Getting Serious about Parsing Plans: a Grammatical Analysis of Plan Recognition

Marc Vilain

The MITRE Corporation
Burlington Road, Bedford, MA 01730
Internet: mbv@linus.mitre.org

Abstract

This paper is concerned with making precise the notion that recognizing plans is much like parsing text. To this end, it establishes a correspondence between Kautz' plan recognition formalism and existing grammatical frameworks. This mapping helps isolate subsets of Kautz' formalism in which plan recognition can be efficiently performed by parsing.

In recent years, plan recognition has emerged as one of the best-understood frameworks for analyzing goal-directed behavior. Interest in plan recognition has led to the development of diverse recognition strategies.[1] One approach suggested several times is that of parsing plan descriptions (Sidner (1985), Ross & Lewis (1987)). A plan is typically described as a sequence of steps, so interpreting some observations in terms of a plan can naturally be seen as a parsing task wherein observations are lexical tokens and plan libraries are grammars.

My aim in this paper is to explore this parsing view of plan recognition by establishing a formal correspondence between an existing plan formalism and context-free grammars. By working through the details of this correspondence, the paper explores parsing algorithms for plan recognition, and delineates classes of problems for which these algorithms are applicable and tractable.

Underlying this work is the plan recognition formalism of Henry Kautz (Kautz & Allen, 1986; Kautz, 1987). His approach is of particular interest because it is formal and well understood. It is also among the broadest of current formalisms, especially in the expressive richness of its plan representation. Finally, since general plan recognition in Kautz' framework is intractable, there is intrinsic interest in identifying those aspects of his approach that cause this intractability, and those that avoid it.

Kautz' Framework

In his dissertation work, Kautz defines a circumscriptive framework for plan recognition. He starts with a simple frame-like hierarchy of plans which is interpretable by first-order meaning postulates. Through a sequence of circumscriptive minimizations, Kautz "closes" the interpretation of the hierarchy, and thereby introduces an additional set of first-order axioms. This expanded set of axioms enables some of the normally *abductive* aspects of plan recognition to be performed through a now *deductive* process.

The Kautz plan representation

The principal component of the Kautz representation is a hierarchy of event (or plan) types. Plans are hierarchically organized according to two relations, *abstraction* and *decomposition*. The former is a subtype (or IS-A) relation; for example, in Kautz' cooking domain, the MAKE-MEAL plan abstracts the MAKE-PASTA-DISH plan. The second relation, decomposition, is borrowed from the non-linear planning literature (Sacerdoti, 1977; Wilkins, 1984, among many others), and identifies the steps making up a plan. For instance, MAKE-PASTA-DISH decomposes into a first step which is a MAKE-NOODLES plan, and a second, a MAKE-SAUCE plan. Each step is given a designator, so the MAKE-NOODLES step of MAKE-PASTA-DISH might be designated S1, and the MAKE-SAUCE step S2. For more examples, see Figure 1.

A plan hierarchy so defined is axiomatized with two meaning postulates, one per relation. For abstraction, let φ_1 and φ_2 be plans such that φ_1 abstracts φ_2. This is interpreted as

$$\forall\, x \;\; \varphi_1(x) \supset \varphi_2(x)$$

For decomposition, let φ be a plan with steps designated $\sigma_1 \ldots \sigma_n$, each of which is restricted to being a plan of type $\psi_1 \ldots \psi_n$. This is interpreted as

$$\forall\, x \;\; \varphi(x) \supset \psi_1(\sigma_1(x)) \wedge \ldots \wedge \psi_n(\sigma_n(x))$$

Finally, Kautz distinguishes an abstract plan class, END, encompassing those plans that are meaningful ends in and of themselves. MAKE-MEAL is abstracted by END, and is taken to be an independently meaningful plan. In contrast, MAKE-NOODLES is not abstracted by END, and is not considered independently meaningful — it only has meaning as a step of some other plan.

Minimal plan models

The bulk of Kautz' work formalizes the notion that plan hierarchies such as these can be treated as a complete encoding of a system's knowledge of plans. Kautz shows that a sequence of circumscriptive minimizations enforces a closed world assumption of sorts for plan hierarchies, with the effect that the hierarchies can

[1] E.g., Schmidt, Sridharan & Goodson (1978), Allen & Perrault (1980), Carberry (1983), Allen (1983), Litman (1986), Pollack (1986), Kautz (1987), Konolige & Pollack (1989), Goodman & Litman (1990).

