Vision Practical 2

Out: week of March 2
Due: week of April 27

"Use as a tutorial"

Many of the early attempts to understand 3D scenes used labeling rules for blocks world scenes. As you will recall from the lectures, these rules represent relationships that hold in 3D, but which need not hold in an arbitrary line drawing. Hence, applying the rules to a line drawing gives us some 3D interpretations.

This practical applies a set of line labeling rules (derived from Huffman & Clowes) to two blocks-world scenes. A labeling is consistent if labels can be assigned to all lines such that the pattern of connection at the vertices is consistent with allowable blocks-world scenes. Before generating the final labeling of the scene, Waltz-filtering is applied to reduce the combinatorial matching needed to find a solution.

In file '/u/ai/s2/ai2teach/vision/pract2/pract2.code' you will find part of a program to do this labeling. In the same directory, you will also find the file 'labels' which will contain the labeling rules, and two test scenes in files 'scene1' and 'scene2'.

This program is incomplete - currently it:

- checks to see if the scene is properly defined,
- finds all possible alternative labels for each vertex and
- applies Waltz filtering to drastically reduce the set of labels.

Two extensions need to be added to complete it:

(a) the predicate 'testinconsist(Labels)' needs to be defined. This predicate checks to see if there is still a feasible solution after the Waltz filtering - which can be detected if some vertex no longer has any possible labelings.

The input 'Labels' is a list of the form:

[.... [vertex, [possible labels]] ....]

(b) The major work is to complete the predicate 'generate_label' which finds all complete labelings for the scene, by searching among alternatives left after the Waltz filtering.

A labeling is consistent if the arcs connecting two vertices have the same labeling at each end according to the vertex label.

Many of the predicates in the program use a list of potential labels for all vertices. This is a list of the form:

[.... [vertex, [possible labels]] ....]

where each vertex appears in the list once.

The format of the 'labels' file is:

label(<vertex_type>,<label_id>,<label_1>,<label_2>,<label_3>).

where:

<vertex_type> ::= ell | fork | arrow | tee

according to the type of vertex

<label_id> is an arbitrary identifier for this label type

and

<label_N> ::= in | out | plus | minus | null

is the line label for each of the arcs leaving the vertex, where the vertices are ordered as:

ell - left to right with the gap at bottom. The third label is always 'null'.
fork - clockwise order from an arbitrary starting point
arrow - from left to right with the point facing upward
tee - left_bar, shaft, right_bar

The scene is described by a set of assertions of the form:

vertex(<vertex_name>,<vertex_type>,<line_name1>,
<line_name2>,<line_name3>).

which lists the vertices and gives their types and a name for the connecting lines. The lines are labeled in the same order as for the 'label' predicate. The other assertions in the scene description are of thee form:

known(vertex_name>,[<label_id>]).

which asserts the designated vertex has a reduced set of possible labelings (here, this usually means only a single labeling is allowed).

The Waltz filtering removes unusable vertex labelings from the potential label set associated with a vertex. A vertex labeling is removed if it has an edge label that cannot match up with any conceivable labeling at the connecting vertex. The 'waltzfilter' predicate checks all possible labelings of each vertex one at a time. If any labeling is deleted, the whole process is re-applied because the deletion may lead to deletions of other labels.

To run the practical, you must consult the program, the labels, the desired test scene and your extensions to the program. Scene labeling is started by invoking the predicate 'labelscene'.

For the practical:

(1) trace through the program with the 'testinconsistent' predicate as always true to see how it works up to the 'generate_label' predicate. This should help you see how the major data structures and the line labelings are used.

(2) implement the 'testinconsistent' predicate described above.

(3) implement the 'generate_label' predicate described above.

(4) show the output complete labeling for both test scenes on the enclosed test scene diagrams.

(5) There should be four labelings for the second scene. Explain why.

(6) Why does the labeling show that the upper left block lies in front of the middle left block when the most reasonable interpretation has them touching (i.e. lines 'line7' and 'line8' are obscuring instead of concave). The same point also applies to the upper and lower right blocks.

(7) [optional] Separate the regions into separate objects isolated by surrounding obscuring and concave boundaries.

