About me

- 3rd year PhD student working in programming languages group
- Contributor to open-source projects & functional programming speaker
- Worked on the F# language during Microsoft Research internships

This proposal

- Programming languages for modern heterogeneous applications
- Developed mathematical foundations for context-aware computations
- Useless without practical implementation!

Language support for context-aware computations

Tomas Petricek
Supervisor: Alan Mycroft
University of Cambridge

Qualcomm Innovation Fellowship Proposal 2013
Modern applications

- Run in a heterogeneous environments, across multiple devices
- Must be aware of the context: GPS sensor, computing capabilities

Future directions

- **Diversity**: mobile application for multiple platforms
- **Internet of things**: even more diverse environments
- **Specialized hardware**: offload computations to GPU, FPGA?
State of the art

Ad-hoc and error-prone solutions

<table>
<thead>
<tr>
<th>Platform version</th>
<th>#if or lazy loading</th>
</tr>
</thead>
<tbody>
<tr>
<td>System capabilities</td>
<td>embedded languages</td>
</tr>
<tr>
<td>Data confidentiality</td>
<td>expensive testing</td>
</tr>
<tr>
<td>Resource &amp; data availability</td>
<td>dynamic checks</td>
</tr>
</tbody>
</table>

All are examples of contextual properties!

Heterogeneous development today

- **Platform version**: cross-compilation using #if leads to maintenance nightmare; loading components at runtime can easily lead to runtime errors
- **System capabilities**: when calling SQL or GPU, code in embedded (domain-specific) languages is translated at runtime; this translation often fails
- **Data confidentiality**: you do not want to send your database password over unsecured network - this needs to be checked at compile time
- **Resource & data availability**: does a function require GPS sensor or database access to run? what will happen if resources are unavailable?

Contextual properties

- Capture information about the context (resources, safety constraints, platform requirements and more...)
Context-aware languages

Programming language matters!

“I have now delivered three business critical projects written in [a better language]. I am still waiting for the first bug to come in.” †

Types for context-aware information

Infer and give developers immediate feedback
Compile time checking to reduce bugs

† F# Testimonials. Simon Cousins, UK power company. http://fsharp.org/testimonials

The quote

• Talk about F# - typed functional language with strong checking
• F# types are inferred – the code is concise, but safe
• F# types are powerful – for example, check units of measure
• Follow the same powerful design principles for context-aware computations

Types for context-aware information

• Integrating context at the language level enables many scenarios
• Type checking – detect bugs at compile time
• Advanced editors – give immediate code information & error feedback
Case study: Mobile news reader

```javascript
let lookupNews(location) =
    let db = query("News", password)
    selectNews(db, location)

let readNews() =
    let loc = gpsLocation()
    remote lookupNews(loc)

let iPhoneMain() =
    createCocoaWidget(readNews)

let windowsMain() =
    createMetroWidget(readNews)
```

What to check?
- Resource usage
- Platform availability
- Responsiveness
- Confidentiality

Extensible!
- Add checking for other properties

Case study

- **lookupNews** requires database – it needs to run on the server-side (remote call), **readNews** requires GPS
- **iPhoneMain** needs iOS and **windowsMain** needs Metro, but **readNews** can be shared between the two
- **readNews** makes call to the server, so it must be called on background thread to avoid blocking the user interface
- **password** is used in database connection and so it is confidential (cannot be used in all contexts)

What to track

- Set of resources used, minimal platform version required
- Not a closed set – users want to add more properties!
Theory of context dependence

Coeffect type systems

\[ \Gamma^\alpha @ \sigma \vdash e : \tau \]

Tracking security and resource usage

\[
\{ \} @ \{\text{gps, ios}\} \vdash \text{iPhoneMain()} : \text{void}
\]

\[
\text{password}^{\text{sec}} : \text{string} @ \{\text{db}\} \vdash \text{query("News", password)} : \text{News}
\]


Coeffect typing judgment

Generalization of so-called effect systems

- Effect systems capture how programs affect the world
- Coeffect systems capture what is required from the world

The expression \( e \) returns a value of type \( \tau \) provided that:

- It has variables \( \Gamma \) annotated with security information \( \alpha \)
- It is running in a context that satisfies \( \sigma \)

Two examples

- \text{iPhoneMain} call requires GPS and iOS platform and does not return anything
- \text{query} call requires database (db) access and a string variable \text{password} that must be marked as confidential
Project goals

**Practical** and **manageable** implementation

**Extensible checking** of types
- Specify structure of properties
- **Set** of resources used, version **number**

**Coeffect typing** rules
- Attach contextual information to functions
- Context required when defined or called

**Practical approach**

- Produce something that developers can play with
- It should not be a single-purpose prototype!
- Compiler tool-chain in a year is impossible...

**Extending the F# compiler**

- Annotate types with custom *checked* information
- Type checking rules for sets of resources, versions, etc.
- Embedded code blocks that track contextual information
Completed so far

- Analyze how context is used in computations
- Coeffect theory to capture context dependence

One year goals

- Extensible checking that can reason about sets, numbers, monoids, ...
- Code blocks can be translated to GPU, JavaScript, etc.
- Available via the web – I did that for my earlier project

Long-term goals

- Track contextual information for a whole program
- Cross-compiler that produces iOS, JVM, .NET, JavaScript, CUDA, etc.
Summary

**Context-aware** software needs better tools!

Programming languages are the key
We developed theoretical foundations
Practical implementation is the next step

Thank you!

Summary

- Context-awareness is fundamental – we need to integrate it at the low level
- Programming languages enable better compilers and tools
- Coeffect theory captures context-dependent properties

Fellowship proposal

- Add context-aware extensions to the F# language
- Make it available for experimentation via the web

Thank you!