Types and Analysis for Scripting Languages (Part 3)

Peter Thiemann

Universität Freiburg, Germany

Mini Course, Tallinn, Estland, 25.-27.2.2008

Grammar-Based Analysis of String Expressions

Grammar-Based Analysis of String Expressions

TLDI 2005

Motivation: Strings in Inappropriate Contexts

Substitute for “real” datatypes in scripting languages
Examples: Strings contain . . .

- style descriptors (numbers with units, percentages, etc)
- HTML and XML
- XPath expressions
- SQL statements

in PHP, JavaScript, but also JDBC and many others

The Problem: Maintenance

- "strings with semantics"

⇒ may lead to runtime errors

- not prevented by usual type systems
- hard to test exhaustively

Objective and NON-Objective

Objective Alleviate debugging and maintenance of existing programs

NON-Objective Framework for constructing programs

Related Approach: String Expression Analysis I

Tabuchi, Sumii, Yonezawa.
Regular expression types for strings in a text processing language.
TIP 2002.

- Type and effect system based on simply-typed lambda calculus
- String type indexed with regular expression
- Output effect specified with regular expression
- Regular pattern matching
- Builds on Hosoya/Pierce regular expression types
- No type inference

Related Approach: String Expression Analysis II

Christensen, Møller, and Schwartzbach.
Precise analysis of string expressions.
SAS 2003.

- Java $\Rightarrow$ extended CFG $\mathcal{G}$
- $\mathcal{G} \Rightarrow$ regular grammar $\mathcal{R}$ with $L(\mathcal{G}) \subseteq L(\mathcal{R})$ using Mohri-Nederhof algorithm [2001]
- regular string assertions in the program can be checked effectively using inclusion of regular languages
- automatic analysis for full Java language (all string operations)

This Work's Approach: String Expression Analysis III

- Based on the **polymorphic** HM (X) typing framework for lambda calculus [Odersky et al, 1999]
- string operations generate language inclusion constraints (generalizing CFGs with polymorphism)
- effective check of **context-free assertions** in the program by parsing inclusion constraints
- a context-free assertion is a sentential form for a context-free reference grammar

Syntax

Alphabet	T		
Symbols	$\mathtt{a}, \mathtt{b}$	$\in$	T
Words	w	$\in$	T^*
Constants	c	$\in$	$T^* \cup \{\cdot, \mathtt{if}\ \ \}$
Expressions	e	$::=$	$c \mid x \mid e\,(e) \mid \mathtt{rec}\ f(x)\ e \mid$
			$\mathtt{let}\ x = e\ \mathtt{in}\ e$

Semantics

$$\begin{array}{llll} \text{Values} & v & ::= & w \mid \texttt{rec}\ f(x)\ e \\ \text{Ev. Contexts} & E & ::= & [\,] \mid E\ (e) \mid e\ (E) \mid E \cdot e \mid v \cdot E \mid \texttt{if}\ E\ e\ e \end{array}$$

Beta reduction

$$(\texttt{rec}\ f(x)\ e)\ (v) \quad \longrightarrow \quad e[x \mapsto v, f \mapsto \texttt{rec}\ f(x)\ e]$$

Delta reduction

$$\begin{array}{lcl} \texttt{if}\ (\texttt{a}w)\ e_1\ e_2 & \longrightarrow & e_1 \\ \texttt{if}\ \varepsilon\ e_1\ e_2 & \longrightarrow & e_2 \\ \texttt{a}_1 \dots \texttt{a}_n \cdot \texttt{b}_1 \dots \texttt{b}_m & \longrightarrow & \texttt{a}_1 \dots \texttt{a}_n \texttt{b}_1 \dots \texttt{b}_m \end{array}$$

Reduction

$$\frac{e \longrightarrow e'}{E[e] \mapsto E[e']}$$

Type Language

Types	τ	$::=$	$\alpha \mid \texttt{Str}(\varphi) \mid \tau \rightarrow \tau$
Language Variables	φ	$\in$	Φ
Constrained Types	ρ	$::=$	$C \Rightarrow \tau$
Constraints	C	$::=$	$\mathsf{true} \mid \mathsf{fail} \mid \tau \mathrel{\dot{\leq}} \tau \mid r \mathrel{\dot{\subseteq}} \varphi \mid C \wedge C$
String Type Indices	r	$::=$	$\varphi \mid \varepsilon \mid \texttt{a} \mid r \cdot r$
Type Schemes	σ	$::=$	$\forall \tilde{\varphi} \tilde{\alpha}.\rho$
Type Environments	Γ	$::=$	$\emptyset \mid \Gamma(x : \sigma)$

