Extensible Type-Driven Parsing for Embedded DSLs in Wyvern

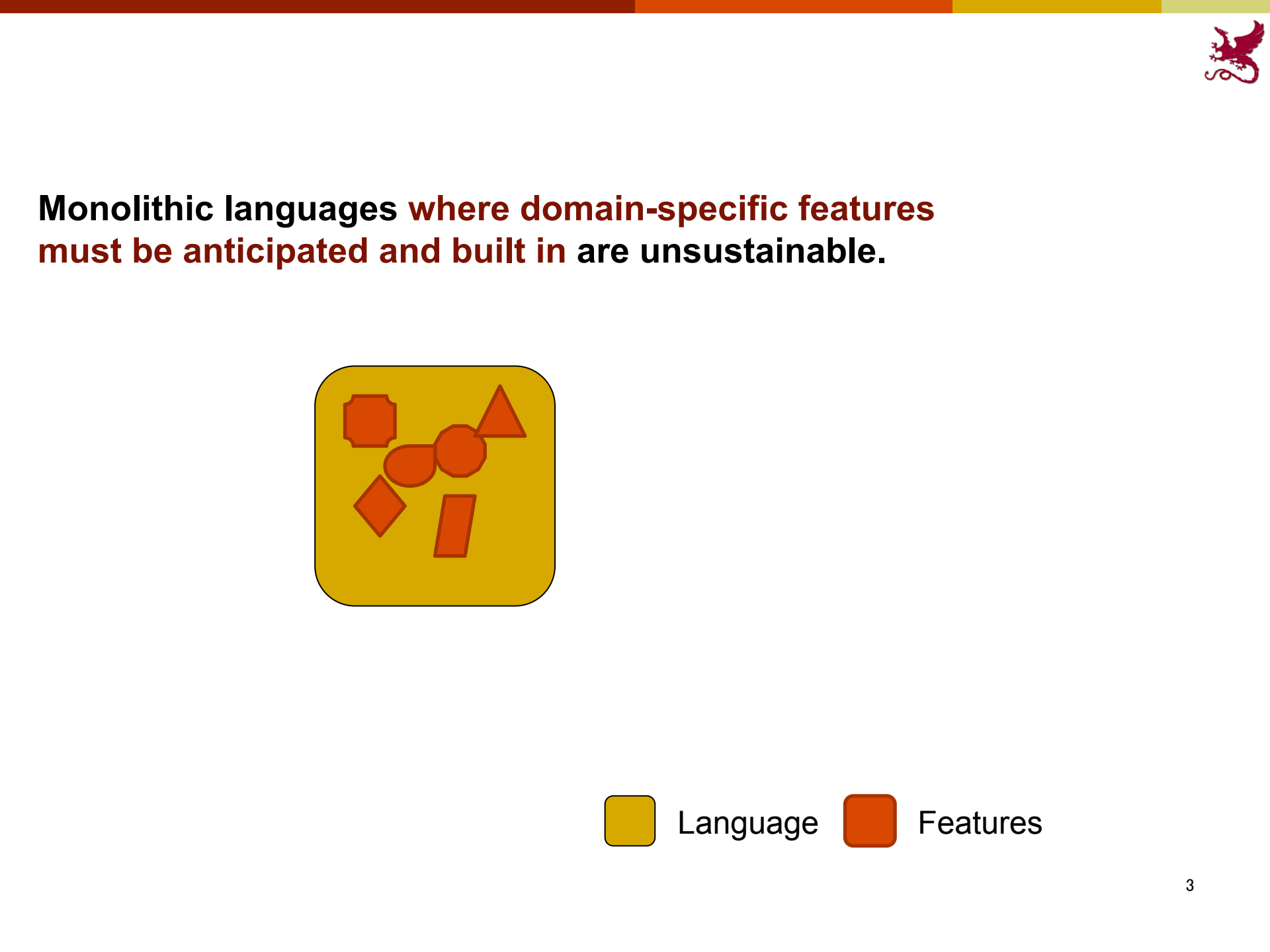
Cyrus Omar
Benjamin Chung
Darya Kurilova
Ligia Nistor
Alex Potanin *(Victoria University of Wellington)*
Jonathan Aldrich

