HUMAN-COMPUTER INTERACTION, 1986, Volume 2, pp. 93-133

Task-Action Grammars: A Model of the Mental Representation of Task Languages

Stephen J. Payne
University of Lancaster

T. R. G. Green
MRC Applied Psychology Unit, Cambridge

ABSTRACT

A formal model of the mental representation of task languages is presented. The model is a metalanguage for defining *task-action grammars* (TAG): generative grammars that rewrite simple tasks into action specifications. Important features of the model are (a) Identification of the "simple-tasks" that users can perform routinely and that require no control structure; (b) Representation of simple-tasks by collections of semantic components reflecting a categorization of the task world; (c) Marking of tokens in rewrite rules with the semantic features of the task world to supply selection restrictions on the rewriting of simple-tasks into action specifications. This device allows the representation of family resemblances between individual task-action mappings. Simple complexity metrics over task-action grammars make predictions about the relative learnability of different task language designs. Some empirical support for these predictions is derived from the existing empirical literature on command language learning, and from two unreported experiments. Task-action grammars also provide designers with an analytic tool for exposing the configural properties of task languages.

Author's present addresses: Stephen J. Payne, Departments of Psychology and Computing, University of Lancaster, Lancaster LA1 4YF, England; T. R. G. Green, MRC Applied Psychology Unit, 15 Chaucer Road, Cambridge CB2 2EF, England.

CONTENTS

1. INTRODUCTION

The definitions of programming languages have frequently been formalized, often in Backus-Naur form (BNF), a notation which describes generative context-free phrase structure grammars. The virtue of BNF was originally seen as its unambiguity: Whereas programming languages had previously been described loosely in English, BNF allowed the syntax (though not the semantics) to be expressed precisely. A second virtue was soon found to be that BNF descriptions could be executed by program, guaranteeing a correspondence between the documentation and the compiler. Yet a third virtue emerged: The descriptions of different languages could be compared through their expression in a uniform, limited metalanguage.

Reisner (1977) introduced the term *action language* to describe the command system of interactive devices, and attempted to transport the virtues of BNF into this realm. She also hoped to achieve a fourth virtue; namely, to predict the psychological complexity of the language:

> A natural index of the complexity (of a statement) might be the number of rewrite rules . . . used to describe it . . . By this we mean to suggest that a BNF description of a language, usually intended to describe a set of valid statements, may have a *psychological* validity. (p. 227)

Reisner went on to propose an experimental search for a "single, consistent, psychological BNF." To some degree her claims were vindicated; her analysis of two graphics systems (Reisner, 1981) signposted areas of complexity in one of them, which were confirmed by empirical observation.

In this article we, like Reisner, offer a formalization of interface languages. Our formalism is intended to model the mental representation of the interface language and so to allow a formal specification of the language *as perceived by the user.* Of course, this goal is ambitious; we will be satisfied if our formal specifications reflect some important aspects of the perceived structure. This formalism must therefore readily express those characteristics that are salient to the user, and will thus address many of the characteristics that determine usability, particularly configural properties.

We aim to capture the notion of regularity or *consistency.* Consistency is difficult to define and therefore difficult to measure, but it is informally recognized to be a major determinant of learnability. The advantages of consistency lie in facilitating generalizations by the user, who having learned some parts of the system can then infer others.

A language can be consistent at several different levels, but at each level the key properties that determine consistency are *configural,* in that they relate to the overall structure of the language, rather than to the nature of individual task–action mappings.

To illustrate the notion of consistency, and informally describe the space of phenomena we intend to address, we present a series of examples. Although we have labeled these examples according to the level of linguistic description at which the interesting properties seem to emerge, we do not pretend to offer a taxonomy of consistency. We recognize that the boundaries between syntax and semantics are fuzzy and theory-dependent, but we have chosen examples which seem comfortably classifiable. A really useful taxonomy of consistency must wait to be stated in terms of a psychological theory, rather than linguistic abstractions.

Syntactic Consistency. Some forms of consistency are syntactic in nature. BNF can only capture a weak version, namely, the consistent use of one expression as a common element in other expressions. The term *arithmetic expression,* for instance, might well be an element common to rules for assignment statements, array bounds, Boolean expressions, and `for`-statements. In command languages it is sometimes the case that every command has an identical form—

perhaps a single letter followed by carriage return. This is a second example of *common structure*, a syntactic consistency that BNF grammars can display.

But there are often *family resemblances* between syntax rules that are simply not expressible in BNF. An example of family resemblance is shown in Figure 1. The three separate types of sequence exhibit a clear similarity (to the human eye), yet require completely separate BNF rewritings. This kind of syntactic consistency can only be captured by notations that have more expressive power.

Lexical Consistency. In natural languages it is usually assumed that lexemes are tied to their meaning by arbitrary connections. For computer languages, in which the lexemes are often English words (or icons, e.g., arrows or waste bins), the relationship is clearly nonarbitrary. The relationship between the external meanings of words and their use is command languages in complex, but one configural aspect which has been shown to benefit learners is *congruence* — the matching of lexical (external to the command language) and semantic (internal) relations (Carroll, 1982). Congruence is discussed in detail below.

Semantic–Syntactic Alignment. In the ideally consistent language, semantic relations will not only be mirrored in the lexical or symbolic relations, but also in the structure of commands. If a "copy file" command requires the existing file to be specified before the new file, then a "copy disk" command should require the source drive to be specified before the target drive: The task semantics should map onto the language syntax in a consistent way. This point is developed at greater length in Section 4.2.

The inverse is also true: A single linguistic element will ideally perform the same semantic function in any context. Green and Payne (1984) noticed that in a commercial word-processing system, the control and escape keys were used to organize the cursor movement semantics, but according to two conflicting *organizing principles*. For some pairs of commands, "control" and "escape" would switch between small and large units; in other cases, "control" and "escape" determined the direction of movement of the cursor. A language-learning experiment confirmed that this conflict troubled learners, to the extent that a language with no meaningful mnemonics, but a single organizing principle, proved reliably easier to learn. A small subset of both of these experimental command languages is shown in Figure 2. (A third language from the experiment, the most easily learned of all, is described later, in Figure 4.)

Semantic Consistency. Our final examples of consistency are properties of the extensional semantics of a language. Again, the key issue is that one part of the language prompts expectations about the remainder. For example, if a word processor allows search for a character string both forward and backward

Figure 1. **Family resemblances between syntactic rules. The three rules express a small fragment of a programming language in BNF. The rules have an obvious resemblance, but that is not directly represented in the grammar, which has no generalized notion of a sequence.**

```
<declaration sequence>    ::- <declaration>
                            | <declaration sequence> + <declaration>
<statement sequence>      ::- <statement>
                            | <statement sequence> + <statement>
<letter sequence>         ::- <letter>
                            | <letter sequence> + <letter>
```

Figure 2. **Example commands from two experimental text-editing languages (Green & Payne, 1984). Language 1 is a subset of a commercially available editor, but has conflicting organizing principles. Language 2 has no mnemonic coding, but is organized according to a single principle.**

Commands	Language 1	Language 2
move cursor forward one character	ctrl-F	ctrl-L
move cursor backward one character	ctrl-B	esc-L
move cursor forward one word	esc-F	ctrl-E
move cursor backward one word	esc-B	esc-E
view next screen	ctrl-V	ctrl-C
view previous screen	esc-V	esc-C

through a file, then we expect it to allow a similar flexibility for a search-and-replace operation. Similarly, in the programming language Pascal, we are disappointed that one can read in values for real identifiers, but not for Boolean identifiers. We term the consistency principle that is being broken here *completeness*. Some richer examples of semantic consistency will be discussed below.

Our central goal in this article is to present a notation for the description of languages which has something to say about all these various aspects of consistency. We attempt to capture generalizations in a psychologically valid manner, and so identify consistency with observable properties of notational descriptions.

Our first efforts toward this goal modeled syntactic generalizations, addressing the family resemblances illustrated here (Payne & Green, 1983). We proposed a context-free variant of van Wijngaarden's two-level grammar (for a good description, see Pagan, 1981), which we called *set-grammar*, because the rewrite rules operated on sets of grammatical objects rather than individual nonterminal or terminal symbols (Payne & Green, 1983). Figure 3 shows the set-grammar representation of the syntactic family resemblances illustrated in Figure 1.

Figure 3. **Set-grammar treatment of family resemblance. The fragment of a programming language shown in Figure 1 is represented here in the set-grammar notation (Payne & Green, 1983). The family resemblances are captured by collapsing three rules into one higher-order rule.**

SET
SEQ-ITEMS :: {declaration, statement, letter}

RULE
SEQ-ITEM sequence → SEQ-ITEM | SEQ-ITEM sequence + SEQ-ITEM

SELECTION RULE
Uniform replacement—the same element is chosen from the set of SEQ-ITEM throughout the rule.

Consistent languages, we argued, could be expressed by a small number of set-grammar rules; whereas inconsistent languages, in which some syntax constructions had little in common with other constructions, could not be reduced to a small number of set-grammar rules. The set-grammar achieved some success as a model of syntactic perception, predicting several laboratory results on the learnability of command languages and miniature artificial languages (Payne, 1985; Payne & Green, 1983). Payne (1985) gave some support to the specific type of family resemblance that was expressible in set-grammars by showing that of two miniature artificial languages, the easier to learn was the one predicted by set-grammar, because fewer set-grammar rules were needed to express its grammar. This result was promising because the languages were carefully devised so that their BNF representations predicted the opposite result. However, the utility of set-grammar as a cognitive model and for the analysis of usability suffered from a severe shortcoming: It possessed no mechanisms for relating the syntax of a language to its semantics. Many of the most important determinants of consistency rely on semantic properties.

The notation presented here supersedes set-grammar and extends its explanatory power beyond the syntactic realm into the semantics of tasks. The notation is a grammar describing a mapping from the users' tasks onto sequences of actions. Because the starting symbols of our grammars are tasks and the terminal symbols are action specifications, our notation is called task-action grammar (TAG). We use the term *task language* to describe the intended domain of application of task-action grammars: A task language is any task-action interface between a person and a machine, including lexical command languages, direct manipulation interfaces, and knobs-and-dials control panels.

Like other grammars, TAG models competence rather than performance, but we offer sketches of related learning and performance theories in this article. Many of our notational devices, such as the use of a feature grammar, the representation of concepts as sets of semantic components, and the notion of a simple or atomic task, have a basis in cognitive psychology and psycholinguistics. This basis will also be discussed.

2. THE AIMS AND NOTATIONAL STRUCTURE OF TAG

In this section we first illustrate the general principles of our notation, which are those of a feature grammar. This feature grammar describes the mapping from the task level to the action level. The central aim of TAG is to formalize that mapping in such a way that simple metrics over the grammar, such as the number of rules, will predict aspects of the psychological complexity of the mapping. These aspects of complexity include time spent learning, intrusive errors during learning, and the ability to generate a forgotten or unknown part of the language from the remainder. (We discuss metrics and predictions in Section 4.) A secondary aim of TAG is to help the analyst appreciate the structure of a task language.

