(An Extension to Haskell / Curry) / (Functional Logic Programming Languages) / (Embedding Prolog in Haskell)

A Thesis Proposal by

Mehul Chandrakant Solanki

230108015

solanki@unbc.ca

28 June 1990

Submitted to the graduate faculty of the

MCPS

in partial fulfillment of the requirements

for the Thesis Proposal and

subsequent Ph.D. in Computer Science

Committee Members:

Dr. David Casperson, Committee Chair

Dr. Alex Aravind

Dr. Mark Shegelski

Outline

Abstract

This paper proposes a different approach to improving and broadening the power, expressibility and capability of the purely functional programming language Haskell by combining and extending the methodologies of embedding of languages into one another and also marrying / merging / combining different programming paradigms. The proposal discusses the act of extending Haskell with logic programming capabilities similar to those of Prolog, a logic programming language. The embeddings and paradigm integrations are more or less towards declarative languages.

This paper proposes a modified approach to improving and broadening the power, expressibility and capability of the purely functional programming language Haskell by combining and extending the methodologies of embedding languages into one another and also marrying / merging / combining different programming paradigms. The proposal discusses the process of extending Haskell with logic programming capabilities similar to those of Prolog, a logic programming language. The embeddings and paradigm integrations are more or less towards declarative languages.

Mehul Chandrakant Solanki
230108015
solanki@unbc.ca

1 Embedding a Programming Language into another Programming Language

Embedding a programming language into another, in this we talk about embedding Prolog in Haskell. The following are the sources or related work that can be found,

1.1 The content on Blogs / Articles / Internet Discussions

1. Lambda The Ultimate, The Programming Languages Weblog,
   `http://lambda-the-ultimate.org/node/112`

2. Takashi's Workplace (Implementation),
   `http://propella.blogspot.in/2009/04/prolog-in-haskell.html`

3. Mini Prolog for Hugs 98 (Implementation), [28]
   The first attempt at embedding Prolog in Haskell, there is not documentation as such. No paper was published either, it was just another unofficial attempt at replicating Prolog implementations in other languages like Lisp, Scheme etc. Again it is labelled to be a "Mini Prolog" and was riginally made for Hugs 1.3 and then updated for Hugs 98. Hugs is not active in development anymore, the last release was for 2006 and mostly everything these days is in GHC/GHCi. The special libraries and other Haskell files are required to run it. So not exactly "new" and also not "happening" anymore.

   Thsi implementation is a complex, because it deals with a lot literature and all of how Prolog Engine works, called Andorra Prolog.

   There is nothing such as out traditional list data structure in the form we know it. We cannot use something like [1,2,3] we have to forcible use, (Cons 1 (Cons 2 (Cons 3 nil))). There are three engines, Lazy Engine(Pure Engine), Andorra Engine and Stack Engine. The Lazy engine can construct and traverse infinite trees because its lazy.

4. Logic Programming in Haskell,
   `http://www.haskell.org/haskellwiki/Logic_programming_example`

5. Haskell vs. Prolog comparison,
   `http://stackoverflow.com/questions/1932770/haskell-vs-prolog-comparison`

6. Haskell vs Prolog, or "Giving Haskell a choice"
   `http://echochamber.me/viewtopic.php?f=11&t=35369`

7. Killing Prolog and losing its steam,
   http://vanemden.wordpress.com/2010/08/21/who-killed-prolog/
   http://www.kmjn.org/notes/prolog_lost_steam.html

1.2 Related Books

1. The Reasoned Schemer, Daniel P. Friedman, William E. Byrd, Oleg Kiselyov
2. Programming Languages: Application and Interpretation, Shriram Krishnamurthi,
   Chapters 33-34 of PLAI discuss Prolog and implementing Prolog

1.3 Related Papers

- Papers from People

  1. Type Logic Variables, K Classen,
     http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.37.2565&rep=rep1&type=pdf
  2. A Type-Safe Embedding of Constraint Handling Rules into Haskell Wei-Ngan Chin, Martin Sulzmann and Meng Wang
     http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.120.3928&rep=rep1&type=pdf
  3. Prological Features in a Functional Setting Axioms and Implementation, R Hinze
     http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.40.1016&rep=rep1&type=pdf
  4. FUNCTIONAL PEARL Combinators for breadth-first search, Micheal Spivey,
     http://journals.cambridge.org/action/displayFulltext?type=1&fid=59750&jid=JFP&volumeId=10&issueId=04&aid=59749
  5. Escape from Zurg: An Exercise in Logic Programming, Martin Erwig
     http://thelacktheref.org/docs/library/cs/functional/Erwig,%20Martin:%20Escape%20from%20Zurg%20-%20An%20Exercise%20in%20Logic%20Programming.pdf
     http://web.engr.oregonstate.edu/~erwig/zurg/

- Papers from Mike Spivey and Silvija Seres

1. Embedding Prolog in Haskell / Functional Reading of Logic Programs, [57]
   `http://spivey.oriel.ox.ac.uk/mike/silvija/seres_haskell99.pdf`

   This is one of the very first attempts to implement Prolog in Haskell, though there have been attempts and / or implementations of Prolog in other languages like Java(GNU Prolog, ISO Prolog as a library), Scheme(Scheme Prolog 1.2, pure Prolog interpreter, late 1980's early 1990's, 1993), Lisp (LogLisp 1982, QLog 1982) among others. There is a Hugs 98 implementation for Prolog(Mini Prolog, 1991-1996) for Hugs 1.3, but there has been no published work.

