# Cognitive Models

Many models are designed to incorporate some representation of the user's abilities, understanding, knowledge, etc..

The aim is to formalise knowledge gleaned by psychologists so that it can be employed in the design of computer systems.

Cognitive models can be broadly categorised as follows:

* **Hierarchical representations of the user's task and goal structure**

These models deal directly with the issues of formulating tasks and goals.

* **Linguistic and Grammatical models**

These models deal with articulation and translation between the system and the user.

* **Physical and Device-Level models**

These models deal with articulation at the human motor level rather than at higher levels.

Some cognitive models directly embody knowledge about human perception, memory, etc..

Other cognitive models do not embody this knowledge directly, but model interaction in a way that makes it easy to identify processes relying on human perception, memory, etc..

## Goal and Task Hierarchies

Many models are based on goal/task hierarchies.

A top-level goal is defined, then broken-down into a series of smaller goals that can be modelled easily.

A major issue in the design and use of such models is selecting the goals and the appropriate level of *granularity*.

* **What is the top-level goal?** Most tasks form part of larger undertakings, so goals can be defined at many levels.

The choice is often determined by the system being modelled.

* **What is the lowest-level sub-goal?** Should we break down the goals until we reach the level of individual finger and eye-movements?

The approach generally adopted is to identify sub-goals that are routine, learned tasks which do not involve problem-solving.

These are referred to as *unit tasks*.

Most modelling languages leave decisions on granularity to the user.

## GOMS

Probably the best-known and most influential model based on goal/task hierarchies is GOMS (Card, Moran and Newell, 1983).

GOMS stands for Goals, Operators, Methods and Selection.

|  |  |
| --- | --- |
| Goals | These describe what the user wishes to achieve. They are also taken as 'memory points', which are used to evaluate what has been done and to which the user may return if he/she makes an error. |
|   |   |
| Operators | These represent the lowest level of analysis, the basic actions that the user must perform in order to use the system. They may affect the system or the user. |
|   |
| Methods | It may be possible to achieve a goal using any of several alternative sub-goals or sequences of sub-goals. These are known as methods. |
|   |
| Selection | Where a goal may be achieved using several alternative methods, the choice of method is determined by a selection rule. |

For example:

|  |
| --- |
| GOAL: ICONISE-WINDOW |
| . | [select | GOAL: USE-CLOSE-METHOD |
| . |   | . | MOVE-MOUSE-TO-WINDOW-HEADER |
| . |   | . | POP-UP-MENU |
| . |   | . | CLICK-OVER-CLOSE-OPTION |
| . |   | GOAL: USE-HOTKEY-METHOD |
| . |   | . | PRESS HOTKEY ] |
|   |   |   |   |

The word select indicates that a selection rule is used to determine the choice of sub-goal. The selection rule might look like this:

|  |  |
| --- | --- |
| Rule 1: | Use the CLOSE-METHOD unless another rule applies |
| Rule 2: | If the application is WORDSMITH, use the HOTKEY-METHOD |

Note that GOMS, like many models based on goal/task hierarchies, does not take account of error.

Errors *can* be modelled, but this must be done explicitly by the user of the model.

## Cognitive Complexity Theory

CCT was developed by Kieras and Polson in 1985.

It builds on the GOMS approach but aims to provide more predictive power.

CCT has two descriptions which operate in parallel:

* A description of the user's goals, based on a GOMS-like hierarchy but expressed through *production rules*.
* A description of the system state, expressed as *generalised transition networks*, a form of state transition network.

The production rules are expressed in the form:

if *condition* then *action*

where:

* the *condition* is a statement about the contents of working memory.
* the *action* is an elementary action, either internal (e.g., a change in the state of working memory) or external (e.g., a key-press).

A set of production rules might look like this:

|  |  |
| --- | --- |
|   | (SELECT-INSERT-SPACE |
|   | IF (AND | (TEST-GOAL perform unit task) |
|   | (TEST-TEXT task is insert space) |
|   | (NOT (TEST-GOAL insert space) ) |
|   | (NOT (TEST-NOTE executing insert space) ) ) |
|   | THEN ( | (ADD-GOAL insert space) |
|   | (ADD-NOTE executing insert space) |
|   | (LOOK-TEXT task is at %LINE %COL) ) ) |

|  |  |
| --- | --- |
|   | (INSERT-SPACE-DONE |
|   | IF (AND | (TEST-GOAL perform unit task) |
|   | (TEST-NOTE executing insert space) |
|   | (NOT (TEST-GOAL insert space) ) ) |
|   | THEN ( | (DELETE-NOTE executing insert space) |
|   | (DELETE-GOAL perform unit task) |
|   | (UNBIND %LINE %COL) ) ) |

|  |
| --- |
|   |
|   | (INSERT-SPACE-1 |
|   | IF (AND | (TEST-GOAL insert space) |
|   | (NOT (TEST-GOAL move cursor) ) |
|   | (NOT (TEST-CURSOR %LINE %COL) ) ) |
|   | THEN ( | (ADD-GOAL move cursor to %LINE %COL) ) ) |
|   |
|   | (INSERT-SPACE-2 |
|   | IF (AND | (TEST-GOAL insert space) |
|   | (TEST-CURSOR %LINE %COL) ) |
|   | THEN ( | (DO-KEYSTROKE 'I') |
|   | (DO-KEYSTROKE SPACE) |
|   | (DO-KEYSTROKE ESCAPE) |
|   | (DELETE-GOAL insert space) ) ) |

To understand how these rules work, consider the situation in which someone is using the **vi** editor and has just noticed a typing mistake - a missing space between two words.

