Decision-Based Specification and Comparison of Table Recognition Algorithms

Richard Zanibbi[1], Dorothea Blostein[2], and James R. Cordy[2]

[1] Department of Computer Science, Rochester Institute of Technology, 102 Lomb Memorial Drive, Rochester, New York, USA 14623-5603, rlaz@cs.rit.edu
[2] School of Computing, Queen's University, Kingston, Ontario, Canada, K7L 3N6 {blostein,cordy}@cs.queensu.ca

Summary. Recognition algorithms are difficult to write and difficult to maintain. There is need for better tools to support the creation, debugging, optimization, and comparison of recognition algorithms. We propose an approach that centers on a process-oriented description. The approach is implemented using a new scripting language called RSL (Recognition Strategy Language), which captures the recognition decisions an algorithm makes as it executes. This semi-formal process-oriented description provides a powerful basis for developing and comparing recognition algorithms. Based on this description, we describe new metrics related to the sequence of decisions an algorithm makes during recognition. The capture of intermediate decision outputs and these new process-oriented metrics greatly extend the limited information available from final results and traditional results-oriented metrics such as recall and precision. Using a simple example, we illustrate how these new metrics can be used to understand and improve decisions within a recognition strategy. We believe these new metrics may also be applied in machine learning algorithms that construct optimal decision sequences from sets of decisions and/or strategies.

1 Introduction

Tables have been used to summarize vast quantities of information in books, papers, text files, electronic documents, and HTML web pages. Significant efforts have been made towards developing automated and semi-automated methods for searching, extracting, summarizing, and integrating the information they contain. A wide variety of algorithms have been published for detecting tables and analyzing their structure, and a number of surveys on table recognition and processing are available [3, 4, 5, 6, 7, 8].

Algorithms that recognize tables search a space of possible interpretations in order to describe the tables present in an input file. The space of interpretations is defined by some model of table locations and structure whose

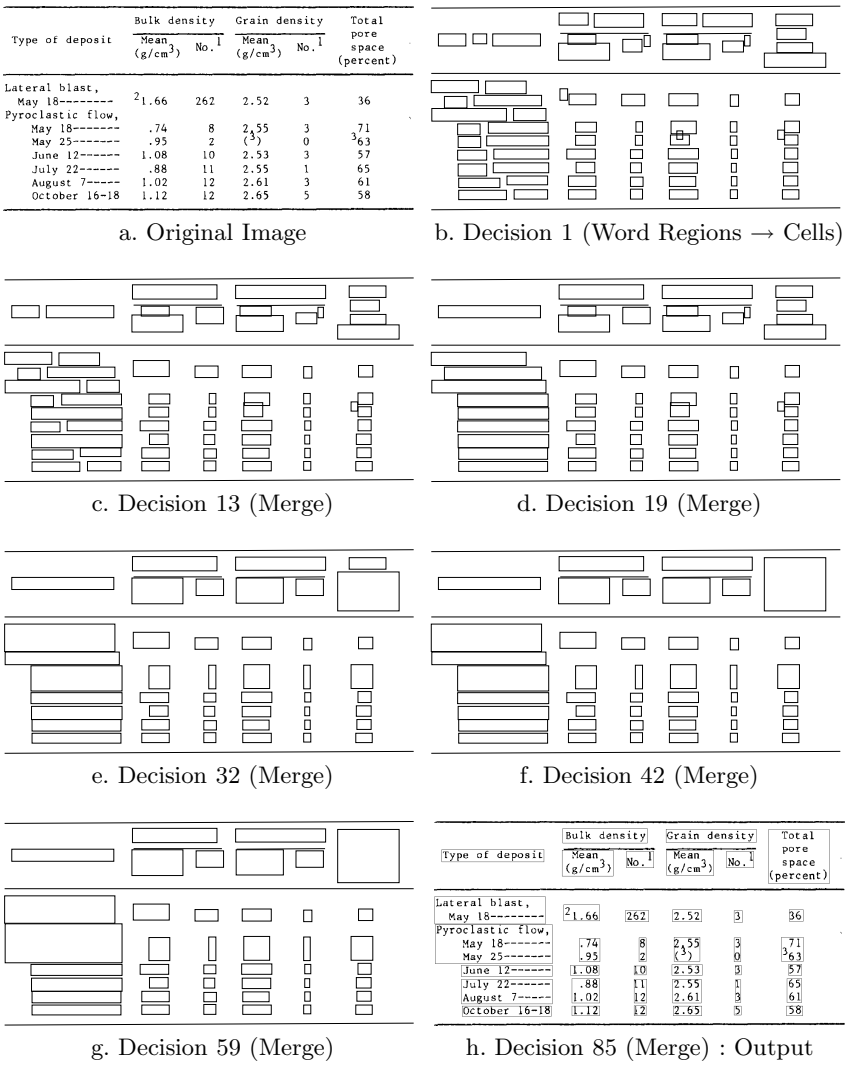


Fig. 1. Production of Cell Hypotheses by the Handley Algorithm [1]. (a) Input image from the University of Washington Database [2], page a038. (b) *Line* and *Word* regions are defined manually. All *Word* regions are immediately classified as cells in the first step. (c) to (h) *Cell* region hypotheses are refined through merging (see caption of Figure 9 for further details)

operations are invoked by the algorithm while searching. The search for tables and/or structural elements of tables proceeds through a series of decisions that determine which model operations to apply, and which interpretations are worth pursuing or are acceptable for output.

Currently it is difficult to compare table recognition algorithms. One difficulty is that the algorithms often address different problems, and do not use the same model to describe tables as a result [9, 10]. For example, some algorithms assume that all table cells are separated by ruling lines, while in others ruling lines are optional or are assumed to be absent. As another example, some algorithms aim to detect cell locations in an image or text file, while other methods assume cell locations are known, and then attempt to recover the indexing structure of the table from header cells to body cells. Also, algorithms are usually defined informally in the literature, often with no explicit description of the table model or its associated space of possible interpretations. This makes table recognition algorithms hard to replicate, compare, and combine.

We do not believe that it is possible or even desirable for researchers to adopt a standard table model. We propose instead to make table models and their operations more explicit, in order to permit easier identification of common elements in models and recognition techniques. A *decision-based specification* of a table recognition algorithm defines a sequence of decisions that each select from an explicit set of table model operations. Each decision is defined in terms of the input data used to make the decision, the set of model operations from which to select (defined using the inputs), and a function that will decide which operations to select at run-time.

In Section 3 we present the Recognition Strategy Language (RSL [11]), a functional scripting language for creating and executing decision-based specifications. RSL provides a small but general set of decision types used to manipulate interpretations represented by attributed graphs. We then demonstrate how decision-based specifications allow algorithms to be compared through decisions affecting common table model elements. Graph-based summaries of table models used for recognition may be produced through static analysis of RSL programs (see Section 4). After execution, the results of intermediate decisions may be observed and compared in addition to the final results. To illustrate, in Section 6 we compare the detection of table cells in two published algorithms [1, 12] that have been re-implemented in RSL, and then combine the algorithms to produce a better result. Figure 1 illustrates how one of these algorithms creates and discards table cell hypotheses.

Commonly, measures such as recall and precision are used to evaluate recognition algorithms. In the case of cells in table recognition, recall is the percentage of ground-truth cells that are detected correctly, and precision is the percentage of detected table cells that are in the ground truth. Using recall and precision, recognition is characterized only in terms of final results, and not in terms of the *process* of recognition; there is no characterization of

whether individual decisions are helpful, detrimental, or irrelevant for accurately recognizing tables within a set of documents.

In conventional performance evaluations there is also no characterization of false negatives: these are valid hypotheses that are created and later rejected by an algorithm. RSL records a complete history of all decision outcomes, which makes it possible to determine which decisions create, reject, and reinstate each generated hypothesis after execution (see Figures 9 and 10). Taking advantage of this additional information, we present *historical recall* and *historical precision*, two new metrics that characterize the complete set of generated hypotheses (e.g. cell locations) in Section 5.

Aside from table recognition, we believe that the decision based-approach to specification, comparison, and evaluation may be usefully applied to many other document recognition and machine learning problems. These include problems in document layout analysis such as page region labeling and interpreting symbol layout within page regions (e.g. in text regions, figures, and diagrammatic notations) where a series of decisions must be made about region types, locations, and relationships, and the problem of classifier combination, where a decision-based specification would support rapid construction and modification of classifier architectures from existing classifiers. Other chapters in this book provide more information about machine learning techniques related to document layout analysis and classifier combination. The decision-based approach provides intermediate interpretations and metrics for decision sequences (e.g. historical recall and precision) that could be used by machine learning algorithms to train or optimize individual decisions within a recognition strategy, or to learn recognition strategies themselves given a set of decisions and/or strategies [13, 14, 15].

We begin our main discussion by introducing table recognition problems as sequential decision-making problems in more detail.

2 Recognizing Tables: A Decision-Making Problem

Researchers have used the term physical structure to refer to explicit properties of an encoding, and logical structure to refer to properties that are implicit in an encoding [5, 16, 17]. Simple examples of physical structure include the number of rows and columns in an image file, the number of characters in each text line of a text file, and the location of a pixel or character within an image or text file (this is usually expressed geometrically, in terms of columns and rows).

Source files for markup language encodings such as HTML and XML have the same physical structure as plain text files, but also represent regions of the file and the relationships between them using text demarcators ('tags'). To recover the information represented by the demarcators requires an inferential process that uses the rules of structure for the appropriate markup language (i.e. we must parse the file using an appropriate grammar). Properties of a file

that must be recovered using an inferential process such as this are referred to as logical structure.

Markup language parsers are formal language recognizers that recover implicit logical structure from text encodings. From a pattern recognition perspective they are considered simple because the information used to deduce the tag structure is largely fixed and unambiguous, and parsers assume that input files contain data in the correct format.

Table recognition is a more difficult problem because which information and inferential processes to use for recovering tables are neither fixed nor unambiguous, and are subjects of ongoing study. Table structure is frequently adapted to the idiosyncratic needs of authors, making it difficult and perhaps impossible to make a single, formal definition. Unless strong assumptions about tables in input files are made, there is the additional problem of selecting which model of table structure to apply.

Even if a valid table structure model is used for an input file, the set of possible model instances may be large, and it is often difficult to define inferencing methods that reliably identify table structures due to noise in the file (e.g. smearing in images) and unanticipated properties of the input. For systems recognizing tables in raw images, all table properties are implicit. At the other extreme, for HTML files often table cells, rows, and columns are already represented; however, tags for tables are often used to arrange arbitrary data visually, and determining which encoded tables are real tables is a difficult problem [18, 19].

