Early Experience:
Teaching the Basics of Functional Language Design with a Language Type Checker

Matteo Cimini (UMass Lowell)
TFP 2019
13th June, Vancouver, BC, Canada
Background
Graduate course “Design of Programming Languages”

Major learning outcomes:

1) Familiarize with the basic tools of PL theory.

2) Use the tools of PL theory to the modeling of type sound toy functional languages
What PL tools for defining languages?

**BNF grammars**

Types
\[ T ::= \text{Int} \mid T \rightarrow T \mid \text{List} \ T \]

Expressions
\[ e ::= n \mid x \mid \lambda x.e \mid e \ e \mid \text{nil} \mid \text{cons} \ e \ e \mid \text{head} \ e \mid \text{tail} \ e \mid \text{error} \]

Values
\[ v ::= n \mid \lambda x.e \mid \text{nil} \mid \text{cons} \ v \ v \]

Errors
\[ e \quad ::= \quad \text{error} \]

Contexts
\[ E ::= E \ e \mid v \ E \mid \text{cons} \ E \ e \mid \text{cons} \ v \ E \mid \text{head} \ E \mid \text{tail} \ E \]

**Inference rules**

**Type System**

\[ \Gamma \vdash e_1 : T \quad \Gamma \vdash e_2 : \text{List} \ T \]

\[ \quad \Gamma \vdash \text{cons} \ e_1 \ e_2 : \text{List} \ T \]

\[ \Gamma \vdash e : \text{List} \ T \]

\[ \quad \Gamma \vdash \text{head} \ e : T \]

...
Type Soundness, in one slide

What is type soundness? You can trust your types!

fun average(n1:Int, n2:Int) {
    return (n1 + n2) / 2
}

type soundness prevents:

unspecified behav.    head [ 1+2 , 4, 5] stuck!
inconsistent behav.   int_to_string(3*4) → "3" * "4"
TypeSoundnessCertifier

A tool for type checking languages and certifying their soundness

https://github.com/mcimini/TypeSoundnessCertifier

A Type System for Language Design

PL Design:
“classify operators in intro forms, elimination forms, errors, etc”

<table>
<thead>
<tr>
<th>intr</th>
<th>intr →</th>
<th>elim →</th>
<th>intr List</th>
<th>intro List</th>
<th>elim List</th>
<th>elim List</th>
<th>error</th>
</tr>
</thead>
<tbody>
<tr>
<td>n</td>
<td>x</td>
<td>λx.e</td>
<td>e</td>
<td>e</td>
<td>nil</td>
<td>cons e e</td>
<td>head e</td>
</tr>
</tbody>
</table>
A Type System for Language Design

PL Design:
“elimination forms manipulate values of some specific data type”

Values
\[ v ::= n | \lambda x.e | \text{nil} | \text{cons} \ v \ v \]

Contexts
\[ E ::= \text{app} \ E \ e | \text{app} \ v \ E | \text{cons} \ E \ e | \text{head} \ E | \text{tail} \ E \]

Intro \rightarrow must be in Values
\[ \text{app} \ (\lambda x.e) \ v \rightarrow e[v/x] \]

principal argument

Elim \rightarrow
\[ \text{app} \ v \ E \]

head nil \rightarrow error
head (\text{cons} \ v_1 \ v_2) \rightarrow v_1

it works for other examples:
Expression E ::= x | (abs T (x)E) | (app E E) | emptyList | (cons E E) | (head E) | (tail E) | myError
Type T ::= (arrow T T) | (list T)
Value V ::= (abs T E) | emptyList | (cons V V)
Error ::= myError
Context C ::= [] | (app C E) | (app V C) | (cons C E) | (cons V C) | (head C) | (tail C)

Gamma |- (abs T1 E) : (arrow T1 T2) <= Gamma, x : T1 |- E : T2.
Gamma |- emptyList : (list T).
Gamma |- (cons E1 E2) : (list T) <= Gamma |- E1 : T /\ Gamma |- E2 : (list T).
Gamma |- (app E1 E2) : T2 <= Gamma |- E1 : (arrow T1 T2) /\ Gamma |- E2 : T1.
Gamma |- (head E) : T <= Gamma |- E1 : (arrow T1 T2) /\ Gamma |- E2 : T1.
Gamma |- (tail E) : (list T) <= Gamma |- E : (list T).
Gamma |- myError : T.

(app (abs T E) V) --> E[V/x].
(head emptyList) --> myError.
(head (cons V1 V2)) --> V1.
(tail emptyList) --> myError.
(tail (cons V1 V2)) --> V2.

