

Advances in Programming Languages

APL12: Coursework Assignment, Review

David Aspinall

School of Informatics
The University of Edinburgh

Thursday 18 February 2010
Semester 2 Week 6



Outline

- 1 Course schedule
- 2 Reminder of topics
- 3 An example: Alice ML
- 4 Writing and submitting
- 5 Summary

What's in the course?

The lectures will cover four sample areas of “advances in programming languages”:

- *Specifying and statically checking behaviour of Java code*
- *Type classes in Haskell can do anything*
- **Patterns and abstractions for programming concurrent code**
- LINQ and cross-language integration in .NET

Lectures also specify reading and exercises on the topics covered. This homework is not assessed, but it is essential in order to fully participate in the course.

What's in the course?

The lectures will cover four sample areas of “advances in programming languages”:

- *Specifying and statically checking behaviour of Java code*
- *Type classes in Haskell can do anything*
- Patterns and abstractions for programming concurrent code
- LINQ and cross-language integration in .NET

Lectures also specify reading and exercises on the topics covered. This homework is not assessed, but it is essential in order to fully participate in the course.

There is substantial piece of written coursework which contributes 20% of students' course grade. This requires investigation of a topic in programming languages and writing a 10-page report with example code.

Assignment schedule

Week 1 Thursday 14 January: Topic announcement

Week 5 Friday 12 February: Intermediate report

Week 6 Thursday 18th February: Assignment review lecture

Week 8 Friday 5 March: Final report

Outline

- 1 Course schedule
- 2 Reminder of topics**
- 3 An example: Alice ML
- 4 Writing and submitting
- 5 Summary

Links: Web Programming without Tiers

Programming the Web with Links

The [Links](#) language unifies the traditional three tiers of web programming: client activity within the web page being viewed; server software directing web site responses; and a back-end database providing content and persistent storage. A single program written in Links is compiled into a different language for each tier and then automatically distributed across them as appropriate.

Links itself is functional, with a range of novel features to present a coherent programming interface for database manipulation, embedded XML, and user interaction flow.

Multiple inheritance

Multiple inheritance in Scala with traits and mixins

The `Scala` language provides *traits* and *mixins* as modularisation constructs.

Mixin composition solves the infamous *multiple inheritance* ambiguity problem: does a class `A` that inherits from `B` and from `C` implement `A.m` as `B.m` or `C.m`? Java forbids multiple inheritance but provides interfaces. However, interfaces cannot contain implementations, leading to code duplication. Scala's trait and mixin constructs remedy this.

Parallel programming in Haskell

Parallel programming in Haskell with `par` and `seq`

The original Haskell '98 language has no specific facilities for concurrent or parallel programming. However, there are several compiler extensions and libraries which make both possible. In particular, operations `par` and `seq` allow a programmer to enable parallel or sequential computation of results, and from these build more complex *strategies* for parallel evaluation across multiple cores or even distributed processors.

Haskell STM library

Software Transactional Memory in Haskell

The *STM* library for the Glasgow Haskell Compiler (GHC) provides high-level language support for coordinating concurrent computation, where multiple threads act simultaneously on shared datastructures.

Remarkably, STM does this without using locks. Instead, it uses efficient and optimistic *software transactions*, giving freedom from deadlock and promoting non-interfering concurrency. These transactions are modular and composable: small transactions can be glued together to make larger ones. Moreover, implementing this within the Haskell type system gives static guarantees that transactions are used correctly.

Asynchronous Workflows in F#

Asynchronous Workflows in F#

Microsoft's F# language provides several facilities for the building and high-level manipulation of computations and metacomputations. One of these, *workflows*, allows libraries to define domain-specific sublanguages for particular kinds of computation.

Using this, the `Async` module gives a way to write code that can execute asynchronously when necessary, without needing to explicitly describe any threads or communication. Actions that might potentially block or be long-running will automatically happen in the background, with their results retrieved as they arrive.

