

# Intelligent Diagramming in the Electronic Online Classroom

Bernd Meyer\*, Kim Marriott\*, Adrian Bickerstaffe\* and Lars Knipping†

\*Monash University, FIT Centre for Research in Intelligent Systems  
Melbourne, Australia

firstname.lastname@infotech.monash.edu.au

†Technische Universität Berlin, Department of Mathematics  
Berlin, Germany

knipping@math.tu-berlin.de

**Abstract**—Digital whiteboards promise to revolutionize the delivery of educational material. Like a traditional whiteboard, they support spontaneous and interactive teaching by allowing free-form editing and scribbling. However, digital whiteboards also allow seamless integration of multi-media content and fit naturally into a web-based teaching environment. The eChalk system has been built to support on-line teaching with digital whiteboards in a university environment. An important feature of eChalk is that hand drawn input is automatically interpreted. This allows sketching and scribbling to be used as an interface to powerful back-end applications. eChalk also allows sketches and scribbles to be transcribed into semantic formats suited to flexible delivery. However, at present developing automatic sketch interpretation engines is extremely time consuming. We describe how we have extended the eChalk system with a generic diagram interpretation component that makes development of new applications considerably faster and easier.

## I. INTRODUCTION

Teachers in many universities and schools use “slideware”, such as Microsoft PowerPoint, as their primary tool for authoring and delivering lecture materials. While allowing the teacher to produce polished presentations with relative ease and greatly simplifying publishing of lecture material, teaching with slideware has received considerable criticism [1]. The fundamental problem is that such tools are not well matched to the requirements of lecturing [2], [3], as they were originally designed to support persuasive presentations which sell a product or idea.

The first problem is that entirely pre-designed presentations tend to promote an overly rapid presentation style, making it difficult for students to keep pace. Second, modern multimedia technology renders it all too easy to design presentations that overload the audience’s sensory system, resulting in presentations that are counterproductive to conveying a rational argument [4]. Thirdly, and perhaps most importantly, teaching with slideware tends to be far less spontaneous than lectures which use more traditional means of presentation such as whiteboards. A whiteboard supports creative thinking and sharing. Spontaneous annotations can be used to draw attention to details, clarify fine points, etc. It is also worth noting that in many areas, such as architecture, the inherent vagueness of freehand drawings is preferable to the rigidity of most computer generated drawings. This can provide an extra source of information [5], [6].

At the same time however, slideware offers significant advantages over a traditional whiteboard: unlike a whiteboard, slides can contain animations, movies, and even interactive components (such as algorithm animations). Furthermore, electronic presentations lend themselves to an integration into web-based teaching environments. Finally, some parts of a lecture simply take too long to be sketched or written down during a lecture and it is advantageous to be able to present them at the push of a button.

It is therefore clear that integration of the two types of systems is desirable. This leads to the concept of *digital whiteboards* [7]. A digital whiteboard consists of a large touch sensitive surface combined with a retro-projection system, or some other form of digitizer-display combination. The lecturer uses a digital pen to write directly on the screen, as on a normal whiteboard. Such systems allow the lecturer to use the same type of free-form editing and scribbling as used with a normal whiteboard, thus maintaining the spontaneity and interactivity of the lecture format. Since it is an electronic display, the digital whiteboard can also integrate seamlessly multi-media content (images, movies, etc) into the lecture. Digital whiteboards are also integrated naturally into web-based teaching environments, allowing online content with a live connection to support active elements. In particular, electronic whiteboards are an ideal medium for distance education as lectures can be attended remotely via web casts.