   The references of this paper fall into the following categories,

   - Surveys / Papers / Thesis about merging Functional and Logical Paradigms, 1,2,5,10,14,16.
   - Functional Logic Languages / Embeddings, 4,6,8,9,13,17,18.
   - Monads and Lazy Evaluation, 12,22,23.
   - Follow up / Related Papers, 19,20,21.
   - Unclassified, 14,15.

   The key points from the paper,

   (a) Prolog Predicate $\rightarrow$ Haskell Function.
   (b) Work on lazy lists, take required input produce solutions and pass it as stream.
   (c) Logical Operations $\rightarrow$ Haskell Operations implemented using concat and map.
   (d) No extension, similar to LOGLisp(strict).
   (e) Functions to support, unification, resolution and search.
   (f) This is not a FLPL, it more of a functional language with logic capabilities, so there is no Narrowing or Residuation which are the key features of a FLPL.
   (g) The principles are general for embedding.
   (h) Only declarative features of Prolog have been implemented, no cut, assert, retract, fail(??).
   (i) Minimalistic extension, only four functions, Disjunction $\|$, Conjunction &, Unify $\doteq$, Existential Quantifier (exists).
   (j) Converting a logical predicate into a pure Haskell function, bind local variables with explicit quantifiers and combining all clauses into a single equation.

(k) Algorithm,

Input → Predicate + Knowledge Base

Output → Stream of Answers

Done Lazily

(l) Prolog Terms are untyped.

(m) The function definitions are relatively simple and backtracking is naturally simulated as the evaluation is lazy.

(n) Support for BFS is included.

(o) The paper claims that other implementations or attempts like Babel, Kernel-LEAF, Escher, Curry **"lack semantic clarity"** (I would have to look into that).

(p) The paper also suggests that the level of abstraction is the same as other embeddings like LOGLisp and QLog.

(q) No implementation only Theoretical Model.

(r) No higher order functions and nested functions.

2. Algebra of Logic Programming, [53]

`http://spivey.oriel.ox.ac.uk/mike/silvija/seres_iclp99.pdf`

The previous paper on embedding a logical language in a functional language [57], two computation models have been proposed, one which is very Prolog like and uses Depth First Search while the other uses Breath First Search. This paper proposes a General Model, independent of the search strategy and which produces the same results.

The abstract semantics help in reasoning and specification while operational semantics help with execution of the program.

??? Herbrand Model ???

Logical Primitive == Haskell Function

The paper claims that their "Embedding Approach" has the "Full Power" of "Functional Logic Languages", ???????????

All the same stuff about the embedding is mentioned again,

DFS : Stream based approach produces results with definite order and multiplicity, just like Prolog. Though & and ‖ do not have all the properties, they can be achieved by using Bags / Sets instead of Streams. The cost of then answers does not matter. The **not** and the **cut** operator has been included. The cut here is not exactly the *cut* in Prolog??????????

BFS: The cost of an answer is the number of resolution steps it takes. A matrix of bag of answers is returned, each bag contains the answers with the same costs. So each node in the tree gets pushed one level down, this "root node". The functions are modified to work with "Matrices" instead of "Streams".

The differences and similarities are highlighted which help in reasoning about the their integration.

General Model: Working with "Forests" rather than "Streams" / "Matrices". They store the cost of each answer which is equivalent to the depth of the tree and then everything gets pushed one step down just like BFS.

$\|$, *not, false* remain the same,

true, &, cut need to be modified to work with forests.

The Monads for the same are,

Model	Map	Return	Join
Stream (DFS)	map	[-]	concat
Matrix (DFS)	mmap	[[-]]	shuffle
Forest (General)	fmap	Leaf -	fgraft

Some other stuff is about Kleisli Composition, $join_T$ is replaced by $\star_T == \text{true}$ (return function).

??????**Extended Monad**??????

(map, return / unit, ???, ???, Kleisli Composition)

$T^+ = (map_T, true_T, false_T, \|_T, \&_T)$

We will have $Stream^+$, $Matrix^+$, $Forest^+$

The above are the Objects of the Category, next the morphisms, specific functions are given which do the following,

DFS $\rightarrow$ Query $\rightarrow$ Stream

BFS $\rightarrow$ Query $\rightarrow$ Matrix

General $\rightarrow$ Query $\rightarrow$ Forest

Forest $\rightarrow$ dfs $\rightarrow$ Stream $\vee$ Forest $\rightarrow$ bfs $\rightarrow$ Matrix

3. The Algebra of Logic Programming,

   `http://spivey.oriel.ox.ac.uk/mike/silvija/seres_thesis.pdf`

4. Optimisation Problems in Logic Programming : An Algebraic Approach,

http://spivey.oriel.ox.ac.uk/mike/silvija/seres_lpse00.pdf Not related to the topic.

5. Higher Order Transformation of Logic Programs,

   http://spivey.oriel.ox.ac.uk/mike/silvija/seres_lopstr00.pdf

   This paper mainly talks about the "compositional approach" to design algorithms in functional programming languages which can be extended to logical programming languages. The idea is to develop a general technique for developing efficient predicates. The transformational technique is the rules and strategies approach for logical programming from another paper,

Rules(Performing Operations)	Strategies(Meta Rules / Sequencing)
Unfold Clause Definitions	Goal Tupling
Create Clause Definitions	Goal Generalization
Delete Clause Definitions	Unnecessary Variable Elimination
Re-arrange Clause Definitions	Predicate Fusion

   The above gives a compositional approach to transform logical programs ????

