Model/Implementation Synchronisation in Domain-Specific Programming Languages

Meng Wang

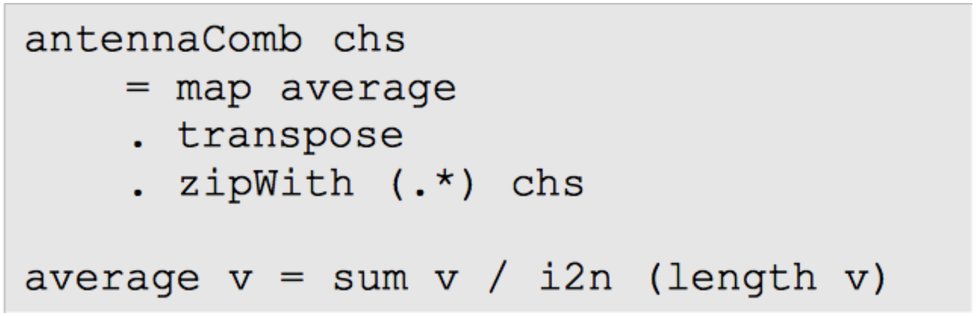
Chalmers University of Technology

Domain-Specific Languages (DSLs)

- Languages for anyone
- Designed with particular application in mind to capture the typical abstractions of the domain
- Often implemented by translating to some general-purpose languages
- Some common pitfalls
  - tool support
  - expressiveness

An Example

- Feldspar: A DSL with Haskell syntax for Digital Signal Processing
- One writes
  - modular
  - high-level
  - pure

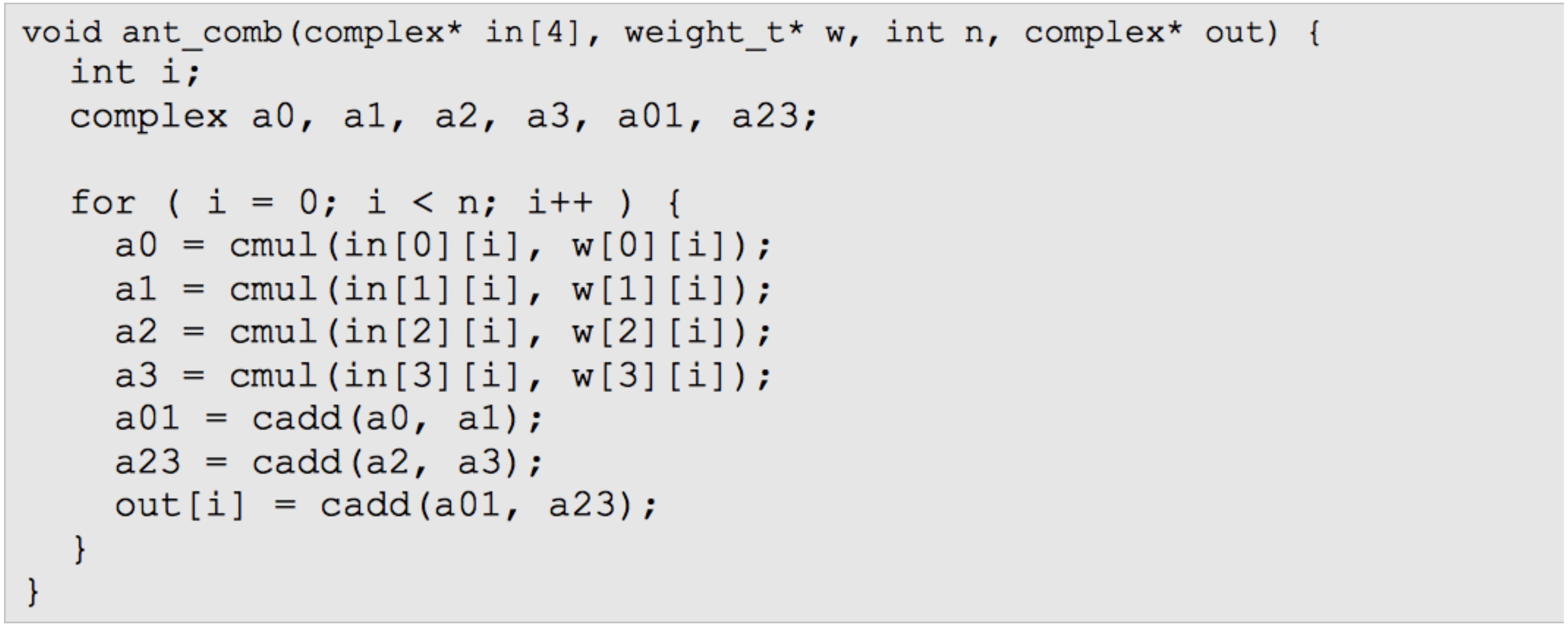


```
antennaComb chs
    = map average
    . transpose
    . zipWith (.*) chs

average v = sum v / i2n (length v)
```

