The Recognition Strategy Language

Richard Zanibbi
Centre for Pattern Recognition and Machine Intelligence
Concordia University, Montreal, Canada
zanibbi@cenparmi.concordia.ca

Dorothea Blostein and James R. Cordy
School of Computing
Queen's University, Kingston, Canada
{*blostein,cordy*}*@cs.queensu.ca*

Abstract

The table recognition literature contains many strategies specified informally as a sequence of operations, obscuring both models of table structure and the effects of individual decisions. Decision making is more transparent in formal model-based approaches (e.g. grammar-based) but these approaches are less flexible than informal ones. We propose an intermediate level of formalization, defining strategies as a sequence of basic graph transformations that correspond to recognition operations (e.g. classification, segmentation). Transformations are parameterized by logical types and decision functions, which together define structure models and executable strategies for interpreting input graphs. We provide an overview of our first attempt at this intermediate level of formalization, the Recognition Strategy Language (RSL). As a proof-of-concept, we reimplement two informally specified table recognition strategies from the literature in RSL. The RSL implementations capture descriptions of the formerly implicit table structure models, and automatically capture all decision making.

1. Introduction

The vast majority of table recognition strategies in the literature are specified informally as a sequence of operations [10]. This has the undesirable side effects that models of table structure are implicit, defined generatively by the sequence of operations, and that the effects of intermediate decisions are often lost, as usually a single interpretation is modified in-place.

We wished to compare the Handley [5] and Hu et al. [6] table structure recognition algorithms, and the complete set of table cell hypotheses they each generated, including any rejected in the final result. This was going to be impossible if we simply rebuilt the systems using procedural code, transforming data structures for interpretations in-place.

We first tried translating the strategies to a formal model-based (specifically grammar-based) framework. A well-designed model-driven system (such as those of Coüasnon [3] or Klein and Fankhauser [7]) makes it easier to observe and record decision making, and can be 'programmed' succinctly by a model specification. However, we found mapping the sequence of operations in the strategies to a model-based description was difficult, and our formal system was requiring frequent and substantial reconfiguration to incorporate unanticipated requirements. Bagdanov has made similar observations for highly formal systems in computer vision research settings [1].

We then considered an intermediate level of formalization, based on the idea of formal strategies for a 'structure recognition' game. We reasoned that using a small set of basic graph transformations corresponding to standard recognition operations (e.g. classification: label nodes, segmentation: partition a set of nodes, and define a relation between a new node and the partition members), we could define abstract 'moves' of a sequential recognition strategy, capturing relationships between logical types in the process. We also wished to record the effects of 'moves' made at run time. Our resulting formalization is the Recognition Strategy Language (RSL [9]).

2. RSL Strategies and Commands

An RSL specification or *strategy* is comprised of one or more *strategy functions*. Each strategy function contains a sequence of RSL commands and strategy function names. Following convention, execution begins with the *main* strategy function, which must be defined (e.g. Figure 1 lines 14-31). An RSL strategy manipulates adaptive parameters, a set of interpretation graphs (the *current interpretations*), and a set of *accepted interpretations*, which is initially empty. The set of current interpretations contains only the input graph at the beginning of a strategy's execution (e.g. *input.g* in Figure 2).

```
1  model regions
       Image Word Cell Row          % 'Region' type defined by default
3  end regions

5  model relations
       adjacent_right               % 'contains' relation defined by default
7  end relations

9  recognition parameters
       aResolution              300 % dpi; initial value
11     sMaxRowSeparation          2 % millimetres
   end parameters
13
   strategy main
15     adapt aResolution using                                    % 1. get scan resolution from
           getScanResolution()                                    %    the (observed) input Image
17         observing
               {Image} regions
19
       classify {Word} regions as {Cell}                          % 2. classify all Words as Cells
21
       relate {Cell} regions with {adjacent_right} using          % 3. define right adjacency
23         defineRightAdjacency(sMaxRowSeparation,aResolution) %    between Cells

25         segment {Cell} regions into {Row} using                % 4. segment Cells in Rows,
               mergeRowsFromCells()                               %    observing Cells and
27             observing                                          %    adjacent_right relation
                   {adjacent_right} relations
29
       accept interpretations                                     % 5. accept all interpretations
31 end strategy
```