An important feature of digital whiteboards is that hand-written and sketched input is much more than a visual image: sketching and scribbling are the interface to powerful back-end applications. For example, handwritten mathematical formulas can be evaluated automatically and simplified by a back-end computer algebra system, plots can be generated etc. The same applies to graphics: sketches of digital circuits, for example, could in principle be recognized and interpreted automatically, enabling simulated operation of the digital circuit, automatic translation of circuits into the logical formulas they represent, and automatic verification of these by a theorem prover which runs in the background. To facilitate flexible online and offline delivery and archiving, lectures should automatically be transcribed, including beautified diagrams, output of back-end applications, etc. This should support a variety of target formats, such as PDF and  $\LaTeX$ , and ideally handwritten material should be converted into a

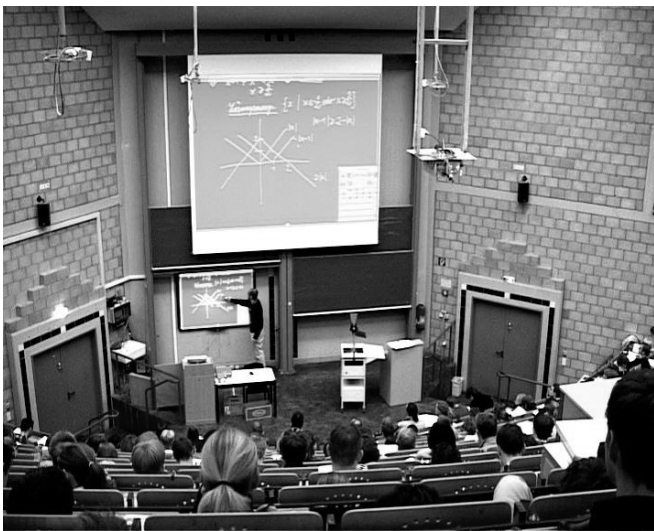


Fig. 1. eChalk used with a digital whiteboard and an extra projector.

semantic format (such as MathML [8]) where appropriate to facilitate effective access via search engines and to support post-processing. It is clear that these capabilities require automatic interpretation of the handwritten and drawn input.

This is a complex list of demands and building such a system is a formidable software engineering task. However, significant progress on this front has been made in recent years with the eChalk system which provides most of the above capabilities, at least to some degree. eChalk is not just a proof-of-concept prototype but is in practical use in a number of universities (see Fig. 1) [9]. However, eChalk has a major shortcoming: lack of generic support for the automatic interpretation of sketches. As in all related projects the lack of *generic* support for sketch understanding makes it very difficult to adapt and extend the system to new domains and different teaching methods. To date, all sketch-interpretation software for eChalk has been laboriously hand-coded and specialized for a particular visual notation. For instance, a whole Master's project was required to develop basic sketch-based digital circuit simulation as an add-on to eChalk [10].

This paper describes how we have extended the eChalk system with a generic diagram interpretation component. This work is based on Cider [11], which provides generic multi-dimensional parsing, and GestureLab, which provides probabilistic gesture recognition. With surprisingly little development effort, Cider and GestureLab allow us to create new applications (called Chalklets) in eChalk which provide automatic, incremental interpretation of new diagrammatic notations. We describe Cider and GestureLab and their integration into eChalk. We also evaluate two case studies that have been examined using the GestureLab-Cider-eChalk combination: a Chalklet for mathematics interpretation and evaluation, and a Chalklet for finite state automata (FSA) interpretation, evaluation, and animation.

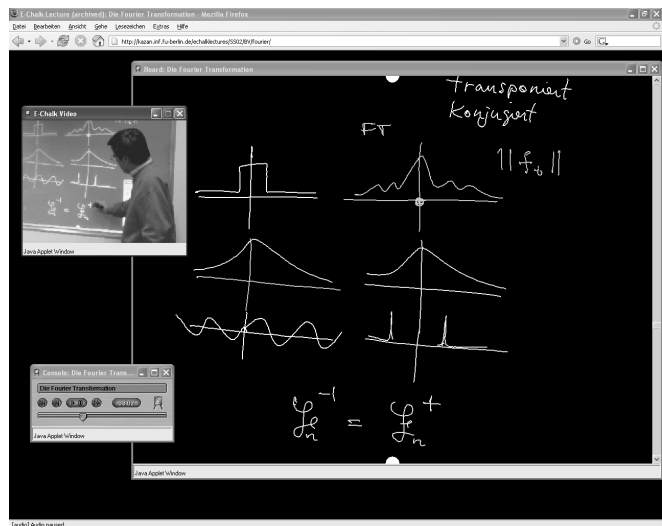


Fig. 2. Replay of a recorded eChalk lecture as seen in a browser.