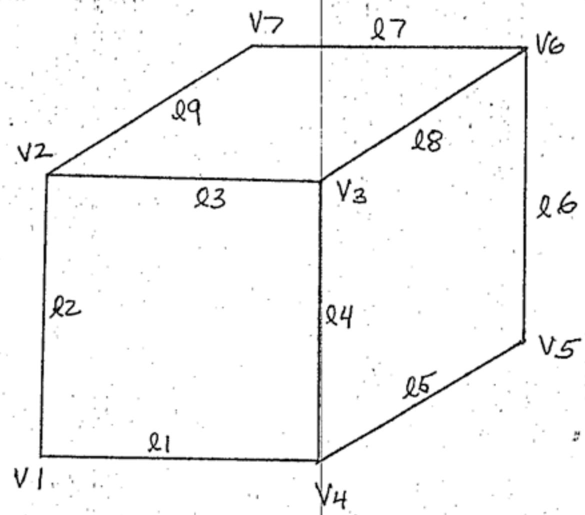
```
/* top level control for labeling a scene:
        gets set of all possible labels at each vertex
        does waltz filtering to reduce the set
        finds each complete labeling of whole scene
                such that all line labels are consistant
*/
labelscene :-

        /* checks scene for being consistent */
        checkscene,

        /* get the possible labelings for this diagram */
        bagof([Vertex,Labels],
                T^L1^L2^L3^(
                        vertex(Vertex,T,L1,L2,L3),
                        bagof(LabelType,
                                LB1^LB2^LB3^label(T,LabelType,LB1,LB2,LB3),
                                Labels)
                        ),
                PossibleLabels),

        /* replace possible by any known labels */
        replace_known(PossibleLabels,KnownLabels),

        /* print initial labels */
        write('Initial Labels'),nl,
        writelabels(KnownLabels),

        /* do waltz filtering to reduce possible label set */
        waltzfilter(KnownLabels,NewPossLabels),

        /* print initial labels */
        write('Labels after Waltz Filtering'),nl,
        writelabels(NewPossLabels),

        /* test for inconsistency here (ie a vertex has no possible
                labels left) */
        testinconsist(NewPossLabels),

        /* generate all possible labelings and separate bodies */
        (
                (generate_label(NewPossLabels,Labeling),
                 writesoln(Labeling),
                 fail   /* force backtracking to generate new labeling */
                )
          ;
                true
        ).

/* PREDICATE: waltzfilter(+In_Labels,-Out_Labels):
        In_Labels - the input label set to the filtering
        Out_Labels - the output label set from the filtering

        does waltz filtering - removes a label from a vertex if
        any connecting vertex doesn't have a corresponding label.
        Keeps re-applying the process until no more changes are made.
*/
waltzfilter(AllLabels,NewLabels) :-
        wf(AllLabels,AllLabels,NewLabels,[],nochange).
wf([[Vertex,VLabels]|Rest],AllLabels,NewLabels,CurrentLabels,InState) :-
        !,filter(Vertex,VLabels,NewVLabels,AllLabels,InState,OutState),
        wf(Rest,AllLabels,NewLabels,[[Vertex,NewVLabels]|CurrentLabels],
                OutState),!.
wf([],_,Labels,Labels,nochange).
wf([],_,NewLabels,CurrentLabels,change) :-
        /* go through whole process again on reduced label set */
```

```
/* PREDICATE: filter(+Vertex,+Vertex_Labels,-New_Labels,+All_Labels,
       +Current_State,-New_State)

        Vertex - id of current vertex
        Vertex_Labels - list of current potential labels
        New_Labels - list of remaining potential labels
        AllLabels - list of current potential labels for all vertices
        Current_State - records 'change'/'nochange' according to whether
                        any changes have been made so far
        New_State - update of  'change'/'nochange' state

        filters out any currently impossible labels for this vertex.
*/
filter(Vertex,[Label|Tail],NewLabels,AllLabels,InState,OutState) :-
        filter(Vertex,Tail,NewTail,AllLabels,InState,TState),

        /* check for removing this label */