   Only generalisation and tupling are required to derive Herbrand Model of the two programs?????

   Standard dfs approach does not give any clear measure of computational complexity.

   The Algebra of Functional Programs says that the functions *foldl* and *foldr* give a general transformation strategy, i.e. with higher order functions. This paper takes the above results and tries to apply it to Logic Programming by translating Prolog programs into Haskell programs which helps in "Reasoning" and "Higher Order Predicates can be implemented as Higher Order Functions". Moreover with Higher Order Functions we do not need Higher Order Unification. With all of the stuff from [57], and the proof of uniqueness of fixed points ???? in [51] ???.

   The paper gives two examples of a program with two variants differing in complexity, but are proved to be "equal".

   Bird and de Moor provide synthesis and transformation techniques for functional programs which are "logicalized" in the paper.

   They say the future is to extend and apply the techniques to "constraint programming".

   and also

   "Cross Fertilized" Program Transformation Techniques for both the Declarative Paradigms

???

6. The Algebra of Searching, [56]

   http://spivey.oriel.ox.ac.uk/mike/silvija/seres_carh99.pdf Looking at a program declaratively, reveals the Logic while the procedural reading provides the Control Information. A Prolog program can be executed using different search strategies, so there should be some logic which takes into account execution / control information.

   Logic Programs are semantically composed of $\cap$ and $\vee$.

   ?????? **The main advantage of "shallow" embedding of Prolog in a Lazy Functional Programming Language over a "deep" embedding i.e. an interpreter that treats logic programs as syntactic objects** ?????

   Some more same stuff about DFS ..... again, like it can get stuck in a infinite branch of a program.

   Some stuff about search trees,

1.4 Related Libraries in Haskell

- Prolog Libraries

  1. Nano Prolog
  2. Prolog
  3. cspm-To-Prolog
  4. prolog-graph and prolog-graph-lib
  5. hswip,

     https://groups.google.com/forum/#!topic/haskell-cafe/3vmCuw7NlWE

- Logic Libraries

  1. logict,

     http://okmij.org/ftp/Computation/monads.html

  2. logic-classes
  3. proplogic
  4. cflp
  5. logic grows on trees

- Unification Libraries
    1. unification-fd
    2. cmu
- Concatenative Programming Libraries
    1. peg
- Constraint Programming and Constraint Handling Rules
    1. monadiccp
    2. monadicccp-gecode
    3. csp
    4. liquid fix point

1.5 Possibly Related Content

1. Unifying Theories of Programming, C.A.R. Hoare,
   `http://www.unifyingtheories.org/`
2. Unifying Theories of Programming with Monads, Jeremy Gibbons,
   `http://www.cs.ox.ac.uk/people/jeremy.gibbons/publications/utp-monads.pdf`

2 Multi Paradigm Languages (Functional Logic Languages)

In this section we talk about marrying or integrating the paradigms, multi paradigm programming language approach. Here we talk about combining the two most important and widely spread declarative paradigms, Functional and Logical Programming Paradigms.

2.1 Some Multi Paradigm Languages

Now, these days if one tries to classify programming languages according to paradigms then , a programming language will always end up being "multi paradigm"(I do not agree with this but a lot of people always tell me, including some unbc profs).

1. Scala, Object Functional Programming Language.
2. Virgil, Object Functional Programming Language.
3. CLOS, Common Lisp Object System.
4. .....????????

2.2 The content on Blogs / Articles / Internet Discussions

- Multi Paradigm Languages

    1. Wikipedia Multiparadigm Programming Languages
       `http://en.wikipedia.org/wiki/Multi-paradigm_programming_language#Multi-paradigm`
       `http://en.wikipedia.org/wiki/List_of_programming_languages_by_type#Multiparadigm_languages`
    2. Mozilla Developer Network MDN,
       `https://developer.mozilla.org/en-US/docs/multiparadigmlanguage.html`
    3. Some blog called c2,
       `http://c2.com/cgi/wiki?MultiParadigmProgrammingLanguage`

- Functional Logic Programming Languages

    1. FLPL Wikipedia,
       `http://en.wikipedia.org/wiki/Functional_logic_programming`
       `http://en.wikipedia.org/wiki/Category:Functional_logic_programming_`

languages

2. Implementation of Functional Logic Languages

   http://web.cecs.pdx.edu/~antoy/research/flp/

3. Functional Logic Programming

   http://www.informatik.uni-kiel.de/~mh/FLP/

2.3 Functional Logic Programming Languages

2.4 People

There are a lot of people working on this but, I found a lot of papers of two of them,

1. Michael Hanus,

   http://www.informatik.uni-kiel.de/~mh/

2. Sergio Antoy,

   http://web.cecs.pdx.edu/~antoy/

3. Uday S Reddy

2.5 Functional Logic Programming Language

1. The intergration of functions into Logic Programming : From Theory to Practice,

   http://www.informatik.uni-kiel.de/~mh/publications/papers/JLP94.html

2. Functional Logic Programming : From theory to curry,

   http://www.informatik.uni-kiel.de/~mh/papers/GanzingerFestschrift.pdf

3. Functional Logic Programming,

   http://dl.acm.org/citation.cfm?doid=1721654.1721675

4. A Higher Order Rewriting Logic for FLP, http://books.google.ca/books?hl=en&lr=&id=TSJDeaVpJyMC&oi=fnd&pg=PA153&dq=functional+logic+programming&ots=Ikp3Y-kZRV&sig=j7XQq-Hi-utdeNG54ZFkE1BeBNw#v=onepage&q=functional%20logic%20programming&f=false

