Types and Analysis for Scripting Languages (Part 2)

Peter Thiemann

Universität Freiburg, Germany

Mini Course, Tallinn, Estland, 25.-27.2.2008

Towards a Type System for Analyzing JavaScript Programs

ESOP 2005

The Type System $\mathcal{DTJ}$ for JavaScript

- Rejects actual runtime errors, *e.g.*,
  - use of a non-function as a function
  - use of `undefined` as an object
- Detects suspicious coercions, *e.g.*,
  - `number` $\rightarrow$ `object`.
  - `string` $\rightarrow$ `object`.
- Provides guarantees about properties, *e.g.*, what is the string value of the property?

Difficulties in Designing $\mathcal{DTJ}$

- `obj.m() = obj["m"]()`
  need singleton types and first-class labels
- `function (obj, x) { return obj[x] }`
  access (or update) an unknown property
- different roles of functions (methods, constructors)
- variable arity, positional access
- type conversion (in particular, what cannot be converted)
- unions and subtyping
- (wrapped) numbers, strings, booleans, and functions with properties

Types for $\mathcal{DTJ}$

Discriminative sum types:
Each type τ is composed of type summands φ

τ	$::=$	α		type variable
	$\mid$	$\varphi_i + \tau$		internal type summand
		where τ `lacks` i	$i \in \{\bot, u, s, b, n, o\}$	constraint
	$\mid$	$\emptyset$		closed type

Type Summands

$$
\begin{array}{lcl}
\varphi_\bot & ::= & \texttt{Undefined} \\
\varphi_u & ::= & \texttt{Null} \\
\varphi_b & ::= & \texttt{Bool}(\xi_b) \\
\varphi_s & ::= & \texttt{String}(\xi_s) \\
\varphi_n & ::= & \texttt{Number}(\xi_n) \\
\varphi_o & ::= & \texttt{Obj}(\omega)(\varrho) \\
\varphi_f & ::= & \texttt{Fun}(\texttt{this}:\tau\ ;\ \varrho \to \tau) \\
\omega & ::= & \sum_{i \in T, T \subseteq \{b,s,n,f,\bot\}} \varphi_i
\end{array}
$$

- `bool`, `string`, and `number` types are indexed with ξ_i
- functions are special objects
- function arguments and property descriptions are *row types* ϱ

Type Indices

$$\begin{array}{rcl} \xi_b & ::= & \texttt{false} \mid \texttt{true} \mid \top \mid \emptyset \mid \psi_b \\ \xi_s & ::= & \mathit{str} \mid \top \mid \emptyset \mid \psi_s \\ \xi_n & ::= & \mathit{num} \mid \top \mid \emptyset \mid \psi_n \end{array}$$

- Index is either a constant, a variable ψ_i, empty, or any value $\top$
- Examples
  - `Bool`($\texttt{false}$) a singleton type
  - `Bool`($\top$) the boolean type
  - `Bool`(ψ_b) indexed boolean type
  - `Bool`($\emptyset$) not a boolean

Row Types

$$
\begin{array}{rcll}
\varrho & ::= & \textit{str} : \tau, \varrho & \text{property} \\
& | & \delta\tau & \text{default type} \\
& | & \rho & \text{row variable}
\end{array}
$$

- Part of object type:
  $\{\texttt{x} : 1\}$
  $: \forall\alpha.\texttt{Obj}(\emptyset)(\texttt{"x"} : \texttt{Number}(1) + \alpha, \delta\texttt{Undefined})$
- Part of function type:
  $\texttt{function}(x)\{\texttt{return}\ x\}$
  $: \forall\rho, \alpha, \beta.\texttt{Obj}(\texttt{Fun}(\texttt{this} : \alpha\ ;\ \texttt{"0"} : \beta, \rho \to \beta))(\delta\texttt{Undefined})$

Constraints

C	$::=$	τ `lacks` $i \quad {}_{i \in \{\bot, u, s, b, n, o\}}$	forces discimination in type sum
	$\mid$	ω `lacks` $i \quad {}_{i \in \{b, s, n, f\}}$	forces discimination in index sum
	$\mid$	ϱ `lacks` $\boldsymbol{str}$	well-formedness of row type
	$\mid$	ϱ `at` ξ_s `is` τ	type-level property access
	$\mid$	$\tau \trianglerighteq \varphi_i \quad {}_{i \in \{u, s, b, n, o\}}$	type conversion

