Kenneth J. Turner. Abstraction and Analysis of Clinical Guidance Trees, Biomedical Informatics, Copyright Elsevier, October 2008.

Abstraction and Analysis of Clinical Guidance Trees

Kenneth J. Turner

Computing Science and Mathematics, University of Stirling, Stirling FK9 4LA, UK

Abstract

Objectives The aims of this work were: to define an abstract notation for interactive decision trees; to formally analyse exploration errors in such trees through automated translation to LOTOS (Language Of Temporal Ordering Specification); to generate tree implementations through automated translation for an existing tree viewer, and to demonstrate the approach on healthcare examples created by the CGT (Clinical Guidance Tree) project.

Approach An abstract and machine-readable notation was developed for describing Clinical Guidance Trees: ADIT (Abstract Decision/Interactive Trees). A methodology has been designed for creating trees using ADIT. In particular, tree structure is separated from tree content. Tree structure and flow are designed and evaluated before committing to detailed content of the tree. Software tools have been created to translate ADIT tree descriptions into LOTOS and into CGT Viewer format. These representations support formal analysis and interactive exploration of decision trees. Through automated conversion of existing CGT trees, realistic healthcare applications have been used to validate the approach.

Results All key objectives of the work have been achieved. An abstract notation has been created for decision trees, and is supported by automated translation and analysis. Although healthcare applications have been the main focus to date, the approach is generic and of value in almost any domain where decision trees are useful.

Key words:
CGT (Clinical Guidance Tree), Decision Tree, Formal Method, Healthcare, LOTOS (Language Of Temporal Ordering Specification)

1 Introduction

1.1 The Role of Clinical Guidance Trees

Decision trees are often used for decision making, including widespread use for decision support in healthcare. This paper discusses an extended kind of decision tree that is oriented towards interactive use by non-specialists.

A Clinical Guidance Tree (CGT) is an enhanced form of decision tree for use in clinical practice. Compared to conventional medical decision trees, Clinical Guid-

Email address: kjt@cs.stir.ac.uk (Kenneth J. Turner).

ance Trees focus on use by non-specialists, and provide support for interactive exploration. Although these trees may be less common in medical decision-making, various authors have argued that they have a useful role (e.g. [21,25]).

A Clinical Guidance Tree has conventional decision, chance and terminal nodes, and also uses probabilities and utilities (valuations of outcomes). However, such a tree differs in a number of respects from the conventional kinds of trees used in medical decision making. From hereon, the terms CGT and medical tree will be used for brevity to mean these kinds of trees.

CGTs are particularly designed to support health decisions by lay users. However, the approach is generic and is not restricted to healthcare. For example, it can readily support decision making in business, finance and risk assessment. However, the main applications so far have been in healthcare applications.

The work reported in this paper was inspired by the CGT project [1,2]. This created an enhanced form of decision tree that is particularly suitable for interactive exploration by patients. The project developed a textual notation for defining guidance trees. This is supported by a viewer program that allows the user to interactively explore treatment options and to evaluate the likely outcomes of these. The approach is oriented towards patients, who need a user-friendly guide to treatment choices. However, it also has value to medical professionals, who can view the evidence for different choices and the implications of these. The following discussion is a broad overview and does not imply sharp distinctions between the types of tree:

Users: Medical trees are usually designed for use by clinicians to help them evaluate a range of treatments or interventions. They often take account of patient views and values in assessing various outcomes. However, the primary user is expected to be a medical professional.

A CGT is mostly oriented towards by use by non-specialists (e.g. patients) to help them choose treatments and lifestyle changes. The advantage of a CGT is that the user can explore choices at leisure, and can also evaluate options that might otherwise not be considered. It is even possible that patients might be more honest with themselves when exploring options in private.

A CGT is also useful to those with medical training but not specialist expertise (e.g. nurses or even General Practitioners). It can be used as a training aid for those wishing to update their knowledge of available treatments.

These considerations mean that a CGT and its supporting software need to be designed in an accessible manner that non-specialists can benefit from.

Interactivity: Medical trees are normally designed to allow a set of treatment decisions to be evaluated. Based on information supplied by the specialist, alternative strategies can be evaluated. This may lead to a single recommendation, though sensitivity analysis is usually performed to determine whether a range of recommendations should be considered.

In contrast, a CGT is focused more on exploration and explanation than on evaluation and decision. As a result, CGTs are designed for interactive exploration. The user can follow various paths through the tree, including backtracking to earlier points and following other branches – perhaps many times.

Explanation: Medical trees are usually designed for use by specialists who wish to evaluate a range of options. As a result, explanation is mainly limited to stating how certain recommendations are arrived at and what the alternatives are.

Because CGTs emphasise use by non-specialists, they contain much more information. This provides explanations such a layman's description of some treatment or research evidence justifying a choice. This information may be dynamic as it depends on information gathered so far.

CGTs are also usually designed to offer explanations for more knowledgeable users. This kind of explanation typically describes current medical knowledge and refers to the literature to back up the statements made.

Information Gathering: Medical trees usually incorporate information just once (e.g. probability or utility values). Some medical trees allow data to be collected as the tree is evaluated (e.g. a variable has not yet been given a value).

Because a CGT is intended for interactive exploration, information is gathered as the tree is explored. Since backtracking is explicitly allowed, information given earlier may be changed during exploration. Some information may be given once (e.g. a patient's height or weight). However, some information may be more malleable (e.g. whether a patient is willing to exercise more, stop smoking, or choose a particular treatment). A CGT allows the user to explore the consequences of such information. For example, someone with hypertension may initially be unwilling to make lifestyle changes. On seeing the consequences of this, they may go back to earlier choices and see the benefits of making changes.

As well as allowing this kind of information to be changed, a CGT user is able to set or change the utility values associated with different outcomes. Again, the exploratory nature of a CGT encourages users to evaluate different utilities (e.g. they may modify their initial assessment of life with a chronic condition).

Design: Considerable experience has been gained in designing medical trees, including advice on structuring and analysis. The same methodology can be used to design the similar aspects of CGTs. However, there is no methodology for designing the aspects of CGTs that make them distinctive (notably interactivity).

Analysis: The most important analysis performed on medical trees is sensitivity analysis. This reflects their primary role in evaluating decision alternatives. Other analyses include looking for balance in trees, i.e. that possible outcomes have a similar share of both risks and benefits [8]. Since a CGT is an extended form of decision tree, it would be pointless to re-invent these techniques for CGTs. However, the interactive and exploratory nature of CGTs raises the need for new kinds of analysis. The way that a CGT is explored and the way that it gathers information can lead to new kinds of flaws that do not arise in medical trees.

Visibility: Medical trees usually have a fixed structure. CGTs also have a fixed structure, but the need for interactivity makes it desirable to allow sub-trees to be hidden if circumstances dictate. For example, if a patient does not have diabetes then the treatment options presented during exploration may need to change. The result is that users see trees that are tailored to their particular circumstances.

Combination: Suppose a tree has number of outcomes that may be combined in arbitrary ways. For example, the Benign Prostatic Hyperplasia study mentioned

in this paper allows for the combination of two outcomes: improvement in symptoms and side effects of medication. A conventional medical tree would have individual nodes that reflect these combinations (four in this example). Since some clinical trees combine many such factors, this can become cumbersome.

In the case of a CGT, the combination brings extra complications. Each outcome is associated with an explanation for the lay user, and often with medical evidence for the specialist. This information would then be repeated in various combinations for all the nodes. To avoid this, a CGT supports composition of individual nodes. These virtual nodes automatically acquire the explanation and reasoning defined for each case.

This paper refers to the structure of a tree: the tree nodes, how they are combined, and what their parameters are (e.g. probability of a chance branch, utility of a terminal node). A conventional medical tree is not likely to need much more than this level of description. For a concrete example of tree structure, see the tree diagram in figure 4 and its ADIT representation in figure 5 in section 3.4.1.

It will be evident from the above that CGTs typically carry much more information than medical trees. This supplementary information is referred to as the content of a tree. When defining CGTs, it is common for tree structure to be just a few percent in size of the tree content. For a concrete example of tree content, see the ADIT representation in figure 6 in section 3.4.2.

1.2 Abstraction and Formal Analysis of Clinical Guidance Trees

The results of the CGT project provided the baseline for the work reported in this paper. Although this project achieved useful capabilities and flexibility, the author found that a number of fundamental improvements were needed to strengthen the results of the CGT project:

- The CGT project notation for describing trees is almost a flat text file. The structure is indicated only by keywords and layout. This is not in keeping with modern methods of representing structured information (such as XML). It would be highly desirable to have some common representation of decision trees.
- The CGT project notation also mixes tree structure and content. In particular, tree nodes often have substantial additional information associated with them, making it difficult to understand and review the overall structure. Since the trees can become very complex, designing structure independently from content leads to a beneficial separation of concerns.
- No rigorous methodology existed previously for creating CGTs. Rather, they have been created through collaboration between medical and computing professionals. Trees have been validated only through manual debugging (i.e. manual exploration of trees).
- When CGTs are used in healthcare, it is important that their design can be relied on – a patient or a professional may make important choices based on the guidance they receive from the tree. Formal (mathematical) methods support rigorous analysis, and so are appropriate for maximising confidence in tree design.
- As will be seen, the CGT project developed enhanced facilities for decision trees.

