


12th Nordic Workshop on Programming Theory  
<http://www.ii.uib.no/~nwpt00>

## Modular Development of Interpreters from Semantic Building Blocks


Jose E. Labra G., Juan M. Cueva, M<sup>a</sup> Candida Luengo D.

Department of Computer Science  
 University of Oviedo (Spain)




Modular Development of Interpreters from Semantic Building Blocks  
 NWPT'00 (<http://www.ii.uib.no/~nwpt00>)

1




## Overview of the Talk




- Motivation
- Language Prototyping System
  - Theoretical background
  - Architecture of the System
  - Example: MLambda
- Conclusions and Future Work

Modular Development of Interpreters from Semantic Building Blocks  
 NWPT'00 (<http://www.ii.uib.no/~nwpt00>)

2



## Motivation




```


  graph TD
    A[Oviedo3 Project] --> B[Integral OO Operating System]
    A --> C[Semantic Specification of several Abstract Machines]
    C --> D[Modular Interpreters]
    D --> E[Tool for Modular Development of Interpreters]
  
```

Modular Development of Interpreters from Semantic Building Blocks  
 NWPT'00 (<http://www.ii.uib.no/~nwpt00>)

3



## Theoretical Background




- Denotational Semantics (Strachey, Scott 67)
  - Semantic functions = Inductive Definitions over the syntax  $\Sigma$
  - Domains = characterise the range of values  $V$ 


$$\Sigma \rightarrow V$$
  - Problems: *Modularity? Extensibility?*
- Monadic Semantics (E. Moggi, 89)
  - Separation between values and computations
  - A Monad  $M$  models a computation
  - Different monads  $\Rightarrow$  Different computations
    - State, environment, continuations, non-determinism,...
$$\Sigma \rightarrow M V$$
  - Problem: *Composing Monads?*

Modular Development of Interpreters from Semantic Building Blocks  
 NWPT'00 (<http://www.ii.uib.no/~nwpt00>)

4




## Theoretical Background




- Modular Interpreters using Monads (Wadler, 92, Espinosa, 93, Steele Jr, 94)
- Modular Monadic Semantics (S. Liang, P. Hudak, M. Jones, 95)
  - **Monad Transformers** add new features to a given monad
  - Several monad transformers:
    - Exceptions, State transformer, Environment reader, Continuations, Non-determinism, Input-Output, Resumptions
    - $Computation = (MT_1 \cdot MT_2 \cdot \dots \cdot MT_n) BasicMonad$
  - Subtyping Mechanism
    - Allows domain value extensibility in Haskell
    - Requires non-standard extensions to Haskell type system
    - $V = V_1 + V_2 + \dots + V_n$

Modular Development of Interpreters from Semantic Building Blocks  
NWPT'00 (<http://www.iitb.ac.in/~mwp00/>)

5




## Theoretical Background




- Initial Algebra and Folds (Meijer, Fokkinga, Paterson 91)
  - Recursive datatype = least fixpoint of a *pattern* functor
  - Polynomial Functors
- Applying folds to MMS (L. Duponcheel, 95)
  - $\Sigma = \mu F$  where  $F = F_1 + F_2 + \dots + F_n$
  - Modular Syntax Specification
- Merging Monads with Folds (Meijer, Jeuring, 95)
- Applying Monadic Folds to MMS (Labra, 99)
  - Separation between recursive evaluation and semantic specification

Modular Development of Interpreters from Semantic Building Blocks  
NWPT'00 (<http://www.iitb.ac.in/~mwp00/>)

6




## Language Prototyping System




- Embedded *Domain Specific Language* for Interpreter Definition
- Host Language = Haskell
- Interactive Interpreter Framework
  - Allows run time interpreted language selection
- 3 examples implemented (*by now*)
  - *MLambda*: Functional language with different call mechanisms and imperative features
  - *While*: Simple Imperative programming language
  - *IL*: Intermediate Language

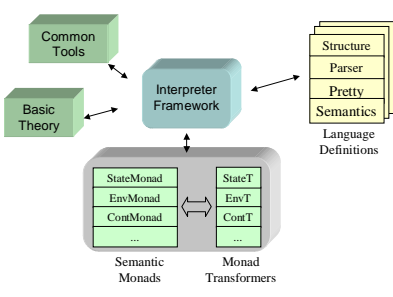
Modular Development of Interpreters from Semantic Building Blocks  
NWPT'00 (<http://www.iitb.ac.in/~mwp00/>)

7



## LPS Architecture





Modular Development of Interpreters from Semantic Building Blocks  
NWPT'00 (<http://www.iitb.ac.in/~mwp00/>)

8

## LPS Architecture: Semantic Monads

<b>Monad</b>	A type constructor $M$ $return: \forall a. a \rightarrow M a$ $bind: \forall a b. M a \rightarrow (a \rightarrow M b) \rightarrow M b$
<b>ExcMonad</b>	$raise: \forall a. \epsilon \rightarrow M a$ $handle: \forall a. M a \rightarrow (\epsilon \rightarrow M a) \rightarrow M a$
<b>EnvMonad</b>	$rdEnv: M \rho$ $inEnv: \forall a. \rho \rightarrow M a \rightarrow M a$
<b>StateMonad</b>	$fetch: M \sigma$ $set: \sigma \rightarrow M ()$
<b>ContMonad</b>	$callcc: \forall a. b. (a \rightarrow M b) \rightarrow M a \rightarrow M a$
<b>IOMonad</b>	$get: M Char$ $put: Char \rightarrow M ()$
<b>DebugMonad</b>	$infoDebug: String \rightarrow M ()$

do-notation  
 $m \text{ 'bind' } (x \rightarrow f)$   
 $=$   
 $do \{ x \leftarrow m; f \}$

... more monads with their operations...

