (TODO)

- clarify integration of CAS-like features
- language design
- clarify integration of programming and proving (TODO)
- integration of code generation
- compiler construction (TODO)
Design issues — focus of talk

We plan to integrate 3 different lines of work:

1. Code generation in Isabelle (Florian Haftmann)
2. CAS in HOL Light (Cezary Kaliszyk)
3. Language/interpreter for tutoring (Walther Neuper)
   based on work by Peter Lucas (“Lucas-interpreter”)

The plan raises design issues and open questions (TODO)

- clarify integration of CAS-like features
- language design
- clarify integration of programming and proving (TODO)
- integration of code generation
- compiler construction (TODO)

← focus of my talk
Design issues — focus of talk

We plan to integrate 3 different lines of work:

1. Code generation in Isabelle (Florian Haftmann)
2. CAS in HOL Light (Cezary Kaliszyk)
3. Language/interpreter for tutoring (Walther Neuper) based on work by Peter Lucas (“Lucas-interpreter”)

The plan raises design issues and open questions (TODO)

- clarify integration of CAS-like features
- language design
- clarify integration of programming and proving (TODO)
- integration of code generation
- compiler construction (TODO)
1 Introduction
   Plans for integrating three lines of work
   Motivation for the language: tutoring

2 Design of the programming language
   Syntax of the language
   Specification is a language element
   Contexts = environments $\cup$ predicates
   We run into “patterns of specifications”

3 Preview and Summary
A functional language, BNF

```
script ::= Script id arg* = body
arg ::= id | ( ( id :: type ) )
body ::= expr
expr ::= id | ( ( id :: type ) )
       | expr o expr (*Isabelle/HOL*)
       | LET id = expr ( ; id = expr)* IN expr
       | IF prop THEN expr ELSE expr
       | listexpr (*lists in Isabelle/HOL*)
       | Repeat expr (*tactics*)
       | Try expr
       | expr Or expr
       | Take ( id | listexpr ) (*tactics*)
       | Rewrite id ( id | listexpr )
       | Rewrite_Set id ( id | listexpr )
       | Substitute subst ( id | listexpr )
       | SubProblem (id, idlist, idlist) arglist
       | ...(geometry !)
```
Isabelle/HOL code example
“bending line”

01 Script bendingLine
02 (l_::real) (q_::real) (v_::real) (b_::real=>real) (ct_::bool list) =
03 (LET
04     (funs_:: bool list) =
05     (SubProblem (Bendingline,[bendingline,integrate],
06         [bendingline,integrate])
07         [real_ q_, real_real_ b_, real_ v_]);
08     (equs_::bool list) =
09     (SubProblem (Bendingline,[bendingline,setConstraints],
10         [bendingline,setConstraints])
11         [bools_ funs_, bools_ ct_, real_ l_]);
12     (sols_::bool list) =
13     (SubProblem (Real,[equation,system,linear],[])
14         [bools_ equs_, reals_ [c,c_2,c_3,c_4]]);
15     B_ = Take (LAST funs_);
16     B_ = ((Substitute sols_) o
17         (Rewrite_Set_Inst [(bdv, v_)] make_ratpoly_in)) B_
18     IN B_)
Example: Isabelle/HOL elements

```isabelle
01  Script bendingLine
02  \((l_::real) (q_::real) (v_::real) (b_::real=\to real) (ct_::bool list) =
03    (LET
04      \(\\text{funs}_\text{\_}\:: bool list) =
05        (\text{SubProblem} (\text{Bendingline},[\text{bendingline,\text{integrate}],}
06         [\\text{bendingline,\text{integrate}])
07        \[\text{real}_\ \ q_\,, \text{real}_\ \ \ \text{b}_\,, \text{real}_\ \ v_\]) ;
08      \(\text{equs}_\text{\_}\::bool list) =
09        (\text{SubProblem} (\text{Bendingline},[\text{bendingline,\text{setConstraints}],}
10         [\text{bendingline,\text{setConstraints}])
11        \[\text{bools}_\ \ \ \text{funs}_\,, \text{bools}_\ \ \text{ct}_\,, \text{real}_\ \ l_]) ;
12      \(\text{sols}_\text{\_}\::bool list) =
13        (\text{SubProblem} (\text{Real},[\text{equation,\text{system,\text{linear}]},[])
14        \[\text{bools}_\ \ \ \text{equs}_\,, \text{reals}_\ [c,c_2,c_3,c_4]]) ;
15 \s_\, = \text{Take} (\text{LAST funs}_\text{\_}) ;
16 \s_\, = (\text{Substitute sols}_\text{\_}) \circ
17    (\text{Rewrite\_Set\_Inst \[(bdv, v\_)] make\_ratpoly\_in}) \s_\;
18 \text{IN} \ \s_\)
```
Example: list functions from Isabelle/HOL