Examples

- `"x"` : `Number`(1), ϱ `at` `String`(`"x"`) `is` `Number`(1)
- `Undefined` $\trianglerighteq$ `String`(`"undefined"`)

Types for $\mathcal{DTJ}$ — Overview

Types

$$\tau \quad ::= \quad \alpha \mid \emptyset \mid \varphi_i + \tau \qquad (i \in \{\bot, u, s, b, n, o\})$$

Rows

$$\varrho \quad ::= \quad str : \tau, \varrho \mid \delta\tau \mid \rho$$

Type environments

$$\Gamma \quad ::= \quad \emptyset \mid \Gamma(x : \forall\overline{\alpha}.C \Rightarrow \tau)$$

Constraints

$$\begin{array}{lcll} C & ::= & \tau \; \texttt{lacks} \; i & i \in \{\bot, u, s, b, n, o\} \\ & \mid & \omega \; \texttt{lacks} \; i & i \in \{b, s, n, f\} \\ & \mid & \varrho \; \texttt{lacks} \; str & \\ & \mid & \tau \trianglerighteq \varphi_i & i \in \{u, s, b, n, o\} \\ & \mid & \varrho \; \texttt{at} \; \xi_s \; \texttt{is} \; \tau & \end{array}$$

Type summands and indices

$$\begin{array}{lcl} \varphi_\bot & ::= & \texttt{Undefined} \\ \varphi_u & ::= & \texttt{Null} \\ \varphi_b & ::= & \texttt{Bool}(\xi_b) \\ \xi_b & ::= & \texttt{false} \mid \texttt{true} \mid \top \mid \emptyset \mid \psi_b \\ \varphi_s & ::= & \texttt{String}(\xi_s) \\ \xi_s & ::= & str \mid \top \mid \emptyset \mid \psi_s \\ \varphi_n & ::= & \texttt{Number}(\xi_n) \\ \xi_n & ::= & num \mid \top \mid \emptyset \mid \psi_n \\ \varphi_f & ::= & \texttt{Fun}(\texttt{this} : \tau \,;\, \varrho \to \tau) \\ \varphi_o & ::= & \texttt{Obj}(\omega)(\varrho) \\ \omega & ::= & \sum_{i \in T, T \subseteq \{b, s, n, f, \bot\}} \varphi_i \end{array}$$

Example Typing

Suppose that

- $\texttt{this} : \texttt{Obj}(\alpha_0)(\rho_1)$
- $\texttt{x} : \tau_1$

Then $\texttt{this}[\texttt{x}] : \tau_2$ provided that these two constraints hold

- $\tau_1 \trianglerighteq \texttt{String}(\xi_s)$
- ρ_1 at ξ_s is τ_2

Example Typing

```
function f (x) {
  return this[x];
}
```

$$f : \forall \alpha_0 \beta_1 \rho_1 \rho_2 \psi.$$
$$(\beta_1 \trianglerighteq \texttt{String}(\psi), \rho_1 \texttt{ at } \psi \texttt{ is } \alpha_1$$
$$, \rho_2 \texttt{ lacks "0"}) \Rightarrow$$
$$\texttt{Obj}(\texttt{Fun}(\texttt{this} : \texttt{Obj}(\alpha_0)(\rho_1) \texttt{ ; "0"} : \beta_1, \rho_2 \rightarrow \alpha_1))(\delta\texttt{Undefine}$$

Example Typing II

```
function f (g, x) {
  return g (x);
}
```

$$
\begin{aligned}
f :\ & \forall \alpha_0 \beta_0 \beta_1 \gamma_0 \gamma_1 \rho_1 \rho_2 \rho_3 \rho_4 \rho_5 \\
& (\beta_0 \trianglerighteq \texttt{Obj}(\texttt{Fun}(\texttt{this} : \gamma_0\, ;\, \texttt{"0"} : \beta_1, \rho_4 \to \gamma_1))(\rho_5) \\
& , \rho_2\ \texttt{lacks}\ \{\texttt{"0"}, \texttt{"1"}\}) \Rightarrow \\
& \texttt{Obj}(\texttt{Fun}(\texttt{this} : \texttt{Obj}(\alpha_0)(\rho_1)\, ; \\
& \qquad\qquad \texttt{"0"} : \beta_0, \texttt{"1"} : \beta_1, \rho_2 \to \gamma_1))(\rho_3)
\end{aligned}
$$

