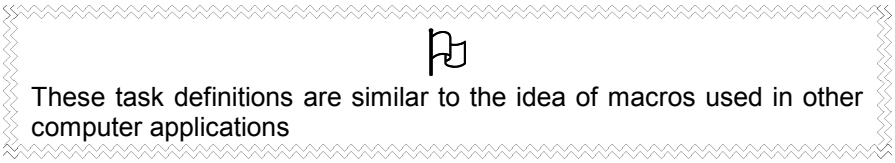
CHAPTER NINE - REASONING WITH AKT TOOLS

9.1 INTRODUCTION

An important part of the AKT application is the task language. This consists of a number of primitive or system tools together with a choice of control structures in which the tools can be combined to perform tasks defined by the user. The resulting definition can then be saved as a user-defined tool for future use. The tools defined in this way can also be incorporated into more complicated tools so allowing complex tasks to be completed automatically.



9.1.1 WHAT IS A TOOL?

A tool is created by writing a 'definition'. A definition is a set of 'functions'[1] linked with appropriate punctuation and control structures and entered into the definition field.

A function calls the underlying program code to perform a specified task. Some functions are part of inference mechanisms that undertake automated reasoning tasks on the knowledge base, others are concerned with the management of inputs and outputs (interfacing) to and from inference mechanisms. Functions are defined by a name one or more arguments (variables) associated with it.

9.1.2 WHAT DO THE TOOLS DO?

The task language allows the user to interrogate the knowledge base and carry out assessments of the knowledge it contains. These can be simple queries such as;

How many conditional statements are there?
How many statements are not represented in the diagram?
What comparison statements are there?

or a more sophisticated analysis of the knowledge base, for example;

What are the immediate causes of soil erosion?
What are all the effects of shading?
Using hierarchical information, which livestock are suitable for small landholdings?

9.1.2.a What is a primitive?

Primitives are small program segments, pieces of generic code, from which a tool can be constructed. They are classified into a number of categories, Cause and Effect, Display, Formal Term, Hierarchies, List, Miscellaneous, Sources, Statement and Test. These primitives can carry out searches on the knowledge base, handle lists and manipulate the formal terms and statements.

[1] A function is a generic term that could represent either a primitive, system tool or a user defined tool

9.1.2.b What is a control structure?

Control structures define special relationships between a set of functions, thereby allowing different actions to be taken depending on the circumstances and allowing repetitive tasks to be specified efficiently. Control structures allow primitives to be 'threaded' together in 'if_then_else'; 'repeat_until' or 'foreach_in_do' type of structures to create tools, which can carry out more complex tasks that a single primitive might achieve.

9.1.2.c What is a tool?

A tool is a collection of primitives together with control information defining the sequence in which they should be carried out. They are classified as;

a) ***Systems tools*** - tools supplied with the AKT application that have been constructed using primitives and control structures to perform some generally useful function which is far too complex for a simple primitive.

b) ***User tools –*** tools created by you, the user, using the primitives, systems tools and control structures provided, to carry out tasks not previously envisaged by the programmers. Once defined, user tools themselves can be used as primitives to create larger tools.

This chapter looks at tools and the mechanics behind the primitives and systems tools. Chapter 10 looks at user tools and teaches you how to create your own.

9.2 WORKING WITH TOOLS

To display the tools provided, select **Tools** from the main **Tools** menu. Then, from the drop down menu, select the type of tool you want:

- Control structures
- Primitives
- System tools
- User tools

9.2.1 PRIMITIVES

9.2.1.a Opening primitives

In order to work with primitives, select **primitives** from the drop-down menu. When **primitives** has been selected another drop-down menu appears, listing the types of primitives available (Figure 9.1).

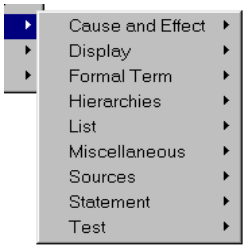


Figure 9.1 *Drop-down menu for primitives*

9.2.1.b Different categories of primitives

The following is an explanation of the different categories of primitive available:

Cause and Effect	Functions for the extraction of information about all types of nodes and the pathways between them.
Display	Functions that allow the user to display the results of running a tool in a text window in various ways. Also allows tools to display messages and request input from the user.
Formal Term	Functions for the manipulation of formal terms.
Hierarchies	Functions for extracting information about objects within object hierarchies.
List	Provide a variety of functions for operating on lists such as sorting, appending and combining lists.
Miscellaneous	These are assorted primitives that do not fall into any of the above categories.
Sources	Function for examining the details of any sources in the knowledge base.
Statement	Functions that operate on statements. Allows searching and manipulation of any statements which exist within the AKT environment.
Test	Functions that allow rue/false checks to be made and a way of aborting the running of a tool

Generally primitives are not run in isolation but are used within tools. Some primitives however can be run independently and give useful results.