In our work we have come to view structural pattern recognition problems such as table recognition as sequential decision-making problems: given a model of logical structure to be recovered from a set of input files, what series of decisions about model operations to apply will produce interpretations of logical structure that maximize a given performance criterion? For recognition tasks, the performance criterion will be some combination of metrics for execution time, storage requirements, and accuracy. Similar views of structural recognition problems have been proposed in the machine learning literature [20, 21]. Accuracy metrics are influenced by the chosen source(s) of ground truth; to accommodate this, a problem formulation in which the source of ground truth is an explicit variable may be used (we have proposed one such formulation [10]).

We wish to be able to easily compare, re-sequence, and combine individual decisions about model operations made by different table recognition algorithms. We also wish to be able to easily evaluate interpretations produced by intermediate decisions, and characterize the complete set of hypotheses generated by an algorithm, not just those accepted at a given point in the algorithm's execution.

Currently in the literature, table recognition algorithms are most commonly characterized as a sequence of operations that are implemented in a general-purpose programming language. Less frequently model-driven specifications are employed, in which a representation of the table model is used

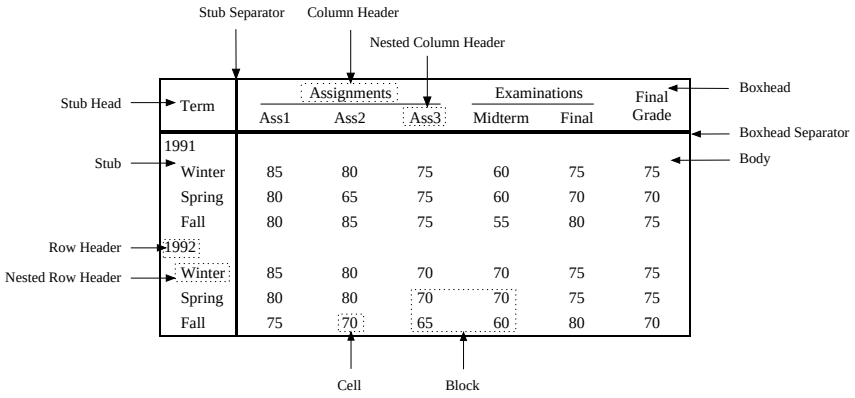


Fig. 2. Table Structure. This example is taken from Wang [26], and uses terminology taken from the Chicago Manual of Style [27]

to 'program' the system (e.g. as an attributed grammar with rules containing associated actions [22, 23, 24]). Given a model specification and an input file, the sequence of decisions to make is determined algorithmically (e.g. by a parser): the model definition is mapped to a decision sequence. The procedural approach has the benefit of being highly flexible, while the model-driven approach has the benefits of concise specification and a level of formality which permits more information about decision-making to be automatically collected.

For our purposes, the primary disadvantage of procedural implementations is that operations for computing input data for decisions, making decisions, and applying model operations are represented in the same way. This makes it difficult to extract individual decisions. A disadvantage of model-driven systems is that their model definitions are usually tied quite tightly to a particular set of recognition techniques, and it can be difficult to modify these systems to accommodate unanticipated requirements and/or new techniques [25].

In Section 3 we present the Recognition Strategy Language (RSL [11]), which provides a way to express recognition algorithms as decision sequences and automatically capture decision outcomes, while maintaining much of the flexibility of procedural implementations. In this paper we present two procedural table recognition algorithms that have been successfully reimplemented in RSL (see Section 3.3). It may be worth investigating whether there would be benefits to having model-driven systems use a decision-based representation for the output algorithm. A model-driven system might compile a model definition into an RSL program, for example. In this way, decision-based specifications provide an intermediate level of abstraction between procedural and model-driven system specifications.

2.1 Table Structure and Terminology

The most abstract representation of a table is the underlying data set that the table visualizes; often a great deal of domain knowledge about the information represented in a table is needed to recover this. A slightly less abstract representation is the indexing structure, which describes how cells in the body of the table are indexed by header cells in the boxhead, stub, and stubhead of the table. As an example, in Figure 2, we might use a dot notation similar to that of Wang [26] to represent the grade in the top-left corner of the body as ((Term.1991.Winter,Assignments.Ass1),85). This also often requires linguistic and/or domain knowledge to recover as well [28], but to a lesser extent.

Closer to the physical structure of images and text files we have the table grid, which describes the arrangement of cells in rows and columns, and the location of ruling lines and whitespace separators. This is often represented by extending all separators to the boundaries of a table, and then assigning cells and separators to locations within the resulting grid.

The main structural elements of a printed table are illustrated in Figure 2. Cells of the table are organized in rows and columns. Some cells such as the column header 'Assignments' are *spanning cells*, which belong to more than one column and/or row of the table. Cells are further distinguished as *header cells*, which determine how *data cells* in the body of the table are indexed. Header cells which are indexed by other header cells are termed *nested row* and *nested column* headers, as appropriate.

Commonly there are four regions of a table, as illustrated in Figure 2. The *body* contains the data cells, which normally are the values intended to be most easily searched and compared within the table. Header cells appear in the *boxhead* (labeling columns) and *stub* (labeling rows, when present). Sometimes, as in Figure 2, there is a header in the top-left region of the table, called the *stub head*. An adjacent set of cells in the table body is referred to as a *block*.

3 The Recognition Strategy Language: Decision-Based Specification

As a first step towards being able to more easily observe intermediate decisions and combine decisions from different recognition algorithms, we have devised the Recognition Strategy Language (RSL [11]). RSL provides a level of formalization that lies between procedural and model-driven specifications (see Section 2). Based on notes from our survey of table recognition [8], the language allows models of table structure to be defined in terms of three basic decisions for regions in an input file: classification, segmentation, and parsing (binary relationships).

We refer to RSL as a decision-based specification because the basic unit of formalization defines the inputs and acceptable outputs for decisions. An

RSL program defines properties of interpretations, a single set of constant and variable parameters used for making decisions, and the sequencing of the decisions. Decision outcomes are provided at run-time by functions referred to in the decision specifications, but which are defined outside of the RSL program. Functions that provide the decision outcomes may use arbitrary techniques to arrive at a decision.

As currently defined, RSL is a simple functional scripting language influenced by the text-based approach taken in Tcl/Tk [29], with some similarities to an expert system shell [30]. Decisions in RSL are similar to rules in an expert system in that both define transformations of a knowledge representation, though RSL decisions define only possible transformations, and not concrete transformations as in the case with rules. The ability to trace rule applications in an expert system shell is replaced by record-keeping and annotation in RSL, and RSL specifications are sequential, and not declarative (expert system shells often support applying rules using various search algorithms).

RSL strategies may be interpreted for rapid development, or compiled. RSL has been implemented using the TXL language [31], which is a functional language for rapid prototyping that has been used for a wide array of applications including source code transformation, design recovery, and pattern recognition [32, 33].

In the remainder of this section we provide an overview of RSL and a simple example of an RSL program and its execution.

3.1 Transforming Logical Structure Interpretations in RSL

Interpretations of logical structure are represented by attributed directed graphs in RSL. Graph nodes represent geometric regions of an input file (R), and graph edges represent relations on regions. The *contains* relation defines the combination of regions into region segments (S), and additional binary relations may be defined (E). The main elements of an interpretation in RSL are:

- V, the set of expressible geometric locations in an input file, defined using a set of functions. Currently in RSL there are only two functions used to define V: one for bounding boxes, v_{BB} (defined by top-left, bottom-right corners), and another for polylines, v_l (a list of points).
- $R \subseteq (I \times V)$, the set of input file regions used in an interpretation (defines nodes of the graph). I is a set of legal identifiers. Each region in R has a unique identifier, but regions may have identical locations
- $S = \{(S_1 \subseteq R, r_1 \in R), \ldots, (S_n \subseteq R, r_n \in R)\}$, the set of region segments (regions containing other regions). Each set of regions $S_j \subseteq R$ is unique, and defines a region $r_j \in R$: currently in RSL this region is located at the bounding box of the regions in the segment (S_j)
- $C = \{(i_1 \in I, C_1 \subseteq R), \ldots, (i_n \in I, C_n \subseteq R)\}$, the regions associated with each region type identified by $i_j \in I$ (e.g. *Word*, *Cell*, *Row*, and *Column*)

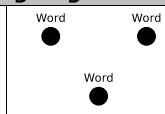
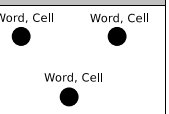
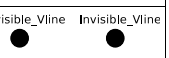
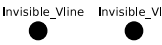
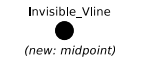
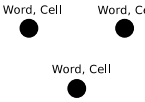
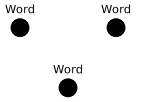
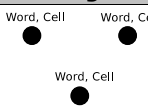
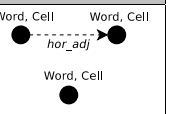
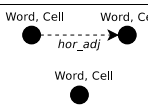
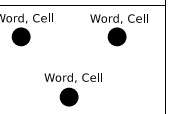
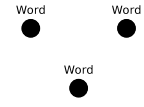
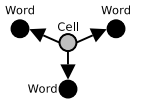
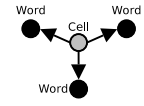
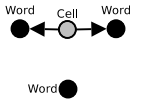
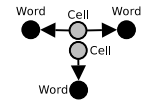
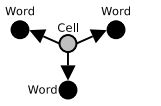
RSL Decision Type (Operation)	Input	Output
Classification: Labeling Regions		
classify { Word } **regions as** { Cell } **classify** { C_1, C_2, ... } **regions as** { C_{o1}, C_{o2}, ... } **using** **...**		
create { Invisible_Vline } **regions using** **...**		
replace { Invisible_Vline } **regions using** **...**		
reject { Cell } **classifications** **reject** { C_1, C_2, ... } **classifications using** **...**		
Relating: Binary Relations on Regions		
relate { Cell, Cell } **regions with** { hor_adj } **using** ...		
reject { hor_adj } **relations** **reject** { C_1, C_2, ... } **relations using** **...**		
Segmentation: Grouping Regions		
segment { Word } **regions into** { Cell } **segment** { C_1, C_2, ... } **regions into** { C_{o1} } **using** **...**		
resegment { Word } **regions into** { Cell } **using** **...** **resegment** { C_1, C_2, ... } **regions into** { C_{o1} } **using** **...**		
merge { Cell } **regions using** **...**		

Fig. 3. RSL Decision Types for Transforming Interpretations. For the example inputs and outputs, only accepted hypotheses are shown.

