Modular Inference Trees for Expository Reports

Jens Mende
University of the Witwatersrand, Johannesburg, South Africa

mendej@sebs.wits.ac.za

Abstract

When people write a report that involves a complex argument towards a conclusion, they can use a design tool called the inference tree, which enables them to outline the argument, and quickly detect reasoning errors in the outline. Yet when the argument is very complex, the inference tree may spread over several pages, so that writers may often have to flip back and forth between those pages. To prevent unnecessary flipping, they can draw the tree as a hierarchy of modules, similar to a modular hierarchy of program flowcharts or structure charts, where a major module controls several minor modules. In drawing the tree, writers can adopt four principles of Computing: modularity, the criterion of minimal coupling between modules, and the methods of forward and backward chaining to draw all the modules.

Keywords: homological transfer, report writing, expository report, report outline, complex argument, inference tree

Introduction

When people write a report, they should first outline it, so as to ensure a coherent overall structure before writing the text in detail. For that purpose they can use various design tools, depending on the type of report. For instance, in the case of a descriptive report, which simply *describes* something by listing its attributes, they could use a simple tabular tool – such as the topic outline or the paragraph outline (Ruch & Crawford, 1988, pp.240-246). Yet for an expository report, which attempts to *prove* something, by presenting a substantial argument, those simple tools are inadequate. In that kind of report, the argument typically involves a complex system of inferential connections between paragraphs, and writers need more advanced tools that enable them to check for a wide variety of potential reasoning errors in and between the inferences. For instance, writers could use the branching diagram (Arnaudet & Barrett, 1984), or the inference tree (Mende, 2004c, 2004d).

Yet those more advanced tools are awkward to use when an argument involves a very complex system of inferential connections between paragraphs. So the present paper now addresses the problem of designing a very complex expository argument.

A solution was found by employing the research method called homological transfer (Mende, 1990). The HT method exploits inter-disciplinary homologies, which are structural or functional similarities between systems that are studied in different academic disciplines. Many such homologies exist, and when they do, researchers can transfer well-known principles of one discipline into another discipline where they are unknown. For that purpose, HT recommends four steps: (1) identify systems homologies; (2) recognise opportunities to transfer principles of a source field into a target field, (3) adapt the source principles to suit the target field, and (4) test the new principles in the target field.

Here, the four HT steps were carried out as follows.

1. A structural homology was identified between design problems posed by a very complex expository argument and a very complex computer program.
2. The homology suggested that Information Systems solutions could be re-used in the process of designing an expository argument.
3. Well known Information Systems principles of modularity and decoupling were transformed into guidelines for designing a hierarchy of inference trees. These guidelines were extended by including two chaining procedures developed in an earlier paper (Mende, 2004c).
4. The guidelines were tested by drawing trees of several very complex expository arguments.

The next two sections review basic concepts that were introduced in the earlier papers (Mende, 2004a+b), so that this paper can be read independently. Subsequent sections then cover the four transfer steps in detail, with emphasis on the results.

Expository Reports and Inference Trees

Like any other report, an expository report consists of paragraphs, and they are typically grouped into introductory, body and concluding sections. The introductory paragraphs tell readers what is to be proved, how it is to be proved, and why they need to know. The body and concluding paragraphs contain the expository argument that proves what is to be proved. Essentially, the argument consists of the core ideas of those paragraphs, plus inferences between the core ideas. The core idea of a paragraph is the central idea, and is surrounded by peripheral ideas such as examples, references, etc. which establish that the core idea is true. The inferences, which are recognisable by keywords such as ‘so’ and ‘therefore’, derive new core ideas from previous core ideas.

Table 1 presents a simple example of an argument that spreads over ten paragraphs. Each paragraph begins with the core idea, and is followed by peripheral ideas. (The details of the peripheral ideas are omitted, because you do not need to know whether core ideas are true in order to detect errors in the inferences between them). The ten core ideas are connected by four inferences: from 1 and 2 to 3; from 4 and 5 to 6; from 3, 6 and 7 to 8, and from 8 and 9 to 10.

The argument in Table 1 has no reasoning errors. The first two inferences are valid inductive inferences from facts to generalisations of narrow scope. The third inference is another valid inductive inference – from the narrow generalisations to a generalization of wider scope. The fourth inference is a valid analogical inference from two generalisations to another generalisation.

Yet in other arguments reasoning errors can easily occur, especially as the number of core ideas increases (Evans, 1982; Wilson, 1998, p. 208). These errors occur because the many core ideas are surrounded by very many peripheral ideas, which overload the writer’s short-term memory, causing him or her to lose track of the flow of reasoning. So in order to detect reasoning errors, writers need some means of ignoring the multitude of peripheral ideas, and focusing their attention on the core ideas plus the inferences between them.

Table 1: IT and Evolution

1.	IBM hardware evolved by selection. For example …
2.	Apple hardware evolved by selection. For example …
3.	So hardware evolves by selection.
4.	MS software evolved by selection. For example …
5.	Lotus software evolved by selection. For example …
6.	So software evolves by selection.
7.	IT includes hardware and software.
8.	So IT evolves by selection. This is confirmed by …
9.	Bio-organisms also evolve by selection. Darwin …
10.	Therefore IT evolves like bio-organisms. This insight is …

For that purpose an earlier paper (Mende, 2004c) exploited a homology between the process of designing a computer program and the process of outlining an expository argument. Programmers need to ensure that the paragraphs of a program fit together coherently, and for that purpose they can draw a flowchart or structure chart. Similarly, writers need to ensure that the core ideas of an argument fit together coherently, and for that purpose they can draw an inference tree.

The inference tree is a hybrid of the flowchart of Programming and the tree-structure of Computer Science. It outlines the gist of an expository argument, by representing the core ideas as boxes, and the inferences as arrows between the boxes. The core ideas are grouped in three columns:

1. *Premises* are core ideas that are not inferred from other core ideas of the argument. Typically, they are empirical observations or references to the findings of other publications.
2. *Intermediates* are inferred from other core ideas, and other core ideas are inferred from them.

