

How to Write a Good Paper and How to Give a Good Talk

(some of the material also applies to “how to teach a class”)



ILLUSTRATION BY ANTHONY TRUSSARD



Liang Huang

Queens College and Graduate Center
The City University of New York

Why I am giving this talk...

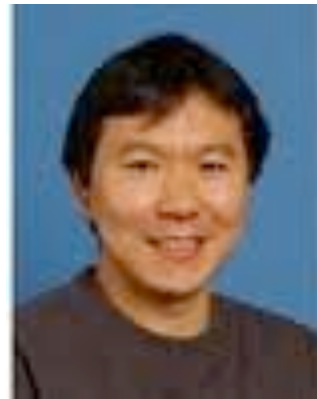
- not because I **am** a good writer (in fact I'm not)...
- but because I **was** a terrible writer!



David Chiang (usc)
paper writing



Kevin Knight (usc)
presentation



Shanghua Teng (usc)
teaching



Ed Hovy (usc)
proposal writing

How I learned writing

- '03: knew nothing about writing (though I wrote several papers in China)
- '04-5: all I wrote was crap; David turned them into beauty
- '06-7: some progress by writing, writing, and writing...
- one of the reviews for a submission with David (rejected)
 - *“in general this paper is written with admirable clarity, except for it doesn't seem to be written by a single author or with the same level of discretion...”* (this made me not sad about the rejection... :P)
 - turns out David had revised all but one section
- first single-author paper (not a good one, but great practice)
- '08 and on: all my submissions got 4 or 5 in “clarity”

How I learned writing

- **fallacy:** students learn to write mainly from advisors
- **truth:** learn from anybody whom you can learn from
- I learned writing mainly from...
- and from writing seminars of...
- and from the slides by...



D. Chiang



K. Knight



L. Saul



B. Pierce



S. Peyton-Jones

the rest of the talk
is largely based on
Simon PJ's slides.



D. Gildea

This is NOT an English class

- writing is not about language, but about logic
 - writing is equally hard for both native and non-native speakers of English
 - a bad paper is bad in any language
- different levels of writing
 - high-level (paper): global shape, logic, argument, style
 - mid-level (discourse): coherence within a paragraph
 - low-level (sentences): ordering of words and phrases
 - lowest-level (words): word choice, grammar

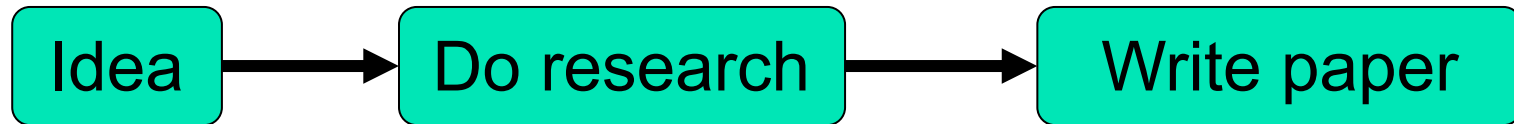
First Principle: Audience-Centric

- always have your audience (the reader) in mind!
- writing is **communication**, NOT **self-expression**!
- **reader-centric** attitude, not **self-centric**



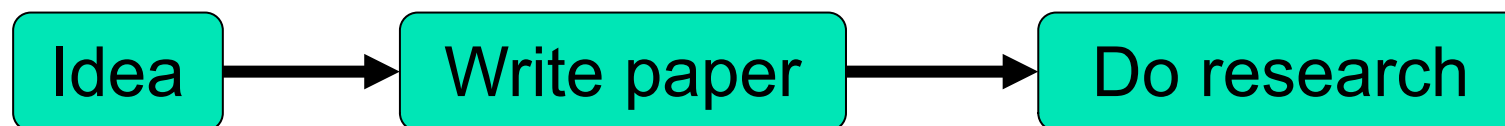
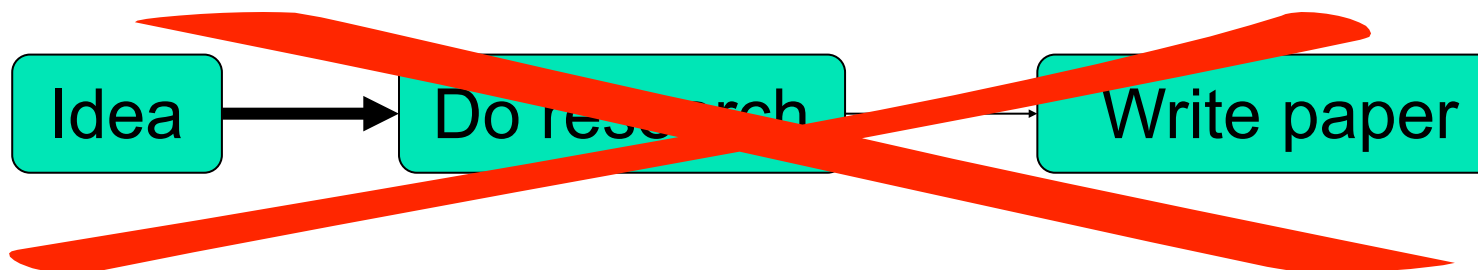


Writing papers: model 1





Writing papers: model 2



J. Eisner

- Forces us to be clear, focused
- Crystallises what we don't understand
- Opens the way to dialogue with others: reality check, critique, and collaboration



Do not be intimidated

Fallacy You need to have a fantastic idea before you can write a paper. (Everyone else seems to.)

Write a paper,
and give a talk, about
any idea,
no matter how weedy and insignificant it
may seem to you



Do not be intimidated

Write a paper, and give a talk, about any idea, no matter how insignificant it may seem to you

- Writing the paper is how you develop the idea in the first place
- It usually turns out to be more interesting and challenging than it seemed at first

LH: talk, write as early as you can;
don't wait until you feel ready;
it doesn't mean you have to publish it.

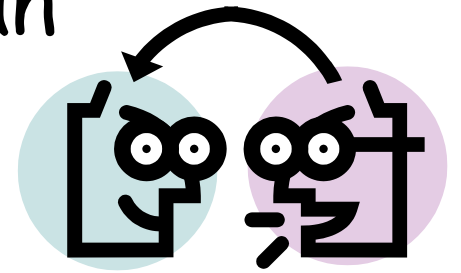


The purpose of your paper



Papers communicate ideas

- Your goal: to infect the mind of your reader with **your idea**, like a virus
- Papers are far more durable than programs (think Mozart)

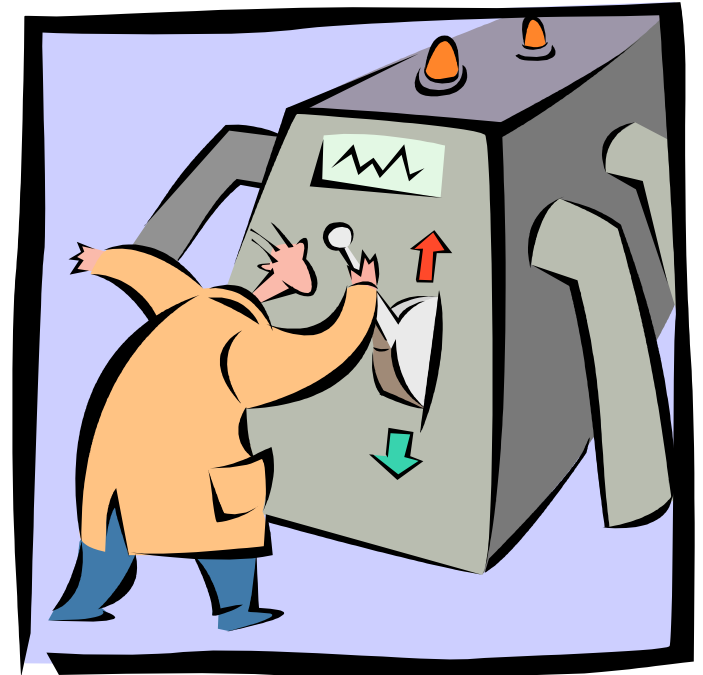


The greatest ideas are (literally)
worthless if you keep them to
yourself



The purpose of your paper is **not**...