A task-action grammar is a formal device. Its input is a description of a task as a set of semantic components. (A semantic component is a particular value on a featural dimension; if **Direction** is a feature, then **Direction = right** is a particular value which might be a component of a task definition, such as **move-one-character-to-the-right**.) Its operation is that of a generative grammar, and its output is a list of the actions required to perform the input task. Thus task-action grammars describe not just the syntax of operations used to control the device, but also the relationship between the actions and commands and the user's view of particular tasks.

Obviously, the users of interactive systems have the ability to perceive certain regularities in task-action mappings, but not others. When regularities exist and can be perceived, they can be used to simplify the structure of the mapping, replacing a number of unrelated mapping rules by a single, more general rule. Our choice of formalization makes an implicit theoretical statement about the limits of that ability. For instance, using a standard context-free phrase structure grammar as a representation of the mapping would postulate that users could perceive a hierarchical structure of rules and subrules, but that they were blind to family resemblances between rules.

In describing the general principles and the particular notational conventions, we will show how TAG representations capture the following important attributes of particular mappings (including aspects of consistency discussed in Section 1):

- family resemblances, or the overall "sentence structure" of commands;
- the degree to which a task language relies on well-learned "world knowledge," in particular on familiar names, such as *up* for the concept UPWARD DIRECTION;
- the "completeness" of a mapping, as opposed to arbitrary absence of expected components;
- the notion of "congruence," or the representation of semantic relations by equivalent lexical relations (notably, semantic opposition by lexical antonymy);

- the organization of tasks and subtasks, and the alignment of semantic and syntactic aspects of the task language.

2.1. The TAG Framework—Simple-Task Dictionary and Rule-Schema Expansion

Figure 4 presents a TAG description of a fragment of an experimental text-editing language used by Green and Payne (1984) and introduced already in Section 1. The description consists of a *Simple-task dictionary,* in which simple-tasks are identified and defined by their semantic components, and a feature grammar in which the dimensions of those components serve as features. (In more complex examples, it aids exposition to add a third component, an explicit list of possible features and their values. This redundant device has been omitted here.)

The TAG description models the mental representation of a user who has learned the four commands. The simple-task dictionary represents all the tasks the user can routinely perform, and defines each as a set of semantic components. These components are to be used in guiding the *rule-schemas* into generating the required actions for each possible task.

The rule-schemas generate action specifications from simple-tasks in a similar manner to standard phrase-structure grammars, but they have the additional notational power of encapsulating several standard phrase-structure rules in a single rule-schema. In this example, the general task-action rule-schema (Rule 4.1) is expanded by assigning the values to all the features in the square brackets, with the constraint that a feature must be assigned with the same value wherever it appears in the rule. A possible result of these assignments is the single-level rewrite rule:

Task [Direction = forward, Unit = word] →
symbol [Direction = forward] + letter [Unit = word]

All the tokens now refer to unique grammatical objects (they *must*; this is a constraint on the features in the square brackets). The simple-task object can be found by looking at the simple-task dictionary; the symbol and letter objects are expanded by further rewrite rules (in this case, Rules 4.2 and 4.4), using the standard production-rule grammar mechanism. These operations yield, in the example:

move-cursor-one-word-forward → "ctrl" + "W"

which is one of the represented commands.

This simple example illustrates most of the important mechanisms of a TAG grammar. The most novel mechanism is the consistent assignment of values to features, combined with the looking up of simple-task descriptors in a dictionary. The strength of the grammar which we have used as this first example is

Figure 4. **TAG's treatment of cursor control in an experimental text editor.**

List of Commands

move cursor one character forward	ctrl-C
move cursor one character backward	meta-C
move cursor one word forward	ctrl-W
move cursor one word backward	meta-W

TAG Definition

DICTIONARY OF SIMPLE-TASKS
move-cursor-one-character-forward {Direction = forward, Unit = char}
move-cursor-one-character-backward {Direction = backward, Unit = char}
move-cursor-one-word-forward {Direction = forward, Unit = word}
move-cursor-one-word-backward {Direction = backward, Unit = word}

RULE SCHEMAS
4.1 Task [Direction, Unit] → symbol [Direction] + letter [Unit]
4.2 symbol [Direction = forward] → "ctrl"
4.3 symbol [Direction = backward] → "meta"
4.4 letter [Unit = word] → "W"
4.5 letter [Unit = char] → "C"

that it allows the basic form of all the task-action mappings to be represented in a single higher-level schema (Rule 4.1). It captures the observed consistency in the organization of the language due to an organizing principle in the semantic–syntactic alignment.

It is intuitively clear that the design of this language could be improved further by the use of mnemonic codes. Just such an improvement was validated in the experiment of Green and Payne (1984). The TAG description of the advantage is illustrated in the following section.

2.2. Sentence Structure, Completeness, and World Knowledge

An important type of consistency in a task language occurs when every command has the same sentence structure. Frequently this structure is a single keypress, which is trivial. A less trivial sentence structure is the postfix-style language structure, as used, for instance, on the Apple Macintosh. Another nontrivial structure is the list of operations, followed by a terminator, familiar to users of TECO and its descendants.

We start by considering an example of a system in which every simple-task is performed by entering a single command name, followed by a carriage return. In this particular system, there are only three commands which are used for moving some object around a screen. The entire task world, therefore, comprises three simple-tasks, which we can denote as follows:

move-up {Direction = up}
move-down {Direction = down}
move-right {Direction = right}

The featural description of each simple-task concept arises from a straightforward *categorization* of the task world. A user who has learned the entire language will represent the dictionary of simple-tasks in this way, reflecting the natural categorization. The features are chosen to describe the important discriminations in the set of simple-tasks; if the current language were extended to include commands that have no movement function, then the direction components of our three tasks would need to be supplemented with a new feature, allowing their specialized movement function to be represented.

If the commands chosen for the three tasks were, for example, **N**, **V**, and **G**, then the user may represent the task-action mappings as follows.

0.1 Task [Direction] → name [Direction] + "RETURN"
0.2 name [Direction = up] → "N"
0.3 name [Direction = down] → "V"
0.4 name [Direction = right] → "G"

We use this trivially simple grammar to make three points. First, we note that the mapping is incomplete. Because TAG descriptions include a dictionary of simple-tasks and their associated semantic components, it is easy to spot combinations of features that are semantically acceptable but that have no associated simple-task. In this example, there is no simple-task with the component **{Direction = left}**. Of course, it is hardly necessary to use a task-action grammar to reveal this particular incompleteness, but, in general, incompleteness can be much harder to spot. Highlighting completeness is a small example of TAG's potential as an analytic device for appreciating the structure of the task language, separate from its main role as a predictor of relative learnability.

Next, the TAG description captures a strong intuition about sentence structure. If a fourth command were to be added, for moving to the left, we would be perplexed if it were not accomplished by typing a single letter followed by **RETURN**. According to the task-action grammar, our difficulty would spring from the fact that a new top-level schema would need to be formed: a special one for the move left command. (Actually, we have not yet represented the fact that every command name in this system is a single letter. TAG can express this, but it depends on a notational device which we have yet to describe.)

Finally, and the main point of this example, we can contrast Rules 0.1 through 0.4 with Rules 0.5 through 0.9 which describe a language in which the arbitrary letters have been replaced by **UP**, **DOWN**, and **RIGHT**.

0.5 Task [Direction] → name [Direction] + "RETURN"
0.6 name [Direction] → known-item [Kind = word, Direction]
0.7 *name [Direction = up] → "UP"

0.8 *name [Direction = down] → "DOWN"
0.9 *name [Direction = right] → "RIGHT"

Since English-speaking users will know that **UP** is the name for the concept, we wish to find a way to differentiate between arbitrary and well-learned names. We do so by constructing a TAG that contains only two effective rules, 0.5 and 0.6, of which Rule 0.6 says, "The subrule for determining the name for a given movement is: Use the word that shares the direction feature of the intended movement." We assume that lexical access to well-learned names is essentially effortless as long as the semantic components of the task genuinely do bring to mind the required words, the size of the vocabulary is not important. Rules 0.7 through 0.9 are therefore marked with asterisks to indicate that they should not be included in any simple metrics over the grammar, such as counting the number of rules. We choose to include them in TAG descriptions because all the terminal symbols of the language are then visible, and because assumptions about world knowledge are then open to inspection. (We can determine, for instance, what aspects of a language might give trouble if users come from a different population from the one envisaged by the designer.)

Rules like 0.6 are called world knowledge rules. Naturally, they need not be exclusively lexical; their right-hand sides can refer to any entities that exist, independently of the task language, in the user's semantic memory. Spatial features of the keyboard or screen are powerful examples. World knowledge rules serve two purposes for the user: They increase the robustness of the grammar in the head, so that a forgotten rule can be regenerated from a world knowledge schema; and they ease the learning load, allowing unlearned rules to be hypothesized from existing ones.

A notational device similar to world knowledge rules is used to denote *action variables* — action specifications that require a feature value to be determined by the user's current goal. (We envisage the value being passed to the grammar as a parameter by the planning system; see Section 6.2.) Action variables are quite common in command languages. For example, most text editors allow the user to search through a file for any string of characters. The search command will have a set syntax, but the particular string to be searched for is an action variable, whose value is determined by the current goal. Often it is convenient to simply denote action variables with standard quoted terminal symbols, specifying the action weakly by a general label which needs to be replaced by an exemplar (as shown in Figure 10). Sometimes, however, if the organization of the task language is to be fully captured, it is necessary to specify features that take values from the goal. Using this convention, a TAG rule for a search command might be:

0.10 Task [Purpose = search] →
"ctrl-S" + action [Kind = key, String = value-from-goal]

2.3. Congruence: Matching Semantic and Lexical Relationships

An important paper by Carroll (1982) provided empirical evidence in support of a naming principle he called *congruence*. The idea can be illustrated by a small subset of two of his experimental languages, as shown in Figure 5.

Carroll discovered that subjects learning a pencil-and-paper simulation game found the names of Language A easier to learn than those of Language B. He explained this with reference to the congruence of the name set and the task world; basically, opposite commands are invoked by opposite words. This is a good general principle, and intuitively likely to be a robust influence on usability, beyond individual differences and context effects.

It is not difficult to devise a notational device to express congruence. On the one hand, we have a set of tasks whose semantic components are identical in every respect except one; thus the tasks whose components are {move, forward} and {move, backward} differ only in the Direction feature. On the other hand, we have a set of names for just those semantic components. To express the idea of congruence between the names *advance* and *retreat,* therefore, we need a notation that describes the set of words, choosing between them on the basis of the Direction feature.