Figure 1. Recognition Strategy Language (RSL) specification *Strategy.rsl* for detecting table cells and rows within a list of *Word* regions. Comments are indicated using the percent symbol (%)

As a strategy progresses, the current interpretations and adaptive parameters are updated; transformations that produce intermediate states are recorded in a log file (*inferences.log* in Figure 2). Using the **accept** command, one or more current interpretations may be moved to the set of accepted interpretations and returned in the output (*accepted_interps.txt* in Figure 2). The simplest form of this command is used in Figure 1 at line 30.

RSL commands perform five tasks: updating adaptive parameters, transforming current interpretations, accepting and rejecting current interpretations, producing terminal and file output, and controlling strategy function application. A complete summary of RSL commands is available elsewhere [9]; the commands in the main strategy function of Figure 1 (lines 14-31) will be described in Section 4.

Our initial RSL prototype is written in the functional language TXL [2]. Decision functions called in an RSL strategy (*inference functions*: see Section 4) are TXL functions defined separately in user libraries (e.g. *MyFunctions.Txl* in Figure 2). The RSL Compiler combines the RSL language library, user libraries, and an RSL strategy to create another TXL program, which may then be interpreted using the TXL interpreter, or compiled to a stand-alone executable (*Strategy.x* in Figure 2). To date, we have nearly always opted to interpret the TXL program, as this allows for faster prototyping.

3. RSL Data: Logical Types, Global Parameters, and Interpretation Graphs

There are four main types of data in RSL: the set of logical region types, the set of logical relation types, a single global set of static and adaptive recognition parameters, and directed graphs representing interpretations of the input (which we will call *interpretations* for brevity). In Figure 1, region types are defined at lines 1-3, relation types at lines 5-7, and recognition parameters at lines 9-12.

Currently parameters may be string or floating point number-valued. Parameter types are determined by initialization values in the recognition parameters section (e.g. Figure 1, lines 9-12). *Adaptive* (variable) parameter names begin with an *a*, while *static* (constant) parameter names begin with an *s*.

As shown in Figure 2, the input to an RSL specification is a graph (*input.g*). The output of an executed specification is a pair of text files. One contains the set of accepted interpretation graphs (*accepted_interps.txt*), and the other a complete log of executed RSL commands and their results (*inference.log*), including the complete history of adaptive parameter values.

For the strategy in Figure 1, interpretation graphs may contain nodes labeled with region types *Image, Word, Cell, Row,* and *Region,* and edges labeled with relation types *adjacent_right* and *contains*. In RSL, nodes represent phys-

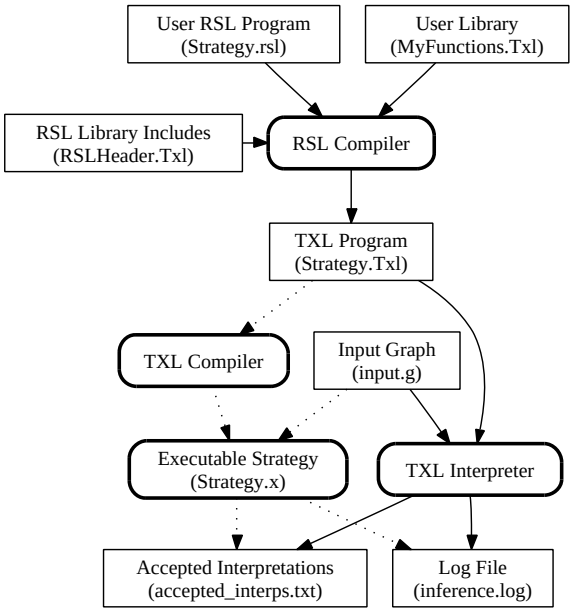


Figure 2. Running *Strategy.rsl* from Figure 1 on graph *input.g* containing an *Image* region and a list of *Word* regions

ical locations (e.g. bounding boxes in a document page) which may be associated with region types. The default region type *Region* includes all nodes in an interpretation, as it defines the set of physical regions. Region composition is defined using the *contains* relation, also defined by default. The *contains* relation is normally manipulated using RSL commands (e.g. **segment**; see Section 4). As an example, a *Cell* region *c1* containing two word regions *w1* and *w2* would be represented as the pairs *(c1,w1), (c1,w2)* in the *contains* relation, where *c1, w1,* and *w2* are graph nodes (i.e. physical locations).