To describe the
WizWoz
system



- Your reader does not have a WizWoz
- She is primarily interested in re-usable brain-stuff, not executable artefacts



Your narrative flow

- Here is a problem
- It's an interesting problem
- It's an unsolved problem
- **Here is my idea**
- My idea works (details, data)
- Here's how my idea compares to other people's approaches

I wish I
knew how
to solve
that!

I see how
that works.
Ingenious!





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)



The abstract

- I usually write the abstract **last**
- Used by program committee members to decide which papers to read
- Four sentences [Kent Beck]
 1. State the problem
 2. Say why it's an interesting problem
 3. Say what your solution achieves
 4. Say what follows from your solution



Example

1. Many papers are badly written and hard to understand
2. This is a pity, because their good ideas may go unappreciated
3. Following simple guidelines can dramatically improve the quality of your papers
4. Your work will be used more, and the feedback you get from others will in turn improve your research



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)



The introduction (1 page)

intro:

LH: this is the hardest part of writing!

1. Describe the problem
2. State your contributions

need to convey:
importance and hardness

...and that is all

ONE PAGE!

abstract:

1. State the problem
2. Say why it's an interesting problem
3. Say what your solution achieves
4. Say what follows from your solution

LH Method for Stating the Problem

- intro = “your *slightly* biased view of the history” [N. Dinesh]
- need to convey: importance and depth

- this is an *important* problem
- the dominant solution is good in A
- but bad in B (and B is important)
- the alternative solution is good in B
- but bad in A
- Q: how to combine their merits?? a *hard* problem!

	A	B
s1	+	-
s2	-	+
new	+	+

Abstract

Conventional n -best reranking techniques often suffer from the limited scope of the n -best list, which rules out many potentially good alternatives. We instead propose *forest reranking*, a method that reranks a packed forest of exponentially many parses. Since exact inference is intractable with non-local features, we present an approximate algorithm inspired by forest rescoring that makes discriminative training practical over the whole Treebank. Our final result, an F-score of 91.7, outperforms both 50-best and 100-best reranking baselines, and is better than any previously reported systems trained on the Treebank.

1 Introduction

Discriminative reranking has become a popular technique for many NLP problems, in particular, parsing (Collins, 2000) and machine translation (Shen et al., 2005). Typically, this method first generates a list of top- n candidates from a baseline system, and then reranks this n -best list with arbitrary features that are not computable or intractable to compute within the baseline system. But despite its apparent success, there remains a major drawback: this method suffers from the limited scope of the n -best list, which rules out many potentially good alternatives. For example 41% of the correct parses were not in the candidates of ~ 30 -best parses in (Collins, 2000). This situation becomes worse with longer sentences because the number of possible in-

	<i>local</i>	<i>non-local</i>
conventional reranking	only at the root	
DP-based discrim. parsing	exact	N/A
<i>this work: forest-reranking</i>	exact	<i>on-the-fly</i>

Table 1: Comparison of various approaches for incorporating local and non-local features.

sentence length. As a result, we often see very few variations among the n -best trees, for example, 50-best trees typically just represent a combination of 5 to 6 binary ambiguities (since $2^5 < 50 < 2^6$).

Alternatively, discriminative parsing is tractable with exact and efficient search based on dynamic programming (DP) if all features are restricted to be *local*, that is, only looking at a local window within the factored search space (Taskar et al., 2004; McDonald et al., 2005). However, we miss the benefits of non-local features that are not representable here.

Ideally, we would wish to combine the merits of both approaches, where an efficient inference algorithm could integrate both local and non-local features. Unfortunately, exact search is intractable (at least in theory) for features with unbounded scope. So we propose *forest reranking*, a technique inspired by forest rescoring (Huang and Chiang, 2007) that approximately reranks the packed forest of exponentially many parses. The key idea is to compute non-local features incrementally from bottom up, so that we can rerank the n -best subtrees at all internal nodes, instead of only at the root node as in conventional reranking (see Table 1). This method can thus



State your contributions

- Write the list of contributions first
- The list of contributions drives the entire paper: the paper substantiates the claims you have made
- Reader thinks "gosh, if they can really deliver this, that's be exciting; I'd better read on"



State your contributions



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

Bulleted list
of
contributions

Do not leave the reader
to guess what your
contributions are!



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.



No “rest of this paper is...”

- Not:

“The rest of this paper is structured as follows. Section 2 introduces the problem. Section 3 ... Finally, Section 8 concludes”.
- Instead, **use forward references from the narrative in the introduction.**

The introduction (including the contributions) should survey the whole paper, and therefore forward reference every important part.



Structure

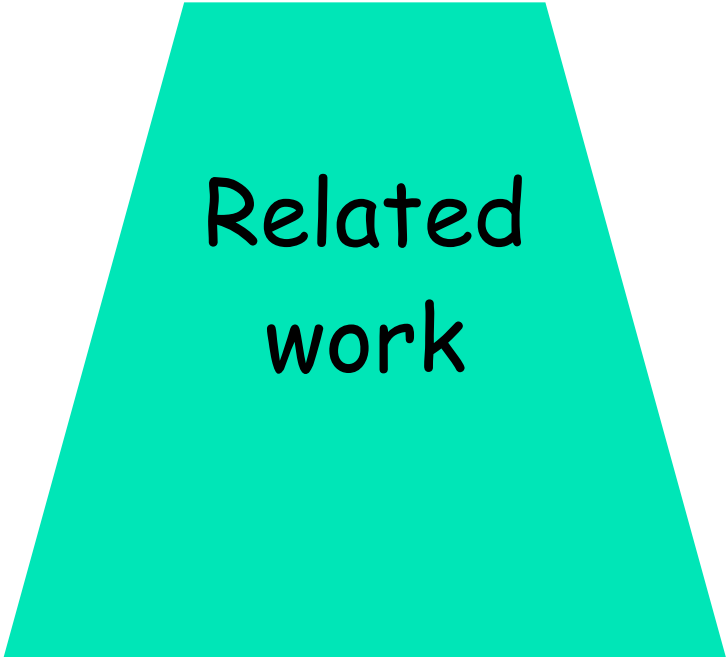
- Abstract (4 sentences)
- Introduction (1 page)
- ~~Related work~~
- The problem (1 page)
- My idea (2 pages)
- The details (5 pages)
- Related work (1-2 pages)



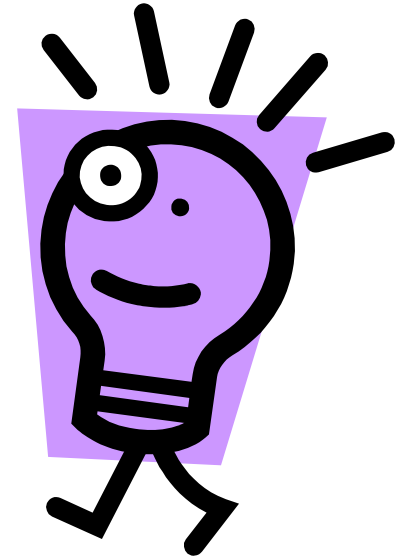
No related work yet!



Your reader



Related
work



Your idea

We adopt the notion of transaction from Brown [1], as modified for distributed systems by White [2], using the four-phase interpolation algorithm of Green [3]. Our work differs from White in our advanced revocation protocol, which deals with the case of priority inversion as described by Yellow [4].