Outline

- 1 Course schedule
- 2 Reminder of topics
- 3 An example: Alice ML**
- 4 Writing and submitting
- 5 Summary

Futures and promises

Futures and promises in Alice ML

The *Alice ML* language is based on Standard ML, with several extensions to support distributed concurrent programming.

In particular it provides *futures* and *promises* for lightweight concurrency: a future represents the result of a computation that may not yet be available, and a promise is a handle to build your own future.

Project homepage

Alice - Mozilla Firefox

File Edit View History Bookmarks Tools Help


http://www.ps.uni-sb.de/alice/ Google

Saarland University
Informatics
Programming Systems

Home
People
Papers

Manual
Download
Contributions
Contact

Wiki
Bugs

 SPB 378

Alice

Alice 1.4 has been released!

Overview

Alice ML is a functional programming language based on [Standard ML](#), extended with rich support for concurrent, distributed, and constraint programming. Alice ML extends Standard ML with several new features:

- **Futures:** laziness and light-weight concurrency with implicit data-flow synchronisation
- **Higher-order modules:** higher-order functors and abstract signatures
- **Packages:** integrating static with dynamic typing and first class modules
- **Pickling:** higher-order type-safe, generic & platform-independent persistence
- **Components:** platform-independence and type-safe dynamic import & export of modules
- **Distribution:** type-safe cross-platform remote functions and network mobility
- **Constraints:** solving combinatorial problems using constraint propagation and programmable search

The Alice System is a rich open-source programming system featuring the following tools:

- **Virtual machine:** a portable VM with support for just-in-time compilation
- **Interactive system:** an interpreter-like interactive toplevel with easy graphical interface
- **Batch compiler:** separate compilation
- **Static linker:** type-safe bundling of components
- **Inspector:** a tool for interactively inspecting data structures
- **Explorer:** a tool for interactively investigating search problems
- **Gtk+:** a binding for the Gnome toolkit GUI library
- **SQL:** a library for accessing SQL databases
- **XML:** a simple library for parsing XML documents

Tutorial

Be our guest on the [Short Tour to Alice!](#)

http://www.ps.uni-sb.de/alice/papers.html

Downloading and installing

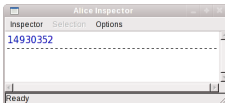
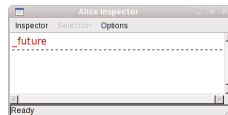
- Often not entirely trivial! Should work for topics recommended.
- Try your preferred environment/machine, resort to DICE as a fallback.
- You should have already solved any problems for your intermediate report.

For Alice, I chose to download the RPM files to install onto my Fedora Linux machine. This required first finding and installing some additional libraries, as my OS is newer than the one for which Alice was packaged.

```
wget http://www.ps.uni-sb.de/alice/download/rpm/alice-complete-1.4-1.i386.rpm
wget http://www.ps.uni-sb.de/alice/download/rpm/gecode-1.3.1-1.i386.rpm
rpm -ivh alice* gecode*
```

Trying examples


```
da@harrison:~  
File Edit View Terminal Tabs Help  
[da@harrison ~]$ alice  
Alice 1.4 ("Kraftwerk 'Equaliser' Album") mastered 2007/04/24  
### loaded signature from x-alice:/lib/system/Print  
### loaded signature from x-alice:/lib/tools/Inspector  
### loaded signature from x-alice:/lib/distribution/Remote  
- val alice = it;  
val alice : unit -> unit = _lazy  
- fun fib 0 = 1 | fib 1 = 1 | fib n = fib (n-1) + fib (n-2);  
val fib : int -> int = _fn  
- val n = spawn fib 35;  
val n : int = _future  
- inspect n;  
  
(alice:27005): Gtk-WARNING **: Unable to locate theme engine in mod  
doka",  
  
(alice:27005): Gtk-WARNING **: Unable to locate theme engine in mod  
doka",  
Gtk-Message: Failed to load module "gnomebreakpad": libgnomebreakpa  
open shared object file: No such file or directory  
Gtk-Message: Failed to load module "canberra-gtk-module": libcanber  
.so: cannot open shared object file: No such file or directory  
val it : unit = ()  
- inspect n;  
val it : unit = ()  
- inspect n;  
val it : unit = ()  
- alice();  
val it : unit = ()  
-  
That's all, said Humpty Dumpty. Good-bye.  
[da@harrison ~]$
```



Learning more about the topic

alice
manual.

