

The Visualization of Constructive Proofs by Compositional Graph Layout: A World-Wide Web Interface

Jutta Eusterbrock and Michalis Nicolaides
 GMD – German National Research Center for Information Technology
 Rheinstr. 75, 64295 Darmstadt, Germany,
 Email: {eusterbr, mnico}@darmstadt.gmd.de

Abstract—Directed graphs are a foundation for the encoding of various formal objects, such as requirement specifications and proof trees, in logic-based software synthesis and automated proving. The visualization of these objects by graph drawing can make formal descriptions more comprehensible and user-friendly. In this paper, we sketch and apply a distributed architecture with a World-Wide-Web user interface where the CoGLay graph layout component has been connected with the SEAMLESS synthesis system.¹ The SEAMLESS synthesis system uses specific graphs where nodes may be graphs themselves for the structured representation of complex objects at different abstraction levels. Shared subgraphs refer to re-used substructures. The graph layout system CoGLay uses algorithmic techniques and declarative knowledge to produce composite layouts for structured graphs at different levels of detail and in an incremental way. Sample applications are shown by screendumps.

I. INTRODUCTION

Automated assistance for the synthesis of software starting from given high-level user requests is highly desirable. Current work in software engineering reflects a widespread consensus that tools which assist software construction processes should be based on formal methods to ensure maintenance. Further, they should provide correctness criteria which may result in reliable software. It has long been recognized that visual and diagram technics for the cognitive presentation of proofs and structures – instead of textual specifications or encodings of cryptical mathematical expressions – can increase users' ability to effectively understand and develop proofs and programs.

SEAMLESS² is a framework which provides generic logic-based methods and knowledge representation means for the synthesis of systems given formal specifications. Synthesis is assumed to be an evolutionary knowledge acquisition process. The prototypical environment is based on a multi-layer architecture for structuring the different kinds of knowledge and specification entities (cf. [Eus96]). Synthesis "artifacts" are dynamic development graphs, specification formulas organized

into lattices, proof terms, and posets, ie. all of them may be regarded as graphs.

In this paper, we present the design for a WWW user-interface of SEAMLESS which integrates a graph layout implementation by a Client-Server architecture. As in [GFG95] the architecture is based on the concept of disjoint or hierarchically organized types for development entities. Types may be associated with non-unique widgets and dialogue components. User-interfaces are generated by composing primitive representation formats. This generic architecture provides desirable features as flexibility and re-usability. The WWW-realization allows globally distributed clients to use synthesis systems, independent of their available platforms. As there exist algorithmic methods for the visual presentation of standard types as tables or charts, the visualization of graphs is far more complex: The graph layout problem arises from the requirement that the nodes and edges have to be positioned into a pleasant design. Various yet unsolved algorithmic problems as the existence of a cross-free layout for a given graph are concerned and have to be tackled by appropriate heuristics. Graphs may be very huge and, therefore, need to be presented carefully. A well-known psychological assumption [Mil53] states that human memories are hierarchical, made up of a limited capacity workspace of about seven locations, which holds and processes items of information currently under attention. Thus to understand visual presentations of huge graphs, one has to impose some hierarchical structure on them. Different parts of a graph may be different autonomous units, which in turn may include other finer units, so together they build a hierarchy. Each unit belongs to distinct graph class according to its type.

CoGLay(**Compositional Graph Layout**) (cf. [Nic95]) is a compositional approach to graph layout. It relies on the assumption that incremental visualization of data can be better understood by the user, hence reducing the cognitive load. Attributes of information (dimension, state, volume, precedence etc.) can be ordered, achieving a better intergration of context and detail. A compositional layout is incremental by definition, since smaller, more comprehensible units are created first, building an overview. The composition rate is adjusted to the user's personal evaluating abilities.

The set of graph types is divided into hierarchically orga-

¹Some system features are being demonstrated (dependent on a running server) at <http://www.darmstadt.gmd.de/~eusterbr/seamless/public.html/seamless.html>.

²SEAMLESS is an acronym for an evolutionary process model for software development: Specification, Example Computation, Abstraction, Modification, Logical Validation, Extraction for System Synthesis (cf. [Eus96]).

nized classes according to their properties. Each class has a corresponding layout algorithm best satisfying the layout criteria imposed by the class. We can transmute the graph instances (by loosening or tightening some appropriate criteria) from one class to another upon the hierarchy. This enables us to be flexible with the application of layout algorithms on the graph instances, thus achieving a better interplay among the algorithms.

The paper is organized as follows. The following section sketches principles for the generic architecture of the distributed CogLay SEAMLESS architecture with WWW interface and the common graph model which captures type information and allows to structure huge objects. Section III presents the CogLay graph layout approach. Section IV exemplifies the visualization of proof terms by the CogLay graph layout server. The paper is concluded with some final remarks and a comparison to related work.