No related work yet

- **Problem 1:** the reader knows nothing about the problem yet; so your (carefully trimmed) description of various technical tradeoffs is absolutely incomprehensible
- **Problem 2:** describing alternative approaches gets between the reader and your idea

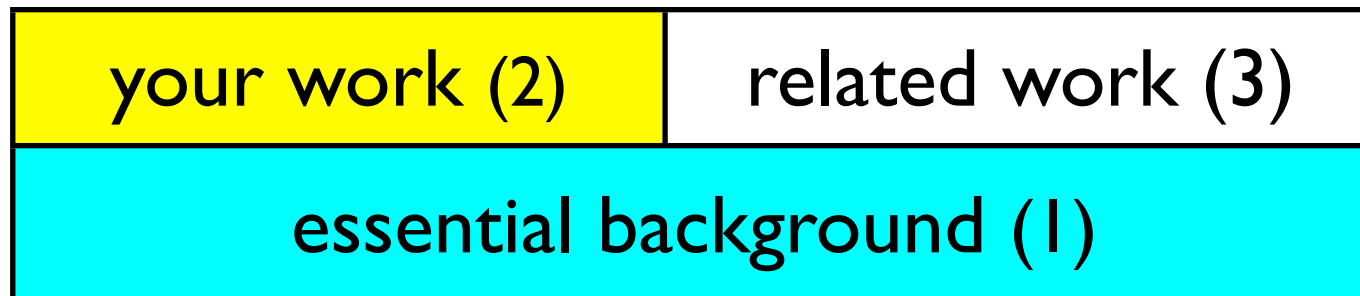
I feel
stupid



I feel
tired

LH: Two Types of Previous Work

- essential background
 - the previous work that your work builds upon
 - or improve upon (“shoulders of giants”)
 - => intro (w/o which readers can’t understand your work)
- related work: other previous work that is just related to yours
 - having them doesn’t change the understanding your work
- simple criteria: can readers understand my work w/o A?





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)



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



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

Examples and Illustrations

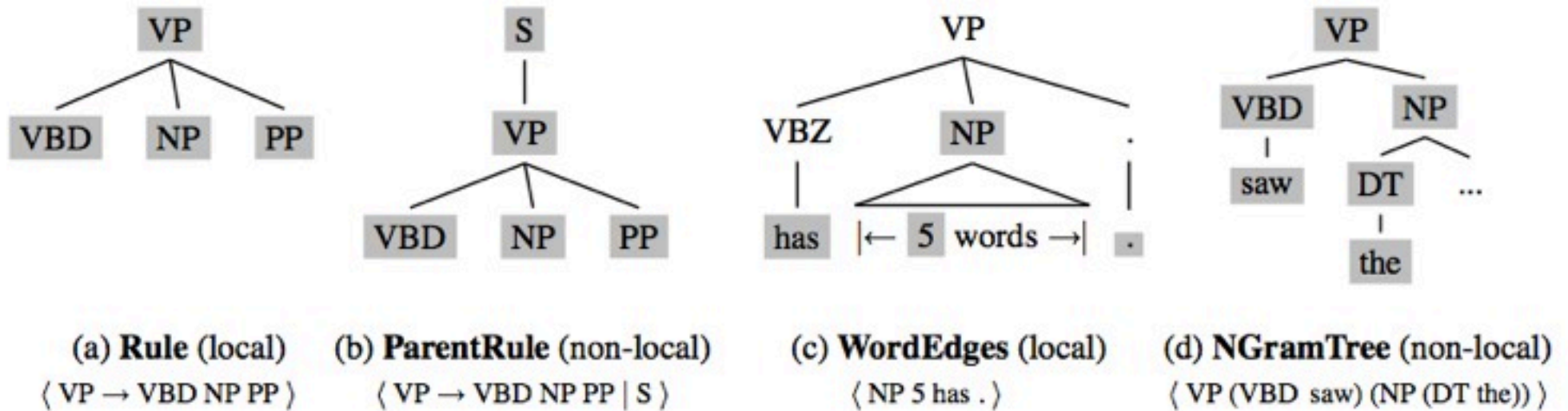


Figure 2: Illustration of some example features. Shaded nodes denote information included in the feature.

tags, which are generated dynamically.

More formally, we split the feature extractor $f = (f_1, \dots, f_d)$ into $f = (f_L; f_N)$ where f_L and f_N are the local and non-local features, respectively. For the former, we extend their domains from parses to hyperedges, where $f(e)$ returns the value of a local feature $f \in f_L$ on hyperedge e , and its value on a parse y factors across the hyperedges (local productions),

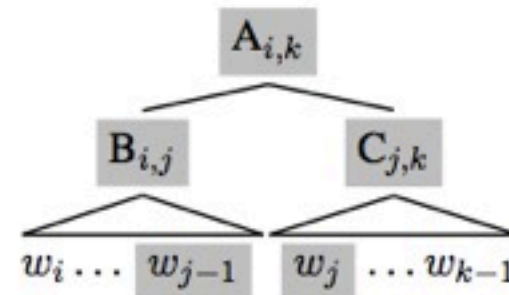


Figure 3: Example of the unit **NGramTree** feature at node $A_{i,k}$: $\langle A\ (B\ \dots\ w_{j-1})\ (C\ \dots\ w_j) \rangle$.

Examples and Illustrations

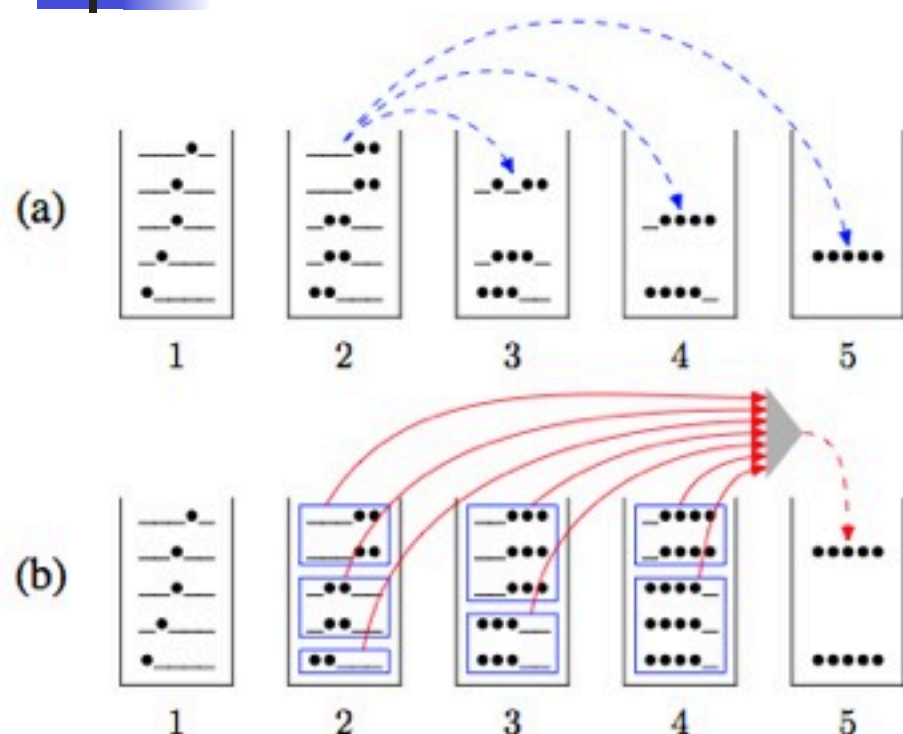


Figure 5: (a) Pharaoh expands the hypotheses in the current bin (#2) into longer ones. (b) In Cubit, hypotheses in previous bins are fed via hyperedge bundles (solid arrows) into a priority queue (shaded triangle), which empties into the current bin (#5).