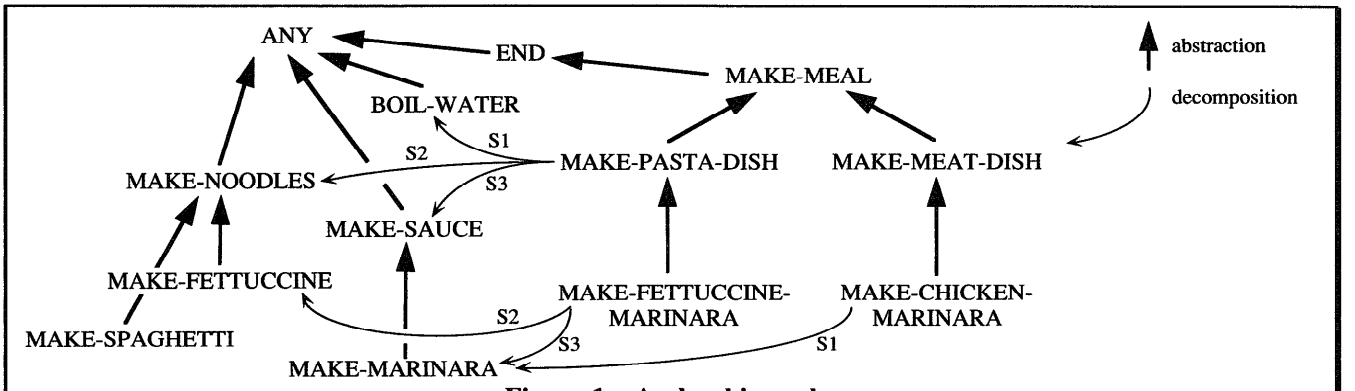

Figure 1: A plan hierarchy

be used to guide plan recognition. Briefly, the closure of a hierarchy proceeds by selecting among models of the hierarchy those models that minimize (1) the extensions of all non-leaf plan types (the hierarchy's inner nodes); (2) the extensions of all plan types but ANY, the hierarchy's root; and (3) the extensions of all non-END plan types.

In effect, the first two minimizations enforce the assumption that no abstraction relations hold over the hierarchy which aren't explicitly mentioned or derivable by transitive closure. The last minimization corresponds to assuming that non-END plans can only occur as components of some other plan. Kautz calls the resulting closed world models of a hierarchy its *covering models*.

Recognizing Kautz' plans

Since they provide a closed world encoding of the hierarchy, covering models form the foundation of the plan recognition process. The recognition problem intuitively consists of finding the most parsimonious interpretation of some observed actions, given the closed world of the hierarchy.

To formalize this notion, let H be a hierarchy described by A and D, the conjunctions of H's abstraction and decomposition axioms respectively, and let ω be a sentence describing some observations. From among the covering models of $A \wedge D \wedge \omega$, Kautz designates those models that explain ω parsimoniously as those that minimize the cardinality of END. This selects those closed world models that postulate the minimum number of end events from H that can possibly account for the observations. To extend Kautz' terminology, I will refer to these models as the minimal covering models of ω with respect to H.

Operationally, a plan recognizer doesn't manipulate models so much as plan descriptions. So in Kautz' framework, one should think of the plan recognizer's task as the mapping of an observation sentence ω to a plan sentence π that is the strongest description of the plans underlying ω, and is valid in all minimal covering models of ω with respect to the plan hierarchy.

Plan Hierarchies as Grammars

There is much similarity between Kautz' plan recognition assumptions and the assumptions underlying parsing. A parser interprets a grammar, in some sense, as a closed world; constituents can't be derived unless they're on the left hand side of a parse rule; constituents that don't derive the start symbol directly must appear in the derivation of some other constituent; *etc.* As noted above, several authors have observed that plan hierarchies can be seen as defining a grammar of plans that could be used to "parse" a string of actions.

This observation is attractive: parsing is a well-understood problem with efficient solutions, so a parsing approach to plan recognition might be expected to yield recognition strategies with better computational characteristics than those developed by Kautz. For certain classes of recognition problems, this is in fact true. With appropriate restrictions on the nature of the plan hierarchies, observations, or expected solutions, a broad class of plan recognition problems becomes tractable in a parsing framework.

Initial Plan Grammar Considerations

These tractability results are shown by constructing a mapping from Kautz' plan hierarchies to context-free grammars. Note that context-free grammatical power is not strictly necessary to encode plan hierarchies in Kautz' representation, since he nominally restricts them to being acyclic. An acyclic hierarchy does not contain any recursive plan definitions, and could in fact be encoded as a regular (finite-state) grammar. The broader context-free class is chosen here in part to allow for the possibility of recursive definitions.

Further, since a major aspect of plan recognition is recovering the plan structure underlying one's observations, the natural grammatical implementation of plan recognition is as a chart-based parser (Kay, 1980; Thompson, 1983). In particular, one can extend Earley's context-free recognition algorithm (Earley, 1970) with a chart, thus enabling it to return the structure of its

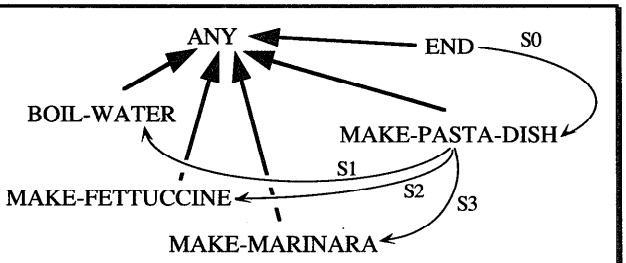

Figure 2: A very simplified hierarchy.

parses. Given such an algorithm, there is little motivation to restrict oneself to a finite-state encoding of plans. This is especially true with Earley's algorithm, since its polynomial bound on context-free recognition time reduces to a linear bound for finite-state grammars.