## II. ECHALK AND ONLINE TEACHING

eChalk initially presents an empty board to draw and write upon. The lecturer works with a digital pen on an interactive wall display (see Fig. 1), however a tablet PC may also be used as an input device. This can either be done in combination with a data projector for classroom teaching or in a stand-alone setting for online delivery. Importantly, eChalk does not force the user to adhere to a prescribed or pre-designed organization of the lecture material. As with a traditional whiteboard, space can be used and subdivided freely. Unlike traditional whiteboards however, the eChalk board is of infinite length, can be scrolled indefinitely, and allows the lecturer to embed multimedia contents. The integration of these features with whiteboard sketching is seamless. For example, the lecturer may embed images from the Web or local storage and may annotate the images by drawing.

Most importantly, eChalk supports live interaction with backend applications. Typical examples are computer algebra systems that can be used to solve or simplify equations, generate plots etc. Currently eChalk is integrated with Mathematica [12] so that handwritten mathematical formulas can be recognized and processed, including complex notations such as differential operators, integral symbols, vectors [13], [14].

eChalk also supports online teaching with bidirectional web integration. Firstly, any web contents can seamlessly be integrated into a lecture delivery. For example, the lecturer can send queries to dynamic Web services and search engines (another form of backend application) and receive the response as text or pictures on the board. Java applets can be run on the board to provide interactive visualizations and additional functionality.

Secondly, eChalk fully supports lecture delivery on the web. All interactions with the board are automatically tracked and stored continuously for viewing through standard web browsers. The development of the board content can be

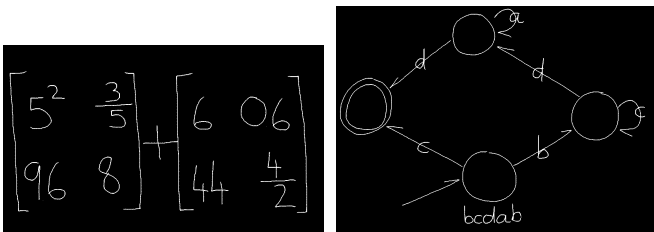


Fig. 3. Cider-driven Chalklets for mathematics and finite state automata recognition.

viewed by a remote learner, both as a live transmission or as an asynchronous replay. The voice of the lecturer may also be recorded and, coupled with the board content, these data streams capture most of the lecture's substance. Optionally, a video stream of the instructor can be added to provide a more personal touch to the remote lesson. An additional transcript of the lecture in PDF format for off-line usage is also generated.

An eChalk session can be replayed by any Java-capable browser (see Fig. 2). The presentation software stores a lecture as a collection of files including an HTML index file which embeds appropriate player applets. These content streams—the animated board image, audio stream and optional video—are handled by a set of individual applets that are synchronized with each other. A fourth applet provides VCR-style navigation through the recording. The system also provides live web transmission of a session (which of course lacks VCR navigation).

Audio and video is streamed using lossy compression and buffering to guarantee interruption-free transmission. The board stream is represented using a vector format, resulting in relatively low bandwidth requirements without any quality losses. Average bandwidth is in practice less than 1 kbps for live recordings [9]. The board's bandwidth requirement is therefore negligible compared to the bandwidth used by audio and optional video, particularly since (proprietary) audio stream codecs between 24 and 256 kbps can be chosen. These performance figures represent maximum values which are rarely reached and, if so, only for a few seconds. Choosing the 64 kbps codec allows remote access to a board and audio stream of sufficient quality with only a dial-up modem connection. Use of a video stream requires an additional 64 kbps. A simple codec based on difference-images and JPEG is used to compress the video signal.

eChalk even supports lecture playback on mobile devices, such as iPods or mobile phones, by combining the records of the visual stream and the audio stream into MPEG-4 videos or Java Midlets [15].