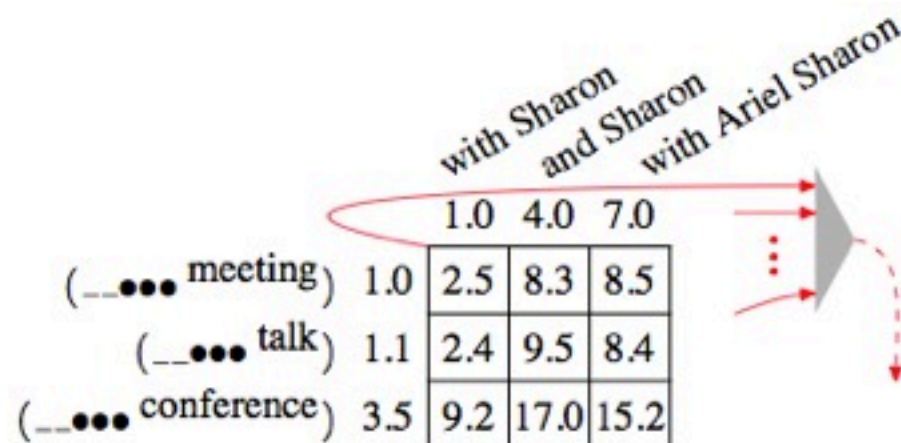


Figure 6: A hyperedge bundle represents all +LM deductions that derives an item in the current bin from the same coverage vector (see Figure 5). The phrases on the top denote the target-sides of applicable phrase-pairs sharing the same source-side.

5.1 Phrase-based Decoding

We implemented *Cubit*, a Python clone of the Pharaoh decoder (Koehn, 2004),³ and adapted cube pruning to it as follows. As in Pharaoh, each bin



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



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)



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)



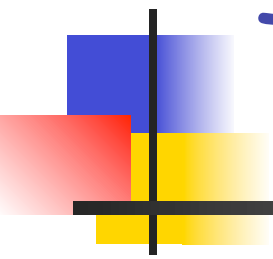
Structure

- 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)



Conclusions and further work

- Be brief.



The process of writing



The process

- Start early. Very early.
 - Hastily-written papers get rejected.
 - Papers are like wine: they need time to mature
- Collaborate
- Use *CVS* to support collaboration



Getting help

Get your paper read by as many friendly guinea pigs as possible

- Experts are good
- Non-experts are also very good
- Each reader can only read your paper for the first time once! So use them carefully
- Explain carefully what you want ("I got lost here" is much more important than "Jarva is mis-spelt".)



Listening to your reviewers

Treat every review like gold dust

Be (truly) grateful for criticism as
well as praise

This is **really, really, really** hard

But it's
really, really, really, really, really, really,
really, really, really, really
important



Listening to your reviewers

- Read every criticism as a positive suggestion for something you could explain more clearly
- DO NOT respond "you stupid person, I meant X". Fix the paper so that X is apparent even to the stupidest reader.
- Thank them warmly. They have given up their time for you.



Language and style

Visual structure

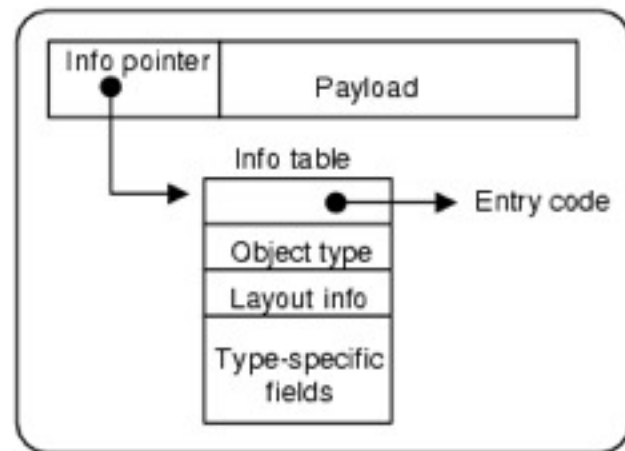


Figure 3. A heap object

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 $CON(C\ a_1 \dots a_n)$ would be represented by an object whose info pointer represented the constructor C and whose payload is the arguments $a_1 \dots a_n$.

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*

Visual Structure

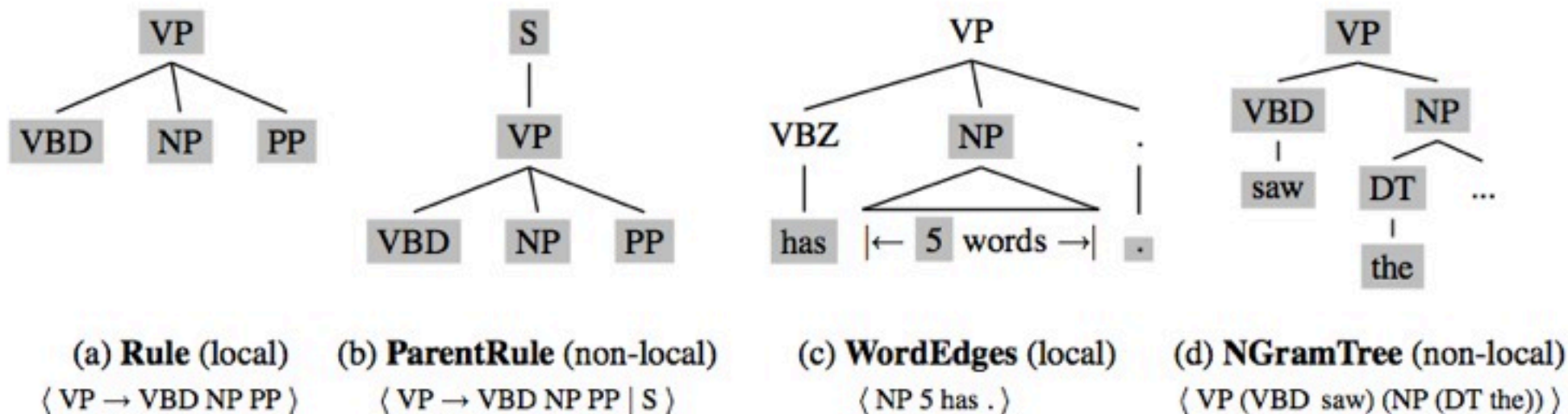


Figure 2: Illustration of some example features. Shaded nodes denote information included in the feature.

tags, which are generated dynamically.

More formally, we split the feature extractor $f = (f_1, \dots, f_d)$ into $f = (f_L; f_N)$ where f_L and f_N are the local and non-local features, respectively. For the former, we extend their domains from parses to hyperedges, where $f(e)$ returns the value of a local feature $f \in f_L$ on hyperedge e , and its value on a parse y factors across the hyperedges (local productions),

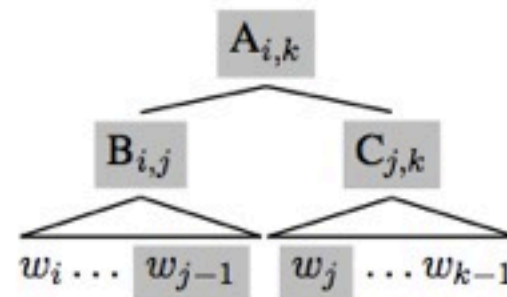


Figure 3: Example of the unit **NGramTree** feature at node $A_{i,k}$: $\langle A\ (B\ \dots\ w_{j-1})\ (C\ \dots\ w_j) \rangle$.



Use the active voice

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

NO

It can be seen that...

34 tests were run

These properties were
thought desirable

It might be thought that
this would be a type error

YES

We can see that...

We ran 34 tests

We wanted to retain these
properties

You might think this would
be a type error

"We" = you
and the
reader

"We" = the
authors

"You" =
the reader

Newton uses the active voice!

- I held the Prism.
- I looked through the Prism
- I stopt the Prism
- I observed the length of its refracted Image
- I removed the Prism out of the Sun's Light and looked



Sir Isaac Newton (1704). Optics.