II. THE DISTRIBUTED SEAMLESS COGLAY ARCHITECTURE WITH WWW USER-INTERFACE

The major purpose of this paper is to sketch the interplay between the inference engine of SEAMLESS and the CogLay layout system. This approach to visualization can be applied to a variety of graph-based representations. A complete account of the technical aspects of the implementation is far beyond the scope of this paper, hence we only give a short overview of the distributed system architecture.

The SEAMLESS system provides generic methods for synthesis and knowledge acquisition and is implemented in Prolog. The SEAMLESS application core is based on a hierarchically organized three-layer architecture for knowledge modeling, each layer divided into data types and declarative specifications. Complex types are built up from basic ones by type constructors as disjunction. The main interaction with SEAMLESS is querying generic methods by submitting its arguments and activating context parameters. The evaluation of generic methods will cause information to be constructed. Arguments and generated entities are typed.

There is a general consensus that application and presentation layers have to be separated in order to built user-interfaces with the desirable features re-usability; flexibility; platform independence. So we defined a new layer on top of the SEAMLESS layer which declaratively models some of the interactive functionality available to the user³ in terms of types and methods. The general design of the user-interface for the SEAMLESS system is based on the idea (cf. [GFG95]) that abstract types are being associated with widgets. Complete interface designs are developed by composing basic design objects.

A. The WWW Realization

A previous user-interface version was implemented using *Tcl/Tk*. We decided to re-implement the user-interface using

³SEAMLESS is implemented in Prolog as set of predicate and type definitions and each of the predicates may be queried.

the upcoming *World-Wide Web* and the recently developed programming language *Java*. The communication is established on the basis of a client-server architecture (cf. Figure 1) where the WWW user-interface is the client process and the SEAMLESS system is a subcomponent of an integrated server system using SchedCom [Mül96], a tool for the synchronisation, buffering, and management of the communication between parallel processes, supporting different protocols. In our case (cf. Figure 1) the SEAMLESS environment and CoGLay system are on the one side of SchedCom and the processes representing the various WWW-clients started through CGI-scripts on the other side. The transported data are just strings. For each SEAMLESS type or entity an appropriate Java Abstract Windowing Toolkit (AWT) widget was chosen if already existed, or otherwise, constructed to comply to demands. Most effort was given to graph panels.

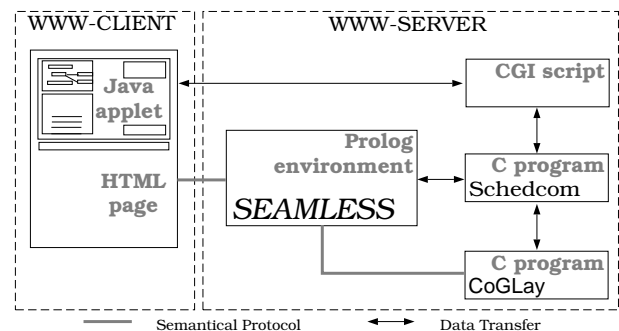


Fig. 1. SEAMLESS-CogLay architecture

B. The Common Graph Model

Directed graphs form the natural foundation for the representation of various synthesis artifacts on different abstraction layers. For example, they may be used to represent posets, specifications, proof terms or process dependencies. In order to provide abstraction, structuring and dynamic refinement mechanism, the usual graph notation is extended.

- Graphs having the same structural properties are grouped together to form a concept or class. Concepts can be ordered by a super-subset relationship. Conceptual relationships are captured by type information and type hierarchies.
- Aggregation of entities on different abstraction levels is realized by metagraphs. Metagraphs are graphs where nodes may be seen as graphs themselves.

The SEAMLESS synthesis systems provides a canonical algebraic data structure (cf. [Eus95]) for representing and constructively reasoning on directed acyclic graphs (DAGs), type information and performing operations. DAGs are identified with recursively constructed terms, called graphterms. Node labels can be associated with type information. Types on the object layer coincide with common graph concepts, types on higher layers are dependent on the representation tasks. For example, formulas are typed by boolean values.

Extended graphterms ('metaterms') may entail subterms which are references to graphterms.