01 Script bendingLine
02 (l_ :: real) (q_ :: real) (v_ :: real) (b_ :: real = \rightarrow real) (ct_ :: bool list) =
03 (LET
04 (funs_ :: bool list) =
05 (SubProblem (Bendingline, [bendingline, integrate],
06 [bendingline, integrate])
07 (real_ q_, real_real_ b_, real_ v_));
08 (equus_ :: bool list) =
09 (SubProblem (Bendingline, [bendingline, setConstraints],
10 [bendingline, setConstraints])
11 (bools_ funs_, bools_ ct_, real_ l_));
12 (sols_ :: bool list) =
13 (SubProblem (Real, [equation, system, linear], [])[bools_ equus_, reals_ [c, c_2, c_3, c_4]]);
15 B_ = Take (LAST funs_);
16 B_ = ((Substitute sols_) o
17 (Rewrite_Set_Inst [(bdv, v_)] make_ratpoly_in)) B_
18 IN B_)}
Example: (no) tacticals

01 Script bendingLine
02 (l_::real) (q_::real) (v_::real) (b_::real=>real) (ct_::bool list) =
03 (LET
04   (funs_:: bool list) =
05     (SubProblem (Bendingline,[bendingline,integrate],
06                     [bendingline,integrate])
07     [real_ q_, real_real_ b_, real_ v_]);
08   (equus_::bool list) =
09     (SubProblem (Bendingline,[bendingline,setConstraints],
10                     [bendingline,setConstraints])
11     [bools_ funs_, bools_ ct_, real_ l_]);
12   (sols_::bool list) =
13     (SubProblem (Real,[equation,system,linear],[])
14     [bools_ equus_, reals_ [c,c_2,c_3,c_4]]);
15   B_ = Take (LAST funs_);
16   B_ = ((Substitute sols_) o
17         (Rewrite_Set_Inst [(bdv, v_) make_ratpoly_in])) B_
18   IN B_)
Example “differentiation”: tacticals

01 Script differentiate \((f_{\text{::real}} \Rightarrow \text{real}) (v_{\text{::real}}) =\)
02 \begin{align*}
03 & \quad \text{LET} \\
04 & \quad f'_{\_} = \text{Take} \ (d_d \ v_{\_} \ f_{\_}) = \\
05 & \quad \text{IN Repeat} \\
06 & \quad (\text{Repeat} \ (\text{Rewrite}_\text{Inst} [(bdv, v_{\_})] \ \text{diff}_\text{sum}) \ \text{Or} \\
07 & \quad \text{Repeat} \ (\text{Rewrite}_\text{Inst} [(bdv, v_{\_})] \ \text{diff}_\text{prod}_\text{const}) \ \text{Or} \\
08 & \quad \text{Repeat} \ (\text{Rewrite}_\text{Inst} [(bdv, v_{\_})] \ \text{diff}_\text{prod}) \ \text{Or} \\
09 & \quad \text{Repeat} \ (\text{Rewrite}_\text{Inst} [(bdv, v_{\_})] \ \text{diff}_\text{quot}) \ \text{Or} \\
10 & \quad \text{Repeat} \ (\text{Rewrite}_\text{Inst} [(bdv, v_{\_})] \ \text{diff}_\text{sin}) \ \text{Or} \\
11 & \quad \text{Repeat} \ (\text{Rewrite}_\text{Inst} [(bdv, v_{\_})] \ \text{diff}_\text{cos}) \ \text{Or} \\
12 & \quad \text{Repeat} \ (\text{Rewrite}_\text{Inst} [(bdv, v_{\_})] \ \text{diff}_\text{pow}_\text{chain}) \ \text{Or} \\
13 & \quad \text{Repeat} \ (\text{Rewrite}_\text{Inst} [(bdv, v_{\_})] \ \text{diff}_\text{pow}) \ \text{Or} \\
14 & \quad \text{Repeat} \ (\text{Rewrite}_\text{Inst} [(bdv, v_{\_})] \ \text{diff}_\text{const}) \ \text{Or} \\
15 & \quad \text{Repeat} \ (\text{Rewrite}_\text{Inst} [(bdv, v_{\_})] \ \text{diff}_\text{var}) \ \text{Or} \\
16 & \quad \text{Repeat} \ (\text{Rewrite}_\text{Set} \text{make}_\text{polynomial}) \\
17 & \quad \text{f'_{\_}}
\end{align*}
Example: tactics (for “calculation”)

01 Script bendingLine
02 (l_::real) (q_::real) (v_::real) (b_::real=>real) (ct_::bool list) =
03 (LET
04   (funs_:: bool list) =
05     (SubProblem (Bendingline,[bendingline,integrate],
06                   [bendingline,integrate])
07       [real_ q_, real_real_ b_, real_ v_]);
08   (equs_::bool list) =
09     (SubProblem (Bendingline,[bendingline,setConstraints],
10                   [bendingline,setConstraints])
11       [bools_ funs_, bools_ ct_, real_ l_]);
12   (sols_::bool list) =
13     (SubProblem (Real,[equation,system,linear],[])
14       [bools_ equs_, reals_ [c,c_2,c_3,c_4]]);
15   B_ = Take (LAST funs_);
16   B_ = ((Substitute sols_) o
17         (Rewrite_Set_Inst [(bdv, v_)] make_ratpoly_in)) B_
18 IN B_)
What about the mix . . . ???

