

Type Systems in Resource-Aware Programming: Opportunities and Challenges

Alcides Fonseca and Guilherme Espada

Developers need to understand the energetic impact of their software.

For this, they need tools!

TYPES OF TOOLS

Static Analysis
Tools

Performance
Benchmarking

Production
Monitoring



Faster feedback

More accuracy

- **SEEP**
 - Uses profiles to adapt to target architectures
 - Bottom-up approach using symbolic and concrete execution
 - Errors of $\sim 0.09\text{mJ}$
 - (<https://www4.cs.fau.de/Research/SEEP/>)
- **CiaoPP**
 - Encodes energy consumption as pre/post conditions
 - Works on the IR level
 - Errors of $\sim 0.09\text{mJ}$
 - (<https://www.cs.nmsu.edu/ALP/2017/10/energy-consumption-analysis-and-verification-using-ciaopp/>)

- **SDK4ED**
 - Uses static analysis on IR to build a local model
 - Uses Machine Learning to train the prediction, based on the code
 - Errors less than $6 \times 10^{-8}J$ or 20%
 - (<https://sdk4ed.eu>)
- **SAAD**
 - Static Analysis over Android Source Code
 - Identifies potential leaks, based on specific API usages
 - Very domain-specific

- EnergyTypes (2012)
 - Programming is explicit about energy modes
 - Energy modes can be dynamically adapted during runtime for energy efficiency.
- Eco (2015)
 - Extends EnergyTypes to be reactive to fine-grained temperature/energy changes.

```
1 phases { graphics <cpu main; main <cpu math; }
2 modes { hifi <: full; lofi <: hifi; }
3 class Main {
4     main () {
5         Recognizer rz = new Recognizer();
6         View v = adapt (new View());
7         while(true) {
8             Gesture g = processInput();
9             int result = rz.recognize(g);
10            v.paintOverlay(adapt[graphics] g, result);
11        }
12    }
13    Gesture processInput()
14    { ...
15        return new Gesture@phase(math)();
16    }
17 }
18 class Recognizer {
19     Matcher@mode(full) m1 = newM(0.995);
20     Matcher@mode(hifi) m2 = newM(0.9);
21     Matcher@mode(lofi) m3 = newM(0.5);
22     DistCal d1 = new Cosine@mode(hifi)();
23     DistCal d2 = new Euclidean@mode(lofi)();
24     int recognize(Gesture g) {
25         mswitch(batteryState()) {
26             case full: return m1.match(g, d1);
27             case hifi: return m2.match(g, d2);
28             case lofi: return m3.match(g, d2);
29         }
30     }
31     Matcher newM(double p) {
32         return adapt (new Matcher(p));
33     }
34     modev batteryState() {
35         if(Util.batterycharged()) return full;
```

- ENT (2017)
 - Extends Eco with dynamic checking and adaptation
- ECA (2015)
 - A language where energy consumption is modeled using dependent types
 - Dependent types are powerful, but much less usable

```
class Agent@mode<?->X> {
  Rule[] rules;
  Agent(String file) { rules = parseConfigurationFile(file); }
  attributor {
    if (Ext.battery >= 0.75) return full_throttle;
    else foreach (r: rules)
      { if (LocalOnlyRule instanceof r) return full_throttle; }
    else if (Ext.battery >= 0.50) return managed;
    else return energy_saver;
  }
  Set work(String url) {
    Site@mode<?> ds = Database.lookupSite(url);
    Site s = snapshot ds [_,X];
    return s.crawl(depth); // crawl
  }
  Rule[] parseConfigurationFile(String file) { ... }
  mcase<int> depth = mcase<int> {
    energy_saver: 1;
    managed: 2;
    full_throttle: 3;
  };
}
```

```
1 SoundSystem.on();
2 repeat #n begin
3   SoundSystem.playBeepAtHz(#hz);
4   System.sleep()
5 end;
6 SoundSystem.off()
```

Fig. 4: Example program.

Challenges

- The energy consumption of a function depends on its data.
- We need to be able to express the shape of data.

- The energy consumption of a function is not deterministic!
- We need to reason about probabilistic costs, not worst-case or averages.
- The industry deals with SLAs (99% of the cases).

- The energy consumption of a function depends on runtime variables.
- Will Garbage Collection kick in?
- Will the code be hot enough to be JIT'ed?
- We need to be able to abstract, but still consider these effects.

- The energy consumption of a function depends on runtime context.
- The same function can be faster or slower depending on what else is running.
- We need modular, but contextual analysis for parallel programs.

Opportunities

- Static Analysis are often ignored by developers
 - (<https://people.engr.ncsu.edu/ermurph3/papers/icse13b.pdf>)
- Type Systems are the most popular static analysis tool.

- Liquid Types combine Type Systems with Symbolic Execution ($\{x:\text{Int} \mid x > 10 + y\}$)
- Liquid Types can be Dependent Types
- Liquid Types can be combined with Resource Analysis (*Liquid Resource Types @ ICFP20*)
- Liquid Types can be used to reason about probabilities (*Safe Couplings: Coupled Refinement Types @ ICFP22*)

$\text{map} :: f : (a \rightarrow b) \rightarrow \{xs : \text{List } a \mid \text{length } xs \sim \text{Pois}(10)\} \rightarrow \{ys : \text{List } b\}$

where

$\text{energy map} = \text{energy } f * \text{length } xs + \$\text{overheads}$

$\text{assert energy } (f (+1) [1,2,3]) < 0.01\text{J}$

- Machine Learning has been used to refine models to consider unknown aspects (GC, JIT).
- Types can be used to extract feature for ML models.
- Genetic Programming can evolve modular cost models that are explainable, unlike Neural Networks.

- Types are helpful in software development
- Types can track resources, but do not consider:
 - Probabilistic reasoning of costs
 - Concurrency
 - Hidden effects, such as GC or JIT
- Liquid Types, tuned with ML can be a solution for this.

Type Systems in Resource-Aware Programming: Opportunities and Challenges

Alcides Fonseca and Guilherme Espada

me@alcidesfonseca.com