Research Methods in computer science

Fall 2013

Lecture 21

Omprakash Gnawali

November 7, 2013

Agenda

Conference Plans

Paper writing (Expts and Evaluation)

Paper review

HW11 questions

HW12

How to write a great research paper

Simon Peyton Jones
Microsoft Research, Cambridge

Structure (conference paper)

- Title (1000 readers)
- Abstract (4 sentences, 100 readers)
- Introduction (1 page, 100 readers)
- The problem (1 page, 10 readers)
- My idea (2 pages, 10 readers)
- The details (5 pages, 3 readers)
- Related work (1-2 pages, 10 readers)
- Conclusions and further work (0.5 pages)

Structure

- Abstract (4 sentences)
- **Introduction** (1 page)
- The problem (1 page)
- My idea (2 pages)
- The details (5 pages)
- Related work (1-2 pages)
- Conclusions and further work (0.5 pages)

Contributions should be refutable

NO!	YES!
We describe the WizWoz system. It is really cool.	We give the syntax and semantics of a language that supports concurrent processes (Section 3). Its innovative features are...
We study its properties	We prove that the type system is sound, and that type checking is decidable (Section 4)
We have used WizWoz in practice	We have built a GUI toolkit in WizWoz, and used it to implement a text editor (Section 5). The result is half the length of the Java version.

Introduction – Another take

1. What is the problem?
2. Why is it interesting and important?
3. Why is it hard?
4. Why hasn't it been solved before?
5. What are the key components of my approach and results?

Courtesy Jennifer Widom

Related work

Fallacy	To make my work look good, I have to make other people’s work look bad

The truth: credit is not like money

Giving credit to others does not diminish the credit you get from your paper

- Warmly acknowledge people who have helped you
- Be generous to the competition. “In his inspiring paper [Foo98] Foogle shows.... We develop his foundation in the following ways...”
- Acknowledge weaknesses in your approach

Credit is not like money

Failing to give credit to others can kill your paper

If you imply that an idea is yours, and the referee knows it is not, then either

- You don't know that it's an old idea (bad)
- You do know, but are pretending it's yours (very bad)

Related Work

- Collect papers
  - Identify related papers and their related papers and their related papers…
  - ACM/IEEE libraries
  - Google/Google Scholar
- Organize the papers
  - Use some structure (tables, graphs, etc.)
- Relate to each work/group you mention

Related Work - Common Structure

Sub sections (3-4)

Discuss 1-5 papers per subsection

It is ok to have "Other" subsection

Try to have subsection titles but at least have separate paragraphs

Presenting the idea

3. The idea

Consider a bifircuated semi-lattice D, over a hyper-modulated signature S. Suppose p_i is an element of D. Then we know for every such p_i there is an epi-modulus j, such that $p_j < p_i$.

- Sounds impressive...but
- Sends readers to sleep
- In a paper you MUST provide the details, but FIRST convey the idea

Presenting the idea

- Explain it as if you were speaking to someone using a whiteboard
- **Conveying the intuition is primary**, not secondary
- Once your reader has the intuition, she can follow the details (but not vice versa)
- Even if she skips the details, she still takes away something valuable

Putting the reader first

- **Do not** recapitulate your personal journey of discovery. This route may be soaked with your blood, but that is not interesting to the reader.

- Instead, choose the most direct route to the idea.