5. Toy a multiparadigm declarative system

6. A unified computation model for functional and logic programming

7. Semantics and Types in Functional Logic Programming
8. Polymorphic Types in FLP
9. A general Computation Scheme for Constraint Logic Programming

List of Functional Logic Languages

1. Mercury
2. Curry
3. Escher
4. ALF
5. Babel
6. Ciao
7. Curry
8. Life
9. LPG
10. Mercury
11. NUE-Prolog
12. Oz (Mozart Programming System)
13. TOY
14. λ Prolog
15. Visual Prolog (They claim it is a OOFLPL)
16. TABLOG

3 Introduction

The main focus of the

3.1 Problem Statement

The problem is that "Prolog is dead", not many people use it and most of the times when it used, the variant is usually "DOWN AND DIRTY PROLOG" and that also many a times in academia. But there are a lot of good things about prolog that should not die away. Moreover, Prolog is ideal for search problems.

So the question is how to have all the good qualities of Prolog without actually using Prolog?

How can the two most important Declarative Programming Paradigms be brought together, in this case the idea is to bring Haskell, a Purely Functional Programming Language, one step closer to being something like Prolog, a Logical Programming Language.

Programming languages pop up from time to time. The number of languages today is in the hundreds or even thousands. Not all of them survive or end up being scarcely used. But many a times the case is that even though a language has a lot going for it the reluctancy to change. Other reasons could be that the need dies out or the language is unable to adapt to the changing requirements.

Flipping the coin to the other side we see, the more specific the language the easier it is to solve the problem. The simple reason being that, the problem need not be moulded according to the capability of the language. For example a problem with a naturally recursive solution cannot take advantage of tail recursion in many imperative languages. Many problems require the system to be mutation free, but have to deal with uncontrolled side-effects and so on.

So putting the above together, Domain Specific Languages are pretty good in doing what they are designed to do, but nothing else, resulting in choosing a different language every time. On the other hand, a general purpose language can be used for solving a wide variety of problems but many a times, the programmer ends up writing some code dictated by the language rather than the problem.

The solution, a programming language with a split personality, in our case, sometimes Functional, sometimes Logical and sometimes both. Depending upon the problem, the language shapes itself accordingly and exhibits the desired characteristics. The ideal situation would be a language with a rich feature set and the ability to mould itself according to the problem. A language with ability to take the appropriate skill set and present it to the programmer will reduce the hassle of jumping

between languages and / or forcibly trying to solve a problem according to a paradigm.

The subject in question here is Haskell and the split personality being Prolog. How far can Haskell be pushed to dawn the avatar of Prolog ? is the million dollar question.

The above will result in a set of characteristics which are from both the declarative paradigms

This can be achieved in two ways,

1. Embedding

   Please see Chapter 8,

   Embedding a Programming Language into another Programming Language.

   This approach involves, translating a complete language into the host language as an extension such as a library. The result is very shallow as all the positives as well as the negatives are brought into the host language. The negatives mentioned being, that languages from different paradigms usually have conflicting characteristics and result in inconsistent properties of the resulting embedding. Examples and further discussion on the same is provided in

2. Paradigm Integration

   Please see Chapter 11,

   Unifying or Marrying or Merging or Combining Programming Paradigms or Theories

   This approach goes much deeper as it does not involve a direct translation. An attempt is made by taking a particular characteristic of a language and merging it with the characteristic of the host language in order to eliminate conflicts resulting in a multi paradigm language.

3.2 Research Approach and Contributions

3.2.1 Contributions

3.3 Thesis Statement

The aim of this thesis is to add and / or extend the logical capabilities of the purely functional programming language Haskell which are derived from the logic programming language Prolog.

3.4 Proposal Organization

3.5 The Plan

4 Background

This section gives a broad overview of the approaches that are adopted in order to tackle the problem discussed.

Programming Languages that fall into the same family, in our case the family of Declarative Programming Languages, can be of different paradigms and can have very contrasting, conflicting characteristics and behaviours.

The two most important paradigms being Functional and Logical Programming Paradigms and the languages in question being Haskell and Prolog respectively. Some differences would Haskell uses Pattern Matching while Prolog uses Unification, Haskell is all about functions while Prolog is on Horn Clause Logic.

Prolog is chosen because it is the most dominant Logic Programming Language and has spawned a number of distributions from academia to industry with the number being near the best part of 20.

Haskell is one the most popular Functional Languages around and is the first language to incorporate Monads for safe IO. Haskell computes results lazily and is strongly typed.

The languages taken up are contrasting in nature and bringing them onto the same plate is tricky. The differences in typing, execution, working among others lead to an altogether mixed bag of properties.

The selection of languages is not uncommon and this not only the case with Haskell, Prolog seems to be the all time favourite for "let's implement Prolog in the language X for proving it's power and expressibility". Prolog has been implemented in other languages like Scheme, Lisp, Java and the list goes on and on.

Over time there has been an approach that branches out, which is Paradigm Integration. A lot of work has been done on Unifying the Theories of Programming. All sorts of hybrid languages which have characteristics from more than one paradigm are coming into the mainstream.

Consider the Object Functional Programming Language Scala, it is purely functional but with objects and classes.