Use simple, direct language

NO

The object under study was
displaced horizontally

On an annual basis

Endeavour to ascertain

It could be considered that the
speed of storage reclamation
left something to be desired

YES

The ball moved sideways

Yearly

Find out

The garbage collector was really
slow

Resources for the Writing Part

- writing resources: <http://www.cis.upenn.edu/~lhuang3/writing/>
- high-level (language-independent)
 - Simon Peyton-Jones: *How to Write a Research Paper*
 - Mark-Jan Nederhof: *Common Pitfalls in Academic Writing*
- low-level (language-specific -- use NLP!)
 - Gopen & Swan: *The Science of Scientific Writing*
 - Williams: *STYLE: Clarity and Grace* series
 - Strunk and White: *The Elements of Style*
 - Cook: *Line by Line*





How to give a good research talk

Simon Peyton Jones

Microsoft Research, Cambridge

1993 paper joint with
John Hughes (Chalmers),
John Launchbury (Oregon Graduate Institute)



How to give a good research talk

Simon Peyton Jones

Microsoft Research, Cambridge

1993 paper joint with
John Hughes (Chalmers),
John Launchbury (Oregon Graduate Institute)



Do it! Do it! Do it!

Good papers and talks are a fundamental part of research excellence

- Invest time
- Learn skills
- Practice

Write a paper, and give a talk, about
any idea,
no matter how weedy and insignificant it
may seem to you



Giving a good talk

This presentation is about how to give a good research talk

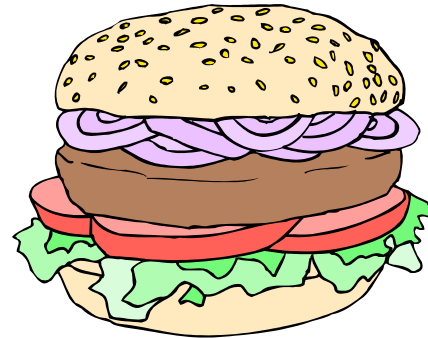
- What your talk is for
- What to put in it (and what not to)
- How to present it





What your talk is for

Your paper = **The beef**



Your talk = **The beef
advertisement**



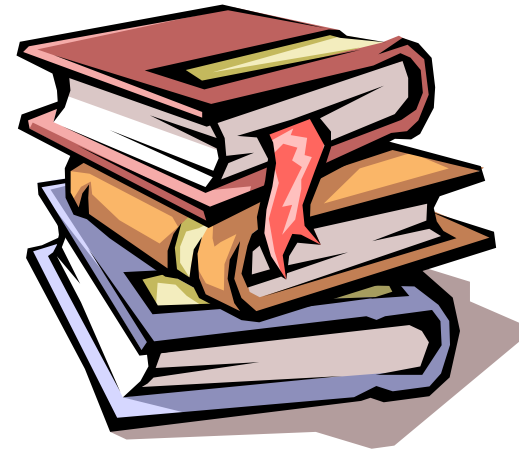
Do not confuse the two



The purpose of your talk...

..is **not**:

- To impress your audience with your brainpower
- To tell them all you know about your topic
- To present all the technical details





The purpose of your talk...

..but is:

- To give your audience an intuitive feel for your idea
- To make them foam at the mouth with eagerness to read your paper
- To engage, excite, provoke them





Your ideal audience...



The audience you would like

- Have read all your earlier papers
- Thoroughly understand all the relevant theory of cartesian closed monoidal categories and monoidal bicategories
- Are all agreed to hear about the latest developments in your work
- Are fresh, alert, and ready for action



Your **actual** audience...

The audience you get

- Have never heard of you
- Have heard of bifunctors, but wish
- Have just had lunch and are ready



Your mission is to

WAKE THEM UP

And make them glad they did



What to put in





What to put in

1. Motivation (20%)
2. Your key idea (80%)
3. There is no 3



Motivation

You have 2 minutes to engage your audience before they start to doze

- Why should I tune into this talk?
- What is the problem?
- Why is it an interesting problem?

Example: Java class files are large (brief figures), and get sent over the network. Can we use language-aware compression to shrink them?

Example: synchronisation errors in concurrent programs are a nightmare to find. I'm going to show you a type system that finds many such errors at compile time.



Your key idea

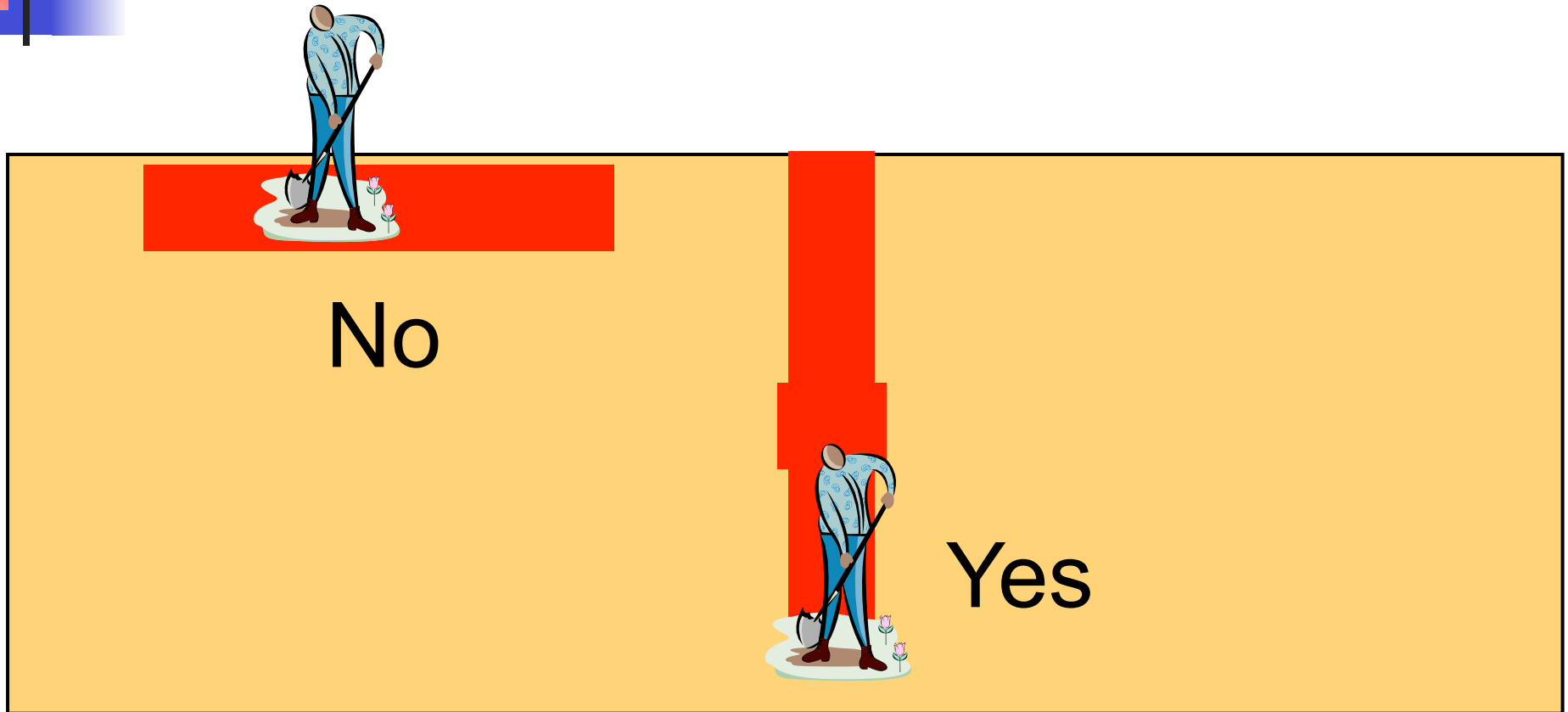
If the audience remembers only one thing from your talk, what should it be?