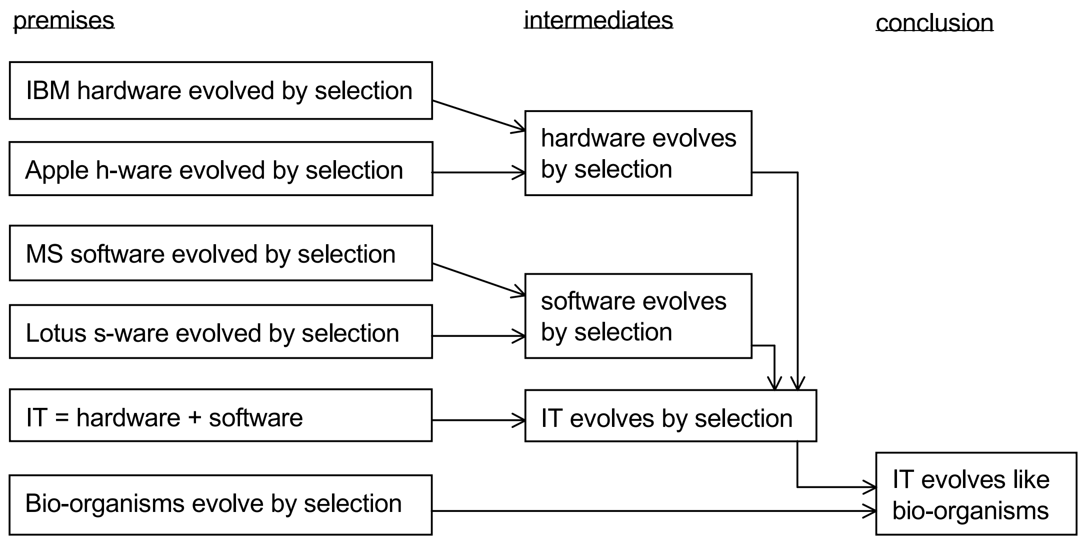
premises
intermediates
conclusion
IBM hardware evolved by selection
Apple h-ware evolved by selection
MS software evolved by selection
Lotus s-ware evolved by selection
IT = hardware + software
Bio-organisms evolve by selection
hardware evolves by selection
software evolves by selection
IT evolves by selection
IT evolves like bio-organisms

Fig. 1. Inference tree of box 1

3. *Conclusions* are inferred from other core ideas, but no other core ideas are inferred from them. Typically, an argument has only one conclusion.

The flow of reasoning is downward through the rows, and from left to right within each row.

For example, Figure 1 is the inference tree of the argument in Table 1.

Many other arguments have similar connections between their core ideas. For instance, you could get three other arguments toward the same conclusion by replacing the word 'selection' in Figure 1 with the word 'inheritance', or the word 'variation', or even the words 'inheritance, variation and selection'. All those arguments would be valid because their inference trees reveal no errors in the connections between core ideas.

Reasoning Errors

Yet many other arguments do have reasoning errors, and those errors are obvious in their inference trees (Mende, 2004d). For example, the tree in Figure 2 has two effectiveness errors that undermine the credibility of the argument, and the two trees in Figures 3 and 4 have five efficiency errors that waste a reader's time. These errors can be detected simply by inspecting the inferential arrows between boxes, or the lack of such arrows – without even knowing whether the core ideas are true, or even understanding those core ideas.

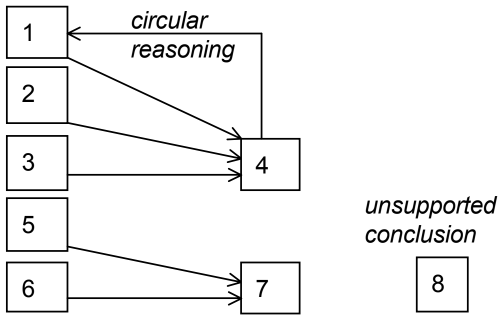

Fig. 2. Two effectiveness errors

Many different kinds of reasoning errors can occur in an argument (Mende, 2002a), and many of them can be detected in an inference tree. Reasoning *effectiveness* errors undermine the credibility of the argument.

- Presumption: the inference input presumes the output.
- Illusory relevance: an input seems to be relevant to the output, but isn't actually.
- Inadequate inference: the inference omits some input that is necessary to support the output.
- Weak induction: the inference has too few inputs to justify the output.
- Formal fallacy: the inference distorts an established rule of valid deductive inference.

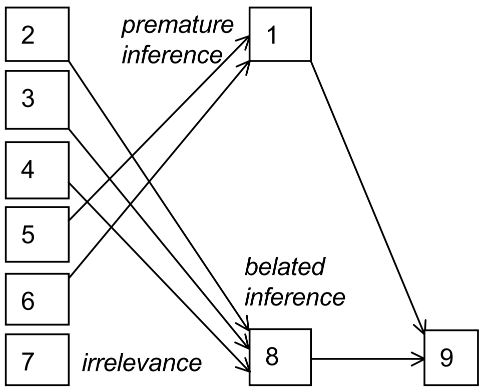

Fig. 3. Three efficiency errors

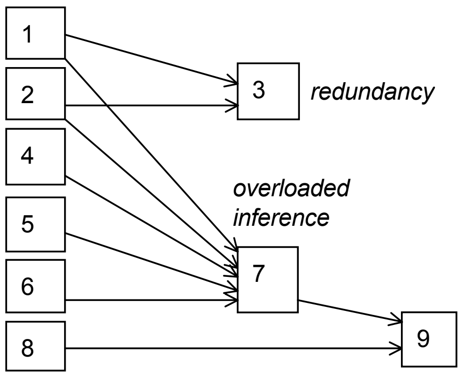

Fig. 4. Another two efficiency errors

- Mismatch: the inference output includes concepts that do not appear in any of the inputs.
- Missing premises: the argument has too few premises to justify the conclusion.
- Circular reasoning: an intermediate is merely a synonym of a premise or of the conclusion.
- Missing the point: the argument supports a conclusion other than the stated conclusion.

Efficiency errors waste a reader's time.