The payload of your paper

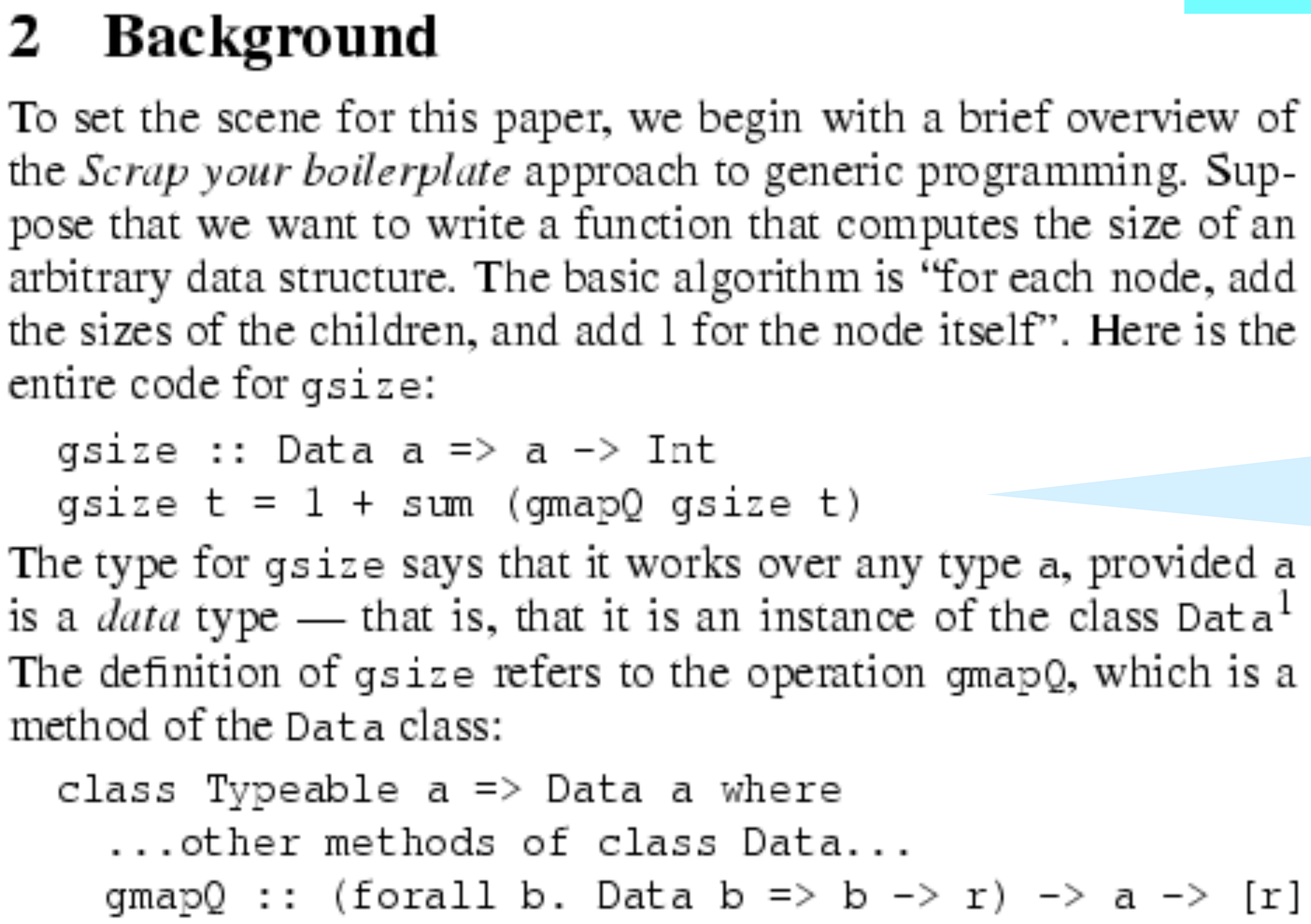
Introduce the problem, and your idea, using

EXAMPLES

and only then present the general case

Using examples

The Simon PJ question: is there any typewriter font?



