Automatic Type Annotations

Ambrose Bonnaire-Sergeant

Ph.D. Qualifying exam
A Story about Annotating...
“How do we convert 600k lines of JavaScript to TypeScript, in an actively developed app?”
Option 1
- “Gradual” typing

Option 2
- Stop and sprint!
Chose: Option 2
- Annual 48 hour hackathon
- No devs working on core product for 48 hours!
CTO’s thoughts:

June 9th

Ben Dilts  8:53 AM
I think there's a zero percent chance you can get all our Closure code successfully building as Typescript in a matter of three days.

Six of us engineers decided to try anyway.
<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>Can do</td>
</tr>
<tr>
<td>3</td>
<td>Can do</td>
</tr>
<tr>
<td>4</td>
<td>Can't do</td>
</tr>
<tr>
<td>5</td>
<td>Can't do</td>
</tr>
<tr>
<td>6</td>
<td>Can't do</td>
</tr>
<tr>
<td>7</td>
<td>Can't do</td>
</tr>
<tr>
<td>8</td>
<td>Can't do</td>
</tr>
<tr>
<td>9</td>
<td>Can't do</td>
</tr>
<tr>
<td>10</td>
<td>Can't do</td>
</tr>
<tr>
<td>11</td>
<td>Can do</td>
</tr>
<tr>
<td>12</td>
<td>Can't do</td>
</tr>
<tr>
<td>13</td>
<td>Can't do</td>
</tr>
<tr>
<td>14</td>
<td>Can't do</td>
</tr>
<tr>
<td>15</td>
<td>Can't do</td>
</tr>
<tr>
<td>16</td>
<td>Can't do</td>
</tr>
<tr>
<td>17</td>
<td>Can do</td>
</tr>
</tbody>
</table>

**Legend**
- Ready
- In Progress
- Done
Clutz - Closure to TypeScript Declarations (.d.ts) generator.

This project uses the Closure Compiler to parse Closure-style JSDoc type annotations from ES5/ES2015 code, and generates a suitable TypeScript type definition file (.d.ts) for the exported API.

Gents - Closure to TypeScript converter

This repository also hosts gents - tool that generates TypeScript code out of Closure annotated .js. We host it in this repo together with clutz because they both wrap Closure Compiler to get the type information. As such gents shares clutz restriction that it only accepts code that is valid well-typed Closure JavaScript.
CONVERTING 600K LINES TO TYPESCRIPT IN 72 HOURS

© November 16, 2017  Paul Draper & Ryan Stringham  15 Comments

Lucidchart
Takeaways

Companies will **heavily invest** in transitioning to typed languages

Translation to typed languages can be **partially automated**
Background for My Research

Clojure
(untyped)

Typed Clojure
(typed sister-language)

Manually annotate

clojure.spec
(contract system)
My research objective

Create *effective tools* to ease the transition to annotated target languages.
Approach

1. Understand target language theory
2. Understand target language practice
3. Compare our tool with similar tools
This Talk

1. Understand target language theory
   • **Quals question**: spec theory
   • Audience questions

2. Understand target language practice
   • **Quals question**: spec practice
   • Audience questions

3. Compare with similar tools
   • **Quals question**: perf analysis
   • Audience questions
1. Understand target language theory
Understand target language theory

Clojure  Tool-assisted

->  clojure.spec

Clojure  Tool-assisted

->  clojure.spec
Quals

Question

(spec theory)

1. Formulate a formal model for clojure.spec
2. Implement model in PLT Redex
3. Formulate consistency property between contracted and uncontracted execution
   (a) Test property in Redex
Proxy-based