Spotting design mistakes

“app is an elimination form but its principal argument is not an evaluation context, hence type soundness does not hold”
Spotting design mistakes

Expression E ::= x | (abs T (x)E) | (app E E) | emptyList | (cons E E) | (head E) | (tail E) | myError

Type T ::= (arrow T T) | (list T)

Value V ::= (abs T E) | emptyList | (cons V V)

Error ::= myError

Context C ::= [] | (app C E) | (app V C) | (cons C E) | (cons V C) | (head C) | (tail C)

\[\Gamma \vdash (\text{abs } T_1 E) : (\text{arrow } T_1 T_2) \iff \Gamma, x : T_1 |- E : T_2.\]
\[\Gamma |- \text{emptyList} : (\text{list } T).\]
\[\Gamma |- (\text{cons } E_1 E_2) : (\text{list } T) \iff \Gamma |- E_1 : T /\ Gamma |- E_2 : (\text{list } T).\]
\[\Gamma |- (\text{app } E_1 E_2) : T_2 \iff \Gamma |- E_1 : (\text{arrow } T_1 T_2) /\ Gamma |- E_2 : T_1.\]
\[\Gamma |- (\text{head } E) : T \iff \Gamma |- E_1 : (\text{list } T).\]
\[\Gamma |- (\text{tail } E) : (\text{list } T) \iff \Gamma |- E : (\text{list } T).\]
\[\Gamma |- \text{myError} : T.\]

\[(\text{app } (\text{abs } T E) V) \to E[V/x].\]
\[(\text{head } \text{emptyList}) \to \text{myError}.\]
\[(\text{head } (\text{cons } V_1 V_2)) \to V_1.\]
\[(\text{tail } \text{emptyList}) \to \text{myError}.\]
\[(\text{tail } (\text{cons } V_1 V_2)) \to V_2.\]
A language type checker is suitable for:

- **Teaching language design principles**
- **Collecting statistics on students’ difficulties and inform best practices in teaching**

“Operator `head` is an elimination form for type `list` but does not have a reduction rule for handling one of the values of `list`: value `emptyList`”
Let’s resize the scope for today...

Functional Language Design:
Only pure functional languages  \[ e \longrightarrow e \]  

Basics:
Covered principles are too simple to be applied to non-trivial functional languages.

Early Experience:
only 1 instance of the course, with low enrollment (statistically not significant)
Course “Design of Programming Languages” (In Class)

- Part of concern was 7 weeks (half semester)

- Benjamin Pierce’s TAPL book (Initial chapters)

- 1 lecture devoted to TypeSoundnessCertifier
Course “Design of Programming Languages”
(Outside of the Class)

- Self-practice with TypeSoundnessCertifier

- Exercise TypeSoundnessCertifier with:

  - **length** \([1, 2, 3, 4] = 4\)
  - **reverseRange** \(4 = [4, 3, 2, 1, 0]\)
  - **map** (standard of functional programming)

**length:**

Expression \(E\) ::= \(\text{zero} \mid (\text{succ } E) \mid (\text{emptyList }) \mid (\text{cons } E \ E) \mid (\text{length } E)\)

Type \(T\) ::= \(\text{int} \mid (\text{list } T)\)

Value \(V\) ::= \(\text{zero} \mid (\text{succ } V) \mid (\text{emptyList } ) \mid (\text{cons } V1 \ V2)\)

Context \(C\) ::= \([] \mid (\text{succ } C) \mid (\text{cons } C \ e) \mid (\text{cons } v \ C) \mid (\text{length } C)\)

\[
\Gamma \vdash (\text{length } E) : \text{int} \iff \Gamma \vdash E : (\text{list } T).
\]

\(\text{length emptyList} \rightarrow \text{zero}.
\(\text{length (cons } V1 \ V2) \rightarrow (\text{succ } (\text{length } V2)).\)
Course “Design of Programming Languages”
(Outside of the Class)

- Self-practice with TypeSoundnessCertifier

- Exercise TypeSoundnessCertifier with:
  
  - **length** \[1, 2, 3, 4\] = 4
  - **reverseRange** 4 = \[4, 3, 2, 1, 0\]
  - **map** (standard of functional programming)

map:

Expression \( E ::= x \mid (\text{abs } T (x)E) \mid (\text{app } E E) \mid \text{zero} \mid (\text{succ } E) \mid \text{emptyList} \mid (\text{cons } E E) \mid (\text{map } E E) \)