- **You must identify a key idea.** "What I did this summer" is No Good.
- Be specific. Don't leave your audience to figure it out for themselves.
- Be absolutely specific. Say "If you remember nothing else, remember this."
- Organise your talk around this specific goal. Ruthlessly prune material that is irrelevant to this goal.





Narrow, deep beats wide, shallow



Avoid shallow overviews at all costs
Cut to the chase: the technical “meat”



Your main weapon 1

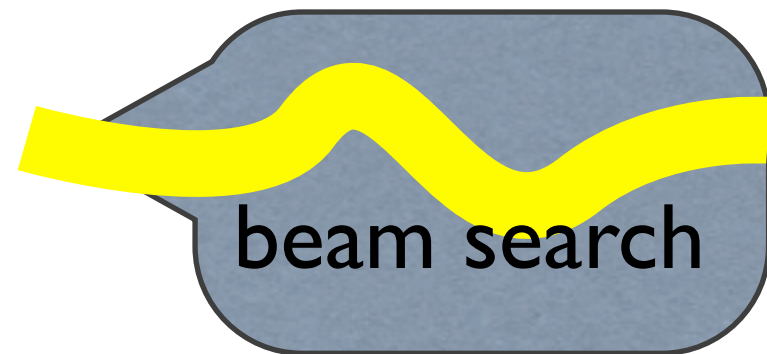
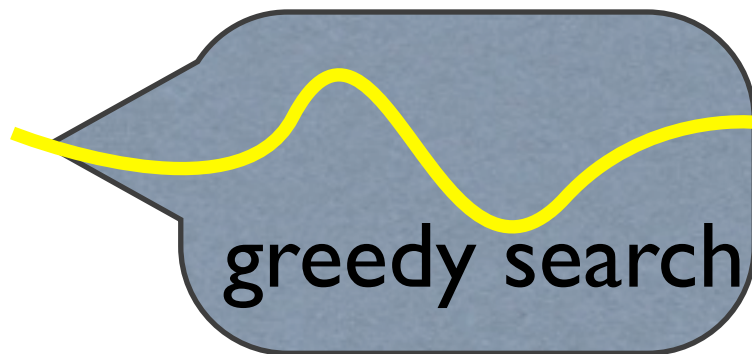
Examples are your main weapon

- To motivate the work
 - To convey the basic intuition
 - To illustrate The Idea in action
 - To show extreme cases
 - To highlight shortcomings
- When time is short, omit the general case,
not the example

LH: Your main weapon #2

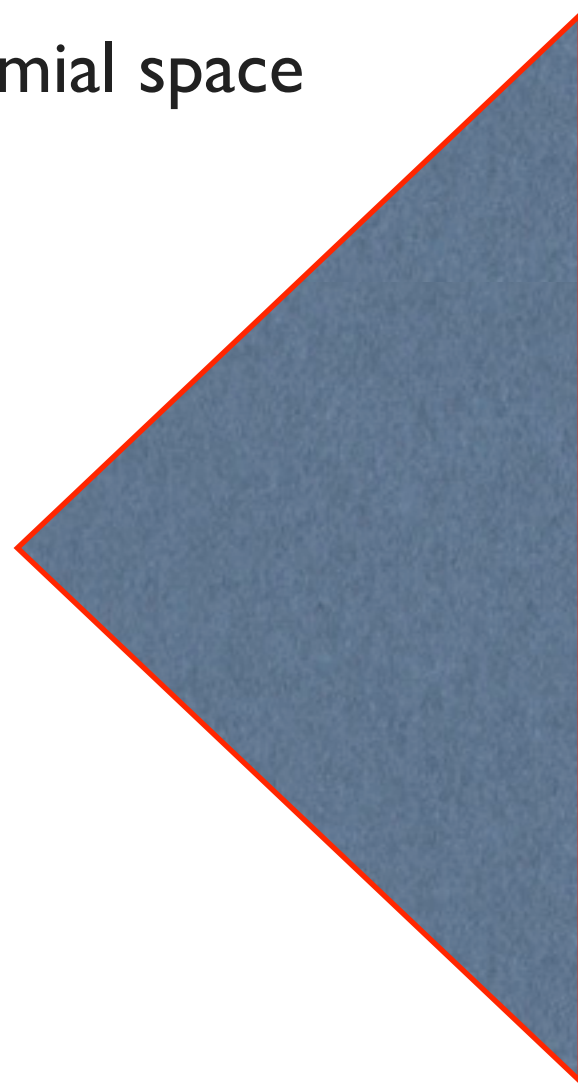
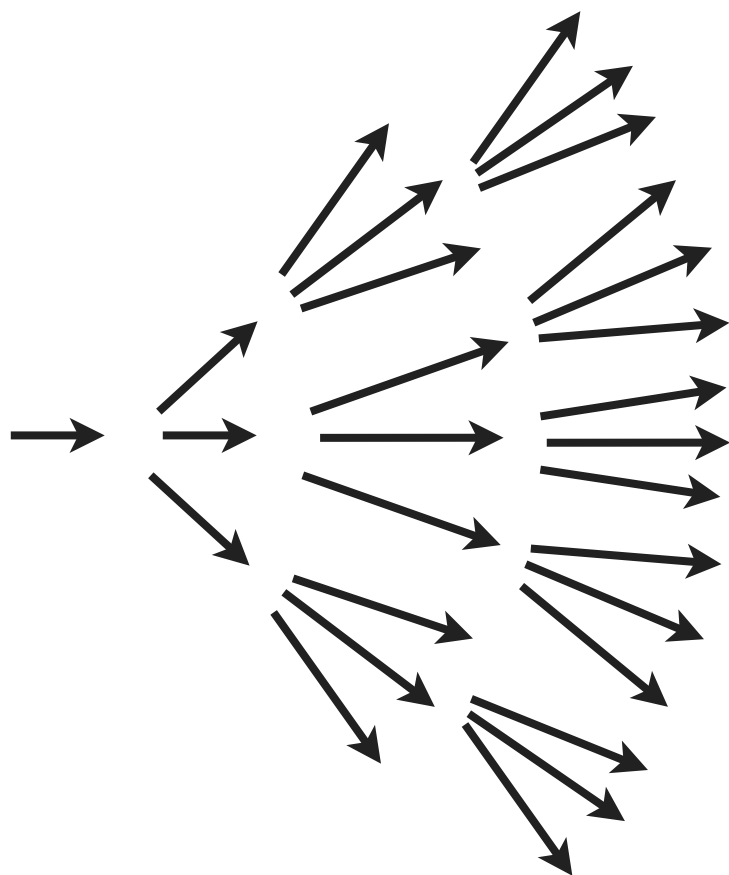
Visualization!

