@Make(Article)
@Device(ln01)
@Style(LineWidth 15.9cm, Spacing 7.5mm, References=STDalphabetic)
@Use[Bibliography="general.bib"]
@begin(format)

@b(A Syntactic Approach to Planning)

Thomas C. Henderson and Eric Muehle
Department of Computer Science
The University of Utah

@b[Abstract]

@end(format)
Formal language theory concerns the study of two major issues:
@begin(enumerate)
@b[Generative Grammars]: a language is specified in terms of an alphabet, vocabulary symbols, rewrite rules and a start symbol; i.e., a method for generating all strings in the language, and

@b[Recognizers]: a language is specified by giving an alphabet, states, memory and state transitions; i.e., a method for deciding for any given string whether or not it is in the language.
@end(enumerate)
Most of the syntactic pattern recognition work has exploited the recognition aspect of formal language theory. What we propose here is the use of generative grammars as a mechanism to help encode plans.
@Blankspace(5mm)
@Section(Introduction)
@Blankspace(5mm)
Formal language theory permits both the analysis and synthesis of strings. Both of these aspects have been explored in the domain of shape analysis and pattern
recognition@cite[Bunke82,Fu73,Fu74,Gonzalez78,Lin84,Rosenfeld72].
Most of the work on synthesis has
concerned the generation of regular patterns and textures.
However, there has been little published on the use of grammars as a mechanism for solving the planning problem.

To solve a problem requires that the appropriate sequence of operations be performed in the correct order. Finding such a sequence is, in general, a difficult problem and many approaches have been proposed@cite[Nilsson71, Winston84].
Most of these methods lack the ability to focus well on a particular part of the problem. We propose that a generative grammar can provide such a mechanism.

@Blankspace(5mm)
@Section(A Syntactic Approach to Planning)
@Blankspace(5mm)
We restrict our attention to planning in the context of a robot workcell. The task to be performed is light assembly. Thus, we are essentially concerned with plans for the assembly of small parts in a well-known environment.

First, it is necessary to have some way to model 3-D shapes and their structure. We have previously worked on the problem of 3-D shape

representation and analysis@cite[Davis81,Henderson81,Henderson85d],
and we will use Stratified Shape
Grammars as our representation scheme. Usually, a shape grammar is defined
to solve the shape recognition problem. Here, however, we will define a
shape and then take advantage of the shape grammar to help plan the
sequence of operations.

Given a shape grammar, it can be used to help solve several aspects of the
planning problem:
@begin(enumerate)
@u[Rewrite rules impose an order on the operations.] That is, given a
parse tree of the 3-D structure to be built, it is straightforward to
analyze the sequence in which the operations must be performed. In
addition, it is also possible that opportunities for parallelism can be
discovered.

@u[Constraints on the positioning of parts can be recovered.] Many such
constraints are explicit in the rewrite rules (see Section 3 below).
However, it is also possible to discover implicit constraints (e.g., by
means of global analysis or constraint propagation).

@u[Focus of attention is achieved.] Since only the appropriate components
appear together on the righthandside of a rewrite rule, it is possible to
determine what parts of the shape are related. Moreover, one can use both
ancestors, neighbors and decendants relations to focus attention.
@end(enumerate)
In addition, the use of generative grammars permits a unified approach to
the shape analysis problem. That is, the same underlying paradigm supports
both the synthesis of the 3-D structures and the later analysis of those
same structures. Thus, any change in the shape (i.e., a change in the
grammar) is automatically reflected in the synthesis and analysis.

Finally, given the CAD-based context, it is possible that grammatical
specifications of the 3-D structure can be synthesized from the CAD design
information, thus getting around the difficult problem of grammar writing.
At the very least, a graphical interface for grammar design would be
reasonably easy to produce.