Although these offer much greater flexibility and interactivity, the price is that tree behaviour can become very complex. Indeed the behaviour is often infinite, with the possibility of loops and unusual transitions between tree branches. It is therefore hard to establish full confidence in a tree design purely through manual debugging. Again, an automated and formally-based analysis is highly desirable.

CGTs are an extension of conventional decision trees. As a result, conventional analyses (e.g. sensitivity analysis) can also be applied to them. The work reported in this paper has therefore concentrated on the particular challenges of verifying CGTs. This requires new kinds of analysis that reflect the flexibility in exploration and information gathering that CGTs support.

To meet the challenges listed above, the author has developed various solutions:

- A methodology has been created for designing CGTs in a way that clearly separates tree structure and content. The methodology also includes rigorous techniques for analysing the design of a CGT. This uses a formal (mathematical) method to establish tree integrity. Both static (definition) and dynamic (exploration) aspects of a CGT are verified.
- An abstract and formal notation has been defined for CGTs. ADIT [1] (Abstract Decision/Interactive Trees) supports automatic translation among a number of concrete formats for decision trees.
- A toolset has been created to automate many aspects of designing CGTs: definition, translation, verification and exploration.

1.3 Related Work

1.3.1 Decision Support in Healthcare

Decision trees (e.g. [13]) are used for decision making in many applications. An online primer on decision trees can be found at *http://www.projectsphinx.com/decision_trees*. More particularly, decision trees have been found valuable in medical and clinical practice (e.g. [5]). An online presentation of decision making in clinical research can be found at *http://symptomresearch.nih.gov/chapter_14*. Healthcare applications of decision trees include their use in clinical practice, nursing and patient care. The journal *Medical Decision Making* has published many articles on the use of decision trees in healthcare. Markov models have been widely adopted as the basis of medical decision trees [23].

Decision Support Systems (DSS) are common in healthcare. Among the many techniques developed are ASBRU [22], EON [17], PRODIGY [12], and PRO*forma* [12]. However, these systems are almost invariably designed for use by clinicians.

Healthcare policies are increasingly stressing the importance of involving patients in treatment decisions (e.g. [7]). Decision aids for direct use by patients are therefore becoming more common. A perspective on the spectrum of decision support in healthcare appears in [20]. In terms of this classification, a Clinical Guidance Tree is a consumer (patient) oriented tool. The concept of a guidance tree was first

[1] An 'adit' is a horizontal entrance to a mine.

explored in [10,11], and subsequently developed and trialled in [1,2,21,25]. The present paper focuses on the special characteristics of CGTs, dealing with their design, representation and analysis.

Decision trees are well supported by commercial and open-source tools. As an extended form of decision tree, a CGT requires additional tools such as an interactive viewer and a behaviour verifier.

A common graphical convention for decision trees [14] uses squares for decision nodes, circles for chance nodes, and rectangles (with utilities) for terminal nodes. The diagrams in this paper use this notation, but with the addition of a diamond symbol for questions. However, these are just diagramming conventions. Arden syntax has been used to define condition-action rules for clinical procedures. GLIF (Guideline Interchange Format [18]) is designed for interchangeable descriptions of clinical guidelines. A comparison of such formats is given in [19].

However, the author is unaware of any standard for machine-readable descriptions of decision trees. This paper proposes a neutral notation for decision trees, including the characteristics required for CGTs. The notation can readily be translated into other notations (textual, structured, graphical, formal).

Well-known techniques and tools exist for analysing conventional decision trees. For example, sensitivity analysis is used to investigate how strategies change as key variables change [9]. Automated techniques can be used to analyse decision trees for design flaws [28]. These same techniques can be applied to CGTs since they are an extended form of decision tree. However, CGTs have distinctive characteristics (notably interactive exploration) that require new forms of analysis – a key goal of the work reported in this paper.

1.3.2 Formal Methods In General

The term 'formal method' is used in medical science of any systematic approach. For example, the use of a decision tree is considered to be a formal method. Monte Carlo simulation is used for statistical analysis of decision trees [6]. Markov models are also commonly used for analysing decision trees (e.g. [23]).

However, a formal method (as used this paper) has a much more specific meaning in computer science. There it refers to a mathematically-based technique for modelling, specification and analysis of computerised systems. This is a very large field with many techniques and tools; see *http://vl.fmnet.info* for a regularly updated overview. Several formal methods have been standardised, notably LOTOS (Language Of Temporal Ordering Specification [15]).

Perhaps surprisingly, formal methods in computer science have seen very limited use for modelling and analysis of decision trees. The only example known to the author is the PRO*forma* approach to decision support [24]. An operational semantics for PRO*forma* is defined with respect to an abstract machine that executes clinical decision procedures or guidelines.

The interactive nature of CGTs mean that they are prone to classes of error that do not arise in conventional decision trees. As a result, a new approach is needed

to analysing decision trees for such errors. This calls for a formal method that can effectively describe and analyse the exploration of a tree of behaviours.

Formal methods support two broad categories of analysis: validation (testing) or verification (proof). Validation is necessarily finite, and is usually incomplete; however, it is practical for complex specifications or those with infinite behaviour. Rigorous validation demonstrates that a specification behaves correctly for a finite set of test cases. Verification is technically much more challenging, and is usually ineffective for complex or infinite behaviours (unless these lend themselves to some form of symbolic verification).

Model checking is a popular verification technique. It establishes whether a specification respects certain desirable properties. Model checking investigates the dynamic behaviour of a system by considering its state space. Generic properties of a specification can be checked, such as freedom from deadlock (further behaviour is blocked) and livelock (an internal loop without external communication). However, it is often necessary to check specific properties (such as whether a particular tree node can be reached or whether a particular constraint holds). These properties are expressed in a temporal or modal logic that allows the evolving or potential behaviour of a system to be described.

1.3.3 The LOTOS Formal Method

LOTOS is a internationally standardised language for formal specification and rigorous analysis. Although conceived for use with communications systems, LOTOS has been used in many other areas. As an example from the medical field, it has been used for modelling and testing of radiotherapy accelerators [26]. LOTOS is classed as an algebraic specification language: abstract data types are specified by equations defining their operations, and behaviour is specified by interacting processes whose behaviour follows algebraic rules. Unlike a number of formalisms, LOTOS fully supports the integrated specification of data and behaviour.

LOTOS was chosen to model CGTs partly because of its flexibility, partly because its capabilities are a good match to the characteristics of CGTs, and partly because of the good tool support for analysis. The main issue with LOTOS for CGTs is that its data type library is rather rudimentary. However, it is extensible – the data types needed for decision trees were added in the course of this work.

For space reasons, an introduction to LOTOS is not provided here. Instead, the LOTOS specification extracts are extensively commented. An overview of LOTOS is given in [3]. Online tutorials can also be found at *http://www.inrialpes.fr/vasy/pub/cadp* and at *http://www.cs.stir.ac.uk/well*.

LOLA (LOTOS Laboratory) is the tool that was used to validate CGT descriptions. LOLA includes various commands to generate the state space subject to various constraints: limiting the exploration depth, recognising revisited states, or combining the behaviour with a test process.

CADP (Construction and Analysis of Distributed Processes, *http://www.inrialpes.fr/vasy/cadp*) is the toolset that was used to verify CGT descriptions. Desirable

properties of trees can be written in XTL (Extended Temporal Logic [16]) and then model checked by CADP. Efficient verification with CADP normally requires key data types to be implemented in C. A conversion tool was also written to generate the additional annotations that CADP needs for CGT data types.

1.4 Overview of The Paper

Section 2 illustrates a Viewer program for interactive exploration of CGTs. The new methodology for developing CGTs is also described. Section 3 explains the limitations of the CGT notation for decision trees. This motivated the definition of the new ADIT notation. Section 4 explains how decision trees in ADIT notation are translated into the LOTOS formal language. The complexity of certain tree features emerges during this discussion. Section 5 discusses how LOTOS specifications of CGTs can be analysed. It is explains the kinds of errors found in the trees developed by the CGT project. Section 6 summarises the results and points to future work.

2 Using Clinical Guidance Trees

2.1 The CGT Viewer

Support for Clinical Guidance Trees was developed by a project on 'The Development and Evaluation of A Computerised Clinical Guidance Tree for Benign Prostatic Hyperplasia and Hypertension' [1,2]. The capabilities of the CGT system are described here as background to the new work in this paper. The primary tool developed by the CGT project was a decision tree viewer. The main focus of CGT was a range of medical conditions: benign prostatic hyperplasia (swelling of the prostate), hypertension (high blood pressure), influenza, and menorrhagia (excessive bleeding during periods). However, the approach is completely general and could be used for decision trees in any other field.