School of Computer Science
Carnegie Mellon University

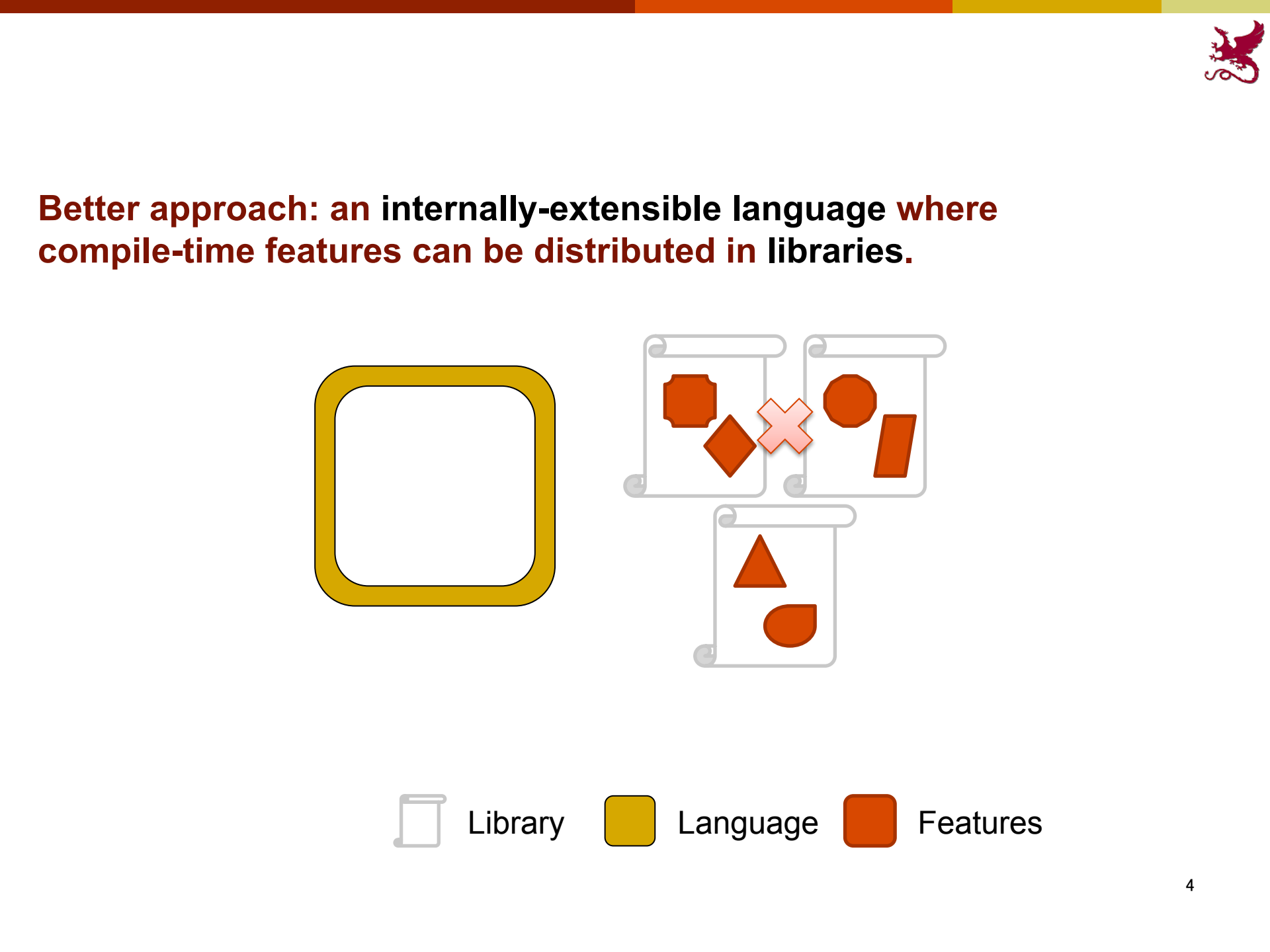
Wyvern

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

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



Monolithic languages where domain-specific features must be anticipated and built in are unsustainable.



Better approach: an internally-extensible language where compile-time features can be distributed in libraries.

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.

Example: SugarJ [Erdweg et al, 2010]

- Libraries can extend the **base syntax** of the language
- These extensions are imported **transitively**
- Extensions can **interfere**:
  - Pairs vs. Tuples
  - HTML vs. XML
  - Different implementations of the same syntax

Our Solution

- Libraries **cannot** extend the **base syntax** of the language
- Instead, **syntax is associated with types**.
- This type-specific syntax can be used to **create values of that type**.

Examples: HTML and URLs

```
serve : (HTML, URL) -> unit
```

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

Type-Specific Literals (TSLs)

- Several inline forms are available
  - `` `dsl syntax here, \`inner backticks\` must be escaped` ``
  - `'dsl syntax here, \'inner single quotes\' must be escaped'`
  - `{dsl syntax here, {inner braces} must be balanced}`
  - `[dsl syntax here, [inner brackets] must be balanced]`
  - others?
- If you use the tilde (~) with whitespace, there are no restrictions on the code inside. Layout determines the end of the block.

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]
- TSL code (and Wyvern expressions inside it) is left **unparsed** during this phase

Phase II: Typechecking and DSL Parsing

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

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

Associating a `Parser` with a type

```
type HTML =
  …
  def tagName : string
  …
  attributes = new
    def parser : Parser = new
      def parse(s : TokenStream) : AST =
        … code to parse HTML …
```

```
type Parser =
  def parse(s : TokenStream) : AST
```

Associating a grammar with a type

```
type HTML =
  …
  def tagName : string
  …
  attributes = new
    def parser : Parser = ~
      start ::= ("<" tag ">" start "</" tag ">")*
              | "{" EXP : HTML "}"
        tag ::= ...
```

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
  - 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

Types Organize Languages

- Types represent an organizational unit for programming language semantics.
- Types are not only useful for traditional verification, but also **safely-composable language-internal (syntax) extensions.**

Examples

```
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

```
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

Ongoing Work

Are all forms equivalent?