The complete mapping from plans into grammars is fairly involved, however. So for the sake of clarity, it is presented here as a sequence of simpler mappings into ever more expressive grammatical formalisms. As an overall simplifying assumption, I'll start by only considering solutions that neither share nor interleave steps. In other words, if more than one plan must be hypothesized to account for some observations, each plan must be fully completed before the next one can begin.

The Decomposition Grammar

The process of converting a plan hierarchy *H* into an equivalent grammar *G* starts with an encoding of the decomposition axioms. Assume for now that the description of the hierarchy contains no abstraction axioms (we can safely ignore the fact that all plan types are actually abstracted by ANY, the hierarchy's root). The hierarchy is thus entirely described by axioms of form

$$\forall x \ \varphi(x) \supset \psi_1(\sigma_1(x)) \wedge \ldots \wedge \psi_n(\sigma_n(x))$$

where φ is a plan, $\sigma_1 \ldots \sigma_n$ are its steps, and $\psi_1 \ldots \psi_n$ are the type restrictions of these steps. In general, these steps may only be partially ordered, and may even overlap if actions are modeled as having non-zero temporal extent. However, as another temporary simplification, assume that the steps of a decomposition are non-overlapping, and totally ordered according to their order in the decomposition axiom.

With these simplifications, decomposition axioms of the form shown above are mapped to parse rules of form

$$\varphi \rightarrow \psi_1 \ldots \psi_n$$

Note that the parse rule strips the names of steps from the decomposition. In addition, the following parse rule is necessary to produce the top of the parse tree, where *S* is the start symbol of the plan grammar.

S → END | END S

This rule introduces enough right-branching structure to account for all END plans appearing in the observations.

For example, say the cooking hierarchy in Figure 1 is simplified so as to contain only one END plan, that

Let $\Omega = \omega_1 \wedge \ldots \wedge \omega_n$ be an observation sentence where ω_i has form PLAN-TYPE$_i$(ACT$_i$).

Let *Chart*[0..n x 0..n] be an initially empty array of derived constituents.

Let *States*[0..n] be an initially empty array of intermediate parse states (dotted rules).

(1) Initialize *Chart[i-1,i]* with each PLAN-TYPE$_i$ in Ω.

(2) Add $S_{(0)} \rightarrow \bullet$ END and $S_{(0)} \rightarrow \bullet$ END S to *States[0]*.

(3) **For** $i \leftarrow 0$ **to** n **do**

(4) *Predict:* If $\alpha_{(j)} \rightarrow \cdots \bullet \beta \cdots$ is a dotted rule in *States[i]*, then add to *States[i]* a dotted rule $\beta_{(j)} \rightarrow \bullet \gamma \cdots$ for each rule deriving β in the grammar.

(5) *Scan:* If $\alpha_{(j)} \rightarrow \cdots \bullet \beta \gamma \cdots$ is in *States[i]* and β is a terminal, then if *Chart[i,i+1]* contains β, add a dotted rule of form $\alpha_{(j)} \rightarrow \cdots \beta \bullet \gamma \cdots$ to *States[i+1]*.

(6) *Complete:* If $\alpha_{(j)} \rightarrow \cdots \beta \bullet$ is a complete dotted rule in *States[i]*, then

(7) Add α to *Chart[j,i]*, using the AND-OR coding scheme.

(8) For each $\gamma_{(k)} \rightarrow \cdots \bullet \alpha \delta \cdots$ in *States[j]*, add to *States[i]* a dotted rule of form $\delta_{(i)} \rightarrow \bullet \chi \cdots$ for each rule deriving δ in the grammar.

Figure 3: Earley's algorithm.

for making fettuccine marinara. This new hierarchy (see Figure 2) would produce the following grammar.

S → END | END S
END → MAKE-PASTA-DISH
MAKE-PASTA-DISH → BOIL-WATER
MAKE-FETTUCCINE
MAKE-MARINARA

Parsing the Decomposition Grammar

As noted earlier, decomposition grammars can be applied to plan recognition with a chart-based version of Earley's algorithm. Briefly, the algorithm operates by maintaining a chart, a two-dimensional table that records the constituents spanning any two positions in the input stream.[2] If a constituent may be derived in more than one way between the same two positions, the multiple derivations are recorded as alternatives in an OR node (each derivation is itself an AND node).

Earley records partial derivations by instantiating "dotted" versions of rules. For example, A -> B C, when first applied, has a dot to the left of B ($A_{(i)} \rightarrow \bullet$ B C). After deriving B, a new instance of the rule is created, with the dot advanced past B ($A_{(i)} \rightarrow$ B $\bullet$ C). The subscripted index in this notation indicates the start position of the leftmost terminal derived by the rule.

The algorithm indexes dotted rules in a set of states: if a dotted rule seeks to derive a constituent starting in some position *k*, the rule is added to the k^{th} state set. When the dot is finally moved past the end of a rule, a derivation has been completed, and the derived

[2] For a sentence of length *n*, the chart runs from 0 to *n*, with each terminal spanning [i, i+1], for some i.