Rule 5.2 in Figure 5 shows how we do this. It can be read as: "The name for movement in a given direction is that word which has all the features of 'advance' except the Direction feature, which it derives from the current task." So, if the Direction feature is forward, the name is *advance*; if it is backward, the word is *retreat*. Thus, to express the idea of congruent sets, we pick an arbitrary member of the name set (*retreat* would have done just as well as *advance*), use the notation F("advance") to refer to all the semantic components of that word, and then replace the Direction feature by the desired direction. Using a single symbol to denote a feature set defining a concept does not add formal power to the TAG notation, and it is, in fact, common practice in the linguistic and psychological literature (e.g., Tversky, 1977).[1]

We are now in a position to compare the TAG descriptions of Carroll's two languages. Language A permits us to use the notational device we have described; Language B does not, because GO and BACK do not form a set differentiated solely by the relevant semantic component Direction (neither do TURN and LEFT). Language B will, therefore, require more rules. Simple metrics,

[1] Notational remark: Essentially, F is a function returning a set of components; we use round parentheses to differentiate it from the feature-marked nonterminal symbols of the grammar. Because Rule 5.2 expresses the idea that "advance" and "retreat" share the same semantic attributes, varying only in direction, the notation could be said to embody a view of antonymy that is in keeping with some suggestions in the linguistic literature (e.g., Katz, 1972). There is a debate surrounding such logical distinctions as contrast and opposition (Lyons, 1977), but we are merely suggesting that a good nameset will reflect differences and similarities between operations by providing names that vary along the same dimensions as the operations they represent.

Figure 5. **Congruence. TAG's treatment of lexical consistency in two experimental languages from Carroll (1982).**

A Subset of Commands From the Languages

	Language A (congruent)	Language B (noncongruent)
Commands the robot to move forward or advance one step	ADVANCE	GO
Commands the robot to move backwards or retreat (in reverse) one step	RETREAT	BACK
Commands the robot to change the direction it faces by moving or turning 90 degrees to the right	RIGHT	TURN
Commands the robot to change the direction it faces by moving or turning 90 degrees to the left	LEFT	LEFT

TAG Descriptions

TASK FEATURES (the same for both languages)

Feature	Possible values
Move/Turn	move, turn
Direction	forward, backward, right, left

SIMPLE TASKS (the same for both languages)

move-robot-forward {Move/Turn = move, Direction = forward}
move-robot-backward {Move/Turn = move, Direction = backward}
turn-robot-right {Move/Turn = turn, Direction = right}
turn-robot-left {Move/Turn = turn, Direction = left}

RULE SCHEMAS, Language A

5.1 Task [Move/Turn, Direction] → name [Move/Turn, Direction]
5.2 name [Move/Turn = move, Direction] → known-item [Type = word, F("advance"), Direction]
5.3 *name [Move/Turn = move, Direction = forward] → "ADVANCE"
5.4 *name [Move/Turn = move, Direction = backward] → "RETREAT"
5.5 name [Move/Turn = turn, Direction] → known-item [Type = word, F("right"), Direction]
5.6 *name [Move/Turn = turn, Direction = right] → "RIGHT"
5.7 *name [Move/Turn = turn, Direction = left] → "LEFT"

RULE SCHEMAS, Language B

5.8 Task [Move/Turn, Direction] → name [Move/Turn, Direction]
5.9 name [Move/Turn = move, Direction = forward] → "GO"
5.10 name [Move/Turn = move, Direction = backward] → "BACK"
5.11 name [Move/Turn = turn, Direction = right] → "TURN"
5.12 name [Move/Turn = turn, Direction = left] → "LEFT"

such as the number of rules, suggest that Language A will be easier to learn and remember, as Carroll found.

2.4. Task Structure, Organizational Consistency and Conflict

Having described the main features of the notation, it remains for us to show how the structure of a more complex interactive system is expressed in these terms. We hope to use this example to demonstrate the secondary function of TAG, as a tool to help the analyst uncover subtle aspects of the structure of an interface. An important feature to bring out is how subtasks are captured by subrules, thus:

Task → subtask 1 + subtask 2

This procedure (which closely resembles subroutining) is assumed to model hierarchical task structure as perceived by the user.

The example we draw on to illustrate the points must necessarily be rather larger than the smaller ones given previously. We have chosen to present an analysis of an idealized interactive graphics drafting system with a mouse-driven interface, based on Apple Computer's MacDraw program for the Macintosh. This program manipulates graphic objects, such as lines, circles, and rectangles. We suppose that the user perceives four main tasks:

- A new object can be created. The user chooses a "tool," or object type, from a menu of line, ellipse, rectangle, and arc. The user specifies the location and size by positioning the mouse in the top left corner of the desired location, pressing the mouse button, and "dragging" the mouse to the bottom right corner. The object is created inside the rectangle thus designated; for example, if an ellipse tool is chosen, an ellipse will be created inside the rectangle, with its origin in the center of the rectangle and its major axis either vertical or horizontal, parallel to the long side of the rectangle. "Style" attributes, such as line width and filling, are set by default.
- The default style attributes can be altered. The user must point to the appropriate style menu and item-within-menu.
- The style attributes of an existing object can be altered. The user selects an object, then points to the appropriate style menu and item-within-menu.
- An existing object can be moved. The user selects an object, then drags it to a new position.

In the genuine MacDraw, users can also delete objects, resize the drawing, write text, and draw freehand. These abilities would add complexity to our description without revealing any further aspects of our notation. A knottier problem that we will not discuss is that users can also manipulate different groups of objects at one time.

Task Structure

We first consider the modeling of hierarchical task structure. The main structure of the simple-tasks will be clear from Figure 6, but how does one decide on the detailed structure? This problem confronts any model purporting to describe perceived structures. In the absence of any empirical data on perceived task structure (e.g., in the style of Robertson & Black, 1983), writers of a task-action grammar must use their judgment. One solution to the problem is to base description on the structures presented in the training manual, as was done by Kieras and Polson (1985).

The solution we have adopted is to search for the most economical representation of a language that we can find. Although this representation may not be achieved by all users, it gives a lower bound on the psychological complexity. Observe that where alternative methods exist for achieving a particular task, the most economical representation will probably only describe one of the methods. This representation is a generative grammar, but not necessarily an acceptance grammar because it may not be able to describe an accurate parse of some users' action sequences.

When, however, the aim of the analyst is to predict confusions that a learner might encounter, the appropriate representation might well be an acceptance grammar, describing all reasonable routes to a task, so that potential confusions between them become visible. We illustrate this problem of *organizational conflict* in the following section.

In the present instance, it seems reasonable to suppose that experienced users perceive the task structure in terms of pointing at places and selecting tools, objects, or styles. If this task structure is directly used as a design basis, it gives the following method for object creation:

Task [Effect = create, Type] →
    select tool + point-to-place + point-to-place

The designers of MacDraw foresaw, however, that this simple presentation created an extra and undesirable mode, in which a tool had been selected and the first place, but not the second, had been marked. To avoid this mode, they imposed the rule that whenever two places are to be marked in succession, the mouse button must be depressed at the first place, held down while traveling to the second, and then released. It is therefore necessary to introduce a subtask, point-to-2-places, to describe this generalized action sequence, giving the structure seen in Rule 6.1.

Organizational Consistency

The dragging operation also illustrates another neat piece of design, in which special cases are managed with a degree of consistency that is very high (but not

Figure 6. **TAG description of an idealized MacDraw interface. In this grammar, the Place features can take values from the user's current goal, denoting action variables as discussed in Section 2.2.**

SIMPLE TASKS
Create new object {Effect = create, Case = regular}
Create special object {Effect = create, Case = special}
Move object {Effect = move, Case = regular}
Move object along restricted path {Effect = move, Case = special}
Change default style attributes {Effect = change-default-style}
Change object style attributes {Effect = change-object-style}

RULE SCHEMAS

```
6.1  Task [Effect = create, Case] →
               select-tool
             + point-to-2-places [Case, Place1 = value-from-goal,
                                  Place2 = value-from-goal]
6.2  Task [Effect = move, Case] →
             point-to-2-places [Case, Place1 = value-from-goal,
                                Place2 = object-location]
6.3  Task [Effect = modify-object-style] →
                                   select-object
                                 + select-style
6.4  Task [Effect = change-default-style] →
                                 select-style
6.5  point-to-2-places [Case = regular, Place1, Place2] →
                              action [Kind = point, Place1]
                            + drag-to-place [Place2]
6.6  point-to-2-places [Case = special, Place1, Place2] →
                            action [Kind = point, Place1]
                          + "depress mouse button"
                          + "depress SHIFT"
                          + action [Kind = point, Place2]
                          + "release mouse button"
                          + "release SHIFT"
6.7  drag-to-place [Place] → "depress mouse button"
                         + action [Kind = point, Place]
                         + "release mouse button"
6.8  select-style →
       point-to-2-places [Place = style-menu, Place2 = menu-item]
6.9  select-object → action [Kind = point, Place = object-location]
                   + "click mouse button"
6.10 select-tool → action [Kind = point, Place = tool-icon]
                 + "click mouse button"
```

perfect, as we will soon see). The graphic objects that can be created by MacDraw include lines, rectangles, ellipses, and arcs. Each of these objects has a special case: vertical or horizontal lines, squares, circles, and quarter circles. (Lines, of course, have more than one special case—we discuss that awkward fact in a moment.) It would be perfectly possible to design a system in which the command create-a-square was different from create-a-rectangle, and in this system the designer could arbitrarily decide not to include a special case for, say, arcs. In MacDraw the designer has opted to use general tools to create both ordinary and special cases, and has created a language that requires every object to have exactly one special case; pressing SHIFT during the dragging part of the **point-to-2-places** subtask will create the appropriate object. In the special case of lines, the line snaps to vertical, horizontal, or 45 degrees, whichever is the best fit, so that three special cases are automatically subsumed into one to preserve consistency of structure.

The same constraints apply also to the movement of graphic objects and are treated the same way: By pressing SHIFT during the dragging operation, the movement of the object can be restricted to vertical, horizontal, or 45 degrees. The task structure shown in Figure 6 captures this consistent use of the SHIFT key by using the same subtask, **point-to-2-places**, both for creating and moving objects.

Organizational Conflict

We have used the term *organizational conflict* to refer to the situation in which two or more competing organizations can be perceived. We know of no test that could alert designers to possible misperceptions of this type, but one countermeasure might be to ensure that each task and subtask of the intended organization included a distinctive action.

TAG analyses help unearth possible organizational conflicts, and examples can be found in our idealized MacDraw. One such conflict involves Rules 6.5, 6.6, 6.7. Of these, Rule 6.7 represents a standard **drag-to-place** method, which is utilized by Rule 6.5. Rule 6.6 does not utilize the standard dragging subtask, however, because the mouse button and point sequence is interrupted by pressing the SHIFT key. An alternative perceived organization for Rule 6.6 would be:

```
6.6a  point-to-2-places [Case = special, Place1, Place2] →
         action [Kind = point, Place1]
      + "depress SHIFT"
      + drag-to-place [Place2]
      + "release SHIFT"
```

This would be simpler. Moreover, it is successful—some of the time. We would therefore predict that some novices would discover Rule 6.6a. But it is only successful in one context, when an object-creation tool has been selected. So when the task is to create an object (Rule 6.1), the alternative form of Rule 6.6 will be successful. When, instead, the task is to move an object, the alternative form of Rule 6.6 will have a different effect, because in that context pressing SHIFT will result in selecting multiple objects (or deselecting them, if already selected).