script ::= Script id arg* = body
arg ::= id | ( ( id :: type ) )
body ::= expr
expr ::= % id . expr (*Isabelle/HOL*)
| expr o expr
| LET id = expr ( ; id = expr)* IN expr
| IF prop THEN expr ELSE expr

| listexpr (*lists in Isabelle/HOL*)
| id | ( ( id :: type ) )
| Repeat expr (*tacticals*)
| Try expr
| expr Or expr

| Take ( id | listexpr ) (*tactics*)
| Rewrite id ( id | listexpr )
| Rewrite_Set id ( id | listexpr )
| Substitute subst ( id | listexpr )
| SubProblem (id, idlist, idlist) arglist
| . . . (geometry !)
1 Introduction
Plans for integrating three lines of work
Motivation for the language: tutoring

2 Design of the programming language
Syntax of the language
Specification is a language element
Contexts = environments $\cup$ predicates
We run into “patterns of specifications”

3 Preview and Summary
Example subproblem

01  Script bendingLine
02  (l_::real) (q_::real) (v_::real) (b_::real=>real) (ct_::bool list) =
03  (LET
04    (funs_:: bool list) =
05      (SubProblem (Bendingline,[bendingline,integrate],
06                     [bendingline,integrate])
07                     [real_ q_, real_real_ b_, real_ v_]);
08    (equus_::bool list) =
09      (SubProblem (Bendingline,[bendingline,setConstraints],
10                     [bendingline,setConstraints])
11                     [bools_ funs_, bools_ ct_, real_ l_]);
12    (sols_::bool list) =
13      (SubProblem (Real, [equation,system,linear], [])
14          [bools_ equus_, reals_ [c, c_2, c_3, c_4]]);
15  B_ = Take (LAST funs_);
16  B_ = ((Substitute sols_) o
17      (Rewrite_Set_Inst [(bdv, v_)] make_ratpoly_in)) B_
18  IN B_)
Example subproblem

01 \textbf{Script bendingLine} \\
02 (l\_::real) (q\_::real) (v\_::real) (b\_::real=>real) (ct\_::bool list) = \\
03 (LET \\
04 (funs\_:: bool list) = \\
05 (SubProblem (Bendingline,[bendingline,integrate], \\
06 [bendingline,integrate]) \\
07 [real\_ q\_, real\_real\_ b\_, real\_ v\_]); \\
08 (equs\_::bool list) = \\
09 (SubProblem (Bendingline,[bendingline,setConstraints], \\
10 [bendingline,setConstraints]) \\
11 [bools\_ funs\_, bools\_ ct\_, real\_ l\_]); \\
12 (sols\_::bool list) = \\
13 (SubProblem (Real, [equation,system,linear], [])) \\
14 [bools\_ equs\_, reals\_ [c, c\_2, c\_3, c\_4]]); \\
15 B\_ = Take (LAST funs\_); \\
16 B\_ = ((Substitute sols\_) o \\
17 (Rewrite Set Inst [(bdv, v\_)] make_ratpoly_in)) B\_ \\
18 \texttt{IN B\_})
Example subproblem

01 Script bendingLine
02 (l_::real) (q_::real) (v_::real) (b_::real=>real) (ct_::bool list) =
03 (LET
04   (funs_:: bool list) =
05     (SubProblem (Bendingline,[bendingline,integrate],
06                      [bendingline,integrate])
07     [real_ q_, real_real_ b_, real_ v_]);
08   (equs_::bool list) =
09     (SubProblem (Bendingline,[bendingline,setConstraints],
10                      [bendingline,setConstraints])
11     [bools_ funs_, bools_ ct_, real_ l_]);
12   (sols_::bool list) =
13     (SubProblem (Real, [equation,system,linear], [])
14      [bools_ equs_, reals_ [c, c_2, c_3, c_4]]);
15   B_ = Take (LAST funs_);
16   B_ = ((Substitute sols_) o
17         (Rewrite_Set_Inst [(bdv, v_)] make_ratpoly_in)) B_
18   IN B_)
SubProblem’s 1\textsuperscript{st} argument is a triple:

1. **Real**: Isabelle \textbf{theory}, contains definitions and theorems for (2) and (3)

2. \texttt{[equation,system,linear]}: key into a tree of \textbf{specification}-patterns; instantiated by Subproblems 2\textsuperscript{nd} argument (= arguments of (3)) to specification during runtime

3. \texttt{[]}: key into a tree(?) of \textbf{programs}; \textit{empty} in this case, because (CAS-like) system finds appropriate method solving equational systems.
Example: subproblem

ad (2): Specification \([\text{equation, system, linear}]\)

\[
\begin{align*}
\text{in} & \quad : \quad \text{equations} \left[ L \cdot q_0 = c, 0 = c_2 + 2 \cdot L \cdot c - \frac{L^2 \cdot q_0}{2}, 0 = c_3, 0 = c_4 \right] , \\
& \text{bound_vars} \ [c, c_2, c_3, c_4] \\
\text{pre} & \quad : \quad \left| \left[ L \cdot q_0 = c, 0 = c_2 + 2 \cdot L \cdot c - \frac{L^2 \cdot q_0}{2}, 0 = c_3, 0 = c_4 \right] \right| = \\
& \quad \quad = \left| [c, c_2, c_3, c_4] \right| \land \left[ c, c_2, c_3, c_4 \right] \\
\text{out} & \quad : \quad \text{solutions } sols_\quad \\
\text{post} & \quad : \quad \forall s \in sols_. \quad \forall e \in [ L \cdot q_0 = c, 0 = c_2 + 2 \cdot L \cdot c - \frac{L^2 \cdot q_0}{2}, \\
& \quad \quad \quad 0 = c_3, 0 = c_4 ]. \quad \text{Substitute } s \ e
\end{align*}
\]