- $E = \{(i_1 \in I, E_1 \subseteq R^2), \ldots, (i_n \in I, E_n \subseteq R^2)\}$, defines binary relations on regions for relation types identified by $i_j \in I$ (e.g. horizontal adjacency)
- A_0, the set of named attributes associated with elements of R, S, and E in the interpretation provided as input (I_0). Attribute values are lists of strings or floating point numbers

Input to an RSL program is an initial interpretation (I_0) that defines the initial sets of regions, segments, region classes, and region relations (R, S, C, and E) and their attributes (A_0). Within A_0, the file described by the interpretation is represented by a single region, with the name of the file provided as an attribute of the region. For example, image files may be represented by a region of type *Image* with a text attribute *FILE_NAME*. Currently only elements in the input interpretation I_0 are permitted to have additional attributes (to avoid side-effects [34]).

The output of an RSL program is a set of accepted interpretations, with each interpretation annotated with a complete history of the model operations that produced it. This allows all intermediate states of an interpretation to be recovered. While RSL supports the selection of multiple alternatives at each decision point [34], here we will consider only the case where each decision selects exactly one alternative, producing a single interpretation as output.

Figure 3 summarizes the available decision types for transforming individual interpretations in RSL. Shown on the left of each row is the first line of an RSL decision specification. Each decision type indicates the interpretation elements that define the set of possible outcomes at run-time. In the case of `create` and `replace`, the alternative outcomes are implicitly defined using the set of all possible input regions (V). Figure 5 illustrates how alternatives are produced for a few decision types.

Examples of input and output interpretations are provided for each of these decision types in Figure 3. Some alternate forms for the decision types are also given. For example, more than one region type may be used to define both the regions to classify and the possible classes in a `classify` operation.

Decisions that generate hypotheses either classify, segment, or relate regions, while the `reject` decision type discards hypotheses. The `replace`, `resegment`, and `merge` operations both assert and reject hypotheses. `replace` returns sets of regions of a given type, replacing each with a new region of the same type. `merge` and `resegment` reject a region type for regions, replacing these with new region segments of the same type. `merge` combines regions and segments into new ones, while `resegment` is more general, and may be used to split segments as well (see examples in Figure 3).

RSL uses a simple method for classification in which all input regions are classified as at most one of the possible output classes (some subset of the region types in C): selecting no class indicates rejection. Segmentation operators simultaneously define segments and assign a type to each segment (altering S and C). Relating operations update the region edge sets in E.

Additional decision types for altering parameters of decision functions and accepting and rejecting interpretations are defined within RSL [34]. Other than to accept all interpretations produced before a strategy completes (see bottom of Figure 4), these are unused in any of the strategies described in this chapter.

3.2 A Simple RSL Strategy

Figure 4 provides an example of a simple RSL strategy, and Figure 5 shows an example execution of the first three decisions of the *main* function. The input interpretation shown in Figure 5 contains three *Word* regions. As discussed in the previous section, the decision type (first line) of each decision operator defines a fixed set of possible outcomes for the associated external decision function. External decision functions and the set of possible alternatives are shown in rounded boxes in Figure 5. Selected alternatives are returned as text records, which are first validated and then used to specify a transformation to apply to the interpretation. RSL records all the decision outcomes, and annotates changes made to the interpretation within the interpretation itself when applying a transformation.

In Figure 4, the `model regions` and `model relations` sections define the set of region and relation types for interpretations used in the strategy (i.e. define the elements of C and E). Any other relations or region types in the input interpretation are left intact, but otherwise ignored by the strategy. Decisions which refer to types not provided in these sections will produce a syntax error. The `recognition parameters` section defines a series of static (constant) and adaptive (variable) parameters for use in the external decision functions. Static parameters are indicated using a prefix 's' (e.g. *sMaxHorDistance* in Figure 4), and adaptive parameters using a prefix 'a' (e.g. *aMaxColSep* in Figure 4). All parameters may be string or floating point number-valued.

Control flow is specified in RSL using *strategy functions*, which define a sequence of decision operations and calls to other strategy functions. Recursion is permitted: see the *recursiveStrategy* in Figure 4. Each strategy function takes the current set of interpretations and parameter values as input, and produces a new set of interpretations as output. For conciseness, this is implicit rather than explicit within an RSL program (i.e. syntactically strategy functions look like procedures, but they are in fact functions).

Following convention, execution begins with the *main* strategy function, which is passed the input interpretation and the initial set of values for the adaptive parameters defined in the `recognition parameters` section.

There is only one form of conditional statement in RSL which acts as a guard for strategy functions. This statement prevents interpretations not selected by an external decision function from being altered by a strategy function, and must appear at the beginning of a strategy function declaration.

For example, in Figure 4 a `for interpretations` statement is used to define the stopping point for the *recursiveStrategy*. The statement given calls

an external decision function *adjacencyIncomplete* for each current interpretation, using the *aMaxColSep* and *aMaxHorSep* parameters. *adjacencyIncomplete* is required to specify which of the current interpretations the *recursiveStrategy* may be applied to.

The `observing` keyword is used to add visible types to the interpretations passed to external decision functions; by default, only the interpretation elements that may be altered by the decision are visible (these are called the *scope* types). For example, for classification operations, the visible regions include those associated with the types of input regions to be classified, but none of the possible *output* classes are visible unless they are also an input type or are explicitly added using `observing`. The `create` decision type is unique in that by default *no* region types are visible in the interpretations passed to the decision function: all visible types must be explicitly listed in an `observing` statement.

However, the sets of regions and region segments ($R \cup S$) in the interpretation are always visible to all external decisions. Consider Decision 2 in Figure 5, where the regions associated with *Word* are visible, but *not* their type. As one would expect, only parameters explicitly passed to the external decision function are visible for each decision.

3.3 Handley and Hu et al. Algorithms in RSL

As a proof of concept, we have re-implemented two table structure recognition algorithms in RSL [1, 12]. All external decision functions were implemented using TXL. These algorithms were chosen because they were part of a small set of algorithms described in enough detail to permit replication, and because they exhibited a degree of sophistication. Both algorithms are described procedurally, as sequences of operations.

As a brief summary, both algorithms recover table structure given a list of input *Line* and *Word* regions (note that the Hu et al. algorithm ignores the *Line* regions). The Handley algorithm uses a strictly geometry-based approach, in which *Word* regions are all hypothesized as *Cell* regions, and then merged in a series of steps which makes use of weighted projection profiles and cell adjacency relationships. The Hu et al. algorithm starts by detecting columns based on a hierarchical clustering applied to the horizontal spans of *Word* regions, detects the table boxhead, stub and rows of the table, and finally *Cell*s in the body of the table. The Hu algorithm makes use of limited (but powerful) lexical information in its analysis; text in input *Word* regions are classified as alphabetic or alphanumeric. Within RSL, the text of words is represented as attributes of the input *Word* regions (i.e. the word text is defined within A_0).

Reflecting their different intended applications, the Handley algorithm seeks only to recover table cells, while the Hu algorithm returns detected

```
model regions
    Word Cell Header Entry Image
end regions

model relations
    adj_right close_to
end relations

recognition parameters
    sMaxHorDistance 5.0 % mms
    aMaxColSep 20.0 % mms
    aMaxHorSep 25.0 % mms
end parameters

strategy main
    relate { Word } regions with { hor_adj } using
        selectHorizAdjRegions(sMaxHorDistance)

    segment { Word } regions into { Cell } using
        segmentHorizAdjRegions()
        observing { hor_adj } relations

    classify { Cell } regions as { Header, Entry } using
        labelColumnHeaderAndEntries()

    ...
    recursiveStrategy

    accept interpretations
end strategy

strategy recursiveStrategy
    for interpretations using
        adjacencyIncomplete(aMaxColSep,aMaxHorSep)
        observing { Word, Cell } regions { close_to } relations
    ...
    resursiveStrategy
end strategy
```

Fig. 4. A Simple RSL Strategy

rows, columns, and cell indexing structure for the table. Please note that we have modified the Hu algorithm with an extra step to define textlines using a simple projection of *Word* regions onto the Y-axis, defining textlines at gaps greater than a given threshold value. This was necessary because the algorithm was originally specified for detecting table structure within text files.

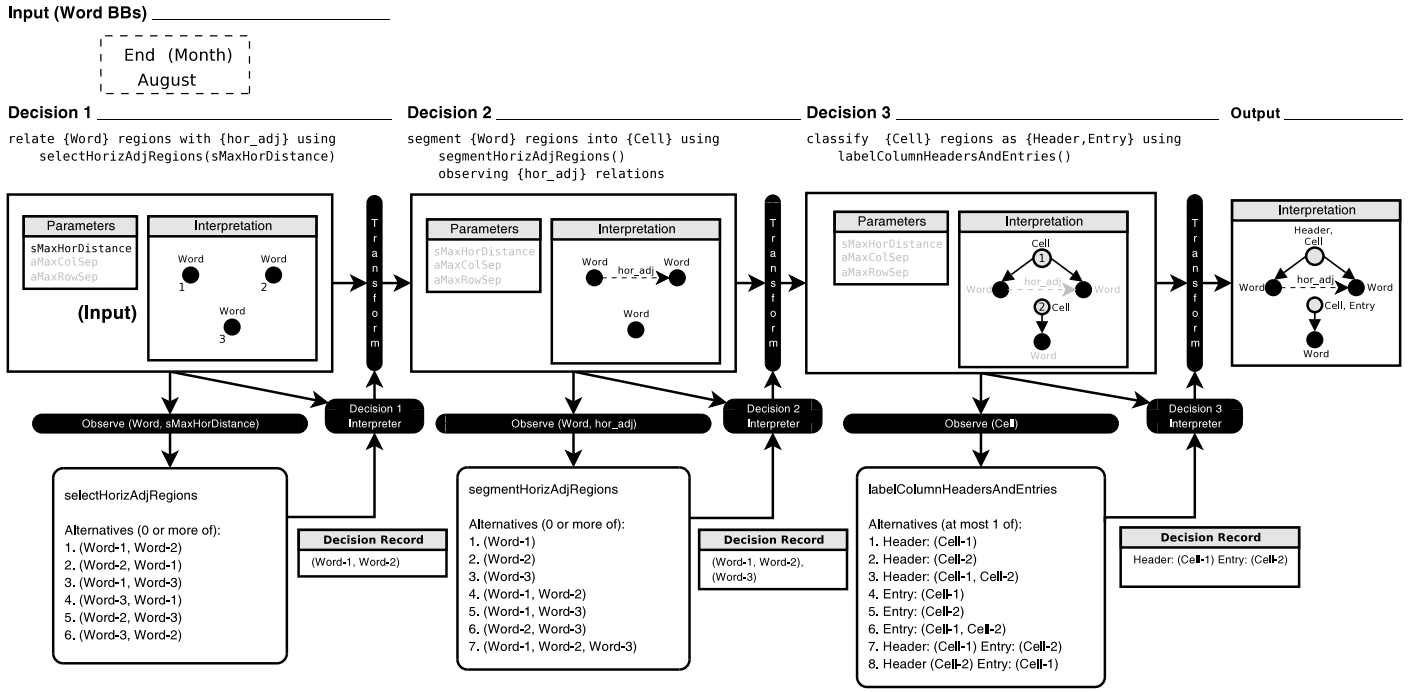


Fig. 5. Execution of the the First Three Decisions for the RSL Strategy Given in Figure 4. Three *Word* regions are provided in the input interpretation, which are 1) related by horizontal adjacency, 2) segmented into *Cell* regions, and 3) classified as *Header* or *Entry* (body) cells. For each decision, parameters and interpretation elements which are not visible to the external decision function are written in light gray. Each external decision function returns a text-based record. A decision interpreter validates chosen alternatives before they are used to transform the current interpretation to produce the input interpretation for the next decision or the output