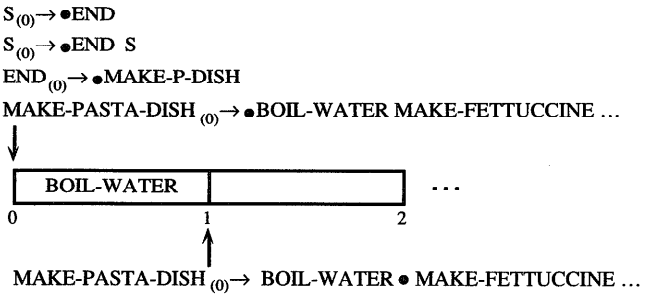

Figure 4a: States of the parser

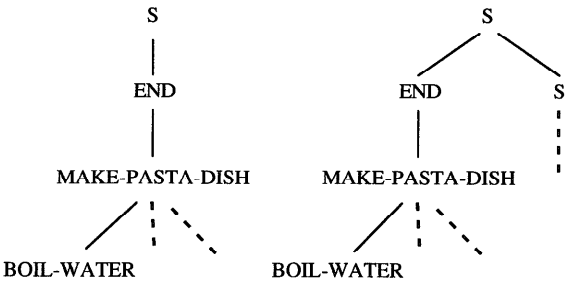

Figure 4b: Corresponding parse trees

constituent is added to cell $[i, j]$ of the chart, where i is the start index of the leftmost terminal derived by the rule, and j is the end index of the rightmost one.

The algorithm is sketched in Figure 3, with details in Earley (1970). With respect to applying the algorithm to plan recognition, three points need to be made.

First, observations are entered into the chart in a straightforward way. To be precise, say we have observed the sentence $\Omega = \omega_1 \wedge \ldots \wedge \omega_n$, where each ω_i has form PLAN-TYPE$_i$(ACT$_i$). Assuming observations are ordered and non-overlapping in time, Ω is entered into the chart by placing the terminal constituent corresponding to each PLAN-TYPE$_i$ in chart cell $[i\text{-}1, i]$.

The next point to note is that Earley's algorithm proceeds left-to-right through the input stream. Each step in the traversal computes all partial derivations that account for the input up to that point. This can be exploited to allow for incremental observations in the manner of Kautz' Incremental and Sticky Algorithms. That is, every time a new event is observed, it is added to the chart, and the main loop of the algorithm is simply restarted on the chart index for that event.

Most important, Earley is a predictive top-down parser. Thus, whenever a terminal symbol is scanned by the parser, the incomplete parse tree that ultimately derives the terminal from the start symbol is implicit in the state sets of the parser. This parse tree can be recovered from the dotted rules making up these sets.

For example, say the parser had been given the simplified cooking grammar corresponding to Figure 2, and say we had observed a BOIL-WATER action, B_1. The state of the parser and its chart would then be as shown in Figure 4a. The parser's state can be interpreted as identifying two distinct incomplete parse trees for the BOIL-WATER observation (see Figure 4b). The first derives a single END plan to account for the observation. The second tree postulates a second END plan following the first (a third would be postulated in the parser's prediction phase if the parser attempted to derive the second END plan, and so forth).

These parse trees can be interpreted as first-order sentences. For terminals, the sentential form is the observation associated with the terminal (a proposition of form PLAN-TYPE(ACT)). For non-terminals, we begin by creating for each non-terminal node an existentially quantified plan instance (ignoring the start node). The links from a node to its children in turn correspond to steps named in the decomposition axiom for the rule that derived the node. The sentential form of a parse tree t, with children $t_1 \ldots t_n$ is then the sentential forms of its children conjoined with

$$\exists\, \xi.\ \kappa(\xi) \wedge \sigma_1(\xi) = \xi_1 \wedge \ldots \wedge \sigma_n(\xi) = \xi_n$$

where $\xi, \xi_1, \ldots, \xi_n$ are the variables for nodes $t, t_1, \ldots, t_n$ respectively (or constants if the nodes are terminals), κ is the constituent associated with t, and the σ_i are the step names associated with the derivation of t. Multiple derivations of a constituent simply introduce into the sentential form the disjunction of their respective parses.

Under this mapping, the first parse in Figure 4b can be interpreted as:

$$\exists\ x,y\ \ \mathrm{END}(x) \wedge \mathrm{S0}(x) = y \wedge \mathrm{MAKE\text{-}PASTA\text{-}DISH}(y) \wedge \mathrm{S1}(y) = B_1 \wedge \mathrm{BOIL\text{-}WATER}(B_1)$$

This interpretation scheme for parse trees is akin to that used by Kautz to interpret his algorithms' E-graphs.

Correctness and complexity

To prove the correctness of the plan parser, it is necessary to show that for a given hierarchy H and observation ω, the parser computes the minimal covering models of ω with respect to H. This can be accomplished in two steps.

First, the algorithm can be shown to compute the covering models of H by relying on a result from Kautz (1987). Kautz shows that the covering models of plan hierarchies with no abstraction are exactly those that satisfy all instantiations of his *component/use* axiom schema. The (slightly modified) schema is given by:

$$\forall\ x\ \varphi(x) \supset \mathrm{END}(x) \vee \exists\ y_1\ \psi_1(y_1) \wedge (\sigma_1(y_1) = x) \vee \ldots \vee \exists\ y_n\ \psi_n(y_n) \wedge (\sigma_n(y_n) = x)$$

where φ is a plan type, and $\psi_1 \ldots \psi_n$ are those plan types that respectively restrict steps $\sigma_1 \ldots \sigma_n$ to be of type φ.