The \textit{pre}-condition in \([\text{equation, system, linear}]\) must be true before starting execution of the sub-problem.
Example: subproblem

ad (2): Specification \([equation, \text{system, linear}]\)

\[
in : \text{equations } [L \cdot q_0 = c, 0 = c_2 + 2 \cdot L \cdot c - \frac{L^2 \cdot q_0}{2}, 0 = c_3, 0 = c_4],
\]

\[
\text{bound-vars } [c, c_2, c_3, c_4]
\]

\[
\text{pre} : |[L \cdot q_0 = c, 0 = c_2 + 2 \cdot L \cdot c - \frac{L^2 \cdot q_0}{2}, 0 = c_3, 0 = c_4]| =
\]

\[
\text{distinct } [c, c_2, c_3, c_4]
\]

\[
\text{out} : \text{solutions } \text{sols}_-
\]

\[
\text{post} : \forall s \in \text{sols}. \forall e \in [L \cdot q_0 = c, 0 = c_2 + 2 \cdot L \cdot c - \frac{L^2 \cdot q_0}{2}, 0 = c_3, 0 = c_4]. \text{Substitute } s \Rightarrow e
\]

The \text{pre-condition in } [equation, \text{system, linear}] \text{ must be true before starting execution of the sub-problem}
1. Introduction
   Plans for integrating three lines of work
   Motivation for the language: tutoring

2. Design of the programming language
   Syntax of the language
   Specification is a language element
   Contexts = environments \cup predicates
   We run into “patterns of specifications”

3. Preview and Summary
Program “bending line”

01  Script bendingLine
02    (l_::real) (q_::real) (v_::real) (b_::real=>real) (ct_:bool list) =
03    (LET
04      (funs_: bool list) =
05        (SubProblem (Bendingline,[bendingline,integrate],
06                                 [bendingline,integrate])
07            [real_ q_, real_real_ b_, real_ v_]);
08      (equs_:bool list) =
09        (SubProblem (Bendingline,[bendingline,setConstraints],
10                                 [bendingline,setConstraints])
11            [bools_ funs_, bools_ ct_, real_ l_]);
12    (sols_:bool list) =
13        (SubProblem (Real,[equation,system,linear],[
14                                 [bools_ equs_, reals_ [c,c_2,c_3,c_4]]));
15    B_ = Take (LAST funs_);
16    B_ = ((Substitute sols_) o
17        (Rewrite_Set_Inst [(bdv, v_)] make_ratpoly_in)) B_
18    IN B_)
Specification “bending line”

Specification for main-program

\[
\begin{align*}
\text{in} & : \text{load } q_0, \text{ length } L, \text{ bound}_\text{var} x \\
\text{pre} & : q_0 \text{ is integrable in } x \land L > 0 \\
\text{out} & : \text{bendline } y(x) \\
\text{post} : y(0) = 0 \land y'(0) = 0 \land V(0) = q_0 \cdot L \land M_b(L) = 0
\end{align*}
\]
Input to program “bending line”

Specification for main-program

\[
\begin{align*}
\text{in} & : \text{ load } q_0, \text{ length } L, \text{ bound}_\text{var} x \\
\text{pre} & : \ q_0 \text{ is}\_\text{integrable}\_\text{in} x \land L > 0 \\
\text{out} & : \text{ bendline } y(x) \\
\text{post} & : \ y(0) = 0 \land y'(0) = 0 \land V(0) = q_0.L \land M_b(L) = 0
\end{align*}
\]
Contexts in “bending line”

\[
\begin{align*}
02 & \quad env_0 = \{(l_1, L), (q_0, q_0), (v_1, x), (b_1, y x), \quad (rb_1, \{y 0 = 0 \land y' 0 = 0 \land V 0 = q_0 L \land M_b L = 0\}) \}\nonumber \\
& \quad pred_0 = \{q_0 \text{ is integrable in } x, \quad L > 0\}
\end{align*}
\]

\[ctxt_i = env_i \cup pred_i\]
Contexts in “bending line”

\[
\begin{align*}
02 \quad env_0 &= \{(l_\_ , L), (q_\_ , q_0), (v_\_ , x), (b_\_ , y x), \\
&\quad (rb_\_ , [y \ 0 = 0 \land y' \ 0 = 0 \land V \ 0 = q_0 \cdot L \land M_b \ L = 0])\} \\
\text{pred}_0 &= \{ q_0 \ is\_integrable\_in\ x, \ L > 0 \}
\end{align*}
\]

\[
ctx_{i} = env_{i} \cup pred_{i}
\]
Contexts in “bending line”