All primitives and system tools have individual descriptions of their functions and parameters together with an example of their use available via the **Details** button on the category dialog box. A full list of the tools, including the same details as above, are also available via the Help/ToolList menu. This list incorporates all primitives system and any user tools.

9.2.1.c Understanding the Details box

In order to demonstrate the 'Tool Details' dialog box, we have selected a primitive, ***formal_term(KB, Term, Type, Result)*** from the **Formal Term** category of primitives. Click on this tool, and the tool details will automatically appear (*Figure 9.2*)

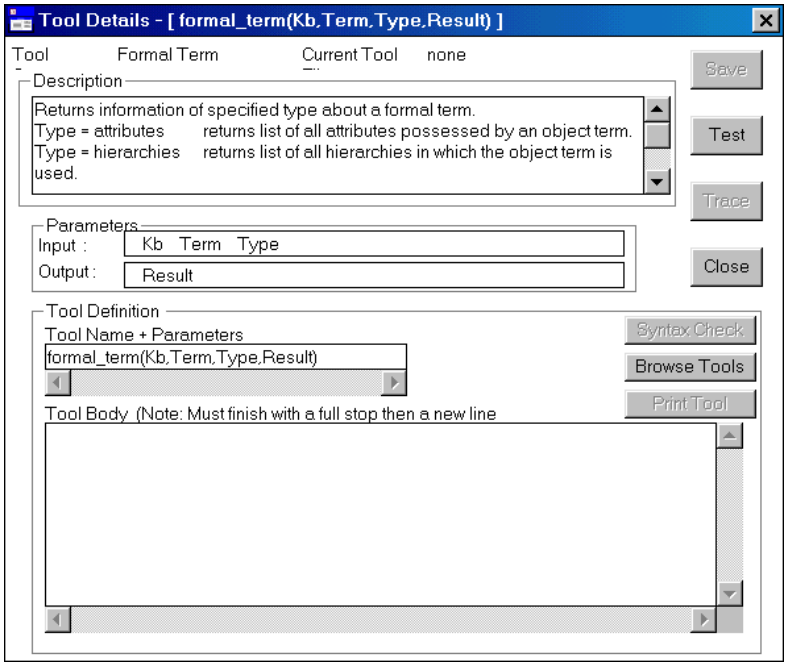


Figure 9.2 *'Tool Details' dialog box of the primitive* ***formal_term(Kb, Term, Type, Result)***

The dialog box title shows the tool category, in this case 'Formal Term', and the current tool file if a user tool file is loaded (in this case there isn't).

Taking the output dialog box apart bit by bit, you have:

Description
Here the function of the tool is described together with a relevant example. Below is the information provided in the Description box for the tool **formal_term(Kb,Term,Type, Result):**

Returns information of specified type about a formal term.	
Type = attributes	returns list of all attributes possessed by an object term.
Type = hierarchies	returns list of all hierarchies in which the object term is used.
Type = synonyms	returns list of all the synonyms used for a formal term.
Type = type	returns the type of a formal term.
Type = use	returns 'used' or 'unused' depending whether term is used by a statement.
Type = values	returns list of all the values assigned to an attribute formal term.
eg:	formal_term(treefodd,erosion,type,Type) will return **Type=process** formal_term(treefodd,area,values,Values) will return **Values=[decrease,increase]**.

Parameters
This shows the parameters that are required as inputs to the tool and the parameters that the tool uses to return information. In the case of the tool formal_term(KB, Term, Type, Result) the parameters required as inputs are:-

- the name of the knowledge base
- the term (the object) to be investigated
- the type of output sought (i.e. attributes or hierarchies)

The output parameter will be:

- the result (i.e. the attributes or hierarchies sought)

In the 'Tool Definition' box the name of the tool and its parameters (in brackets) are shown. The name of the tool must always begin with a **lower case letter** or be enclosed in single quotes eg: 'JohnsUserTool' if it is necessary to use a capital letter for a real name. The name of each parameter listed after the tool name must always begin with a **capital letter** to denote a *variable*. If the parameter is an *atom*, the first letter will either be in lower case (e.g. *soil*) or it will be enclosed in single quotes for proper names (e.g. 'Nepal').