Typing Rules

Standard HM(X)

$$\frac{(x : \sigma) \in \Gamma}{C, \Gamma \vdash x : \sigma} \qquad \frac{C, \Gamma \vdash e : \tau \qquad C \Vdash \tau \dot{\leq} \tau'}{C, \Gamma \vdash e : \tau'}$$

$$\frac{C, \Gamma \vdash e_1 : \tau_2 \rightarrow \tau_1 \qquad C, \Gamma \vdash e_2 : \tau_2}{C, \Gamma \vdash e_1\ (e_2) : \tau_1}$$

$$\frac{C, \Gamma(f : \tau_2 \rightarrow \tau_1)(x : \tau_2) \vdash e : \tau_1}{C, \Gamma \vdash \texttt{rec}\ f(x)\ e : \tau_2 \rightarrow \tau_1}$$

$$\frac{C, \Gamma \vdash e : \sigma \qquad C, \Gamma(x : \sigma) \vdash e' : \tau'}{C, \Gamma \vdash \texttt{let}\ x = e\ \texttt{in}\ e' : \tau'}$$

$$\frac{C \wedge D, \Gamma \vdash e : \tau \qquad \{\tilde{\alpha}, \tilde{\varphi}\} \cap (fv(C) \cup fv(\Gamma)) = \emptyset}{C, \Gamma \vdash e : \forall \tilde{\alpha}\tilde{\varphi}.D \Rightarrow \tau}$$

$$\frac{\begin{array}{c} C, \Gamma \vdash e : \forall \tilde{\alpha}\tilde{\varphi}.D \Rightarrow \tau \\ S = [\tilde{\alpha} \mapsto \tilde{\tau}, \tilde{\varphi} \mapsto \tilde{\varphi}'] \qquad C \Vdash S(D) \end{array}}{C, \Gamma \vdash e : S(\tau)}$$

Language Inclusion Constraints

- arise from the type signatures of the string operations

constant	w	:	$\forall\varphi.(w \dot{\subseteq} \varphi) \Rightarrow$ $\texttt{Str}(\varphi)$
concatenation	$\cdot$	:	$\forall\varphi_1, \varphi_2, \varphi.(\varphi_1 \cdot \varphi_2 \dot{\subseteq} \varphi) \Rightarrow$ $\texttt{Str}(\varphi_1) \to \texttt{Str}(\varphi_2) \to \texttt{Str}(\varphi)$
conditional	`if`	:	$\forall\alpha, \varphi.$ $\texttt{Str}(\varphi) \to \alpha \to \alpha \to \alpha$

- give rise to subtyping:

$$\frac{\varphi_1 \dot{\subseteq} \varphi_2}{\texttt{Str}(\varphi_1) \dot{\leq} \texttt{Str}(\varphi_2)}$$

Solution of a Constraint

- A solution of constraint C is a pair (S, Θ) of
  - an idempotent substitution S of type variables by types
  - a language assignment $\Theta : \Phi \to \mathcal{P}(T^*)$

  so that $S, \Theta \models C$ holds.
- The meaning of left hand sides of constraints

$$
\begin{array}{lcl}
L_\Theta(\varphi) & = & \Theta(\varphi) \\
L_\Theta(\varepsilon) & = & \{\varepsilon\} \\
L_\Theta(\mathtt{a}) & = & \{\mathtt{a}\} \\
L_\Theta(r_1 \cdot r_2) & = & \{w_1 \cdot w_2 \mid w_1 \in L_\Theta(r_1), w_2 \in L_\Theta(r_2)\}
\end{array}
$$