RSL specifications for the two algorithms are available [34]; the Handley algorithm is specified in roughly 540 lines of RSL, and the Hu algorithm in about 240 lines. These lengths include significant commenting, but not code for the external decision functions. The external functions were implemented in roughly 5000 lines of TXL for the Handley algorithm, and roughly 3000 lines of TXL for the Hu algorithm. Small changes were made to these RSL strategies in order to produce the results shown later in this chapter, in particular renaming common region types to make them more explicit. Some bugs in the implementation of the external decision functions for the Handley algorithm were also corrected, and the performance results provided later in this chapter for the Handley algorithm are better than those reported earlier [34, 35] as a result.

4 Static Analysis of RSL Specifications

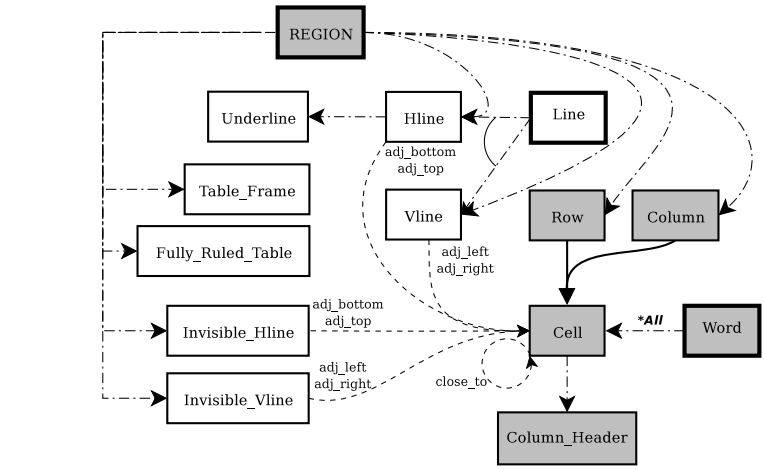
Useful insights can be gained from examining static dependencies within an RSL specification, and by comparing static dependencies between algorithms. In this section, we illustrate this process using Figure 6, which provides table model summaries for the Handley and Hu algorithms. First we describe how these table model summaries are derived from a static analysis of the RSL specifications. Next we discuss insights obtained from examining Figure 6, regarding comparisons and contrasts between the Handley and Hu algorithms. This discussion is based on static analysis; dynamic aspects of the algorithms are treated in Section 5.

4.1 Construction of Table Model Summaries

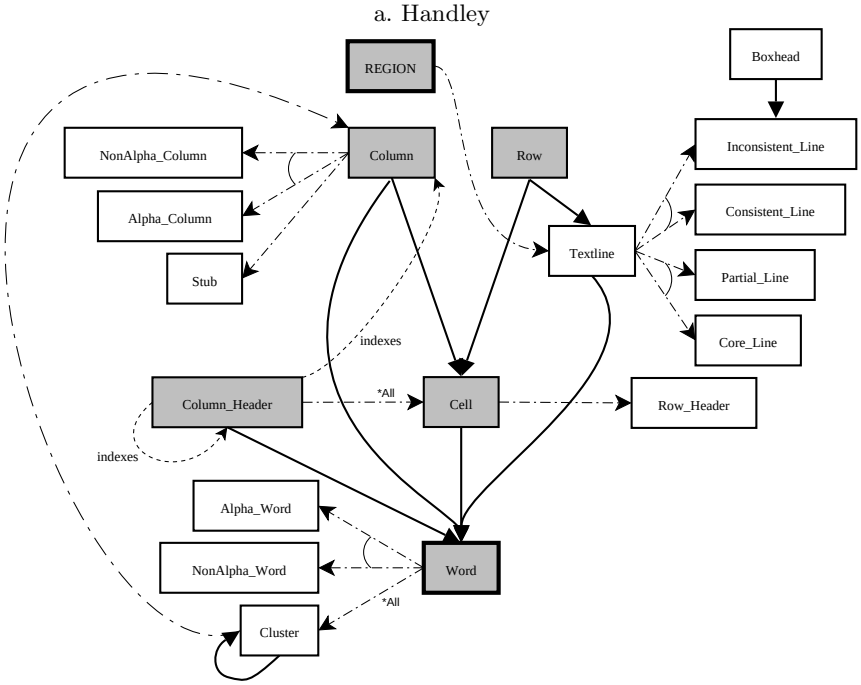
RSL specifications consist of a sequence of decision operations. Each decision operation has an associated set of regions and relation types, a decision function, and decision parameters. Each region or relation type T that may be altered by an RSL decision operator R has four dependencies:

1. On region types used to define the output model operations for R. These are the scope types provided as input to a decision (see Section 3.2)
2. On observed region or relation types (these follow the `observing` keyword)
3. On the decision function used for R
4. On parameters used by the decision function for R

These per-decision dependencies may be used to construct data dependency graphs. These graphs describe how inputs are used to produce outputs, and are commonly used in software engineering and analysis [36]. Once a data



a. Handley



b. Hu et al.

Fig. 6. . Table Model Summaries for Two Algorithms: (a) Handley and (b) Hu et al. Boxes represent region types, with thick borders indicating input types (*Word, Line*, and *REGION*). Gray boxes show types common to both algorithms. Labeled dashed lines are relations between region types. Solid lines represent segmentation operations, and dash-dotted lines represent classification operations

dependency graph has been constructed, additional analyses of an RSL specification can be made. For example, we can automatically determine the set of decision operations and types associated with a particular decision function. As another example, we can determine which region and relation types may be affected by a given decision parameter. These analyses are analogous to the techniques of backward and forward *program slicing* [37], respectively (please note that slicing requires information about decision sequences not shown in Figure 6).

The summaries of table model structure shown in Figure 6 are simple data dependency graphs which are produced as follows. We compute only dependencies of type 1: dependencies between output types and scope types that define the alternative outcomes. Then we filter all operations that merge and reject regions and relations. The resulting graph summarizes the table model in terms of relationships between scope types and output types. Figure 6 does not represent reject and merge operations because these only modify interpretations within a single type, either removing or combining elements of that type, and we wish primarily to represent relationships between types. Alternative summaries that incorporate merge and reject operations are of course possible.

4.2 Discussion of Table Model Summaries

In Figure 6, boxes represent region types. The three input types (*Word*, *Line*, and *REGION*) are shown in boxes with thick borders. Region types common to both algorithms are shown in gray boxes. Relations between region types are represented using labeled dashed lines (e.g. *indexes* for the Hu algorithm).

The three basic inference types used in RSL are represented using different arrow types: segmentation operations are represented by solid lines, classification by dash-dotted lines, and relations by labeled dashed lines. Segmentation and relation dependencies are drawn as arrows from output types to the scope types on which they depend. For ease of reading, we have reversed the arrow direction for classification operations; arrows representing classification are drawn from scope types (those to be classified) to output types.

To indicate where classifications have more than one output class, connecting arcs are used. For example, in the Handley algorithm, *Line* regions may be (exclusively) classified as horizontal (*Hline*) or vertical (*Vline*) lines, or neither (see Section 3). The annotation **All* is used to indicate labelings, trivial classifications where all regions of the input type have been labeled as the output type. For example, for at least one decision all *Word* regions are labeled as *Cell* regions in the Handley algorithm, and *Cluster* regions in the Hu algorithm. Though neither implemented strategy does so, arcs could also be used to represent segmentation operations which combine multiple region types into a segment (region) type.

The type *REGION* includes the set of all input regions expressible in RSL. Currently this is all bounding boxes and polylines expressible within the input

image, as defined by the set V (see Section 3.1). Elements of *REGION* are not promoted directly to a model region type unless a `create` or `replace` operation is used. In the Handley algorithm, many input regions are directly promoted to various types after geometric analyses (e.g. after projecting cells and finding minima in histograms, to define rows and columns). In the Hu algorithm, only *Textline* regions are produced by directly classifying input regions, in the preprocessing step that we added to the algorithm.

The graphs shown in Figure 6 can be interpreted similarly to semantic networks [38]. Segmentation edges correspond roughly to 'has-a' edges, and classification edges correspond roughly to 'is-a' edges, with the remaining edges defining other binary relationships (e.g. adjacency). Unlike a semantic net, non-binary relationships are represented in the graph, using and-or relationships. In this way, each unique set of relationships between scope and output types are represented separately, as an 'or' of 'ands'.

To illustrate the information that can be read directly from Figure 6, consider the Textline regions in the Hu algorithm. The graph edges connecting to the Textline box in Figure 6b tell us the following:

1. *Textline* regions may be segmented into *Row* regions
2. *Word* regions may be segmented into *Textline* regions
3. Image *REGION*s may be classified as a *Textline* region
4. A *Textline* region may be classified as either an *Inconsistent_Line* or *Consistent_Line*, or neither
5. A *Textline* region may be classified as either a *Partial_Line* or *Core_Line*, or neither

Despite their simplicity, these table model summaries provide useful information for analyzing the implemented algorithms. First we discuss the region types which are common and unique to each algorithm. Both algorithms utilize *Word, Cell, Row, Column,* and *Column_Header* regions. However, the *Handley* algorithm takes lines (underlines and ruling lines in the table) into account, and defines spatial relationships that are not used in the *Hu* algorithm. The Hu algorithm on the other hand makes greater use of classification operations, particularly for *Column, Textline,* and *Word* regions. The Hu algorithm also explicitly defines *Boxhead* and *Stub* regions, which the Handley algorithm does not.