        (checkposslabel(Vertex,Label,AllLabels)
         -> (NewLabels = [Label|NewTail], OutState = TState)
         ;  (NewLabels = NewTail, OutState = change)
        ).
filter(_,[],[],_,State,State).

/* PREDICATE: checkpossiblelabel(+Vertex,+Label,+AllLabels)

        Vertex - id of current vertex
        Label - id of current test label
        AllLabels - all current potential labels for all vertices

        see if this label at this vertex is compatible with labels at
        connecting vertices
*/
checkposslabel(Vertex,Label,AllLabels) :-
        /* check each connecting vertex */
        vertex(Vertex,_,Line1,Line2,Line3),
        label(_,Label,Line1Type,Line2Type,Line3Type),
        checkpossvertex(Vertex,Line1,Line1Type,AllLabels),
        checkpossvertex(Vertex,Line2,Line2Type,AllLabels),
        (Line3 = null
         -> true
         ;  checkpossvertex(Vertex,Line3,Line3Type,AllLabels)
        ).

/* PREDICATE: checkpossvertex(+Vertex,+Line,+LineType,+AllLabels)

        Vertex - id of current vertex
        Line - id of line being tested
        LineType - the line label being tested
        All_Labels - all current potential labels for all vertices

        see if connecting vertex has a compatible line_type on the
        given line
*/
checkpossvertex(Vertex,Line,LineType,AllLabels) :-
        vertex(CVertex,_,Line,_,_),Vertex == CVertex,!,
        findassoc(CVertex,AllLabels,PosLabels),
        labelmatch1(LineType,PosLabels).
checkpossvertex(Vertex,Line,LineType,AllLabels) :-
        vertex(CVertex,_,_,Line,_),Vertex == CVertex,!,
        findassoc(CVertex,AllLabels,PosLabels),
        labelmatch2(LineType,PosLabels).
checkpossvertex(Vertex,Line,LineType,AllLabels) :-
        vertex(CVertex,_,_,_,Line),Vertex == CVertex,!,
        findassoc(CVertex,AllLabels,PosLabels),
```

```
/* PREDICATE: labelmatchN(+LineLabel,+ConnectingLabels)

        LineLabel - the line label being tested
        ConnectingLabels - all possible labelings for the connecting
                vertex

        checks if label type matches in Nth label position for at least
        one of the possible labelings.
*/
labelmatch1(LineLabel,[Labeling|_]) :-
        label(_,Labeling,TestLabel,_,_),
        compatible(LineLabel,TestLabel),!.
labelmatch1(LineLabel,[_|Rest]) :-
        labelmatch1(LineLabel,Rest).
labelmatch2(LineLabel,[Labeling|_]) :-
        label(_,Labeling,_,TestLabel,_),
        compatible(LineLabel,TestLabel),!.
labelmatch2(LineLabel,[_|Rest]) :-
        labelmatch2(LineLabel,Rest).
labelmatch3(LineLabel,[Labeling|_]) :-
        label(_,Labeling,_,_,TestLabel),
        compatible(LineLabel,TestLabel),!.
labelmatch3(LineLabel,[_|Rest]) :-
        labelmatch3(LineLabel,Rest).

/* PREDICATE: replace_known(+Potential_Labels,-Known_Labels)

        Potential_Labels - the initial set of potential labels for
                each vertex
        Known_Labels - the same, only with the labels appearing
                in the 'known' predicate replacing those in
                the initial set.

        replace possible labelings by known labelings
*/
replace_known([],[]).
replace_known([[Vertex,Poss]|Rest],[[Vertex,Labels]|KRest]) :-
        known(Vertex,Labels),!,replace_known(Rest,KRest).
replace_known([Head|Rest],[Head|KRest]) :-