2 Background

To set the scene for this paper, we begin with a brief overview of the *Scrap your boilerplate* approach to generic programming. Suppose that we want to write a function that computes the size of an arbitrary data structure. The basic algorithm is "for each node, add the sizes of the children, and add 1 for the node itself". Here is the entire code for `gsize`:

```
gsize :: Data a => a -> Int
gsize t = 1 + sum (gmapQ gsize t)
```

The type for `gsize` says that it works over any type `a`, provided `a` is a *data* type — that is, that it is an instance of the class `Data`[1] The definition of `gsize` refers to the operation `gmapQ`, which is a method of the `Data` class:

```
class Typeable a => Data a where
  ...other methods of class Data...
  gmapQ :: (forall b. Data b => b -> r) -> a -> [r]
```

Example right away

Diagrams to help explain the concepts

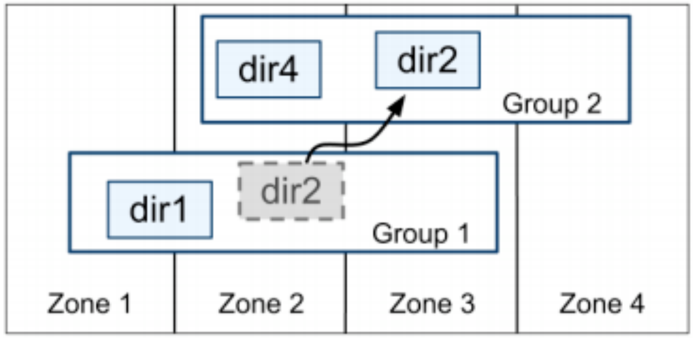


Figure 3: Directories are the unit of data movement between Paxos groups.

The details: evidence

- Your introduction makes claims
- The body of the paper provides **evidence to support each claim**
- Check each claim in the introduction, identify the evidence, and forward-reference it from the claim
- Evidence can be: analysis and comparison, theorems, measurements, case studies

Describing Experiment

Sufficient details for an expert in your field to redo the experiment.

Datasets, Systems, Code version, Algorithms

Something seemingly unimportant might be important!

In our experiments on frame-semantic parsing, we use two sets of data:

1. **SemEval 2007 data:** In benchmark experiments for comparison with previous state of the art, we use a dataset that was released as part of the **SemEval 2007 shared task** on frame-semantic structure extraction (Baker, Ellsworth, and Erk 2007). Full text annotations in this dataset consisted of a few thousand sentences containing multiple targets, each annotated with a frame and its arguments. The

2. **FrameNet 1.5 release:** A more recent version of the FrameNet lexicon was released in 2010.[8] We also test our statistical models (only frame identification and argument identification) on this dataset to get an estimate of how much improvement additional data can provide. Details of this dataset are shown in the second column of Table 1. Of the 78 documents in this release with full text annotations, we selected 55 (19,582 targets) for training and held out the remaining 23 (4,458 targets) for testing. There are fewer target annotations per sentence in the test set than the training set.[9] Das and Smith (2011, supplementary material) give the names of the test documents for fair replication of our work. We also randomly

7. Experiments

Section 7.1 details the datasets, experimental set-up, and classifiers used. We first compare the proposed methods to construct sets A of measure τ, reported in Section 7.2, and the proposed estimation methods for the moments of $P_{X|z}$, reported in Section 7.3. Then in Section 7.4, we show that applying a dimensionality reduction method can greatly reduce the computation needed at test time. Lastly, we compare our recommended reliable classifier to other approaches to early classification.

7.1 Experimental Set-up and Details

We demonstrate performance using all of the time-series datasets available on the *UCR Time-Series Classification and Clustering Page* (E. Keogh and Ratanamahatana, 2006)

5 Evaluation

We first measure Spanner's performance with respect to replication, transactions, and availability. We then provide some data on TrueTime behavior, and a case study of our first client, F1.

5.1 Microbenchmarks

Table 3 presents some microbenchmarks for Spanner. These measurements were taken on timeshared machines: each spanserver ran on scheduling units of 4GB RAM and 4 cores (AMD Barcelona 2200MHz). Clients were run on separate machines. Each zone contained one spanserver. Clients and zones were placed in a set of datacenters with network distance of less than 1ms. (Such a layout should be commonplace: most applications do not need to distribute all of their data worldwide.) The test database was created with 50 Paxos groups with 2500 directories. Operations were standalone reads and writes of 4KB. All reads were served out of memory after a compaction, so that we are only measuring the overhead of Spanner's call stack. In addition, one unmeasured round of reads was done first to warm any location caches.