Formal Framework: Syntax of Expressions

Auxiliary

$str \in String\ Constants$

Expressions

e	$::=$	`this`	self reference in method calls
	$\mid$	x	variable
	$\mid$	c	constant (number, string, boolean)
	$\mid$	$\{str : e, \ldots\}$	object literal
	$\mid$	`function` $x(x, \ldots)\{$ $x, \ldots; s$ $\}$	function expression
	$\mid$	$e[e]$	property reference
	$\mid$	`new` $e(e, \ldots)$	object creation
	$\mid$	$e(e, \ldots)$	function call
	$\mid$	$e = e$	assignment
	$\mid$	$p(e, \ldots)$	primitive operators (addition, etc.)

Formal Framework: Syntax of Statements

Statements

s	::=	`skip`	no operation
	|	e	expression statement
	|	s`;`s	sequence
	|	`if (`e`) then {`s`} else {`s`}`	conditional
	|	`while (`e`) {`s`}`	iteration
	|	`return` e	function return

Typing Judgments

$\Gamma \vdash e : \tau$	typing where value is required.
$\Gamma \vdash_{ref} e : \tau/\tau'$	typing where a reference may be required.
$\Gamma \vdash_{lhs} e : \tau$	typing left-hand side of an assignment.
$\Gamma \vdash_{stm} s \rhd \tau$	typing a statement.
$\vdash_{acc} \varrho \,@\, \tau \mapsto \tau'$	object access.
$\vdash_{upd} \varrho \,@\, \tau \hookleftarrow \tau'$	object update.

Typing Expressions in Value Context

$$\frac{\Gamma \vdash e : \tau \quad \tau <: \tau'}{\Gamma \vdash e : \tau'} \qquad \frac{\Gamma \vdash_{ref} e : \tau/\tau'}{\Gamma \vdash e : \tau}$$

Typing Expressions in a Reference Context

Variables and constants

$$\frac{(x : \tau) \in \Gamma}{\Gamma \vdash_{ref} x : \tau / \emptyset} \qquad \Gamma \vdash_{ref} c : \mathit{TypeOf}(c) / \emptyset$$

Object literals

$$\frac{\Gamma \vdash e_1 : \tau_1 \quad \ldots \quad \Gamma \vdash e_n : \tau_n}{\begin{array}{ll}\Gamma \vdash_{ref} & \{str_1 : e_1, \ldots, str_n : e_n\} \\ & : \mathtt{Obj}(\emptyset)(str_1 : \tau_1, \ldots, str_n : \tau_n, \delta\mathtt{Undefined}) / \emptyset\end{array}}$$

Typing Functions

$$\frac{\begin{array}{c}\tau_0 = \mathtt{Obj}(\omega)(\varrho') \qquad \Gamma' = \Gamma(\mathtt{this} : \tau_0)(f : \tau)(x_1 : \tau_1)\ldots(x_n : \tau_n) \\ \Gamma'' = \Gamma'(\mathtt{arguments} : \mathtt{Obj}(\emptyset)(\varrho)) \\ \Gamma''(y_1 : \tau'_1)\ldots(y_n : \tau'_n) \models_{stm} s \rhd \tau' \\ \tau = \mathtt{Obj}(\mathtt{Fun}(\mathtt{this} : \tau_0\,;\ \varrho \rightarrow \tau'))(\delta\mathtt{Undefined}) \\ \varrho = \mathtt{"length"} : \mathtt{Number}(n), \lceil 0 \rceil : \tau_1, \ldots, \lceil n-1 \rceil : \tau_n, \varrho''\end{array}}{\Gamma \models_{ref} \mathtt{function}\ f(x_1, \ldots, x_n)\{y_1, \ldots, y_m; s\} : \tau / \emptyset}$$