        replace_known(Rest,KRest).

/* PREDICATE: checkscene

        check the scene for consistency:
                each line connects to exactly two vertices
                each vertex connects to 2 or 3 lines according to type
*/
checkscene :-
        ((vertex(Vertex,Type,Line1,Line2,Line3),
          checkvertex(Vertex,Type,Line1,Line2,Line3),
          fail
         )
         ;
         true
        ),
        ((vertex(Vertex,Type,Line1,Line2,Line3),
          checkline(Line1),
          checkline(Line2),
          checkline(Line3),
          fail
         )
         ;
         true
```

```
        write('Scene check done'),nl.

/* PREDICATE: checkvertex(+Vertex,+Type,+Line1,+Line2,+Line3)

        Vertex - which vertex
        Type - type of vertex
        LineN - connecting lines

        check the given vertex for consistency:
                it connects to 2 or 3 lines according to type
*/
checkvertex(Vertex,Type,Line1,Line2,Line3) :-
        ((Type = ell ; Type = fork ; Type = arrow ; Type = tee),
          Line1 == null,
          Line2 == null,
          (Type = ell
                -> Line3 = null
                ;  Line3 == null
          ),
          Line1 == Line2,
          Line2 == Line3,
          Line3 == Line1,!
        )
        ;
        write('Vertex '),write(Vertex),write(' is bad'),nl.

/* PREDICATE: checkline(+Line)

        Line - desired line

        makes sure line connects to exactly 2 vertices
*/
checkline(null) :- !.
checkline(Line) :-
        bagof(Vertex,
                T^L1^L2^(
                        vertex(Vertex,T,Line,L2,L1) ;
                        vertex(Vertex,T,L2,Line,L1) ;
                        vertex(Vertex,T,L2,L1,Line)
                        ),
                Vertices),
        sizeof(Vertices,N),
        (N == 2
                -> (write('Line '),write(Line),
                    write(' does not connect to exactly two vertices'),nl
                   )
                ; true
        ).


/* PREDICATE: size(Set,Size)

        +Set - processed set
        -Size - output size

finds the size of a set
*/
sizeof(Set,N) :-
        sz(Set,0,N).
sz([],N,N) :- !.
sz([H|T],M,N) :- M1 is M + 1, sz(T,M1,N).

/* PREDICATE:   writesoln(+Labeling) - for solution labelings
                writelabel(+Labeling) - for any other labelings

        +Labeling - a labeling of the diagram

        writes out a labeling
*/
writesoln(L) :-
        nl,write('*** Solution Labeling ***'),nl,writelabels(L).
writelabels([]) :- !.
writelabels([[Vertex,Labels]|Tail]) :-
        write('Vertex '),write(Vertex),
        write(' has labels '),write(Labels),nl,
        writelabels(Tail).

/* PREDICATE: findassoc(+Vertex,+All_Labels,-Labels)

        Vertex - the desired vertex
        All_Labels - all current labels for all vertices
        Labels - The current labels for the given vertex

        finds whats currently associated with a vertex
*/
findassoc(Vertex,[[Vertex,Assoc]|_],Assoc) :- !.
findassoc(Vertex,[_|Rest],Assoc) :-
        findassoc(Vertex,Rest,Assoc),!.

/* PREDICATE: compatible(+Label1,+Label2)

        Label1,Label2 - the line labels at opposite ends of a line

        test label compatibility
*/
compatible(minus,minus).
compatible(plus,plus).
compatible(in,out).
compatible(out,in).
```