02  \( \text{env}_0 = \{(l_\_, \ L), (q_\_, q_0), (v_\_, x), (b_\_, y \ x), \ (rb_\_, [y \ 0 = 0 \land y' \ 0 = 0 \land V \ 0 = q_0.L \land M_b \ L = 0])\} \)

\( \text{pred}_0 = \{ q_0 \text{ is integrable in } x, \ L > 0 \} \)

07  \( \text{env}_1 = \text{env}_0 \cup \{(\text{funs}_\_, [Vx = c - q_0 \cdot x, M_b x = c_2 + c \cdot x - \frac{q_0}{2} \cdot x^2, \ 
\ y' x = c_3 - \frac{1}{E_l} \cdot (c_2 + \frac{c}{2} \cdot x^2 - \frac{q_0}{6} \cdot x^3), \ 
\ y x = c_4 + c_3 \cdot x - \frac{1}{E_l} \cdot (\frac{c_2}{2} \cdot x^2 + \frac{c}{6} \cdot x^3 - \frac{q_0}{24} \cdot x^4)\) \}

\( \text{pred}_1 = \text{pred}_0 \cup \{ q_0 \text{ is integrable in } x, \ y'' x = -\frac{1}{E_l} \cdot M_b x, \ 
\ y''' x = -\frac{1}{E_l} \cdot V x, \ y'' x = q_1 x \} \)

\( ctxt_i = env_i \cup pred_i \)
CTP-based languages
Florian Haftmann, Cezary Kaliszyk, Walther Neuper

Introduction
Three lines of work
Motivation: tutoring

Language
Syntax
Specification
Contexts
Spec. patterns

Preview and Summary

Contexts in “bending line”

\[ env_0 = \{(l_\_, L), (q_\_, q_0), (v_\_, x), (b_\_, y \ x) ,
          (rb_\_, [y 0 = 0 \land y' 0 = 0 \land V 0 = q_0 \cdot L \land M_b L = 0])\} \]

\[ pred_0 = \{ q_0 \textit{ is integrable in } x \ , \ L > 0 \} \]

\[ env_1 = env_0 \cup \{(funs_\_, [Vx = c - q_0 \cdot x , M_b x = c_2 + c \cdot x - \frac{q_0}{2} \cdot x^2
           y' x = c_3 - \frac{1}{EI} \cdot (c_2 + c \cdot x^2 - \frac{q_0}{6} \cdot x^3 ) ,
           y x = c_4 + c_3 \cdot x - \frac{1}{EI} \cdot (\frac{c_2}{2} \cdot x^2 + \frac{c}{6} \cdot x^3 - \frac{q_0}{24} \cdot x^4)
           y'' x = -\frac{1}{EI} \cdot V x , \ y''' x = q_1 x \} \]

\[ pred_1 = pred_0 \cup \{ q_0 \textit{ is integrable in } x , \ y'' x = -\frac{1}{EI} \cdot M_b x , \]

\[ y''' x = -\frac{1}{EI} \cdot V x , \ y''' x = q_1 x \}

\[ env_2 = env_1 \cup \{(equs_\_, [L \cdot q_0 = c , \ 0 = c_2 + 2 \cdot L \cdot c - \frac{L^2 \cdot q_0}{2} ,
           0 = c_4 , \ 0 = c_3])\} \]

\[ pred_2 = pred_1 \cup \{\text{pre-condition skipped , post-condition skipped} \}

\[ ctxt_i = env_i \cup pred_i \]
Contexts in “bending line”

02 \[ \text{env}_0 = \{ (l, L), (q, q_0), (v, x), (b, y) , (rb, [y \cdot 0 = 0 \land y' \cdot 0 = 0 \land V \cdot 0 = q_0 \cdot L \land M_b \cdot L = 0]) \} \]
\[ \text{pred}_0 = \{ q_0 \text{ is integrable in } x \land L > 0 \} \]

07 \[ \text{env}_1 = \text{env}_0 \cup \{(\text{funs}, [Vx = c - q_0 \cdot x, M_b \cdot x = c_2 + c \cdot x - \frac{q_0}{2} \cdot x^2 , 
     y' \cdot x = c_3 - \frac{1}{E_l} \cdot (c_2 + c \cdot x^2 - \frac{q_0}{6} \cdot x^3) , 
     y \cdot x = c_4 + c_3 \cdot x - \frac{1}{E_l} \cdot (c_2 \cdot x^2 + c \cdot x^3 - \frac{q_0}{24} \cdot x^4) \]) \}
\]
\[ \text{pred}_1 = \text{pred}_0 \cup \{ q_0 \text{ is integrable in } x \land y'' \cdot x = \frac{1}{E_l} \cdot M_b \cdot x , 
     y''' \cdot x = -\frac{1}{E_l} \cdot V \cdot x , y'' \cdot x = q_1 \cdot x \} \]

11 \[ \text{env}_2 = \text{env}_1 \cup \{(\text{equus}, [L \cdot q_0 = c , \ 0 = c_2 + 2 \cdot L \cdot c - \frac{L^2 \cdot q_0}{2} , 
     0 = c_4 , \ 0 = c_3]) \}
\]
\[ \text{pred}_2 = \text{pred}_1 \cup \{ \text{pre-condition skipped, post-condition skipped} \} \]

