

Assignment #7 -- Computer Vision: Constraint Propagation in the Blocks World

Due: November 19th, 2009

The goal of this assignment is to write a program that solves just the line-labeling step of the blocks-world picture-analysis problem. The full problem starts from a gray-scale pixel array, and ends with a list and description of all the objects in the picture. One way of breaking down the full problem is as follows (the part you are responsible for is in bold):

Initial input representation: a pixel array of gray-scale values, ranging from 0 for black to some upper value for white.

Step 1: Smoothing and edge detection (low-level vision).

After first step: A pixel array of bits (1's and 0's), with 1 representing an edge, and 0 representing everything else -- that is, background and object interiors but not object edges.

Step 2: Vertex and line detection. This step is a problem in line-growing, and is largely mathematical, though it does pose some interesting AI questions (e.g., how to decide if a not-perfectly-straight line is really straight enough to count). Unary constraints (border-finding) may also be handled at this stage.

After second step: A list of vertices and lines, with the vertices represented by names and x-y coordinate locations on the two-dimensional coordinate plane. Some lines may be labeled as external border lines.

Step 3: Vertex type analysis. This step involves the mathematics of angles.

After third step: A list of vertices and lines, with the vertices now represented by their type (using the L-junction, Y-junction, T-junction, and arrow-junction types) and with the lines populated with all possible labels, or with just an arrow label if it is an external border.

Step 4: Line labeling with constraint propagation. Go from a partially constrained labeling (in which you know vertex types and unary constraints) to a full-constrained labeling.

After fourth step: At this stage the lines should all be labeled with a single line label (plus, minus, left-arrow, or right-arrow) unless the picture is truly ambiguous.

Step 5: Object separation.

After fifth step: Objects should now be separated; each object is defined as a set of lines (and vertices) that are part of it (rep5 in test data).

Step 6: Object identification analysis. Determining from a fully-constrained labeling of a set of lines known to belong to an object, exactly what that object is.

After sixth step: The objects are identified and named by type -- that is, as a pyramid or block in our very simple example.

Your Assignment

Your assignment is to solve Step 4 of the complete problem as defined above. On the next page are three examples of input data for various line drawings, along with the anticipated changes to that data after a successful constraint propagation. The test data file is also available as a separate download, and is online at `~lib220/asst7/testdata.l` on the FAS machines.

You may dispense with my test data and choose your own representation if your program can be written more easily using some other format, and you may use a language other than LISP if you prefer; however, constraint propagation lends itself nicely to a LISP-ish solution.

You may wish to begin by reverse-engineering the original pictures from which the test data were drawn, to get a better visual image and to convince yourself that the provided output is correct.

You may also wish to draw some more complex images and create additional test data.

What to Hand In

As usual, turn in a well-commented listing of your program, and a runtime script showing your programming running on the test data.

Optional Extensions

This assignment has its own extensions built right in. You may work on the rest of the blocks-world computer vision problem. Of course, you'll need to provide your own test data.