- instead of …

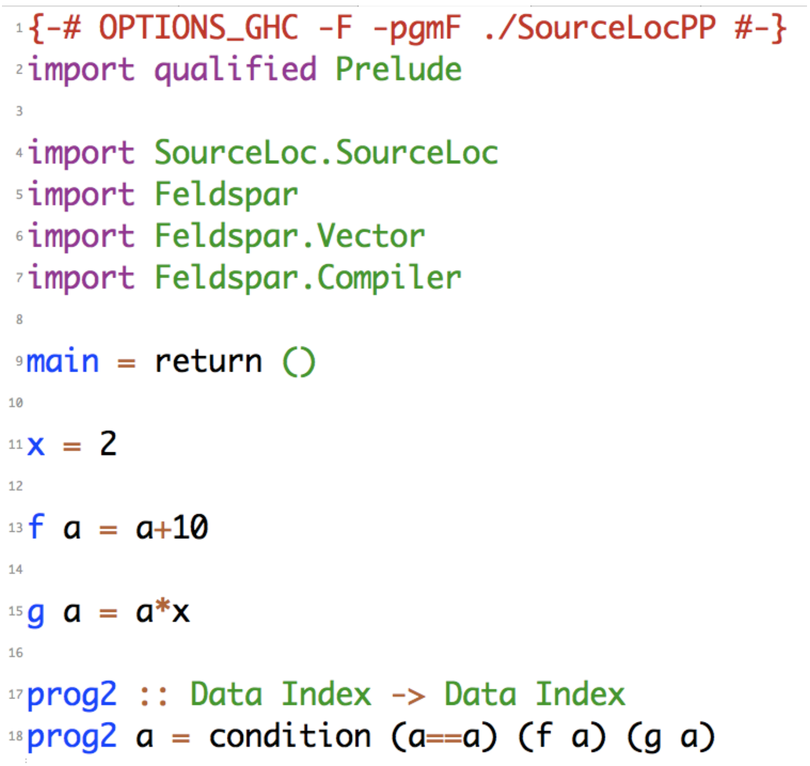
An Example (Cont.)

- instead of

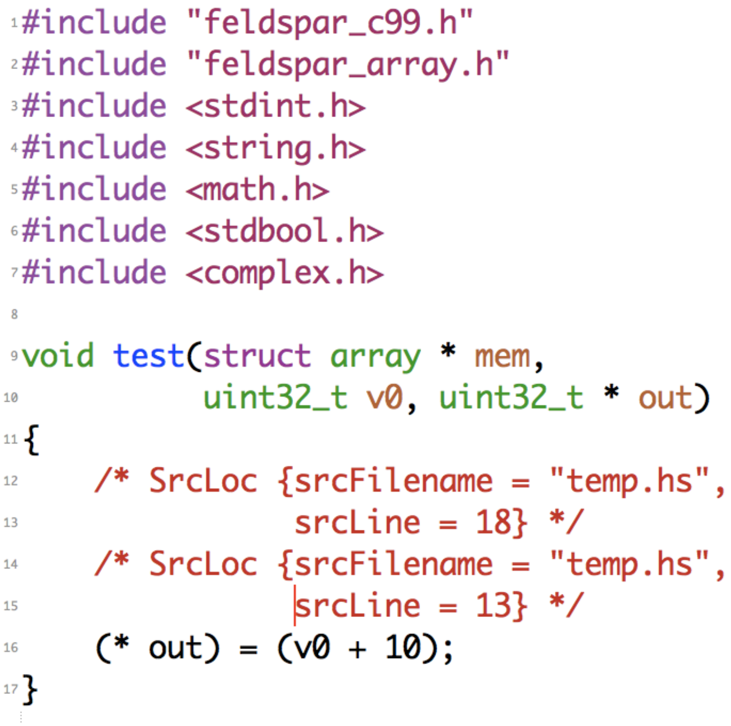


```
void ant_comb(complex* in[4], weight_t* w, int n, complex* out) {
  int i;
  complex a0, a1, a2, a3, a01, a23;

  for ( i = 0; i < n; i++ ) {
    a0 = cmul(in[0][i], w[0][i]);
    a1 = cmul(in[1][i], w[1][i]);
    a2 = cmul(in[2][i], w[2][i]);
    a3 = cmul(in[3][i], w[3][i]);
    a01 = cadd(a0, a1);
    a23 = cadd(a2, a3);
    out[i] = cadd(a01, a23);
  }
}
```