Contents
Tour
Language
Library
CP Tutorial
Tools
Index



futures

Overview

A *future* is a place-holder for the undetermined result of a (concurrent) computation. Once the computation delivers a result, the associated future is eliminated by globally replacing it with the result value. That value may be a future on its own.

Whenever a future is *requested* by a concurrent computation, i.e. it tries to access its value, that computation automatically synchronizes on the future by blocking until it becomes determined or failed.

There are four kinds of futures:

- *concurrent futures* stand for the result of a concurrent computation,
- *lazy futures* stand for the result of a computation that is only performed on request,
- *promised futures* stand for a value that is promised to be delivered later by explicit means,
- *failed futures* represent the result of a computation that terminated with an exception.

Next questions:

- how do I use futures?
- what advantages do they bring? what drawbacks?
- how are they related to other language features?
- do they have well understood foundations? a good implementation?
- how and when were futures invented?

Resources

- Useful sites to search the academic literature:
<http://citeseerx.ist.psu.edu/> CiteSeer^X, formerly the best search and citation index for computer science.

Resources

- Useful sites to search the academic literature:
<http://citeseerx.ist.psu.edu/> CiteSeer^X, formerly the best search and citation index for computer science.
<http://www.informatik.uni-trier.de/~ley/db/> DBLP: an invaluable bibliography, with links to electronic editions.

Resources

- Useful sites to search the academic literature:
<http://citeseerx.ist.psu.edu/> CiteSeer^X, formerly the best search and citation index for computer science.
<http://www.informatik.uni-trier.de/~ley/db/> DBLP: an invaluable bibliography, with links to electronic editions.
<http://scholar.google.com> beware: Google's idea of an academic article is broader than most.
- Lambda the Ultimate: Programming languages weblog.
Some astonishing enthusiasm for heavy programming language theory.
- <http://developers.slashdot.org>
One channel on the self-proclaimed *News for Nerds*. Occasional programming language issues, lots of comments but can be thin on content. Good for searching for news. Beware of the trolls.
- `comp.lang.<almost-any-language>`, `comp.lang.functional`
Programming language newsgroups, some very busy. `c.l.f` has a endless supply of questioners, and some very patient responders.

One resource for everything?

Wikipedia is an invaluable first stop resource for many topics, but has a number of drawbacks for scholarly use:

- it's a wiki! — pages can change at any time, and be changed by anyone;
- it is an electronic format: a URL alone is not a sufficient citation;
- by definition, it is not a primary source: peer reviewed articles, whitepapers and system documentation will be (more) authoritative.

See Wikipedia's own entries on *caution before citing Wikipedia* and *caution on academic use of Wikipedia*.

Advances in Programming Languages

Lecture log and discussion

Lecture 8: ESC/Java2

Posted 5 February, 2009 by David Aspinall

Categories: [Lecture log](#)

ESC/Java2: a verification tool combining several analysis techniques (types, dataflow, proof). An overview of the checks it performs: exception freedom for common exceptions (null pointers, array indices, class casts). The lack of soundness and completeness: false positives and defects missed. Some further and specialised annotations extending core JML (non_null, unreachable, modifies, pure). Specification inheritance.