Figure 6 also shows interesting differences between the relationships that occur among the common regions. In the Handley algorithm, *Cell* regions are classified as *Column Header* regions, while at some point in the Hu algorithm, all *Column Header* regions are classified as *Cell*s. In the Handley algorithm, *Column* and *Row* regions contain *Cell*s. In contrast, the Hu algorithm composes *Column* and *Row* regions as follows: *Column* regions contain either *Cell* or *Word* regions (but not both), whereas *Row* regions contain either *Cell* or *Textline* regions, but not *Word* regions. The Hu algorithm defines an indexing relation from column headers to *Column*s of headers, while the Handley

algorithm has no representation of indexing structure (as the algorithm was not designed to address that problem).

This simple table model summary provides a useful course-grained view of similarities and differences between the table models used by these two algorithms. The relationships provided in the table model summaries are also useful when debugging and during evaluation, as we will see in the next section.

5 Evaluating the Accuracy of Decisions

The three main criteria for evaluating a recognition algorithm are accuracy, speed, and storage requirements. Here, we concern ourselves solely with accuracy but we acknowledge that speed and storage requirements are nearly as important for real-world applications (e.g. for on-line interactive applications), and that some form of trade-off often needs to be made between these three criteria.

Evaluating the accuracy of an algorithm means assessing the ability of the algorithm to produce interpretations that meet the requirements of a given task [10]. This is normally done using test data, for which the ground-truth is known (significant difficulties in defining ground truth for tables have been discussed in the literature [39, 40]). Normally evaluation of recognition algorithms focuses on comparing the final interpretations accepted by algorithms [41, 3, 42, 43, 44, 45].

In this section we use the decision-based approach to evaluate accuracy of recognition by considering the decision process used to produce results. We discuss characterizing individual decisions made by a recognition algorithm as good or bad (Section 5.1), and we augment the traditional measures of recall and precision with the new measures of *historical recall* and *historical precision* (see Section 5.2).

5.1 Evaluating the Accuracy of Individual Decisions

Our goal is to measure the accuracy of individual recognition decisions, and the accuracy of sequences of recognition decisions. This detailed information is useful for planning improvements to a recognition algorithm, and provides a basis for learning algorithms which seek to automatically improve recognition performance.

In our evaluation of decision accuracy, we are concerned only with decisions which affect the comparison with ground truth. It is common for algorithms to hypothesize many objects and relationships that aren't part of the interpretation space used for evaluation. For example, the algorithm comparisons we report in Section 6 use a ground truth that defines the location of cells, but the ground truth does not identify which subset of cells are header cells. The reason we did not record header cells in our ground truth is that one

of the algorithms does not aim to identify all header cells. The ground truth, and the comparisons based on ground truth, must be restricted to objects and relationships that are identified by all the algorithms.

In order to define what constitutes a 'good' decision, we first define *correct*, *complete*, and *perfect* decisions. Each decision to apply an interpretation model operation results in asserting and/or rejecting hypotheses. In RSL, each decision record produced by a decision function corresponds to a set of model operations to apply to the current interpretation, chosen from a space of alternative model operations (see Figure 5). Examples for decision types are provided in Section 6.3.

A decision is *correct* if all operations selected generate only valid hypotheses and/or reject only invalid hypotheses. A decision is *complete* if it selects the set of all correct operations in the set of alternatives that *alter* the interpretation (e.g. re-classifying a *Word* as a *Cell* is no more complete than not selecting this redundant operation). A *perfect* decision is both correct and complete; all selected alternatives are correct, and all correct alternatives are in the set. For the case where no alternatives are selected, this is either a perfect decision (when all alternatives are incorrect), or an incomplete decision (when correct alternatives exist).

Using these definitions, 'good' decisions lie somewhere between perfect decisions and those that are totally incorrect. One could characterize a decision using recall and precision metrics (see the next subsection), to give the proportion of correct alternatives selected, and the proportion of selected alternatives that are correct. These metrics could then be thresholded, or used to assign fuzzy membership values for the set of 'good' decisions. More informally, one might consider any decision that is perfect, correct but incomplete, or complete and mostly correct to be a 'good' one.

We could characterize a sequence of decisions (at those decisions for which evaluation information exists) similarly, in terms of distributions of recall and precision for selecting valid model operations. The metrics might also be weighted based on the location of valid operations within the sequence of decision outcomes, to weight earlier decisions more heavily for example.

These are internal performance measures which characterize decisions based on their associated alternative outcomes. While we will only touch on these briefly here, we believe that these may provide a basis for devising table recognition algorithms based on game-theoretic principles such as mini-max optimization [46, 10].

5.2 Historical Recall and Precision

The traditional detection metrics recall and precision measure how similar an algorithm's final interpretation is to the ground truth interpretation. Here we introduce historical versions of these measures [35]. Informally stated, the historical measures give an algorithm credit for correct hypotheses that

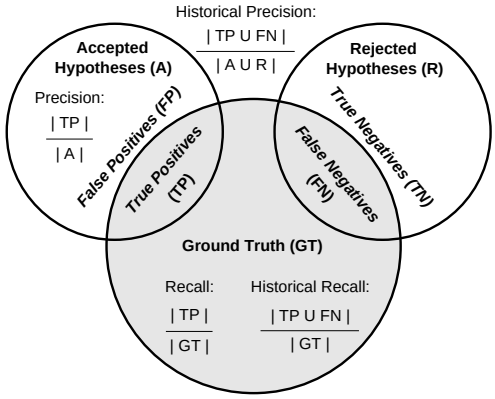


Fig. 7. Recall, Precision, Historical Recall, and Historical Precision

it made somewhere along the way, even if the algorithm later rejected these hypotheses. The historical measures can be evaluated at any point during algorithm execution; this information provides valuable insight into the algorithm's treatment of hypothesis generation and rejection.

Figure 7 illustrates sets of hypotheses and assertions used in our discussion. At a given point in time, the set of generated hypotheses produced by an algorithm (e.g. cell locations) is defined by the union of accepted (A) and rejected (R) hypotheses. We assume that at any given time, every hypothesis is either accepted or rejected, but not both. The validity of individual hypotheses within A and R is determined using GT, a set of ground truth declarations which are taken to be valid (e.g. a set of cell locations taken to be valid). The set of true positives (TP) is defined by the intersection of accepted hypotheses and ground truth ($A \cap GT$). Similarly, the set of false negatives (FN), which consists of ground truth elements that have been proposed and rejected, is defined by the intersection of rejected and ground truth elements ($R \cap GT$).

Also shown in Figure 7 are recall and precision metrics, which describe the ratio of true positives to recognition targets ($|TP|/|GT|$) and accepted hypotheses ($|TP|/|A|$), respectively. *Historical* recall and precision describe the recall and precision of the set of generated hypotheses ($A \cup R$). Together, the true positives and false negatives comprise the set of ground truth elements that have been generated ($TP \cup FN$). Historical recall is the proportion of ground truth hypotheses that have been generated ($|TP \cup FN|/|GT|$), while historical precision is the proportion of generated hypotheses that match ground truth ($|TP \cup FN|/|A \cup R|$). Note that if no hypotheses are rejected (i.e. $R = \{\}$), then the 'conventional' and historical versions of recall and precision are the same. The key difference here is that the historical metrics take rejected hypotheses into account, while the conventional ones do not. For an example of this, compare Figure 11a to Figure 11b; in Figure 11b, cell hypotheses are never rejected.

Conventional and historical recall can be directly compared, as they both describe coverage of the set of ground truth elements. Note that historical recall will always be greater than or equal to recall (refer again to Figure 11). Also, historical recall never decreases during a recognition algorithm's progress, while recall may increase or decrease at any point. The difference between historical and conventional recall is the proportion of recognition targets that have been falsely rejected ($|FN|/|GT|$).

It is harder to relate conventional and historical precision. Precision measures the accuracy of what is accepted as valid, while historical precision measures the accuracy (or *efficiency*) of hypothesis generation. Put another way, historical precision quantifies the accuracy of hypotheses that the algorithm generates and accepts at some point.

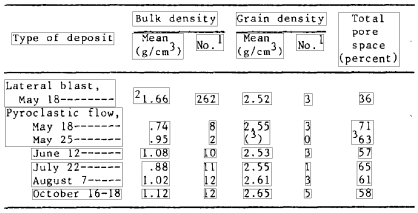
6 Decision-Based Comparison of Algorithms in RSL

In this section we will compare the recognition accuracy of our RSL implementations of the Handley and Hu et al. table structure recognition algorithms. We first consider a conventional results-based evaluation, in which the complete sequence of recognition decisions are evaluated as a whole, without reference to rejected hypotheses. We then contrast this with a decision-based comparison, in which the effects of individual decisions may be observed, and rejected hypotheses are taken into account. Using this information, we then design a new strategy which combines the observed strengths of the two algorithms, to produce a better final result. All metrics presented here are in terms of 'external' accuracy, i.e. we compare the state of the interpretation to ground truth after each decision affecting the hypothesis types in question.

Our goal here is not to evaluate these two algorithms in any real sense, but to illustrate decision-based comparisons of algorithms. We will consider only results for a single, reasonably challenging table as input, on which we will try (informally) to optimize recognition. For a real-world application we would train these algorithms by optimizing performance metrics such as conventional and historical recall and precision over a representative sample of the set of tables that we wish to recognize.

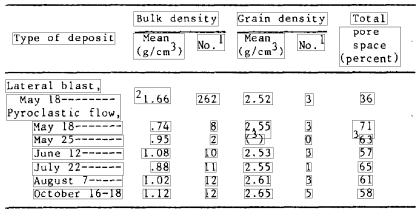
Input to both algorithms is a set of *Word* regions with an associated text attribute (set to 'a' for words containing mostly alphabetic characters, and '1' otherwise), and a set of *Line* regions (as seen in Figure 1b). All words in cells were provided as input; any words not within the table (e.g. the table title and footnotes) were not provided.

As can be seen in Figure 6, the Hu algorithm does not pay any attention to the *Line* regions, and leaves them in the produced interpretation untouched. As the classification decisions shown in Figure 6 suggest, the Hu algorithm makes use of the text attribute associated with words; the Handley algorithm makes its analysis based on region geometry and topology alone, and ignores these attributes.