The CGT Viewer is a graphical application that takes the user through several stages, illustrated here when exploring BPH (Benign Prostatic Hyperplasia):

(1) The user is first given background information on what a particular decision tree covers, e.g. the nature of some medical condition.
(2) The user is then allowed to explore the tree graphically, e.g. to investigate treatment options and their consequences. In figure 1, the user has navigated to the point where a particular medication is described (Finasteride). As shown at the bottom right of this figure, the user can explore various outcomes by clicking on treatment choices. General navigation is shown at the bottom left of this figure, where the user can move to an alternative branch or can backtrack. Professionals can opt to see research evidence that describes each intervention.
(3) The user is then asked to associate utilities with the outcomes of the decision tree. If necessary, the user can later backtrack to this stage and adjust these utilities in the light of changed priorities.
(4) The user's weighting of outcomes then determines the best path through the tree, i.e. which treatment choice best suits the user. As shown in figure 2, phytotherapy (herbal medicine) has the best score. The user can ask for an

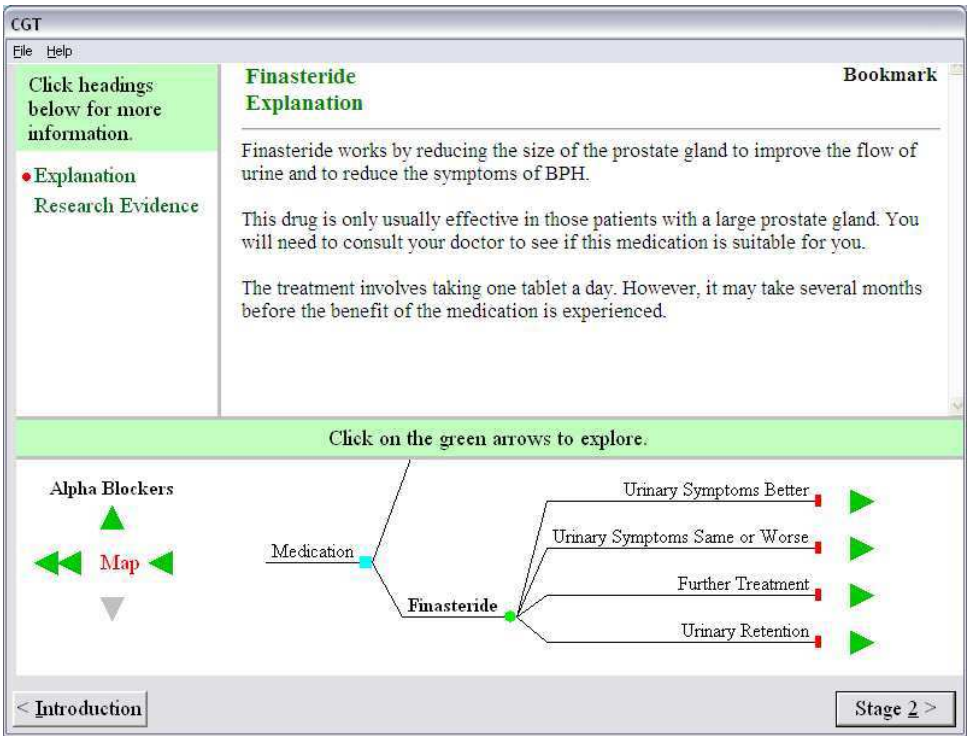

Fig. 1. The Option of Finasteride Treatment for Benign Prostatic Hyperplasia

explanation of each choice.

The CGT Viewer reads decision trees in a textual notation, with keywords and layout used to structure the text. Besides explanation, exploration, analysis and recommendation, the CGT Viewer has other useful functions such as summarising an exploratory session and recording the user's choices for statistical analysis.

2.2 Methodology for CGT Design

Figure 3 summarises the new methodology for defining CGTs using ADIT (Abstract Decision/Interactive Trees). An initial concept is refined manually into the basis of a decision tree. This stage focuses on structure and flow in the tree. A formal specification of the tree is automatically generated and evaluated using the techniques described in section 5.2. This may identify problems in the design, leading to a revised tree and re-evaluation. Now the detailed content can be added to the tree. Again, this can be automatically formalised and evaluated. The abstract tree design is then automatically converted into an executable representation for use with a tree viewer (currently the CGT Viewer). The implementation can be used many times to generate advice and recommendations.

A two-stage design (structure then content) is not enforced, nor is use of formal analysis. However, both of these are useful and desirable – especially when a complex or critical decision tree is being designed.

Trees can be defined directly in the ADIT notation. However, a separate decision

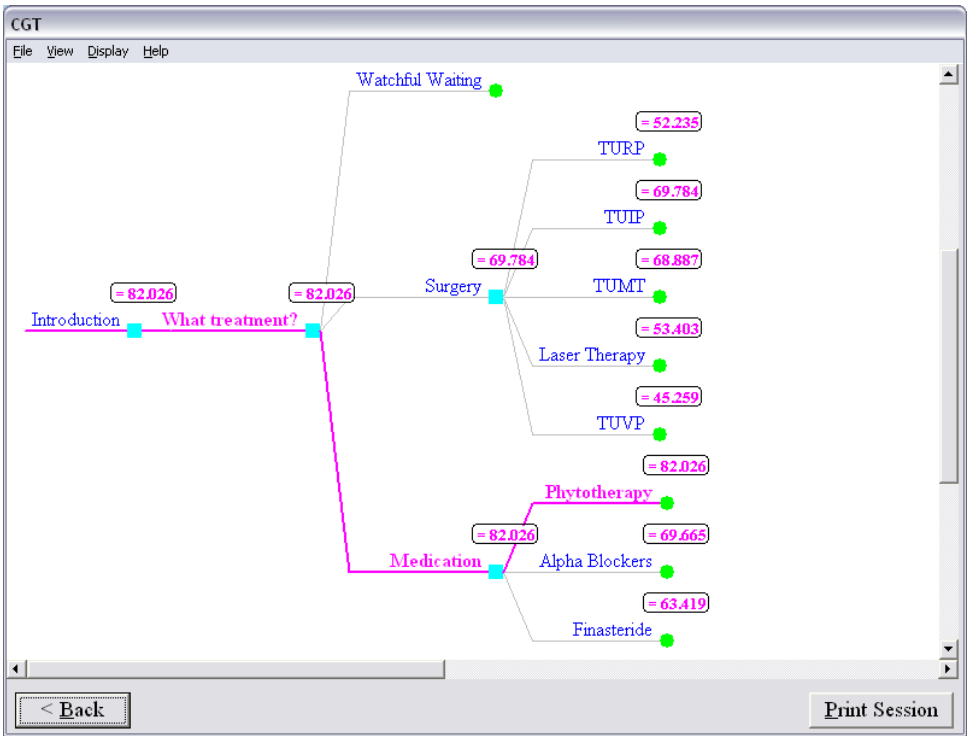

Fig. 2. Best Path through The Decision Tree for Benign Prostatic Hyperplasia

tree editor (developed by Ross MacKenzie, University of Stirling) allows trees to be defined using a graphical interface. This editor reads and creates the XML form of ADIT. Work is also under way to link the ADIT toolset to open-source tree editing software (TreeForm, *http://sourceforge.net/projects/treeform*) and commercial decision tree software (TreeAge, *http://www.treeage.com*).

3 Defining Clinical Guidance Trees

3.1 CGT Notation

The CGT Viewer is an effective and user-friendly tool. However, defining decision trees using its notation is an intricate and error-prone task. The CGT project notation suffers from a number of problems:

- A tree is defined by an almost flat text file. Realistic trees require very long files (some thousands of lines of text) and are therefore hard to grasp.
- Node descriptions often have large amounts of explanatory text (several pages). As a result, it is very hard to see the tree structure because the notation does not clearly separate structure and content. The notation does not readily allow structure to be developed first, and then populated with content.
- The commonest and most serious errors in CGT design are structural, e.g. unconnected nodes or linking the wrong nodes. As will be seen later, the CGT notation allows complex flows that bypass nodes, unusual transitions between branches, and conditional inclusion of portions of a tree. Unfortunately, this flexibility risks

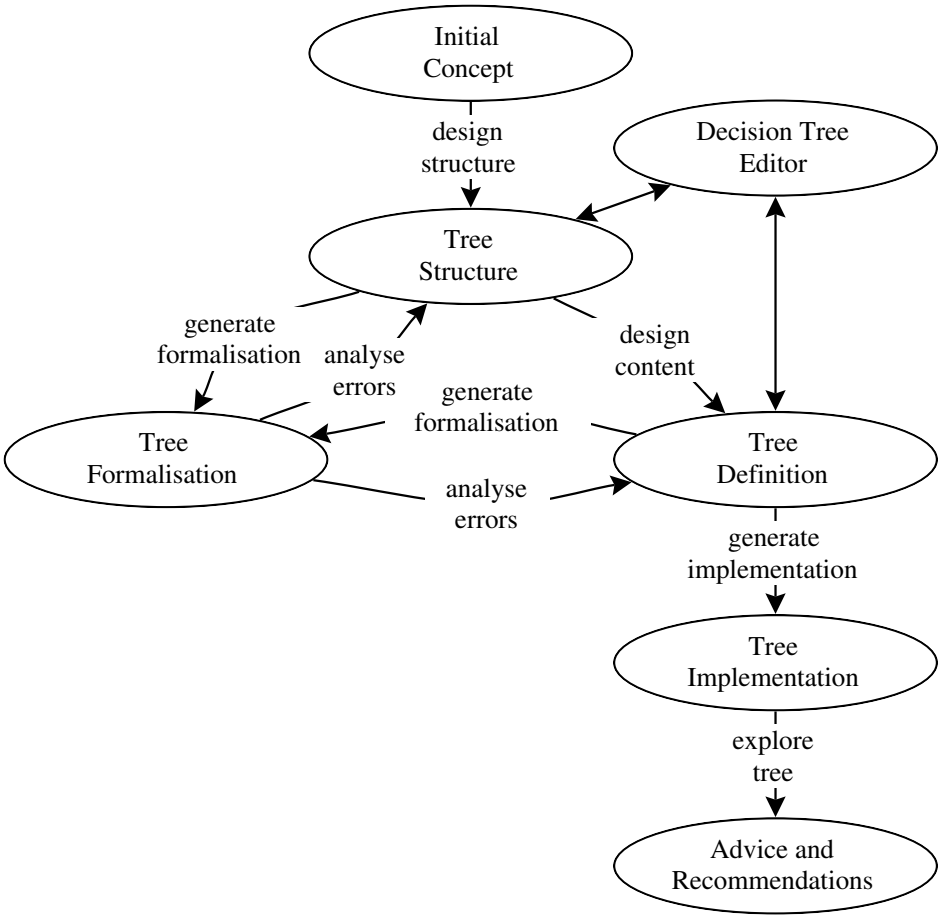

Fig. 3. ADIT Methodology

making errors in the tree flow.