Extending this schema to parse trees is straightforward. In light of the sentential interpretation of parse trees, the schema can be rephrased in terms of $t, t_1,\ldots,t_n$, variables ranging over parse tree nodes:

$$\forall\ t\ \varphi(t) \supset \text{END}(t) \vee \exists\ t_1 \ldots t_n\ \ \psi_1(t_1) \wedge (t_1 \Rightarrow t) \wedge \ldots \wedge \psi_n(t_n) \wedge (t_n \Rightarrow t)$$

where φ is the constituent corresponding to a plan type, and $\psi_1,\ldots,\psi_n$ are all those constituents that derive φ in the grammar. The notation $\varphi(t)$ indicates that node t has constituent type φ, and the notation $t \Rightarrow t'$ indicates that node t derives node t' in the parse tree. For the purpose of this schema, the start node S, which is only used to introduce END nodes, is once again ignored.

It is easy to see that in this form, the component/use schema is fully instantiated by Earley's algorithm. The t_i are introduced into the parse tree by the prediction step, and the derivations are recorded by the completion step. By fully instantiating the schema, the algorithm thus computes the covering models of an observation with respect to a hierarchy.

To obtain the *minimal* covering models, note that each alternative parse tree attached to the start symbol S will postulate the existence of some number of END plans. These END plans can be enumerated simply by traversing the topmost right-branching structure of the tree introduced by the rule S → END S. The minimal models are those that apply this derivation a minimal number of times. This, along with the preceding discussion, informally shows the following proposition.

Proposition 1: Under the sentential interpretation of parse trees, Earley's algorithm computes the minimal covering models of an observation ω with respect to H, a decomposition hierarchy with ordered unshared steps.

Earley (1970) shows that the run time of his recognizer for a sentence of length n is bounded by a factor of $O(n^3)$. Barton, Berwick, and Ristad (1987) note that this bound can be refined to $O(G^{\circ 2} n^3)$, where G° is the total number of possible dotted rules afforded by the grammar G.

The addition of a chart, as is done in the algorithm of Figure 3, extends Earley's recognizer into a parser, but can introduce performance degradation. Tomita (1986), for example, describes some pathological combinations of grammars and input strings that can require of his chart-based parser $O(n^5)$ space utilization, and a corresponding degradation in parse time. However, Billot and Lang (1989) suggest that by using structure sharing to implement the chart's AND-OR graphs, the space requirements for storing the chart are bounded by $O(n^3)$, while the parse times also remain cubic.

The Uses of Abstraction

The preceding results are of some interest in establishing the tractability of plan recognition for one class of plan hierarchies, those expressible without abstraction. However, the resulting formal apparatus is so impoverished as to be useless in practice.

Beyond allowing for the identification of significant abstract constructs of a domain, abstraction is used in Kautz' framework to encode three different phenomena: the multiple expansions of a plan; the isolation of substeps common to all expansions of a plan (which are then shared through inheritance); and, indeterminate observations.

Adding Abstraction to the Grammar

It is easy to extend the mapping from plan hierarchies to grammars so as to allow for abstraction in the descriptions of plan types. Abstract observations, however, impose additional considerations, which I will return to later. Assume for now that observations are given in terms of base plan types (the leaves of the plan hierarchy), and again restrict plan steps to be fully ordered and unsharable. The mapping from a hierarchy H to a grammar G proceeds from the top of the hierarchy to its leaves, distinguishing two cases:

Case 1: Say φ is a plan type decomposing into steps $\sigma_1 \ldots \sigma_n$, and say φ has children $\chi_1 \ldots \chi_m$. Then, for each χ_i, copy each σ_j (and its type restriction ψ_j) to the decomposition of χ_i, unless χ_i happens to further restrict σ_j. Then for each χ_i, add a rule to the grammar of form

$\varphi \rightarrow \chi_i$

Case 2: Say φ is a childless plan type that decomposes into (possibly inherited) steps $\sigma_1 \ldots \sigma_n$, with step restrictions $\psi_1 \ldots \psi_n$. Then add to the grammar a decomposition rule of form

$\varphi \rightarrow \psi_1 \ldots \psi_n$

Again, the root of the hierarchy (ANY) is ignored, and again, the grammar is completed by adding the initial parse rule

S → END | END S

Note that this scheme eliminates from a hierarchy the explicit decompositions of abstract actions, enforcing them implicitly by inheritance instead. Thus, returning to the plan hierarchy in Figure 1, the subhierarchy rooted at MAKE-PASTA-DISH would be encoded as

MAKE-PASTA-DISH
  → MAKE-FETTUCCINE-MARINARA
MAKE-FETTUCCINE-MARINARA
  → BOIL-WATER
    MAKE-FETTUCCINE
    MAKE-MARINARA