### III. DIAGRAMMING IN ECHALK

As highlighted in Section I, a key strength of the electronic whiteboard concept is that sketching and handwriting can be used as a versatile and flexible interface to powerful back-end applications. Typical examples needed in introductory computer science lectures are the integration of computer

algebra systems for performing complex manipulations of handwritten mathematical formulas and the interpretation, animation, and evaluation of manually sketched diagrams for digital circuits and state automata (Fig 3). These applications have been developed.

eChalk offers a very general architecture to embed such "active" sketches, the Chalklet interface. A Chalklet is an active area of the board that receives pen strokes as input from the board and returns pen-strokes to be rendered on the board. Chalklets are programmed via a Java API. While extremely flexible, this generic interface has the disadvantage that every new Chalklet for a different application has to be developed virtually from scratch without any support for handling diagram structure and interpretation on an adequate level.

In order to react adequately to sketched input (for example, to simulate a circuit), the system must in some form be capable of "understanding" the drawing. While this would of course be an unattainable goal in the general case, it becomes feasible if the drawings are limited to a specific domain and type of diagrams. In these cases, we can often formally describe the two-dimensional syntax of the diagrammatic notation (for example, circuit diagrams) via a declarative formalism. A syntax-directed translation driven by a multi-dimensional parser may then interpret the diagram and translate it into a representation that can be processed by the back-end application. Fortunately, there has been a considerable amount of research into automatic recognition of diagrams as the basis for smart diagrammatic environments and some generic diagram interpreter engines based on multi-dimensional parsers are available [16].

In previous work some of the authors have developed a generic diagram interpretation tool, Cider [11], that has recently been extended to support sketch recognition in pen-based environments [17]. This uses a generic two-stage approach in which syntax analysis (parsing) is preceded by lexical analysis (gesture recognition). This requires the lexical analysis phase to perform a probabilistic recognition since ambiguities on the token level can sometimes only be resolved by taking context on the syntactic level into account. To automate the development of the lexical recognizers as much as possible we have developed a second tool, GestureLab, that generates probabilistic recognizers that are specialized for a given corpus of gestures (see Section III-B).

Together Cider and GestureLab provide a generic tool suite for the construction of interactive sketch interpretation systems required by Chalklets. Such a Chalklet has three main components: the application proper, the Cider runtime environment, and the recognition engine. The recognition engine consists of the lexical recognizer generated by GestureLab and the syntax recognizer (i.e. parser) generated by Cider (see Fig. 4). The application proper passes gestures from eChalk to the recognition engine which builds the interpretation in real-time.

After a lexical recognizer has been trained (in the ideal case just by providing a corpus of example gestures), the second

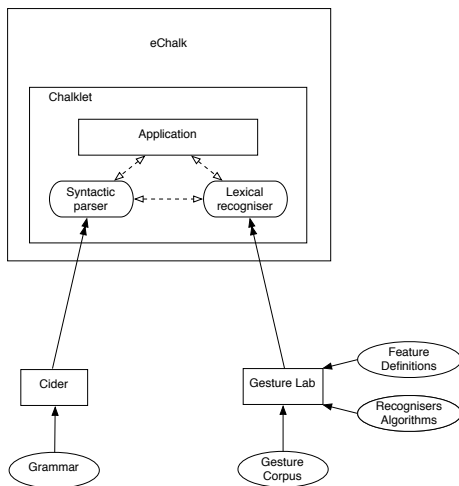


Fig. 4. System Components (filled arrows identify input to a component, double-headed arrows show that a runtime component is generated by a tool, dashed arrows designate runtime communication).

step is to develop a grammar which describes the graphical syntax of the diagram language. Like a parser generator, Cider compiles this grammar into a static library that encapsulates the syntactic recognition engine. The Chalklet communicates with the recognition engine through the Cider controller API. Primarily, the controller handles the addition, modification and deletion of gestures on the front-end side, as well as the addition of non-terminal symbols (recognition of syntactic components), and their modification and deletion. Non-terminals are added, modified and deleted by Cider as the consequence of a production application or in reaction to a structure preserving manipulation. Communication between the front-end and Cider occurs in terms of requests made to Cider from the front-end. Responses from Cider are handled asynchronously via callbacks registered for events such as the creation or removal of a non-terminal symbol, or symbol attribute changes.