The 'Tool Body' box in this instance remains blank, because the tool formal_term is a primitive and cannot be broken down into smaller parts. The systems tools and user tools will have details of the body of the tools showing the chain of primitives and control structures (and possibly simpler systems tools and user tools) from which the current tool is constructed.

On the right hand side of the 'Tool Details' box are three active buttons, **Test**, **Close** and **Browse Tools** for primitives and system tools. The remaining buttons will only be active for User tools.
Browse Tools allows the user to browse any tool and will be dealt with in Chapter 10, below.
The **Close** button removes the 'Tool Details' dialog box
The **Test** button allows you to run an existing tool or try out a new user tool.

If you press **Test**, a dialog box appears listing all the input parameters which need to be filled in. In this case, we are searching for the attributes of the object 'badahar' (a Nepalese tree), in the knowledge base 'treefodd'. The input parameters are entered into the template (Figure 9.3). Then press **Continue**. The 'Tool output' dialog box then appears (Figure 9.4). Under 'Result' are the two attributes describing the tree badahur, height and growth_rate.

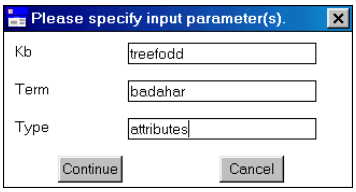


Figure 9.3 *Dialog box with input parameters filled in for the tool* ***formal_terms(Kb, Term, Type, Result)***

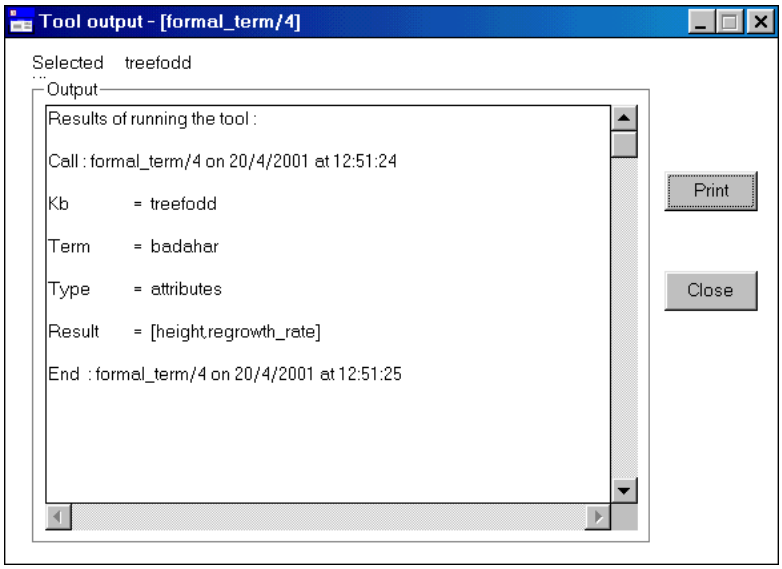


Figure 9.4 *Tool Output box for the* ***primitive formal_terms(Kb, Term, Type, Result)*** *used on the knowledge base 'treefodd'.*

(In the 'Tool output' dialog box, it gives the tool name as formal_term/4. This is merely a shortened form, the '/4' referring to the number of parameters (input and output) associated with the tool).

This time try out the tool for real. Press **Close** on the 'Tool output' dialog box and **Close** on 'Tool Details' dialog box. The screen will revert to the 'Tools' dialog box. The tool should still be highlighted in the list of tools. Press **Run**. A dialog box appears requiring you to fill in the input parameters. Fill it out, as in Figure 9.3 but this time instead of entering *attributes* in the 'Type' box, enter *hierarchies*. Press **Continue**. The 'Tool output' dialog box appears, giving a list of all the hierarchies to which the tree badahar belongs:

All_trees;
Fodder_animal_type;
Fodder_overripening_month;

Fodder_quality;
Fodder_ripening_month;
Leaf_flushing_month;
Leaf_shedding_month;
Posilo_kam_posilo_fodder_trees;
crop_land_trees

Although the user has the option to run any tool including the primitives as shown above, normally only system or user tools will be run like this, primitives tend to be used within the bodies of other tools.

9.2.2 CONTROL STRUCTURES

There are six control structures available in the AKT task language for use in the creation of reasoning tools. To view the available control structures, select the control structures from the drop down menu under Tools in the main Tools menu.