A Simpler Example



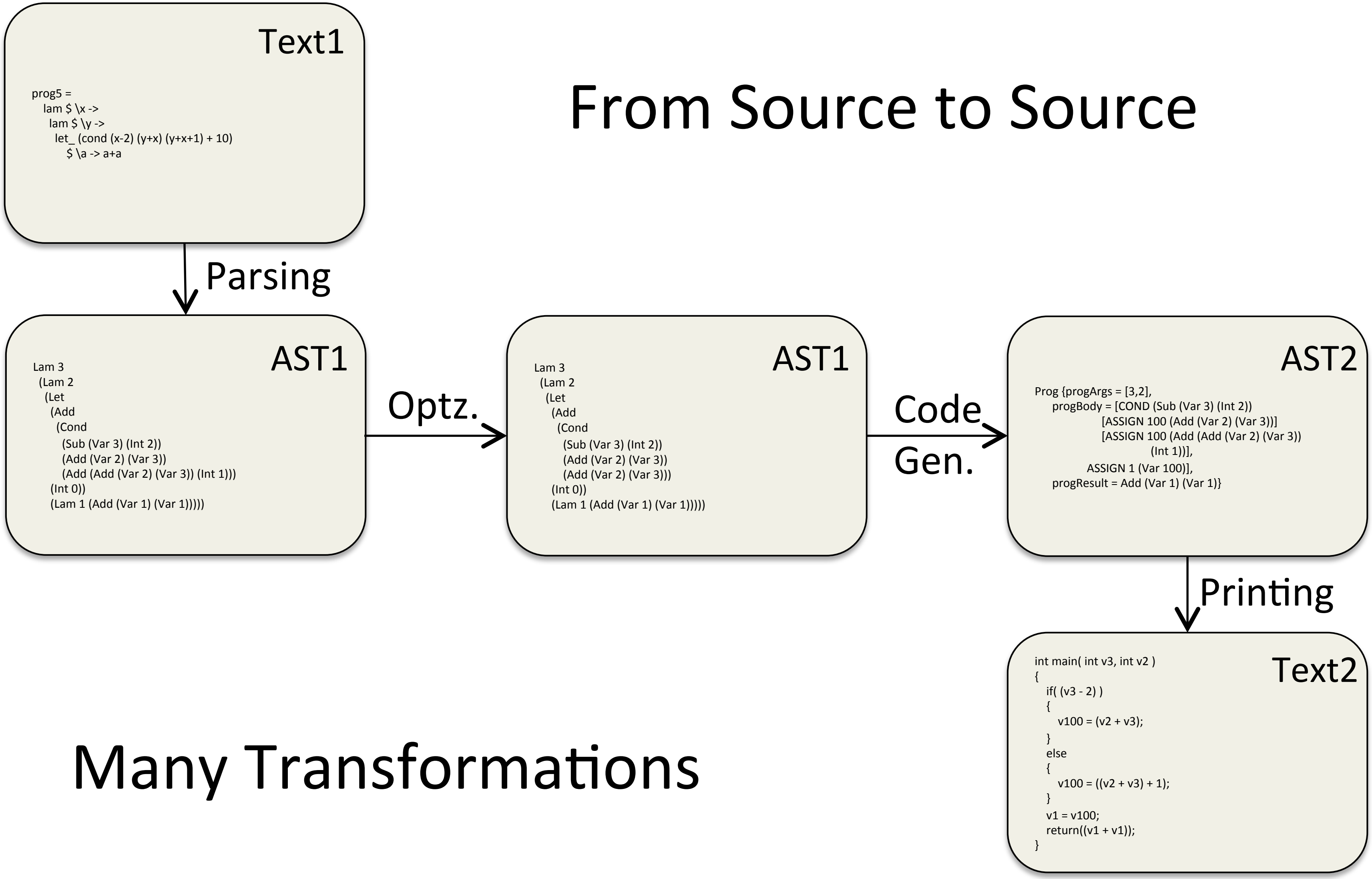
```haskell
{-# OPTIONS_GHC -F -pgmF ./SourceLocPP #-}
import qualified Prelude

import SourceLoc.SourceLoc
import Feldspar
import Feldspar.Vector
import Feldspar.Compiler

main = return ()

x = 2

f a = a+10

g a = a*x

prog2 :: Data Index -> Data Index
prog2 a = condition (a==a) (f a) (g a)
```



```c
#include "feldspar_c99.h"
#include "feldspar_array.h"
#include <stdint.h>
#include <string.h>
#include <math.h>
#include <stdbool.h>
#include <complex.h>

void test(struct array * mem,
          uint32_t v0, uint32_t * out)
{
    /* SrcLoc {srcFilename = "temp.hs",
               srcLine = 18} */
    /* SrcLoc {srcFilename = "temp.hs",
               srcLine = 13} */
    (* out) = (v0 + 10);
}
```