Type of deposit	Bulk density Mean (g/cm³)	Bulk density No.[1]	Grain density Mean (g/cm³)	Grain density No.[1]	Total pore space (percent)
Lateral blast, May 18	[2]1.66	262	2.52	3	36
Pyroclastic flow,					
May 18	.74	8	2.55	3	71
May 25	.95	2	(3)	0	[3]63
June 12	1.08	10	2.53	3	57
July 22	.88	11	2.55	1	65
August 7	1.02	12	2.61	3	61
October 16-18	1.12	12	2.65	5	58

a. Handley Cells



Type of deposit	Bulk density Mean (g/cm³)	Bulk density No.[1]	Grain density Mean (g/cm³)	Grain density No.[1]	Total pore space (percent)
Lateral blast, May 18	[2]1.66	262	2.52	3	36
Pyroclastic flow,					
May 18	.74	8	2.55	3	71
May 25	.95	2	(3)	0	[3]63
June 12	1.08	10	2.53	3	57
July 22	.88	11	2.55	1	65
August 7	1.02	12	2.61	3	61
October 16-18	1.12	12	2.65	5	58

b. Hu et al. Cells

	Handley	Hu et al.	Common
TP	25	45	24
FP	13	7	0
S-GT	0	3	0
M-GT	27	4	4
SM-GT	0	0	0
O-GT	0	0	0
FA	0	0	0
Recall	48.1%	86.5%	
Precision	65.8%	86.5%	

c. Cell Hypothesis Sets and Metrics

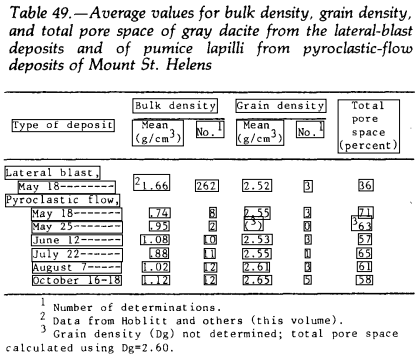


Table 49.—Average values for bulk density, grain density, and total pore space of gray dacite from the lateral-blast deposits and of pumice lapilli from pyroclastic-flow deposits of Mount St. Helens

Type of deposit	Bulk density Mean (g/cm³)	Bulk density No.[1]	Grain density Mean (g/cm³)	Grain density No.[1]	Total pore space (percent)
Lateral blast, May 18	[2]1.66	262	2.52	3	36
Pyroclastic flow,					
May 18	.74	8	2.55	3	71
May 25	.95	2	(3)	0	[3]63
June 12	1.08	10	2.53	3	57
July 22	.88	11	2.55	1	65
August 7	1.02	12	2.61	3	61
October 16-18	1.12	12	2.65	5	58

[1] Number of determinations.
[2] Data from Hoblitt and others (this volume).
[3] Grain density (Dg) not determined; total pore space calculated using Dg=2.60.

d. Ground Truth Cells

Fig. 8. Cell Output for UW Database Table (Page a038). Shown in (c) are the cell hypothesis set sizes (*True Positives (TP), False Positives (FP), Split Ground Truth (S-GT), Merged Ground Truth (M-GT), Split-and-Merged Ground Truth (SM-GT), Missing Ground Truth (O-GT), and False Alarms (FA)*) and recall and precision metrics. For each hypothesis set type, the size of the intersection of the Handley and Hu sets is shown in the *Common* column

6.1 Conventional Evaluation

Normally in the table recognition literature when comparing two algorithms, we consider only the final interpretations, for example as shown in Figure 8. Shown are the final cell hypotheses for both algorithms, along with the ground truth interpretation they were evaluated against. Also shown are the sets of true positive, false positive, and errors along with the resulting recall and precision metrics. No errors of omission (cells whose words are entirely missed) or 'false alarms' (cells whose contents do not belong to any ground truth cells) are made, because we provide all words in cells, and only words in cells in the input. There are also no 'spurious' cells (splitting and merging of ground truth producing many-to-many matches with ground truth cells). The approach to error analysis we are using here is based on that of Liang [44]. However, we are presenting errors in terms of ground truth cells here;

for example, the Hu algorithm produces seven false positives, which incorrectly split three ground truth cells and merge four. The Handley algorithm produces thirteen false positives, which incorrectly merge 27 cells.

As can be seen at a glance, both algorithms assign words to the appropriate column. It is the decisions which assign words and cells to rows that have produced the errors in the final interpretations.

In the rightmost column of Figure 8c, we've shown the size of the intersection of each cell hypothesis set type. 24 of the 25 true positives proposed by the Handley algorithm are also proposed by the Hu algorithm; the remaining cell is the 'Total pore space (percent)' column header (see Figure 8d). In Figure 8c we can see that the false positives proposed by the two algorithms are disjoint; the algorithms make different errors.

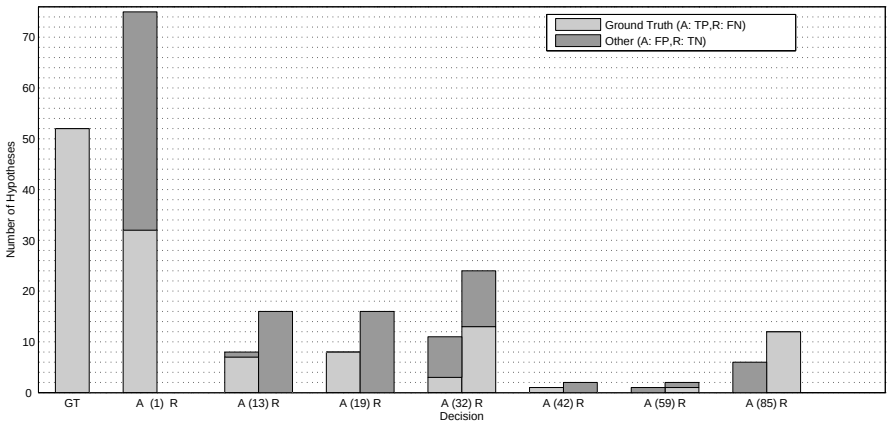
Figure 8c also describes how ground truth cells are mis-recognized by the two algorithms (i.e. what specific errors are associated with the false positives). All four of the cells incorrectly merged with words from other cells by the Hu algorithm are also over-merged by the Handley algorithm ($M - GT$). Looking at Figures 8a, 8b, and 8d, two of these over-merged cells are located in the leftmost column, and two are located in the table body in the fourth and sixth columns, near the superscripted threes (^{3}). All remaining errors for the Handley algorithm result from merging cells across rows of the table ($M - GT$), while the Hu algorithm also splits ($S - GT$) two cells near the superscripted threes, and splits the rightmost column header ('Total pore space (percent)', as mentioned earlier).

Looking at the recall and precision metrics, the Hu algorithm has higher values for both. From this and our prior analysis, it appears that the Hu algorithm performs better recognition of the cells in this particular table (i.e. makes better decisions) than the Handley algorithm.

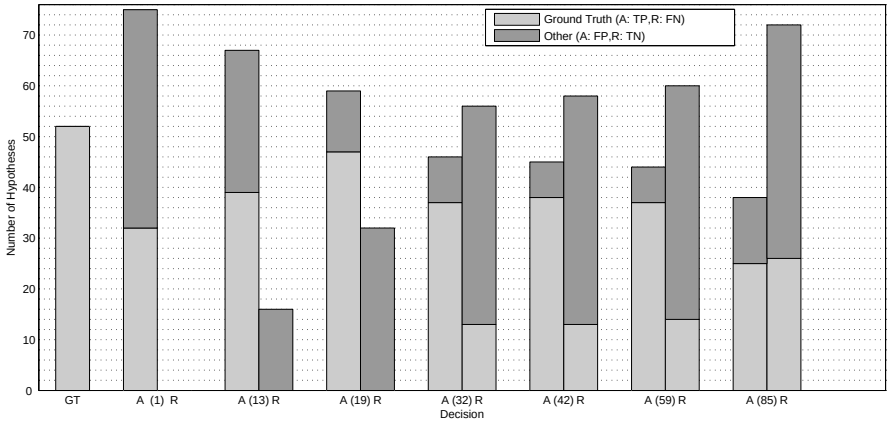
6.2 Metrics for Individual Decisions

We now have a reasonably detailed comparison of the outputs of the two algorithms. We also have determined which types of decisions might be studied more closely in order to improve recognition of the table shown in Figure 8: those that affect cells and rows. So our next question is this: which are the decisions that need to be improved, and where are they within our algorithm implementations?

Currently in common practice, answering this question would be dealt with in an ad-hoc way, often by outputting intermediate interpretations at various points, evaluating them, and then trying to determine which parts of the algorithm implementation do not generate or filter cell and row hypotheses as needed. This often produces informal and partial error analyses. We believe that the flaws in this process are in large part due to the effort required to detect errors in decision-making when decisions invoking model operations are not distinguished within the syntax of the implementation language, nor in the output interpretations.



a. Changes in Cell Hypotheses



b. Cumulative Cell Hypotheses

Decisions

1 All words classified as cells
13 Merge cells with little horizontal separation which overlap vertically by roughly more than half the height of the taller bounding box
19 Merge cells within columns that overlap vertically by roughly more than half the height of the taller bounding box
32 Merge cells sharing column and row assignments
42 'Total pore space (percent)' header cell detected in boxhead
59 Merge cell in leftmost column alone in its row with the cell below ('Pryoclastic flow,')
85 Merge cells sharing estimated line and whitespace separators for rows and columns

Fig. 9. Detection of Cells Shown in Figure 1 by the Handley Algorithm. *A* represents accepted hypotheses, and *R* represents rejected hypotheses

This is where the benefits of decision-based specification become most apparent. The syntax of a decision-based specification language such as RSL explicitly represents decisions to invoke model operations; algorithms are specified as sequences of possible model operation applications. Further, the outputs of a decision-based language contain the entire history of model operation applications, including *which decision(s) selected them*. Hypotheses are uniquely identified, and their complete history of creation, rejection, and reinstatement are available in the output. Detecting decisions that cause errors and reverting the output to intermediate states is carried out using simple filtering and transformation of the output interpretation, rather than requiring guesswork.

Changes in accepted cell hypotheses for the Handley algorithm are shown in Figure 1, with each change indexed by the decision which produced it (each corresponds to a decision operation in the RSL specification). Decisions concerning cells which had no effect on the interpretation are not shown.

Figure 9 provides additional information about each decision in the Handley algorithm that changed the cell hypotheses. At the bottom of the figure a brief summary for each of these decisions is provided. Figure 9a illustrates the changes made by each decision to the accepted and rejected sets of cell hypotheses. These are shown as the number of newly proposed or rejected cells that are accepted by each decision, and the number of accepted hypotheses which are rejected by each decision. On the far left of the graph the number of cells in ground truth is shown for comparison. For each decision we show the decision number (in parentheses), and the number of cell hypotheses added to the accepted (A) and rejected (R) sets. Ground truth hypotheses are shown in light gray (true positive for A, false negative for R), and other hypotheses are shown in dark gray (false positive for A, true negative for R).