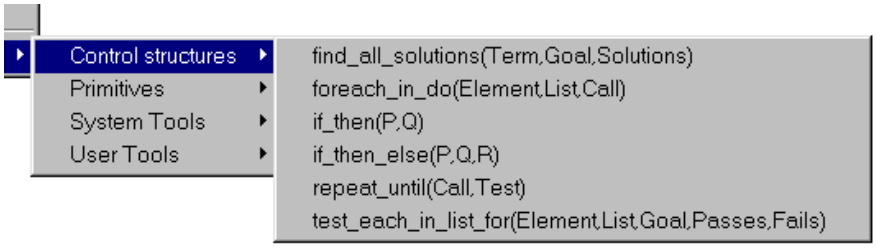


Figure 9.5 Drop-down menu for control structures

Control structures are pieces of code, which are used to link one primitive to another, to construct a tool – they are, if you like, the 'glue' that binds the primitives together.

Name of control structure	Function of control structure
Find..all..solutions(Term, Goal, Solutions)	This structure allows you to find all solutions to a specified goal. The goal can be a primitive, a system or a user tool or any combination of them.
Foreach..in..do(Element, List, Call)	The operation detailed in Call (or Goal) is executed in turn for every item in the specified list
If..then(P, Q)	In this structure 'P' is a primitive or tool or any combination of primitives and tools, which returns a response 'true' or 'false'. If the response is *true* then 'Q' (a primitive, tool or a combination of both) is invoked before continuing with the next primitive in the tool definition. If the response is *false* then 'Q' is ignored and the tool continues with the next primitive in the definition.
If..then..else(P,Q,R)	It is possible to elaborate on the above tool by using the if_then_else control structure. This is similar to the if_then structure except that, if the test fails, the 'else' call is implemented rather than simply exiting the control structure tool.
Repeat..until(Call, Test)	This control structure enables you to repeat a Call (i.e. an instruction or chain of instructions) until a Test is successful.

Test..each..in..list..for(Element, List, Goal, Passes, Fails)	This control structure tests each item in a list for a specified condition and returns two lists, one containing the items that meet the condition and one containing the items that do not meet the condition.

* a 'call' is goal consisting of an instruction or a sequence of instructions where an instruction can be a primitive or system or user tool

9.3 SYSTEM TOOLS

System tools are the tools already created and supplied within the AKT program. If you select **System Tools** from the 'Tools' dialog box the following drop-down menu appears (Figure 9.6):

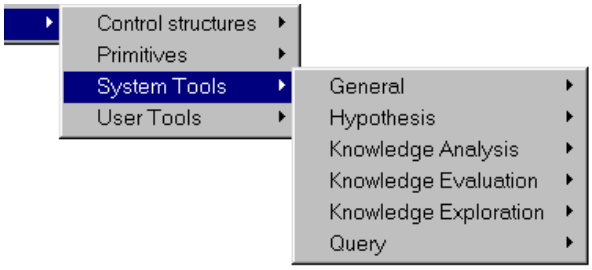


Figure 9.6 *The drop-down menu for System Tools*

The following is an explanation of the different categories of systems tool available:

General	These are general systems tools not incorporated into the other categories
Knowledge Analysis	Analyses the effect(s) of a statement or formal term(s) on the knowledge base
Knowledge Evaluation	Evaluates the contents of the knowledge bases and checks for example inconsistencies or redundant information within the knowledge bases
Knowledge Exploration	Explores the knowledge base for the implications of processes and management actions
Query	Allows the user to determine if anything in the knowledge base matches any type of formal statement specified by the user

9.3.1 SOME EXAMPLES OF SYSTEMS TOOLS

a) System Tool Category: Knowledge Evaluation
System Tool: kb_report/1

The System Tool kb_report/1 is in the category Knowledge Evaluation (Figure 9.7);

