# Linguistic and Grammatical Models

Another category of Cognitive Models includes Linguistic and Grammatical models.

These use formalisms such as BNF (Backus-Naur Form) to describe interactions.

The intention is to represent the cognitive difficulty of the interface so that it can be analysed.

## Backus-Naur Form

BNF can be used to define the syntax of a language.

It is based on techniques developed for use with natural languages, but was specifically designed for use with computing languages.

BNF defines a language in terms of *Terminal symbols*, *Syntactic constructs* and *productions*.

*Terminal symbols* are the elementary symbols of the language, such as words and punctuation marks.

In the case of computing languages, these may be variable-names, operators, reserved words, etc..

*Syntactic constructs* (or *non-terminal symbols*) are phrases, sentences, etc.

In the case of computing languages, these may be conditions, statements, programs, etc..

Syntactic constructs are built from:

* *terminal symbols*, and/or
* other (previously-defined) *syntactic constructs*

This is done using a set of rules known as *productions*

Reisner (1981) proposed the use of BNF to describe the dialog grammar of a system.

This describes the syntax of the language used in the interaction, but not its semantics.

For example, consider a graphics application that has a 'polyline' function.

The user selects the function from a menu, clicks at each of the points the line is to link, then double-clicks to finish drawing the line.

This interaction might be described as follows:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|   | draw-line | ::= |   | select-line + choose-points + last-point |
|   |
|   | select-line | ::= |   | position-mouse + CLICK-MOUSE |
|   |
|   | choose-points | ::= |   | choose-one |
|   | | | choose-one + choose-points |
|   |
|   | choose-one | ::= |   | position-mouse + CLICK-MOUSE |
|   |
|   | last-point | ::= |   | position-mouse + DOUBLE-CLICK-MOUSE |
|   |
|   | position-mouse | ::= |   | empty | MOVE-MOUSE + position-mouse |

Note that:

* Terminal symbols are shown in upper case.
* Syntactic constructs are shown in lower case.
* Only syntactic constructs may appear on the left side of a definition.
* Syntactic constructs are defined in terms of terminal symbols and/or other syntactic constructs using the symbol ::= which means 'is defined as'.
* The symbol + indicates items performed in sequence
* The symbol | indicates a choice.

It can be seen that:

* All the syntactic constructs in the description are defined in terms of:
	+ terminal symbols (e.g., MOVE-MOUSE) and/or
	+ other syntactic constructs which are defined further down the description.
* Some syntactic constructs are recursive.

For example, position-mouse may be either empty (meaning 'do nothing') or involve a MOVE-MOUSE followed by another position-mouse.

Thus position-mouse can be any number of mouse-moves.

Once an interaction has been described using BNF, its complexity can be measured in various ways.

One measure is simply to count the rules; the more rules, the greater the complexity.

However, this method depends on the precise formulation of the rules.

It is usually possible to reduce the number of rules needed to describe an interaction by using complex rather than simple rules (e.g., by inserting more choices into the rules).

A better measure is to count both the rules and the + and | operators in the description.

This yields a more reliable measure than merely counting the rules, but is still dependent on the framing of the rules.

It is usually possible to minimise the count for a given interaction by 'trading-off' complex rules against simpler rules.

In practice, these problems are of little concern provided the BNF descriptions are generated by the same person or team, and pains are taken to ensure consistency in their generation.

## Task Action Grammar

A failing of BNF-based measures is that they ignore many features of language and the user's knowledge.

For example, consider a BNF description of the UNIX commands cp, mv and ln:

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
|   | copy | ::= |   | 'cp' | + | filename | + | filename |
|   |   |   | | | 'cp' | + | filenames | + | directory |
|   |
|   | move | ::= |   | 'mv' | + | filename | + | filename |
|   |   |   | | | 'mv' | + | filenames | + | directory |
|   |
|   | link | ::= |   | 'ln' | + | filename | + | filename |
|   |   |   | | | 'ln' | + | filenames | + | directory |

The syntax used in these commands is consistent - each accepts either a source filename followed by a destination filename, or a list of source filenames followed by a destination directory.

Now suppose that one of these commands does not share this consistency.

For example, suppose that ln takes its directory argument first.

This would be reflected in a BNF description, but would not affect the number of rules and operators used.

Thus it would not affect measures of complexity derived from the description.

Task Action Grammar or TAG (Payne & Green, 1986) is designed to capture some of the information ignored by BNF-based approaches.

TAG identifies consistency by allowing the use of generic descriptions.

For example, the UNIX commands cp, mv and ln would be described like this:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|   | file-op[Op] | := |   | command[Op] + filename + filename |
|   | | | command[Op] + filenames + directory |
|   |
|   | command[Op=copy] | := |   | 'cp' |
|   | command[Op=move] | := |   | 'mv' |
|   | command[Op=link] | := |   | 'ln' |

This approach ensures that the greater the consistency in the commands, the smaller the number of rules used, thus providing a more reliable measure of complexity.

Successful operation of an interface often relies on general knowledge and previously-learned skills.

For example, in order to use many interfaces it's necessary to understand concepts such as 'up', 'down', etc..

It can be argued that an interface which exploits existing skills and knowledge will be easier to learn and use than one which requires the acquisition of new skills and knowledge.

However, most modelling languages take no account of such knowledge.

In TAG, all labels, named operations, etc., must be declared either as new knowledge or as a "known-item".

Determining what is and what is not a "known-item" is left to the TAG user. Thus the categorisation may be subjective.

However, forcing TAG users to make such decisions does make them aware of the issue in a way that is not necessarily the case with other modelling languages.

## Problems with Cognitive Models

Most cognitive modelling languages and techniques are based on ideas and approaches developed for use with command-line interfaces (CLIs) rather than GUIs.

Many were developed from formalisms used to describe natural languages, and later computer languages.

This is true of both Goal and Task Hierarchies and Linguistic and Grammatical models.

Moreover, most early GUIs were merely window-managers for command-line systems.

Since all interactions on such systems were ultimately translated into textual operations, there was little reason to doubt the adequacy of a textual description.

It is only recently that researchers have begun to question the validity of using cognitive modelling languages to analyse interactions within GUIs.

Whilst the functionality of graphical interactions can be described using such languages, it's not clear that the resulting descriptions capture all the factors of importance in the interaction.

For example, many tasks can be carried out more quickly and effectively using a GUI than a command-line interface.

However, if the same functions are available in both the CLI and the GUI, many modelling languages will yield the same description for both.

Thus they will fail to predict the efficiency improvement offered by the GUI.

A particular problem when using grammars to describe graphical interactions is choosing the lowest-level lexical structure or *lexeme*.

* A key-press is a reasonable choice of lexeme: there is little point in decomposing this action into lower-level actions.
* Mouse-movements present more problems. Should the lexeme be:
	+ a complete mouse-movement (e.g., move mouse onto target)?
	+ a lower-level operation (e.g., move mouse one pixel)?

Most grammars represent a mouse-movement as a single action (target-acquisition).

However, it can be demonstrated that users often check and correct the path when moving the mouse more than a short distance.

Thus mouse-movements may consist of several execute-evaluate phases. But how many? Is it necessary to model mouse interactions at a pixel level?

A grammar that fails to take account of these issues will not be able to analyse or predict the efficiency of a graphical interface.