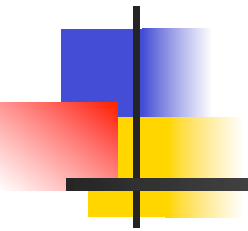


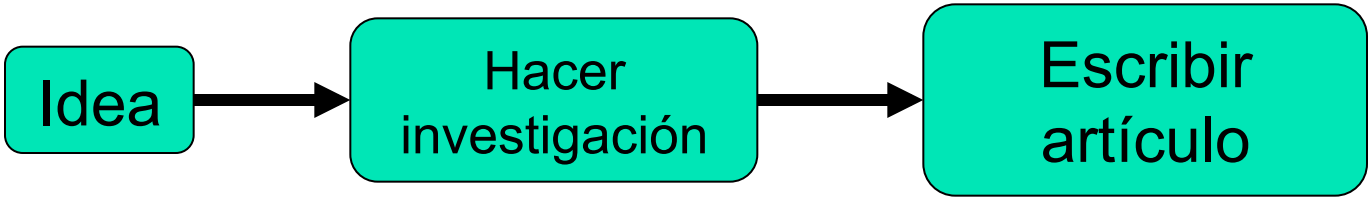
Cómo escribir un gran artículo de investigación

Simon Peyton Jones
Microsoft Research, Cambridge

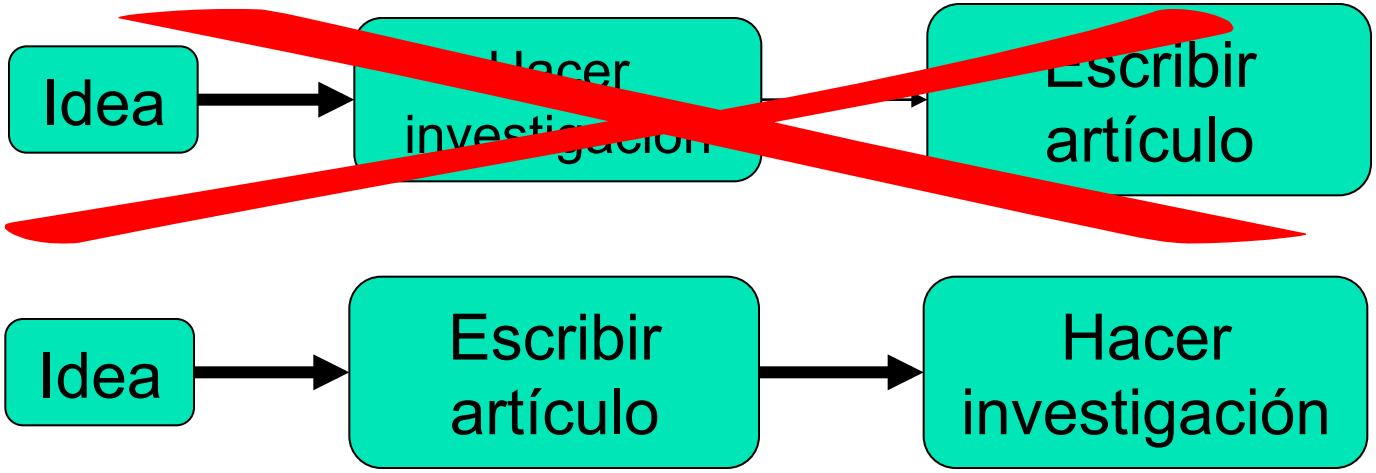
Escribir artículos es una habilidad

- Muchos artículos están mal escritos
- La buena escritura es un habilidad que se puede aprender
- Es una habilidad que vale la pena aprender para:
  - Obtener más puntitos para evaluación (más artículos aceptados, etc)
  - Tus ideas tendrán más impacto
  - Tendrás mejores ideas

Escribiendo artículos: modelo 1



Escribiendo artículos: modelo 2



- Nos forza a ser claros y enfocados
- Cristaliza aquéllo que no entendemos
- Abre el camino al diálogo con otros: verificación de factibilidad, crítica y colaboración

No te intimides

Falacia Necesitas tener una idea fantástica antes de poder escribir un artículo. (Todo el mundo parece hacerlo.)