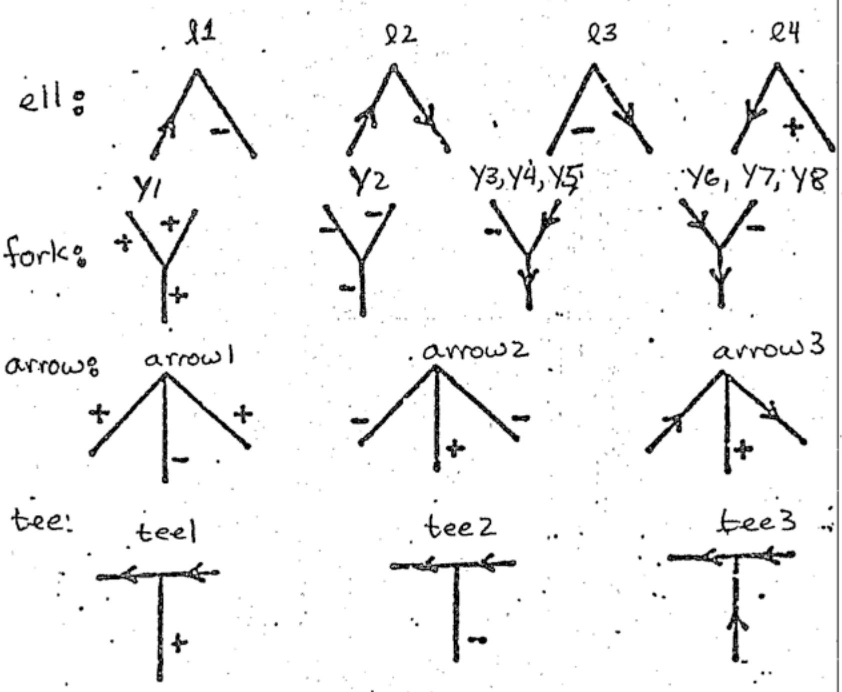
SCENE 1



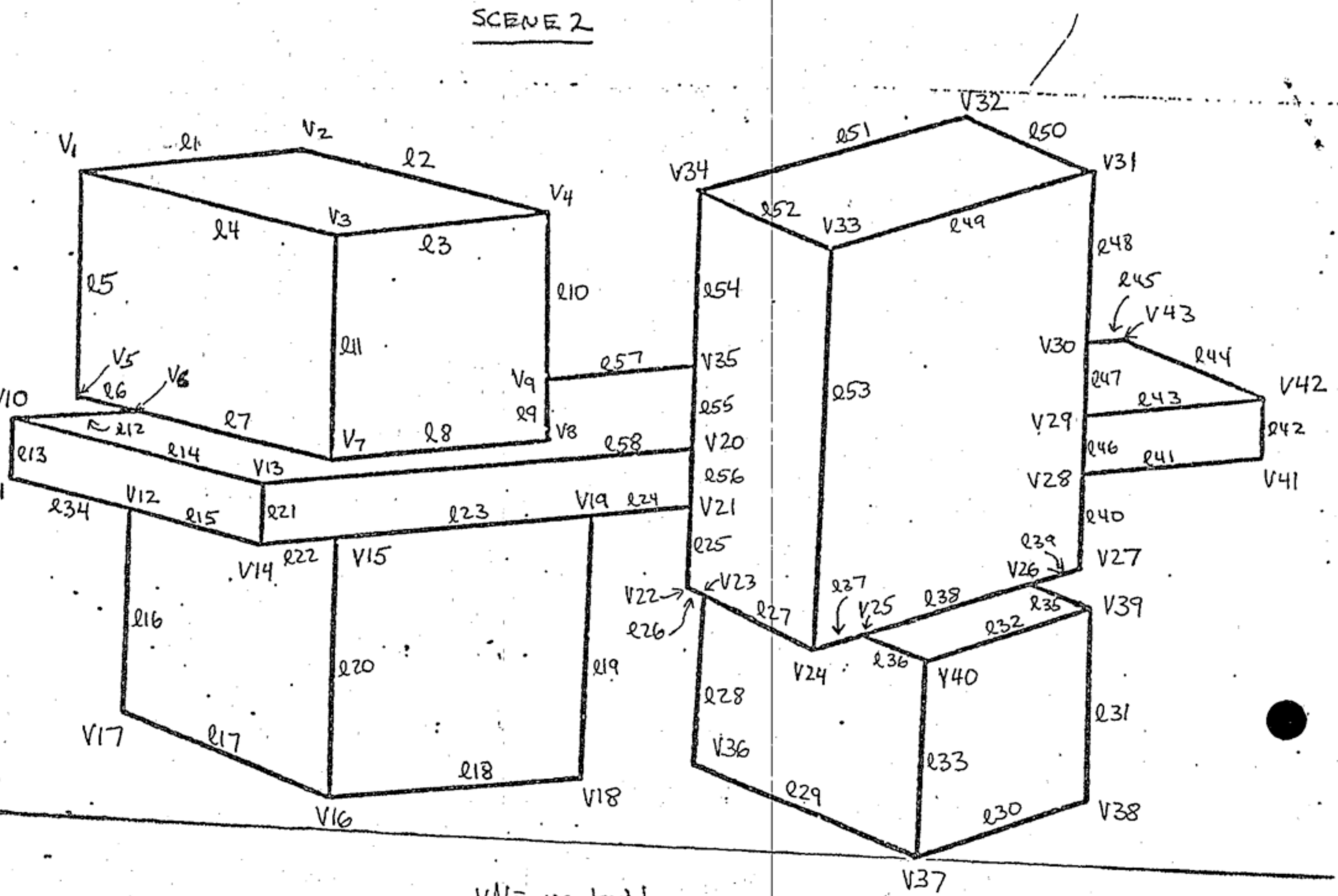
VN ≡ vertexN

lN ≡ lineN

LABELS



SCENE 2



VN ≡ vertex N
ℓN ≡ line N