With the above in mind, coming back to the problem of implementing Prolog in Haskell. There have been quite a few attempts to "merge" the two programming languages from different programming paradigms. The attempts fall into two categories as follows, 1. Embedding, where Prolog is merely translated to the host language Haskell. 2. Paradigm Integration, developing a hybrid programming language i.e. a Functional Logic Programming Language.

The above will be discussed in the chapters to come.

5 Proposed Work

As discussed in the sections above, either an embedding or integration approach is taken up for the job. So there is either a very shallow approach which does not fully utilize the constructs available in the host language and just results in a mere translation of the characteristics. While the other is the fairly complex process of which results in tackling the conflicting nature of different programming paradigms, resulting in a toned down compromised language that neither takes advantages of either paradigms, sure the both the sides need to integrated but integrated languages have never really worked. Mostly the trend is to build a library for extension to replicate

6 Related Work

6.1 Related terms

1. Prolog in Haskell
2. Embedding One language into another language
3. Constraint Programming
4. Constraint Handling Rules
5. Concatenative Programming
6. Functional Logic Programming Languages
7. Residuation
8. Narrowing
9. Warren Abstraction Machine

6.2 Prolog Libraries in Haskell

1. Nano Prolog, [60] This is basically a very small interpreter for Prolog. Feed in a prolog file and, the clauses are read and an REPL asks for a goal. No good list support No practical Prolog features No Monads Nothing special here, right now
2. Prolog, [49] The best attempt at embedding Prolog in Haskell, it comes equipped with a quasi qouter, parser, monads and cuts. Does not recognize all forms of lists that Prolog supports.
3. cspm-To-Prolog
4. prolog-graph and prolog-graph-lib
5. hswip
6. Embedding Prolog in Haskell, JM Spivey, `\*http://spivey.oriel.ox.ac.uk/mike/silvija/seres_haskell99.pdf`
7. Type Logic Variables, K Classen, `\*http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.37.2565&rep=rep1&type=pdf`

8. Takashi's Workplace, [68], This is an unofficial implementation at embedding Prolog in Haskell, the reasoned behind it being that the only existing implementation was for Hugs 98 and is very complicated. The selling point of this implementation is simplicity. The implementation features no Monads or any other things from [9]. What it basically does is provide an REPL to add facts to the knowledge base, they are entered as strings and stored in some form of internal data structures. A query is requested which will do a depth first search, recursively finding substitutions for unifying the goal and the clauses from the knowledge base.

   The Prolog implemented is not full though, it is "Pure Prolog", no cuts, no fail, and other stuff. Moreover the REPL cannot do all the stuff that a swi prolog can do, for example you cannot declare variables / assignment statements and so on. Also you cannot right a "program file" as such, the REPL is all one gets to do stuff like adding clauses or querying etc.

   So you cannot write a program, you cannot do much with the REPL, its not a full blown Prolog.

6.3 Logic Libraries in Haskell

1. logict
2. logic-classes
3. proplogic
4. cflp
5. logic grows on trees

6.4 Unification Libraries in Haskell

1. unification-fd
2. cmu

6.5 Concatenative Programming Libraries in Haskell

1. peg

6.6 Constraint Programming and Constraint Handling Rules

1. monadiccp
2. monadicccp-gecode
3. csp
4. liquid fix point

6.7 Functional Logic Programming Language

1. The intergration of functions into Logic Programming : From Theory to Practice, http://www.informatik.uni-kiel.de/~mh/publications/papers/JLP94.html
2. Functional Logic Programming : From theory to curry, http://www.informatik.uni-kiel.de/~mh/papers/GanzingerFestschrift.pdf
3. Functional Logic Programming, http://dl.acm.org/citation.cfm?doid=1721654.1721675
4. A Higher Order Rewriting Logic for FLP, http://books.google.ca/books?hl=en&lr=&id=TSJDeaVpJyMC&oi=fnd&pg=PA153&dq=functional+logic+programming&ots=Ikp3Y-kZRV&sig=j7XQq-Hi-utdeNG54ZFkE1BeBNw#v=onepage&q=functional%20logic%20programming&f=false
5. Toy a multiparadigm declarative system
6. A unified computation model for functional and logic programming
7. Semantics and Types in Functional Logic Programming
8. Polymorphic Types in FLP
9. A general Computation Scheme for Constraint Logic Programming

1. Lambda Prolog
2. Mercury
3. Curry
4. Escher

7 Embedding a Programming Language into another Programming Language

Embedding a language into another language,

As discussed in the

7.1 Theory

1. Papers
   (a) Embedding an interpreted language using higher-order functions, [43]
   (b) Building domain-specific embedded languages, [25]
   (c) Embedded interpreters, [6]
   (d) Cayenne – a Language With Dependent Types, [2]
   (e) Foreign interface for PLT Scheme, [5]
   (f) Dot-Scheme: A PLT Scheme FFI for the .NET framework, [38]
   (g) Application-specific foreign-interface generation, [44]
   (h) Embedding S in other languages and environments, [34]
2. Books
   (a) ?????????
3. Articles / Blogs / Discussions
   (a) Embedding one language into another, [32]
   (b) Application-specific foreign-interface generation, [33]
   (c) Linguistic Abstraction, [36]
   (d) LISP, Unification and Embedded Languages, [37]
4. Websites
   (a) Embedding SWI-Prolog in other applications, [15]

7.2 Implementations

1. Lots of them I guess

7.3 Important People

1. ????