- Irrelevance: some of the inference inputs are not necessary to infer the output.
- Omitted inference: the inputs imply an output, but the inference output is not stated explicitly.
- Overloaded inference: an inference has unnecessarily many inputs and/or outputs, and can be split into simpler inferences which are easier to understand.
- Redundancy: parts of the argument are unnecessary to prove the conclusion.
- Belated inference: the inference output is placed late in the argument, but some of the inputs are placed early in the argument.
- Premature inference: the output is placed early in the argument, but the inputs are placed late in the argument.
- Incoherence: similar propositions are grouped together in the same section, and inferences connect those propositions not to one another but to propositions in other sections.
- Inconclusive argument: the argument has no real conclusion: the conclusion may be missing entirely, or may be trivial (e.g. 'much has been written on this topic').

So when writers want to produce a complex expository argument, they should begin by outlining it in the form of an inference tree, and then use the tree to check for reasoning errors, in order to eliminate them before writing the argument in detail.

Very Complex Arguments

Yet in real life, many an expository argument is very much more complex than Table 1. So reasoning errors are even more likely to occur, and an inference tree would be even more useful for error-checking. Yet in such cases the tree would be very much larger than Figure 1, and would spread over several A4 pages. So it would be awkward to draw, and to check, because the writer must continually flip back and forth among the various pages.

To address this problem, HT step 1 identified a homology between the process of designing a very complex expository argument and the process of designing a very complex computer program. If a computer program is very complex, its flowchart also spreads over several pages, making it awkward to draw and to check. So programmers and writers face similar problems.

HT step 2 then exploited this homology by recognised an opportunity for knowledge transfer from Information Systems to Report Writing. The similarity between problems suggested that IS solutions could be re-used in the process of designing an expository argument.

To solve the flowcharting problem, programmers long ago evolved the technique of modularisation (Yourdon, 1975, pp. 93-130), which lets an overall control module direct the execution of several detailed processing modules. So instead of one large flowchart, programmers draw a hierarchy of small flowcharts: a high-level flowchart THAT outlines the control module, and several low-level flowcharts THAT outline the processing modules. The high-level flowchart is connected to the low-level flowcharts, but the low-level flowcharts are largely independent of each other. This arrangement led to the 'structure chart' (Jackson, 1975), which represents any program as a tree structure such as Figure 5.

HT step 3 showed that writers can also use the modularity principle to decompose a very complex expository argument. They can divide the argument into a concluding argument at the end of the report, and several supporting arguments in the body sections of the report. A high-level tree outlines the concluding argument, and several low-level trees outline the body arguments. The high-level tree is connected to the low-level trees, but the low-level trees are not connected to one another, and can be drawn (and checked) independently.

For example, the three low-level trees in Figures 6 to 8, plus the high-level tree in Figure 9, outline a much-extended form of the argument of Table 1. The premises consist of historic evidence of the three mechanisms of Inheritance, Variation and Selection (IVS). The three low-level trees outline low-level arguments from the premises towards low-level conclusions that computers evolved by IVS, that networks evolved by IVS, and that software evolved by IVS. Then the high-level tree outlines the concluding argument from the three low-level conclusions through the intermediate conclusion that IT evolved by IVS to the final high-level conclusion that IT evolves like bio-organisms.