The contents of working memory (w.m.) at this stage in the process are:

|  |  |
| --- | --- |
|   | (GOAL perform unit task) |
|   | (TEXT task is insert space) |
|   | (TEXT task is at 5 23) |
|   | (CURSOR is at 8 7) |

Note that the user is assumed to be storing information about the goal, the text being worked upon, and the cursor, all in working memory.

The user now selects a production rule, using the conditional tests attached to each rule to find one which is appropriate.

Of the four production rules described earlier, the only one for which all the conditions are satisfied is SELECT-INSERT-SPACE

The conditions for this rule to operate are:

|  |  |  |
| --- | --- | --- |
|   | IF (AND | (TEST-GOAL perform unit task) |
|   | (TEST-TEXT task is insert space) |
|   | (NOT (TEST-GOAL insert space) ) |
|   | (NOT (TEST-NOTE executing insert space) ) ) |

These conditions match the contents of working memory, so the rule *fires* and the actions associated with this rule are carried out.

The actions associated with this rule are:

|  |  |  |
| --- | --- | --- |
|   | THEN ( | (ADD-GOAL insert space) |
|   | (ADD-NOTE executing insert space) |
|   | (LOOK-TEXT task is at %LINE %COL) ) |

The effect of these actions is to change the state of working memory:

|  |  |
| --- | --- |
|   | (GOAL perform unit task) |
|   | (TEXT task is insert space) |
|   | (TEXT task is at 5 23) |
|   | (NOTE executing insert space) |
|   | (GOAL insert space) |
|   | (LINE 5) |
|   | (COL 23) |
|   | (CURSOR is at 8 7) |

The user again selects a production rule.

Of the four production rules described, the only one for which all the conditions are satisfied is INSERT-SPACE-1. The conditions for this production rule are:

|  |  |  |
| --- | --- | --- |
|   | IF (AND | (TEST-GOAL insert space) |
|   | (NOT (TEST-GOAL move cursor) ) |
|   | (NOT (TEST-CURSOR %LINE %COL) ) ) |

These conditions match the contents of working memory, so the rule fires and the associated actions are carried out.

The associated actions are:

|  |  |  |
| --- | --- | --- |
|   | THEN ( | (ADD-GOAL move cursor to %LINE %COL) ) |

This adds the goal move cursor to %LINE %COL to working memory.

At this point, a further rule or rules governing the movement of the cursor would come into operation.

Eventually, a situation would be reached where the cursor is positioned correctly at 5,23.

At this point, the conditions relating to rule INSERT-SPACE-2 would be satisfied. These conditions are:

|  |  |  |
| --- | --- | --- |
|   | IF (AND | (TEST-GOAL insert space) |
|   | (TEST-CURSOR %LINE %COL) ) |

These conditions match the contents of working memory, so the rule fires and the associated actions are carried out:

|  |  |  |
| --- | --- | --- |
|   | THEN ( | (DO-KEYSTROKE 'I') |
|   | (DO-KEYSTROKE SPACE) |
|   | (DO-KEYSTROKE ESCAPE) |
|   | (DELETE-GOAL insert space) ) |

This rule performs the necessary key-strokes to insert the missing space, then deletes the goal insert space from working memory.

Now that this goal has been removed, the conditions for the rule INSERT-SPACE-DONE are satisfied. These conditions are:

|  |  |  |
| --- | --- | --- |
|   | IF (AND | (TEST-GOAL perform unit task) |
|   | (TEST-NOTE executing insert space) |
|   | (NOT (TEST-GOAL insert space) ) |

These conditions match the contents of working memory, so the rule fires and the associated actions are carried out:

|  |  |  |
| --- | --- | --- |
|   | THEN ( | (DELETE-NOTE executing insert space) |
|   | (DELETE-GOAL perform unit task) |
|   | (UNBIND %LINE %COL) ) |

This rule tidies-up working memory, freeing space for the next stage. In particular, it 'unbinds' the variables LINE and COL, removing them from memory.

## CCT - Strengths and Weaknesses

The principal function of CCT is to allow the measurement of complexity in interfaces.

Kieras and Polson claim that complexity of an interface is reflected in the number of production rules required to describe the system using CCT.

The more rules, the harder the interface is to learn.

However, some elements of the notation are purely structural, included solely to enable the system to function.

For example, it's sometimes necessary to place entries in working memory merely to serve as 'flags' which allow production rules to fire at the right moment.

It's not clear that such entries represent any genuine cognitive load.

Another problem with CCT is that the amount of code required to describe even a small part of an interface can be enormous.

This is a common problem with description methods.

Unlike GOMS, with its sequential hierarchies, CCT can represent quite complex plans.

The production rules can, in theory, operate in parallel, so CCT can be used to represent concurrent activity.

CCT production rules normally describe 'expert' behaviour - the most appropriate sequence of actions to achieve the intended result.

However, CCT also supports the use of 'style' rules which modify the way in which conditions and actions operate in production rules.

Style rules can be used to modify a CCT model to mimic different types of user, such as novices.

Bovair, Kieras and Polson produced a list of style rules which can be used to reflect different types of user in a CCT description.

CCT production rules normally describe error-free performance, but there is nothing in the structure of CCT to prevent users writing production rules that model error conditions.

However, in such cases the error behaviour must be explicitly specified in advance. Thus CCT cannot be used to predict errors.