The grammatical treatment of abstraction introduces additional complexity to the sentential interpretation of parse trees. Indeed, one must now distinguish nodes that were introduced by abstraction parse rules from those that were introduced by decomposition parse rules. As before, say t is a node with children $t_1 \ldots t_n$, and say t was introduced by a decomposition rule produced in

Case 2 of the grammatical mapping. Then the sentential form of t is that of its children conjoined with

$\exists \xi.\ \kappa(\xi) \wedge \sigma_1(\xi) = \xi_1 \wedge \ldots \wedge \sigma_n(\xi) = \xi_n$

For abstraction, say $\varphi \rightarrow \psi$ is a parse rule introduced to encode an abstraction axiom in Case 1 of the grammatical mapping. Let t be a node in a parse tree that derives a node t' by this rule. Then the sentential interpretation of the tree rooted at t is that of t' conjoined with the expression $\varphi(\xi)$, where ξ is the plan variable associated with the interpretation of t'.

Properties of Abstraction Grammars

The correctness and complexity properties of Earley's algorithm for decomposition grammars are easy to verify for abstraction grammars. For correctness, one must once again show that the algorithm computes the minimal covering models of some observation ω with respect to H, a hierarchy with abstraction. To begin with, Kautz (1987) showed that the covering models of a hierarchy with abstraction are precisely those that fully instantiate three axiom schemata, one of which (component/use) appeared above in a simplified form.

Disjunction: let φ be a plan type that directly abstracts types $\psi_1,\ldots, \psi_n$. Then:

$\forall x\ \varphi(x) \supset \psi_1(x) \vee \ldots \vee \psi_n(x)$

Exclusion: let φ_1 and φ_2 be incompatible plan types, i.e., ones for that there exists no φ_3 which is abstracted (directly or indirectly) by both φ_1 and φ_2. Then:

$\forall x\ \neg\varphi_1(x) \vee \neg\varphi_2(x)$

Component/use: let φ be a plan type, and let $\psi_1 \ldots \psi_n$ be all those plan types that restrict some step $\sigma_1 \ldots \sigma_n$ to be φ or a plan type compatible with φ. Then:

$$\forall x\ \varphi(x) \supset \mathrm{END}(x) \vee \exists y_1\ \psi_1(y_1) \wedge (\sigma_1(y_1) = x) \vee \ldots \vee \exists y_n\ \psi_n(y_n) \wedge (\sigma_n(y_n) = x)$$

Under the sentential interpretation of parse trees, the disjunction schema can be recast in terms of t and t', variables ranging over nodes in the parse tree:

$\forall t\ \varphi(t) \supset \exists t'\ (t \Rightarrow t') \wedge (\psi_1(t') \vee \ldots \vee \psi_n(t'))$

Assuming that ω mentions only base level observations, this schema is verified by noting that a parse node corresponding to an abstract plan type φ is only introduced into the chart (during the completion step of the algorithm) if one of $\psi_1,\ldots, \psi_n$ was previously introduced into the chart.

A similar argument can be used to show that, assuming base level observations, the parser fully instantiates the exclusion schema. The argument can also be used to extend the proof of Proposition 1 in order to show that the parser fully instantiates the extended version of the component/use schema. This informally demonstrates that the algorithm computes the covering models of ω with respect to H. The minimal covering models can be obtained as before from those parses introducing fewest END nodes under S, thus showing:

Proposition 2: Under the sentential interpretation of parse trees, Earley's algorithm computes the minimal covering models of a base-level observation ω with respect to H, a hierarchy with ordered unshared steps.

As before, the time complexity of parsing an observation "sentence" is $O(G^{\circ 2} n^3)$. The G° term is related to the original hierarchy description in the following way. There is exactly one dotted rule for each abstraction axiom, and the latter's number is bounded by P, the size of the set of plan types in the hierarchy. The original decomposition axioms are also bounded in number by P, and it is easy to verify that after step inheritance, the number of corresponding dotted decomposition rules is bounded by Pd, where d is the number of steps mentioned in the longest decomposition axiom. The overall size of G° is thus $O(Pd)$.

Abstract Observations

The preceding discussion crucially relies on observed actions' not being abstract. This is a severe limitation, since abstract plan types simplify the expression of indeterminate observations. For example, in Kautz' cooking world, one might like to encode uncertainty on whether an agent is making fettuccine or spaghetti with an abstract MAKE-NOODLES observation. In grammatical terms, this amounts to allowing non-terminal categories to appear directly in the input stream.

For the plan parser to interpret these observations correctly is tricky. The problem is that to ensure that the minimal covering models are computed, the parser must expand the abstract observation into its possible specializations, and hypothesize that each may have occured. It would be appealing if this expansion could be effectuated by compiling additional parse rules out of the plan hierarchy. Unfortunately, though various naive strategies for doing so are conceivable, they all seem to have unacceptable problems.

For instance, one could allow for abstract observations to be specialized *down* the abstraction hierarchy with "reverse" abstraction rules (e.g., MAKE-PASTA-DISH → MAKE-MEAL). However, this leads directly to violations of the disjunction schema. An alternative (and equally naive) approach would produce additional decomposition rules introducing the possible abstractions of a base action, e.g.:

MAKE-FETTUCCINE-MARINARA→ MAKE-NOODLES
MAKE-SAUCE

However, the number of such rules is bounded by $O(d^P)$, leading to an exponential increase in the size of the grammar and an exponential increase in parse times.