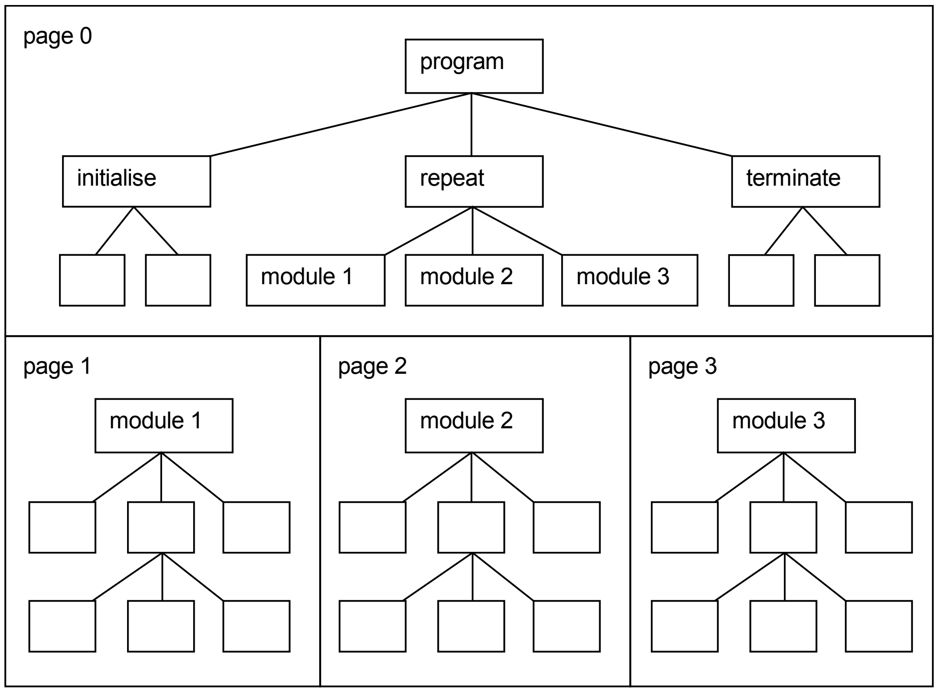

Fig. 5. Modular program structure chart

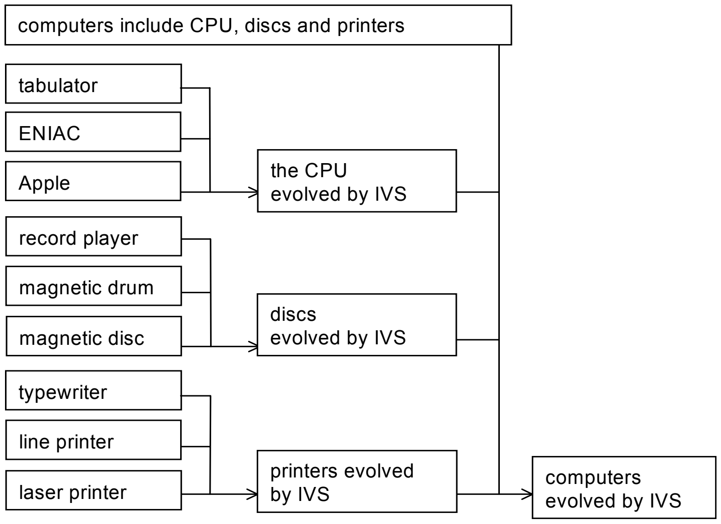

Fig. 6. Low-level inference tree on the evolution of computers

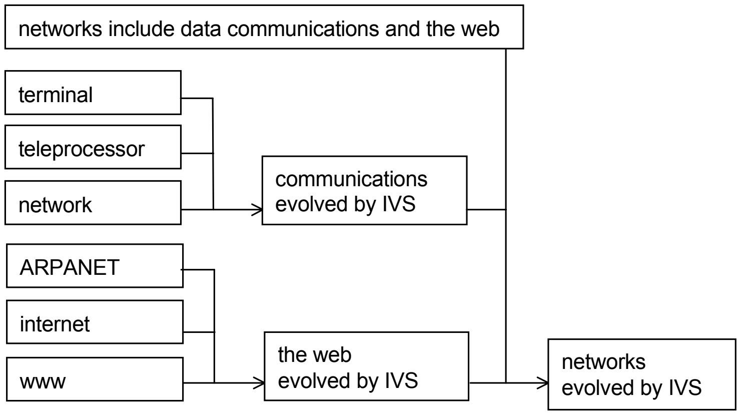

Fig. 7. Low-level inference tree on the evolution of networks

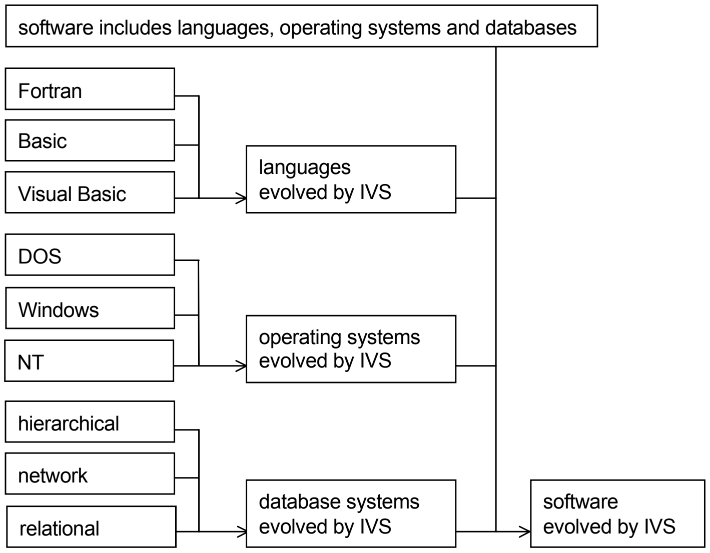
software includes languages, operating systems and databases
Fortran
Basic
Visual Basic
languages evolved by IVS
DOS
Windows
NT
operating systems evolved by IVS
hierarchical
network
relational
database systems evolved by IVS
software evolved by IVS

Fig. 8. Low-level inference tree on the evolution of software

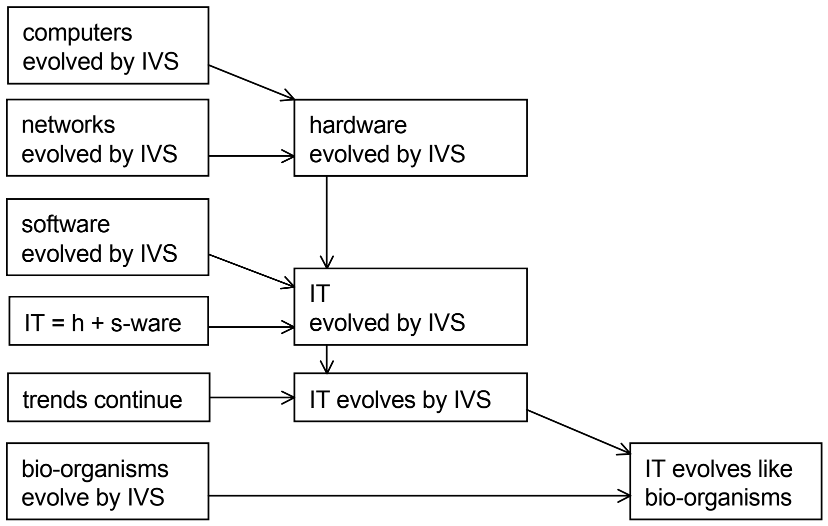

Fig. 9. High-level inference tree on the evolution of IT

The four trees enable you to check for reasoning errors, and eliminate them before writing the report in detail. Then the trees would serve as useful blueprints for detailed report writing. Therefore writers can gain substantial benefits by adopting the modularity principle to outline a very complex expository argument in the form of a hierarchy of inference trees. But *how* should they proceed to draw the hierarchy of inference trees?

Design Procedures

An answer was found by extending a pair of procedures that were recommended in an earlier paper (Mende, 2004b). They are manual procedures that emulate the forward and backward chaining algorithms of Artificial Intelligence.

The *forward chaining* procedure lets writers start with the facts, and design low-level trees of minor arguments from the facts towards low-level conclusions. Then they can use the conclusions of the low-level trees as premises in designing a high-level tree of the major argument towards the final conclusion.

For example, the low-level tree of Figure 8 could be designed in five steps.

1. Focus on historic cases that involve software.
2. Select the cases of evolving languages, and infer that languages evolved by IVS.
3. Select the cases of operating systems, and infer that operating systems evolved by IVS.
4. Select the cases of database systems, and infer that database systems evolved by IVS.
5. From the results of steps 2-4, conclude that software evolved by IVS.

Then, the high-level tree of Figure 9 could be designed by using the conclusions of the low-level trees 6-8 as premises, and combining them into wider generalisations until the desired conclusion emerges.

1. Select the premises ‘computers evolved by IVS’ and ‘networks evolved by IVS’ from the conclusions of Figures 6 and 7, and combine them into ‘hardware evolved by IVS’.
2. Add the premise ‘software evolved by IVS’ from the conclusion of Figure 8, and together with the premise ‘IT = h-ware + s-ware’, combine them into ‘IT evolved by IVS’.
3. From the additional premise ‘trends continue’, infer ‘IT evolves by IVS’.
4. From the additional premise ‘bio-organisms evolve by IVS’, infer that ‘IT evolves like bio-organisms’.