7.4 Miscellaneous / Possibly Related Content

1. ????

8 Prolog in ____

Prolog in _____

8.1 Theory

- Papers
    1. QLog, [29]
    2. LogLisp Motivation, design, and implementation, [46]
- Books
    1. Warrens Abstract Machine A TUTORIAL RECONSTRUCTION, [1]
    2. LOGLISP: an alternative to PROLOG, [47]
- Articles / Blogs / Discussions
    1. Hello
- Websites
    1. Hello

8.2 Implementations

1. Castor : Logic paradigm for C++, [35]
2. GNU Prolog for Java, [22]
3. JLog - Prolog in Java, [26]
4. JScriptLog - Prolog in Java, [27]
5. Quintus Prolog, [39]
6. Yield Prolog, [40]
7. Racklog, [55]

8.3 Important People

1. ???

8.4 Miscellaneous / Possibly Related Content

1. ???

9 Prolog in Haskell

Prolog in Haskell

9.1 Theory

- Papers

    1. Embedding Prolog in Haskell / Functional Reading of Logic Programs, [57]
    2. Algebra of Logic Programming, [53]
    3. The Algebra of Logic Programming, [51]
    4. Optimisation Problems in Logic Programming : An Algebraic Approach, [52]
    5. Higher Order Transformation of Logic Programs, [54]
    6. The Algebra of Searching, [56]
    7. FUNCTIONAL PEARL Combinators for breadth-first search, [58]
    8. Type Logic Variables, K Classen, [9]
    9. A Type-Safe Embedding of Constraint Handling Rules into Haskell Wei-Ngan Chin, Martin Sulzmann and Meng Wang, [8]
    10. Prological Features in a Functional Setting Axioms and Implementation, R Hinze, [23]
    11. Escape from Zurg: An Exercise in Logic Programming, [17]

- Books

    1. The Reasoned Schemer, Daniel P. Friedman, William E. Byrd, Oleg Kiselyov, [13]
    2. Programming Languages: Application and Interpretation, Shriram Krishnamurthi, Chapters 33-34 of PLAI discuss Prolog and implementing Prolog, [30]

- Articles / Blogs / Discussions

    1. Lambda the Ultimate, Programming Languages, [31]
    2. Takashi's Workplace (Implementation), [68]
    3. Haskell vs. Prolog Comparison, [59]

- Websites

    1. Logic Programming in Haskell, [64]

9.2 Implementations

1. A Prolog in Haskell, Takashi's Workplace, [68]
2. Mini Prolog for Hugs 98, [28]
3. Nano Prolog, [60]
4. Prolog, [49]
5. cspm-To-Prolog, [19]
6. prolog-graph, [4]
7. prolog-graph-lib, [48]
8. hswip, [61]

9.3 Important People

1. Mike Spivey
2. Silvija Seres

9.4 Miscellaneous / Possibly Related Content

1. Unification Libraries
   (a) unification-fd, [62]
   (b) cmu, [42]
2. Logic Libraries
   (a) logicct, [11], [12]
   (b) logic-classes, [**?**]
   (c) proplogic, [20]
   (d) cflp, [18]
   (e) logic-grows-on-trees, [10]
3. Concatenative Programming
   (a) peg, [14]

4. Constraint Programming and Constraint Handling Rules

   (a) monadiccp, [45]

   (b) monadicccp-gecode, [63]

   (c) csp, [3]

   (d) liquid fix point, [50]

10 Unifying or Marrying or Merging or Combining Programming Paradigms or Theories

Unifying / Marrying / Merging / Combining Programming Paradigms / Theories

10.1 Theory

- Papers
    1. Unifying Theories of Programming with Monads, [21]
    2. Symposium on Unifying Theories of Programming, 2006, [16].
    3. Symposium on Unifying Theories of Programming, 2008, [7].
    4. Symposium on Unifying Theories of Programming, 2010, [41].
    5. Symposium on Unifying Theories of Programming, 2012, [67].
- Books
    1. Unifying Theories of Programming, [24]
- Articles / Blogs / Discussions
    1. ???
- Websites
    1. ???

10.2 Implementations

1. Scala
2. Virgil
3. CLOS, Common Lisp Object System
4. Visual Prolog
5. ????

10.3 Miscellaneous / Possibly Related Content

1. ???

11 Functional Logic Programming Languages

Functional Logic Programming Languages

11.1 Theory

- Paper
    1. FLPL Introdunction Theory
        (a) Hello
    2. FLPL Surveys
        (a) Hello
    3. Narrowing in FLPL
        (a) Hello
    4. Residuation in FLPL
        (a) Hello
    5. Computation Model for FLPL
        (a) Hello
- Books
    1. Hello
- Articles / Blogs / Discussions
    1. Hello
- Websites
    1. Hello

11.2 Implementations

1. Hello

11.3 Miscellaneous / Possibly Related Content

1. Hello

12 Quasiquotation

12.1 Theory

1. Papers
    (a)
2. Books
    (a)
3. Articles / Blogs / Discussions
    (a)
4. Websites
    (a) Quasiquotation Wikipedia, [66]
    (b) Quasiquotation in Haskell, [65]

12.2 Implementations

1.

12.3 Miscellaneous / Possibly Related Content

1.