- 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

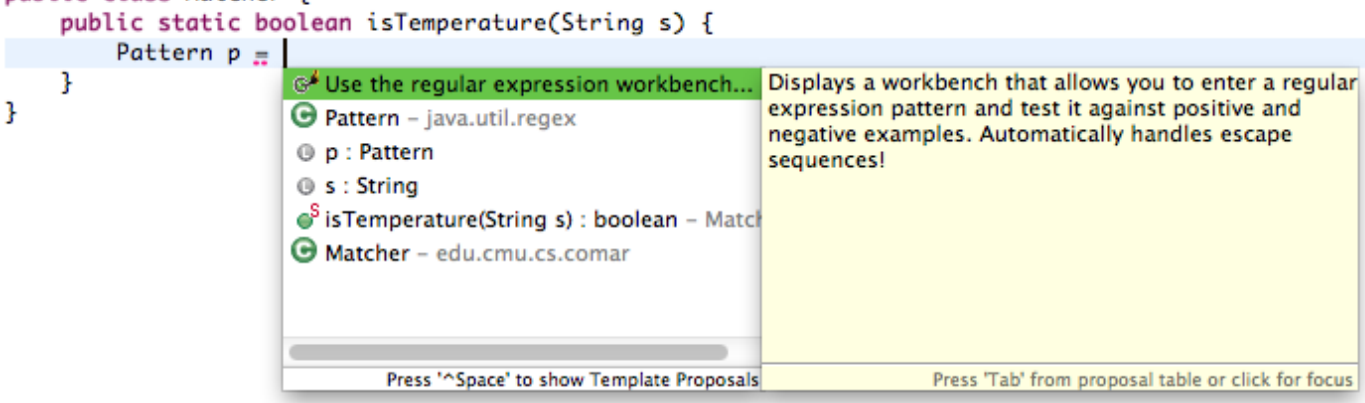
Active Libraries [Veldhuizen, 1998]

- 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...

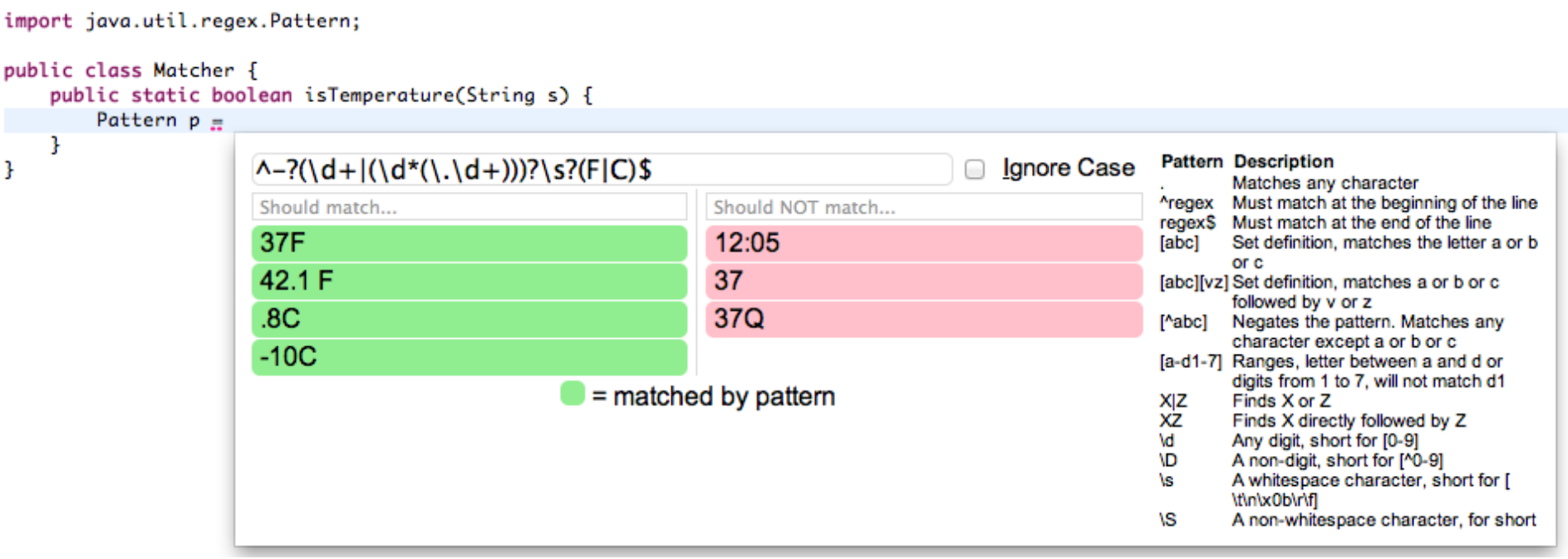
*(enter **positive** test cases above, pressing ENTER between each one)*

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 at DSLDI 2013]

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