**a picture is worth a
thousand words!**



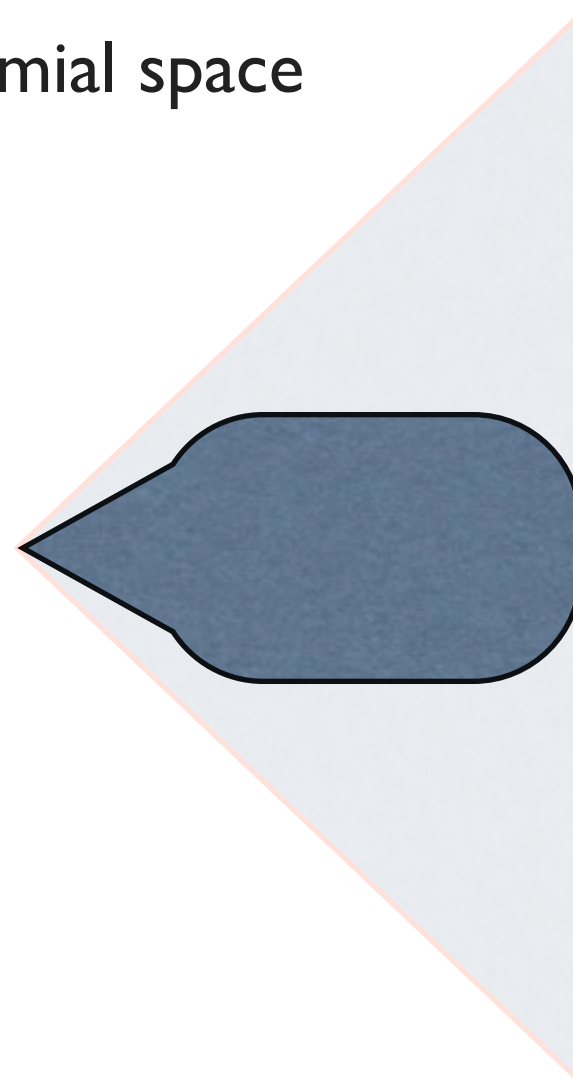
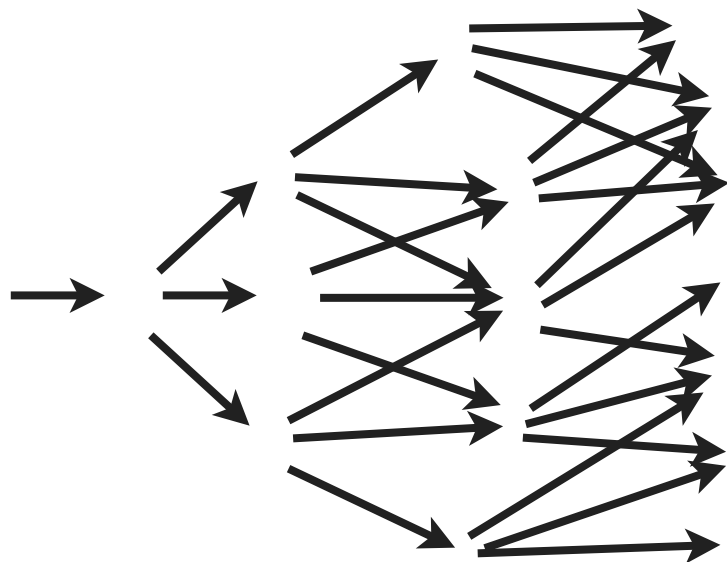
Example: Dynamic Programming

- each state \Rightarrow three new states (shift, l-reduce, r-reduce)
- key idea of DP: **share** common subproblems
 - merge equivalent states \Rightarrow polynomial space



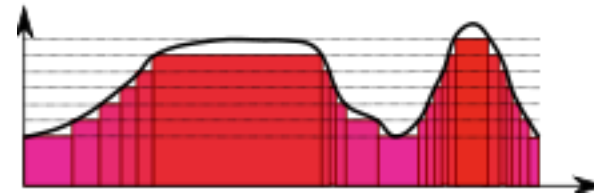
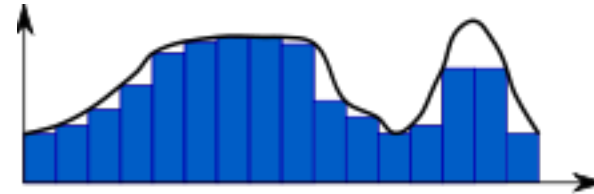
Example: Dynamic Programming

- each state \Rightarrow three new states (shift, l-reduce, r-reduce)
- key idea of DP: **share** common subproblems
 - merge equivalent states \Rightarrow polynomial space



Real Life Analogy: Lebesgue Integral

- Riemann Integral (Newton-Leibniz Style)
 - intuitive, but left many important functions unintegrable



- Lebesgue Integral

- greatly extended the domain of integrable functions



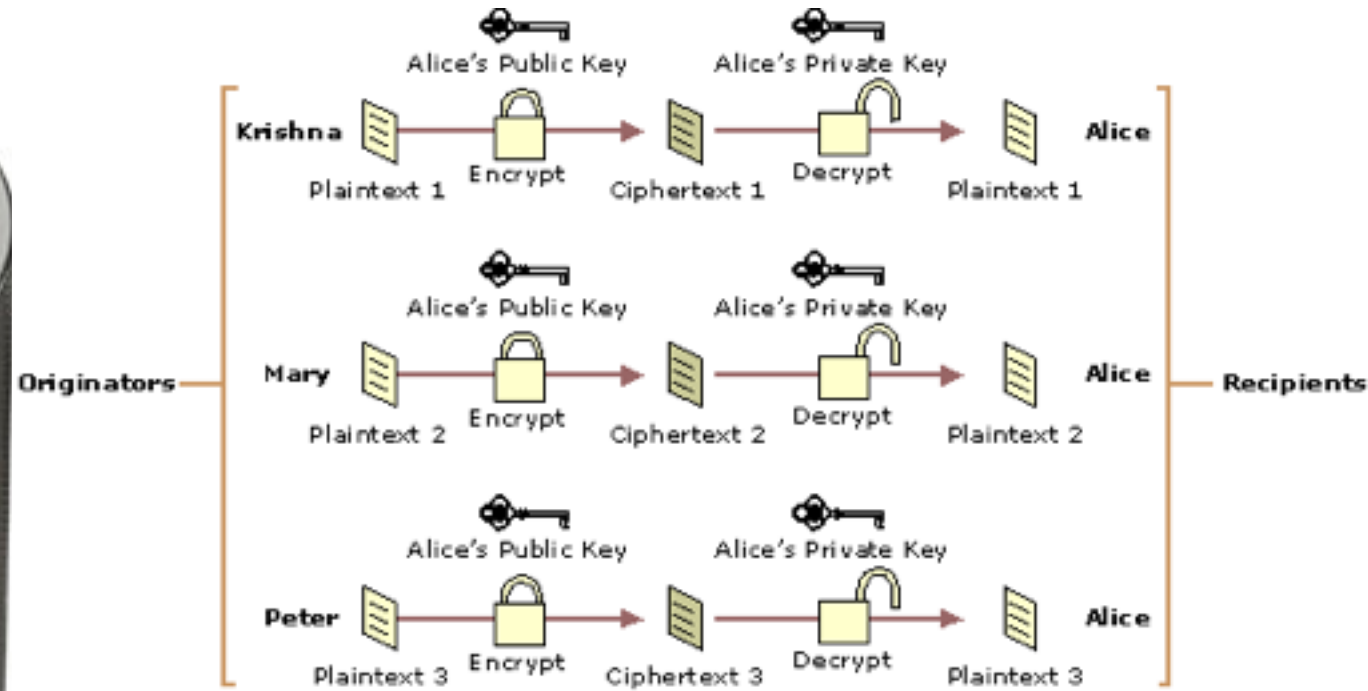
- Real Life Analogy



Lebesgue to [Paul Montel](#):

I have to pay a certain sum, which I have collected in my pocket. I take the bills and coins out of my pocket and give them to the creditor in the order I find them until I have reached the total sum. This is the Riemann integral. But I can proceed differently. After I have taken all the money out of my pocket I order the bills and coins according to identical values and then I pay the several heaps one after the other to the creditor. This is my integral. —Source: ([Siegmond-Schultze 2008](#))

Real Life Analogy: Public-Key





What to leave out



~~Outline of my talk~~

- Background
- The FLUGOL system
- Shortcomings of FLUGOL
- Overview of synthetic epimorphisms
- π -reducible decidability of the pseudo-curried fragment under the Snezkowski invariant in FLUGOL
- Benchmark results
- Related work
- Conclusions and further work





No outline!