Graphs

Refer to each graph or figure in the text

Describe what is on the graph

Also

- What do we learn?
- Why does this happen?
- Exceptions?

Caption

Watch for redundancy between labels, legends and caption

Essential: short summary of what we learn

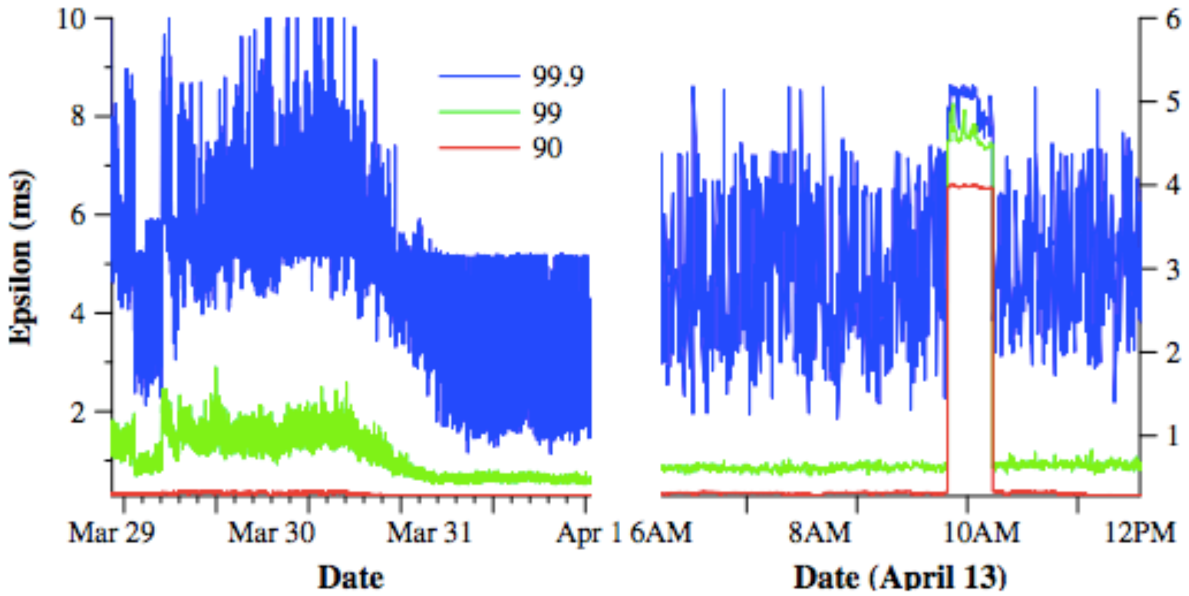


Figure 6: Distribution of TrueTime ϵ values, sampled right after timeslave daemon polls the time masters. 90th, 99th, and 99.9th percentiles are graphed.