Typing Property Access

$$\frac{\Gamma \vdash e_1 : \tau_1 \quad \tau_1 \trianglerighteq \texttt{Obj}(\omega_1)(\varrho_1) \quad \Gamma \vdash e_2 : \tau_2 \quad \models_{acc} \varrho_1 \mathbin{@} \tau_2 \mapsto \tau'}{\Gamma \models_{ref} e_1[e_2] : \tau' / \tau_1}$$

- Key design decision: all elimination constructs extract a summand from a type using the conversion relation $\tau \trianglerighteq \varphi$
- Property access yields a defined base type

Typing Function Call and Method Invocation

$$\frac{\begin{array}{c}\tau_0 \trianglerighteq \texttt{Obj}(\texttt{Fun}(\texttt{this} : \tau' \,;\; \lceil 0 \rceil : \tau_1, \ldots, \lceil n-1 \rceil : \tau_n, \varrho \to \tau))(\varrho') \\ \Gamma \models_{ref} e_0 : \tau_0 / \tau' \quad \Gamma \vdash e_1 : \tau_1 \quad \ldots \quad \Gamma \vdash e_n : \tau_n\end{array}}{\Gamma \models_{ref} e_0(e_1, \ldots, e_n) : \tau / \emptyset}$$

- Also an elimination rule

Type Soundness

- There is a small-step operational semantics of Core JavaScript.
- There is a type soundness proof for the language.

Type Inference for Scripting Languages

(Anderson, Giannini, Drossopoulou) ECOOP 2005

- extends *Type Checking for JavaScript* (Anderson, Giannini), WOOD'04, ENTCS

Different focus:

- extension of a standard record type system
- adds notion of definedness of a field
- specified type inference algorithm

Overview

- JavaScript features considered in JS0
  - functions creating objects
  - dynamic addition of fields and methods
  - reassignment of fields and methods
- Type Soundness Proof
- Type Inference Algorithm

Overview II

JavaScript Features not Included

- libraries of functions,
- native calls,
- global this (through a global object),
- dynamic variable creation,
- functions as objects,
- dynamic removal of members,
- delegation,
- prototyping

Types Informally

- Types have the form

$$t = \mu\alpha.[m_1 : (t_1, \psi_1), \ldots, m_n : (t_n, \psi_n)]$$

- $\psi \in \{\bullet, \circ\}$ indicating
  - $\circ$ potentially present field
  - $\bullet$ definitely present field
- Function types have the form

$$t = \mu\alpha.(O \times t_1) \to t_2$$

A Typical JS0 Program

```
function Date(x) {
    this.mSec = x;
    this.add = addFn;
    this
}

function addFn(x) {
    this.mSec = this.mSec + x.mSec; this
}

//Main
x = new Date(1000);
y = new Date(100);
x.add(y);
```

Syntax, Formally

$P \in \mathit{Program}$	$::=$	F^*	
$F \in \mathit{FuncDecl}$	$::=$	$\texttt{function}\ f(x)\{e\}$	
$e \in \mathit{Exp}$	$::=$	var	locals
		f	function identifier
		$\texttt{new}\ f(e)$	object creation
		$e; e$	sequence
		$e.m(e)$	member call
		$e.m$	member select
		$f(e)$	global call
		$\mathit{lhs} = e$	assignment
		$\texttt{null}$	null
		n	integer
$\mathit{var} \in \mathit{EnvVars}$	$::=$	$\texttt{this} \mid x$	
$\mathit{lhs} \in \mathit{LeftHandSide}$	$::=$	$x \mid e.m$	
Identifiers			
$f \in \mathit{FuncID}$	$::=$	$f \mid f' \mid \ldots$	
$m \in \mathit{MemberID}$	$::=$	$m \mid m' \mid \ldots$	

Typed Version of the JS0 Program

```
function Date(x) : (t1 × int → t2) {
    this.mSec = x;
    this.add = addFn;
    this
}
function addFn(x) : (t2 × t2 → t2) {
    this.mSec = this.mSec + x.mSec; this
}
//Main
t2 x = new Date(1000);
t2 y = new Date(100);
x.add(y);
```

$$
\begin{array}{lll}
t_1 & = & [\mathtt{mSec} : (\mathtt{int}, \circ), \mathtt{add} : (t_2 \times t_2 \rightarrow t_2, \circ)] \\
t_2 & = & \mu\alpha.[\mathtt{mSec} : (\mathtt{int}, \bullet), \mathtt{add} : (\alpha \times \alpha \rightarrow \alpha, \bullet)]
\end{array}
$$