“Outline of my talk”: conveys near zero information at the start of your talk

- But you can put up an outline for orientation **after your motivation**
- ...and signposts at pause points



~~Related work~~

- [PMW83] The seminal paper
- [SPZ88] First use of epimorphisms
- [PN93] Application of epimorphisms to wibblification
- [BXX98] Lacks full abstraction
- [XXB99] Only runs on Sparc, no integration with GUI



Do not present related work

But

- You absolutely must know the related work; respond readily to questions
- Acknowledge co-authors (title slide), and precursors (as you go along)
- Do not disparage the opposition
 - X's very interesting work does Y; I have extended it to do Z

Technical detail

$$\begin{array}{c}
 \frac{}{\Gamma \vdash k : \tau_k} \quad \frac{\Gamma \cup \{x : \tau\} \vdash e : \tau'}{\Gamma \vdash \lambda x. e : \tau \rightarrow \tau'} \quad \frac{\Gamma \vdash e_1 : \text{ST } \tau^\circ \tau \quad \Gamma \vdash e_2 : \tau \rightarrow \text{ST } \tau^\circ \tau'}{\Gamma \vdash e_1 \gg e_2 : \text{ST } \tau^\circ \tau'} \\
 \\
 \frac{\Gamma \vdash e : \tau}{\Gamma \vdash \text{returnST } e : \text{ST } \tau^\circ \tau} \quad \frac{\Gamma \vdash e : \tau}{\Gamma \vdash \text{newVar } e : \text{ST } \tau^\circ (\text{MutVar } \tau^\circ \tau)} \quad \frac{\Gamma \vdash e : \text{MutVar } \tau^\circ \tau}{\Gamma \vdash \text{readVar } e : \text{ST } \tau^\circ \tau} \\
 \\
 \frac{\Gamma \vdash e_1 : \text{MutVar } \tau^\circ \tau \quad \Gamma \vdash e_2 : \tau}{\Gamma \vdash \text{writeVar } e_1 \ e_2 : \text{ST } \tau^\circ \text{Unit}} \quad \frac{}{\Gamma \cup \{x : \forall \alpha_i. \tau\} \vdash x : \tau[\tau_i / \alpha_i]} \\
 \\
 \frac{\Gamma \vdash e : \tau' \rightarrow \tau \quad \Gamma \vdash e' : \tau'}{\Gamma \vdash e \ e' : \tau} \quad \frac{\Gamma \vdash e : \text{ST } \alpha^\circ \tau \quad \alpha^\circ \notin FV(\Gamma, \tau)}{\Gamma \vdash \text{runST } e : \tau} \\
 \\
 \frac{\forall j. \Gamma \cup \{x_i : \tau_i\}_i \vdash e_j : \tau_j \quad \Gamma \cup \{x_i : \forall \alpha_{j_i}. \tau_i\}_i \vdash e' : \tau'}{\Gamma \vdash \text{let } \{x_i = e_i\}_i \text{ in } e' : \tau'} \quad \alpha_{j_i} \in FV(\tau_i) - FV(\Gamma)
 \end{array}$$

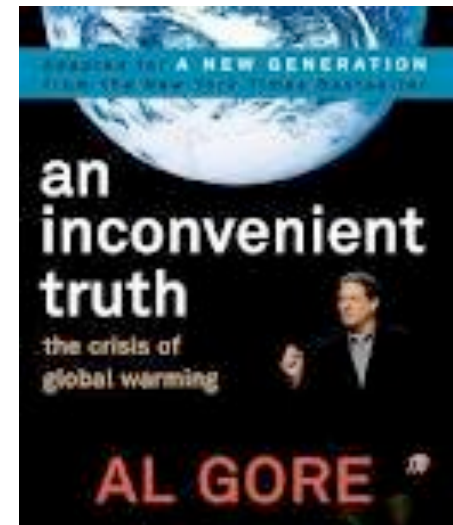
Figure 1. Typing Rules



Omit technical details



- Even though every line is **drenched** in your **blood** and **sweat**, dense clouds of notation will send your audience to sleep
- Present specific aspects only; refer to the paper for the details
- By all means have backup slides to use in response to questions





Presenting your talk





Polish your slides the night before

(...or at least, polish it then)

Your talk absolutely must be fresh in your mind

- Ideas will occur to you during the conference, as you obsess on your talk during other people's presentations
- Do not use typeset slides, unless you have a laptop too
- Handwritten slides are fine
 - Use permanent ink
 - Get an eraser: toothpaste does not work



How to present your talk

By far the most important thing is to

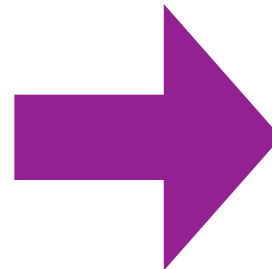
be enthusiastic





Enthusiasm

- If you do not seem excited by your idea, why should the audience be?
- It wakes 'em up
- Enthusiasm makes people dramatically more receptive
- It gets you loosened up, breathing, moving around



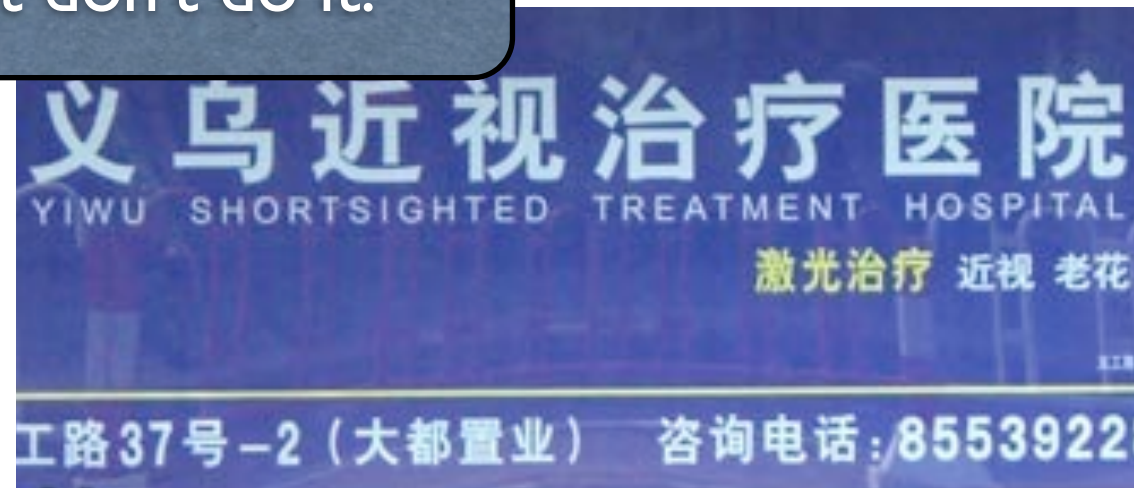
LH: Be Fun -- Three Jokes Rule

- Include as many **relevant** jokes as possible
- three jokes rule
 - one at the beginning (motivation)
 - one at the middle (to wake up people)
 - and one at the end
- especially important in job talks!

Relevant Jokes: Translation Errors



liang's rule: if you see
“**X carefully**” in China,
just don't do it.



Relevant Jokes: Translation Errors



Relevant Jokes: Translation Errors



clear evidence that MT is used in real life.

LH: Use a wireless presenter

- wireless click + laser pointer + [USB disk]
- smoothes your transition!



Face the audience

- avoid looking and pointing at your laptop
- look at the audience (70%), screen (25%), and laptop (5%)
- where is the place to stand as a speaker?





Finishing

Absolutely without fail,
finish on time

- Audiences get restive and essentially **stop listening** when your time is up. Continuing is very counter productive
- Simply truncate and conclude
- Do **not** say "would you like me to go on?" (it's hard to say "no thanks")



There is hope



The general standard is
so low that you don't
have to be outstanding
to stand out

You will attend 50x as many talks as you give.
Watch other people's talks intelligently, and pick up
ideas for what to do and what to avoid.

Conclusion: Technical Communication



ILLUSTRATION BY ANTHONY RUSSI



interaction

almost zero

a little or a lot

a whole lot

technical details

needed

not needed

needed

difficulty

hardest

easiest