Tag-test

Generative-testing based
<table>
<thead>
<tr>
<th>Model</th>
<th>Features</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\lambda c$</td>
<td><img src="image1" alt="Diagram" /></td>
</tr>
<tr>
<td>$\lambda c_s$</td>
<td><img src="image2" alt="Diagram" /></td>
</tr>
<tr>
<td>$\lambda c^f$</td>
<td><img src="image3" alt="Diagram" /></td>
</tr>
<tr>
<td>$\lambda c^f_s$</td>
<td><img src="image4" alt="Diagram" /></td>
</tr>
</tbody>
</table>
E ::= C | L | X
   | (E E ...)
   | (if E E E)
C ::= N | O | B | nil | H | ERR
X ::= variable-not-otherwise-mentioned
ERR ::= (error any any ...)
L ::= (fn [X ...] E) | (fn X [X ...] E)
NONFNV ::= B | H | nil | N
V ::= O | L | NONFNV
V^e ::= V | ERR
H ::= (HashMap (V V) ...)
B ::= true | false
N ::= number
Z ::= natural
O ::= P
   | inc | dec
   | + | * | dissoc
   | assoc | get
P ::= zero? | number? | boolean? | nil?
C ::= [] | (if C E E) | (V ... C E ...)
Spec ERROR:
expected nil, found 1

Consistent?
Consistent?
Consistent?
Break for Questions
1. Understand target language theory
2. Understand target language practice
3. Build useful and performant tools based on this knowledge
Our tool’s output → Updated code
1. Examine clojure.spec usage in real-world code bases
2. Analyze frequency and precision of higher-order function annotations
Searches say generative testing is not that popular

\[ s/\text{fspec} = \text{Gen testing-based function contract} \]
Different function contracts rarely occur in the same project

<table>
<thead>
<tr>
<th>Search terms</th>
<th># Projects</th>
<th>Ratio of (&lt;\text{Tag-test}&gt;:&lt;\text{Gen-testing}&gt;) function specs</th>
</tr>
</thead>
<tbody>
<tr>
<td>Search 1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>clojure.spec &amp;&amp; fspec</td>
<td>18</td>
<td>3:79</td>
</tr>
<tr>
<td>Search 2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>clojure.spec &amp;&amp; ifn?</td>
<td>17</td>
<td>188:0</td>
</tr>
</tbody>
</table>
Our tool's output

(ifn?
  f
)

Gen arg
Gen arg
Gen arg
f
f
f
Break for Questions
1. Understand target language theory
2. Understand target language practice
3. Compare with similar tools
3. Compare with similar tools

Why is this useful?

- Ensure performance of our tool is reasonable compared to existing tooling
- Better understand tradeoffs we made by comparing with other approaches
Quals

Question

(perf analysis)

1. Compare time+space complexity vs. Daikon
2. Can we reuse Daikon’s optimizations?
3. How expressive are Daikon annotations?
Our Tool’s Type Reconstruction

<table>
<thead>
<tr>
<th>$t=0$</th>
<th>Initial</th>
</tr>
</thead>
<tbody>
<tr>
<td>$t=1$</td>
<td>\textit{Observed} \newline \text{\textbf{x is an int}}</td>
</tr>
<tr>
<td>$t=2$</td>
<td>\textit{Observed} \newline \text{\textbf{y is a bool}}</td>
</tr>
<tr>
<td>$t=3$</td>
<td>\textit{Observed} \newline \text{\textbf{x is a bool}}</td>
</tr>
<tr>
<td>$t=4$</td>
<td>\textit{Observed} \newline \text{\textbf{x is a symbol}}</td>
</tr>
</tbody>
</table>
# Daikon’s Type Reconstruction

| $t=0$ | **Initial** | 
|-------|----|---|---|
| even(x) | even(y) | even(z) |
| odd(x)  | odd(y)  | odd(z)  |

| $t=1$ | **Observed** | $x = 3$ | 
|-------|------------|------|---|---|
| even(x) | even(y) | even(z) |
| odd(x)  | odd(y)  | odd(z)  |

| $t=2$ | **Observed** | $y = 4$ | 
|-------|------------|------|---|---|
| even(x) | even(y) | even(z) |
| odd(x)  | odd(y)  | odd(z)  |
Processing traces on-line

Dynamic Instrumentation

Type Reconstruction
Break for Questions
Recap

I want to create *effective tools* to ease the transition to annotated target languages.

Approach:
1. Understand target language theory
2. Understand target language practice
3. Compare our tool with similar tools
Thanks

“Intertwined arrows” designed by Freepik