The *backward chaining* procedure lets writers begin by hypothesising the desired conclusion, and design a high-level tree from that hypothesis towards intermediate hypotheses and finally facts. For example, the high-level Figure 9 could be designed in five steps.

1. Hypothesise the desired conclusion ‘IT evolves like bio-organisms’.
2. From the fact ‘bio-organisms evolve by IVS’, derive the new hypothesis ‘IT evolves by IVS’.
3. From the assumption ‘trends continue’, derive the new hypothesis ‘IT evolved by IVS’.
4. Divide the new hypothesis into hardware and software hypotheses.
5. Divide the hardware hypothesis into computer and network hypotheses.

Then the three low-level trees of Figures 6-8 could be designed by selecting the intermediate hypotheses about computers, networks and software, one by one, and working backward to the relevant historic facts. For example, Figure 8 would be designed in three steps.

1. Get the intermediate hypothesise ‘software evolved by IVS’ from the high-level tree.
2. From the fact that software includes languages, operating systems and databases, derive the new hypotheses ‘languages evolved by IVS’, ‘operating systems evolved by IVS’, and ‘databases evolved by IVS’.
3. For each of these hypotheses, adduce evidence from historical cases.

Therefore writers can easily design a modular hierarchy of inference trees by using forward or backward chaining (or a combination of the two). But a problem now arises, because writers can use the chaining procedures to produce several alternative argument hierarchies from identical premises towards the identical conclusion – and some of these hierarchies are less efficient than others.

Avoiding Inefficient Designs

An argument hierarchy is inefficient if an alternative hierarchy would reach the same conclusion with fewer paragraphs, sentences and words. In the absence of appropriate guidelines, writers can easily produce an inefficient hierarchy. For example, when sixty third-year university students were asked to prove that IT evolves like bio-organisms, almost all of them chose an inefficient hierarchy, similar to the one below. Its high-level tree is Figure 10; its first low-level tree is Figure 11, and the other two low-level trees are similar, with the word ‘inheritance’ merely being

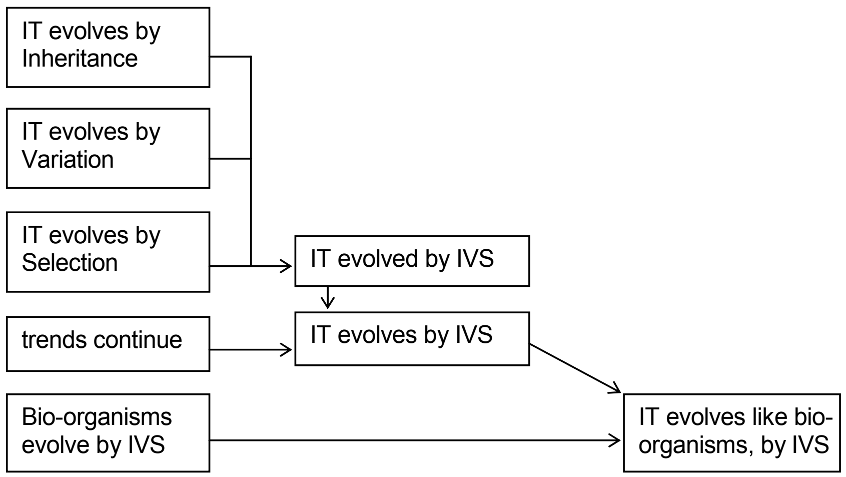

Fig. 10. Inefficient high-level tree for the evolution of IT

replaced by ‘variation’ and ‘selection’ respectively.

This arrangement is inefficient because the case evidence is distributed over several low-level trees, which requires that each tree must involve repeated references to all the cases. So when a reader encounters ENIAC on page 2 of the resulting report, under the heading Inheritance, and then encounters ENIAC again on page 8 under the heading Variation, he would have forgotten important historical details that were mentioned on page 2.