16 \[ \text{env}_5 = \text{env}_4 \cup \{(B, y \cdot x = (0) + (0) \cdot x - \frac{1}{E_l} \cdot (\frac{L \cdot q_0}{6} \cdot x^3 - \frac{q_0}{24} \cdot x^4) \} \]
\[ \text{pred}_5 = \text{pred}_4 \cup \{ \} \]

\[ \text{ctxt}_i = \text{env}_i \cup \text{pred}_i \]
**Contexts in “bending line”**

02 \( \text{env}_0 = \{ (l_\_ , L) , ( q_\_ , q_0) , ( v_\_ , x) , ( b_\_ , y x) , \\
\quad \text{and} (r b_\_ , [ y 0 = 0 \land y' 0 = 0 \land V 0 = q_0 \cdot L \land M_b L = 0 ]) \} \)

\( \text{pred}_0 = \{ q_0 \text{ is integrable in } x \ , \ L > 0 \} \)

07 \( \text{env}_1 = \text{env}_0 \cup \{ (\text{funs}_\_ , [ V x = c - q_0 \cdot x , M_b x = c_2 + c \cdot x - \frac{q_0}{2} \cdot x^2 \]
\quad y' x = c_3 - \frac{1}{EI} \cdot (c_2 + \frac{c}{2} \cdot x^2 - \frac{q_0}{6} \cdot x^3) ,
\quad y x = c_4 + c_3 \cdot x - \frac{1}{EI} \cdot (\frac{c_2}{2} \cdot x^2 + \frac{c}{3} \cdot x^3 - \frac{q_0}{24} \cdot x^4) \}
\)

\( \text{pred}_1 = \text{pred}_0 \cup \{ q_0 \text{ is integrable in } x \ , \ y'' x = -\frac{1}{EI} \cdot M_b x \ ,
\quad y''' x = -\frac{1}{EI} \cdot V x , \ y'' x = q_1 x \} \)

11 \( \text{env}_2 = \text{env}_1 \cup \{ (\text{equ}_\_ , [ L \cdot q_0 = c , 0 = c_2 + 2 \cdot L \cdot c - \frac{L^2 \cdot q_0}{2} ,
\quad 0 = c_4 , 0 = c_3 ]) \} \)

\( \text{pred}_2 = \text{pred}_1 \cup \{ \text{pre-condition skipped} , \text{post-condition skipped} \} \)

16 \( \text{env}_5 = \text{env}_4 \cup \{ (B_\_ , y x = (0) + (0) \cdot x - \frac{1}{EI} \cdot (\frac{-L^2 \cdot q_0}{2}) \cdot x^2 +
\quad \frac{(L \cdot q_0)^2}{6} \cdot x^3 - \frac{q_0}{24} \cdot x^4) \} \)

\( \text{pred}_5 = \text{pred}_4 \cup \{ \} \)

17 \( \text{env}_6 = \text{env}_5 \cup \{ (B_\_ , y x = \frac{q_0 \cdot L^2}{4 \cdot EI} \cdot x^2 - \frac{q_0}{6 \cdot EI} \cdot x^3 + \frac{q_0}{24 \cdot EI} \cdot x^4) \}
\)

\( \text{pred}_6 = \text{pred}_5 \cup \{ \neg (y x \text{ contains } \{ c , c_2 , c_3 , c_4 \}) \} \)

\( \text{ctxt}_i = \text{env}_i \cup \text{pred}_i \)
Contexts in “bending line”

02 \( \text{env}_0 = \{(l_\_, L), (q_\_, q_0), (v_\_, x), (b_\_, y\ x), (\text{rb}_\_, [y\ 0 = 0 \land y'\ 0 = 0 \land V\ 0 = q_0\cdot L \land M_b\ L = 0])\} \)

\( \text{pred}_0 = \{ q_0 \text{ is_integrable_in } x, L > 0 \} \)

07 \( \text{env}_1 = \text{env}_0 \cup \{(\text{funs}_\_, [\forall x = c - q_0 \cdot x, M_b\ x = c_2 + c \cdot x - \frac{q_0}{2} \cdot x^2, y'\ x = c_3 - \frac{1}{EI} \cdot (c_2 + \frac{c}{2} \cdot x^2 - \frac{q_0}{6} \cdot x^3), y\ x = c_4 + c_3 \cdot x - \frac{1}{EI} \cdot (\frac{c_2}{2} \cdot x^2 + \frac{c}{6} \cdot x^3 - \frac{q_0}{24} \cdot x^4)\}, \)

\( \text{pred}_1 = \text{pred}_0 \cup \{ q_0 \text{ is_integrable_in } x, y''\ x = -\frac{1}{EI} \cdot M_b\ x, y'''\ x = -\frac{1}{EI} \cdot V\ x, y''\ x = q_1\ x \} \)

11 \( \text{env}_2 = \text{env}_1 \cup \{(\text{equs}_\_, [L \cdot q_0 = c, 0 = c_2 + 2 \cdot L \cdot c - \frac{L^2 \cdot q_0}{2}, 0 = c_4, 0 = c_3])\} \)