Modular Development of Interpreters from Semantic Building Blocks  
NWPT'00 (<http://www.iitb.ac.in/~mwp00/>)

## LPS Architecture: Monad Transformers

- A monad transformer transforms a monad  $M$  into a new monad  $M'$  with new operations
  - $mt :: M \rightarrow M'$
- Conditions**
  - $M'$  must really be a monad
  - The old operations from  $M$  should behave in the same way in  $M'$
  - This *lifting* of operations is made manually for some computational features

<b>ExcT <math>\epsilon</math></b>	$ExcT \epsilon m v = m (\epsilon + v)$
<b>StateT <math>\sigma</math></b>	$StateT \sigma m v = \sigma \rightarrow m (v, \sigma)$
<b>EnvT <math>\rho</math></b>	$EnvT \rho m v = \rho \rightarrow m v$
<b>ContT <math>\alpha</math></b>	$ContT \alpha m v = (v \rightarrow m \alpha) \rightarrow m \alpha$
<b>IOT</b>	$IOT m v = StateT IOChannels m v$
<b>DebugT</b>	$DebugT m v = StateT DebugInfo m v$

... more monad transformers

Modular Development of Interpreters from Semantic Building Blocks  
NWPT'00 (<http://www.iitb.ac.in/~mwp00/>)

## LPS Architecture: Interactive Framework

- Different languages (with different types) in the same data structure
- Independent development of each language under a common interface
- Solution:** Existential types + type classes (Lauer, 94)

```

class (Show s) => Syntax s where
  exec :: s -> Computation

data Language = forall s. Syntax . MkL s

instance Syntax Language where
  exec (MkL s) = exec s

instance Syntax L1 where
  exec s = ...

ls :: [Language]
ls = [MkL lang1, MkL lang2, ...]

test :: [Language] -> Int -> Computation
test ls n = exec (ls !! n)
    
```

$MkL :: (\exists s \in Syntax . s) \rightarrow Language$

Any type  $\in$  Syntax can be included in the list

Modular Development of Interpreters from Semantic Building Blocks  
NWPT'00 (<http://www.iitb.ac.in/~mwp00/>)


## Example: MLambda

- Functional Programming Language
  - lambda abstraction (call by value)
- 2 primitive types: Booleans and Integers (no type checking)
- Imperative features
  - input/output, exceptions, first-class continuations, references and assignment
- Simple Debugger


```

new 0;
0 := read v;
let fac =
  \x -> if (x < 0) then raise "negative arg"
        else if (x==0) then 1
            else x * fac (x - 1)
in fac (get 0)
    
```

Modular Development of Interpreters from Semantic Building Blocks  
NWPT'00 (<http://www.iitb.ac.in/~mwp00/>)



## MLambda: Structure of the Language



- Extensible Abstract Syntax using Least Fixpoint of pattern functors
- Parser: *Parsec Library* (Daan Leijen, 2000)

**Common Definitions (Meijer 92)**

Sum of functors:  $(F \oplus G) x = F x + G x$

Fixpoint of a functor  $\mu F = f(\mu F)$

Initial F-algebra  $\text{In} :: F(\mu F) \rightarrow \mu F$

Fold or catamorphism  $(\text{fold } \phi) (\text{In } x) = \phi(\text{map } (\text{fold } \phi) x)$

F-algebras over Sum of Functors  
 $\phi_{F \oplus G} x = \phi_F x + \phi_G x$

**Pattern Functors**

$N x = \text{Num Int} \mid x \text{'Add' } x \mid \dots$

$V x = \text{Var String}$

$C x = \text{B Bool} \mid x \text{'Less' } x \mid \dots$

$D x = \text{Let String } x \ x$

$F x = \text{Lambda String } x \mid \text{Apply } x \ x$

$R x = \text{Get } x \mid \text{New } x \mid \text{Assign } x \ x \mid \text{Seq } x \ x$


$E x = \text{Raise String}$

**Abstract Syntax**


$ML = N \oplus V \oplus C \oplus D \oplus F \oplus R \oplus E$

$MLambda = \mu ML$

Modular Development of Interpreters from Semantic Building Blocks  
 NWPT'00 (<http://www.i.uib.no/~mwp00/>)