Subtyping
(Abridged)

$$\frac{\psi' = \bullet \Longrightarrow \psi = \bullet}{\psi \leq \psi'} \qquad \frac{\psi \leq \psi'}{(t, \psi) \leq (t, \psi')} \qquad t \leq t$$

$$\frac{\forall m. O'(m) = (t', \psi') \Longrightarrow (O(m) = (t, \psi) \wedge (t, \psi) \leq (t', \psi'))}{O \leq O'}$$

Typing of Expressions

- Judgment $\Gamma \vdash e : t \parallel \Gamma'$
- Assume an explicitly typed language:
  Each function carries a type annotation
- Selection of some standard rules

$$\frac{}{\Gamma \vdash x : \Gamma(x) \parallel \Gamma} \qquad \frac{}{\Gamma \vdash \texttt{null} : O \parallel \Gamma} \qquad \frac{}{\Gamma \vdash n : \texttt{int} \parallel \Gamma}$$

$$\frac{\Gamma \vdash e_1 : t_1 \parallel \Gamma' \qquad \Gamma' \vdash e_2 : t_2 \parallel \Gamma''}{\Gamma \vdash e_1 ; e_2 : t_2 \parallel \Gamma''}$$

Typing Field Access and Variable Assignment

$$\frac{\Gamma \vdash e : O \parallel \Gamma' \qquad O(m) = (t', \bullet)}{\Gamma' \vdash e.m : t' \parallel \Gamma'}$$

$$\frac{\Gamma \vdash e : t \parallel \Gamma' \qquad t \leq \Gamma'(x)}{\Gamma \vdash x = e : t \parallel \Gamma'}$$

Typing Method Call

$$\frac{\begin{array}{c}\Gamma \vdash e_1 : O \parallel \Gamma' \\ O(m) = (G, \bullet) \\ \Gamma' \vdash e_2 : t_2 \parallel \Gamma'' \\ t_2 \leq G(x) \\ O \leq G(\texttt{this})\end{array}}{\Gamma \vdash e_1.m(e_2) : G(\texttt{ret}) \parallel \Gamma''}$$

Typing Function Call

$$\frac{\begin{array}{c}\Gamma \vdash e : t \parallel \Gamma' \\ \texttt{function}\ f(x) : G \\ t \leq G(x) \\ \{m \mid G(\texttt{this})(m) = (t', \bullet)\} = \emptyset\end{array}}{\begin{array}{c}\Gamma \vdash \texttt{new}\ f(e) : G(\texttt{ret}) \parallel \Gamma' \\ \Gamma \vdash f(e) : G(\texttt{ret}) \parallel \Gamma'\end{array}}$$

Typing Property Assignment

$$\frac{\Gamma \vdash e : t \parallel \Gamma' \quad \Gamma'(v) = O \quad O(m) = (t'', \circ) \quad t \leq t'' \quad \Gamma'' = \Gamma'[v \mapsto O[m \mapsto (t'', \bullet)]]}{\Gamma' \vdash v.m = e : t \parallel \Gamma''}$$

Typing Property Update

$$\frac{\begin{array}{c}\Gamma \vdash e_1 : O \parallel \Gamma' \\ \Gamma' \vdash e_2 : t \parallel \Gamma'' \\ O(m) = (t'', \bullet) \\ t \leq t''\end{array}}{\Gamma'' \vdash e_1.m = e_2 : t \parallel \Gamma''}$$

Type Inference

Approach

- Generate constraints between type variables
- Solve the constraints

Type Variables

- A type variable is associated with each occurrence of an expression in a program.
- Additional variables for method calls.
- Example

$$[\![\texttt{this\_Date}]\!]$$
$$[\![\texttt{this\_1}]\!]$$
$$[\![\texttt{ret\_Date}]\!]$$
$$[\![\texttt{x\_Date}]\!]$$
$$[\![\texttt{this\_Date.mSec}]\!]$$
$$[\![\texttt{this\_5}]\!]$$