A more practical alternative is to leave the grammar unchanged, and treat an abstract observation $\varphi(e)$ disjunctively as $\vee_i \psi_i(e)$, where the ψ_i are those plan types that maximally specialize φ (i.e., that specialize φ and have no specializations in turn). This strategy can be seen as explicitly enforcing the disjunction axiom schema on abstract observations.

At the grammatical level, this strategy has a natural analogue in *lexical ambiguity*, the ambiguity encountered when a terminal (e.g. the English word *can*) is derivable by more than one pre-terminal (e.g. *V*, *N*, or *AUX*). In linguistic parsers the terminals in a string are usually replaced with the corresponding pre-terminals, so lexical ambiguity can be simply dealt with by adding each ambiguous pre-terminal directly into the same cell of the chart. Similarly, the plan parsing algorithm in Figure 3 can be amended to enter any abstract observation $\varphi(e)$ into its chart cell as the set of ψ_i that it abstracts. A MAKE-NOODLES observation, for example, would be entered in the chart as a set of two terminals: {MAKE-SPAGHETTI MAKE-FETTUCCINE}.

This approach to abstract observations preserves the correctness of the plan parser. As noted above, it directly enforces the disjunction schema. That the exclusion schema is enforced can be seen by noting that no two ambiguous terminal entries appear in the same parse. Consequently, under the sentential interpretation of parse trees, no two ambiguous but incompatible types can hold true of the same plan variable or plan constant. Finally, for each distinct parse, it is easy to verify that the algorithm will fully instantiate the component/use schema.

The disjunctive treatment of abstraction also maintains the polynomial tractability of the plan parser. This can be seen by noting that the $O(G^{\circ 2} n^3)$ time bound on parsing an observation string of length n is obtained from a $O(G^{\circ 2} n^2)$ bound on each step of the main loop of the parser (which is iterated n times). Informally, one can think of an ambiguous observation φ in position $[i\text{-}1, i]$ as temporarily "augmenting" the grammar for iteration step i. The augmentation consists of introducing a new category OBSERVED-φ and new rules of form OBSERVED-$\varphi \rightarrow \psi_i$, for each ψ_i maximally specializing φ. The abstract observation is then encoded as a token of OBSERVED-φ. This has a net effect of temporarily adding no more than P rules to the grammar at each step of the parser's main iteration, and so G° remains bounded at each step by $O(Pd)$. This leaves the overall time complexity of parsing unaffected.

The following proposition summarizes the discussion of the past few pages.

Proposition 3: There is a $O(n^3)$-time plan recognition algorithm for hierarchies with ordered, unshared steps, and for disjunctive or abstract observations.

Further Extensions

This result is of significant value, as it delineates a subset of Kautz' plan formalism for which plan recognition is tractable. The parsing approach underlying this result can in fact be extended to cover further aspects of Kautz' formalism, but unfortunately not without also sacrificing recognition tractability.

Partial Step Order

A number of recent linguistic formalisms refine the traditional phrase structure rules into two sets of rules: (1) indirect dominance (ID) rules, which specify which subconstituents may be derived by a constituent, and (2) linear precedence (LP) rules, which determine the left-to-right order of these subconstituents. This ID/LP strategy can be applied to plan hierarchies to allow for a compact encoding of partial step ordering. For example, the following two sketchy rules specify that the BOIL-WATER step of the MAKE-PASTA-DISH plan must be ordered before the MAKE-NOODLES step, but leaves all other step relations unordered.

PASTA $\xrightarrow{\text{ID}}$ BOIL NOODLES SAUCE *(ID rule)*
BOIL < NOODLES *(LP rule)*

In effect, an ID rule of length n stands for an equivalent $n!$ ordered context-free rules, some of which are then eliminated if they fail to satisfy the LP rules. In principle, one could thus parse an ID/LP grammar with Earley's algorithm by first compiling it into the corresponding context-free rules. However, as the number of such rules is combinatorially explosive, the size of the resulting grammar would be correspondingly large, and parse times correspondingly lengthy. To alleviate this problem, Shieber (1983) produced a simple extension to Earley's algorithm that allows for direct parsing of ID/LP grammars, thus circumventing the combinatorial explosion produced by compilation.

Unfortunately, Shieber's parser does not escape intractability. Barton *et al.* (1987) show that ID/LP parsing is NP-complete under certain conditions. The argument is complex, but for the purposes of this paper it suffices to note that a sufficient condition for NP-completeness is the kind of lexical ambiguity used above to encode abstract observations. In fact, this NP-completeness result can easily be extended to show the following proposition (offered here without proof).

Proposition 4: Recognizing plans with abstraction and partial step order is NP-complete, regardless of recognition tactic.

This pessimistic result must be taken in perspective. Shieber's algorithm performs well in practice, and truly extreme derivational ambiguity is required to lead it to exponential performance. In fact, Barton *et al.* suggest that tractability may actually be regained by ensuring that the unordered steps of an ID decomposition are derivationally distinct. This is the case, for example, with the ID rule decomposing MAKE-PASTA-DISH, each of whose steps derives a set of constituents distinct from those derived by the others. However, a general plan distinguishability criterion has yet to be formulated.