An important aspect for interactive diagramming is that Cider also supports structure preserving diagram manipulation. This means that specifications can be written such that once a syntactic diagram component has been recognized, the syntactic structure is automatically maintained when the user manipulates one of the component's constituents. For example, if the user moves a circle in a FSA diagram the enclosed text and attached arrows and labels will follow. This is achieved using a constraint solver in the diagram processor. The constraint solver updates attribute values of tokens automatically so that the specification remains consistent with the visual state of the diagram.

Parsers produced by Cider are fully incremental which means that users can add, delete, or modify components of a diagram at any time and that the interpretation engine automatically maintains a consistent interpretation of the diagram state.

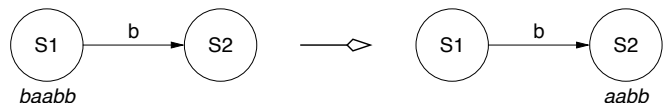


Fig. 5. Animation of an FSA transition.

### A. Cider

We will illustrate Cider usage with excerpts from the Chalklet that we have built to teach students about Finite State Automata (FSAs). This Chalklet recognizes hand-drawn FSA diagrams and can demonstrate visually the process which determines whether a particular input string is accepted by the FSA.

In addition to automatic diagram interpretation, Cider allows the specification of syntactic transformations. Transformations provide a powerful mechanism for encoding diagram manipulations and user interactions. One of the more common uses of transformations is for diagram animation. For example, consider the processing of input strings in a FSA diagram. For a given input and state (identified by writing the input string under the current state), if there is a transition from that state whose label matches the first character of the input string, then the input string can be processed (see Fig. 5).

Cider uses a variant of Constraint Multiset Grammars (CMG [18]), a kind of attributed multi-set grammar to specify the syntax of the visual notation and transformation rules. We can only briefly sketch the general structure of such CMGs here. For full details regarding Cider and its grammar system please refer to [11], [19].

A specification has two parts: symbol definitions and production rules. Symbols have geometric and semantic attributes. Each type is either *terminal* or *non-terminal*. Terminal symbols correspond to the tokens on the whiteboard that are the output of the lexical recognition phase, while non-terminal symbols are more complex objects built from terminal symbols. In the case of FSAs, the definition

```
Circle (terminal) radius:real, mid:point;
```

specifies a terminal symbol `Circle` with a real-valued attribute `radius` and an attribute `mid` of type point, while

```
Transition (non-terminal)
  from:string, to:string, label:string;
```

specifies a non-terminal symbol type with three string-valued attributes that identify the state the transition comes `from`, the state it goes `to` and the transition `label`.

Production rules are used to specify how non-terminal symbols are composed of other symbols. One of the simplest productions for FSAs is that which defines an arc `r` to be an arrow `a` with a textual label `t`. This rule specifies that the arrow and text must have the same mid-point, that the arc has the same start, end and mid-point as the arrow, and that the arc has a label which matches the text:

```
r:Arc ::= a:Arrow, t:Text
  where ( a.mid == t.mid ) {
    r.start := a.start; r.label := t.label;
    r.mid := a.mid; r.end := a.end;
  }
```

During parsing, this production is used to generate new instances of the `Arc` symbol from `Arrow` and `Text` symbols which have a common mid-point. The attributes of the new `Arc` symbol are set by the expressions given between the braces. We call `a` and `t` the right-hand side (RHS) symbols and `r` the left-hand side (LHS) symbol.

Production rules are more generally of the form

$$R_1, \dots, R_m ::= S_1, \dots, S_n \text{ where } C \text{ using } E.$$

A rule is applied by testing whether there is an assignment of symbols in the parse forest to  $S_1, \dots, S_n$ , whose attributes satisfy the conditions  $C$ . If the evaluation is successful, the LHS symbols  $R_1, \dots, R_m$  are produced. The attribute values of these symbols are computed from the attribute values of the RHS symbols according to the expressions in  $E$ .

To allow deterministic incremental parsing, conditions may include tests that particular tokens do not exist. For example, a negative context test can be used in the production for normal states: for a normal state to be formed, there must not exist a second circle which shares its mid-point with the circle input symbol:

```
n:NormalState ::= c1:Circle, t:Text
  where ( c1.mid == t.mid && not exists c2:Circle
          where (c2.mid == c1.mid)
        ) {
  n.mid := c1.mid; n.radius := c1.radius;
  n.label := t.label;
}
```

## B. Lexical Recognition

Given the huge variety of lexical tokens in different grammatical domains, it is clear that we could not possibly use only a single generic recognizer to classify all lexical tokens. A better solution is to build a trainable recognizer system that can be configured for each Chalklet and application domain by letting it learn a new set of gestures. Ideally, the developer should provide only the set of example gestures.