An Extreme Example…

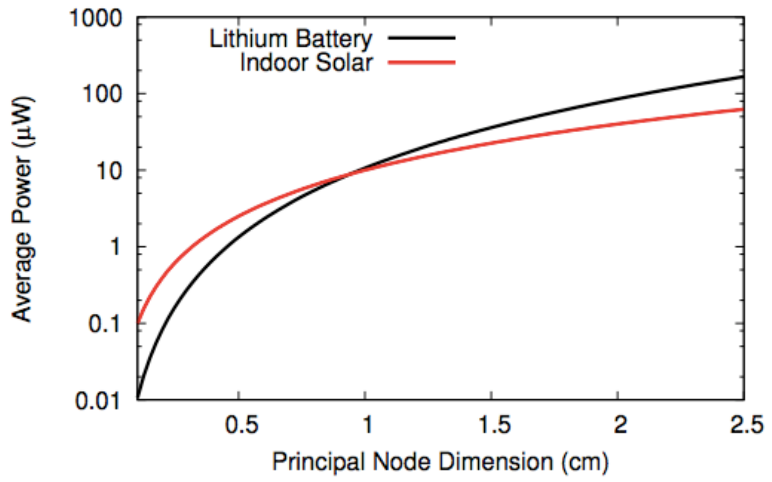


Figure 1: An energy-harvesting reality check. Shows how power harvested from indoor solar compares with power drawn from an internal battery. As a cubic sensor's length L falls below a centimeter, a solar cell of size L^2 can deliver higher average power than a Lithium battery of size L^3, over a seven year horizon. The key to continued sensor scaling lies in shifting the primary energy supply from battery to solar, and dealing with the implications of a dramatically reduced supply.

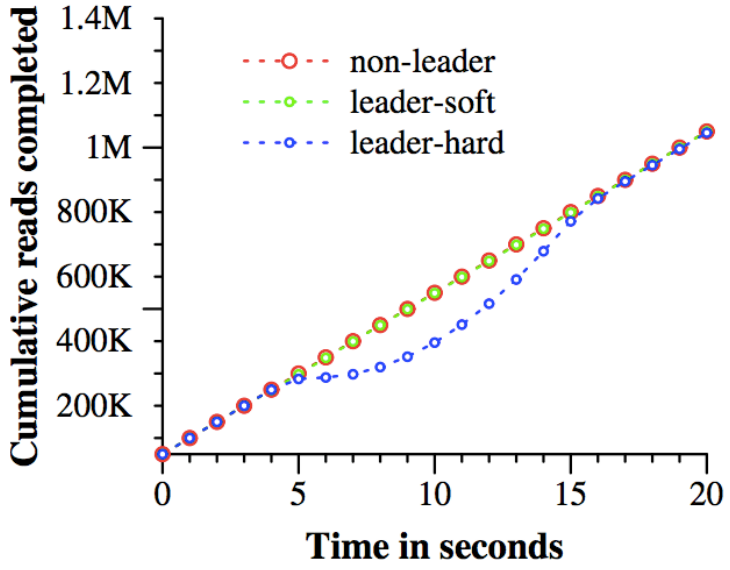


Figure 5 illustrates the availability benefits of running Spanner in multiple datacenters. It shows the results of three experiments on throughput in the presence of datacenter failure, all of which are overlaid onto the same time scale. The test universe consisted of 5 zones Z_i, each of which had 25 spanservers. The test database was sharded into 1250 Paxos groups, and 100 test clients constantly issued non-snapshot reads at an aggregrate rate of 50K reads/second. All of the leaders were explicitly placed in Z_1. Five seconds into each test, all of the servers in one zone were killed: *non-leader* kills Z_2; *leader-hard* kills Z_1; *leader-soft* kills Z_1, but it gives notifications to all of the servers that they should handoff leadership first.