The most economical TAG grammar for the idealized MacDraw can ignore Rule 6.6a, but an acceptance grammar would have to represent both that and Rule 6.6. The result would be complicated. We would expect learners who discover the structure by experience, rather than tuition, to suffer some difficulties before they acquire the more economical form of representation.

A second conflict occurs because our idealized MacDraw supports a second organization for Rule 6.2. (In fact, this second organization was the one held by the authors until performing this analysis!) To explain the second organization, we will start with the task of modifying the style attributes of an object. To do so, the user selects the object (which causes MacDraw to highlight it) and then chooses a new style attribute. Naturally, this is presented in Figure 6 as **select-object** followed by **select-style**. It turns out in practice that many users seem to regard **move-an-object** as a task with a similar structure, in which the first subtask is to select the object. This gives them the following structure:

6.2a Task [Effect = move] →
select-object
+ drag-to-place [Place = value-from-goal]

Users then have a very consistent structure for all four tasks, because they all start by selecting something, whether tool, object, or style. There are two consequences of that perceived organization. First, unnecessary keystrokes are performed, although because these are no more than an otiose release and redepressing of the mouse button, they are not expensive in effort. Second, the consistent use of the SHIFT key is not visible in that organization, because it does not use the subtask **point-to-2-places**; the subtask **drag-to-place** might or might not make use of the SHIFT key to constrain direction of movement, entirely independently and apparently arbitrarily. It would therefore be quite possible to believe, as the authors did, that the SHIFT key produced special effects during object creation, but had no further uses in MacDraw.

Summary

In this analysis of MacDraw, we have attempted to demonstrate that TAG can capture quite subtle aspects of organizational consistency. Indeed, by performing the TAG analysis we have ourselves been led to perceive initially obscure subtleties. This observation might lead to a criticism that TAG is capturing properties of the language that are not salient to users, and so do not influence complexity. Our response to this criticism is twofold. First, we would argue that it is a mistake to necessarily tie salience and complexity to awareness—consistency may reap benefits for users without being articulated by them. Second, TAG's psychological validity should not be second-guessed, but should be subject to empirical scrutiny. This theme is taken up in Section 4.

3. FORMAL SPECIFICATION OF THE TAG METALANGUAGE

The preceding examples illustrate all the notational conventions of the TAG metalanguage and leave us in a strong position to define it precisely. We have chosen, however, not to offer such a definition, for two reasons. First, we do not regard the notation as fixed: It seems sensible to keep the way open for refinements and extensions. Other attempts at formal definition in computer science do suggest that case-driven development is likely to be necessary (e.g., Lee, 1972). Second, such a definition would add little to the precision of our case; if the examples given here are taken as definitive rather than merely illustrative, we are able to demonstrate the formal properties of TAG straightforwardly. It will be helpful, however, to offer a complete list of the symbolic expressions and operations used in the current notation.

A task-action grammar has three parts: (a) an optional list of features for categorizing the simple-tasks, (b) a dictionary of simple-tasks, and (c) a set of rule-schemas. The list of features is redundant in the grammar's workings and is provided purely to aid exposition.

The dictionary of simple-tasks lists a label for each operation which the user can perform routinely, together with a featural description of that simple-task in terms of semantic components which categorize the entire task world. The component-set is denoted by brackets, { }. Each component is denoted by a descriptive term for the feature, the equal sign, =, and a term for the value of that feature. The symbols + and − are used to denote presence or absence of binary features.

Every rule-schema contains a single element on the left-hand side (LHS); each LHS is a term consisting of an arbitrary label and a feature-set contained in square brackets, which may contain any number of unvalued or valued features (components).[2] The LHS is separated from the right-hand side (RHS) by an arrow → which denotes rewriting, or definition, in the standard phrase-structure grammar sense. The RHS of a rule contains an ordered sequence of terms (in exactly the same format as the LHS) and of terminal symbols. Entire sets of features may be abbreviated by the symbol F(label), which denotes the set of defining features of the labeled token. Where a feature-set appears together with a specified feature, the specific feature takes precedence, and the feature-set is understood to denote only the unspecified defining features. Unvalued feature-sets may also be specified, denoted F( ).

To expand a rule-schema, any unvalued feature must be assigned values uniformly throughout the rule, whether they are ordinary features or feature-sets. When this assignment is complete, every term must denote a unique grammatical object. If a term refers to a simple-task (given the generic label **task**), then the valued features will specify exactly one of the entries in the simple-task dictionary. Where the term refers to a nonterminal symbol, that nonterminal will itself appear as the LHS of a rule schema. The third possibility, if the rule is a

[2] The meaninglessness of all nonterminal symbols is a virtue not shared by earlier formulations of the TAG notation. It results from a simple change to the syntax of world knowledge rules, and was suggested by Tom Moran.

world knowledge rule, is that the term (denoted **known-item**) refers to a unique object assumed to be in the user's semantic memory. Terminal symbols are action specifications, denoted by terms in quotation marks, or action variables (denoted **action**) which contain features whose value may be determined by the current goal.

The most important formal property of TAG is its generative power. It is easy to see that TAG only has context-free capability. Each ordinary rule schema can be expanded into a finite number of single-level (context-free) rewrite rules, simply by assigning values to features in all admissible ways. World knowledge rules do not affect this capability, as they meet the same constraint, albeit through the device of features and components that exist independently of the system being described. Nevertheless, as we have noted, it is inevitable that some extension to the core metalanguage will be demanded as wider aspects of usability are considered.

4. USING TAG TO ASSESS LEARNABILITY

Having described the workings of TAG and shown how it addresses some interesting properties of the interface, we now explain how to apply TAG in order to make predictions of task language learnability. First, we discuss some complexity metrics which allow direct comparisons between one TAG description and another. Next, we examine the empirical literature on command language learning to consider the extent to which TAG, allied to the complexity metrics, predicts the important results; this section is extended by summaries of two novel experiments testing central TAG learnability predictions.

4.1. Complexity Metrics Over Grammars

The prime applied focus of TAG is to assess the relative learnability of different task languages. The central argument is that because a TAG description models mental representation of the language, simple metrics over the grammar will reflect psychological complexity, and thus learnability. We have already mentioned these metrics; in this section we inspect them more closely.

This argument suffers two potential difficulties. First, in the absence of a specified learning mechanism one might argue that learnability is badly underdetermined, because a mechanism could be devised that learned complex grammars more easily than simple ones. Our response to this is straightforward: Although we have not specified a learning mechanism, we simply assume one with the apropriate properties. Indeed, we are more or less bound to such an assumption by the nature of our enterprise for the status of any representation as an explanatory device depends crucially on implicit processing assumptions. If the empirical predictions made by TAG, through its alliance with learning assumptions, should fail, then we must modify the grammar, not the assumptions.

The second difficulty with our learnability argument is that any number of TAG grammars can be written to define a single task language. How do we know which definition to use? This degrees-of-freedom problem is an inevitable consequence of using a grammar as a competence model, but it also applies to other attempts at modeling human-computer interaction, such as the production systems of Kieras and Polson (1985). The problem in our case is rather less severe, for our express goal is the prediction of relative complexity, so that any consistent approach to choosing a single decription can be justified. The approach that we choose is governed by our proposed complexity metrics over TAG descriptions.

Any grammatical description has a large number of properties; which are the ones that determine complexity? The most important index of complexity derived from a TAG definition is the number of simple-task rule schemas, for these rules define the top-level configuration of the task language. For two languages that possess the same number of simple-task rule schemas, complexity comparisons should utilize the total number of rule-schemas, including those that rewrite the nonterminal symbols of the grammar. Because of the prime importance of the simple-task rule schemas, our solution to the degrees-of-freedom is to base comparisons on descriptions of languages that minimize the number of simple-task rule schemas.

It is worth noting that the degrees-of-freedom problem potentially can be turned into a positive advantage. Because several TAG descriptions of a given language are possible, we have the means of modeling individual differences in perceived structure. We have already seen some benefits of using TAG in this fashion in our discussion of the MacDraw interface. In that case, by departing from our usual constraint of minimizing the number of simple-task rules, we were able to display interesting alternative structurings of the task language. Our complexity metrics depend on the assumption that although alternative TAG descriptions are necessary to model individual perceptions of structure, the minimal description models regularities that generally will be perceived and is therefore the best available approximate guide to the intrinsic complexity of the design.

4.2. Empirical Support

TAG's main empirical prediction is that, of two similar task languages, the one that will be the easier to learn and remember is the one with the fewer simple-task rule schemas or, should the languages be equivalent in this respect, the language with the fewer rules altogether (not counting rules that are captured by world knowledge schemas). Empirical support for this claim can be gathered from the existing experimental literature on command language learnability. TAG captures many results on command language names and syntax. From the naming literature, we have already seen TAG descriptions of the main effect demonstrated by Black and Moran (1982), Carroll (1982), and Green and Payne (1984). The advantage of using structured namesets, as demonstrated by Scapin (1982) and Carroll (1982), also can be captured. Scapin's experimental namesets, and a TAG description of each, are shown in Figure 7.

Results on command language syntax are somewhat less concrete. The experiment by Barnard, Hammond, Morton, Long, and Clark (1981) is hard to interpret, but it offers some support for the principle of placing a common parameter in a constant serial position within a command string—a principle that is supported by TAG in a fairly obvious way (see Figure 8). The observational reports of Reisner (1981) also support the notion of a consistent syntax; again, TAG predicts these findings (as did Reisner's own grammar, but only through an informal extension to the notation).

