Type-Directed, Whitespace-Delimited Parsing for Embedded DSLs


Cyrus Omar


Benjamin Chung

http://cs.cmu.edu/~dkurilov/
Darya Kurilova


Alex Potanin


Jonathan Aldrich

School of Computer Science
Carnegie Mellon University

Wyvern

- **Goals:** Secure web and mobile programming within a single statically-typed language.

- Language-level support for a variety of **domains**:
  - Security policies and architecture specifications
  - Client-side programming (HTML, CSS)
  - Server-side programming (Databases)

Benefits of DSLs

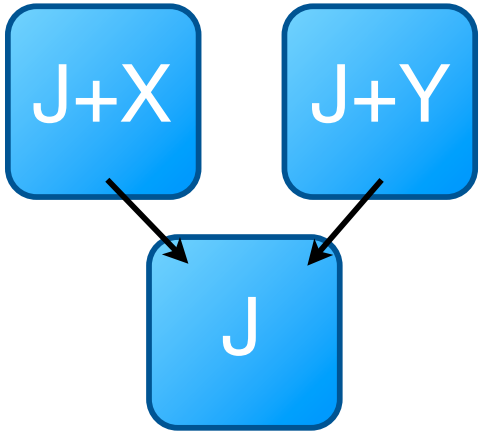
- Specialized **syntax** improves **ease-of-use**
- Specialized **typechecking rules** improve **verifiability**
- Specialized **translation strategies** improve **performance** and **interoperability** with existing technologies
- Specialized **tool support** improves **ease-of-use**.

Types of DSLs

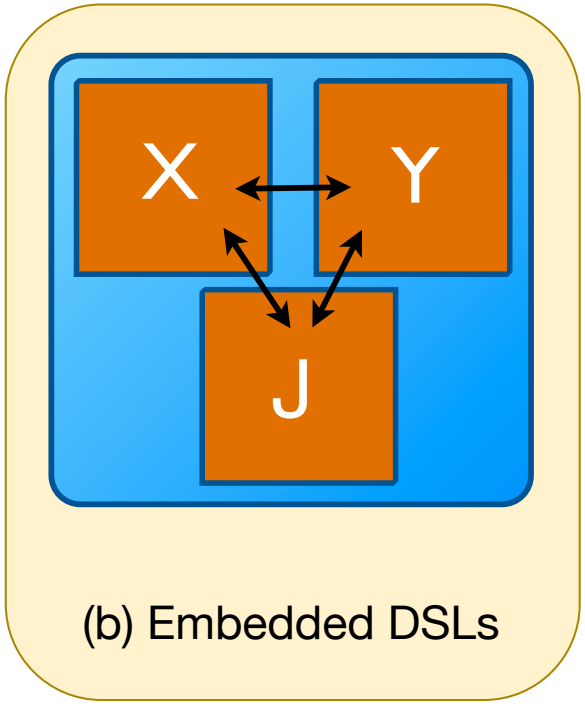
- **Standalone DSLs** are **external**; must call into each other via *interoperability layers*.

- **Embedded DSLs** use mechanisms **internal** to a host general-purpose language; distributed and accessed as libraries.

Natural Interoperability



(a) Standalone Languages



(b) Embedded DSLs

naturally compatible

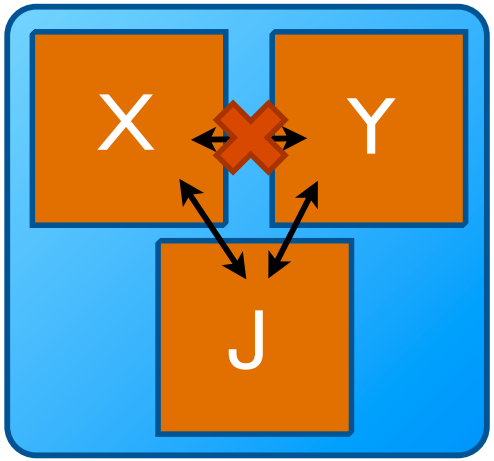
X, Y, J
primitive constructs

library

language

Caveat: Expressivity vs. Safety

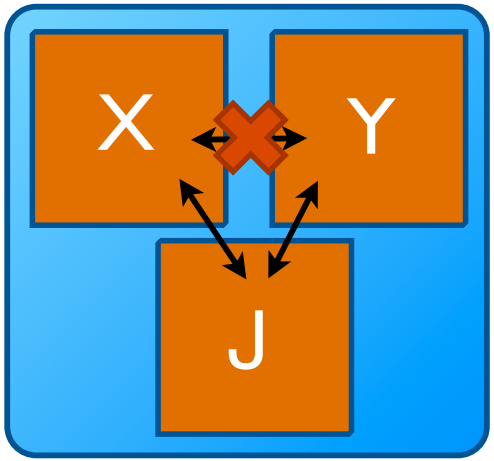
- Want **expressive (syntax) extensions**.
- But if you give each DSL too much control, they may **interfere with one another** at link-time.



