Semantic Software Adaptation Using Verified Lifting

Alvin Cheung
Primary Language During Career

Grad student / Asst. Prof / Junior Researcher

Haskell / Python / C++

LaTeX

Powerpoint

Someone else’s powerpoint

Someone else’s ppt w/o reading beforehand

Credit (blame) to Donald Kossman, ETH Zurich / MSR
Adapting Software Today: Human in the loop

Original Implementation → Specification → New Implementation

Reverse Engineering  → Specification → Coding
Automatic Inference + Synthesis: Verified Lifting

domain-specific language for specifications

original implementation

inference

specification

new implementation

synthesis
Case Study 1: Leveraging Databases
Java → SQL

1. Define ordered lists and their operations

\[ \text{listUsers} = \pi_{\ell}(\bowtie_{\varphi}(\text{users}, \text{roles})) \]

where

\[ \varphi(e_{\text{users}}, e_{\text{roles}}) := e_{\text{users}}.\text{roleID} = e_{\text{roles}}.\text{roleID} \]

\( \ell \) contains all the fields from the \textit{User} class

2. Synthesizer infers spec from source

3. Retarget spec to databases
codegen

List getUsersWithRoles () {
    return executeQuery(
        "SELECT u FROM users u, roles r
        WHERE u.roleID == r.id
        ORDER BY u.roleID, r.id""); }

Lifted code can be optimized by DBs
100-1000x speedup

[PLDI 13, DE Bulletin 14]
Case Study 2: Distributed Processing Frameworks
Sequential Java \rightarrow Hadoop

1. Define ordered lists and their operations

2. Synthesizer infers spec from source

   // sequential implementation
   void regress(Point [] points)
   {
   int SumXY = 0;
   for(Point p : points){
       SumXY += p.x * p.y;
   }
   return SumXY;
   }

3. Retarget spec to Hadoop
codegen

   void map(Object key, Point [] value)
   {
   for(Point p : points)
   { emit("sumxy", p.x * p.y); }
   
   void reduce(Text key, int [] values)
   {
   int SumXY = 0;
   for (Integer val : values)
   { SumXY = SumXY + val;
   emit(key, SumXY); }
   }

Verified Lifting

Lifted code can be optimized by Hadoop

6x speedup
Case Study 3: Leveraging GPUs

Fortran → Halide

1. Define DSL based on theory of arrays

```
post(a,b) ≡ ∀imin+1 ≤ i ≤ imax, jmin ≤ j ≤ jmax.
    a(i,j) = b(i-1,j) + b(i,j)

invariant(a,b,j) ≡ j ≤ jmax + 1 ∧
    ∀imin+1 ≤ i ≤ imax, jmin ≤ j' < j.
    a(i,j') = b(i-1,j') + b(i,j')
```

2. Synthesizer infers spec from source

```
procedure sten(imin,imax,jmin,jmax,a,b)
    real, dim(imin:imax,jmin:jmax) :: a
    real, dim(imin:imax,jmin:jmax) :: b
    do j=jmin,jmax
        t = b(imin,j)
        do i=imin+1,imax
            q = b(i,j)
            a(i,j) = q + t
            t=q
        enddo
    enddo
end procedure
```

3. Retarget spec to GPU

```
int main() {
    ImageParam b(type_of<double>(),2);
    Func func;
    Var i, j;
    func(i,j) = b(i-1,j) + b(i,j);
    func.compile_to_file("ex1", b);
    return 0;
}
```

Lifted code can be executed on GPUs

17x speedup
void flowlet(Packet p) {
  p.new_hop = hash3(p.sport, p.dport, p.arrival) % HOPS;
  p.id = hash2(p.sport, p.dport) % FLOWS;
  if (p.arrival - last_time[p.id] > THRESHOLD)
    saved_hop[pkt.id] = p.new_hop;
  ...
}

Compile 10 data plane algorithms to HW + run at line rate
Verified Lifting

• User provides a simple DSL
• Automatically infers clean spec from input
• Tool generates a compiler that adapts code to new implementation
• Applied technique to various domains

Alvin Cheung
akcheung@cs.washington.edu