More importantly, TAG makes predictions about syntactic consistency that go beyond the findings of Barnard or Reisner. The categorization of simple-tasks predicts that syntactic consistencies across entire semantic categories will be appreciated by users, as will syntactic consistencies within semantic categories, but consistencies across groupings of tasks orthogonal to the semantic organization will be of no benefit. This prediction has been tested in an experiment on syntax induction by Payne (1985). In this test, subjects learned to operate a toy "lost property office" computer system to perform three categories of task: request information from a database, end messages to colleagues in other locations, and play games (a scenario not unrepresentative of real world computer use). In each of these three main categories several distinct task-types could be performed—information could be requested about missing dogs, cars, jewelry, or valuables; messages could be sent privately or in public; two completely different games could be played. The experimental results demonstrated that the best syntax utilized a consistent higher-level structure (as captured by TAG schemas) for all possible tasks (Language 1, Figure 9). A design which incorporated only two different syntactic command structures, but which used these for groups of task-types that cut across the major categories (Language 3, Figure 9), was harder to learn than a design using three separate command structures, one for each of the categories (Language 2, Figure 9). TAG descriptions of the experimental languages are shown in Figure 10. Counting the number of simple-task rule schemas predicts the observed results. We should note that this prediction holds even though the TAG notation for Language 3 has the smallest total number of rules. The reason for this is that the naming and abbreviation conventions (which are exactly the same for all three languages, of course) are represented by additional grammatical rules in Languages 1 and 2, precisely because the structure of these languages allows the generalized simple-task schemas. In Language 3, because individual rule

Figure 7. **Structured namesets. TAG's treatment of the experimental namesets from Scapin (1982).**

Namesets

Structured	*Unstructured*
SEND MESSAGE WAITING	SEND MESSAGE WAITING
SEND MESSAGE RECEIVED	TRANSMIT MESSAGE RECEIVED
FILE MESSAGE WAITING	FILE MESSAGE WAITING
FILE MESSAGE RECEIVED	CLASSIFY MESSAGE RECEIVED
FILE MESSAGE MAILED	STORE MESSAGE MAILED
CREATE MESSAGE WAITING	WRITE MESSAGE WAITING
CREATE GROUP	FORM GROUP
CREATE REFERENCE	MAKE REFERENCE
CREATE APPOINTMENT	TAKE APPOINTMENT
MODIFY MESSAGE WAITING	CORRECT MESSAGE WAITING
MODIFY GROUP	MODIFY GROUP
MODIFY REFERENCE	TRANSFORM REFERENCE
MODIFY APPOINTMENT	CHANGE APPOINTMENT

TAG Descriptions

SIMPLE TASKS (same for both namesets)

send-message-waiting {Operation = send, Object = waiting-message}
send-message-received {Operation = send, Object = received-message}
file-message-waiting {Operation = file, Object = waiting-message}
file-message-received {Operation = file, Object = received-message}
file-message-mailed {Operation = file, Object = mailed-message}
create-message-waiting {Operation = create, Object = waiting-message}
create-group {Operation = create, Object = group}
create-reference {Operation = create, Object = reference}
create-appointment {Operation = create, Object = appointment}

RULE SCHEMAS (Structured nameset)

7.1 Task [Operation, Object] → name1 [Operation] + name2 [Object]
7.2 name1[Operation] → known-item [Kind-word, Operation]
7.3 *name1[Operation = send] → "SEND"
7.4 *name1[Operation = file] → "FILE"
7.5 *name1[Operation = create] → "CREATE"
7.6 *name1[Operation = modify] → "MODIFY"
(name2[Object] → THE SAME FOR BOTH NAMESETS)

RULE SCHEMAS (Unstructured)

7.7 Task [Operation, Object] → name1 [Operation, Object] + name2 [Object]
7.8 name1[Operation = send,Object = waiting-message] → "SEND"
7.9 name1[Operation = send,Object = received-message] → "TRANSMIT"
7.10 name1[Operation = file,Object = waiting-message] → "FILE"
7.11 name1[Operation = file,Object = received-message] → "CLASSIFY"
7.12 name1[Operation = file,Object = mailed-message] → "STORE"
7.13 name1[Operation = create,Object = waiting-message] → "WRITE"
7.14 name1[Operation = create,Object = group] → "FORM"
7.15 name1[Operation = create,Object = reference] → "MAKE"
7.16 name1[Operation = create,Object = appointment] → "TAKE"
etc. for modify operations . . .
(name2[Object] → THE SAME FOR BOTH NAMESETS)

Figure 8. **Consistent versus inconsistent syntax. Consider a command language in which each command is issued by typing a verb and two parameters, one of which is common to all commands in the language (cf. Barnard et al., 1981). Two separate commands from such a language are shown for a consistent and inconsistent dialect (the argument extends obviously to three or more commands). According to the metrics developed in the text, the consistent dialect will be easier to learn because it can be fully described using fewer simple-task rewrite rules.**

Consistent Dialect

verb1 common-parameter var-parameter1
verb2 common-parameter var-parameter2

Inconsistent Dialect

verb1 common-parameter var-parameter1
verb2 var-parameter2 common-parameter

TAG Definitions

SIMPLE TASKS (same for both languages)
task1 {(Feature = f1}
task2 {Feature = f2}

RULE SCHEMAS (Consistent dialect)
8.1 Task [Feature] →
 name [Feature]
 \+ common-parameter
 \+ var [Feature]
8.2 name [Feature = f1] → verb1
8.3 name [Feature = f2] → verb2
8.4 var [Feature = f1] → var-parameter1
8.5 var [Feature = f2] → var-parameter2

RULE SCHEMAS (Inconsistent dialect)
8.6 Task [Feature = f1] → verb1 + common-parameter + var-parameter1
8.7 Task [Feature = f2] → verb2 + var-parameter2 + common-parameter

schemas are required for each simple-task, the abbreviation conventions can be absorbed into them, and no additional rules are needed.

TAG also makes predictions about the optimal strategies for abbreviating command names. Evidence has accumulated recently to suggest that, for encoding operations at least, the preferred abbreviation algorithm is simple truncation (e.g., Hirsch-Pasek, Nudelman, & Schneider, 1982). However, in life-size languages, any such simple scheme will lead to clashes, where two command names suggest the same abbreviation. The only suggestion in the literature known to the authors for resolving conflicts is to utilize a secondary rule (perhaps vowel deletion) for clashing items (Ehrenreich & Porcu, 1982). Certainly this strategy appears to be better than using a minimum-to-distinguish

***Figure 9.* Example commands from the three languages used by Payne (1985) in an experiment on semantic-syntactic alignment. These commands were used as examples in the learning phase of the experiment.**

Language 1

DOG BOXER HIGH BROWN	"high priority request for information about a missing brown boxer dog"
CAR FORD VERY LOW D	"very low priority request for information about a D registered Ford"
FIN CASH VERY HIGH 10	"very high priority request for information about ten pounds cash lost"
JEW BROOCH LOW SILVER	"Low priority request for information about jewelry, in particular, a silver brooch"
TOM PEEP HIGH PPP	"high priority message to Tom Peep, whose password is PPP"
HEAD LOW HULL	"low priority message to the head of the Hull office"
BLUFF VERY LOW DEFS	"very low priority, play the definitions version of Call My Bluff" (the other version was WORDS)
QUOTE LOW AUTS	"low priority, play the authors verion of the quotations game" (the other version was QS for quotes)

Language 2

DOG BOXER BROWN HIGH
CAR FORD D VERY LOW
FIN CASH 10 VERY HIGH
JEW BROOCH SILVER LOW
TOM PEEP HIGH PPP
HEAD LOW HULL
VERY LOW BLUFF DEFS
LOW QUOTE AUTS

Language 3

DOG HIGH BOXER BROWN
CAR FORD VERY LOW D
FIN VERY HIGH CASH 10
JEW BROOCH LOW SILVER
HIGH TOM PEEP PPP
HEAD LOW HULL
VERY LOW BLUFF DEFS
QUOTE LOW AUTS

truncation algorithm throughout the command set (Ehrenreich & Porcu, 1982). However, TAG predicts that use of a secondary rule will be a poor technique, as the conflict set need bear no relation to the semantic organization of the tasks: TAG has no mechanism for capturing regularities between arbitrary (nonsemantic) groupings of tasks. Instead, TAG predicts that abbreviation conflicts should be solved by splitting the command language into separate task-oriented categories and utilizing a different abbreviation rule within each

Figure 10. **TAG definitions for the three languages in Payne's (1985) experiment on semantic-syntactic alignment. In this grammar, lower case terms in quotes describe action variables, which must be replaced by exemplars (see Section 2.2.).**

SIMPLE TASKS (the same for all three languages)

request-info-about-dog {Category = info, Type = dog}
request-info-about-financial-matters {Category = info, Type = fin}
request-info-about-cars {Category = info, Type = car}
request-info-about-jewelry {Category = info, Type = jew}
send-public-message {Category = message, Type = public}
send-private-message {Category = message, Type = private}
play-call-my-bluff {Category = game, Type = bluff}
play-quotations {Category = game, Type = quote}

RULE SCHEMAS, Language 1

10.1 Task [Category, Type] → header[Category, Type]
+ descriptor[Type]
+ priority + parameter[Type]
10.2 header[Category = info, Type] → three-letter-header [Type]
10.3 header[Category = message, Type] → NULL
10.4 header[Category = game, Type] → NULL
10.5 three-letter-header [Type = car] → "CAR"
10.6 three-letter-header [Type = fin] → "FIN"
10.7 three-letter-header [Type = jew] → "JEW"
10.8 three-letter-header [Type = dog] → "DOG"
10.9 descriptor [Type = dog] → "dog-breed"
10.10 parameter [Type = dog] → "dog-colour"
10.11 descriptor [Type = fin] → "valuable-object"
10.12 parameter [Type = fin] → "monetary-amount"
10.13 descriptor [Type = jew] → "jewelry-item"
10.14 parameter [Type = jew] → "material"
10.15 descriptor [Type = car] → "car-make"
10.16 parameter [Type = car] → "registration-letter"
10.17 descriptor [Type = public] → "job"
10.18 parameter [Type = public] → "town-office"
10.19 descriptor [Type = private] → "name"
10.20 parameter [Type = private] → "password"
10.21 descriptor [Type = quote] → "QUOTE"
10.22 parameter [Type = quote] → "quote-game-type"
10.23 descriptor [Type = bluff] → "BLUFF"
10.24 parameter [Type = bluff] → "bluff-game-type"
10.25 priority → "VERY HIGH"
10.26 priority → "VERY LOW"
10.27 priority → "HIGH"
10.28 priority → "LOW"

RULE SCHEMAS, Language 2

10.29 Task [Category = info, Type] → header [Type]
+ descriptor [Type]
+ parameter [Type]
+ priority

Figure 10. **(Continued)**

```
10.30 Task [Category = message, Type] → descriptor [Type]
                                          + priority
                                          + parameter [Type]
10.31 Task [Category = game, Type] → priority
                                       + descriptor [Type]
                                       + parameter [Type]
```

RULE SCHEMAS, Language 3

```
10.32 Task [Category = info, Type = dog] → "DOG" + priority
                                             + "dog-breed" + "dog-color"
10.33 Task [Category = info, Type = fin] → "FIN" + priority
                                             + "valuable-object"
                                             + "monetary-amount"
10.34 Task [Category = info, Type = jew] → "JEW" + "jewelry-item"
                                             + priority + "material"
10.35 Task [Category = info, Type = car] → "CAR" + "car-make"
                                             + priority + "reg-letter"
10.36 Task [Category = message, Type = private] → priority + "name"
                                                    + "password"
10.37 Task [Category = message, Type = public] → "town-office"
                                                   + priority + "job"
10.38 Task [Category = game, Type = quote] → "QUOTE" + priority
                                               + "quote-game-type"
10.39 Task [Category = game, Type = bluff] → priority + "BLUFF"
                                               + "bluff-game-type"
10.40 priority → "VERY HIGH"
10.41 priority → "VERY LOW"
10.42 priority → "HIGH"
10.43 priority → "LOW"
```

category. There are obviously begged questions in this recommendation, such as what to do about remaining clashes, and how to cope with very large languages of many semantic categories; it may well be that such problems need to be solved at a level above the choice of abbreviations. Nevertheless, the suggestion seems useful in many circumstances, and is supported by an experiment reported by Payne (1985).