Substitutions

- A substitution maps type variables to types.
- Example

$$
\begin{array}{lcl}
S_0[\![\texttt{this\_Date}]\!] & = & [\texttt{mSec} : (\texttt{int}, \circ), \texttt{add} : (t_2 \times t_2 \rightarrow t_2, \circ)] \\
S_0[\![\texttt{this\_1}]\!] & = & [\texttt{mSec} : (\texttt{int}, \bullet), \texttt{add} : (t_2 \times t_2 \rightarrow t_2, \circ)] \\
S_0[\![\texttt{ret\_Date}]\!] & = & t_2 \\
S_0[\![\texttt{x\_Date}]\!] & = & \texttt{int} \\
S_0[\![\texttt{this\_Date.mSec}]\!] & = & \texttt{int} \\
S_0[\![\texttt{this\_5}]\!] & = & [\texttt{mSec} : (\texttt{int}, \bullet)]
\end{array}
$$

Constraints

τ	$::=$	$[\![e]\!]$	type variable
ρ	$::=$	$\tau \mid \sigma \mid [m : (\tau, \psi)]$	constraint rhs
σ	$::=$	$(\tau \times \tau \rightarrow \tau) \mid \texttt{int}$	function/int
c	$::=$	$\tau \leq \rho \mid \tau \lhd_m \tau \mid \tau^\circ$	constraint
C	$::=$	$\emptyset \mid c \mid C \cup C$	constraint set

Constraint Solutions

$$\frac{}{S \vdash \emptyset} \qquad \frac{S \vdash C_1 \quad S \vdash C_2}{S \vdash C_1 \cup C_2}$$

$$\frac{S(\tau) \leq S(\tau')}{S \vdash \tau \leq \tau'} \qquad \frac{S(\tau) \leq (S(\tau_1) \times S(\tau_2) \rightarrow S(\tau_3))}{S \vdash \tau \leq (\tau_1 \times \tau_2 \rightarrow \tau_3)}$$

$$\frac{S(\tau) = \texttt{int}}{S \vdash \tau \leq \texttt{int}} \qquad \frac{S(\tau)(m) \leq (S(\tau'), \psi)}{S \vdash \tau \leq [m : (\tau', \psi)]}$$

$$\frac{\forall m' \neq m. S(\tau)(m') = S(\tau')(m') \quad S(\tau)(m) \leq S(\tau')(m)}{S \vdash \tau \lhd_m \tau'}$$

$$\frac{\{m \mid S(\tau)(m) = (t, \bullet)\} = \emptyset}{S \vdash \tau^\circ}$$

Constraint Generation

Example: Constraints for `this.add = addFn`

$$
\begin{array}{lcl}
[\![\texttt{this\_1}]\!] & \leq & [\texttt{add} : ([\![\texttt{this\_1.add}]\!], \circ)] \\
[\![\texttt{this\_2}]\!] & \leq & [\texttt{add} : ([\![\texttt{this\_2.add}]\!], \bullet)] \\
[\![\texttt{this\_2}]\!] & \lhd_{add} & [\![\texttt{this\_1}]\!] \\
[\![\texttt{addFn}]\!] & \leq & [\![\texttt{this\_2.add}]\!] \\
[\![\texttt{addFn}]\!] & \leq & [\![\texttt{this\_2.add} = \texttt{addFn}]\!]
\end{array}
$$

$$
\frac{\begin{array}{ccc}\Gamma \vdash e : t \parallel \Gamma' & \Gamma'(\texttt{this}) = O & O(\texttt{add}) = (t'', \circ) \\ t \leq t'' & \multicolumn{2}{c}{\Gamma'' = \Gamma'[\texttt{this} \mapsto O[\texttt{add} \mapsto (t'', \bullet)]]}\end{array}}{\Gamma' \vdash \texttt{this}.m = e : t \parallel \Gamma''}
$$

