Data constraints at run time

Or: a whirlwind tour of some crazy ideas.

Stephen Kell

Stephen.Kell@cl.cam.ac.uk

Computer Laboratory

University of Cambridge

Why run time?

Type systems, static analysis etc. are great, but

- need to cope with heterogeneity
- run-time is a great unifier
- run-time is a necessary “last station” (for checking, . . .)
- dynamic analysis is cheaper than ever (multicore)

Crazy idea 1: consistency in transactional memory

```
try { tx.BeginTransaction();
      // txnl code
      done = tx.CommitTransaction();
} catch (AbortException e) { /* ... */}
```

Transactional code + data constraints = ?

- current TMs are not ACID (just A_I_)
- problem: performance!
  - rejoinder 1: treat it like assert()—can turn off
  - rejoinder 2: manycore might make this irrelevant
- worth investigating added actual performance costs

Idea 2: using data constraints in debuggers

Known data constraints are gold when debugging.

- programmer knows where consistency is expected
- debuggers can automate search for violations (if. . .)

Requires debugging information which records data constraints

- either programmer-provided (in-band, of out-of-). . .
- . . . or perhaps derived (from analysis, . . .)
- we will see more reasons why this is a good fit. . .

Crazy idea 3a: mining disk↔memory lenses

Lots of software uses ad-hoc data formats.

- config files, logs, undocumented file formats, . . .
- some too complex to be learnable from data alone

Use the programs themselves to mine the file formats.

- might use static analysis sensitive to filesystem calls
- but: lots of source languages (incl. Perl, sed, . . .)
- might simply analyse *traces* (harder)
  - of file accesses, branch instructions
  - of primitive translations (e.g. atoi(), sscanf(), et al)
- structure is latent in trace—can use PADS-like algorithm?

Crazy idea 3b: sharing the constraints around

Suppose we know constraints for file format, but not the corresponding in-memory data—or vice-versa.

- i.e. one or other documented, but not both
- want to share these around
- idea: propagate constraints through lenses

Useful: debug info can describe both disk and mem formats

- (with some minor extensions)
- (bonus: already have tools for writing such lenses manually)

(Slightly off-topic) idea 4: using lenses to save memory

Memory remains scarce (unlike CPU time…).

- Consider a program reading in some large files.
- Given disk↔mem lenses…
- can discard replicas, even *non-verbatim* (transformed)
- … when computational distance is small

Generalises from existing memory-saving approaches

- web cache design: explicit disk↔mem transfer is bad
  - ♦ also avoids bad interaction with demand paging
  - ♦ only works when disk/mem reps are *verbatim*
    - cf. small computational distance
- page-sharing work (Satori, Difference Engine, …)

Conclusions

We've seen a few runtime-centric ideas about data.

- mostly easily added to existing tools
- mostly under-thought!

Message: extending runtimes is good

- static analysis innovations need dynamic complement
- propagate innovations into shared infrastructure…
- … establishing commonality in the process

Subliminal message: debug info is a good place to innovate

- mediates language and runtime views
- can capture persistent *and* in-memory representations