\( \text{pred}_2 = \text{pred}_1 \cup \{\text{pre-condition skipped, post-condition skipped}\} \)

16 \( \text{env}_5 = \text{env}_4 \cup \{(B_\_, y\ x = (0) + (0) \cdot x - \frac{1}{EI} \cdot (\frac{-L^2 \cdot q_0}{2} \cdot x^2 + \frac{(L \cdot q_0)}{6} \cdot x^3 - \frac{q_0}{24} \cdot x^4))\} \)

\( \text{pred}_5 = \text{pred}_4 \cup \{\} \)

17 \( \text{env}_6 = \text{env}_5 \cup \{(B_\_, y\ x = \frac{q_0 \cdot L^2}{4 \cdot EI} \cdot x^2 - \frac{q_0 \cdot L}{6 \cdot EI} \cdot x^3 + \frac{q_0}{24 \cdot EI} \cdot x^4)\} \)

\( \text{pred}_6 = \text{pred}_5 \cup \{\neg(y\ x \text{ contains } \{c, c_2, c_3, c_4\})\} \)

\( \text{ctxt}_i = \text{env}_i \cup \text{pred}_i \)
Introduction

Plans for integrating three lines of work

Motivation for the language: tutoring

Design of the programming language

Syntax of the language

Specification is a language element

Contexts = environments \( \cup \) predicates

We run into “patterns of specifications”
Example “bending line”

Specification for main-program

\[
\begin{align*}
\text{in} &: \quad \text{load } q_0, \text{ length } L, \text{ bound_var } x \\
\text{pre} &: \quad q_0 \text{ is integrable in } x \land L > 0 \\
\text{out} &: \quad \text{bendline } y(x) \\
\text{post} &: \quad y(0) = 0 \land y'(0) = 0 \land V(0) = q_0 L \land M_b(L) = 0
\end{align*}
\]

However, program “bending line” works on a class of examples!
We have the program’s actual arguments for instatiation of a “specification-pattern” to a specification.
Example “bending line”

Specification for main-program

\[
\begin{align*}
\text{in} &: \quad \text{load } q_0, \text{ length } L, \text{ bound}_\text{var} x \\
\text{pre} &: \quad q_0 \text{ isintegrable}_\text{in} x \land L > 0 \\
\text{out} &: \quad \text{bendline } y(x) \\
\text{post} &: \quad y(0) = 0 \land y'(0) = 0 \land V(0) = q_0.L \land M_b(L) = 0
\end{align*}
\]

However, program “bending line” works on a class of examples!

We have the program’s actual arguments for instantiation of a “specification-pattern” to a specification.
Example “bending line”

Specification for main-program

\[
\begin{align*}
\text{in} & : \text{ load } q_0, \text{ length } L, \text{ bound } \_\text{var } x \\
\text{pre} & : \ q_0 \ \text{is\_integrable\_in } \_\text{var } x \land L > 0 \\
\text{out} & : \text{ bendline } y(x) \\
\text{post} & : \ y(0) = 0 \land y'(0) = 0 \land V(0) = q_0.L \land M_b(L) = 0
\end{align*}
\]

However, program “bending line” works on a class of examples!
We have the program’s actual arguments for instatiation of a “specification-pattern” to a specification.
**Specification - pattern ?!**

Program + environment

```
program
Script B (q, L, v, Cs) =
  LET
    funs = Subproblem (thy, pbl, met) q, L, v
    equs = Subproblem ...
    sols = Subproblem ...
  B = Take (LAST funs)
  B = ((Substitute sols)@ (Rewrite_Set poly)) B
IN B
```

output
```
y(x) = (q.L^2)/(4.EI) . x^2 - (q.L)/(6.EI) . x^3 + q /(24.EI) . x^4
```

Specification-pattern + environment

**In:** function q, Length L
**pre:** q is_integrable
**post:** y(0)=0 Δ y'(0)=0
Δ V(0)=q.L
Δ M_b(L)=0
Specification-pattern for example-class “bending line”

\[
\begin{align*}
\text{in} : & \quad \text{load } q\_, \text{ length } l\_, \text{ bound\_var } bdv\_
\\
\text{pre} : & \quad q\_ \text{ is\_integrable\_in } bdv\_ \land l\_ > 0
\\
\text{out} : & \quad \text{bendline } b\_
\\
\text{post} : & \quad \bigwedge_{i=1}^{4} \text{Predicate}_i(b\_, b\_\ ', M_b, V, \text{load}, l\_, bdv\_) \land\\
& \quad b\_''(bdv\_) = -\frac{1}{EI} \cdot M_b(bdv\_) \land\\
& \quad b\_'''(bdv\_) = -\frac{1}{EI} \cdot V(bdv\_) \land\\
& \quad b\_''\prime(v)(bdv\_) = \frac{1}{EI} \cdot q(bdv\_) \land\\
& \quad \neg (b\_ \text{ bdv\_ contains } \{c, c_2, c_3, c_4\})
\end{align*}
\]
Preview to the future

- We want an elegant and lean programming language exploiting the power of CTP-technology.