Constraint Satisfaction

$$\frac{\Theta \models S(C)}{S, \Theta \models C} \qquad \frac{\Theta \models C_1 \qquad \Theta \models C_2}{\Theta \models C_1 \wedge C_2}$$

$$\Theta \models \text{true} \qquad \Theta \models \alpha \mathrel{\dot{\leq}} \alpha$$

$$\frac{\Theta(\varphi_1) \subseteq \Theta(\varphi_2)}{\Theta \models \texttt{Str}(\varphi_1) \mathrel{\dot{\leq}} \texttt{Str}(\varphi_2)}$$

$$\frac{\Theta \models \tau_2' \mathrel{\dot{\leq}} \tau_2 \quad \Theta \models \tau_1 \mathrel{\dot{\leq}} \tau_1'}{\Theta \models \tau_2 \rightarrow \tau_1 \mathrel{\dot{\leq}} \tau_2' \rightarrow \tau_1'}$$

$$\frac{L_\Theta(r) \subseteq \Theta(\varphi)}{\Theta \models r \mathrel{\dot{\subseteq}} \varphi}$$

Example: Render a Tree in HTML

```
render t = case t of
  Tip num ->
    int2string num
  Fork l r ->
    "<ul><li>" ++ render l ++ "</li><li>"
               ++ render r ++ "</li></ul>"
```

Assuming

$$\begin{array}{lcl} C_1 & = & (0 \cdot \varphi'' \dot{\subseteq} \varphi' \wedge 1 \cdot \varphi'' \dot{\subseteq} \varphi' \wedge \cdots \wedge \varepsilon \dot{\subseteq} \varphi'' \wedge \varphi' \dot{\subseteq} \varphi'') \\ \texttt{int2string} & : & \forall \varphi', \varphi''.(C_1) \Rightarrow \texttt{Bool} \rightarrow \texttt{Str}(\varphi') \end{array}$$

We find that

$$\begin{array}{lcl} C_2 & = & \texttt{
} \cdot \varphi \cdot \texttt{
} \cdot \varphi \cdot \texttt{
} \dot{\subseteq} \varphi \wedge \varphi' \dot{\subseteq} \varphi \\ \texttt{render} & : & \forall \varphi, \varphi', \varphi''.(C_2 \wedge C_1) \Rightarrow \texttt{Tree} \rightarrow \texttt{Str}(\varphi) \end{array}$$

Does `render` create HTML output?

Normalizing Constraints

$$
\begin{array}{lcll}
\mathtt{Str}(\varphi) \dot{\leq} \mathtt{Str}(\varphi') & \Leftrightarrow & \varphi \dot{\subseteq} \varphi' & \\
\mathtt{Str}(\varphi) \dot{\leq} \tau \rightarrow \tau' & \Leftrightarrow & \text{fail} & \\
\mathtt{Str}(\varphi) \dot{\leq} \alpha & \Rightarrow & \alpha \doteq \mathtt{Str}(\varphi') & \textbf{if } \varphi' \textbf{ fresh} \\
\tau \rightarrow \tau' \dot{\leq} \mathtt{Str}(\varphi') & \Leftrightarrow & \text{fail} & \\
\tau \rightarrow \tau' \dot{\leq} \tau_1 \rightarrow \tau_1' & \Leftrightarrow & \tau_1 \dot{\leq} \tau \wedge \tau' \dot{\leq} \tau_1' & \\
\tau \rightarrow \tau' \dot{\leq} \alpha & \Rightarrow & \alpha \doteq \alpha' \rightarrow \alpha'' & \textbf{if } \alpha', \alpha'' \textbf{ fresh}, \alpha \notin \mathit{fv}(\tau, \tau') \\
\alpha \dot{\leq} \mathtt{Str}(\varphi') & \Rightarrow & \alpha \doteq \mathtt{Str}(\varphi) & \textbf{if } \varphi \textbf{ fresh} \\
\alpha \dot{\leq} \tau \rightarrow \tau' & \Rightarrow & \alpha \doteq \alpha' \rightarrow \alpha'' & \textbf{if } \alpha', \alpha'' \textbf{ fresh}, \alpha \notin \mathit{fv}(\tau, \tau') \\
\alpha \dot{\leq} \alpha & \Leftrightarrow & \text{true} & \\
\alpha \dot{\leq} \alpha' \wedge \alpha' \dot{\leq} \alpha'' & \Rightarrow & \alpha \dot{\leq} \alpha'' & \\
\alpha \dot{\leq} \alpha' \wedge \alpha' \dot{\leq} \alpha & \Leftrightarrow & \alpha \doteq \alpha' & \\
\alpha \doteq \alpha & \Leftrightarrow & \text{true} & \\
\varphi \doteq \varphi & \Leftrightarrow & \text{true} & \\
\varphi \dot{\subseteq} \varphi & \Leftrightarrow & \text{true} & \\
\varphi \dot{\subseteq} \varphi' \wedge \varphi' \dot{\subseteq} \varphi & \Leftrightarrow & \varphi \doteq \varphi' &
\end{array}
$$