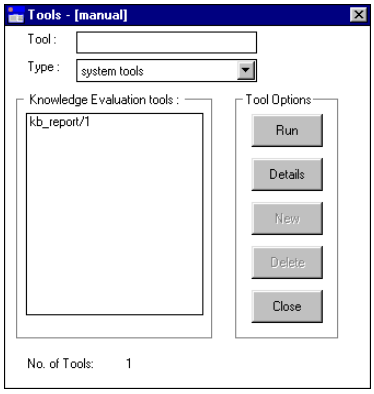


Figure 9.7 *The dialog box for Knowledge Evaluation System Tools*

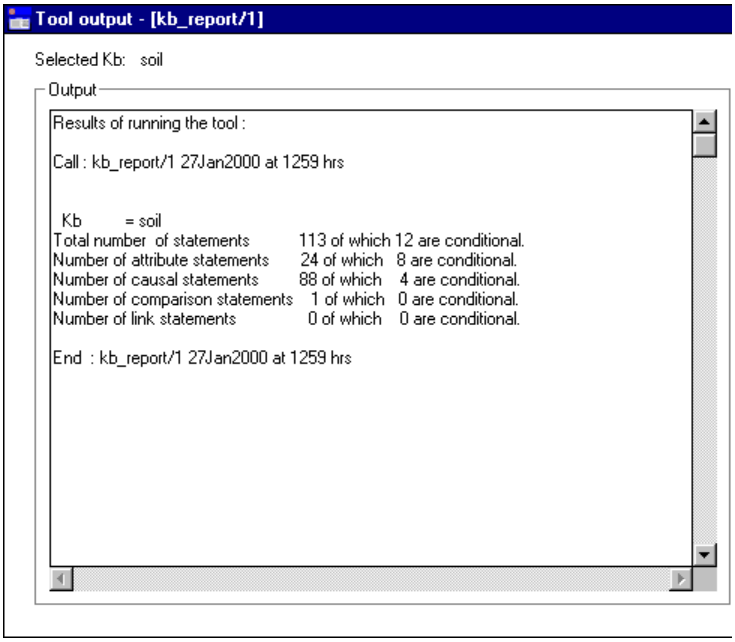


Figure 9.8 *Output screen for **kb_report/1** on the 'soil' knowledge base*

This is a simple system tool requiring you only to press **Run** and enter the name of the knowledge base to be interrogated**.** Alternatively it can be incorporated within a user tool as just part of a more extensive user report.
Running the tool gives a report (Figure 9.8) on how many statements within the specified knowledge base fall into the following categories:

- All statements of which n number are conditional
- Simple attribute value statements of which n number are conditional
- Causal statements of which n number are conditional
- Comparison statements of which n number are conditional
- (User defined) Link statements of which n number are conditional

b) System tool category:Query
statement_query(Kb): This tool allows the user to analyse the knowledge base by submitting a query of arbitrary complexity. The query is specified in the form of a formal statement that conforms to the AKT grammar, for example, typing
att_value(Ob1,Att1,Val1) causes2way att_value(Ob2,Att2,Val2) in the query box will allow the user to find all the causal statements in the knowledge base that match the query.

In the example below (Figure 9.9 and 9.10) all the 'soil' knowledge base statements have been examined and any attribute phrases matching the AKT expression att_value(O,A,V) have been found.

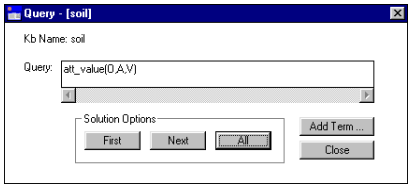


Figure 9.9 *Using statement_query.0 to find all attribute value phrases of a certain format*

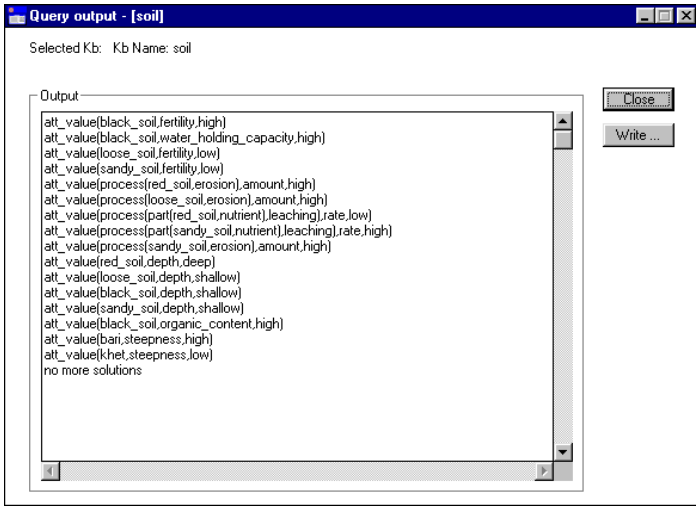


Figure 9.10 *The output window for the statement query: att_value(O,A,V)*

c) System tool category:General

merge_External_Statements: This tool enables you to merge statements from an external knowledge base into the current knowledge base. If you select this tool and press **Run**, a directory appears, with the legend 'Kb containing the statements to be merged?' from which you select the knowledge base to be integrated into current knowledge base.

Although this tool makes the merging of two knowledge bases extremely easy, it should be used with caution. Only if the definitions of formal terms used in both knowledge bases are identical is it possible to merge the two knowledge bases without fear of misunderstandings or internal contradictions arising.