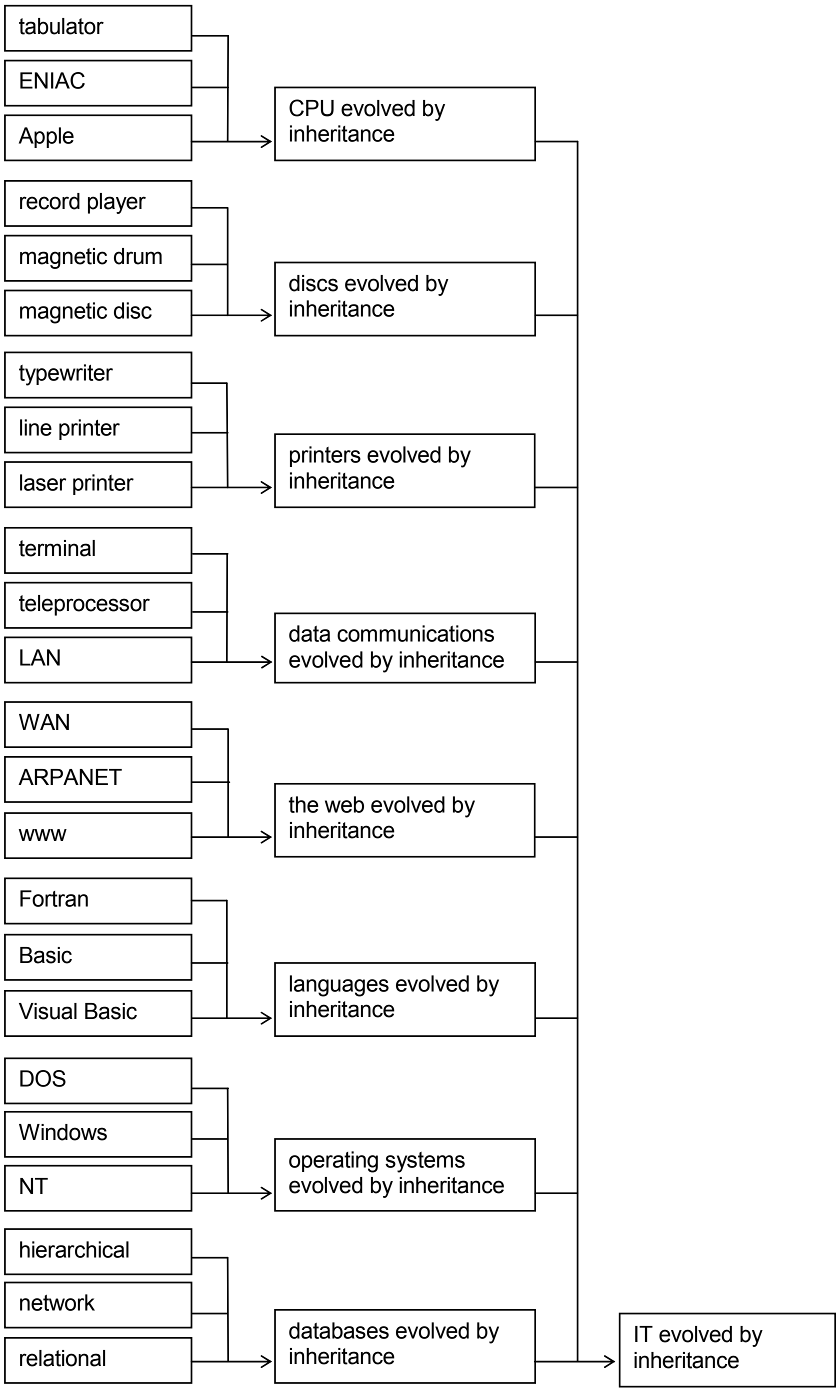
tabulator
ENIAC
Apple
CPU evolved by inheritance
record player
magnetic drum
magnetic disc
discs evolved by inheritance
typewriter
line printer
laser printer
printers evolved by inheritance
terminal
teleprocessor
LAN
data communications evolved by inheritance
WAN
ARPANET
www
the web evolved by inheritance
Fortran
Basic
Visual Basic
languages evolved by inheritance
DOS
Windows
NT
operating systems evolved by inheritance
hierarchical
network
relational
databases evolved by inheritance
IT evolved by inheritance

Fig. 11. Inefficient low-level tree for IT inheritance

To provide a simple guideline for avoiding efficient hierarchies, HT step 3 adapted the classic decoupling rule of Structured Information System Design (Yourdon & Constantine, 1979, chap. 6 & 7):

> group strongly coupled components together into the same module,
> and separate weakly coupled components in different modules.

This is applicable in Report Writing because core ideas would be strongly coupled if they are directly connected by inferences or if they are supported by the same peripheral ideas. Conversely, they would be weakly coupled if there are no inferences between them, or if their peripheral ideas are different.

If writers had considered coupling in the inefficient IT evolution example of Figures 10 and 11, they might have seen that the evidence of inheritance, variation and selection is very strongly coupled within each individual technology, and strongly coupled within each technological class; but that technological classes are weakly coupled, and individual technologies are very weakly coupled. So the decoupling rule demands that each low-level tree should be premised exclusively on the strongly coupled IVS evidence within one selected technology or technological class, and should omit weakly coupled evidence from the other technologies or technological classes. Thus the three low-level trees should prove that computers have evolved by IVS, networks have evolved by IVS and software has evolved by IVS, leaving the high-level tree to combine those generalisations to the conclusion that IT evolves by IVS – as in the efficient IT evolution example of Figures 6 - 9.

Therefore when writers draw a hierarchy of inference trees, they should use not only the modularity principle and the chaining procedures but also the decoupling rule.

Another Example

The last recommendation was based on inductive and deductive arguments in the previous sections. Yet ‘the proof of the pudding is in the eating’. So HT step 4 was necessary to provide empirical evidence that modular inference trees really work, by applying them to several examples of very complex expository arguments. One of those examples was presented above – name

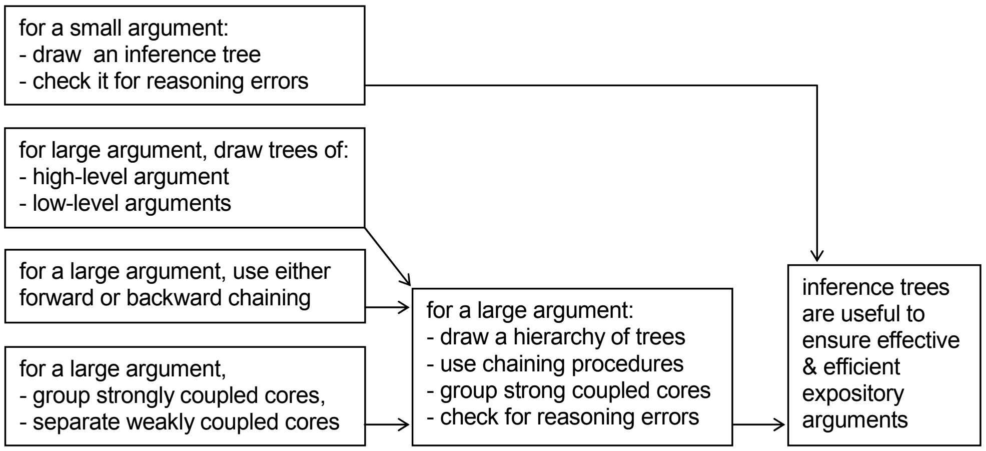
for a small argument:
- draw an inference tree
- check it for reasoning errors
for large argument, draw trees of:
- high-level argument
- low-level arguments
for a large argument, use either forward or backward chaining
for a large argument,
- group strongly coupled cores,
- separate weakly coupled cores
for a large argument:
- draw a hierarchy of trees
- use chaining procedures
- group strong coupled cores
- check for reasoning errors
inference trees are useful to ensure effective & efficient expository arguments

Figure 12: High-level tree for Modular inference trees.

ly the case of IT evolution. Another example is presented below. It outlines the argument towards modular inference trees in the present paper. Its modular hierarchy is shown in Figures 12-16**.** The trees were drawn after a first draft of the paper had been written, not beforehand, as recommended above (because the author had only a vague idea of what to prove). Nevertheless they revealed dozens of reasoning errors – which were then corrected, in the trees and the paper.