Action Parameters

Kautz allows plans to have parameters, such as an agent. As with other aspects of plan recognition, action

parameters have a grammatical analogue, in this case with unification grammars (another extension of the context-free class). Without going into details, it is straightforward to show that plan parameters and constraints on these parameters can be encoded in the unification formalism. However, parsing unification grammars is again NP-complete in the presence of derivational ambiguity (Barton *et al.* (1987)).

Plan Parsing in Perspective

There are additional aspects of Kautz' approach that may not be convincingly treated with a parsing strategy. Shared and interleaved steps are a particularly salient example of this. It is admittedly possible to formulate some kind of *type 0* or perhaps *context-sensitive* phrase structure rules to encode the sharing or interleaving of steps. However, it is not at all clear how to do so without endowing the plan formalism with enough machinery to make plan recognition intractable or even undecidable (type 0 grammars being Turing-equivalent).

Nevertheless, the main thrust of this work is not to show that all of Kautz' approach can be reformulated as parsing, as much as it is to find those aspects of his approach that become tractable when so reformulated. Beyond the immediate gains of tractability, the parsing approach does provide an operational advantage over Kautz' algorithms. Namely, it focuses recognition by predicting the existence of only those END plans sanctioned by all the observations taken together. Kautz' algorithms perform the prediction on each individual observation, independent of the others, and then combine the resulting predictions. This is computationally much more onerous, but may turn out to be unavoidable if one wants to allow for sharing and interleaving of steps.

Finally, I should note that there are many similarities between the parsing strategies described here and the plan recognition strategies in MITRE's King Kong interface (Burger & Sider, 1990). As part of our current research, my colleagues and I are investigating further extensions to King Kong that rely on parsing strategies.

Acknowledgements

This work has benefitted from discussions over the years with James Allen and Henry Kautz. Special thanks to Ellen Hays for her untiring editorial attention.

References

Allen, J. (1983). Recognizing intentions from natural language utterances. In Brady, M. & Berwick, R. (eds) *Computational Models of Discourse*. Cambridge, MA: The MIT Press.

Allen, J. & Perrault, R. (1980). Analyzing intention in dialogue. *Artificial Intelligence* **23**(2), 832-843.

Barton, G. E., Berwick, R. & Ristad, E. (1987). *Computational Complexity and Natural Language*. Cambridge, MA: The MIT Press.

Billot, S. & Lang, B. (1989). The structure of shared forests in ambiguous parsing. In *Proceedings of ACL 89*, 143-151.

Burger, J. & Sider, J. (1990). *Discourse Understanding in Expert System Interfaces*. In preparation.

Carberry, S. (1983). Tracking goals in an information seeking environment. In *Proceedings of AAAI 83*, 59-63.

Earley, J. (1970). An efficient context-free parsing mechanism. *Communications of the ACM* **13**(2), 94-102. Reprinted in Grosz et al. (1986).

Goodman, B. & Litman, D. (1990). Plan recognition for intelligent interfaces. In *Proceedings of the IEEE Conference on Artificial Intelligence Applications 1990*.

Grosz, B., Sparck Jones, K., Webber, B. L. (1986). *Readings in Natural Language Processing*. San Mateo, CA: Morgan Kaufmann.

Kay, M. (1980). *Algorithm Schemata and Data Structures in Syntactic Processing*. Tech Report CSL-80-12, Xerox PARC. Reprinted in Grosz et al. (1986).

Kautz, H. (1987). *A Formal Theory of Plan Recognition*. PhD dissertation, Dept. of Computer Science, University of Rochester. Available as Tech. Report 215.

Kautz, H. & Allen A. (1986). Generalized plan recognition. In *Proceedings of AAAI 86*, 32-37.

Konolige, K. & Pollack, M. (1989). Ascribing plans to agents — preliminary report. In *Proceedings of IJCAI 89*, 924-930.

Litman, D. (1986). Linguistic coherence: A plan-based alternative. In *Proceedings of ACL 86*, 215-223.

Pollack, M. (1986). A model of plan processing which distinguishes between the beliefs of actors and observers. In *Proceedings of ACL 86*, 207-214.

Sacerdoti, E. (1977). *A Structure for Plans and Behavior*. New York: North-Holland.

Schmidt, C. , Sridharan, N., & Goodson J. (1978). The plan recognition problem: An intersection of artificial intelligence and psychology. *Artificial Intelligence*, **11**(1), 45-83.

Shieber, S. (1983). Direct parsing of ID/LP grammars. *Linguistics and Philosophy* **7**(2), 135-154.

Sidner, C. (1985). Plan parsing for intended response recognition in discourse. *Computational Intelligence*, **1**(1), 1-10.

Thompson, H. (1983). MCHART: A flexible, modular chart parsing system. In *Proceedings of AAAI 83*, 408-410.

Tomita, M. (1986). *Efficient Parsing for Natural Language*. Boston: Kluwer Academic Publishers.

Wilkins, D. (1984). Domain-independent planning: representation and generation. *Artificial Intelligence* **22**, 269-301.