Escribe un artículo,
y da una presentación, acerca de
cualquier idea,
no importa qué simple o insignificante te parezca

No te intimides

Escribe un artículo, y da una presentación, acerca de cualquier idea, no importa qué simple o insignificante te parezca

- **Escribiendo el artículo es como en primer lugar desarrollas la idea**
- Generalmente resulta ser más interesante y desafiante de lo que parece al principio

El propósito de tu artículo

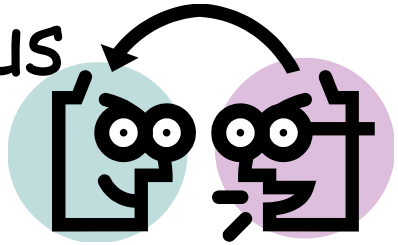
¿Porqué molestarse?

Falacia

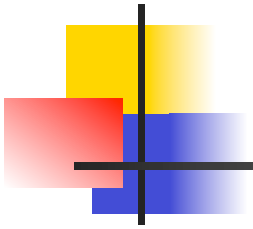
Escribimos artículos y damos presentaciones principalmente para impresionar a otros, ganar reconocimiento y ser ascendidos

Los artículos comunican ideas

- Tu objetivo: infectar la mente de tu lector con **tu idea**, como un virus
- Los artículos son mucho más durables que los programas (piensa en Mozart)



Las grandes ideas son (literalmente) inútiles si las guardas para tí mismo



La Idea

Idea
Una instrospección re-usable,
Útil para el lector

- Averigua cuál es tu idea
- Asegúrate que el lector no tiene duda sobre qué es tu idea. Se 100% explícito:
  - "La idea principal de este artículo es ..."
  - "En esta sección presentamos las principales contribuciones del artículo."
- Muchos artículos contienen buenas ideas, pero no exponen cuáles son.

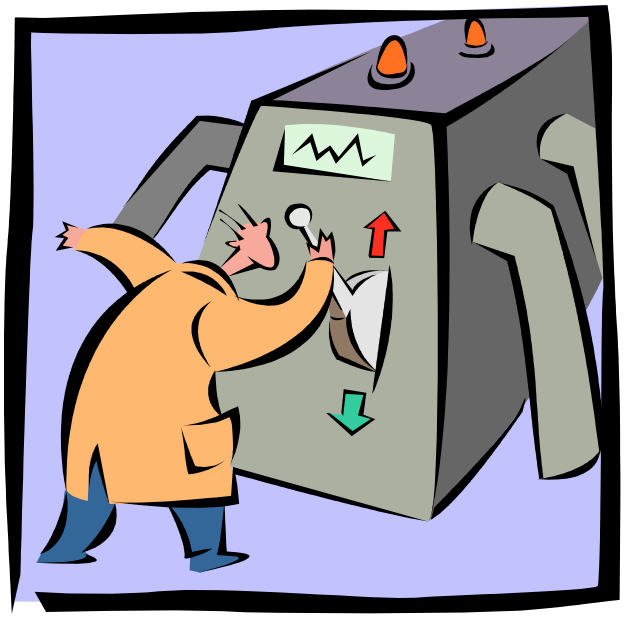
Un “ping”

- Tu artículo debe tener solo un “ping”: una idea clara y nítida
- Lee tu artículo de nuevo: ¿puedes oir el “ping”?
- Puede que no sepas exactamente cuál es el “ping” cuando comienzas a escribir; pero debes saberlo cuando terminas
- Si tienes muchas ideas, escribe muchos artículos

Gracias a Joe Touch por un “ping”

El propósito de tu artículo no es...

Describir el sistema WizWoz



- Tu lector no tiene un WizWoz
- Está principlamente interesado en inteligencia re-usable, no artefactos ejecutables

Tu flujo narrativo

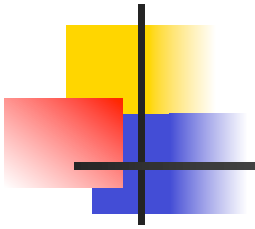
- Aquí hay un problema
- Es un problema interesante
- Es un problema no resuelto
- **Aquí está idea**
- Mi idea funciona (detalles, datos)
- Así es como mi idea se compara con las aproximaciones de otras personas

¡Desearía saber cómo resolver eso!

Ya veo cómo funciona. ¡Ingenioso!