In that experiment subjects learned a command language to manipulate small graphic symbols (a pointer and some blocks) on the display of a microcomputer. For one group, the task was described as a robot game, the pointer representing a robot and the blocks being mines. For the second group, instructions were abstract. This instructional manipulation does not concern us here, for the command names remained identical, abbreviation conditions were fully crossed with instructions, and no interactions emerged in the results. The experimental command language consisted of 18 commands, for moving the robot around, adjusting position of the robot's claw, and manipulating the mines. The namesets were extended versions of the congruent namesets used by Carroll (1982). The learnability of two separate abbreviation schemes was compared. The first scheme, Secondary Rule (SR), adopted two-letter truncation as the primary rule and utilized vowel deletion as a secondary rule in the situations in which truncations led to clashes (5 of the 18 commands). Choosing members of the conflict set to be subject to the secondary rule was done ad hoc, but with the important constraint that the exceptions should be spread across the semantic categories of the language, to allow a distinction with the second scheme. The second scheme, Categories with Exception (CE), adopted rules in the manner advocated by the TAG model. This scheme used two-letter truncation for commands that moved the robot without directly acting on a mine (12 commands), and vowel deletion for commands that directly manipulated mines. Even this scheme led to a single clash in the truncation abbreviations, so one of the 12 movement commands, PUSH, was abbreviated by vowel deletion.

An important point to note about the two abbreviation schemes is that the CE scheme, which TAG predicts is better, actually contains more items using the worst abbreviation algorithm (vowel drop). Further, for no less than 12 of the 18 commands, abbreviations were the same under the two schemes.

The results of the experiment showed a reliable advantage for the CE scheme. Subjects learning abbreviations generated by this scheme made fewer abbreviation errors, consulted the help facility on fewer occasions, and actually solved the problems more efficiently. This last finding is particularly important, because it shows an influence of language structure on deeper aspects of human-computer interaction.

5. THE COGNITIVE SCIENCE BASIS OF THE TAG NOTATION

In this section we look at the particular devices utilized by TAG and examine their relation to various cognitive science concerns. We begin with the thorny question of task analysis, describing our approach to simple-tasks. We next discuss the use of semantic features to describe task concepts and to mark rewrite rules. We raise the issue of notational power, and finally discuss a possible extension to TAG's notation.

TAG makes two novel contentions about the representation of tasks. First, it identifies the special simple-tasks that can be performed without any problem solving or iteration. Second, it represents simple tasks as concepts, whose semantic interrelationship plays a crucial role in the representation of the language.

5.1. Simple-Tasks

We have defined a simple-task as any task that the user can routinely perform with no demand for a control structure, such as branching or iteration, that requires monitoring of plan progress. We believe that higher-level, more complex, tasks requiring coordination between task sequences are best handled by a separate planning component. Our motivation for this needs a little unpicking.

First and foremost, we argue that this class of simple-tasks is psychologically important. Against this, critics may claim that, as simple iteration can be easily routinized (Card, Moran, & Newell, 1983) and as we distinguish iterative from noniterative tasks, the class of simple-tasks cannot be a psychologically relevant category. This argument has some force for task *performance*, but for task language *learning* it is faulted. A novice user who has been told the command for, say, deleting words, will induce the iterative method for deleting sentences with little trouble. This point has been demonstrated empirically by Douglas (1983), who showed that novices could correctly induce how to perform any task that relied on a simple combination of tutored tasks. Our focus on the prediction of learning effort renders such distinctions crucial: Simple-tasks are the set of tasks for which distinct action sequences have to be learned, or induced, from the particular structure of the task language.

We should note at this stage that if TAG is eventually to play a role in theories of performance, as we intend, simply ignoring iteration will not do. A tactic of "leaving iteration to the planning system" may handle the repetition of entire task-action sequences, but it would be critically weak in one important respect: Many tasks have iterable subcomponents. For example, to format disks on the CPM operating system one must run the Format program, and respond to the prompt that yes, you really do want to format the disk in Drive B. To format more than one disk, one could, of course, exit the Format program and loop through the entire sequence, but an option is supplied to allow users to efficiently repeat the last step only. Our current thoughts on planning with task-action grammars suggest tackling this problem by allowing control tokens on subtasks and action specifications, such as can-be-iterated. These developments will not be discussed in this article, because they do not affect our central concerns with learning and learnability.

The concern with learnability also dictates the second property of simple-tasks. The simple-tasks for a given system are determined as much by the device as by the external task domain. In Moran's (1983) terms, simple-tasks are "internal" rather than "external" tasks. To adapt one of Moran's examples, consider a very simple cut-and-paste display editor. Although in the external task domain the user can distinguish between such entities as sentence and paragraph, in the internal world of the system this distinction may disappear; both words and sentences are treated as strings. For example, to delete a sentence one has to mark the beginning and end with the mouse and choose the cut command. The job of a task-action grammar in this case is to describe the operation sequence required for the "delete string" command and its relation to other aspects of the task language, not to illuminate the nature of the mapping from external task to internal task.

Our position is in close agreement with Moran (1983), in that we see the need for a psychological mapping from external to internal tasks as well as the acquisition of task language semantics and syntax. This view agrees with our claim that simple-tasks are a psychological construct.

Simple-tasks, then, are equivalent entities in human-computer interaction to operators in the classical problem space view of problem solving (Newell & Simon, 1972). It is instructive to explore this connection.

One obvious difference between human-computer interaction and the kind of puzzles studied by problem-solving theorists is the need for a task language to map the operators onto action sequences. The psychological implications of this mapping are the central concern of the TAG model. To address this issue we have found it necessary to depart from the treatment of operators in the puzzle-solving literature. Puzzle-solving operators are more or less atomic entities, whose use is determined by preconditions and postconditions: The relationship of one operator to another is not explicitly represented, except with regard to roles on solution paths (e.g., the preconditions of one operator may become the goal state which prompts application of a second). In contrast, TAG treats simple-task operators as semantic concepts which are organized into mental categories.

To describe learning effects in problem-solving domains, it is necessary to allow operators to be chunked into macro-operators (e.g., Chase & Simon, 1973). We regard simple-tasks to be dynamic in a similar way. For the novice user, simple-tasks are, roughly, all those tasks for which there is a distinct command or operation in the task language (and which have been learned). It is this level of analysis that we have found to be most useful in assessing the learnability of task language designs. For the more advanced user, several simple tasks may have been composed to form more complex tasks that can nevertheless be performed wthout a problem-solving effort. We illustrate this view of practice in Section 6.

5.2. Tasks as Concepts

We hypothesize that simple-tasks are mentally represented as concepts. The thrust of this suggestion is that the internal structure of tasks and the intentional relations among tasks are both of major importance in the user's mental representation of task languages.

There exists in the literature a lively debate about the mental representation of concepts (Smith & Medin, 1981). Are concepts represented by sets of defining features, as the classical view maintains (Bruner, Goodnow, & Austin, 1956; Katz, 1972), or as schematic prototypes (e.g., Rosch & Mervis, 1975), or even as networks of exemplars (e.g., Medin & Schaffer, 1978)? So lively is the debate and so difficult the issues that they have led some commentators to conjecture that many important tensions will never be resolved (Armstrong, Gleitman, & Gleitman, 1983). The safest general view of concepts would appear to be a permissive one: All of these representational forms exist, but they are used for different purposes. By taking this stance, one is able to offer a theory of certain aspects of conceptual performance without being necessarily committed to a unitary view of concepts. This is exactly the approach taken by Tversky (1977) in his theory of similarity computations. Following Tversky, TAG utilizes feature-set representations of task concepts, and of lexical command names, without insisting that other representations are redundant or that the debate in the linguistics literature (e.g., Lyons, 1977; Miller & Johnson-Laird, 1976) is dead.

In view of the importance of featural descriptions of concepts in TAG, it is well to be clear exactly what is meant by a feature. The term is being used in exact accord with the conventions of semantic theory—anything that can take a value with respect to a term. (See Rosenberg, 1982, for a rigorous mathematical treatment.)

As in mainstream semantics, the features and components that are specified in a TAG description should have psychological validity, in that they are important for the categorization of the task world. Unfortunately, again just as in semantics, there is nothing that analysts can do to ensure this ideal, except rely on their intuition.

5.3. Feature-tagged Rule Schemas

With regard to their role in the rewrite rules, it may be helpful to view features as strongly typed variables, for which the entire range of values is defined by an *n*-tuple, usually small. The assignment of values to features in TAG rule schemas is therefore parallel to the assignment of values to variables in advanced production system architectures, and indeed sometimes serves similar purposes, such as the capturing of certain kinds of generalization. However, the strong typing of features, and the fact that they play an important role not only in the rule-schemas but also in the categorization of task concepts and of the action world, does mean that rule-schemas are heavily constrained compared to generalized production rules and often make quite different predictions.

The use of semantic features in syntactic rules is a major break from the devices employed by linguists' theories of syntax, most of which stress the role of syntactic features. The break reflects the simplicity of the syntactic structures of task languages relative to natural language. Syntactic features are simply not needed to compactly describe the regularities; but semantic features are needed to express important characteristics of the mental representation of even such simple syntax.

A similar device is employed in the "semantic grammars" used by Burton (1976) to implement natural language dialogues in intelligent computer-assisted instruction and in "attribute grammars" originated by Knuth (1968) as a means of specifying the "semantics of context-free languages."

Attribute grammars supply a corresponding semantic rule for every rewrite rule, specifying the attributes of the left-hand side nonterminal in terms of the attributes of the right-hand side, or some already meaningful symbol (e.g., a number). This technique allows the specification of a language's intentional semantics. Our convention of associating values to attributes consistently throughout a rule is a very limited version of this idea. The limitation represents an important constraint: that the componential semantics of an interactive command can be derived in a simple additive fashion from the command's constituents. Further research is required to investigate the validity and implications of this constraint; it may prove advantageous in the long term to provide separate semantic rules in the analysis.

5.4. The Competence Hypothesis and Formal Power

In identifying and defining the class of simple-tasks we hope to clarify the important distinction between the user's knowledge of the task language and the goal-driven problem solver (unspecified but constrained by the TAG model) which interprets that knowledge. The separation distinguishes this work from other attempts at formal modeling in human-computer interaction (Card et al., 1983; Kieras & Polson, 1985) but dovetails with current thinking in computational linguistics, where it is dubbed the competence hypothesis: "A reasonable model of language use will incorporate, as a basic component, the generative grammar that expresses the speaker–hearer's knowledge of the language . . . " (Chomsky, 1965, p. 9, quoted in Bresnan & Kaplan, 1983).

The competence hypothesis affords us a crucial advantage by enabling a model of the user's knowledge with a tightly specified and very limited formal power, despite its large expressive power. Previous attempts at modeling human-computer interaction have not been able to offer this degree of restraint, instead offering systems of unspecified and unclear, but worryingly powerful, capabilities.