(b) Embedded DSLs

Example: SugarJ [Erdweg et al, 2010]

- Libraries can extend the **base syntax** of the language
- These extensions are imported **transitively**
- Even seemingly simple extensions can **interfere**:
  - Pairs vs. Tuples
  - HTML vs. XML



(b) Embedded DSLs

Our Solution

- Libraries **cannot** extend the **base syntax** of the language
- Instead, **syntax is associated with types**.
  - Type-specific syntax can be used to **create values of that type**.
- How? By placing a tilde (~) where an expression of that type is expected, and beginning an **indented block** after the containing declaration/statement.

Example: Architecture Specification

```
1  val dashboardArchitecture : Architecture = ~
2     external component twitter : Feed
3        location www.twitter.com
4     external component client : Browser
5        connects to servlet
6     component servlet : DashServlet
7        connects to productDB, twitter
8        location intranet.nameless.com
9     component productDB : Database
10       location db.nameless.com
11    policy mainPolicy = ~
12       must salt servlet.login.password
13       connect * -> servlet with HTTPS
14       connect servlet -> productDB with TLS
```

Wyvern DSL: Architecture Specification

Example: Queries

```
1  val newProds = productDB.query(~)
2      select twHandle
3      where introduced - today < 3 months
4  val prodTwt = new Feed(newProds)
5  return prodTwt.query(~)
6      select *
7      group by followed
8      where count > 1000
```

Wyvern DSL: Queries

Examples: HTML and URLs

```
1  serve(page, loc) where
2    val page = ~
3      html:
4        head:
5          title: Hot Products
6          style: {myStylesheet}
7          body:
8            div id="search":
9              {SearchBox("products")}
10           div id="products":
11             {FeedBox(servlet.hotProds())}
12   val loc = ~
13     products.nameless.com
```

Wyvern DSLs: Presentation and URLs

Type-Associated Grammars

```
val dashboardArchitecture : Architecture = ~
   external component twitter : Feed
      location www.twitter.com
   external component client : Browser
      connects to servlet
   component servlet : DashServlet
      connects to productDB, twitter
      location intranet.nameless.com
   component productDB : Database
      location db.nameless.com
   policy mainPolicy = ~
      must salt servlet.login.password
      connect * -> servlet with HTTPS
      connect servlet -> productDB with TLS
```

```
1  type Architecture
2    grammar ::= (component|policy)+
3    component ::= "external"? "component"
4                  ID ":" TYPE
5                  ((componentAttr)*)?
6    componentAttr ::= "location " URL.grammar
7                    | "connects to" (ID ",")* ID
8    policy ::= "policy" ID "=" (EXP : Policy)
```

(provisional syntax)

Composition: Wyvern Productions

```
val dashboardArchitecture : Architecture = ~
   external component twitter : Feed
      location www.twitter.com
   external component client : Browser
      connects to servlet
   component servlet : DashServlet
      connects to productDB, twitter
      location intranet.nameless.com
   component productDB : Database
      location db.nameless.com
   policy mainPolicy = ~
      must salt servlet.login.password
      connect * -> servlet with HTTPS
      connect servlet -> productDB with TLS
```

```
1  type Architecture
2    grammar ::= (component|policy)+
3    component ::= "external"? "component"
4                  ID ":" TYPE
5                  ((componentAttr)*)?
6    componentAttr ::= "location " URL.grammar
7                    | "connects to" (ID ",")* ID
8    policy ::= "policy" ID "=" (EXP : Policy)
```

(provisional syntax)

Composition: Imported Productions

```
val dashboardArchitecture : Architecture = ~
   external component twitter : Feed
      location www.twitter.com
   external component client : Browser
      connects to servlet
   component servlet : DashServlet
      connects to productDB, twitter
      location intranet.nameless.com
   component productDB : Database
      location db.nameless.com
   policy mainPolicy = ~
      must salt servlet.login.password
      connect * -> servlet with HTTPS
      connect servlet -> productDB with TLS
```

```
1  type Architecture
2    grammar ::= (component|policy)+
3    component ::= "external"? "component"
4                  ID ":" TYPE
5                  ((componentAttr)*)?
6    componentAttr ::= "location " URL.grammar
7                    | "connects to" (ID ",")* ID
8    policy ::= "policy" ID "=" (EXP : Policy)
```