13 Related Terms or Keywords

Related Terms / Keywords

1. Prolog in Other Languages
2. Prolog in Haskell
3. Embedding One language into another language
4. Constraint Programming
5. Constraint Handling Rules
6. Concatenative Programming
7. Functional Logic Programming Languages
8. Residuation
9. Narrowing
10. Warren Abstraction Machine
11. Foreign Function Interfaces
12. Quasiquotation
13. Programming Theory Unification

14 Haskell or Why Haskell ?

Haskell / Why Haskell ?

1. Hello

15 Prolog or Why Prolog ?

Prolog / Why Prolog ?

1. Hello

16 Miscellaneous or Possibly Related Content

Miscellaneous / Possibly Related Content

1. ???

17 Conclusion

Bibliography

[1] Hassan Aït-Kaci and Forêt Des Flambertins. Warrens abstract machine a tutorial reconstruction. 1999.

[2] Lennart Augustsson. Cayenne – a language with dependent types. In *IN INTERNATIONAL CONFERENCE ON FUNCTIONAL PROGRAMMING*, pages 239–250. ACM Press, 1998.

[3] Andrei Barbu. The csp package, August 2013. `http://hackage.haskell.org/package/csp`.

[4] Matthias Bartsch. The prolog-graph package, September 2011. `http://hackage.haskell.org/package/prolog-graph`.

[5] Eli Barzilay and Dmitry Orlovsky. Foreign interface for plt scheme. *on Scheme and Functional Programming*, page 63, 2004.

[6] Nick Benton. Embedded interpreters. *Journal of Functional Programming*, 15(4):503–542, 2005.

[7] Andrew Butterfield, editor. *Unifying Theories of Programming, Second International Symposium, UTP 2008, Dublin, Ireland, September 8-10, 2008, Revised Selected Papers*, volume 5713 of *Lecture Notes in Computer Science*. Springer, 2010.

[8] Wei-Ngan Chin, Martin Sulzmann, and Meng Wang. A type-safe embedding of constraint handling rules into haskell. *Technical reportSchool of Computing, National University of Singapore, Boston, MA, USA*, 2003.

[9] Koen Claessen and Peter Ljunglöf. Typed logical variables in haskell. *Electr. Notes Theor. Comput. Sci.*, 41(1):37, 2000.

[10] Gregory Crosswhite. The logicgrowsontrees package, September 2013. `http://hackage.haskell.org/package/LogicGrowsOnTrees`.

[11] DanDoel. The logict package, August 2013. `http://hackage.haskell.org/package/logict`.

[12] DanDoel. The logict package example, August 2013. `http://okmij.org/ftp/Computation/monads.html`.

[13] Oleg Kiselyov Daniel P. Friedman, William E. Byrd. *The Reasoned Schemer*. The MIT Press, Cambridge Massachusetts, London England, 2005.

[14] Dustin DeWeese. The peg package, April 2012. `http://hackage.haskell.org/package/peg`.

[15] SWI Prolog Documentation. Embedding swi-prolog in other applications, June 2013. `http://www.swi-prolog.org/pldoc/man?section=embedded`.

[16] Steve Dunne and Bill Stoddart, editors. *Unifying Theories of Programming, First International Symposium, UTP 2006, Walworth Castle, County Durham, UK, February 5-7, 2006, Revised Selected Papers*, volume 4010 of *Lecture Notes in Computer Science*. Springer, 2006.

[17] Martin Erwig. Escape from zurg: an exercise in logic programming. *Journal of Functional Programming*, 14(03):253–261, 2004.

[18] Sebastian Fischer. The cflp package, June 2009. `http://hackage.haskell.org/package/cflp`.

[19] Marc Fontaine. The cspm-toprolog package, August 2013. `http://hackage.haskell.org/package/CSPM-ToProlog`.

[20] David Fox. The proplogic package, April 2012. http://hackage.haskell.org/package/PropLogic.

[21] Jeremy Gibbons. Unifying theories of programming with monads. In *Unifying Theories of Programming*, pages 23–67. Springer, 2013.

[22] GNU. Gnu prolog for java, August 2010. http://www.gnu.org/software/gnuprologjava/.

[23] Ralf Hinze et al. Prological features in a functional setting axioms and implementation. In *Fuji International Symposium on Functional and Logic Programming*, pages 98–122. Citeseer, 1998.

[24] Charles Anthony Richard Hoare and Jifeng He. *Unifying theories of programming*, volume 14. Prentice Hall Englewood Cliffs, 1998.

[25] Paul Hudak. Building domain-specific embedded languages. *ACM Comput. Surv.*, 28(4es):196, 1996.

[26] JLogic. Jlog - prolog in java, September 2012. http://jlogic.sourceforge.net/index.html.

[27] JLogic. Jscriptlog - prolog in javascript, September 2012. http://jlogic.sourceforge.net/index.html.

[28] Mark P Jones. Mini-prolog for hugs 98, June 1996. http://darcs.haskell.org/hugs98/demos/prolog/.

[29] H Jan Komorowski. Qlog: The programming environment for prolog in lisp. *Logic Programming*, pages 315–324, 1982.

[30] Shriram Krishnamurthi. *Programming languages: Application and interpretation*, chapter 33-34, pages 295–305, 307–311. Brown Univ., 2007.

[31] The Programming Languages Weblog Lambda The Ultimate. Embedding prolog in haskell, July 2004. http://lambda-the-ultimate.org/node/112.

[32] The Programming Languages Weblog Lambda The Ultimate. Embedding one language into another, March 2005. http://lambda-the-ultimate.org/node/578.