Statistical recognition is the most promising method to build trainable gesture recognizers of this nature. We follow the standard approach and begin with digital ink which captures information about pen position, timing, pressure, and angle data. Statistical summary features such as the total length of the gesture, initial stroke angle, and maximum curvature are extracted from this data and used by a classifier algorithm to predict class labels (gesture types).

We use Support Vector Machines (SVMs) as the default mechanisms for learning new recognizers. SVMs are a popular approach to supervised classifier learning because they are well-understood, theoretically well-founded and have shown excellent performance across a broad variety of applications [20]. A standard SVM is a binary wide-margin classifier with the associated supervised learning algorithm. Standard SVMs provide only non-probabilistic two-way classification and this is clearly insufficient for our token recognition:  $k$ -way recognition is required for a complete alphabet, in addition to probabilistic recognition so that lexical ambiguities can be handled. Extending SVMs for multi-class classification is a challenging task and the focus of much

current research [21]. The standard techniques to build  $k$ -way SVMs are one-against-all [22], one-against-one [22], and DAGSVM schemes [23]. A significant issue with the DAGSVM schema is that its performance depends on the order in which classes are processed, and no practical method is known to optimize this order. One-against-one schemes, on the other hand, tend to overfit and there is no bound on the generalization error of one-against-all. The most important problem, however, is that all three methods suffer from very long training times and this issue is further compounded for large data sets such as our corpus of over 10000 gestures. The reason for the excessive training time is that DAGSVM and one-against-one must both train  $O(k^2)$  binary SVMs, while the one-against-all schema only uses  $O(k)$  SVMs, but each of these must be trained on the full corpus.

In contrast, our approach requires only  $O(k)$  binary SVMs for a  $k$  way classification and reduces the training sample set with each decision. To classify a token it uses only  $O(\log k)$  decision nodes in the best case and  $O(k)$  in the worst case. Using binary SVMs we build a Minimum Cost Spanning Tree (MCST) decision tree which is based on average feature distance (i.e. relative class similarity). Each of the leaves corresponds to a target class and the interior nodes group classes into progressively more disjoint sets. For each internal node in the MCST an SVM is trained to separate all samples belonging to classes in its left subtree from those in the right subtree. This schema does not rely on a predefined class order [17] and has the added advantage of using decision nodes which are based upon class similarity.

A core challenge for any two-phase approach which splits lexical and syntactic analysis is that lexical recognition may be ambiguous; contextual information from syntax analysis may be needed to disambiguate the lexical classification. This disambiguation must be delayed until the parsing stage when contextual syntactic information is readily available. To support this, we construct probabilistic recognizers that return membership probabilities for all possible token classes instead of a single most likely class. The MCST-SVM approach facilitates inference of probability distributions for prediction errors during the training phase in a simple manner: state of these distributions represent one of the target classes to which a  $k$ -way recognizer. after completing the training of all recognizer nodes, a test prediction for each training sample is made and the frequencies of predicting class  $c_i$  for a data item of true (known) class  $c_j$  are tabulated. From these results, maximum likelihood probability distributions are computed for each leaf node of the SVM decision tree.

Coupled with a standard set of feature extractors, the probabilistic MCST-SVM recognizers produce state-of-the-art recognition rates [24] (see Table I).

**GestureLab:** Implementing or simply training such a recognizer without any additional tools is a tedious and error prone task at the best of times. requires us to deploy new chalklets with effort. GestureLab [17] that automates this process to a large degree. GestureLab supports all phases of the recognizer development process: collecting, manipulating and sharing gesture corpora, and automatic training and cross-

TABLE I  
BASIC CHALKLET STATISTICS.