Normalizing Constraints II

$$
\begin{array}{lcl}
C@(C' \wedge & \Rightarrow & r_1 r r_2 \,\dot{\subseteq}\, \varphi' \quad \textbf{if } \varphi \notin \mathit{reach}(C, \varphi) \\
\quad r \,\dot{\subseteq}\, \varphi \wedge & & \\
\quad r_1 \varphi r_2 \,\dot{\subseteq}\, \varphi') & & \\
C \wedge \varphi \doteq \varphi' & \Leftrightarrow & C[\varphi \mapsto \varphi'] \wedge \varphi \doteq \varphi' \\
C \wedge \alpha \doteq \tau & \Leftrightarrow & C[\alpha \mapsto \tau] \wedge \alpha \doteq \tau \quad \textbf{if } \alpha \notin \mathit{fv}(\tau) \\
C \wedge \alpha \doteq \tau & \Leftrightarrow & \text{fail} \quad \textbf{if } \alpha \in \mathit{fv}(\tau) \wedge \tau \neq \alpha \\
C \wedge \text{true} & \Leftrightarrow & C \\
C \wedge \text{fail} & \Leftrightarrow & \text{fail}
\end{array}
$$

where

$$
\begin{array}{lcl}
\mathit{reach}(C, \varphi) & = & \mu V.F(V) \cup F(\{\varphi\}) \\
F(V) & = & \{\varphi' \mid r_1 \varphi' r_2 \,\dot{\subseteq}\, \varphi \in C, \varphi \in V\}
\end{array}
$$

Context-Free Languages

- A normalized set of inclusion constraints can be read as a context-free grammar $\mathcal{G}$.
  - The language variables serve as nonterminal symbols.
  - Each constraint $r \mathrel{\dot{\subseteq}} \varphi$ serves as a production $\varphi \to r$.
- Normalization unfolds productions up to recursion.

Context-Free Languages

- A normalized set of inclusion constraints can be read as a context-free grammar $\mathcal{G}$.
  - The language variables serve as nonterminal symbols.
  - Each constraint $r \mathrel{\dot\subseteq} \varphi$ serves as a production $\varphi \to r$.
- Normalization unfolds productions up to recursion.
- Problem
  - Given a context-free reference grammar $\mathcal{G}_0$,
  - How can we check that a sublanguage of $\mathcal{G}$ is contained in a sublanguage of $\mathcal{G}_0$?

A Reference Grammar for HTML

$S \rightarrow$ `<html>` $H\ B$ `</html>`
$S \rightarrow H\ B$
$H \rightarrow$ `<head>` T `</head>`
$T \rightarrow$ `<title>` P `</title>`
$B \rightarrow$ `<body>` C `</body>`
$P \rightarrow$
$P \rightarrow P\ P$
$P \rightarrow 0$
$P \rightarrow 1$
$\vdots$

$C \rightarrow$
$C \rightarrow P$
$C \rightarrow C\ C$
$C \rightarrow$ < p > C < /p >
$C \rightarrow$ < b > C < /b >
$C \rightarrow$ < tt > C < /tt >
$C \rightarrow$ `<em>` C `</em>`
$C \rightarrow$ `<i>` C `</i>`
$C \rightarrow$ `<u>` C `</u>`
$C \rightarrow$ `<s>` C `</s>`
$C \rightarrow$ `<ol>` L `</ol>`
$C \rightarrow$ `<ul>` L `</ul>`
$L \rightarrow$ `<li>` C `</li>`
$L \rightarrow L\ L$