Each accepted graph returned in the output (*accepted_interps.txt* in Figure 2) is annotated with a complete history of the creation, rejection, and reinstatement of physical regions, region types, and relation pairs during the interpretation's construction. We call this the *hypothesis history* of the interpretation [9].

4. Example: Detecting Table Cells and Rows from Words

The strategy in Figure 1 starts by obtaining the scan resolution of the input *Image* region, recording this in the adaptive parameter *aResolution* (via the **adapt** command, lines 15-18). Next, all *Word* regions are labeled as *Cell* regions using the **classify** command (line 20); this occurs because only one class is given ('**as** { Cell }'), and no *inference function* is associated with the command.

When present, inference functions are used to provide decisions made at run time for RSL operations (e.g. what value to assign adaptive parameters, what classes to assign, which segments to define, and which pairs of regions to relate). Inference functions always follow the keyword **using**, and are called from lines 16, 23, and 26 in Figure 1. They must be defined separately in a user library (e.g. *MyFunctions.Txl* in Figure 2). Inference functions return structured text records that are recorded by RSL at run time in a log file (e.g. *inference.log* in Figure 2).

Next, at lines 22-23 the relation *adjacent_right* is defined on *Cell* regions using the **relate** command. At run-time, the inference function *defineRightAdjacency()* will return one or more sets of *Cell* region pairs, to define right adjacency for *Cell* regions. If *defineRightAdjacency()* returns two or more alternative results (sets), then copies of the current interpretation graph are made, and each alternative result is applied to a copy. All RSL commands that manipulate interpretation graphs handle alternative results returned by an inference function in this way, creating a tree of interpretations.

At lines 25-28 *Cell* regions are segmented into *Row* regions by defining new pairs of physical regions in the contains relation (e.g. *(r1,c1), (r1,c2)*). If the previous **relate** command produced multiple alternatives, the **segment** command will be applied separately to each, calling *mergeRowFromCells()* each time. The entire set of current interpretations is then returned as output at line 30 (using **accept**).

Both the **adapt** command at lines 15-18 and the **segment** command at lines 25-28 use *observation specifications*, indicated by the keyword **observing**. These control the visibility of logical types for inference functions. By default, only logical types in the scope type of an RSL command are visible to an inference function (e.g. *aResolution* for the **adapt** command, and *Cell* regions for the **segment** command in Figure 1). Otherwise, regions and relations must be explicitly named to be visible. *Image* regions are made visible to the inference function *getScanResolution()* at lines 17-18, and the *adjacent_right* relation is made visible to *mergeRowsFromCells()* at lines 27-28.

5. Proof of Concept

Starting with an initial definition and implementation, we iteratively improved RSL until we were able to implement both the Handley and Hu et al. table structure recognition strategies to our satisfaction. Including spaces and comments, our Handley strategy in RSL is nine pages long, and the Hu et al. strategy is three pages long (a page contains

roughly sixty lines) [9]. Libraries were also created for the inference functions called in each strategy.

The RSL syntax allowed us to automatically capture the models of table structure used by each strategy, shown in Figures 3a and 4. These models describe the types of regions that a region type may contain (shown with solid arrows), types which a region class may be further classified as (dashed arrows), and relations on region types (dotted arrows with labels, such as for the *indexes* defining indexing structure from headers to columns in Figure 3a).

We also managed to capture dependencies defined by observation specifications and parameters passed to inference functions, such as shown for the Hu et al. strategy in Figure 3b. There we can see that *Cell* regions are defined making reference to *Row* and *Column* regions, each of which have their own dependencies (e.g. both depend on the parameter *sScanResolution*).

The original Handley and Hu et al. strategies and our RSL reimplementations manipulate and produce single interpretations. We could modify inference functions to return multiple interpretations (alternatives) if we wanted, without having to modify the RSL strategies themselves. New strategies could also be created by replacing inference functions, or by reordering RSL operations. We could even *combine* the two strategies by cutting and pasting from the two existing RSL strategies into a new strategy specification.