(provisional syntax)

Composition: Typed Wyvern Expressions

```
val dashboardArchitecture : Architecture = ~
   external component twitter : Feed
      location www.twitter.com
   external component client : Browser
      connects to servlet
   component servlet : DashServlet
      connects to productDB, twitter
      location intranet.nameless.com
   component productDB : Database
      location db.nameless.com
   policy mainPolicy = ~
      must salt servlet.login.password
      connect * -> servlet with HTTPS
      connect servlet -> productDB with TLS
```

```
1  type Architecture
2    grammar ::= (component|policy)+
3    component ::= "external"? "component"
4                  ID ":" TYPE
5                 ((componentAttr)*)?
6    componentAttr ::= "location " URL.grammar
7                    | "connects to" (ID ",")* ID
8    policy ::= "policy" ID "=" (EXP : Policy)
```

(provisional syntax)

Phase I: Top-Level Parsing

- The top-level layout-sensitive syntax of Wyvern can be parsed first without involving the typechecker
  - **Useful for tools like documentation generators**
  - **Wyvern's grammar can be written down declaratively using a layout-sensitive formalism** [Erdweg et al 2012; Adams 2013]
- DSL blocks are left as unparsed **"DSL literals"** during this phase

Phase II: Typechecking and DSL Parsing

- When a tilde expression (~) is encountered during typechecking, its **expected type** is determined via:
  - Explicit annotations
  - Method signatures
  - Type propagation into **where** clauses

- The subsequent DSL literal is now parsed according to the **type-associated grammar**.
  - Any internal Wyvern expressions are also parsed (I & II) and typechecked recursively during this phase.

Benefits

- **Modularity and Safe Composability**
  - DSLs are distributed in libraries, along with types
  - No link-time errors
- **Identifiability**
  - Can easily see when a DSL is being used via ~ and whitespace
  - Can determine which DSL is being used by identifying expected type
  - DSLs always generate a value of the corresponding type
- **Simplicity**
  - Single mechanism that can be described in a few sentences
  - Specify a grammar in a natural manner within the type
- **Flexibility**
  - Whitespace-delimited blocks can contain *arbitrary* syntax

Ongoing Work

Inline DSL Literals

- Whitespace-delimited blocks admit arbitrary syntax but…
  - May be unwieldy for simple DSLs (e.g. URLs, times, dates, etc.)
  - Only allow one DSL block per declaration/statement

- Solution: Alternative inline forms for DSL literals (with same type-directed semantics)
  - Collection of common delimiter forms
    - `“DSL literal”`
    - `` `DSL literal` ``
    - `{DSL literal}`
    - `<DSL literal>`
    - `[DSL literal]`
    - `/DSL literal/`
    - …

Inline DSL Literals

- That is, these three forms could be exactly equivalent, assuming `f` takes a single argument of type `URL`
  - `f(~) http://github.com/wyvernlang/wyvern`
  - `f(`http://github.com/wyvernlang/wyvern`)`
  - `f([http://github.com/wyvernlang/wyvern])`
  - `f("http://github.com/wyvernlang/wyvern")`

  (String literals are simply a DSL associated with the `String` type!)

- Alternatively, types could restrict the valid forms of identifier to allow the language itself to enforce conventions.

Keyword-Directed Invocation

- Most language extension mechanisms invoke DSLs using functions or keywords (e.g. macros), rather than types.
- The keyword-directed invocation strategy can be considered a special case of the type-directed strategy.
  - The keyword is simply a function taking one argument.
  - The argument type specifies a grammar that captures one or more expressions.

Example: Control Flow

```
if : bool -> (unit -> a), (unit -> a) -> a
```

IfBranches

```
if(in_france, ~)
    do_as_the_french_do()
  else
    panic()
```

```
if(in_france)
    do_as_the_french_do()
  else
    panic()
```

Interaction with Subtyping

- With subtyping, multiple subtypes may define a grammar.
- Possible Approaches:
  - Use only the declared type of functions
  - Explicit annotation on the tilde
  - Parse against all possible grammars, disambiguate as needed
  - Other mechanisms?

Interaction with Tools

- Syntax interacts with syntax highlighters + editor features.
- Still need to figure out how to support type-specific syntax in these contexts.
  - Borrow ideas from language workbenches?