Adapted from the paper

Solving Constraints by Parsing

- Substitute nonterminals of the reference grammar for the language variables in the constraints (with Θ)
- Check for each constraint $r \dot{\subseteq} \varphi$ if $\Theta(\varphi) \stackrel{*}{\Rightarrow} \Theta(r)$
- If successful, replace constraint by *assignment constraint* Θ

In the example

$$
\begin{array}{lcl}
\mathtt{0} \cdot \varphi'' \dot{\subseteq} \varphi' & \rightsquigarrow & [\varphi' \mapsto P, \varphi'' \mapsto P] \\
\varepsilon \dot{\subseteq} \varphi'' & \rightsquigarrow & [\varphi'' \mapsto C] \vee [\varphi'' \mapsto P] \\
\varphi' \dot{\subseteq} \varphi'' & \rightsquigarrow & \varphi' = \varphi'' \vee [\varphi' = P, \varphi'' = C] \\
\varphi' \dot{\subseteq} \varphi & \rightsquigarrow & \varphi' = \varphi \vee [\varphi' = P, \varphi = C] \\
\multicolumn{3}{l}{\mathtt{
} \cdot \varphi \cdot \mathtt{
} \cdot \varphi \cdot \mathtt{
} \dot{\subseteq} \varphi} \\
 & \rightsquigarrow & [\varphi = C]
\end{array}
$$

- Result: a disjunction of assignment constraints

Resolving Assignment Constraints

Simplify disjunction of assignment constraints:

$$
\begin{array}{ll}
 & [\varphi' \mapsto P, \varphi'' \mapsto P] \bowtie ([\varphi'' \mapsto C] \vee [\varphi'' \mapsto P]) \bowtie (\varphi' = \varphi'' \vee [\varphi' = P, \varphi'' \\
= & [\varphi' \mapsto P, \varphi'' \mapsto P] \bowtie (\varphi' = \varphi'' \vee [\varphi' = P, \varphi'' = C]) \bowtie (\varphi' = \varphi \vee [\varphi' = P \\
= & [\varphi' \mapsto P, \varphi'' \mapsto P] \bowtie (\varphi' = \varphi \vee [\varphi' = P, \varphi = C]) \bowtie [\varphi = C] \\
= & ([\varphi' \mapsto P, \varphi'' \mapsto P, \varphi \mapsto P] \vee [\varphi' \mapsto P, \varphi'' \mapsto P, \varphi \mapsto C]) \bowtie [\varphi = C] \\
= & [\varphi' = P, \varphi'' = P, \varphi = C]
\end{array}
$$

Hence the simplified type scheme

$$\texttt{render} : \forall \varphi . [\varphi = C] \Rightarrow \texttt{Tree} \rightarrow \texttt{Str}(\varphi)$$

Earley's Parser Solves Parsing Constraints

init		$[\rightarrow \bullet\gamma, 0] \in E_0$.
scan	$[A \rightarrow \alpha \bullet X\beta, j] \in E_i$ and $X_{i+1} = X \Rightarrow$	$[A \rightarrow \alpha X \bullet \beta, j] \in E_{i+}$
pred	$[A \rightarrow \alpha \bullet B\beta, j] \in E_i$ and $B \rightarrow \gamma \in P \Rightarrow$	$[B \rightarrow \bullet\gamma, i] \in E_i$.
red	$[B \rightarrow \gamma\bullet, j] \in E_i$ and $[A \rightarrow \alpha \bullet B\beta, k] \in E_j \Rightarrow$	$[A \rightarrow \alpha B \bullet \beta, k] \in E_i$.