@Blankspace(5mm)
@Section(An Example)
@Blankspace(5mm)
Consider a very simple example: the construction of Lincoln log houses.
The 3-D shape primitives (terminal symbols) are the following:
@Blankspace(7.5inches)
A stratified shape grammar for building a simple 4-walled house is:
@begin(verbatim)

```
Grammar for Lincoln Log House

SIGMA = {base, log1, log4, log7, arch, roof}
V = {house, top, bottom, 4wall, top-wall, bottom-wall}

base{E1,mE,E2}[Al,As]
log1{E,U}[Al,As]
log4{E1,E2,U1,U2}[Al,As]
log7{E1,Em,E2,U1,Um,U2}[Al,As]
roof{E1,S1,E2,S2}[Al,As]
arch{E1,E2,H1,H2}[Al,As,S1,S2]

house{}[h,w,l] :=  top{T1,T2,T3,T4}[Tl,Ts,Ht,Wt,Lt]
                 + bottom{B1,B2,B3,B4,B5,B6}[Bl,Bs,Hb,Wb,Lb]

C : Ti near Bi (i < 5)
```

```
S : Tl || Bl and Ts || Bs

Ga : 0

Gs : h = Ht + Hb; W = (Wt + Wb)/ 2; L = (Lt + Lb) / 2;

top{E1,E2,E3,E4}[Al,As,h,w,l] :=

                    arch{E1',E2',H1',H2'}[As',Al',S1',S2']
                  + arch{E1'',E2'',H1'',H2''}[As'',Al'',S1'',S2'']
                  + roof{E1''',S1''',E2''',S2'''}[Al''',As''']
                  + roof{E1'''',S1'''',E2'''',S2''''}[Al'''',As'''']

C : S1''' next-to S2'''' and
    E1''' touches E1'''' and
    E2''' touches E2'''' and
    S2''' touches H1'    and
    S2''' touches H1''   and
    S1'''' touches H2'   and
    S1'''' touches H2''

S : As''' || S1'  and  As''' || S1''  and  As'''' || As''''  and
    As'''' || S2''  and  Al' || Al''
    and  distance(H1',H1'') > length(log1)

Ga : E1 = E1'; E2 = E2'; E3 = E1''; E4 = E2''

Gs : Al = (Al''' + Al'''') / 2; As = (Al' + Al'') / 2; h = height(arch);
     l = length(Al);  w = length(As);

bottom{B1,B2,B3,B4,B5,B6}[Al,As,h,w,l] :=

        4wall{J1,J2,J3,J4,J5,J6,U1,U2,U3,U4,U5,U6}[Wl,Ws,h',w',l']
      + base{E1,M,E2}[Al',As']
      + base{E1',M',E2'}[Al'',As'']


C : U1 near E1  and  U2 near E2  and  U3 near E1'  and  U4 near E2'
    and  U5 near M  and  U6 near M'

S : Al'' || Al'  and  As'' || As'  and  distance(E1,E1') > length(log1)

Ga : Bi = Ji

Gs : Al = Wl; As = Ws; h = h' + height(base); w = length(log7);
      l = length(log4);

4wall{E1,E2,E3,E4,M1,M2,U1,U2,U3,U4,U5,U6}[Al,As,h,w,l] :=

top-wall{E1',E2',E3',E4',M1',M2',U1',U2',U3',U4',U5',U6'}[Al',As',h',w',l']
+ bottom-wall{E1'',E2'',E3'',E4'',M1'',M2'',U1'',U2'',U3'',U4'',U5'',U6''}
              [Al'',As'',h'',w'',l'']

C : U1' near E1''  and  U2' near E2''  and  U3' near E3''
    and  U4' near E4'' and  U5' near M1''  and  U6' near M2''

S : Al' || Al''  and  As' || As''

Ga : Ei = Ei'; Mi = Mi';

Gs : Al = Al'; As = As'; h = h' + h'';

4wall{E1,E2,E3,E4,M1,M2,U1,U2,U3,U4,U5,U6}[Al,As,h,w,l] :=
```

```
 top-wall{E1',E2',E3',E4',M1',M2',U1',U2',U3',U4',U5',U6'}
          [Al',As',h',w',l']
+bottom-wall{E1'',E2'',E3'',E4'',M1'',M2'',U1'',U2'',U3'',U4'',U5'',U6''}
              [Al'',As'',h'',w'',l'']
+4wall{E1''',E2''',E3''',E4''',M1''',M2'',U1''',U2''',U3''',
        U4''',U5''',U6'''}
        [Al''',As''',h''',w''',l''']


C : U1' near E1''  and  U2' near E2''  and  U3' near E3''
    and  U4' near E4'' and  U5' near M1''  and  U6' near M2''

S : Al' || Al'' || Al''' and  As' || As'' || As'''

Ga : Ei = Ei'''; Mi = Mi'''; Ui = Ui''';

Gs : h = h' + h'';


top-wall{E1,E2,E3,E4,M1,M2,U1,U2,U3,U4,U5,U6}[Al,As,h,w,l] :=

          log7[E1',Em',E2',U1',Um',U2'}[Al',As']
        + log7[E1'',Em'',E2'',U1'',Um'',U2''}[Al'',As'']

C : 0

S : Al' || Al''  and  As' || As''

Ga : E1 = E1'; E2 = E2'; E3 = E1''; E4 = E2''; M1 = Em'; M2 = Em'';
     U1 = U1'; U2 = U2'; U3 = U1''; U4 = U2''; U5 = Um'; U6 = Um'';

Gs : h = height(log7);


bottom-wall{E1,E2,E3,E4,M1,M2,U1,U2,U3,U4,U5,U6}[Al,As,h,w,l] :=

         log4{E1',E2',U1',U2'}[Al',As']
       + log4{E1'',E2'',U1'',U2''}[Al'',As'']
       + log1{E''',U'''}[Al''',As''']
       + log1{E'''',U''''}[Al'''',As'''']

C : 0

S : Ax' || Ax'' || Ax''' || Ax''''  x = (l,s)

Ga : E1 = E1'; E2 = E1''; E3 = E2'; E4 = E2''; M1 = E'''; M2 = E'''';
     U1 = U1'; U2 = U1''; U3 = U2'; U4 = U2''; U5 = U'''; U6 = U'''';

Gs : h = height(log4);
```