The benefits of restricting the formal power of computational models may not be immediately apparent. After all, both production systems and semantic networks have unlimited computational power, yet are regarded as successful models by many. We do not want to suggest that unrestricted power necessarily removes empirical content. Consideration of strong equivalence (as opposed to mere duplication of input/output pairs) disallows such a simple argument (see

Pylyshyn, 1980, 1984); production systems may accurately predict the time-complexity of different mental operations. Nevertheless, the formal power of metalanguages is a particularly important flavor of theoretical parsimony. For if a metalanguage is to prove useful as a theory, it must be applied to a large number of different language constructs (in our case, different task languages). Yet the more powerful the metalanguage, the greater the choice of grammatical descriptions of any given construction—the degrees-of-freedom problem we discussed above. As Pylyshyn (1980) put it: "The more constrained a notation or architecture, the greater the explanatory power of resulting models. It [the architecture or metalanguage] provides a principled rationale for why the model [the grammar] takes one particular form, as opposed to other logically possible ones" (p. 126).

In our work on TAG we have only managed to go part way to this ideal, and so we have adopted "style rules" such as minimizing the number of simple-task rule schemas to further constrain possible descriptions. TAG itself is a highly constrained metalanguage (as we have seen, it only has context-free power); we believe that it is constrained according to psychologically plausible mechanisms.

5.5. A Possible Extension

We fully expect that the TAG notation will need to be developed and extended to tackle more diverse issues than we have yet been able to consider. One such extension from the core context-free grammar is required to deal with command name abbreviation algorithms. As we have already seen, people learning abbreviations can capitalize on consistent algorithms used in their generation (Hirsh-Pasek et al., 1982). The obvious way to represent such algorithms in TAG is to allow functions, such as **take-first-three-letters**, to operate on tokens of the grammar. (This approach was adopted to formalize the predictions in Payne's, 1985, experiment on abbreviations, summarized in Section 4.2.)

There are some problems with this extension. First, the complexity of the functions will not be apparent at all if the functions are merely denoted symbolically. Against this, it may be possible to assign unanalyzed complexity indices to the functions on the basis of empirical guidelines—certainly this is true of abbreviation algorithms. Second, embedded functions can arbitrarily increase the power of the notation. To maintain a maximally constrained notation we must define a limited class of permitted functions.

The most straightforward way to define a class of permitted functions is to restrict the type of the input and output parameters. Abbreviation rules always take a word or phrase (a string of characters) and return a different string of characters. The only other functions that we have yet found use for in TAG analyses are some suggested by the psychology of pattern perception, such as taking the next item in a defined alphabet. All these functions are syntactic functions, in the sense that they transform input parameters independently of meaning. (Of course, all effective procedures are purely syntactic at some level, by definition, but our proposed constraint goes deeper by specifying that the transformation performed by the function must remain syntactic when described at the level of tasks and actions.)

We do not pretend to yet have a fully specified theory for allowing and disallowing embedded functions in TAG descriptions, but we do feel that embedding could prove a useful technique for integrating TAG with further, as yet unconsidered aspects of users' mental representation.

6. TAG'S RELATION TO LEARNING AND PERFORMANCE THEORIES

In this section we endeavor to demonstrate that TAG is compatible with current psychological notions of learning and performance.

6.1. Learning

We illustrate TAG's relation to models of learning with some further examples, based on a description of EG, the example message system used by Moran (1981) in his presentation of the Command Language Grammar (CLG). Figure 11 shows a TAG description of EG. It is immediately noticeable from the figure how much more compact this description is than the CLG version; but, of course, CLG expresses some aspects of the interface that TAG ignores on principle—particularly the expert user's learned methods.

Following Moran (1981) we consider learning in terms of the simple framework of Rumelhart and Norman (1978), who distinguished three modes of learning: accretion (basically, the addition of new elements of knowledge without affecting existing knowledge), tuning (changes to the form of representation of knowledge to improve economy, robustness, and performance), and restructuring (the modification of existing knowledge in the light of new). All three kinds of learning could play a part in the dynamics of a user's task-action grammar.

First, the user may learn how to perform simple-tasks that are unrelated to those already learned. A new entry will be added to the simple-task dictionary, and a new rule-schema added to the task-action mappings. Because of the emphasis TAG places on the semantic organization of the entire task world and the use of discriminating features in the task dictionary, we posit that this kind of bald accretion is rare in task language learning, except when learning big, seriously inconsistent systems such as Unix.

Second, tuning can be represented in TAG by allowing simple-tasks to be composed into new (macro-operator) simple-tasks. This device may allow some

Figure 11. **A task-action grammar of Moran's (1981) EG message system. (The token message-no is an action variable which must be replaced with an exemplar.)**

TASK FEATURES

Feature	Possible values
Context	operating-system, EG
End-state	operating-system, EG
Condition	new-mail, nil
Delete/Display	delete, display
Specifier	number, next

SIMPLE TASKS

Enter {Context = operating-system, End-state = EG, Condition = nil}
Enter-if-new-mail {Context = operating-system, End-state = EG,
Condition = new-mail}
Delete-current-message {Context = EG, End-state = EG,
Delete/Display = delete
Display-next-message {Context = EG, End-state = EG,
Delete/Display = display, Specifier = next}
Display-specified-message {Context = EG, End-state = EG,
Delete/Display = display, Specifier = number}
Quit {Context = EG, End-state = operating-system}

RULE SCHEMAS

11.1 Task [Context = operating-system, Condition] →
"EG" + extra[Condition] + "RETURN"
11.2 extra [Condition = nil] → nil
11.3 extra [Condition = new-mail] → "/N"
11.4 Task [Context = EG, Delete/Display = delete] → "D"
11.5 Task [Context = EG, Delete/Display = display, Specifier = number] →
"M" + "message-no" + "RETURN"
11.6 Task [Context = EG, Delete-Display = display, Specifier = next] → "N"
11.7 Task [Context = EG, End-state = operating-system] → "Q"

aspects of practice to be captured by TAG, but any analysis of this possibility is beyond the scope of this article. We can exemplify the idea of simple-task composition in the EG system. Imagine that a frequent task was to enter EG and read a single message before exiting again (to use a different software system perhaps). The required sequence of actions requires no control structure, so it can be represented in a TAG rewrite rule, as shown in Figure 12. Composition of this kind bears a close relation to the production system learning mechanisms of Anderson (1982, 1983).

Finally, new simple-tasks that are related to those already learned may be encountered, calling for restructuring of the task-action grammar. If we assume that a user has already learned the EG system described in Figure 11, what would happen if the system were extended by a command to display a message from a specified sender? The new simple-task is clearly related to show-

Figure 12. **Tuning the task-action grammar for EG to learn the macro-operator of entering EG, reading a particular message, and exiting to the operating system.**

ADDITIONAL SIMPLE TASKS
Enter-read-exit {Context = operating-system,
End-state = operating-system,
Condition = nil,
Delete/Display = display,
Specifier = number}
ADDITIONAL RULE SCHEMAS
12.1 Task [F(Enter-read exit)] →
Task [Context, Condition]
+ Task [Delete/Display, Specifier]
+ Task [End-state]

message-number; it shares most components, but it differs in terms of its specifier. The required additions/alterations to the grammar are shown in Figure 13.

Even this simple example of restructuring highlights some interesting issues. It seems clear that very little active generalization would be required, and one wonders whether, in some respects, Rule 13.1 (Figure 13) was there all along. One might conjecture a learning mechanism in which every task-action mapping was represented as a maximally general hypothesis, in keeping with the ideas on active learning of Carroll and Mack (1985) and Hayes-Roth (1983). The alternative is a system that learns single task-action mappings in a rather rigid way, and that requires an effort of generalization to represent a second mapping, with the third and subsequent related mappings being easier to learn. This mechanism is closely related to suggestions for learning in a production system framework (e.g., Anderson, 1983).

6.2. Performance

Although TAG is a competence theory, it necessarily constrains performance. Indeed, in contrast to most generative grammars, TAG can be incorporated into a performance theory straightforwardly because it defines a mapping from tasks to actions (rather than merely generating all possible action sequences).

Task-action grammars are task based; the rules are driven by state transformations that the user knows how to effect. Performance, on the other hand, is goal based; it is driven by end-states which the user wishes to achieve. To perform using task-action grammar knowledge, users must assess the current state with respect to their goal state, and devise a sequence of simple-tasks that will transform one into the other. In other words, as stated before, simple-tasks play a similar role to operators and macro-operators in classical problem-solving ar-

Figure 13. **Restructuring the task-action grammar for EG to learn a new command for displaying a message from a particular author. Note that Rule 11.5 is no longer needed. (The tokens message-no and author-name are action variables which must be replaced with exemplars.**

ADDITIONAL FEATURES

Feature	Possible values
Specifier	number, next, author

ADDITIONAL SIMPLE TASKS

Display-message-by-author {Context = EG, End-state = EG,
Delete/Display = display, Specifier = author}

ADDITIONAL RULE SCHEMAS

13.1 Task [Delete/Display = display, Specifier →
"M" + id [Specifier] + "RETURN"

13.2 id [Specifier = number] → "message-no."

13.3 id [Specifier = author] → "author-name"

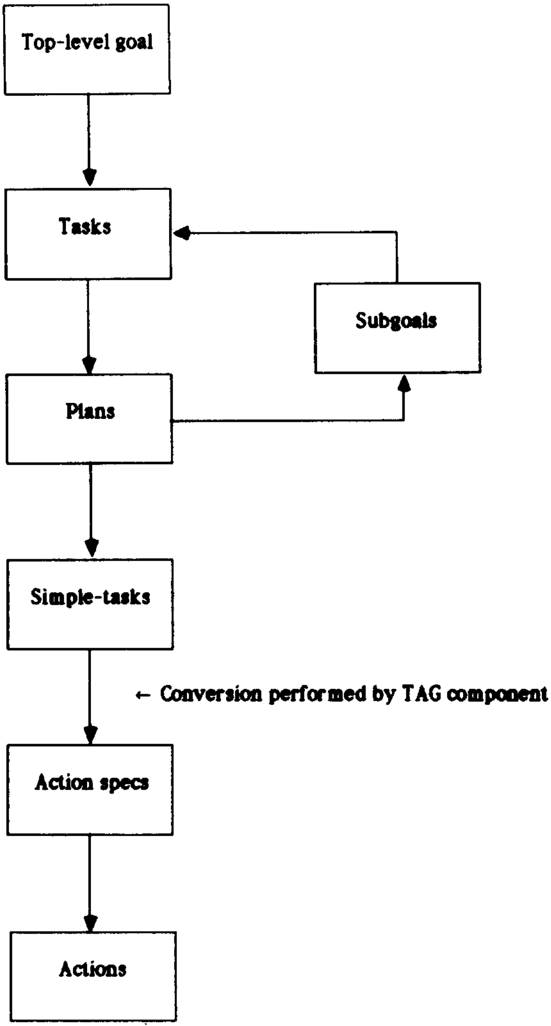
chitectures. Two additional stages are required: (a) Users must expand all the simple-tasks into action specifications, using the task-action grammar; and (b) they must produce physical actions from the action specifications.