Estructura (artículo de conferencia)

- Título (1000 lectores)
- Resumen (4 enunciados, 100 lectores)
- Introducción (1 página, 100 lectores)
- El problema (1 página, 10 lectores)
- Mi idea (2 páginas, 10 lectores)
- Los detalles (5 páginas, 3 lectores)
- Trabajo relacionado (1-2 páginas, 10 lectores)
- Conclusiones y trabajo futuro (0.5 páginas)

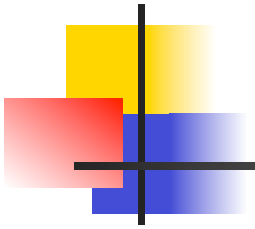


El resumen (abstract)

- Generalmente, yo escribo el resumen al final
- Se usa por los miembros de los comités de programa para decidir qué artículos leer
- Cuatro enunciados [Kent Beck]
  1. Enuncia el problema
  2. Dí porqué es un problema interesante
  3. Dí qué logra tu solución
  4. Dí que sigue después de tu solución

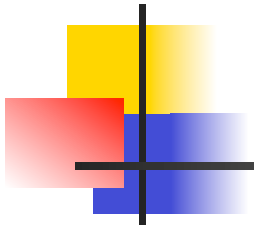
¿Qué hacer?

1. Muchos artículos están mal escritos y son difíciles de entender
2. Esto es una pena, porque sus buenas ideas pueden no ser apreciadas
3. Siguiendo algunas guías sencillas se puede dramáticamente mejorar la calidad de tud artículos
4. Tu trabajo será más usado, y la realimentación que obtengas de otros al mismo tiempo mejora tu investigación



Estructura

- Resumen (4 enunciados)
- **Introducción** (1 página)
- El problema (1 página)
- Mi idea (2 páginas)
- Los detalles (5 páginas)
- Trabajo relacionado (1-2 páginas)
- Conclusiones y trabajo futuro (0.5 páginas)

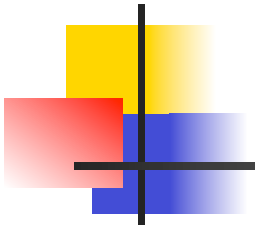


La introducción (1 página)

1. **Describe el problema**
2. **Enuncia tus contribuciones**

...y eso es todo

¡UNA PÁGINA!



Describe el problema

1 Introduction

There are two basic ways to implement function application in a higher-order language, when the function is unknown: the *push/enter* model or the *eval/apply* model [11]. To illustrate the difference, consider the higher-order function **zipWith**, which zips together two lists, using a function **k** to combine corresponding list elements:

```
zipWith :: (a->b->c) -> [a] -> [b] -> [c]
zipWith k []     []     = []
zipWith k (x:xs) (y:ys) = k x y : zipWith xs ys
```

Here **k** is an *unknown function*, passed as an argument; global flow analysis aside, the compiler does not know what function **k** is bound to. How should the compiler deal with the call **k x y** in the body of **zipWith**? It can't blithely apply **k** to two arguments, because **k** might in reality take just one argument and compute for a while before returning a function that consumes the next argument; or **k** might take three arguments, so that the result of the **zipWith** is a list of functions.

Usa un ejemplo para introducir el problema

Enuncia tus contribuciones

- Escribe primero la lista de contribuciones
- La lista de contribuciones conduce todo el artículo: el artículo substancía las pretenciones que has hecho
- El lector piensa "caramba, si pueden lograr esto, eso debe ser interesante; mejor continúo leyendo"

Enuncia tus contribuciones

Which of the two is best in practice? The trouble is that the evaluation model has a pervasive effect on the implementation, so it is too much work to implement both and pick the best. Historically, compilers for strict languages (using call-by-value) have tended to use eval/apply, while those for lazy languages (using call-by-need) have often used push/enter, but this is 90% historical accident — either approach will work in both settings. In practice, implementors choose one of the two approaches based on a qualitative assessment of the trade-offs. In this paper we put the choice on a firmer basis:

- We explain precisely what the two models are, in a common notational framework (Section 4). Surprisingly, this has not been done before.
- The choice of evaluation model affects many other design choices in subtle but pervasive ways. We identify and discuss these effects in Sections 5 and 6, and contrast them in Section 7. There are lots of nitty-gritty details here, for which we make no apology — they were far from obvious to us, and articulating these details is one of our main contributions.

  In terms of its impact on compiler and run-time system complexity, eval/apply seems decisively superior, principally because push/enter requires a stack like no other: stack-walking

Lista por puntos de las contribuciones

¡No dejes al lector adivinar cuáles son tus contribuciones!