@end(verbatim)
As can be seen, many of the constraints are explicit in the rewrite rules (e.g., Near, Parallel, etc.).

We are currently exploring the use of FROBS (@u[fr]ame @u[ob]ject@u[s]@cite[Muehle86]) to express the shape grammar in an expert system format. For example, frobs can be defined for the vocabulary symbols:
@begin(verbatim)

```
*** Frobs ***

(def-class struct nil :slots (axis h w l))
```

```
(def-class house {struct} :slots (top bottom))

(define-class top {struct} :slots (joints))

(define-class bottom {struct} :slots (joints))

;;; assume that there are some instances of these 3 classes

@end(verbatim)
```

and a rule can be expressed as:

```
@begin(verbatim)
(define-forward-rule identify-house {rule}
  (premise (and (?house top {?})
                (?house bottom {?})
                (?top joints ?j1)
                (?bottom joints ?j2)
                ^(near-p ?j1 ?j2)
                (?top axis ?ta)
                (?bottom axis ?ba)
                ^(parallel-p ?ta ?ba)
                (?top h ?th)
                (?top w ?tw)
                (?top l ?tl)
                (?bottom h ?bh)
                (?bottom w ?bw)
                (?bottom l ?bl)))
  (conclusion (and (?house top ?top)
                   (?house bottom ?bottom)
                   (?house axis ?ta)
                   (?house joints ?j1)
                   (?house h ^(+ th bh))
                   (?house w ^(/ (+ tw bw) 2))
                   (?house l ^(/ (+ tl bl) 2)))))

@end(verbatim)
```

All structures have an axis frame, a height slot, a width slot, and a length slot. The axis frame contains the long and short axis of orientation. The height, width, and length slots contain those values for that particular structure.
A house frame is a subclass of the structure frame, but also has slots for the bottom, and top frames of the house.
The top and bottom frames are structures with a set of joints for reasoning about the connectivity between structures.

The identify-house rule says:
@begin(format)

> If there exists a house with a top and bottom undefined, and there exists a top and bottom whose joints are near each other, and the top and bottom axises are parallel to each other then there exists a house that has a top and bottom, with a set of joints, a set of axises, and a height width, and length.

@end(format)

Let us now see how this is useful in planning the construction of a Lincoln Log house. We describe a problem-reduction technique that successively reduces the state-space search by guiding the reduction through the use of rewrite rules.

Given a set of start states, S, a set of operators, F, that map states onto states, and a set of goal states, G, the rewrite rules of the grammar can be used to identify subgoals which must first be solved before the final goal can be achieved.

For example, given the goal of constructing a Lincoln Log house, the grammar gives
us two immediate subgoals: build the top of the house and the bottom of the house. Given the geometric semantics of the relations on the attachment parts of the vocabulary symbols, it can be inferred that @i(top) is "OnTopOf" @i(bottom), and therefore, that top should be built after bottom. We are currently exploring an implementation of these ideas to generate plans for a PUMA 560 to assemble Lincoln Log houses.

@Blankspace(5mm)
@Section(Conclusion)
@Blankspace(5mm)
We propose that the syntactic approach may be used as the basis for planning. A grammar permits the natural recovery of sequence information, recovery of constraints and the focus of attention. We are presently exploring light assembly tasks in a robotics workcell.

@Blankspace(5mm)
@Section(Acknowledgments)
@Blankspace(5mm)
This work was supported in part by NSF Grants MCS-8221750, DCR-8506393, and DMC-8502115.