	FSA	Maths
# Token Types	7	26
# Recognition Rate	99.67 %	98.90 %
# Production Rules	12	49
# Transformation Rules	10	n/a

validation of feature extraction and recognizer mechanisms. In the ideal case, the Chalklet developer only has to compile a corpus of sample gestures and use GestureLab to train a suitable recognizer without any additional intervention. The trained recognizer can then be used as a plugin in the Chalklet implementation to provide the lexical recognition for Cider. recognizer is the MCST-SVM decision tree with RBF kernels. The training performs a two-dimensional grid-search to optimize the kernel parameters using 5-fold cross-validation. built-in feature extraction and recognizer mechanisms are insufficient, GestureLab can readily be extended with new features and new recognizer algorithms via a plugin interface.

#### IV. CASE STUDIES

To evaluate our extension to eChalk we have built two Chalklets using the Cider/GestureLab tool suite. Full details and movies of these chalklets can be found at <http://www.csse.monash.edu.au/~berndm/Chalklets>. Basic statistics about these applications are given in Table I.

The first Chalklet is the one introduced in Section III to teach students about FSAs. Examples of the Chalklet's operation are shown in Figure 3 and example rules from the grammar were given in Section III.

The second Chalklet is designed to recognize hand-drawn mathematics. The Chalklet handles standard algebraic notation—addition, multiplication, division, subtraction and exponentiation—and matrix operations—addition, multiplication, subtraction. Matrix elements may contain arbitrary algebraic expressions. Thus the Chalklet handles a significant subset of mathematical notation. Mathematical formulas can be transcribed into  $\LaTeX$  and can also be evaluated automatically. A snapshot of the Chalklet's operation was shown in Figure 3. The following example production illustrates the interpretation process. The production defines a division term  $t$  as composed of two numerals  $a$  and  $b$  with a horizontal division line:

```
t:Term ::= l:Line
  exist a:Numeral, b:Numeral
    where immediately_above(a.bbox, l.bbox) and
          immediately_below(b.bbox, l.bbox) and
          hor_centered(l.bbox, a.bbox) and
          hor_centered(l.bbox, b.bbox)
    { t.value = a.value / b.value }
```

Both Chalklets use GestureLab's standard recognizer and feature set. Training only required that example gestures were supplied. The corpora were collected from a variety of subjects using GestureLab's corpus acquisition features. For the FSA Chalklet these symbols were arrows, circles, and

lowercase characters, while the Mathematics Chalklet required alphanumeric characters and the relevant mathematical symbols. A key question is that of gesture recognition accuracy. Initial testing returns recognition accuracy of 99.67% for the FSA Chalklet and 98.90% for the Mathematics Chalklet. This compares very competitively with hand-coded gesture recognition rates [24].

Since eChalk already provides a hand-constructed implementation of mathematics recognition and evaluation it is instructive to compare this implementation with the new Mathematics Chalklet. The hand-coded solution required approximately 12 man-months to develop, whilst the Chalklet developed in conjunction with Cider/GestureLab required only 2 man-months to develop.

#### V. CONCLUSIONS

Digital whiteboards will almost certainly be part of future teaching environments both for online teaching and face-to-face teaching. Such digital whiteboards have the potential to take sketching, which has always been a natural part of teaching, to a new dimension by interpreting sketches automatically and allowing sketched diagrams to become active intelligent components of a lecture.

Building such diagram components from scratch is however a complex and labour intensive software engineering task. We have shown that this task can be simplified greatly by integrating a digital whiteboard environment with generic diagram interpretation software. Our project has proven the viability of this approach for two comparatively simple prototype cases. In the future we will extend our system to handle other diagram types from a variety of domains. While the combination of eChalk, Cider, and GestureLab has already solved many of the underlying engineering tasks, we expect ambiguity resolution to be one of the most challenging aspects of handling more complex diagram types. Another aspect is to automatically determine the kind of visual notation being used and hence the appropriate interpretation component.