We used the hypothesis histories recorded by RSL (see Section 3) to revert interpretation graphs to earlier states, and to compare the recall and precision of *all* cell hypotheses generated by the two strategies, including any rejected in the final result. We call these new metrics *historical recall* and *historical precision*. In addition, the hypothesis histories made debugging and error analysis easier [9].

From this experience, we feel that RSL provides a useful *intermediate* level of formalization between informal operation sequences and model-based specifications for recognition strategies. RSL strategies are transparent similar to the model-based approaches, while keeping much of the flexibility of informal operation sequences.

6. Conclusion

We expect RSL to be a useful tool for our research, at least in the short term. A limitation of RSL is that it is poorly equipped to deal with pixel-level information (each pixel would have to be defined as a graph node). We are interested in incorporating lower-level transformations and features into RSL in some sensible way [1, 4, 8]; the appropriate level of abstraction for this is not yet clear to us. We are also interested in knowing whether RSL can be usefully applied in other, very different domains (e.g. with one (audio) and three (CT slice) dimensional data).

Nothing prevents inference functions from returning user interface results, and so the approach outlined in this paper may also be useful for defining ground truth protocols (observation specifications might control the visibility of logical types, for example). RSL might also be used to formalize *human* interpretation of inputs obtained by a graphical user interface.

Acknowledgements

We wish to thank Dr. C.Y. Suen and CENPARMI for providing the resources to write this paper. This research was funded by the Natural Sciences and Engineering Research Council of Canada.

References

[1] A. Bagdanov. *Style Characterization of Machine Printed Texts*. PhD thesis, University of Amsterdam (Netherlands), May 2004. Chapter 6, pp. 89–130.

[2] J. Cordy. TXL - a language for programming language tools and applications. In *Proc. LDTA 2004, ACM 4th International Workshop Language Descriptions, Tools, and Applications*, pages 1–27, Barcelona (Spain), Apr. 2004.

[3] B. Coüasnon. DMOS: A generic document recognition method, application to an automatic generator of musical scores, mathematical formulae and table recognition systems. In *Proc. Sixth Int'l Conf. Document Analysis and Recognition*, pages 215–220, Seattle (USA), 2001.

[4] P. Dosch, K. Tombre, C. Ah-Soon, and G. Masini. A complete system for analysis of architectural drawings. *Int'l J. Document Analysis and Recognition*, 3(2):102–116, Dec. 2000.

[5] J. Handley. Table analysis for multi-line cell identification. In *Proc. Document Recognition and Retrieval VIII (IS&T/SPIE Electronic Imaging)*, volume 4307, pages 34–43, San Jose (USA), 2001.

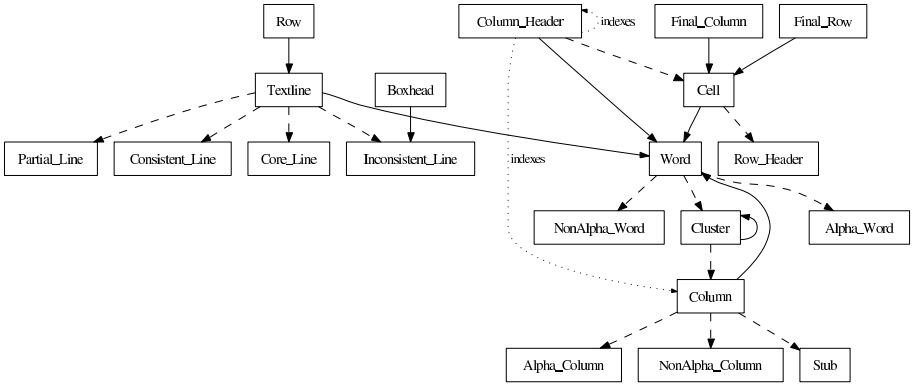
[6] J. Hu, R. Kashi, D. Lopresti, and G. Wilfong. Table structure recognition and its evaluation. In *Proc. Document Recognition and Retrieval VIII (IS&T/SPIE Electronic Imaging)*, volume 4307, pages 44–55, San Jose (USA), 2001.

