Adv. Course in Programming Languages

Yukiyoshi Kameyama

Department of Computer Science, University of Tsukuba

Application of Program Generation – Domain-Specific Language

Papers for reports

Basic/General:

- **A Gentle Introduction to Malt-Stage Programming**, Taha, Dagstuhl Seminar, 2003.
- **GoMeta! A Case for Generative Programming and DSLs in Performance Critical Systems**, Rompf et al., SNAPL'15.

Application/Specific

- **Terra: A Multi-Stage Language for High-Performance Computing**, DeVito et al., PLDI'13.
- **Halide: A Language and Compiler for Optimizing Parallelism, Locality, and Recomputation in Image Processing Pipelines**, Ragan-Kelly et al., PLDI'13.
- **Functional Pearl: A SQL to C Compiler in 500 Lines of Code**, Rompf et al., ICFP'15.

Today

A Gentle Introduction to Multi-Stage Programming by Walid Taha.

- DSL (Domain-Specific Language)
- Interpreter for DSL
- Staging Interpreter
- Various Tricks (not in this lecture)
- Performance

DSL

General-Purpose Languages vs Domain-Specific Languages

GPL:

- You know many GPL (FORTRAN, C, C++, Java, Python, Perl, Ruby, OCaml, Lisp, etc.)
- You think GPL is universal (everything can be written.)
- You trust GPL. You don't have to design GPL.

DSL:

- You are a user of your DSL.
- You need to design, implement, and modify a DSL.
- You throw away a DSL, and design a new one from scratch.

What is DSL?

No rigid definition for DSL:

- DSL is purpose-built, purpose-oriented.
  - Parser: BNF, PEG, ...
  - Database: SQL, ...
  - Hardware: Verilog, VHDL, ...
  - Robotics: ...
  - GUI: ...
  - Game programming:
  - etc etc etc
- DSL is relatively small, restricted, simple.
- DSL need not be Turing complete.

How to design a DSL?

No general solution, but there are good recipe.

千葉滋著「2 週間でできる Script 言語の作りかた」, 技術評論社, 2012.

(Note. the author's 2 weeks is 2 years for ordinary people.)

How to implement DSL?

This is the paper's topic.

- Write a naive (slow) interpreter for the DSL.
- Convert it to a program generator (by staging).
- Then we get much better performance.

In more detail....

- An interpreter is a program which, given a DSL program and a few parameters, returns the value of the computation.
- Staging: The input DSL program is static, and the parameters are dynamic.
- Generated code: given the parameters, it returns the result.

DSL in this paper

Example of DSL programs:

```
let rec f x =
  if x = 0 then 1
  else x * (f (x-1))
in f 10
```

Power function, again??? No, it's factorial.

Embedding DSL into MetaOCaml:

```
DSL       Embedding
1 + 2     Add(Int 1, Int 2)
fact(x-1) App("fact",Sub(Var "x",Int 1))
```

Naive DSL Interpreter

```
let rec eval1 e env fenv =
match e with
| Int i    ->  i
| Add (e1,e2) ->  (eval1 e1 env fenv)
                + (eval1 e2 env fenv)
...
(* sample run *)
eval1 (Add((Int 2), (Int 3))) env0 fenv0 ==> 5
```

Very slow: if we run the factorial function by this eval1, it runs 15 to 20 times slower than the hand-written factorial function. (Interpretive overhead)

Staging the Naive DSL Interpreter

Inputs for the function eval:

- expression e is static
- values of variables etc.: dynamic

Namely, we specialize 'eval' by the expression (the program).

Staged DSL Interpreter

```
let rec eval2 e env fenv =
match e with
| Int i    ->  <i>
| Add (e1,e2) -> < ~(eval1 e1 env fenv)
                + ~(eval1 e2 env fenv)>
...
(* sample run *)
eval2 (Add((Int 2), (Int 3))) env0 fenv0 ==>
    <2 + 3>
```

Quite similar to the staging process for the power function.

Staged Interpreter is a translator

Surprising result:

```
(* sample run *)
eval2 (factorial 10) env0 fenv0 ==>
<let rec f x = if x = 0 then 1 else x * (f (x
   -1))
 in f 10>
```

Just like a hand-written factorial function, so the optimization is perfect! (No interpretive overhead)

The rest of the paper

Life is not so easy

- The simple DSL interpreter can be staged very easily.
- But we will meet many obstacles to prevent staging: if we have the division-by-zero error etc.
- For each time, some 'tricks' will solve the problem.

Possible report on this paper

One page per the item:

- Summary of the paper (Examples: you should at least read the abstract, introduction, and conclusion. What is 'interpretive overhead'? What is staging?)
- Your evaluation/impression/thoughts about the paper (Examples: what/how do you feel about the paper.)
- Your topics related to this paper: (Examples: do you use DSL in your research? if yes, what kind of DSL? do you think DSL is useful or a bad idea? do you want to do meta-programming in your domain?)