[33] The Programming Languages Weblog Lambda The Ultimate. Application-specific foreign-interface generation, October 2006. http://lambda-the-ultimate.org/node/2304.

[34] Duncan Temple Lang. Embedding s in other languages and environments. In *Proceedings of DSC*, volume 2, page 1, 2001.

[35] MPprogramming.com. Castor : Logic paradigm for c++, August 2010. http://www.mpprogramming.com/cpp/.

[36] Kurt Nrmark Department of Computer Science Aalborg University Denmark. Linguistic abstraction, July 2013. http://people.cs.aau.dk/~normark/prog3-03/html/notes/languages_themes-intro-sec.html#languages_intro-sec_section-title_1.

[37] University of Maryland Medical Center. Lisp, unification and embedded languages, October 2012. http://www.cs.unm.edu/~luger/ai-final2/LISP/.

[38] Pedro Pinto. Dot-scheme: A plt scheme ffi for the .net framework. In *Workshop on Scheme and Functional Programming*. Citeseer, 2003.

[39] Quintus Prolog. Embeddability, December 2003. http://quintus.sics.se/isl/quintuswww/site/embed.html.

[40] Yield Prolog. Yield prolog, October 2011. http://yieldprolog.sourceforge.net/.

[41] Shengchao Qin, editor. *Unifying Theories of Programming - Third International Symposium, UTP 2010, Shanghai, China, November 15-16, 2010. Proceedings*, volume 6445 of *Lecture Notes in Computer Science*. Springer, 2010.

[42] John Ramsdell. The cmu package, February 2013. http://hackage.haskell.org/package/cmu.

[43] Norman Ramsey. Embedding an interpreted language using higher-order functions and types. In *Proceedings of the 2003 workshop on Interpreters, virtual machines and emulators*, pages 6–14. ACM, 2003.

[44] John Reppy and Chunyan Song. Application-specific foreign-interface generation. In *Proceedings of the 5th international conference on Generative programming and component engineering*, pages 49–58. ACM, 2006.

[45] Maik Riechert. The monadiccp package, July 2013. http://hackage.haskell.org/package/monadiccp.

[46] J Alan Robinson and Ernest E Sibert. Loglisp: Motivation, design, and implementation, 1982.

[47] John Alan Robinson and EE Silbert. *LOGLISP: an alternative to PROLOG*. School of Computer and Information SCience, Syracuse University, 1980.

[48] Daniel Seidel. The prolog-graph-lib package, June 2012. http://hackage.haskell.org/package/prolog-graph-lib.

[49] Daniel Seidel. The prolog package, June 2012. http://hackage.haskell.org/package/prolog.

[50] Eric Seidel. The liquid-fixpoint package, September 2013. http://hackage.haskell.org/package/liquid-fixpoint.

[51] Silvija Seres. *The algebra of logic programming*. PhD thesis, 2001.

[52] Silvija Seres and Shin-Cheng Mu. Optimisation problems in logic programming: an algebraic approach. 2000.

[53] Silvija Seres, J Michael Spivey, and CAR Hoare. Algebra of logic programming. In *ICLP*, pages 184–199, 1999.

[54] Silvija Seres and Michael Spivey. Higher-order transformation of logic programs. In *Logic Based Program Synthesis and Transformation*, pages 57–68. Springer, 2001.

[55] Dorai Sitaram. Racklog: Prolog-style logic programming, January 2014. http://docs.racket-lang.org/racklog/index.html.

[56] JM Spivey and Silvija Seres. The algebra of searching. *Festschritf in hounour of CAR Hoare*, 1999.

[57] JM Spivey and Silvija Seres. Embedding prolog in haskell. In *Proceedings of Haskell*, volume 99, pages 1999–28. Citeseer, 1999.

[58] Michael Spivey. Functional pearls combinators for breadth-first search. *Journal of Functional Programming*, 10(4):397–408, 2000.

[59] Stackoverflow. Haskell vs. prolog comparison, December 2009. http://stackoverflow.com/questions/1932770/haskell-vs-prolog-comparison.

[60] Jurrien Stutterheim. The nanoprolog package, December 2011. http://hackage.haskell.org/package/NanoProlog.

[61] Evgeny Tarasov. The hswip package, August 2010. http://hackage.haskell.org/package/hswip.

[62] Wren Thornton. The unification-fd package, July 2012. http://hackage.haskellhttp://yieldprolog.sourceforge.net/.org/package/unification-fd.

[63] Jan Tikovsky. The monadiccp-gecode package, January 2014. http://hackage.haskell.org/package/monadiccp-gecode.

[64] Haskell Website. Logic programming example, February 2010. http://www.haskell.org/haskellwiki/Logic_programming_example.

[65] Haskell Website. Quasiquotation in haskell, January 2014. http://www.haskell.org/haskellwiki/Quasiquotation.

[66] Wikipedia. Quasiquotation, Novemeber 2013. http://en.wikipedia.org/wiki/Quasi-quotation.

[67] Burkhart Wolff, Marie-Claude Gaudel, and Abderrahmane Feliachi, editors. *Unifying Theories of Programming, 4th International Symposium, UTP 2012, Paris, France, August 27-28, 2012, Revised Selected Papers*, volume 7681 of *Lecture Notes in Computer Science*. Springer, 2013.

[68] Takashi's Workplace. A prolog in haskell, April 2009. http://propella.blogspot.in/2009/04/prolog-in-haskell.html.