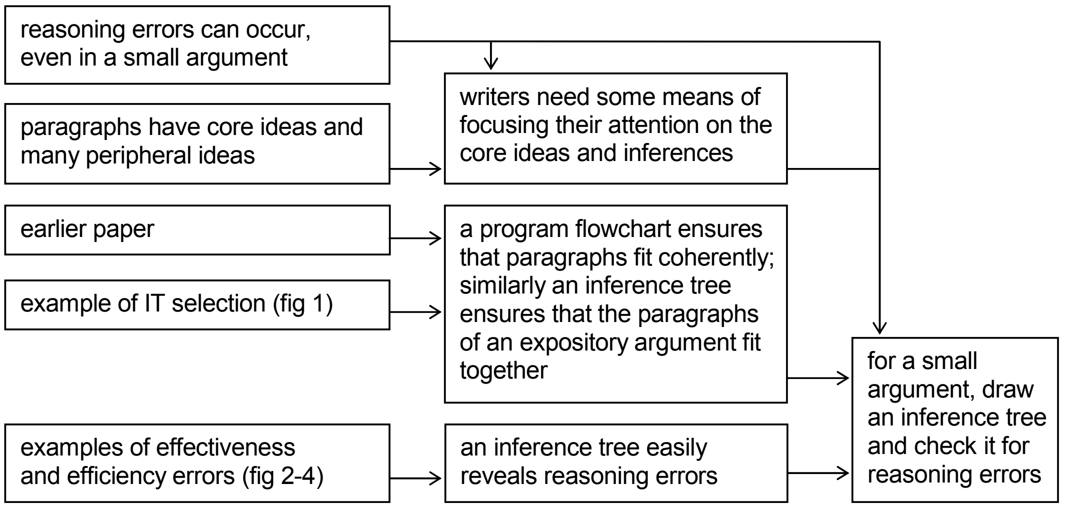
reasoning errors can occur, even in a small argument
paragraphs have core ideas and many peripheral ideas
earlier paper
example of IT selection (fig 1)
examples of effectiveness and efficiency errors (fig 2-4)
writers need some means of focusing their attention on the core ideas and inferences
a program flowchart ensures that paragraphs fit coherently; similarly an inference tree ensures that the paragraphs of an expository argument fit together
an inference tree easily reveals reasoning errors
for a small argument, draw an inference tree and check it for reasoning errors

Figure 13: Low-level tree for a small argument

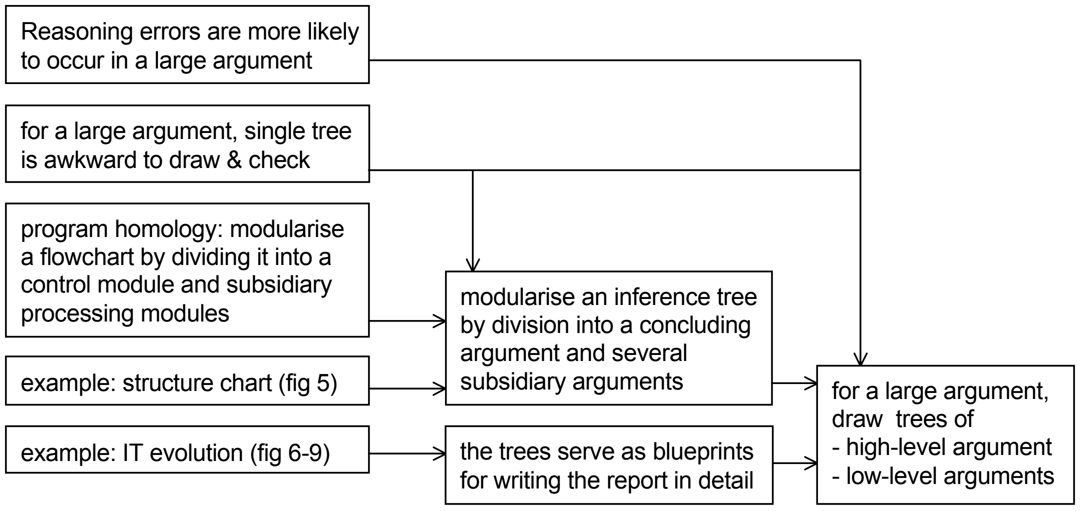
Reasoning errors are more likely to occur in a large argument
for a large argument, single tree is awkward to draw & check
program homology: modularise a flowchart by dividing it into a control module and subsidiary processing modules
example: structure chart (fig 5)
example: IT evolution (fig 6-9)
modularise an inference tree by division into a concluding argument and several subsidiary arguments
the trees serve as blueprints for writing the report in detail
for a large argument, draw trees of
- high-level argument
- low-level arguments

Figure 14: Low-level tree for a modular hierarchy

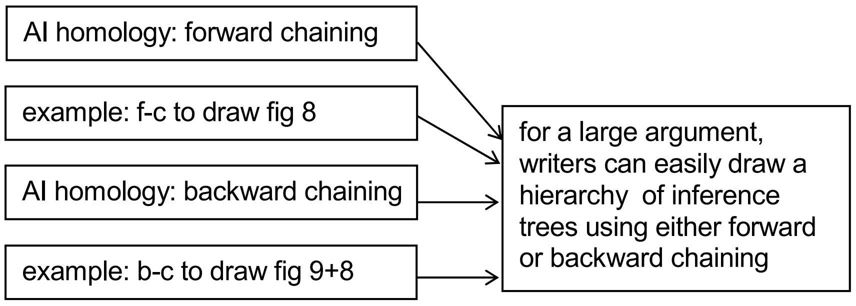
AI homology: forward chaining
example: f-c to draw fig 8
AI homology: backward chaining
example: b-c to draw fig 9+8
for a large argument, writers can easily draw a hierarchy of inference trees using either forward or backward chaining