[Read the rest of this post »](#)

Comments: [Be the first to comment](#) [Edit](#)

Lecture 7: JML - the Java Modeling Language


Posted 2 February, 2009 by David Aspinall

February 2009

M	T	W	T	F	S	S
						1
2	3	4	5	6	7	8
9	10	11	12	13	14	15
16	17	18	19	20	21	22
23	24	25	26	27	28	
« Jan						

Links

[About this blog](#)
[Programming Language](#)
[Research Engine](#)

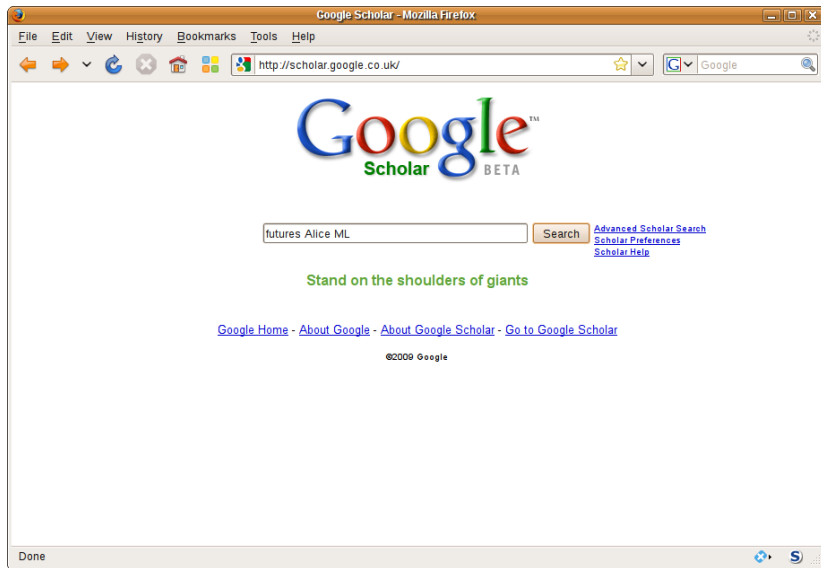
 [Lambda the Ultimate](#)

[Lisp Conference, March 22-25](#)

[Project Euler](#)

[DI Grand Challenges](#)

Finding relevant papers



Finding relevant papers

futures Alice ML - Google Scholar - Mozilla Firefox

File Edit View History Bookmarks Tools Help

http://scholar.google.co.uk/scholar?q=futures+Alice+ML&hl=en&lr= Google

Google Scholar BETA

futures Alice ML Search

[Advanced Scholar Search](#)
[Scholar Preferences](#)
[Scholar Help](#)

Scholar All articles - **Recent articles** Results 1 - 10 of about 20,100 for **futures Alice ML**. (0.03 seconds)

[PDF] [▶ Alice ML through the looking glass](#)
A Rossberg, D Le Botlan, G Tack, T Brunklaus, G ... - Trends in Functional Programming, 2005 - emn.fr
... state and **futures**. Unlike **Alice**, Oz is based on a relational core language and is not statically typed. Acute [22] is an experimental, **ML**-based language for ...
[Cited by 7](#) - [Related articles](#) - [View as HTML](#) - [Web Search](#) - [All 3 versions](#)

[A concurrent lambda calculus with futures](#) - [▶ inria.fr](#) [PDF]
J Niehren, J Schwinghammer, G Smolka - Theoretical Computer Science, 2006 - Elsevier
... The final step by handle.bind binds the **future** z to itself. Analogously, longer cyclic chains of **futures** may be constructed. **Futures** in **Alice ML**: There are two ...
[Cited by 26](#) - [Related articles](#) - [Web Search](#) - [BL Direct](#) - [All 17 versions](#)

[A Randomized Controlled Trial to Reduce Cancer Risk Through African-American Churches](#)
G Corbie-Smith, AS Ammerman, ML Katz, DMM St. ... - J Gen Intern Med, 2003 - Blackwell Synergy
... Giselle Corbie-Smith, MD, MSc.; **Alice** S. Ammerman, DrPH, RD.; Mira L. Katz, PhD ... will be helpful in guiding the design and implementation of **future** CBPR efforts. ...
[Cited by 32](#) - [Related articles](#) - [Web Search](#) - [BL Direct](#) - [All 6 versions](#)