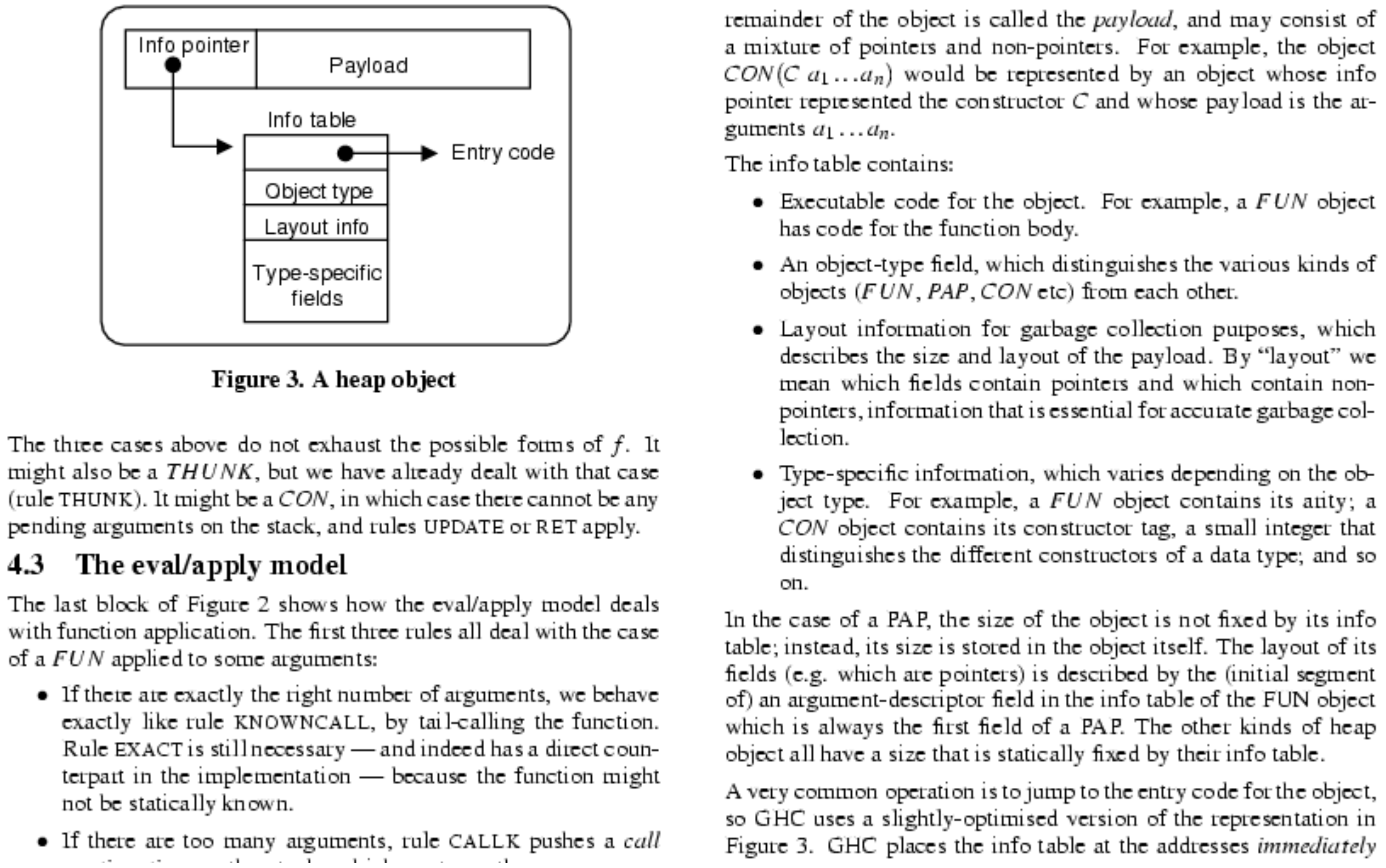
Killing Z_2 has no effect on read throughput. Killing Z_1 while giving the leaders time to handoff leadership to a different zone has a minor effect: the throughput drop is not visible in the graph, but is around 3-4%. On the other hand, killing Z_1 with no warning has a severe effect: the rate of completion drops almost to 0. As leaders get re-elected, though, the throughput of the system rises to approximately 100K reads/second because of two artifacts of our experiment: there is extra capacity in the system, and operations are queued while the leader is unavailable. As a result, the throughput of the system rises before leveling off again at its steady-state rate.

We can also see the effect of the fact that Paxos leader leases are set to 10 seconds. When we kill the zone, the leader-lease expiration times for the groups should be evenly distributed over the next 10 seconds. Soon after each lease from a dead leader expires, a new leader is elected. Approximately 10 seconds after the kill time, all of the groups have leaders and throughput has recovered. Shorter lease times would reduce the effect of server deaths on availability, but would require greater amounts of lease-renewal network traffic. We are in the process of designing and implementing a mechanism that will cause slaves to release Paxos leader leases upon leader failure.