Test Data File

```
;;; How to use this testdata file:
;;;
;;; (1) Load the file into LISP.
;;;
;;; (2) Load an example by running the respective function, e.g. (example1).
;;;
;;; (3) Since all properties are globally-defined, you MUST exit LISP and restart
;;;     between examples.
;;;
;;;=====

;;; PUTPROP is provided for readability and compatibility with older
;;; versions of LISP.

(defun putprop (atom property value)
  (setf (get atom property) value))

;;;=====

;;;=====
```

```

;;; EXAMPLE 1

;;; INPUT REPRESENTATION
;;; In this representation, the vertex types for each point have been
;;; determined and stored on the "vertex" property.
;;;
;;; The labels for each line have been initialized and stored on a "label"
;;; property. The unary constraints have already been handled for you at
;;; this stage, by labeling the "outside" lines unambiguously with the
;;; appropriate arrows. All the other lines have been ambiguously labeled
;;; with a list of all possible labels.

;;; The ordering of the adjacent vertices on the 'vertex property is
;;; very important. I have chosen a left-to-right representation with
;;; the vertex uppermost. In the case of a Y-junction, the ordering
;;; is immaterial.
;;;
;;; For line-labels, the label L-ARROW means the arrow is going towards
;;; the first endpoint (the CAR of the endpoint property value), and
;;; R-ARROW means the arrow is going towards the second endpoint.

(defun example1 ()
  (setq POINTS '(A B C D))

  (putprop 'A 'coords '(4 . 3))
  (putprop 'B 'coords '(9 . 14))
  (putprop 'C 'coords '(14 . 6))
  (putprop 'D 'coords '(10 . 0))

  (setq LINES '(AB BC CD DA DB))

  (putprop 'AB 'endpoints '(A . B))
  (putprop 'BC 'endpoints '(B . C))
  (putprop 'CD 'endpoints '(C . D))
  (putprop 'DA 'endpoints '(D . A))
  (putprop 'DB 'endpoints '(D . B))

  (putprop 'A 'vertex '(el D B))
  (putprop 'B 'vertex '(arrow A D C))
  (putprop 'C 'vertex '(el B D))
  (putprop 'D 'vertex '(arrow C B A))

  (putprop 'AB 'label '(R-ARROW))
  (putprop 'BC 'label '(R-ARROW))
  (putprop 'CD 'label '(R-ARROW))
  (putprop 'DA 'label '(R-ARROW))
  (putprop 'DB 'label '(PLUS MINUS L-ARROW R-ARROW)))

;;; OUTPUT REPRESENTATION
;;; The only difference between this representation and the previous one
;;; is that the line-labeling has been completed; every line now has a
;;; single labeling (unless the drawing is truly ambiguous).

(defun example1-output ()
  (example1)

  (putprop 'DB 'label '(PLUS)))

;;;=====