Las contribuciones deben ser refutables

¡NO!	¡SÍ!
Describimos el sistema WizWoz. Es realmente fabuloso.	Damos la sintaxis y semántica de un lenguaje que soporta procesos concurrentes (Sección 3). Sus características innovadoras son...
Estudiamos sus propiedades	Comprobamos que este tipo de sistema es consistente, y que la verificación de tipos (type checking) es decidible (Sección 4)
Hemos usado WizWoz en la práctica	Se ha construido una herramienta GUI en WizWoz, y se ha usado para implementar un editor de texto (Sección 5). El resultado tiene la mitad de longitud de una versión en Java.

Omite "el resto de este artículo es..."

- No:

> "El resto de este artículo se estructura como sigue. La Sección 2 introduce el problema. La Sección 3 ... Finalmente, la Sección 8 concluye".

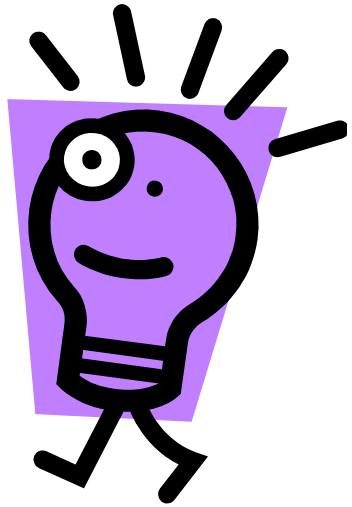
- En su lugar, **usa referencias hacia adelante en la descripción de la introducción**.
  La introducción (incluyendo las contribuciones) debe revisar todo el artículo, y por tanto, debe hacer referencias hacia adelante a toda parte importante.

Estructura

- Resumen (4 enunciados)
- Introducción (1 página)
- ~~Trabajo relacionado~~
- El problema (1 página)
- Mi idea (2 páginas)
- Los detalles (5 páginas)
- Trabajo relacionado (1-2 páginas)
- Conclusiones y trabajo futuro (0.5 páginas)

¡Todavía no "trabajo relacionado"!

Trabajo relacionado



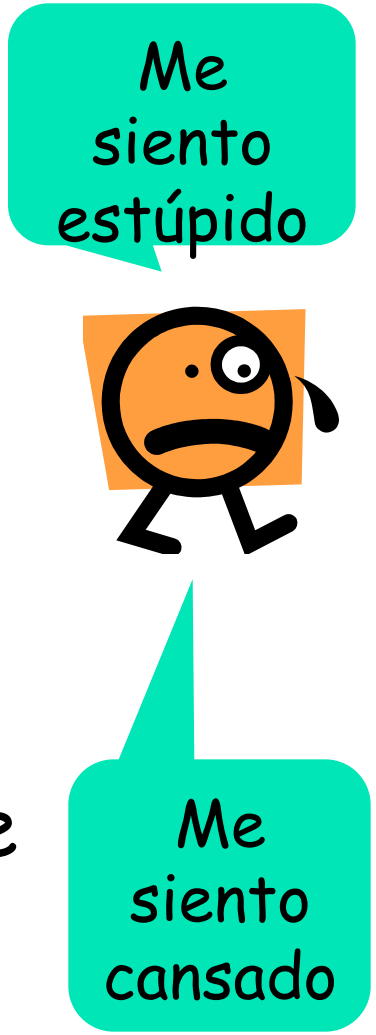
Tu lector

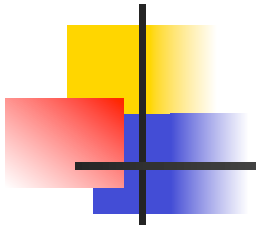
Tu idea

Adoptamos la noción de transacción de Brown [1], como se modifica para sistemas distribuidos por White [2], usando el algoritmo de interpolación de cuatro-fases de Green [3]. Este trabajo difiere de White en nuestro avanzado protocolo de revocación, que tiene que ver con el caso de inversión de prioridad como ha sido descrito por Yellow [4].

Todavía no trabajo relacionado

- Problema 1: el lector no sabe acerca del problema aún; tu (cuidadosamente preparada) descripción de varios pros y contras técnicos queda absolutamente incomprensible
- Problema 2: describir aproximaciones alternativas se atraviesa entre el lector y tu idea





Estructura

- Resumen (4 enunciados)
- Introducción (1 página)
- El problema (1 página)
- Mi idea (2 páginas)
- Los detalles (5 páginas)
- Trabajo relacionado (1-2 páginas)
- Conclusiones y trabajo futuro (0.5 páginas)

Presentando la idea

3. La idea

Considere una semi-malla bifurcada D, sobre una interfaz hipermodulada S. Suponga p_i es un elemento de D. Entonces, sabemos que para cada p_i hay un epimódulo j, tal que $p_j < p_i$.