Visual structure

- Give strong visual structure to your paper using
  - sections and sub-sections
  - bullets
  - italics
  - laid-out code
- Find out how to draw pictures, and use them

Visual structure



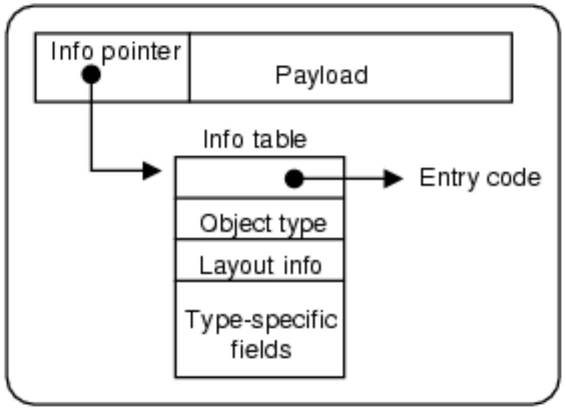


Figure 3. A heap object

The three cases above do not exhaust the possible forms of f. It might also be a *THUNK*, but we have already dealt with that case (rule THUNK). It might be a *CON*, in which case there cannot be any pending arguments on the stack, and rules UPDATE or RET apply.

4.3 The eval/apply model

The last block of Figure 2 shows how the eval/apply model deals with function application. The first three rules all deal with the case of a *FUN* applied to some arguments:

- If there are exactly the right number of arguments, we behave exactly like rule KNOWNCALL, by tail-calling the function. Rule EXACT is still necessary — and indeed has a direct counterpart in the implementation — because the function might not be statically known.
- If there are too many arguments, rule CALLK pushes a *call*

remainder of the object is called the *payload*, and may consist of a mixture of pointers and non-pointers. For example, the object $CON(C\ a_1 \ldots a_n)$ would be represented by an object whose info pointer represented the constructor C and whose payload is the arguments $a_1 \ldots a_n$.

The info table contains:

- Executable code for the object. For example, a *FUN* object has code for the function body.
- An object-type field, which distinguishes the various kinds of objects (*FUN*, *PAP*, *CON* etc) from each other.
- Layout information for garbage collection purposes, which describes the size and layout of the payload. By "layout" we mean which fields contain pointers and which contain non-pointers, information that is essential for accurate garbage collection.
- Type-specific information, which varies depending on the object type. For example, a *FUN* object contains its arity; a *CON* object contains its constructor tag, a small integer that distinguishes the different constructors of a data type; and so on.

In the case of a PAP, the size of the object is not fixed by its info table; instead, its size is stored in the object itself. The layout of its fields (e.g. which are pointers) is described by the (initial segment of) an argument-descriptor field in the info table of the FUN object which is always the first field of a PAP. The other kinds of heap object all have a size that is statically fixed by their info table.

A very common operation is to jump to the entry code for the object, so GHC uses a slightly-optimised version of the representation in Figure 3. GHC places the info table at the addresses *immediately*

Use the active voice

The passive voice is "respectable" but it DEADENS your paper. Avoid it at all costs.

NO	YES
It can be seen that...	We can see that...
34 tests were run	We ran 34 tests
These properties were thought desirable	We wanted to retain these properties
It might be thought that this would be a type error	You might think this would be a type error

"We" = you and the reader

"We" = the authors

"You" = the reader

Use simple, direct language

NO	YES
The object under study was displaced horizontally	The ball moved sideways
On an annual basis	Yearly
Endeavour to ascertain	Find out
It could be considered that the speed of storage reclamation left something to be desired	The garbage collector was really slow

Feedback

Three essential components

- Summary
- Strength
- Weakness

Should cover style and substance

Paper Review

Papers are judged for

- Novelty
- Technical Rigor
- Claims and their proofs

Fatal flaws

Provide detailed comments to the authors to improve the next version of the paper

Some examples

- Commenting software systems
- Sample reviews