- In general, there is no tool support for designing correct CGT trees. The designer must do this manually, and rely on extensive debugging to find problems.

3.2 Defining Tree Structure with ADIT

The design of ADIT was heavily influenced by the CGT project. However, the new approach deals with the difficulties noted above, is more abstract, and supports formal analysis. In ADIT, the tree structure is completely separate: it need define only the tree nodes and their relationships. Content is normally defined through reference to separate node attributes. However, it is still possible to include literal attributes with a node, typically for simple values such as a probability or a payoff.

ADIT has various syntaxes that can be interconverted. The simplest one is applicative (function-like). This is compact, easily read and easily parsed by machine. In this syntax, a tree is defined by nested nodes in the form *node(parameters,children)*. The applicative syntax is used in this paper because it is the most compact. However, for easy interchange with other programs, ADIT also has an XML syntax where nodes correspond to elements and parameters correspond to attributes. A

Directive	Meaning
// *text*	an explanatory comment about the tree that is removed in the translated output
chance(*id,label,attributes,node1,..*)	a probabilistic ('system') choice of child nodes
comment(*text*)	an explanatory comment about the tree that is transcribed to the translated output
decision(*id,label,attributes,node1,..*)	a deterministic (user) choice of child nodes
question(*id,label,attributes,node1,..*)	a request for user input prior to child nodes
terminal(*id,label,attributes*)	a leaf node
tree(*id,label,attributes,node*)	the whole tree with a single root node
value(*name,value*)	a textual, numeric or code definition

Table 1
Summary of ADIT Directives

third syntax is that used by the CGT Viewer program. All three formats are automatically translated into each other, so the choice is up to the designer. The various syntaxes are also automatically converted into other forms, such as the LOTOS representation used for formal specification and analysis.

The top-level directives for defining tree structure are listed in table 1. As in other forms of decision trees, there are **chance**, **decision** and **terminal** nodes. To allow for capture of user input during tree exploration, there is also a **question** node. The whole structure is defined by a **tree** whose structure is defined by the nested node definitions. Each node has an identifier, a short label and attributes. All nodes except terminals may have child nodes; a tree has a single child node as the root. A **value** definition is used to associate content with a node.

Question nodes allow greater interactivity with the user. At selected points, the user can be prompted to provide input (e.g. about symptoms or lifestyle). This information can be used to influence the later behaviour of the tree. In essence, the user is asked a question whose answer is stored in a tree variable. However, question nodes add considerable complexity (and were found to be rather loosely specified by the CGT project).

Interactive navigation allows the user to backtrack to a question that was answered previously. At this point the user can retain the previous answer or can change it. The user is allowed to skip a question (or a series of questions). This is permitted because the user may be undecided, or may be unwilling to make a commitment at this point in the exploration. The answers to such questions remain undefined or are given default values, depending on the design of the tree. Skipping questions can cause surprising transitions between nodes, going from one branch of a tree directly to another. Question nodes may also be rendered visible or invisible as the tree is traversed, again resulting in potentially surprising or incorrect behaviour.

Section 4.2.5 discusses the complexities of question nodes in more detail.

The specification of a question includes a definition of the permitted answers. The units for an answer (e.g. height in metres) are stated in a question and so are implicit in an answer. An answer is validated by checking that it falls within the permitted set of values (an enumerated set or numerical range).

3.3 Defining Tree Content with ADIT

The content of a tree is normally defined by separate **value** attributes. Indeed the ADIT design philosophy encourages the designer to focus initially on just the tree structure and flow. At this stage, the attributes are largely unimportant and can mostly omitted. Only once the tree structure has been defined and validated is it necessary to elaborate the content.

Tree content is defined by means of the node attributes listed in table 2. A number of attributes are common, while some apply only to particular kinds of nodes. Attributes can be defined literally or by reference. Since ADIT is based on work by the original CGT project, [2] can be consulted for more detail on how these attributes are used. The more specialised ones are discussed briefly below.

ADIT encourages short definitions of nodes so that it is easier to understand the tree structure (as defined at the end of section 1.1). A literal definition is usually given only if it is simple (such as a probability value or a variable name). A literal attribute has a form such as *probability=″0.3″*. Usually only those aspects necessary to define the tree structure are defined along with a node. This is the information that is normally associated with a conventional decision tree: the node type, identifier, and (if relevant) probability and utility value.

Extended attributes are preferably defined separately from their associated nodes. Suppose that extensive explanation is needed of a particular treatment choice (e.g. of the research evidence that underlies it). Rather than enlarge the node definition by several pages of text, the explanation should be defined in a separate attribute and referenced in the node. The corresponding value definition is named after the node and the attribute. For example, a node (e.g. identifier *WatchfulWaiting*) may refer to a separate attribute (e.g. *reason*). The attribute reference and definition are linked through a name in the form *node_attribute*.

ADIT also includes features such as expressions, text markup, question nodes, conditional visibility and node composition.

Nodes typically have substantial textual content defined in separate attributes. All nodes have implicit *display* attributes for explanations to the user. Several other attributes such as *query* and *reason* are also defined as text. Although not illustrated here, HTML markup can be used in text. In addition, text can contain macros. These are conventional macros (possibly with parameters) that expand to pieces of text. They are useful for things like common explanations or shared calculations.

Sub-trees are visible by default, but may be rendered invisible during exploration if certain conditions are met. For example, the BPH decision tree computes an

Attribute	Meaning	chance	decision	question	terminal	tree
composed	expression for using composed node headings					✓
conjunction	text to join composed nodes					✓
dictionary	name of a glossary file					✓
display	text for user display (assumed by default)	✓	✓	✓	✓	
error	text for reporting a validation error			✓		
format	format for user input			✓		
label	long label for a node	✓	✓	✓	✓	
macros	global macros					✓
neutral	payoff between positive/negative outcomes					✓
payoff	expression for a payoff (i.e. utility value)				✓	
perform	user instruction text	✓	✓	✓	✓	
print	expression for summarising a node	✓	✓	✓	✓	
probability	expression for probability of a choice	✓	✓	✓	✓	
query	text for question to user			✓		
reason	text for explaining a choice	✓	✓	✓	✓	
scale	expression for scaling composed node payoff	✓	✓	✓	✓	
valid	expression to validate a question answer			✓		
variable	question variable			✓		
variables	tree variables					✓
version	tree notation version					✓
visible	expression to check node visibility	✓	✓	✓	✓	

Table 2
Summary of ADIT Attributes

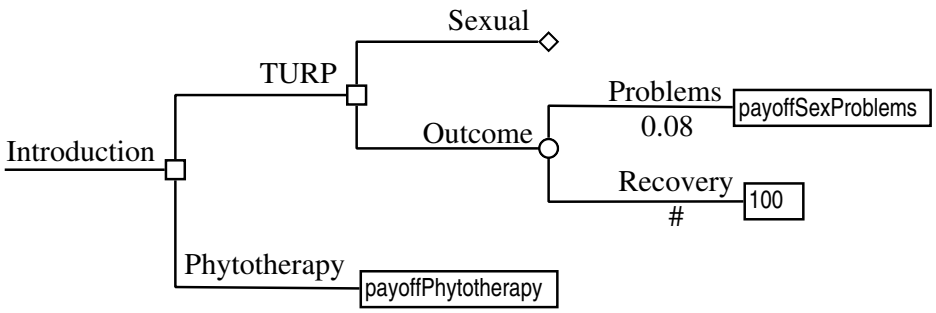

Fig. 4. Decision Tree for Benign Prostatic Hyperplasia

AUA (American Urological Association) symptom score based on what the user reports. Watchful waiting is appropriate only if the symptom score is less than 20; a higher value requires active intervention. The visibility of the *WatchfulWaiting* node (and its children) is therefore defined by a *visible* attribute. During a particular exploration, a sub-tree may be deemed irrelevant and should therefore be hidden. Nonetheless, even an invisible node may have an effect on others. For example, it may affect their relative probability or utility values.

Many decision trees lead to terminal nodes that are not disjoint. This can lead to considerable duplication in the tree – particularly of the explanatory information that would be identical in all cases. It would be tedious and error-prone if the tree designer had to define all such cases explicitly, since there would be substantial overlap in their descriptions. Instead, a virtual node may be composed from others by combining their short labels with '&', e.g. 'Urinary Symptoms Better & Finasteride Side Effects'.