Figure 15: Low-level tree for chaining

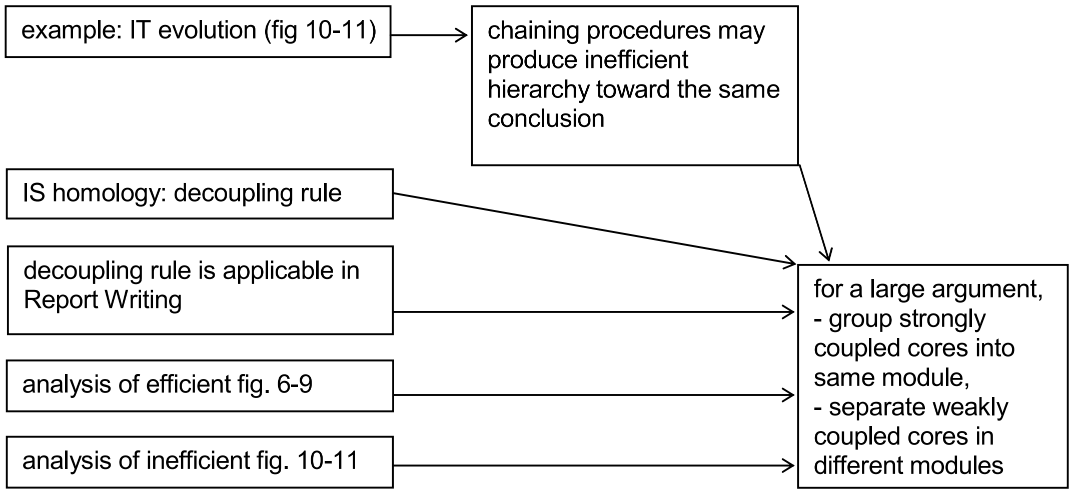
example: IT evolution (fig 10-11)
chaining procedures may produce inefficient hierarchy toward the same conclusion
IS homology: decoupling rule
decoupling rule is applicable in Report Writing
analysis of efficient fig. 6-9
analysis of inefficient fig. 10-11
for a large argument, - group strongly coupled cores into same module, - separate weakly coupled cores in different modules

Figure 16: Low-level tree for decoupling

So writers can use modular inference trees to outline a complicated argument.

Conclusion

The homological transfer method was successfully applied here to transfer modularity, decoupling and chaining principles of Computing into the field of Report Writing. Although these principles are well known in Computing, they are not well known in Report Writing: so writers can learn something new. In particular, the new principles should appeal to writers in Information Systems and Computer Science, because they would readily appreciate the underlying homologies.

The new modularity and decoupling principles of Report Writing are intended to be used in drawing inference trees of expository arguments. An expository argument consists of the core ideas of the body-paragraphs in an expository report, plus inferences between them. An inference tree outlines an expository argument by means of boxes and arrows: the boxes represent core ideas in the various paragraphs of the report, and arrows represent the inferences between the core ideas, from premises through intermediates to the conclusion. The tree can be used to detect reasoning errors in an expository argument: both effectiveness errors that undermine the credibility of the conclusion, and efficiency errors that waste the reader's time.

Except in a very simple argument – involving only three or four inferences – reasoning errors can easily occur. So in order to eliminate them, writers should draw an inference tree, and check it, preferably before writing the report in detail, or otherwise after writing a first draft.

For a not-very complex argument that involves less than a dozen inferences, writers may draw a single inference tree. However if they attempt to do that for a very complex argument, then the tree would spread over several pages, and it would be awkward to draw and to check. Instead, writers should draw a modular hierarchy of inference trees. This includes a top-level concluding argument and several low-level body arguments. To draw those trees, writers may emulate the AI techniques of forward or backward chaining. In forward chaining, they can design low-level trees from the facts to intermediate conclusions, and then design a high-level tree from the intermediate conclusions to the final conclusion. In backward chaining, they can design a high-level tree from the final conclusion to intermediate conclusions, and then design low-level trees from the intermediate conclusions to the facts. With either technique, they should ensure that strongly coupled core ideas are grouped together in the same low-level tree, and that weakly coupled core ideas are separated in different trees. After drawing the modular hierarchy of inference trees, they can use the trees to check for reasoning errors that undermine credibility or waste the reader's time.

So inference trees are useful tools for ensuring effective and efficient argument in expository reports: single trees for less-complex reports, and a modular hierarchy of trees for more complex reports. Readers are invited to experiment with these tools, and if they meet expectations, report their experiences in conferences and journals so that others can also use those tools.

Then in due course someone may even automate them, so that word processors such as MS-Word would draw inference trees at the press of a button, and perhaps even check the reasoning.

References

Arnaudet, M. L. & Barrett M. E. (1984). *Approaches to academic reading and writing*. Englewood Cliffs, NJ: Prentice-Hall.

Evans, J. (1982). *The psychology of deductive reasoning*. London: Routledge & Kegan Paul.

Jackson, M. A. (1975). *Principles of program design*. London: Academic Press.

Mende, J. (1990). Homological transfer - An information systems research method. *South African Computer Journal*, *2*, 6-11.

Mende, J. (2004a). A classification of reasoning errors. Available from http://www.isys.wits.ac.za.

Mende, J. (2004b). Two chaining procedures for outlining expository reports. Available from http://www.isys.wits.ac.za.

Mende, J. (2004c). Two logic tools for outlining expository reports. Available from http://www.isys.wits.ac.za.

Mende, J. (2004d). Using inference trees to detect reasoning errors in expository reports. Available from http://www.isys.wits.ac.za.

Ruch, W. V. & Crawford M. L. (1988). *Business reports*. Boston: PWS-KENT.

Wilson, E. O. (1998). *Consilience: The unity of knowledge*. New York: Alfred A Knopf.

Yourdon, E. (1975). *Techniques of program structure and design*. Englewood Cliffs, NJ: Prentice-Hall.

Yourdon, E. & Constantine L. L. (1979). *Structured design*. Englewood Cliffs, NJ: Prentice-Hall.

Biography

Jens Mende has a bachelor's degree in applied mathematics, a diploma in computer science, a master's in management, and fifteen years' practical experience, mainly in information system analysis, design and programming. He has spent the last twenty-five years at the University of the Witwatersrand, teaching programming, system design, IS management and report writing. He has published three dozen papers on computer education, system design and IS management, and has written another two dozen on logic, report writing, research method and evolution everywhere – which are currently in the process of publication.