Constraint Generation

Example: Constraints for `new Date(1000)`

$$
\begin{array}{lcl}
[\![\texttt{this\_Date}]\!]^{\circ} & & \\
[\![\texttt{1000}]\!] & \leq & [\![\texttt{x\_Date}]\!] \\
[\![\texttt{new Date(1000)}]\!] & \leq & [\![\texttt{ret\_Date}]\!]
\end{array}
$$

$$
\frac{\texttt{function Date}(x) : G \qquad \begin{array}{c}\Gamma \vdash e : t \parallel \Gamma' \\ t \leq G(x)\end{array} \qquad \{m \mid G(\texttt{this})(m) = (t', \bullet)\} = \emptyset}{\Gamma \vdash \texttt{new Date}(e) : G(\texttt{ret}) \parallel \Gamma'}
$$

Constraint Generation

Example: Constraints for `x.mSec`

$$[\![\texttt{x\_2}]\!] \quad \leq \quad [\texttt{mSec} : ([\![\texttt{x\_2.mSec}]\!], \bullet)]$$

$$\frac{\Gamma \vdash e : O \parallel \Gamma' \qquad O(m) = (t', \bullet)}{\Gamma' \vdash e.m : t' \parallel \Gamma'}$$

Constraint Generation

Example: Constraints for `x.add(y)`

$$
\begin{array}{lcl}
[\![\texttt{x\_Main}]\!] & \leq & [\texttt{add} : ([\![\texttt{x\_Main.add}]\!], \bullet)] \\
[\![\texttt{x\_Main.add}]\!] & \leq & ([\![\texttt{call\_this\_5}]\!] \times [\![\texttt{call\_x\_5}]\!] \rightarrow [\![\texttt{call\_ret\_5}]\!] \\
[\![\texttt{x\_Main}]\!] & \leq & [\![\texttt{call\_this\_5}]\!] \\
[\![\texttt{y\_Main}]\!] & \leq & [\![\texttt{call\_x\_5}]\!] \\
[\![\texttt{call\_ret\_5}]\!] & \leq & [\![\texttt{x\_Main.add(y\_Main)}]\!]
\end{array}
$$

$$
\frac{\begin{array}{c}\Gamma \vdash e_1 : O \parallel \Gamma' \qquad O(m) = (G, \bullet) \\ \Gamma' \vdash e_2 : t_2 \parallel \Gamma'' \qquad t_2 \leq G(x) \qquad O \leq G(\texttt{this})\end{array}}{\Gamma \vdash e_1.m(e_2) : G(\texttt{ret}) \parallel \Gamma''}
$$

Constraint Soundness

- If constraint generation for e yields C and
- S is a solution of C
- then there are environments Γ and Γ' such that
- $\Gamma \vdash e : t \parallel \Gamma'$ and $t \leq S(\llbracket e \rrbracket)$.

Solving Constraints

- Apply closure rules
- $\tau \leq \tau', \tau' \leq \rho \longrightarrow \tau \leq \rho$
- $\tau \lhd_{m'} \tau', \tau' \leq [m : (\tau'', \psi)] \longrightarrow \tau \leq [m : (\tau'', \psi)]$
- $\tau \leq \tau', \tau \leq \sigma \longrightarrow \tau' \leq \sigma$
- $\tau \lhd_{m'} \tau', \tau \leq [m : (\tau'', \psi)] \longrightarrow \tau' \leq [m : (\tau'', \circ)]$
- $\tau \leq [m : (\tau', \psi')], \tau \leq [m : (\tau'', \psi'')] \longrightarrow \tau' \leq \tau'', \tau'' \leq \tau'$
- $\tau \leq (\tau_1 \times \tau_2 \rightarrow \tau_3), \tau \leq (\tau'_1 \times \tau'_2 \rightarrow \tau'_3) \longrightarrow \tau_1 \leq \tau'_1, \tau_2 \leq \tau'_2, \tau_3 \leq \tau'_3, \tau'_1 \leq \tau_1, \tau'_2 \leq \tau_2, \tau'_3 \leq \tau_3$
- Assure that no conflicts arise
- Read off the solution

Conclusions & Questions

- typing approach seems viable
- language weirdness concentrated in the conversion relation
- high complexity: singleton types, rows, subtyping, polymorphism, ...
- status: frontend completed, ongoing work on constraint rewriting
- Further work
  - sequential typing to model Anderson's definedness
    `f : { x: tau } * int -> { x: tau, y: int }`
  - abstract interpretation?
  - more than singleton types required?