Some of the attributes in table 2 deal with composed nodes. The *composed* attribute indicates that the labels of the composing nodes should be used as a heading in the composed node. The *conjunction* attribute defines text to join that of the composing nodes. The *scale* attribute is used for scaling payoffs where a composing node is not visible in the tree. The *neutral* attribute defines the boundary between advantageous and disadvantageous payoff values, for use in calculating the payoff of composed nodes. As explained in [2], the neutral point is used to classify composed nodes automatically as desirable or undesirable from the user's point of view.

3.4 A Decision Tree Example

3.4.1 Tree Structure

To illustrate the ADIT notation, an extract has been taken from the full tree for Benign Prostatic Hyperplasia (BPH). The extract serves only to explain the notation, and is not particularly meaningful in isolation. Figure 4 shows the tree structure, while figure 5 shows this in ADIT form.

The tree explores the consequences of choosing certain BPH treatments. Nodes in a tree have an identifier such as *BPH* and a short label such as 'Benign Prostatic Hyperplasia'. The top-level tree element has a *variables* attribute that defines tree

tree(BPH, Benign Prostatic Hyperplasia, variables,
  decision(Introduction, BPH Introduction, ,
    decision(TURP, Transurethral Resection, reason,
      question(Sexual, Sexually Active?,
        query format=″Edit(1)″ variable=″sexual″ valid),
      chance(Outcome, Surgery Outcome, ,
        terminal(Problems, Sexual Problems,
          probability=″0.08″ payoff=″payoffSexProblems″ visible),
        terminal(Recovery, Full Recovery,
          probability=″#″ payoff=″100″ reason))),
    terminal(Phytotherapy, Phytotherapy Treatment,
      payoff=″payoffPhytotherapy″ reason)))

Fig. 5. Decision Tree Structure for Benign Prostatic Hyperplasia

variables. This is followed by *Introduction* as the root node. Two main branches can then be followed the user:

TURP: This decision node allows the user to investigate Transurethral Resection of the Prostate (TURP) as a surgical option. A *reason* attribute gives research evidence for this option. The user can now decide between answering a question about sexual activity and the outcome of TURP:

Sexual: This question node asks whether the user is sexually active. The *query* attribute refers to a separately defined question. The *format* attribute defines the answer as an *Edit* (i.e. free-form) response of one character. The *variable* attribute defines which variable (*sexual*) will receive the answer. The *valid* attribute refers to a separately defined a check on the answer.

Outcome: This chance node leads to different terminal nodes, each with associated probability and payoff:

Problems: This terminal node corresponds to sexual problems after surgery. The *visible* attribute of *Problems* refers to a separately defined check on the visibility of this node. If the user is sexually active, the node is visible during tree exploration. If not, the node is invisible (i.e. only the *Recovery* node is visible).

Recovery: This terminal node corresponds to full recovery after surgery. The *probability* attribute uses '#' to mean the residual probability. This will be 0.92 if *Problems* is visible, or 1.00 if not.

Phytotherapy: This terminal node allows the user to consider phytotherapy (treatment with herb or plant extracts).

This example is a little artificial for illustrative purposes. Since the *Sexual* question is in the context of a decision, the user can choose to ignore it. For this reason, a default answer must be defined (by initialising *sexual* to 1). The *Sexual* question requires a free-form answer and therefore must be validated. In practice the tree designer would use the format *Radio(no,yes)*, which is more obvious to the user and requires no validation. If the user is not sexually active, *Outcome* does not involve a chance alternative. The full BPH tree has many more branches at this and other points.

value(BPH_ Display, Benign Prostatic Hyperplasia is ...)
value(BPH_Variables, payoffPhytotherapy = 80; payoffSexProblems = 20; sexual = 1)
value(Introduction_ Display, In most cases the best treatment is ...)
value(Outcome_ Display, Surgery may lead to a variety of outcomes ...)
value(Phytotherapy_ Display, The use of plants and herbs (phytotherapy) ...)
value(Phytotherapy_ Reason, This treatment is suitable only for ...)
value(Problems_ Display, This surgery can lead to sexual problems ...)
value(Problems_Visible, sexual)
value(Recovery_ Display, Full recovery is possible after this surgery ...)
value(Recovery_ Reason, 29 RCTs and the UK prostatectomy audit ...)
value(Sexual_ Display, If you are sexually active then ...)
value(Sexual_ Query, Are you sexually active? (0 = no, 1 = yes))
value(Sexual_Valid, sexual == 0 or sexual == 1)
value(TURP_ Display, Transurethral Resection of the Prostate is ...)
value(TURP_ Reason, 30 reports informed this advice ...)

Fig. 6. Decision Tree Content for Benign Prostatic Hyperplasia

3.4.2 Tree Content

The detailed tree content is shown separately in figure 6. Of necessity, the information here is highly abbreviated as it occupies four pages. However, this highlights the point that is helpful to separate content from structure. The tree content is linked to the tree structure by combining node and attribute names. For example, value *TURP_ Reason* corresponds to node *TURP* and attribute *reason*. The definition of *TURP_ Reason* in figure 6 is automatically used in the *TURP* node of figure 5.

The implicit *display* attributes correspond to *Display* values. The top-level tree definition has special attributes such as *BPH_Variables* here to declare and optionally initialise tree variables.

The attributes *valid* and *visible* define boolean expressions. The approach of the CGT project to expressions has been respected: these are patterned after the C programming language and its derivatives. In fact, it would be better to call these statements rather than expressions: they are statement sequences that yield a value. For example, an assignment to a variable is an expression that yields the new value. If there is a sequence of statements or a conditional statement, the last calculated expression defines the overall result.

Problems_Visible simply returns the value of the *sexual* variable. *Sexual_Valid* returns true if *sexual* is 0 or 1. However, in general such expressions can be complex and have side-effects.

In the trees developed by the CGT project, expressions with side-effects (variable assignments) are frequently used. Although this is convenient from the tree designer's point of view, it makes the semantics of the tree much more complex. In the author's opinion, purely functional expressions would have been preferable. Side-effects could have been achieved through separate assignments. However, ADIT respects the original CGT work for backwards compatibility.

Application	ADIT		CGT		LOTOS	
	Lines	Attrs.	Lines	Nodes	Lines	Processes
Benign Prostatic Hyperplasia	3592	492	3258	96	2001	116
Influenza	583	49	577	16	369	17
Hypertension	6510	331	6205	75	3285	124
Menorrhagia	1034	191	926	53	1010	55

Table 3
Representations of Healthcare Applications

3.5 Applications of ADIT

Although ADIT addresses the problems of the CGT notation discussed in section 3.1, its value lies in being able to convert tree descriptions into other notations. A suite of tools has been created to automate the following tasks:

- For use with the CGT Viewer (section 2.1), ADIT descriptions can be converted to/from the notation used by this tool. This allows users to benefit from existing work on developing trees for a variety of conditions.
- For easier interchange with other tools, ADIT can be converted to/from XML (based on work by Richard Bland, University of Stirling). This also allows use of the decision tree editor mentioned in figure 3.
- To support formal analysis of decision trees, ADIT can be converted into LOTOS as discussed in the next section.

ADIT also lends itself to translation into graphical languages such as GraphML [4], which is supported by a range of graphical editors.

The ADIT conversion tools consists of around 3600 lines of code, written in the Perl scripting language and the M4 macro language. These are not, perhaps, obvious choices for implementing a translator. However, the syntax of ADIT is very simple and does not merit the use of normal compiler tools. The author also has had good experience of developing several translators with Perl and M4.

To give some idea of the decision trees that have been developed, table 3 presents statistics on various healthcare applications. This shows the number of text lines in each representation, plus the number of separately defined attributes (ADIT), tree nodes (CGT) or processes (LOTOS). The table gives some idea of the scale of these examples: the larger trees are non-trivial. The table also gives some idea of how the different tree representations compare in size and number of constructs.

These healthcare applications were created by the original CGT project. The CGT team members were mostly medical professionals, with computer scientists being responsible for defining the CGT notation and CGT Viewer. The procedure for development of these applications was as follows:

(1) The medical professionals identified various conditions where an interactive decision aid would be valuable. The concept for each application was then

elaborated. This included a preliminary structure for the tree, coupled with an outline of the explanations and research evidence required.

(2) The computer scientists then coded the trees. As noted in section 1.2, there were several methodological weaknesses in this process: lack of a rigorous design methodology, tree structure and content were not clearly separated, and lack of formal semantics for tree definitions. Many iterations were needed, with the computer scientists discovering gaps in the tree definition and the medical professionals filling this.

(3) The CGT team now manually debugged the trees. This identified technical errors (such as missing nodes or incorrect tree transitions) and explanation errors (such as incomplete advice). Again, many iterations were needed.

(4) Following ethical approval, the trees were now tried by volunteer patients. This stage was mainly focused on evaluating the capabilities of the approach.

(5) Finally, selected trees were used in real trials with patients: the hypertension guidance tree [25], and the menorrhagia guidance tree [21]. The author is not aware of any RCTs so far using the BPH tree.

Prior to deployment, roughly two man-years of effort had gone into thorough development of the trees by medical and computing professionals. The trees had also been informally evaluated by patients. All significant flaws should therefore have been eliminated prior to the work reported here. Section 5 describes the kinds of formal analysis that the trees were later subjected to, and what emerged from this investigation.