Related Work

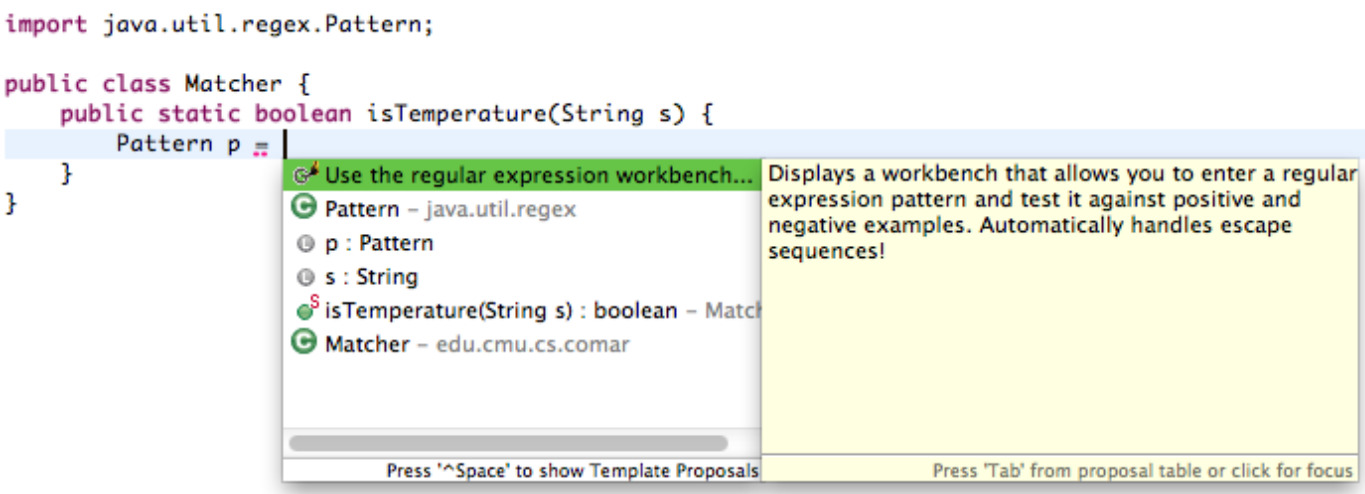
Active Libraries [Veldhuizen, 1998]

- Active libraries are not passive collections of routines or objects, as are traditional libraries, but take an active role in generating code.

Active Code Completion [Omar et al, ICSE 2012]

- Use types similarly to control the IDE's code completion system.

Active Code Completion with GRAPHITE



Active Code Completion with GRAPHITE

```
import java.util.regex.Pattern;

public class Matcher {
    public static boolean isTemperature(String s) {
        Pattern p =
    }
}
```

Enter your regular expression pattern here.

Ignore Case

Should match...

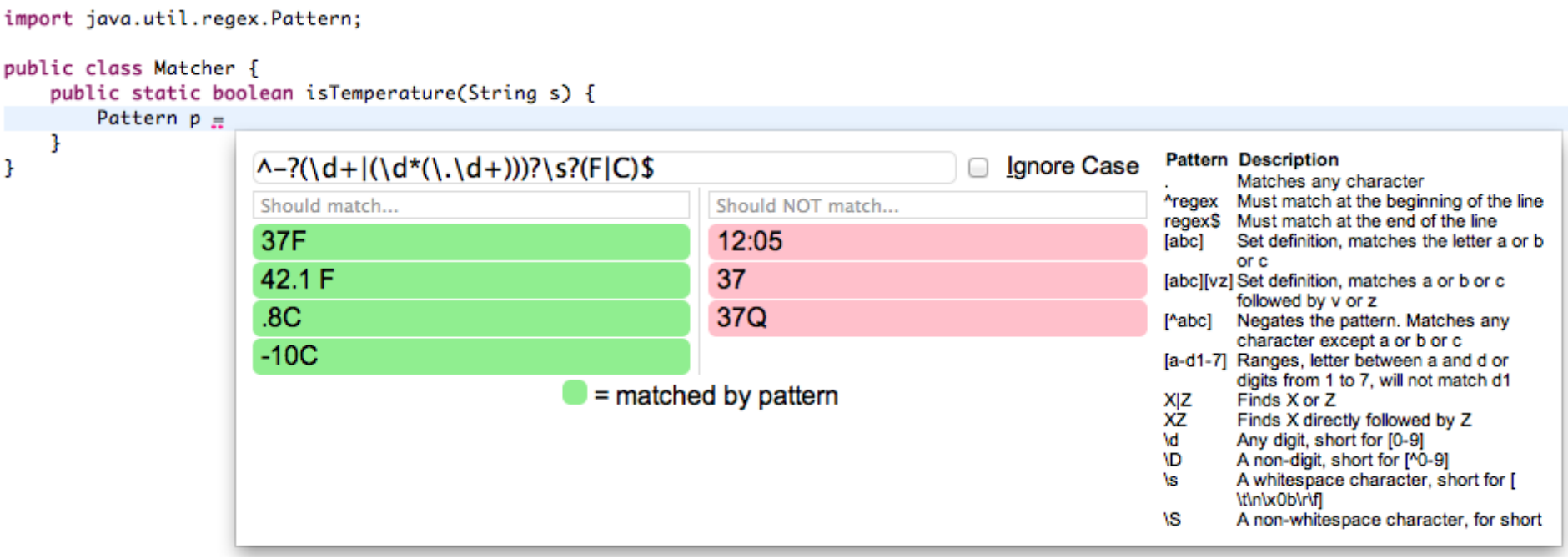
*(enter **positive** test cases above, pressing ENTER between each one)*