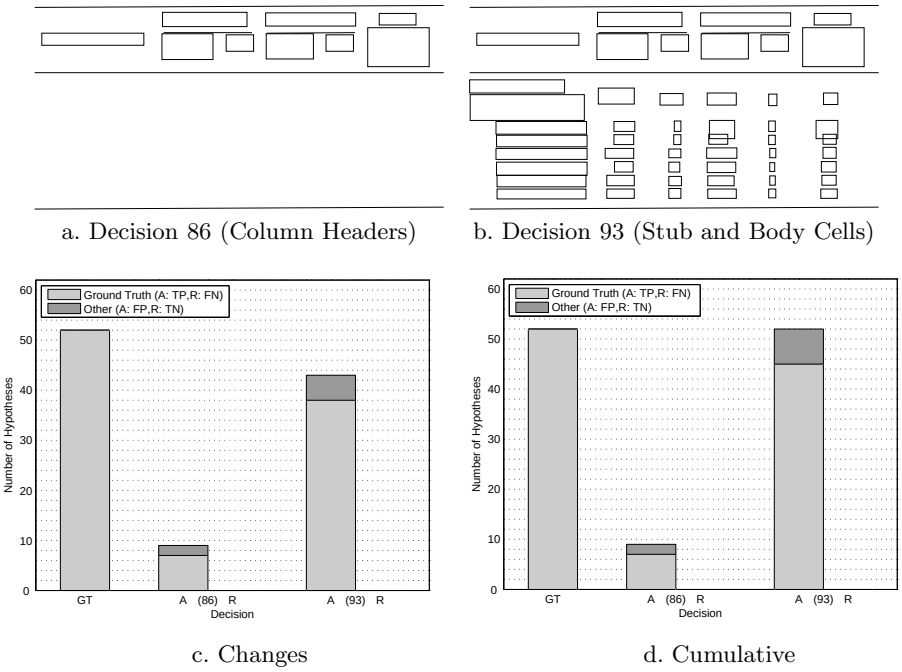
Figure 9b illustrates the sets of accepted and rejected hypotheses after each decision, i.e. the cumulative effects of the changes shown in Figure 9a. Again, light gray is used to illustrate the number of accepted and rejected hypotheses that match ground truth.

The cells produced by decisions in the Hu algorithm are shown in Figures 10a and 10b. Unlike the Handley algorithm, the Hu algorithm revises cell hypotheses only twice for our example table. This is because the Hu algorithm detects rows and columns before cells, and detects boxhead and stub head cells before body and stub cells.

Similar to Figure 9a, Figure 10c illustrates changes in the sets of accepted and rejected cell hypotheses for the Hu algorithm, while Figure 10d illustrates the cumulative effect of the decisions (similar to Figure 9b). Because the Hu algorithm never rejects cell hypotheses, R is always empty.

6.3 Evaluation and Error Analysis for Individual Decisions

Let us briefly try to characterize which are the 'good' decisions shown in Figures 9 and 10. As discussed in Section 5, an *internal* evaluation of decisions



a. Decision 86 (Column Headers)

b. Decision 93 (Stub and Body Cells)

c. Changes

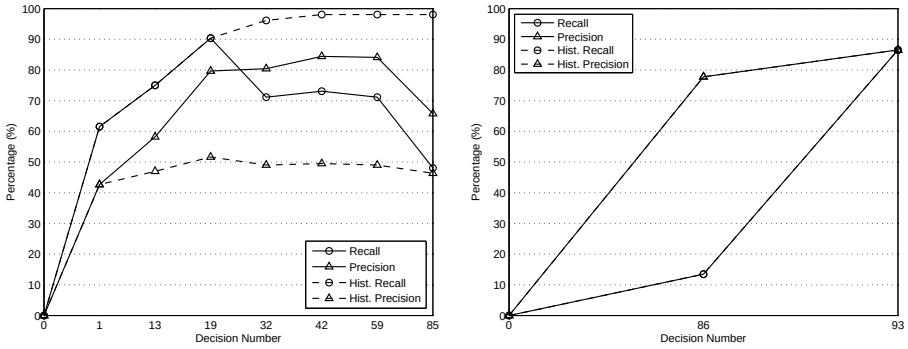
d. Cumulative

Fig. 10. Cell Detection by Hu et al. Over Time. *A* represents accepted hypotheses, and *R* represents rejected hypotheses

is with respect to their associated alternative outcomes, while the type of metrics presented in Figures 9 and 10 are *external* evaluations, which compare interpretations after each decision to goals (i.e. ground truth).

In terms of an internal evaluation, the Handley decision at time 42 is a *perfect* decision which is both correct and complete; that is to say, of all the possible cell merges, only the single valid alternative is selected. All other cells used to define the possible merges are already correct or over-merged cells (as can be seen from the cells shown at time 32 in Figure 1). The Handley decision at time 19 is not a perfect decision, but we will claim that it is quite good, as it is entirely correct (only valid merges are selected), though incomplete: some valid merges within the set of alternatives are not selected, e.g. for some of the column headers.

Both internally and externally, the remaining decisions made by the two algorithms incorrectly merge cells, and lie somewhere between fairly good, with a small number of errors (e.g. the Hu decisions, Handley time 13), and detrimental, producing only errors, as at Handley time 85, when all assertions and rejections are false. Looking at time 85 in Figure 9a, in the left bar we see that all cells accepted by the decision are false positives (dark shading indicates cells not in ground truth), and in the the right bar all cells rejected



a. Handley b. Hu et al.

Fig. 11. Performance Metrics for Cell Hypotheses

by the decision are false negatives (light shading indicates cells in ground truth).

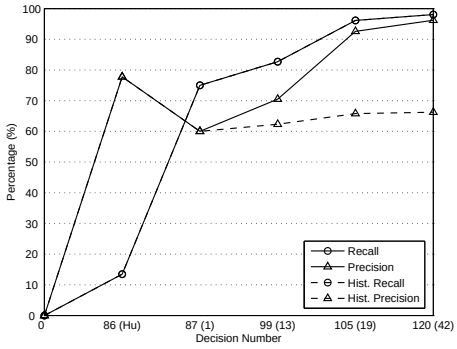
Now let us identify which decisions in our RSL implementations caused the errors in the final interpretations that we observed in Section 6.1. The Hu algorithm splits a column header cell at Decision 86, and then proposes the cells that have been incorrectly merged and split across rows in the stub and body at Decision 93. We should point out that the errors related to superscripts are caused by our projection-based textline detection addition to the Hu algorithm, which does not take super and sub-scripts into account; the original algorithm was designed for text files, which of course come with textlines already defined. The Handley algorithm completes recognizing the stub head and boxhead cells correctly, using a series of decisions ending at Decision 42, but over-merges cells across rows in the stub and body at Decisions 32, 59, and 85.

These errors may be determined automatically by searching the hypothesis history for false positive cell hypotheses recorded in the output interpretation. Here these errors may be found simply by looking at Figures 1 and 10.

6.4 Evaluating and Improving the Decision-Making Process

In addition to this hypothesis-level view, we can also use historical recall and precision along with conventional recall and precision to give us an external, higher-level view of the decision-making process. Figure 11 presents all four of these metrics for each decision shown in Figures 1 and 10.

Most strikingly, note that our implementation of the Handley algorithm has higher recall after Decision 19 than the recall of the Hu algorithm at any point; if the algorithm had stopped at this point, it would have fared better



a. Performance Metrics

Type of deposit	Bulk density Mean (g/cm^3)	Bulk density No.	Grain density Mean (g/cm^3)	Grain density No.	Total pore space (percent)
Lateral blast,					
May 18-------	^{2}1.66	262	2.52	3	36
Pyroclastic flow,					
May 18-------	.74	8	2.55	3	71
May 25-------	.95	2	(^{3})	0	^{3}63
June 12------	1.08	10	2.53	3	57
July 22------	.88	11	2.55	1	65
August 7-----	1.02	12	2.61	3	61
October 16-18	1.12	12	2.65	5	58

b. Final Result

Fig. 12. Results for Combining Decisions of the Hu et al. and Handley Algorithms. First the Hu algorithm is applied, stopping after its first cell decision (for boxhead and stub head cells). The original decision numbers for the Handley algorithm are shown in brackets in (a). In the output, there are only two false positives, which split one ground truth cell in the rightmost column (^{3}63)

in the conventional evaluation given in Section 6.1. At Decision 19 for the Handley algorithm, the stub and body cells have all been correctly detected with the exception of the cell with a prefix superscript (3) in the last column of the table. Only five cells have not been correctly located: the incorrect body cell, and four of the header cells (these correspond to the twelve false positives at Decision 19, shown as the gray portion of the bar for accepted hypothesis set in Figure 9b).

After Decision 19, the Handley algorithm recall and precision measures start to decrease, and false negatives start being created. At Decision 32, thirteen valid cells are rejected; however, three new ground truth cells (column headers) are produced, causing an increase in historical recall. As mentioned earlier, Decision 42 is a perfect decision, and detects the rightmost column header (increasing all metrics). The final two decisions decrease recall, precision, and historical precision, as they propose invalid merges. The historical precision for the Handley algorithm is uniformly low, partly because of proposing all words as cells initially. This results in many invalid hypotheses being generated. However, the historical recall is very high, and in fact only one ground truth cell is never generated and considered: the '^{3}63' cell in the rightmost column of the body.

The Hu algorithm decisions have fairly high precision, and high final recall (at Decision 93). However, the historical recall is considerably lower than that of the Handley algorithm (it does not generate the ground truth cell missing from the Handley algorithm's hypothesis set, along with other cells). Note

that the historical and conventional metrics are identical for the Hu algorithm, because no cell hypotheses are ever rejected.

Can we combine the existing decisions of the Handley and Hu algorithms to produce a result better than either algorithm in isolation? The answer is yes, and the result for one such combination is shown in Figure 12. The combination is produced by first running the Hu algorithm, stopping it after it finishes identifying cells in the boxhead (Decision 86). In the output, we filter all but *Word*, *Line*, and *Cell* region types. We then run the Handley algorithm on the filtered output of the Hu algorithm, after making the following changes to the Handley algorithm:

Decision 1: only words that do not already belong to a cell are classified as cells (this preserves the detected header cells)

Decisions 32, 59, 86: are removed along with their supporting decision sequences (e.g. estimates of row and column structure)

No decision function parameters were altered.