4 Formalising Decision Trees

4.1 Specification Approach

4.1.1 Level of Abstraction

ADIT descriptions of decision trees are automatically translated into LOTOS for analysis. The translation of a decision tree is an abstraction of what the user sees when exploring it. Certain aspects are intentionally excluded because the focus is on analysing the structure and flow of the tree. Explanatory text is not included in the specification. This is considered to be content and is best developed manually.

An important aspect covered by the formalisation is user exploration of the tree. In particular, the user is allowed to backtrack in the tree as well as move forwards when making choices. As will be seen when node checks and question nodes are discussed, it is easy to make mistakes in these aspects of the tree flow. Tree navigation is therefore supported in the formal specifications.

Numerical aspects such as probabilities and payoffs are not currently handled in the specification. These will be incorporated in future using a probabilistic variant of LOTOS. The current emphasis of ADIT is thus on the functional behaviour of the tree, i.e. on defining the tree flow and on detecting possible errors in this.

4.1.2 Tree Variables

A number of specialised data types were defined in LOTOS for the ADIT library. All tree variables are floating point numbers – a type that the standard LOTOS library does not support. A rather complex specification was developed for real numbers in the form *<sign,whole,fraction>*.

Typical trees have many variables (hundreds in some cases). Although each variable could be a separate process parameter in LOTOS, this would be extremely unwieldy. Instead, variable values are stored in a *map* as a single value. This is the usual concept of a map from variable names to values.

Since LOTOS does not have global variables, the variable *map* is passed into and out of every process to reflect changing variable values. Some special variables are used for internal purposes: *backing* (whether the user chose to backtrack), *satisfied* (whether a question was answered correctly), *skipping* (whether the user chose to skip all questions of a series), *valid* (whether a question answer is valid), and *visible* (whether a node is visible).

4.1.3 Tree Expressions

A number of attributes (e.g. for validity or visibility checks) define expressions. As discussed in section 3.4.2, expressions can have side-effects. Furthermore, expressions often use conditional or sequential forms. As a result, a tree expression cannot simply be translated as a LOTOS value expression (which is always purely functional). Instead, a tree expression is translated into a LOTOS process that takes a variable map and produces a variable map. Such processes behave almost like functions, but can have side-effects, conditions and sequences.

Since probabilities and payoffs are currently abstracted away, any variables dealing with these are eliminated during translation into LOTOS. For the same reason, expressions involving these variables are also removed. The trees produced by the CGT project often used checks with a value of *false*, i.e. some questions may never be answered correctly or some sub-trees may never be traversed. It was found that the CGT project needed these only during early stages and intended them to become dormant. The LOTOS formalisation handles this by eliminating unreachable portions of a tree during translation.

The translation of a typical validity check (*Sexual_Valid* in figure 6) is shown in figure 7. The effect of this process is to set the validity of the answer to the *Sexual* question. Variable values are retrieved from a map via *get* and stored with *set*. In this case, the value of variable *sexual* is retrieved and checked to be 0 or 1. The resulting boolean value is stored in the map as the last result calculated. The outcome of a check is a new map; this is accepted and stored in a variable named after the node (*Sexual*). A similar translation strategy is followed for visibility checks. If there are side-effects or complicated expressions, validity and visibility checks have fairly complex translations to LOTOS.

LOTOS processes resemble procedures or methods in a programming language. Process parameters may be given in parentheses (e.g. *map* for tree variables). Pro-

```
Process SexualValid (map:Map) : Exit(map) :=                    (* validate sexual *)
  Exit(set(get(sexual,map) == 0 or get(sexual,map) == 1, map))  (* exit with validity *)
>> Accept map:Map In                                            (* get new map *)
  Exit(setValid(Sexual,map))                                    (* set validity for Sexual *)
EndProc                                                         (* end SexualValid *)
```

Fig. 7. Validation Process for *Sexual*

```
Process Recovery [user] (map:Map) : Exit(Map) :=                (* terminal Recovery *)
  user !Recovery;                                               (* enter Recovery *)
  (
    Exit({})                                                    (* exit leaf node *)
  []                                                            (* or *)
    user !Back;                                                 (* go back *)
    Outcome [user] (map)                                        (* to parent *)
  )
EndProc                                                         (* end Recovery *)
```

Fig. 8. Terminal Node Process for *Recovery*

cesses may exit with optional values (e.g. the resulting *map*). A LOTOS process communicates via 'gates' (like ports) that are given in brackets. Process outputs have the form *gate !value*, while process inputs from have the form *gate ?variable:type*. Comments in LOTOS appear in '(* ... *)'.

4.2 Node Specifications

4.2.1 Nodes in General

Each tree node is translated to a LOTOS process. The automatically generated specification is neatly laid out and fully commented – the examples in this paper are literal extracts from the translator output. This ensures that the specification can be readily related to the original ADIT description. The tree example in figures 5 and 6 is translated to 430 lines of LOTOS and 10 processes.

4.2.2 Terminal Nodes

The translation of a typical terminal node (*Recovery* from figure 5) is shown in figure 8. All nodes start by advising the user they have been entered using an event of the form 'user !*node*'. As it is a leaf node, a terminal can exit the entire specification with an empty map ('{}') as the map is no longer significant at this point. However, the user may backtrack to the node's parent (*Outcome*) by issuing a *Back* command. Backtracking may cause behaviour to become infinite, making formal analysis more difficult. For this reason, backtracking can be omitted from the specification through a translator option.

4.2.3 Decision Nodes

After entry, a decision node simply allows a deterministic (i.e. user) choice of its child nodes using the LOTOS '[]' ('or') operator.

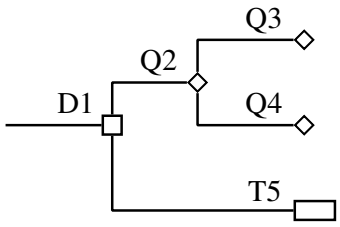

Fig. 9. A Decision Tree with Questions

4.2.4 Chance Nodes

After entry, a chance node allows non-deterministic exploration of its child nodes (i.e. the user cannot influence the selection). In LOTOS, non-deterministic alternatives are selected by an internal event **I**. This is anonymous, i.e. the exact nature of this event is not stated. As a translator option, this may be made explicit (using a hidden event gate for each of the choices). Internal events in LOTOS represent actions within a system that are not explicitly identified or controlled. Typically they are used in LOTOS to affect choices. The LOTOS use of 'internal event' should not be confused with, say, internal event triggers in guideline models like GLEE (Guideline Execution Engine [18]). Equally, the LOTOS use of 'choice' for alternatives does not mean a 'decision' in a decision tree sense.

Every node may have an associated visibility condition. Section 4.1.3 explains that such checks are translated as separate processes. Because these may contain internal events (caused by **Exit**), they may not be used in the context of a LOTOS choice since an internal event determines which branch is taken. Instead, visibility checks for children of a chance node must be performed at the start of a chance node so that all alternatives are possible. The results of these checks are used in calls of the *visible* operation.

4.2.5 Question Nodes

While exploring a tree, the user can move forwards (making choices) and backwards (reversing choices). The user can choose to skip a question (or a series of them). If a question is revisited, the user can preserve the previous answer or change it. The formalisation faithfully respects these aspects because they are a likely source of errors in the tree design.

A correctly designed tree will allow the user to explore it interactively, to backtrack, and to change answers to questions. An incorrectly designed tree may fail to take account of this, leading to incorrect outcomes. This is particularly a problem if questions are interdependent. CGTs therefore require thorough analysis to check that these kinds of problems do not arise.

Exit from a question node essentially progresses to the next available node. Consider the sample tree in figure 9 with decision nodes (D), question nodes (Q) and terminal nodes (T). Skipping Q3 moves to Q4, while skipping Q4 moves to T5. When at any question node, choosing to skip all questions also moves to T5.

If the user chooses to navigate back from Q4, what now happens depends on the

```
Process Sexual [user] (map:Map) : Exit(map) :=                (* question Sexual *)
  SexualAux [user] (map)                                        (* ask question *)
>> Accept map:Map In                                          (* get new map *)
  (
    [backing(map)] ->                                           (* going back? *)
      TURP [user] (unback(map))                                 (* to TURP, not going back *)
  []                                                            (* or *)
    [not(backing(map))] ->                                      (* not going back? *)
      Let map:Map = unskip(map) In                              (* stop skipping questions *)
        Outcome [user] (map)                                    (* to Outcome *)
  )
Where                                                           (* local definitions *)
  Process SexualAux [user] (map:Map) : Exit(Map) :=             (* auxiliary process *)
    ...
  EndProc                                                       (* end SexualAux *)
EndProc                                                         (* end Sexual *)
```

Fig. 10. Question Node Process for *Sexual*

prior answers. If Q2 was previously skipped, it will be asked again. Choosing to skip Q2 again will cause Q3 to be considered (and ignored if previously answered). If Q2 was previously answered correctly, the user will be asked whether the prior answer should be preserved or changed.

In fact the situation is even more complex. For example, Q2 may be conditionally visible. Suppose Q2 is initially visible so the user is prompted to answer it. However, the answer to Q2 or Q3 may change this visibility. If the user chooses to go back, it may be found that Q2 has effectively disappeared; the next node will be T5. As a concrete example, suppose Q2 initially asks whether the user is willing to answer questions about sexual behaviour. If the user declines, then Q2 can be made invisible to future exploration. This would be an unusual but permissible design for a tree; the semantics must therefore give it a precise interpretation.