[7] B. Klein and P. Fankhauser. Error tolerant document structure analysis. *Int'l J. Digital Libraries*, 1(4):344–357, Dec. 1997.

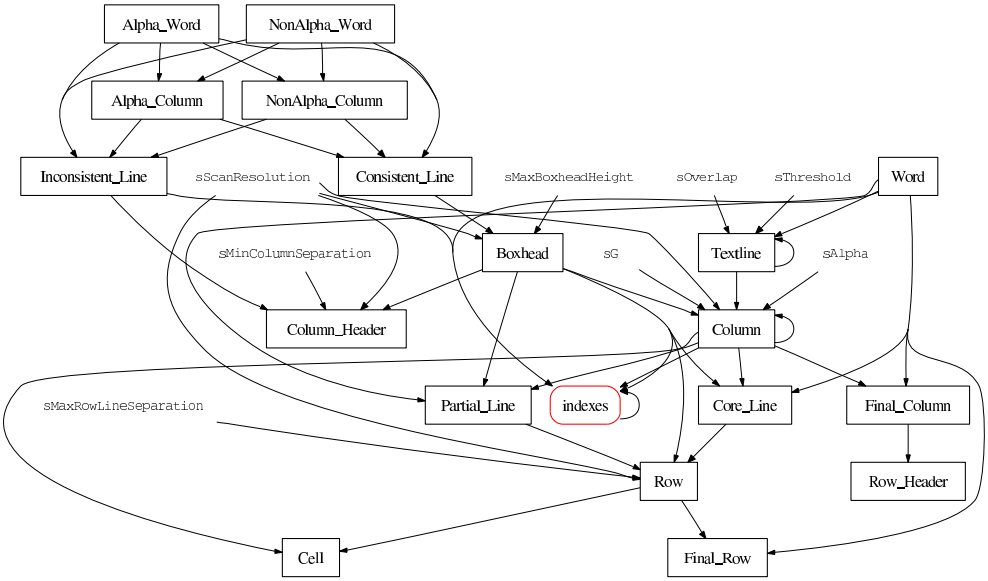
[8] J. Rendek, G. Masini, P. Dosch, and K. Tombre. The search for genericity in graphics recognition applications: Design issues of the Qgar software system. In *Proc. Sixth Int'l Workshop Document Analysis Systems*, pages 366–377, Florence (Italy), Sept. 2004.

[9] R. Zanibbi. *A Language for Specifying and Comparing Table Recognition Strategies*. PhD thesis, Queen's University, Kingston (Canada), Dec. 2004.

[10] R. Zanibbi, D. Blostein, and J. Cordy. A survey of table recognition: Models, observations, transformations, and inferences. *Int'l J. Document Analysis and Recognition*, 7(1):1–16, Sept. 2004.



(a) table model for Hu et al. RSL implementation



(b) region and relation dependencies for Hu et al. RSL implementation

Figure 3. Table model (a) and dependencies of regions and relations (b) for RSL implementation of the Hu et al. table recognition strategy [6]

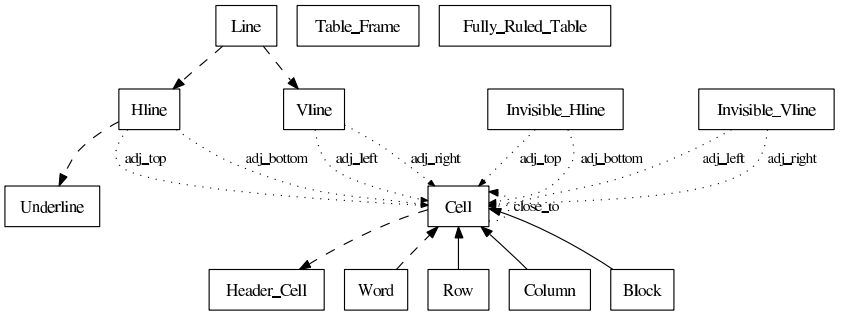


Figure 4. Table model for RSL implementation of the Handley table structure recognition strategy [5]