In the combined strategy all the ground truth cells generated by the Handley algorithm are detected, with identical historical and conventional recall after each decision (one cell is still mis-recognized, as for the original algorithms). At the final time, precision is higher than it is for either of the individual algorithms, while the historical precision has been improved relative to the original Handley algorithm. While effective, this particular decision sequence is likely over-fit to this particular table.

7 Conclusion

We have presented decision-based methods for specifying and comparing recognition algorithms. The Recognition Strategy Language (RSL) is a first attempt at formalizing recognition algorithms as sequences of decisions which select among alternative model operations at run-time. Decision-based specifications make a table recognition algorithm's search within the space of possible interpretations defined by a model explicit, and allow more information about an algorithms' decision process to be automatically captured both statically and at run time. We have illustrated the decision-based approach using RSL re-implementations of the Handley [1] and Hu et al. [12] table structure recognition algorithms, and demonstrated by example how we can use the additional information made available to improve recognition by combining decisions from both.

Decision-based specification makes it possible to directly determine which decisions in the recognition process affect a hypotheses. By capturing all hypotheses including those that are rejected, we can measure new metrics on the set of hypotheses generated by an algorithm. We have presented two simple but useful metrics that characterize generated hypotheses in terms of coverage of the set of recognition goals (historical recall) and the accuracy/efficiency

of hypothesis generation (historical precision). We believe that other useful metrics may be defined, such as some measure of 'fickleness' that characterizes how frequently hypotheses shift from between the sets of accepted and rejected hypotheses.

In the future we wish to extend RSL to better support feature computations directly within an RSL program; currently all feature computations are defined externally within the functions that make decisions at run-time. To further support studying the combination of recognition strategies, we also wish to explore new decision types that combine and select among decisions and decision sequences within RSL programs. This would allow decision combinations to be produced automatically or semi-automatically, rather than manually as we did in Section 6. Ultimately we would like to use decision-based languages such as RSL for studying the problem of learning recognition strategies [13, 14, 15].

Acknowledgement. We wish to acknowledge Dr. Ching Suen, who provided resources used by the first author to write this chapter while at the Centre for Pattern Recognition and Machine Intelligence (Concordia University, Montreal, Canada), and Joshua Zimler who helped with the design of some figures. This research was supported by the Natural Sciences and Engineering Research Council of Canada.

References

1. Handley, J.: Table analysis for multi-line cell identification. In: Proc. Document Recognition and Retrieval VIII (IS&T/SPIE Electronic Imaging). Volume 4307., San Jose, CA (2001) 34–43
2. Phillips, I., Chen, S., Haralick, R.: CD-ROM document database standard. In: Proc. Second Int'l Conf. Document Analysis and Recognition, Tsukuba Science City, Japan (1993) 478–483
3. Silva, A.e., Jorge, A., Torgo, L.: Design of an end-to-end method to extract information from tables. International Journal on Document Analysis and Recognition **8** (2006) 144–171
4. Embley, D., Hurst, M., Lopresti, D., Nagy, G.: Table-processing paradigms: a research survey. International Journal on Document Analysis and Recognition **8** (2006) 66–86
5. Handley, J.: Document recognition. In: Electronic Imaging Technology. IS&T/SPIE Optical Engineering Press, Bellingham, WA (1999)
6. Lopresti, D., Nagy, G.: Automated table processing: An (opinionated) survey. In: Proc. Third Int'l Workshop on Graphics Recognition, Jaipur, India (1999) 109–134
7. Lopresti, D., Nagy, G.: A tabular survey of automated table processing. In: Lecture Notes in Computer Science. Volume 1941. Springer-Verlag, Berlin (2000) 93–120
8. Zanibbi, R., Blostein, D., Cordy, J.: A survey of table recognition: Models, observations, transformations, and inferences. Int'l J. Document Analysis and Recognition **7**(1) (2004) 1–16
9. Hurst, M.: Towards a theory of tables. International Journal on Document Analysis and Recognition **8** (2006) 123–131
10. Zanibbi, R., Blostein, D., Cordy, J.R.: Recognition tasks are imitation games. In: Lecture Notes in Computer Science. Volume 3686. (2005) 209–218
11. Zanibbi, R., Blostein, D., Cordy, J.R.: The recognition strategy language. In: Proc. Eighth Int'l Conf. Document Analysis and Recognition. (2005) 565–569 Vol. 2
12. Hu, J., Kashi, R., Lopresti, D., Wilfong, G.: Table structure recognition and its evaluation. In: Proc. Document Recognition and Retrieval VIII (IS&T/SPIE Electronic Imaging). Volume 4307., San Jose, CA (2001) 44–55
13. Bottoni, P., Mussio, P., Protti, M.: Metareasoning in the determination of image interpretation strategies. Pattern Recognition Letters **15** (1994) 177–190
14. Draper, B.: Learning control strategies for object recognition. In Ikeuchi, K., Veloso, M., eds.: Symbolic Visual Learning. Oxford Press, New York (1997) 49–76
15. Draper, B., Bins, J., Baek, K.: Adore: Adaptive object recognition. Videre **1**(4) (2000) 86–99 (online journal).
16. Haralick, R.: Document image understanding: Geometric and logical layout. In: Proc. Conf. Computer Vision and Pattern Recognition, Seattle, WA (1994) 385–390
17. Nagy, G.: Twenty years of document image analysis in PAMI. IEEE Trans. Pattern Analysis and Machine Intelligence **22**(1) (2000) 38–62
18. Hurst, M.: Layout and language: Challenges for table understanding on the web. In: Proc. First Int'l Workshop on Web Document Analysis, Seattle, WA (2001) 27–30

19. Wang, Y., Hu, J.: Detecting tables in HTML documents. In: Lecture Notes in Computer Science. Volume 2423., Berlin, Springer-Verlag (2002) 249–260
20. Daumé, H., Langford, J., Marcu, D.: Search-based structured prediction. (unpublished) (2006)
21. Daumeé, H., Marcu, D.: Learning as search optimization: Approximate large margin methods for structured prediction. In: Proc. International Conference on Machine Learning, Bonn (Germany) (2005) 169–176
22. Amano, A., Asada, N., Mukunoki, M., Aoyama, M.: Table form document analysis based on the document structure grammar. International Journal on Document Analysis and Recognition **8** (2006) 201–213
23. Coüasnon, B.: DMOS, a generic document recognition method: Application to table structure analysis in a general and in a specific way. International Journal on Document Analysis and Recognition **8** (2006) 111–122
24. Takasu, A., Satoh, S., Katsura, E.: A document understanding method for database construction of an electronic library. In: Proc. Twelfth Int'l Conf. Pattern Recognition, Jerusalem, Israel (1994) 463–466
25. Bagdanov, A.: Style Characterization of Machine Printed Texts. PhD thesis, University of Amsterdam (2004)
26. Wang, X.: Tabular Abstraction, Editing and Formatting. PhD thesis, University of Waterloo, Canada (1996)
27. Grossman, J., ed.: 12 (Tables). In: Chicago Manual of Style. 14th edn. University of Chicago Press (1993)
28. Hurst, M.: Layout and language: An efficient algorithm for detecting text blocks based on spatial and linguistic evidence. In: Proc. Document Recognition and Retrieval VIII (IS&T/SPIE Electronic Imaging). Volume 4307., San Jose, CA (2001) 56–67
29. Ousterhout, J.: Tcl and the Tk Toolkit. Addison-Wesley (1994)
30. van Melle, W., Shortliffe, E., Buchanan, B. In: EMYCIN , A Knowledge Engineer's Tool for Constructing Rule-Based Expert Systems. Addison-Wesley (1984) 302–328
31. Cordy, J.: The TXL source transformation language. Science of Computer Programming **61**(3) (2006) 190–210
32. Cordy, J., Dean, T.R. Malton, A., Schneider, K.: Source transformation in software engineering using the TXL transformation system. Journal of Information and Software Technology **44**(13) (2002) 827–837
33. Zanibbi, R., Blostein, D., Cordy, J.R.: Recognizing mathematical expressions using tree transformation. IEEE Trans. Pattern Analysis and Machine Intelligence **24** (2002) 1455–1467
34. Zanibbi, R.: A Language for Specifying and Comparing Table Recognition Strategies. PhD thesis, Queen's University, Kingston (Canada) (2004)
35. Zanibbi, R., Blostein, D., Cordy, J.R.: Historical recall and precision: summarizing generated hypotheses. In: Proc. Eighth Int'l Conference on Document Analysis and Recognition. (2005) 202–206 Vol. 1
36. Horwitz, S., Reps, T.: The use of program dependence graphs in software engineering. In: Proc. 14th International Conference on Software Engineering, New York, NY, USA, ACM Press (1992) 392–411
37. Weiser, M.: Program slicing. In: Proc. Fifth Int'l Conference on Software Engineering, Piscataway, NJ, USA, IEEE Press (1981) 439–449
38. Quillian, M.: Semantic memory. In Minsky, M., ed.: Semantic Information Processing. MIT Press (1968) 216–270

39. Hu, J., Kashi, R., Lopresti, D., Nagy, G., Wilfong, G.: Why table ground-truthing is hard. In: Proc. Sixth Int'l Conf. Document Analysis and Recognition, Seattle, WA (2001) 129–133
40. Lopresti, D.: Exploiting WWW resources in experimental document analysis research. In: Lecture Notes in Computer Science. Volume 2423., Berlin, Springer-Verlag (2002) 532–543
41. Cesarini, F., Marinai, S., Sarti, L., Soga, G.: Trainable table location in document images. In: Proc. Sixteenth Int'l Conf. Pattern Recognition. Volume 3., Québec City, Canada (2002) 236–240
42. Hu, J., Kashi, R., Lopresti, D., Wilfong, G.: Evaluating the performance of table processing algorithms. Int'l J. Document Analysis and Recognition **4**(3) (2002) 140–153
43. Kieninger, T., Dengel, A.: Applying the T-RECS table recognition system to the business letter domain. In: Proc. Sixth Int'l Conf. Document Analysis and Recognition, Seattle, WA (2001) 518–522
44. Liang, J.: Document Structure Analysis and Performance Evaluation. PhD thesis, University of Washington (1999)
45. Lopresti, D., Wilfong, G.: Evaluating document analysis results via graph probing. In: Proc. Sixth Int'l Conf. Document Analysis and Recognition, Seattle,WA (2001) 116–120
46. Colman, A.: Game Theory & its Applications in the Social and Biological Sciences. Butterworth-Heinemann Ltd., London (1995)