Should NOT match...

*(enter **negative** test cases above, pressing ENTER between each one)*

Pattern	Description
.	Matches any character
^regex	Must match at the beginning of the line
regex$	Must match at the end of the line
[abc]	Set definition, matches the letter a or b or c
[abc][vz]	Set definition, matches a or b or c followed by v or z
[^abc]	Negates the pattern. Matches any character except a or b or c
[a-d1-7]	Ranges, letter between a and d or digits from 1 to 7, will not match d1
X|Z	Finds X or Z
XZ	Finds X directly followed by Z
\d	Any digit, short for [0-9]
\D	A non-digit, short for [^0-9]
\s	A whitespace character, short for [\t\n\x0b\r\f]
\S	A non-whitespace character, for short

Active Code Completion with GRAPHITE



Active Code Completion with GRAPHITE

```
import java.util.regex.Pattern;

public class Matcher {
    public static boolean isTemperature(String s) {
        Pattern p = Pattern.compile("^-?(\\d+|(\\d*(\\.\\d+)))?\\s?(F|C)$");
        /*
         * Should match:
         *   37F
         *   42.1 F
         *   .8C
         *   -10C
         *
         * Should NOT match:
         *   12:05
         *   37
         *   37Q
         *
         */

    }
}
```

Active Typechecking & Translation

[Omar and Aldrich, presented yesterday at DSLDI 2013]

- Use types to control typechecking and translation.
- Implemented in the **Ace** programming language.

Benefits of DSLs

- Specialized **syntax** improves **ease-of-use**
- Specialized **typechecking rules** improve **verifiability**
- Specialized **translation strategies** improve **performance** and **interoperability** with existing technologies
- Specialized **tool support** improves **ease-of-use**.

Types Organize Languages

- Types represent an organizing principle for programming languages.
- Types are not simply useful for traditional verification, but also **safely-composable language-internal extensibility.**

Type-Directed, Whitespace-Delimited Parsing for Embedded DSLs


Cyrus Omar


Benjamin Chung

http://cs.cmu.edu/~dkurilov/
Darya Kurilova


Alex Potanin


Jonathan Aldrich

School of Computer Science
Carnegie Mellon University

Examples

```
val dashboardArchitecture : Architecture = ~
   external component twitter : Feed
      location www.twitter.com
   external component client : Browser
      connects to servlet
   component servlet : DashServlet
      connects to productDB, twitter
      location intranet.nameless.com
   component productDB : Database
      location db.nameless.com
   policy mainPolicy = ~
      must salt servlet.login.password
      connect * -> servlet with HTTPS
      connect servlet -> productDB with TLS
```

Wyvern DSL: Architecture Specification

```
val newProds = productDB.query(~)
   select twHandle
   where introduced - today < 3 months
val prodTwt = new Feed(newProds)
return prodTwt.query(~)
   select *
   group by followed
   where count > 1000
```

Wyvern DSL: Queries

```
serve(page, loc) where
 val page = ~
  html:
   head:
    title: Hot Products
    style: {myStylesheet}
    body:
     div id="search":
      {SearchBox("products")}
     div id="products":
      {FeedBox(servlet.hotProds())}
 val loc = ~
  products.nameless.com
```

Wyvern DSLs: Presentation and URLs