It is therefore not surprising that the translation into LOTOS of question nodes is complex. In fact, the documents describing the CGT work [1,2] were found to be rather loose (and even incorrect) in their description of question nodes.

The translation of a typical question node (*Sexual* from figure 5) is shown in figure 10. The real work of a question is performed by an auxiliary process that yields the question answer. Since *SexualAux* requires a page of specification, it is omitted here. If the user decides to backtrack, the question moves to its parent node. Otherwise, the question moves to the next node. The user can choose to skip the current question and others that follow it in a series. When a series of questions ends, skipping is cancelled.

5 Analysing Decision Trees

5.1 Analysing LOTOS Specifications

Developing the strategy for translating ADIT into LOTOS was a valuable exercise in its own right. The resulting semantics give a precise notion of what CGTs mean. In

a number of cases discussed with the CGT team, the meaning of various constructs was unclear or undefined – and sometimes surprising even to the CGT team. One example is what should happen if a node is hidden. The problem is that checking its visibility may have side-effects. These may change how the rest of the tree behaves, even if the node is invisible. Similarly, validity checks and other calculations may have unexpected and unintended consequences.

As noted in section 1.3.1, CGTs have distinctive characteristics (e.g. interactive exploration) that require new forms of analysis. The aim of the formal analysis described here is to find flaws that are particular to CGTs. More conventional analyses are used to discover other kinds of flaws in CGTs and are not discussed here.

Having generated a LOTOS specification for a tree, any kind of formal analysis can be used. In the present context, the goal is to discover the new kinds of flaws that may arise in CGT design. ADIT automates this analysis as far as possible. In fact, the ideal is that the designer just works with the description of a tree – formal analysis would ideally be fully automated. The author has developed several techniques to ease validation and verification, though these still require effort to define appropriate validation tests and verification properties.

MUSTARD (Multiple-Use Scenario Testing and Refusal Description [27]) is designed for automatically validating specifications. Tests are expressed in a neutral language that is independent of the application domain, the specification language, and the validation tool. For example, MUSTARD has been used to validate specifications of voice services, web/grid services and radiotherapy devices. MUSTARD support includes LOTOS and its various toolsets.

PCL (Parameter Constraint Language [26]) deals with the problem that specifications often use variables with infinite sets of values (e.g. a simple number, as in decision trees). This may make formal analysis impracticable or impossible. However, as in conventional software testing, it is usually necessary to check only critical values. Suppose an input must lie in a numerical range. Values just outside the range, just inside it, and in the middle should be tested. PCL allows a LOTOS specification to be annotated to indicate the key values to check. PCL annotations are automatically translated into LOTOS and used to constrain the analysis.

5.2 Analysing Clinical Guidance Trees

The kinds of errors that can be made in a CGT fall into the following categories:

Syntactic errors mean the tree is badly formed. For example, a terminal node may have children or a question node may not define the answer format.

Static semantic errors mean the tree description is well formed but can be found to be incorrect without traversing it. For example, an expression may use an uninitialised variable or a variable may be undeclared.

Path errors mean that exploring the tree leads to problems. For example, a question may be asked indefinitely or a node may be unreachable.

Numeric errors mean that calculations are incorrect. For example, probabilities do not sum to 1 or payoffs are wrongly determined.

Explanation errors mean that explanations given to the user are wrong, incomplete or misleading. For example, a medical condition might be incorrectly explained or the consequences of a treatment might be incompletely described.

Because of the level of abstraction chosen for formalising decision trees, numeric errors and explanation errors are currently not covered in the analysis (though detection of numeric errors will be undertaken in future). These aspects fall within the definition of tree content, which is not the current focus in analysing CGTs. The other kinds of error are concerned with the tree structure and flow, and so are checked. Since CGTs allow interactive exploration, the principal focus is on path errors. This requires the dynamic behaviour of the tree to be analysed, i.e. the state space of its specification to be checked. The analysis is performed automatically to uncover the kinds of flaws that a user might meet during actual exploration.

The ADIT approach was used on the healthcare studies developed by the CGT project. These have been briefly described in earlier sections. In particular, section 3.5 describes how these applications were developed by the CGT team, and what quality assurance procedures were used. Note that these applications had *already* been thoroughly evaluated by medical and computing professionals, had been informally evaluated by patients, and (in two cases) had been used in trials.

The analysis reported in this paper was therefore performed after the fact. This was therefore a severe test of the new work. If formal analysis could still find errors in thoroughly evaluated trees, this would give confidence that it would prove even more useful when developing new trees from scratch.

ADIT tree descriptions were automatically generated from four healthcare CGTs: benign prostatic hyperplasia, influenza, hypertension and menorrhagia. The analytic techniques described in section 5.1 were applied to these. The ADIT translators detect syntactic and static semantic errors automatically. The checks for dynamic flaws were focused on path errors, such as those arising from incorrect validity or visibility checks. The following techniques were used to detect path errors.

State space exploration was used to assess general classes of error. Deadlocks can occur if a node should be visible but is not; the effect is that certain leaf nodes become unreachable. Although livelock is not strictly possible in CGTs, infinite loops can arise if a validation condition is incorrect and causes a question to be asked repeatedly. If the user is allowed to backtrack in the tree, its behaviour becomes infinite. Although backtracking can be omitted through a translation option, it is normally enabled as it allows detection of certain validation or visibility errors.

Property checking was used to assess particular types of error. For example, if the question about sexual activity is correctly answered then this question should not be repeated. Conversely, an incorrect answer must lead to immediate repetition of the question. If the user is sexually active, then the possibility of sexual problems following surgery must be a permitted option (and vice versa). Properties like these were formulated using either specific LOTOS test processes or XTL (Extended Temporal Logic [16]).

Like other temporal logics, XTL is used to define properties that must hold for all dynamic behaviours of a system (i.e. during interactive exploration of a CGT). Temporal properties fall into certain broad classes. Safety properties state what a system must never do (e.g. if the user is not sexually active, then sexual problems must not be presented as a possible problem). Liveness properties state what a system must eventually do (e.g. if the user chooses to undergo TURP surgery, then the outcomes of this must eventually be presented). Fairness properties ensure that all permitted behaviours of a system do occur (e.g. exploring the TURP branch does not lead to exclusion of the phytotherapy branch). In addition to generic properties like these, application-specific properties were formulated (e.g. that the best payoff for BPH results from choosing TURP).

The following categories of errors were found through formal analysis of the four healthcare studies. The ADIT tree descriptions were obtained from the original CGT trees through automatic translation, so these errors were also present in the originals.

Structural errors A number of cases were found where a sub-tree could not be explored because a question could never be answered correctly or the sub-tree was always invisible. In terms of formal analysis, there were unreachable states. These errors were corrected by eliminating sub-trees during translation (because they were intentionally dormant) or by correcting their conditions. Structural errors are detected either during translation (e.g. a question that is not followed by anything) or when checking the static semantics of the translated specification (e.g. nodes processes are technically inconsistent). The original CGT trees had already been thoroughly checked, so complex semantic errors were not found in the work reported here. But it is anticipated that in new developments it will be valuable to have significant semantic errors detected automatically.

Initialisation errors Variables may be initialised in the wrong order. For example *probUnwell* was initialised to *1 - probWell*, but initialisation of *probWell* came later. Such errors result in failure during exploration due to undefined variables.

Macro errors Macros may be used only in text values, yet in several cases they were used in numeric calculations (e.g. of visibility). Such errors result in failure during exploration due to invalid calculations.

Condition errors Validity and visibility conditions should yield *true* or *false*. In several cases, however, conditions were found to end with an assignment such as *probNoSideEffects = revisedRisk * (1 - probSideEffects)*. The problem here is a subtle one. In most cases, *probNoSideEffects* will be assigned a non-zero value which counts as *true*. However if it is assigned a zero value then the condition will yield *false*, leading to an invalid or invisible result. This results in seemingly random behaviour during exploration, depending on the exact values for certain variables that are set dynamically.

Missing range checks All free-form inputs should be checked for validity as they are just numbers. In a number of cases, no range check was given. This allowed meaningless values such as a negative value or 200 for an age. Such errors result in the user being able to enter meaningless values and receiving erroneous advice

based on these.

Incorrect range checks Some range checks were incorrect. For example, blood pressure inputs should be checked for validity. In one case the intention was to check that systolic blood pressure was in the range 100 to 220 mm Hg inclusive. However, the range check was for over 99 and less than 221. Such errors mean the user can enter fractional values outside the intended range, conceivably leading to incorrect advice being offered.

Besides these technical errors, explanation errors were also found as a by-product of the formalisation. There were small editorial problems such as formatting errors and small technical errors.

The trees had already been thoroughly evaluated by the CGT team, by patients informally, and through RCTs. Nonetheless, the formal analysis reported here found a number of problems. Some flaws would cause run-time exceptions in the CGT Viewer. Unless the user is technically minded and checks the Viewer log, these errors would go unnoticed. Some flaws could cause erratic behaviour or result in incorrect advice from the CGT Viewer. Particularly for trees of a critical nature, flaws like these are important. Supplementing manual debugging with automated analysis is thus beneficial.

It is believed that the work reported in this paper will be of value in future development of CGTs. There is now a rigorous methodology for defining and verifying CGTs. The separation of concerns into design of tree structure and tree content supports manageable development of each aspect. The abstract tree notation allows a single description to be used for multiple purposes: design, formal analysis, interchange with other tools, and implementation. The new techniques are also complementary to existing approaches for design of medical decision trees.