Type \( T ::= \text{int} \mid (\text{arrow } T T) \mid (\text{list } T) \)

Value \( V ::= (\text{abs } T E) \mid \text{zero} \mid (\text{succ } V) \mid \text{emptyList} \mid (\text{cons } V1 V2) \)

Context \( C ::= [] \mid (\text{succ } C) \mid (\text{cons } C e) \mid (\text{cons } v C) \mid (\text{app } C e) \mid (\text{app } v C) \mid (\text{map } C e) \mid (\text{map } v C) \)

Gamma |- (map E1 E2) : (list T2) <= Gamma |- E1 : (list T1) \(\setminus\) Gamma |- E2 : (arrow T1 T2).

(map emptyList V) --> emptyList.

(map (cons V1 V2) V3) --> (cons (app V3 V1) (map V2 V3)).
Course “Design of Programming Languages” (Evaluation)

- Modeling a language with TypeSoundnessCertifier in front of the instructor

- 20 minutes to complete the task

- Invoke TypeSoundnessCertifier at will

- Exam Language:

  extend a language with bool, functions and lists with:

  • option types: none, some e, get e.
  • filterOpt isEven [1,2,3,4,5] = [none, some 2, none, some 4, none]
Course “Design of Programming Languages” (Evaluation)

- Exam Language: extend a language with bool, functions and lists with:
  
  • **option types:** none, some \( e \), get \( e \).
  
  • **filterOpt** `isEven [1,2,3,4,5] = [none, some 2, none, some 4, none]` 

Expression \( E ::= \ldots | \text{none} | (\text{some } E) | (\text{get } E) | (\text{filterOpt } E \ E) \)

Type \( T ::= \ldots | (\text{option } T) \)

Value \( V ::= \text{none} | (\text{some } V) \)

Error ::= \text{myerror}

Context \( C ::= \ldots | (\text{some } C) | (\text{get } C) | (\text{filterOpt } C \ E) | (\text{filterOpt } V \ C) \)

\[
\Gamma \vdash \text{none} : (\text{option } T).
\]

\[
\Gamma \vdash (\text{some } E) : (\text{option } T) \iff \Gamma \vdash E : T.
\]

\[
\Gamma \vdash (\text{get } E) : T \iff \Gamma \vdash E : (\text{option } T).
\]

\[
(\text{get } \text{none}) \rightarrow \text{myerror}.
\]

\[
(\text{get } (\text{some } V)) \rightarrow V.
\]

\[
\Gamma \vdash (\text{filterOpt } E1 \ E2) : (\text{list } (\text{option } T)) \iff \Gamma \vdash E1 : (\text{list } T)
\]

\[
\text{[]} \quad \Gamma \vdash E2 : (\text{arrow } T \text{ bool}).
\]

\[
(\text{filterOpt } \text{emptyList } V) \rightarrow \text{emptyList}.
\]

\[
(\text{filterOpt } (\text{cons } V1 \ V2 \ V3)) \rightarrow (\text{cons } (\text{if } (\text{app } V3 \ V1) (\text{some } V1) \ \text{none}) (\text{filterOpt } V2 \ V3)).
\]
Results

Total: 11 students (NOT statistically significant)

• 6 out of 11 completed the task:
  • 3 students at the 3rd attempt
  • 1 student at the 4th attempt
  • 1 student at the 6th attempt
  • 1 student at 1st attempt

  avg: 3.333, avg minus outlier: 3.8

• 5 out of 11 forgot an evaluation context
• 4 out of 11 have mistaken a type preserving rule
Results (survey)

*The TypeSoundnessCertifier tool helped me model languages* correctly. 3 3

I have a better understanding on how to model languages thanks to the TypeSoundnessCertifier tool. 3 2 1

I modeled languages with confidence for trusting that the TypeSoundnessCertifier tool would catch my mistakes. 2 4

Feedback from the TypeSoundnessCertifier tool helped me do better during the evaluation. 2 3 1

[*] We mean those languages that are in the scope described in this paper and not, generally, any language.

SA = strongly agree
SWA = somewhat agree
Neith = neither agree nor disagree
SD = strongly disagree
SWD = somewhat disagree
NA = not available
/ don’t remember
Conclusions

There is an encouraging indication that language type checkers are worth exploring for teaching the basics of functional language design.

Future Work

• Include subtyping in the evaluation
• Extend the tool to languages with state
• Automate grading
• Conduct more, and MORE experiments.

Thank you!

https://github.com/mcimini/TypeSoundnessCertifier