- Suena impresionante... pero
- Manda a tu lector a dormir
- En un artículo DEBES proveer los detalles, pero PRIMERO transmitir la idea

Presentando la idea

- Explícalo como si hablaras con alguien usando un pizarrón
- **Transmitir la intuición es primario**, no lo secundario
- Una vez que tu lector tiene la intuición, puede seguir con los detalles (pero no al revés)
- Aún si el lector se salta los detalles, aún le queda algo valioso

Poner al lector primero

- **No** recapitules tu "viaje personal de descubrimiento". Esa ruta está empapada de tu sangre y sudor, pero eso no le interesa al lector.

- En lugar de eso, escoge la ruta más directa a la idea.

Lo sustantivo de tu artículo

Introduce el problema, y tu idea, usando

EJEMPLOS

y solo entonces presenta el caso general

Usando ejemplos

La pregunta Simon PJ: ¿hay un tipo de letra de máquina de escribir?

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)
```

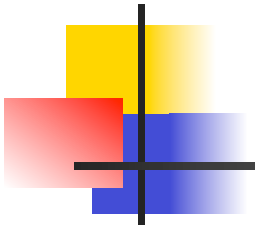
The type for `gsize` says that it works over any type `a`, provided `a` is a *data* type — that is, that it is an instance of the class `Data`[1] The definition of `gsize` refers to the operation `gmapQ`, which is a method of the `Data` class:

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

Ejemplo de inmediato

Los detalles: la evidencia

- Tu introducción hace pretensiones
- El cuerpo del artículo provee **evidencias que soportan cada pretensión**
- Verifica cada pretensión en la introducción, identifica la evidencia, y referencíalo hacia adelante desde la pretensión
- Evidencia puede ser: análisis y comparasión, teoremas, mediciones, casos de estudio



Estructura

- Resumen (4 enunciados)
- Introducción (1 página)
- El problema (1 página)
- Mi idea (2 páginas)
- Los detalles (5 páginas)
- **Trabajo relacionado** (1-2 páginas)
- Conclusiones y trabajo futuro (0.5 páginas)

Trabajo relacionado

Falacia — Para hacer ver bien mi trabajo, tengo que hacer ver mal el trabajo de otras personas

La verdad: el crédito no es como el dinero

Dar crédito a otros no disminuye el crédito que obtienes por tu artículo

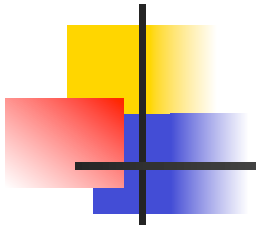
- Cálidamente reconoce a las personas que te han ayudado
- Sé generoso al final. "En su inspirado artículo [Foo98] Foogle muestra.... Desarrollamos sus fundamentos en las siguientes formas..."
- Reconoce las debilidades en tu aproximación

El crédito no es como el dinero

No dar crédito a otros puede matar tu artículo

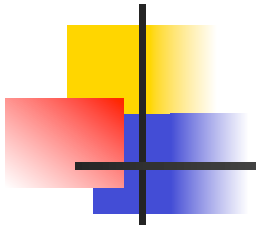
Si insinuas que una idea es tuya, y el árbitro sabe que no es así, entonces puede creer que

- Tú no sabes que es una idea ya conocida (malo)
- Tú lo sabes, pero quieres pasarla como tuya (muy malo)



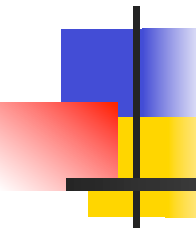
Estructura

- Resumen (4 enunciados)
- Introducción (1 página)
- El problema (1 página)
- Mi idea (2 páginas)
- Los detalles (5 páginas)
- Trabajo relacionado (1-2 páginas)
- Conclusiones y trabajo futuro (0.5 páginas)

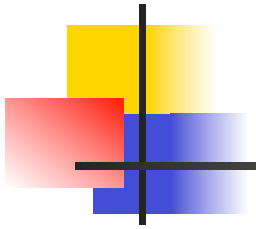


Conclusiones y trabajo futuro