Performance can therefore be represented by the schematic flow diagram in Figure 14.

As shown in the figure, goals are combined with knowledge of the current state to provide tasks (and, sometimes, to pass parameters to the grammar; see the discussion of action variables in Section 2.4); plans are generated for the performance of tasks, typically consisting of subgoals, which in turn must be transformed into tasks. At the bottom level, a plan is some structure of simple-tasks (possibly a single simple-task); every simple-task can be performed by "running" the task-action grammar, and interpreting the resulting action specification. Of course, this control cycle is subject to the usual iterations and interactions and must not be thought of as operating in strictly serial stages. Our restricted definition of simple-tasks allows us to assume that any online monitoring of performance is done outside the task-action grammar, by the planner or by the action interpreter.

This picture is not meant to grind any particular theoretical axe; it is intended to present a noncontroversial view of goal-based performance. It illustrates the way in which "weak" problem-solving theories (e.g., Laird & Newell, 1983; Newell & Simon, 1972) may be extended to deal with the medium of a task language. It also highlights a major distinction between our endeavor and the models of Card et al. (1983) and Kieras and Polson (1985); namely, that their process models treat all aspects of the user's performance from high-level planning down to low-level interaction, whereas task-action grammars simply describe knowledge of task languages in a way that can be utilized by some unspecified planning system.

Figure 14. A simple schematic of goal-oriented performance with a computer system.



7. CONCLUSIONS

We do not view TAG as a fixed and immutable notation. Instead, we present TAG as an advance in the formal definition of human-computer interfaces. Many inadequacies, however, will only be addressed by extensions of the current theory.

TAG's strength is that it allows reasonably compact definitions of task languages that are sensitive to structural properties perceived by users. This strength enables simple metrics over grammars to predict relative learnability of different interface designs. These metrics are indices of complexity based on configural properties of the target language. Further aspects of learnability and usability may be dependent on the internal structure of single rules. Clearly, the current specification of TAG does not allow any assessment of tradeoffs between configural properties and microstructure issues. The issues themselves can be addressed, but only by developing an extra layer of interpretation to be applied by the analyst to the rule representations of the model. TAG has been designed to make the complexity of the entire language explicit; it is hypothesized to be related directly to the number of simple-task rule schemas. TAG therefore offers an explanation of configural effects on complexity. To offer a similar explanation of complexity effects due to individual rule schemas, it would be necessary to develop detailed low-level theories about how the rules are mentally processed, for example how a right-hand side is activated and how a left-hand side is accessed. No effort has been directed to this enterprise for two reasons: (a) We believe that configural aspects of languages typically outweigh microstructures of individual rules and (b) we are eager to keep the TAG notation compact so that it may provide a usable tool for designers.

We feel that the simplicity and compactness of TAG distinguishes our attempt to provide formal tools for the assessment of psychological complexity from the related enterprises of Card et al. (1983), Moran (1981), and Kieras and Polson (1985). The simplicity of TAG has been bought at a price; in particular, we have been content to make predictions about relative complexity of designs, rather than providing quantitative measures. Because of this focus, TAG is at its most useful when comparing task languages that are similar in most respects. However, a second utility of TAG is for analyzing alternative organizations of the same language. We exemplified this approach with our MacDraw analysis in Section 2.4.

Our plans for extending TAG are to address some of the many aspects of user interfaces, and of the user's mental representation, that have not yet been considered. A priority is to integrate TAG with a planning system, as suggested in Section 6. A second important concern is with users' conceptions of the external task world and their mental model of the machine. We believe that by advancing a simple metalanguage that is low in formal power, yet gains much in expressive power from a treatment of the semantics of tasks, we have provided a good platform for such developments.

Acknowledgments. The authors are especially grateful to Tom Moran for detailed critical comments on an earlier draft. Helpful comments have also been received, at various stages of the work, from Don Norman, Phil Johnson-Laird, and Tim O'Shea.

REFERENCES

Anderson, J. R. (1982). Acquisition of cognitive skill. *Psychological Review, 89,* 369–406.

Anderson, J. R. (1983). *The architecture of cognition.* Cambridge, MA: Harvard University Press.

Armstrong, S. L., Gleitman, L.R., & Gleitman, H. (1983). What some concepts might not be. *Cognition, 13,* 263–308.

Barnard, P. J., Hammond, N. V., Morton, J., Long, J., & Clark, I. A. (1981). Consistency and compatibility in human-computer dialogue. *International Journal of Man-Machine Studies, 15,* 87–134.

Black, J. B., & Moran, T. P. (1982). Learning and remembering command names. *Proceedings of the CHI '82 Conference on Human Factors in Computer Systems,* 8–11. New York: ACM.

Bresnan, J., & Kaplan, R. M. (1983). Introduction: Grammars as mental representations of languages. In J. Bresnan (Ed.), *The mental representation of grammatical relations* (pp. xvii–lii). Cambridge, MA: MIT Press.

Bruner, J. S., Goodnow, J., & Austin, G. (1956). *A study of thinking.* New York: Wiley.

Burton, R. R. (1976). *Semantic grammar: An engineering technique for constructing natural language understanding systems* (BBN Report No. 3453). Cambridge, MA: Bolt Beranek & Newman.

Card, S. K., Moran, T. P., & Newell, A. (1983). *The psychology of human-computer interaction.* Hillsdale, NJ: Lawrence Erlbaum Associates, Inc.

Carroll, J. M. (1982). Learning, using and designing command paradigms. *Human Learning, 1,* 31–62.

Carroll, J. M., & Mack, R. L. (1985). Metaphor, computing systems and active learning. *International Journal of Man-Machine Studies, 22,* 39–57.

Chase, W. G., & Simon, H. A. (1973). Perception in chess. *Cognitive Psychology, 4,* 55–81.

Chomsky, N. (1965). *Aspects of the theory of syntax.* Cambridge, MA: MIT Press.

Douglas, S. (1983). *Learning to text edit: Semantics in procedural skill acquisition.* Unpublished doctoral dissertation, Stanford University, Palo Alto, CA.

Ehrenreich, S. L., & Porcu, T. (1982). Abbreviations for automated systems: Teaching operators the rules. In A. Badre & B. Shneiderman (Eds.), *Directions in human-computer interaction* (pp. 111–135). Norwood, NJ: Ablex.

Green, T. R. G., & Payne, S. J. (1984). Organisation and learnability in computer languages. *International Journal of Man-Machine Studies, 21,* 7–18.

Hayes-Roth, F. (1983). Using proofs and refutations to learn from experience. In R. S. Michaelski, J. G. Carbonell, & T. M. Mitchell (Eds.), *Machine learning* (pp. 221–240). Palo Alto, CA: Tioga Press.

Hirsh-Pasek, K., Nudelman, S., & Schneider, M. (1982). An experimental evaluation of abbreviation schemes in limited lexicons. *Behaviour and Information Technology, 1,* 359–369.

Katz, J. J. (1972). *Semantic theory.* New York: Harper & Row.

Kieras, D. E., & Polson, P. G. (1985). An approach to the formal analysis of user complexity. *International Journal of Man-Machine Studies, 22,* 365–394.

Knuth, D. E. (1968). Semantics of context-free languages. *Mathematical Systems Theory, 2,* 127–145.

Laird, J., & Newell, A. (1983). *A universal weak method* (Tech. Rep. CMU-CS-83-141). Pittsburgh: Carnegie-Mellon University, Computer Science Department.

Lee, J. (1972). *Computer semantics.* New York: Van Nostrand Reinholt.

Lyons, J. (1977). *Semantics (Vols. 1 and 2).* Cambridge: Cambridge University Press.

Medin, D. L., & Schaffer, M. M. (1978). A context theory of classification learning. *Psychological Review, 85,* 207-238.

Miller, G. A., & Johnson-Laird, P. N. (1976). *Language and Perception.* Cambridge: Cambridge University Press.

Moran, T. P. (1981). The command language grammar: A representation for the user interface of interactive computer systems. *International Journal of Man-Machine Studies, 15,* 3-50.

Moran, T. P. (1983). External-internal task-mapping analysis. *Proceedings of the CHI '83 Conference on Human Factors in Computer Systems,* 45-49. New York: ACM.

Newell, A., & Simon, H. A. (1972). *Human problem solving.* Englewood Cliffs, NJ: Prentice-Hall.

Pagan, F. G. (1981). *Formal specification of programming languages: A panoramic primer.* Englewood Cliffs, NJ: Prentice-Hall.

Payne, S. J. (1985). *Task-action grammars: The mental representation of task languages in human-computer interaction.* Unpublished doctoral dissertation, University of Sheffield.

Payne, S. J., & Green, T. R. G. (1983). The user's perception of the interaction language: A two-level model. *Proceedings of the CHI '83 Conference on Human Factors in Computer Systems,* 202-206. New York: ACM.

Pylyshyn, Z. W. (1980). Computation and cognition: Issues in the foundations of cognitive science. *The Behavioural and Brain Sciences, 3,* 111-169.

Pylyshyn, Z. W. (1984). *Computation and cognition: Issues in the foundations of cognitive science.* Cambridge, MA: Bradford Books.

Reisner, P. (1977). Use of psychological experimentation as an aid to development of a query language. *IEEE Transactions on Software Engineering, SE-3,* 218-229.

Reisner, P. (1981). Formal grammar and design of an interactive system. *IEEE Transactions on Software Engineering, SE-5,* 229-240.

Robertson, S. P., & Black, J. B. (1983). Planning units in text editing behaviour. *Proceedings of the CHI '83 Conference on Human Factors in Computer Systems,* 217-221. New York: ACM.

Rosch, E., & Mervis, C. B. (1975). Family resemblance studies in the internal structure of categories. *Cognitive Psychology, 7,* 573-605.

Rosenberg, J. K. (1982). Evaluating the suggestiveness of command names. *Behaviour and Information Technology, 1,* 118-128.

Rumelhart, D. E., & Norman, D. A. (1978). Accretion, tuning and restructuring: Three modes of learning. In J. W. Cotton & R. Klatzky (Eds.), *Semantic factors in cognition* (pp.37-53). Hillsdale, NJ: Lawrence Erlbaum Associates, Inc.

Scapin, D. L. (1982). Generation effect, structuring and computer commands. *Behaviour and Information Technology, 1,* 401-410.

Smith, E. E., & Medin, D. L. (1981). *Categories and concepts.* Cambridge, MA: Harvard University Press.

Tversky, A. (1977). Features of similarity. *Psychological Review, 69,* 344-354.

HCI Editorial Record. First manuscript received January 6, 1986. Revision received July 14, 1986. Accepted by Tim O'Shea and Thomas Moran. — *Editor*