[Alice Through the Looking Glass](#) - [▶ 33367.com](#) [PDF]
A Rossberg, D Le Botlan, G Tack, T Brunklaus, G ... - Intellect, 2006 - books.google.com
... **Alice**, supporting distributed state and **futures**, for the ... SLW+ 04] is probably closest in spirit to **Alice**. It is an experimental **ML**-based language for typed ...
[Cited by 14](#) - [Related articles](#) - [Web Search](#) - [All 7 versions](#)

[A Cross-Cultural Analysis of the Behavior of Women and Men: Implications for the Origins of Sex ...](#) - [▶ duke.edu](#) [PDF]
W Wood, AH Eagly - PSYCHOLOGICAL BULLETIN, 2002 - content.apa.org

Done

First references for Alice ML

- The online Alice ML documentation is excellent for potential users. The papers explain the design and implementation of the language.
- The first paper is technical (but fun for typed λ -calculus fans). The second is a practical overview of Alice ML language features.

References

- Andreas Rossberg. *Alice Manual: A Tour to Wonderland*. At <http://www.ps.uni-sb.de/alice/manual/tour.html>. Retrieved on 8th February 2009, at 22:00 UTC.
- Joachim Niehren, Jan Schwinghammer, Gert Smolka: *A concurrent lambda calculus with futures*. Theoretical Computer Science. 364(3): 338-356 (2006)
- Andreas Rossberg, Didier Le Botlan, Guido Tack, Thorsten Brunklaus, Gert Smolka: *Alice through the looking glass*. Trends in Functional Programming 2004: 79-95.

Further references for Alice ML

For AliceML, there are many papers provided on the home page, with publication dates between 2001-2007.

What about the direct influences that lead to the features of AliceML being studied?

Is there any relevant work which has been published since, building on AliceML? Or interesting case studies or industrial applications?

Further References

- Halstead, R. H. 1985. *MULTILISP: a language for concurrent symbolic computation*. ACM TOPLAS 7, 4 (Oct. 1985), 501-538.
- caf - Concurrency Abstractions using Futures. At <http://sites.google.com/site/cafwiki/>. Retrieved on 17th February 2010, at 16:35 UTC.
- ...

Outline

- 1 Course schedule
- 2 Reminder of topics
- 3 An example: Alice ML
- 4 Writing and submitting**
- 5 Summary

Report formats

Reports must be submitted electronically as a PDF document. The recommended method for creating these is [pdflatex](#) with the [article](#) document class.

In addition, [OpenOffice](#) is freely available for Windows and Linux, installed on Informatics machines, and can write PDF. Mac OS X natively creates PDF. Microsoft provide PDF output as a plugin for Word 2007.

Submission instructions are on the coursework web page.

Please use recommended filenames!

Final report: recommended outline

Heading Title, date, author

Abstract This report describes ...

Introduction Content summary, overview of report structure

Context The problem domain

Main topic What it is, how it works; advantages and limitations

Example Annotated code, explanation, screenshot

Salt: the example must in some way concern books or reading (library catalogue, author database, electronic book store, ...).

Resources For notable resources used (article, tutorial, manual), give a summary in your own words of what it contains

Related work Other approaches to the problem

Conclusion What <topic> does, good and bad points

Bibliography **Full references** for all resources used

Total 8–10 A4 pages. See course coursework web page for further details.

Exemplar reports

Futures and Promises in Alice ML

March 13, 2008

Abstract

Various languages have tried to allow useful tools and solutions to concurrent programming. Alice ML is one of these. Alice ML uses futures and promises to help solve the problem of data synchronisation and concurrency. This report explains how these features work, displays a detailed example, performs comparisons to other language solutions and presents related work in this area.

1 Introduction

Computer technology is changing and increasing all the time. Multi core systems are being produced yet to truly exploit this feature of systems, programs will have to be parallelised. This leads to the development of concurrent programming. Concurrent programming is difficult to do well in practice. The concepts and benefits of a good concurrent program are worthwhile. Increased speed, efficient use of resources, better user response time etc are a few of the advantages.