- Sé breve.



El proceso de escritura



El proceso

- Comienza pronto. Muy pronto.
  - Los artículos escritos con prisas son rechazados.
  - Los artículos son como el vino: necesitan tiempo para madurar
- Colabora
- Usa CVS para apoyar la colaboración

Obteniendo ayuda

Haz que lean tu artículo tantos amigos "conejillos de indias" como sea posible

- Los expertos son buenos
- Los no-expertos son también muy buenos
- ¡Cada lector puede leer tu artículo por primera vez y una sola vez! De modo que úsalos con cuidado
- Explica cuidadosamente qué es lo que deseas ("Me perdí aquí" es mucho más importante que "Jarva está mal escrito".)

Obteniendo ayuda experta

- Un buen plan: cuando pienses que ya terminaste, envía un borrador a la competencia diciendo "¿podrían ayudarme a asegurar que describo su trabajo con justicia?".
- Frecuentemente responderán con crítica útil (están interesados en el área)
- Ellos pueden muy probablemente ser tus árbitros de cualquier modo, así que obtener sus comentarios o críticas con antelación es MUY BUENO.

Escucha a tus revisores

Trata cada revisión como polvo de oro

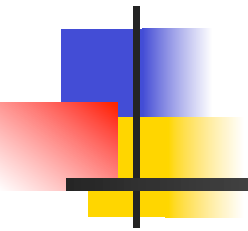
Sé (sinceramente) agradecido por la crítica así como por la alabanza

Esto es realmente, realmente, realmente duro

Pero es
realmente, realmente, realmente, realmente, realmente, realmente, realmente, realmente, importante

Escuchando a tus revisores

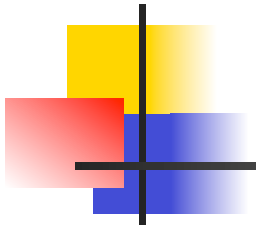
- Lee cada crítica como una sugerencia positiva de algo que puedes explicar más claramente
- NO respondas "tú, estúpido, quiero decir X". Corrige el artículo de modo que X sea notorio hasta para el lector más estúpido.
- Agradéceles calurosamente. Han dado parte de su tiempo por tí.



Lenguaje y estilo

Cuestiones básicas

- Somete respetando el plazo
- Manten las restricciones de longitud
  - No recortes los márgenes
  - No **uses font de 6 puntos**
  - Ocasionalmente, provee evidencia de apoyo (v.gr. Datos experimentales, o una demostración escrita) en un apéndice
- Siempre utiliza corrector de ortografía



Estructura visual

- Dale una estructura visual fuerte a tu artículo usando
  - secciones y sub-secciones
  - viñetas
  - cursivas
  - código trazado
- Averigua cómo dibujar figuras, y úsalas

Estructura visual

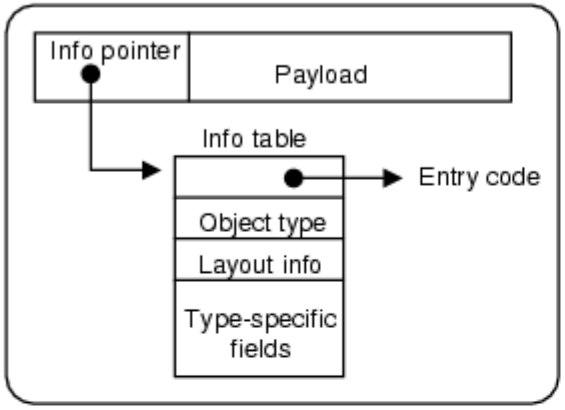


Figure 3. A heap object

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*

Usa la voz activa

La voz pasiva es "respetable" pero MATA tu artículo. Evítala a toda costa.

NO	SÍ
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

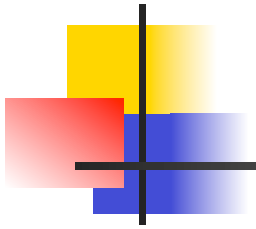
"We" = tú y el lector

"We" = los autores

"You" = el lector

Usa lenguaje simple y directo

NO	SÍ
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



Resumen

Si no recuerdas otra cosa, recuerda esto:

- **Identifica tu idea clave**
- **Haz tus contribuciones explícitas**
- **Usa ejemplos**

Un buen punto de inicio:

"Advice on Research and Writing"

http://www-2.cs.cmu.edu/afs/cs.cmu.edu/user/mleone/web/how-to.html