- This language shall be handy for applications of math. Since the Lucas-interpreter generates tutoring mechanically from programs written in this language, we get tutoring for free.

- Pushing the interpretation into the ML-compiler (?) would make the language efficient for production.

- But first, the language must be a good language . . .
Preview to the future

- We want an elegant and lean programming language exploiting the power of CTP-technology.

- This language shall be handy for applications of math. Since the Lucas-interpreter generates tutoring mechanically from programs written in this language, we get tutoring for free.

- Pushing the interpretation into the ML-compiler (?) would make the language efficient for production.

- But first, the language must be a good language . . .
Preview to the future

- We want an elegant and lean programming language exploiting the power of CTP-technology.

- This language shall be handy for applications of math. Since the Lucas-interpreter generates tutoring mechanically from programs written in this language, we get tutoring for free.

- Pushing the interpretation into the ML-compiler (?) would make the language efficient for production.

- But first, the language must be a good language …
Preview to the future

- We want an elegant and lean programming language exploiting the power of CTP-technology.

- This language shall be handy for applications of math. Since the Lucas-interpreter generates tutoring mechanically from programs written in this language, we get tutoring for free.

- Pushing the interpretation into the ML-compiler (?) would make the language efficient for production.

- But first, the language must be a good language . . .
Preview to the future

- We want an elegant and lean programming language exploiting the power of CTP-technology.

- This language shall be handy for applications of math. Since the Lucas-interpreter generates tutoring mechanically from programs written in this language, we get tutoring for free.

- Pushing the interpretation into the ML-compiler (?) would make the language efficient for production.

- But first, the language must be a good language …
Preview to the future

- We want an elegant and lean programming language exploiting the power of CTP-technology.

- This language shall be handy for applications of math. Since the Lucas-interpreter generates tutoring mechanically from programs written in this language, we get tutoring for free.

- Pushing the interpretation into the ML-compiler (?) would make the language efficient for production.

- But first, the language must be a good language . . .
Summary

What are good/bad features in the language design?

- Syntax combines: Isabelle/HOL + tactics + tacticals
  - is the mix reasonable?
- Specification is a language element
  - Implementing subproblems mixes programming and proving — workflow?
- Contexts = environments \( \cup \) predicates
  - This allows automatic handling of partiality!
  - This supports checking pre- and post-conditions!
- We run into “patterns of specifications”
  - What elements instantiate to specifications, \( \bigwedge_{i=1}^{4} \text{Predicate}_i(\text{vars}) \)?
Summary

What are good/bad features in the language design?

- **Syntax combines:** Isabelle/HOL + tactics + tacticals
  - is the mix reasonable?
- **Specification is a language element**
  - Implementing subproblems mixes programming and proving — workflow?
- **Contexts = environments \( \cup \) predicates
  - This allows automatic handling of partiality!
  - This supports checking pre- and post-conditions!
- **We run into “patterns of specifications”**
  - What elements instantiate to specifications, \( \bigwedge_{i=1}^{4} \) \( \text{Predicate}_i(v) \)?
What are good/bad features in the language design?

- Syntax combines: Isabelle/HOL + tactics + tacticals
  - is the mix reasonable?

- Specification is a language element
  - Implementing subproblems mixes programming and proving — workflow?

- Contexts = environments $\cup$ predicates
  - This allows automatic handling of partiality!
  - This supports checking pre- and post-conditions!

- We run into “patterns of specifications”
  - What elements instantiate to specifications, $\land_{i=1}^{4} \text{Predicate}_i(vars)$?
Summary

What are good/bad features in the language design?

- Syntax combines: Isabelle/HOL + tactics + tacticals
  - is the mix reasonable?
- Specification is a language element
  - Implementing subproblems mixes programming and proving — workflow?
- Contexts $\equiv$ environments $\cup$ predicates
  - This allows automatic handling of partiality!
  - This supports checking pre- and post-conditions!
- We run into “patterns of specifications”
  - What elements instantiate to specifications, $\bigwedge_{i=1}^{4} \text{Predicate}_i(\text{vars})$?
Summary

What are good/bad features in the language design?

- Syntax combines: Isabelle/HOL + tactics + tacticals
  - is the mix reasonable?
- Specification is a language element
  - Implementing subproblems mixes programming and proving — workflow?
- Contexts = environments \( \cup \) predicates
  - This allows automatic handling of partiality!
  - This supports checking pre- and post-conditions!
- We run into “patterns of specifications”
  - What elements instantiate to specifications, \( \bigwedge_{i=1}^{4} Predicate_i(vars) \)?
Summary

What are good/bad features in the language design?

- Syntax combines: Isabelle/HOL + tactics + tacticals
  - is the mix reasonable ?????
- Specification is a language element
  - Implementing subproblems mixes programming and proving — workflow ?????
- Contexts = environments ∪ predicates
  - This allows automatic handling of partiality!
  - This supports checking pre- and post-conditions!
- We run into “patterns of specifications”
  - What elements instantiate to specifications, $\bigwedge_{i=1}^{4} \text{Predicate}_i(\text{vars})$ ????

Thank you! We kindly ask for your opinion.