DSL Compilation

From Source to Source



Text1

```
prog5 =
   lam $ \x ->
     lam $ \y ->
       let_ (cond (x-2) (y+x) (y+x+1) + 10)
           $ \a -> a+a
```

AST1

```
Lam 3
  (Lam 2
    (Let
      (Add
        (Cond
          (Sub (Var 3) (Int 2))
          (Add (Var 2) (Var 3))
          (Add (Add (Var 2) (Var 3)) (Int 1)))
      (Int 0))
      (Lam 1 (Add (Var 1) (Var 1)))))
```

AST1

```
Lam 3
  (Lam 2
    (Let
      (Add
        (Cond
          (Sub (Var 3) (Int 2))
          (Add (Var 2) (Var 3))
          (Add (Var 2) (Var 3)))
      (Int 0))
      (Lam 1 (Add (Var 1) (Var 1)))))
```

AST2

```
Prog {progArgs = [3,2],
     progBody = [COND (Sub (Var 3) (Int 2))
                      [ASSIGN 100 (Add (Var 2) (Var 3))]
                      [ASSIGN 100 (Add (Add (Var 2) (Var 3))
                                (Int 1))],
                ASSIGN 1 (Var 100)],
     progResult = Add (Var 1) (Var 1)}
```

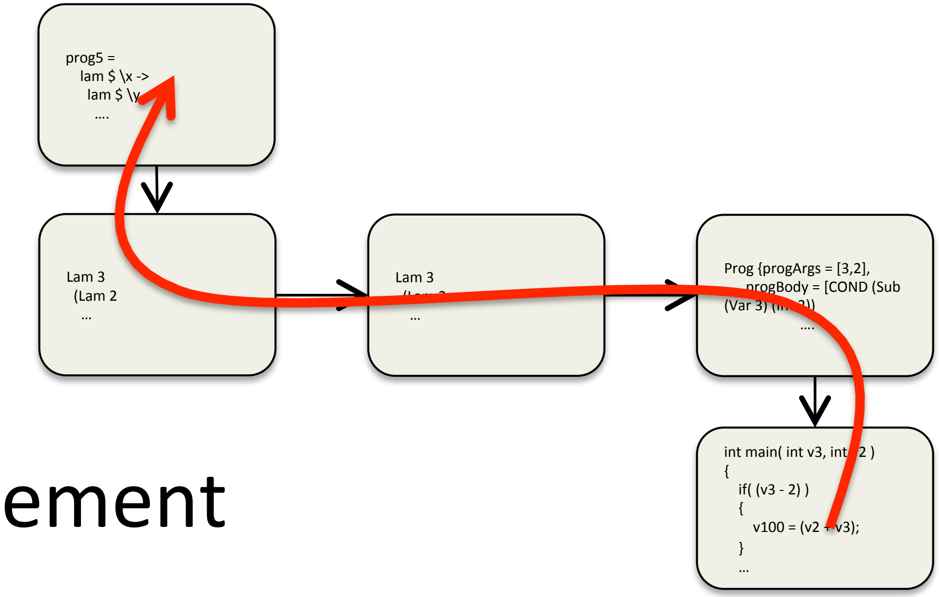
Text2

```
int main( int v3, int v2 )
{
   if( (v3 - 2) )
   {
      v100 = (v2 + v3);
   }
   else
   {
      v100 = ((v2 + v3) + 1);
   }
   v1 = v100;
   return((v1 + v1));
}
```

Many Transformations

Changes to Generated Code

- Inexpressiveness of the DSL
- Lack in tool support for the DSL
- Performance enhancement
- Other pragmatic reasons



```
prog5 =
   lam $ \x ->
     lam $ \y
      ....
```

```
Lam 3
  (Lam 2
    ...
```

```
Lam 3
  (Lam 2
    ...
```

```
Prog {progArgs = [3,2],
    progBody = [COND (Sub
(Var 3) (Int 2))
             ....
```

```
int main( int v3, int v2 )
{
   if( (v3 - 2) )
   {
      v100 = (v2 + v3);
   }
   ...
```

A Challenge

- Many kinds of (complicated) transformations
- Complex data (ASTs)
- Potentially very complicated updates
- A largely untapped area with great potentials
- Connection to SE? Models?

Example of Changes

- Change of Types
- Change of parameter order
- Coding style (spacing, indentation etc.)
- Moving statements
- Removing statements, variables etc.
- Various further optimizations
- Adding code
- …