## MLambda: Semantics



- Computational Monad  
 $\text{Computation} = \text{ExcT Exc}(\text{StateT Heap}(\text{EnvT Table}(\text{ContT Answer}(\text{DebugT IO})))$
- Domain Value  
 $\text{Value} = \text{Int} + \text{Bool} + (\text{Computation Value} \rightarrow \text{Computation Value})$
- For each pattern functor F, we define a F-algebra  $\phi_F$  with carrier Computation Value

```


 $\phi_N(\text{Num } n) = \text{return } (\text{inj } n)$ 
 $\phi_N(m_1 \text{'Add' } m_2) = \text{do } \{ v_1 \leftarrow m_1$ 
    ;  $v_2 \leftarrow m_2$ 
    ;  $\text{return } (\text{inj } (v_1 + v_2))$ 
    }
    
```

With monadic folds we could eliminate the recursive evaluation  
 $\phi_N(v_1 \text{'Add' } v_2) = \text{return } (\text{inj } (v_1 + v_2))$


```

    ...
 $\phi_V(\text{Var } x) = \text{do } \{ \text{table} \leftarrow \text{rdEnv}; \text{return } (\text{lookup table } x) \}$ 
 $\phi_{CB}(\text{B } b) = \text{return } (\text{inj } b)$ 
 $\phi_{CL}(\text{Less } m_2) = \text{do } \{ v_1 \leftarrow m_1$ 
    ;  $v_2 \leftarrow m_2$ 
    ;  $\text{return } (\text{inj } (v_1 < v_2))$ 
    }
    
```

Modular Development of Interpreters from Semantic Building Blocks  
 NWPT'00 (<http://www.i.uib.no/~mwp00/>)



## MLambda: Semantics (II)




```

 $\phi_D(\text{Let } x \ m_1 \ m_2) = \text{do } \{ \text{table} \leftarrow \text{rdEnv}; \text{inEnv } (\text{update table } x \ m_1) \ m_2 \}$ 
 $\phi_R(\text{Get } m) = \text{do } \{ \text{loc} \leftarrow m; \text{lookupTable loc} \}$ 
 $\phi_A(\text{New } m) = \text{do } \{ v \leftarrow m; \text{loc} \leftarrow \text{alloc } (\text{return } v); \text{return } (\text{inj loc}) \}$ 
 $\phi_{AS}(\text{Assign } m_1 \ m_2) = \text{do } \{ \text{loc} \leftarrow m_1; \text{val} \leftarrow m_2; \text{updateLoc loc } (\text{return val}) \}$ 
 $\phi_S(\text{Seq } m_1 \ m_2) = \text{do } \{ m_1; m_2 \}$ 
 $\phi_F(\text{Lambda } x \ m) = \text{do } \{ \text{table} \leftarrow \text{rdEnv}$ 
    ;  $\text{return } (\text{inj } (\lambda c \rightarrow \text{do } \{ v \leftarrow c; \text{inEnv } (\text{update table } x \ (\text{return } v)) \ m \}))$ 
    }
 $\phi_{AP}(\text{Apply } m_1 \ m_2) = \text{do } \{ f \leftarrow m_1; \text{table} \leftarrow \text{rdEnv}$ 
    ;  $f (\text{inEnv table } m_2)$ 
    }
 $\phi_{ER}(\text{Raise msg}) = \text{raise msg}$ 
    
```


$\text{sem} :: \text{MLambda} \rightarrow \text{Computation Value}$

$\text{sem} = (\text{fold } \phi_{N \oplus V \oplus C \oplus D \oplus F \oplus R \oplus E})$

Modular Development of Interpreters from Semantic Building Blocks  
 NWPT'00 (<http://www.i.uib.no/~mwp00/>)



## Conclusions



- A very modular way to develop interpreters for programming languages
  - It is possible to extend the syntax, the semantics and the domain value without modifying existing components
- An interactive system for language prototyping and testing
- Another *Embedded Domain Specific Language* in Haskell
  - Advantages
    - Easier Development
    - Reusability of semantic components
    - Incorporates a fairly good Type System
    - Access to other language tools
  - Disadvantages
    - Awkward Error Messages (even for the developer)
    - Limitations of Haskell98 Type System

Modular Development of Interpreters from Semantic Building Blocks  
 NWPT'00 (<http://www.i.uib.no/~mwp00/>)



## Future Work and work in progress

- More theoretical work
  - Monad transformers (lifting), monadic folds, coalgebras
- Deriving Compilers from MMS (Liang, 98, Harrison, Kamin, 2000)
- First class polymorphism (monads as first class values)
  - Monad transformer as a function
  - Extensible records (TRES in Hugs, not in GHC)
  - Extensible variants (not implemented)
- Assess the development of a *domain specific language* for semantic specifications
- GUI for the Interactive Interpreter Framework
- Specification of practical languages (work in progress)
  - A Reflective Object Oriented Abstract Machine
  - An Internet programming Language



## Further Information

- Language Prototyping System  
<http://lsi.uniovi.es/~labra/LPS/LPS.html>
- Oviedo3 Project:  
<http://www.uniovi.es/~oviedo3>

The End