#### REFERENCES

- [1] T. Creed, "PowerPoint no! Cyberspace, yes," *The National Teaching & Learning Forum*, vol. 6, no. 4, pp. 1–4, 1997.
- [2] E. R. Tufte, "The cognitive style of PowerPoint," Chesire (CT), USA, May 2003.
- [3] —, "Power corrupts. PowerPoint corrupts absolutely," *Wired*, pp. 118–119, September 2003.
- [4] R. Clark, "The cognitive sciences and human performance technology," in *Handbook of Human Performance Technology: Improving Individual and Organizational Performance Worldwide*, H. Stolovitch and E. Keeps, Eds. San Francisco: ISPI, 1999.
- [5] M. Gross and E.-L. Do, "Ambiguous intentions: a paper-like interface for creative design," in *ACM UIST*, Seattle, WA, November 1996.
- [6] J. Landay and B. Myers, "Sketching interfaces: Toward more human interface design," *Computer*, vol. 34, no. 3, pp. 56–64, 2001.
- [7] R. Rojas, L. Knipping, G. Friedland, and B. Frötschl, "Ende der Kreidezeit - Die Zukunft des Mathematikunterrichts," *DMV Mitteilungen*, vol. 2, pp. 32–37, 2001.
- [8] W3C, <http://www.w3.org/TR/MathML/>.
- [9] L. Knipping, "An electronic chalkboard for classroom and distance teaching," Ph.D. dissertation, FB Mathematik und Informatik, FU Berlin, Berlin, 2005.
- [10] M. Liwicki and L. Knipping, "Recognizing and simulating sketched logic circuits," in *Knowledge-Based Intelligent Information and Engineering Systems*, Melbourne, Australia, September 2005.

- [11] A. Jansen, K. Marriott, and B. Meyer, "Cider: A component-based toolkit for creating smart diagram environments," in *Int Conf. on Distributed and Multimedia Systems*, Miami, September 2003.
- [12] S. Wolfram, *The Mathematica Book*. Wolfram Media, 2003.
- [13] E. Tapia and R. Rojas, "Recognition of on-line handwritten mathematical expressions using a minimum spanning tree construction and symbol dominance," in *Graphics Recognition (GREC)*, Barcelona, July 2003.
- [14] G. Friedland, L. Knipping, and E. Tapia, "Web based lectures produced by AI supported classroom teaching," *Int. J. of Artificial Intelligence Tools*, vol. 13, no. 2, pp. 367–382, 2004.
- [15] A. Lüning, G. Friedland, L. Knipping, and R. Rojas, "Visualizing large-screen electronic chalkboard content on handheld devices," in *IEEE Multimedia Technologies for E-Learning*, Taichung, Taiwan, 2007, pp. 369–373.
- [16] B. Meyer, K. Marriott, and G. Allwein, "Intelligent diagrammatic interfaces: state of the art," in *Diagrammatic Representation and Reasoning*, P. Olivier, M. Anderson, and B. Meyer, Eds. London: Springer, 2001.
- [17] A. Bickerstaffe, A. Lane, B. Meyer, and K. Marriott, "Building smart diagram environments with domain-specific gesture recognizers," in *Graphics Recognition (GREC)*, Curitiba, Brazil, September 2007.
- [18] K. Marriott, "Constraint multiset grammars," in *IEEE Symposium on Visual Languages*, 1994, pp. 118–125.
- [19] A. Jansen and A. Bickerstaffe, "Cider reference manual," Monash University, Tech. Rep., 2004, <http://www.csse.monash.edu.au/~adrianb/CIDER/>.
- [20] B. Schölkopf and A. Smola, *Learning with kernels*. The MIT Press, 2002.
- [21] Y.-C. Wanga and D. Casasent, "Soft-decision hierarchical classification using SVM-type classifiers," in *IEEE World Congress on Computational Intelligence*, Hong Kong, 2008.
- [22] C. W. Hsu and C. J. Lin, "A comparison of methods for multi-class support vector machines," *IEEE Transactions on Neural Networks*, vol. 13, 2002.
- [23] J. C. Platt, N. Cristinini, and J. Shawe-Taylor, "Large margin DAGs for multiclass classification," *Advances in Neural Information Processing Systems*, vol. 12, pp. 547–553, 2000.
- [24] U. Garain and B. B. Chaudhuri, "Recognition of online handwritten mathematical expressions," *IEEE Transactions on Systems, Man, and Cybernetics - Part B*, vol. 34, no. 6, pp. 2366–2376, 2004.