6 Conclusions

6.1 Evaluation

The work described in this paper has achieved a number of important goals:

Abstraction The aim was to define an abstract notation for CGTs that allows structure to be separated from content. ADIT allows content to be defined by means of separate attributes that are referenced from the tree structure. This also allows the tree structure to be defined and investigated in advance of defining the main content. Once the key aspects of tree flow and exploration have been verified, the designer can add content to fill out the tree.

Formalisation Another important objective was to formally specify and analyse CGTs. The ADIT notation is automatically translated into LOTOS, opening up many possibilities for validating and verifying tree behaviour. This allowed the author to find a number of problems in the trees developed by the CGT project.

Implementation It was also important to use the same tree description to create implementations as well as formalisations. The ADIT notation is automatically translated into the format used by the CGT Viewer, allowing a tree description

to be evaluated by theoretical means and also explored by practical means.

Application Finally, the approach has been demonstrated to be useful on some realistic trees. These trees provide advice to lay users on some important medical conditions. Through automated conversion from healthcare examples developed by the CGT project, a range of sizable studies has been conducted.

In the development of an abstract and formal model of CGTs, a number of subtle issues have been clarified. Indeed, ADIT now gives a denotational semantics for these kinds of decision tree. A number of errors were also found in realistic medical trees through formal analysis.

Although ADIT has largely been used with healthcare applications so far, it is certainly not restricted to these. For example, applications in data mining, finance and risk assessment can readily be imagined. The notation is abstract, and therefore independent of the application domain and the decision tree tool.

For the benefit of the community, the ADIT tools, examples and CGT Viewer have been made available at *http://www.cs.stir.ac.uk/~kjt/research/adit.html*.

6.2 Future Work

So far, ADIT has been used retrospectively on trees that were already developed and thoroughly tested. However, the existence of a rigorous methodology can now be exploited in new developments. For example, the development of CGTs for heart disease is planned. Work is also under way to develop CGTs that advise athletes on diet, exercise and training. Although it is believed that ADIT is generic, it is possible that new application domains will suggest further enhancements.

Further work on aspects of verification (proof) would be desirable. So far, formal analysis has mainly focused on testing and on model checking. An interesting avenue to explore would be symbolic verification, where properties are proven in general rather than for specific values of variables. It is also intended to use a probabilistic variant of LOTOS to support the analysis of probabilities and payoffs.

At present, ADIT supports translation only to the implementation format supported by the CGT Viewer. Translation will be investigated to/from the proprietary formats of commercial decision support tools (e.g. PRODIGY or PRO*forma* in healthcare).

Acknowledgements

The CGT project was funded by the Chief Scientist's Office in Scotland from March 2000 to June 2003. Richard Bland was the chief designer of the CGT system, Claire Beechey was the main implementer, and Dawn Dowding (now at the University of York) was the driving force behind the project. The clinical guidance trees were mainly designed by Pat Thomson and Claire Beechey (University of Stirling), Chris Mair (Forth Valley Primary Care Research Group), and Joanne Protheroe (University of Manchester).

The author is very grateful to Richard Bland and Claire Beechey for their help during the new work reported in this paper. They provided the CGT software and

examples used in this work, and explained various subtleties of the CGT notation.

References

[1] R. Bland, C. E. Beechey, and D. Dowding. Extending the model of the decision tree. Technical Report CSM-162, Department of Computing Science and Mathematics, University of Stirling, UK, Aug. 2002.

[2] R. Bland, C. E. Beechey, and D. Dowding. Writing decision trees for the CGT viewer. Technical Report CSM-163, Department of Computing Science and Mathematics, University of Stirling, UK, Oct. 2003.

[3] T. Bolognesi and E. Brinksma. Introduction to the ISO specification language LOTOS. *Computer Networks*, 14(1):25–59, Jan. 1988.

[4] U. Brandes, M. Eiglsperger, I. Herman, M. Himsolt, and M. S. Marshall. GraphML progress report: Structural layer proposal. In H.-J. Kugler, editor, *Proc. 9th International Symposium Graph Drawing*, number 2265 in Lecture Notes in Computer Science, pages 501–512. Springer, Berlin, Germany, 2002.

[5] G. B. Chapman and F. A. Sonnenberg. *Decision Making in Health Care*. Cambridge University Press, UK, 2000.

[6] G. C. Critchfield and K. E. Willard. Probabilistic analysis of decision trees using Monte Carlo simulation. *Medical Decision Making*, 6(2):85–92, 1986.

[7] Department of Health. Choosing health: Making healthy choices easier. Department of Health, London, UK, 2004.

[8] A. S. Detsky, G. Naglie, M. D. Krahn, D. Naimark, and D. A. Redelmeier. Primer on medical decision analysis: Part 2 – Building A tree. *Medical Decision Making*, 17(2):126–135, Apr. 1997.

[9] A. S. Detsky, G. Naglie, M. D. Krahn, D. Naimark, and D. A. Redelmeier. Primer on medical decision analysis: Part 4 – Analyzing the model and interpreting the results. *Medical Decision Making*, 17(2):142–151, Apr. 1997.

[10] J. Dowie. Decision analysis in guideline development and clinical practice. In H. K. Selbmann, editor, *Guidelines in Health Care*, pages 162–193. Nomos Verlagsgesellschaft, Baden-Baden, Germany, 1998.

[11] J. Dowie. What decision analysis can offer the clinical decision maker. *Hormone Research*, 51(Supplement 1):73–82, June 1999.

[12] J. Fox and R. Thomson. Decision support and disease management: A logic engineering approach. *Information Technology in Biomedicine*, 2(4):217–228, Dec. 1998.

[13] K. Friedman. *The Decision Tree*. Heart Publishing, Rainier, Washington, USA, 1996.

[14] M. Helfand. Information for authors. *Medical Decision Making*, 27(4):508–510, July 2007.

[15] ISO/IEC. *Information Processing Systems – Open Systems Interconnection – LOTOS – A Formal Description Technique based on the Temporal Ordering of Observational Behaviour*. ISO/IEC 8807. International Organization for Standardization, Geneva, Switzerland, 1989.

[16] R. Mateescu and H. Garavel. XTL: A meta-language and tool for temporal logic model-checking. In T. Margaria, editor, *Proc. International Workshop on Software Tools for Technology Transfer*, July 1998.

[17] M. A. Musen, S. W. Tu, A. K. Das, and Y. Shahar. EON: A component-based approach to automation of protocol-directed therapy. *J. American Medical Informatics Association*, 3(6):367–388, Nov. 1996.

[18] M. Peleg, O. Ogunyemi, S. W. Tu, A. A. Boxwala, Q. Zeng, R. A. Greenes, and E. H. Shortliffe. Using features of Arden syntax with object-oriented medical data models for guideline modeling. In *Proc. American Medical Informatics Association Symposium*, pages 523–537. American Medical Informatics Association, Los Angeles, USA, Oct. 2001.

[19] M. Peleg, S. W. Tu, J. Bury, P. Ciccarese, J. Fox, R. A. Greenes, R. Hall, P. D. Johnson, N. Jones, A. Kumar, S. Miksch, S. Quaglini, A. Seyfang, E. H. Shortliffe, and M. Stefanelli. Comparing computer-interpretable guideline models: A case-study approach. *J. American Medical Informatics Association*, 10(1):52–68, Jan. 2003.

[20] M. Pignone. Incorporating decision analysis in decision aids. *Medical Decision Making*, 27(5):547–549, Sept. 2007.

[21] J. Protheroe, P. Bower, C. Chew-Graham, T. J. Peters, and T. Fahey. Effectiveness of a computerized decision aid in primary care on decision making and quality of life in menorrhagia: Results of the MENTIP randomized controlled trial. *Medical Decision Making*, 27(5):575–584, Sept. 2007.

[22] Y. Shahar, S. Miksch, and P. Johnson. The Asgaard project: A task-specific framework for the application and critiquing of time-oriented clinical guidelines. *Artificial Intelligence in Medicine*, 14(1-2):29–51, Sept. 1998.

[23] F. A. Sonnenberg and J. R. Beck. Markov models in medical decision making: A practical guide. *Medical Decision Making*, 13(4):322–338, Oct. 1993.

[24] D. R. Sutton and J. Fox. The syntax and semantics of the PRO*forma* guideline modelling language. *J. American Medical Informatics Association*, 10(5):433–443, Sept. 2003.

[25] P. Thomson, D. Dowding, V. Swanson, R. Bland, C. Mair, A. Morrison, A. Taylor, C. Beechey, and C. A. Niven. A computerised guidance tree (decision aid) for hypertension, based on decision analysis: Development and preliminary evaluation. *European J. Cardiovascular Nursing*, 5(2):146–149, June 2006.

[26] K. J. Turner. Test generation for radiotherapy accelerators. *Software Tools for Technology Transfer*, 7(4):361–375, Aug. 2005.

[27] K. J. Turner. Validating feature-based specifications. *Software Practice and Experience*, 36(10):999–1027, Aug. 2006.

[28] M. P. Wellman, M. H. Eckman, C. Fleming, S. L. Marshall, F. A. Sonnenberg, and S. G. Pauker. Automated critiquing of medical decision trees. *Medical Decision Making*, 9(4):272–284, Oct. 1989.