A dominant solution to concurrent programming seems to be threads. This solution is not perfect and many languages have tried to overcome the many disadvantages and difficulties posed by this solution.

Alice ML is an extension of Standard ML and supports concurrency by the use of futures and promises. This solution to concurrent programming appears to be a valid and reasonable one. This report explains futures and promises in Alice ML in more detail, it provides examples and mentions the advantages and disadvantages to this solution. Other language solutions to the concurrent programming problem are also mentioned and compared to Alice ML.

1

Regular Expression Types and Patterns in CDuce

Advances in Programming Languages

Paul McEwan (0452900)

14/03/2008

Abstract

This report examines the CDuce language, a typed functional programming language designed for general purpose programming. Unlike other functional programming languages, CDuce incorporates native support for XML documents in the language. This report looks at the language, related work and then at the use of regular expression types and patterns. Specifically, how these particular types and patterns are used to query/manipulate the XML data, as well as allow static checking by the compiler that the XML data used is always valid.

1. Introduction

The CDuce language is a functional, typed programming language allowing the creation of general purpose programs. The key difference between CDuce and other typed, functional programming languages (such as Haskell and ML) is that it was designed to be used with XML from the start. The language has features included that allow the programmer to manipulate and query XML trees directly in the code, instead of using additional tree parsers (such as Document Object Model (DOM) parsers). The language allows for XML files to be read in or created directly in the code and be exported back to a file. The XML handled by programs written in CDuce is guaranteed to be valid (both well-formed/syntactically correct and corresponding to a specific structure).

Of particular interest, the CDuce language allows the inclusion of regular expressions when defining types and patterns. The use of these regular expressions allows the programmer to not only enquire and alter the XML, it also allows the compiler to perform checks statically on the code to ensure the XML is valid. This report first examines the CDuce language at a high level (§2), then a look at some related work (including languages which inspired the creation of CDuce) (§3). The use of regular expressions in patterns and types will then be examined (§4) with an example program created using these features of the language (§5). Finally, the report will be concluded by looking back at the use of regular expressions in the types and patterns of the CDuce language (§6).

2. CDuce

The paper [CDuceXGPG] presents CDuce in detail and is an ideal resource to use to understand the language. Here, points raised in the paper shall be summarised in order to present a general introduction to the CDuce language: what it is, how it came about and how it works.

As stated previously, the CDuce language is a typed functional programming language designed to allow the processing of XML data directly in the language while still being a "general purpose" language (i.e. not specific to XML processing but allowing programs to be created that include the functionality). The CDuce project was an off-shoot extension of the XDuce language, but was designed to be less "XML-centric". To this effect, the CDuce language extends upon XDuce by addressing - what the paper referred to as - limitations in three areas:

• Type System

The XDuce type system allowed the user to create types specific to dealing with XML data, which included having "regular expression types" and "type-based patterns". The use of the

1

See the course web page for two good submissions from previous students on the course.

Suitable working practices

Working practices

- Start with a blank document; all the words must be yours.
- Do not cut and paste from other documents.
 - Except for direct quotations, which must have source declared.
- Do not let others read your text; nor read theirs.

Aims of this coursework

- To learn about the chosen topic
- To improve researching and learning skills
- To demonstrate said knowledge and skills

The tangible outcome is a document, composed and written by you, demonstrating what you have learnt.

Outline

- 1 Course schedule
- 2 Reminder of topics
- 3 An example: Alice ML
- 4 Writing and submitting
- 5 Summary

Summary

Topics

- Links: Web Programming without Tiers
- Multiple inheritance in Scala
- Parallel programming in Haskell
- Software Transactional Memory in Haskell
- Asynchronous Workflows in F#

Intermediate report

- Topic choice, three initial references, screenshot.
- Complete. All submissions OK.

Final report

- Introduction and discussion of the topic;
- example annotated code, screenshots;
- resources consulted and related work;
- concluding summary and opinions;
- bibliography with proper references.
- Due: Friday 5th March.