- $X_1 \ldots X_n$ input word with $X_i \in T \cup N$
- $E_0, \ldots, E_n$ sets of items
- accept, if $[\rightarrow \gamma\bullet, 0] \in E_n$
- only difference to standard Earley in **scan**
- extension: construct assignment while parsing (paper)

Left/Right Bias

Consider

- the left-recursive reference grammar $S \rightarrow \texttt{a} \mid S\texttt{b}$
- the right-recursive expression
  $\texttt{a} \cdot ((\texttt{rec}\ f(x)\ \texttt{if}\ (x = 0)\ \varepsilon\ \texttt{b} \cdot (f\ (x - 1)))\ 10)$

Left/Right Bias

Consider

- the left-recursive reference grammar $S \to \mathtt{a} \mid S\mathtt{b}$
- the right-recursive expression
  $\mathtt{a} \cdot ((\mathtt{rec}\ f(x)\ \mathtt{if}\ (x = 0)\ \varepsilon\ \mathtt{b} \cdot (f\ (x - 1)))\ 10)$
- has type $\mathtt{Str}(\varphi)$
- where $\mathtt{a} \cdot \varphi' \mathrel{\dot{\subseteq}} \varphi, \varepsilon \mathrel{\dot{\subseteq}} \varphi', \mathtt{b} \cdot \varphi' \mathrel{\dot{\subseteq}} \varphi', [\mathtt{a} \mathrel{\dot{\subseteq}} \varphi]$

Left/Right Bias

Consider

- the left-recursive reference grammar $S \rightarrow \texttt{a} \mid S\texttt{b}$
- the right-recursive expression
  $\texttt{a} \cdot ((\texttt{rec}\ f(x)\ \texttt{if}\ (x = 0)\ \varepsilon\ \texttt{b} \cdot (f\ (x - 1)))\ 10)$
- has type $\texttt{Str}(\varphi)$
- where $\texttt{a} \cdot \varphi' \dot{\subseteq} \varphi, \varepsilon \dot{\subseteq} \varphi', \texttt{b} \cdot \varphi' \dot{\subseteq} \varphi', [\texttt{a} \dot{\subseteq} \varphi]$
- We can assign $[\varphi \mapsto S]$, but there is no possible assignment for φ'

Left/Right Bias

Consider

- the left-recursive reference grammar $S \to \mathtt{a} \mid S\mathtt{b}$
- the right-recursive expression
  $\mathtt{a} \cdot ((\mathtt{rec}\ f(x)\ \mathtt{if}\ (x = 0)\ \varepsilon\ \mathtt{b} \cdot (f\ (x - 1)))\ 10)$
- has type $\mathtt{Str}(\varphi)$
- where $\mathtt{a} \cdot \varphi' \mathrel{\dot\subseteq} \varphi, \varepsilon \mathrel{\dot\subseteq} \varphi', \mathtt{b} \cdot \varphi' \mathrel{\dot\subseteq} \varphi', [\mathtt{a} \mathrel{\dot\subseteq} \varphi]$
- We can assign $[\varphi \mapsto S]$, but there is no possible assignment for φ'
- But for right-recursive grammar $A \to \mathtt{a}B, B \to \varepsilon \mid \mathtt{b}B$ we find the assignment $[\varphi \mapsto A, \varphi' \mapsto B]$.

Left/Right Bias

Consider

- the left-recursive reference grammar $S \rightarrow \mathtt{a} \mid S\mathtt{b}$
- the right-recursive expression
  $\mathtt{a} \cdot ((\mathtt{rec}\ f(x)\ \mathtt{if}\ (x = 0)\ \varepsilon\ \mathtt{b} \cdot (f\ (x - 1)))\ 10)$
- has type $\mathtt{Str}(\varphi)$
- where $\mathtt{a} \cdot \varphi' \dot{\subseteq} \varphi, \varepsilon \dot{\subseteq} \varphi', \mathtt{b} \cdot \varphi' \dot{\subseteq} \varphi', [\mathtt{a} \dot{\subseteq} \varphi]$
- We can assign $[\varphi \mapsto S]$, but there is no possible assignment for φ'
- But for right-recursive grammar $A \rightarrow \mathtt{a}B, B \rightarrow \varepsilon \mid \mathtt{b}B$ we find the assignment $[\varphi \mapsto A, \varphi' \mapsto B]$.
- Moral: do not introduce left or right bias in the grammar

Conclusions

- + String expression analysis that deals with context-free languages
- + Essentially type inference for HM(Earley)
- + Constraint solver implemented
- + Working and efficient implementation for PHP
- + Can now deal with deconstruction (not shown)
- – Special grammar needed
  - character-based, not token-based
  - should not be left- or right-biased, ambiguous grammars preferred
- – More string operations