```

```
;;; EXAMPLE 2
```

```
;;; INPUT REPRESENTATION
```

```
(defun example2 ()  
  (setq POINTS '(A B C D E F G))  
  
  (putprop 'A 'coords '(5 . 2))  
  (putprop 'B 'coords '(5 . 12))  
  (putprop 'C 'coords '(8 . 16))  
  (putprop 'D 'coords '(21 . 16))  
  (putprop 'E 'coords '(21 . 5))  
  (putprop 'F 'coords '(17 . 2))  
  (putprop 'G 'coords '(17 . 12))  
  
  (setq LINES '(AB BC CD DE EF FA BG DG FG))  
  
  (putprop 'AB 'endpoints '(A . B))  
  (putprop 'BC 'endpoints '(B . C))  
  (putprop 'CD 'endpoints '(C . D))  
  (putprop 'DE 'endpoints '(D . E))  
  (putprop 'EF 'endpoints '(E . F))  
  (putprop 'FA 'endpoints '(F . A))  
  (putprop 'BG 'endpoints '(B . G))  
  (putprop 'DG 'endpoints '(D . G))  
  (putprop 'FG 'endpoints '(F . G))  
  
  (putprop 'A 'vertex '(el F B))  
  (putprop 'B 'vertex '(arrow A G C))  
  (putprop 'C 'vertex '(el B D))  
  (putprop 'D 'vertex '(arrow C G E))  
  (putprop 'E 'vertex '(el D F))  
  (putprop 'F 'vertex '(arrow E G A))  
  (putprop 'G 'vertex '(y B D F))  
  
  (putprop 'AB 'label '(R-ARROW))  
  (putprop 'BC 'label '(R-ARROW))  
  (putprop 'CD 'label '(R-ARROW))  
  (putprop 'DE 'label '(R-ARROW))  
  (putprop 'EF 'label '(R-ARROW))  
  (putprop 'FA 'label '(R-ARROW))  
  (putprop 'BG 'label '(PLUS MINUS L-ARROW R-ARROW))  
  (putprop 'DG 'label '(PLUS MINUS L-ARROW R-ARROW))  
  (putprop 'FG 'label '(PLUS MINUS L-ARROW R-ARROW))
```

```
;;; OUTPUT REPRESENTATION
```

```
(defun example2-output ()  
  (example2)  
  
  (putprop 'BG 'label '(PLUS))  
  (putprop 'DG 'label '(PLUS))  
  (putprop 'FG 'label '(PLUS)))
```

```
;;;=====
```

```
;;; EXAMPLE 3
```

```
;;; INPUT REPRESENTATION
```

```
(defun example3 ()  
  (setq POINTS '(A B C D E F G H I J))  
  
  (putprop 'A 'coords '(2 . 4))  
  (putprop 'B 'coords '(2 . 11))  
  (putprop 'C 'coords '(6 . 15))  
  (putprop 'D 'coords '(9 . 15))  
  (putprop 'E 'coords '(11 . 22))  
  (putprop 'F 'coords '(18 . 5))  
  (putprop 'G 'coords '(13 . 2))  
  (putprop 'H 'coords '(10 . 4))  
  (putprop 'I 'coords '(7 . 6))  
  (putprop 'J 'coords '(9 . 11))  
  
  (setq LINES '(AB BC CD DE EF FG GH HA BJ HI IJ JD))  
  
  (putprop 'AB 'endpoints '(A . B))  
  (putprop 'BC 'endpoints '(B . C))  
  (putprop 'CD 'endpoints '(C . D))  
  (putprop 'DE 'endpoints '(D . E))  
  (putprop 'EF 'endpoints '(E . F))  
  (putprop 'FG 'endpoints '(F . G))  
  (putprop 'GH 'endpoints '(G . H))  
  (putprop 'HA 'endpoints '(H . A))  
  (putprop 'BJ 'endpoints '(B . J))  
  (putprop 'HI 'endpoints '(H . I))  
  (putprop 'IJ 'endpoints '(I . J))  
  (putprop 'JD 'endpoints '(J . D))  
  
  (putprop 'A 'vertex '(el H B))  
  (putprop 'B 'vertex '(arrow A J C))  
  (putprop 'C 'vertex '(el B D))  
  (putprop 'D 'vertex '(tee E C J))  
  (putprop 'E 'vertex '(arrow D G F))  
  (putprop 'F 'vertex '(el E G))  
  (putprop 'G 'vertex '(arrow F E H))  
  (putprop 'H 'vertex '(tee I A G))  
  (putprop 'I 'vertex '(el H J))  
  (putprop 'J 'vertex '(tee D B I))  
  
  (putprop 'AB 'label '(R-ARROW))  
  (putprop 'BC 'label '(R-ARROW))  
  (putprop 'CD 'label '(R-ARROW))  
  (putprop 'DE 'label '(R-ARROW))  
  (putprop 'EF 'label '(R-ARROW))  
  (putprop 'FG 'label '(R-ARROW))  
  (putprop 'GH 'label '(R-ARROW))  
  (putprop 'HA 'label '(R-ARROW))  
  (putprop 'BJ 'label '(PLUS MINUS L-ARROW R-ARROW))  
  (putprop 'HI 'label '(PLUS MINUS L-ARROW R-ARROW))  
  (putprop 'IJ 'label '(PLUS MINUS L-ARROW R-ARROW))  
  (putprop 'JD 'label '(PLUS MINUS L-ARROW R-ARROW)))
```

```
;;; OUTPUT REPRESENTATION
```

```
(defun example3-output ()  
  (example3)  
  
  (putprop 'BJ 'label '(PLUS))  
  (putprop 'HI 'label '(R-ARROW))  
  (putprop 'IJ 'label '(R-ARROW))  
  (putprop 'JD 'label '(R-ARROW)))
```