The CogLay graph layout systems assumes that, for each class, there is a layout algorithm for calculating its layout. A root class exists, whose corresponding layout algorithm can layout all graph instances—functioning as the default layout algorithm. The normal graph representation $G = (V, E)$, is supplemented by including first the class of the graph $c_G \in C$, and second a list of pairs $(S_i, I_{G,S_i})_{i=1,\dots,n}$, where S_i denotes a semantically connected subgraph of G , i.e. $S_i <_{sub} G$,⁴ and I the corresponding set of edges between the nodes of G and S_i , ie. $I_{G,S_i} = \{(v, u) \mid (v \in V_G \wedge u \in V_{S_i}) \vee (v \in V_{S_i} \wedge u \in V_G)\}$ — we will refer to such an edge as *inter-graph-edge*. Thus the graph representation results in the 4-tuple $G = (V, E, c_G, \langle (S_1, I_{G,S_1}), \dots, (S_n, I_{G,S_n}) \rangle)$. Inter-graph-edges can exist only when the $<_{sub}$ relation directly occurs.⁵

In the case of directed acyclic graphs, there is a one-to-one relationship among these two representation formats. The SEAMLESS graph syntax allows the linking of meta subterms to multiple predecessors. This case is not yet considered by the CoGLay graph syntax.⁶

III. COMPOSITIONAL GRAPH LAYOUT

The main motivation for the CoGLay system derives from the need to layout substructures of a heterogenous graph according to their requirements and specifications. The current layout systems can be divided in two main categories: the *declarative approaches* emphasize the intrinsic properties of the data—though an active interaction of the user is needed. On the other hand, the *algorithmic approaches* optimise the automatization of the layout process producing layout algorithms based on concrete *layout aesthetics* yielding to relatively poorer layouts when the applied graph does not conform to them.

CoGLay aims to combine the two methods by using various existing layout algorithms to layout distinct subareas of the graph, whose layouts are then composed to a whole. To achieve this goal we have to solve two problems, first the description and handling of the distinct subareas of the graph, and second the incompatibility of the various layout algorithms. We propose to deal with these problems by introducing abstraction in the graph definition and the use of generic layout algorithms, i.e. layout algorithm abstraction.

Figure 2 shows a brief description of the recursive layout method. Using a database of layout algorithms the layouts of the subgraphs are calculated first. These are then composed

⁴ $S_i <_{sub} G$ means that S_i is a subgraph of G . $S_i \cap S_j = \emptyset$.

⁵I.e. if \exists inter-graph-edge e between S_i and $G \Rightarrow$

$(S_i <_{sub} G) \wedge \exists S_j : (S_j <_{sub} G) \wedge (S_i <_{sub} S_j)$

⁶Hence, each representation format has further advantages. The CogLay format allows to deal with arbitrary graphs and can be easily embedded into conventional programming languages like *C* and *Java*. The canonical recursive term representation of DAGs provides major advantages for efficient automated reasoning and conceptual analysis, based on unification, term rewriting and induction. For example, graphterm unification procedures for efficient handling the information on taxonomic organized types have been devised.

```

call  $L_G = layout\_graph(G)$ 
input graph  $G$  (a 4-tuple):
   $G = (V, E, c_G, \langle (S_1, I_{G,S_1}), \dots, (S_n, I_{G,S_n}) \rangle)$ 
   $V$ : nodes set,  $E$ : edges set
   $c_G$ : graph class
   $\langle (S_1, I_{G,S_1}), \dots, (S_n, I_{G,S_n}) \rangle$ : list of pairs
   $S_i$ : graph
   $I_{G,S_i}$ : inter-graph-edges set
output layout  $L_G$  (a 4-tuple):
   $L_G = a(G)$ 
   $L_G = compose(L_G^*, L_{S'_1}, \dots, L_{S'_n})$ 
layout_graph( $G$ ) {
  if ( $G$  is plain) {
    /* find appropriate algorithm  $a$  */
     $a = algmap(c_G)$ 
    /* layout  $G$  with  $a$  */
    return  $a(G)$ 
  }
  else if ( $G$  is complex) {
    /* find appropriate algorithm  $a$  */
     $a = algmap(c_G)$ 
    /* make preliminary layout with  $a$  */
     $L_G^* = a(G)$ 
    /* for all subgraphs  $S_i$  */
    for all ( $S_i$ ) do {
      /* transfer layout information to  $S_i$  */
       $S'_i = update(S_i, L_G^*)$ 
      /* recursive call */
       $L_{S'_i} = layout\_graph(S'_i)$ 
    }
    /* join all layouts of subgraphs */
    return  $compose(L_G^*, L_{S'_1}, \dots, L_{S'_n})$ 
  } }

```

Fig. 2. The recursive layout method.

yielding the layout of the graph containing them. Because it may also be a subgraph itself, the procedure continues till the layout of the upper graph is composed. The abstraction mechanisms enable graph class and layout algorithm transparency during the composition.

In order to deal with the inter-graph-edges (the edges between subgraphs) a preliminary layout is made for each complex graph (a graph containing subgraphs) with the subgraphs shown only as single nodes (their metanodes). It is made before the subgraphs are layouted, in order to gain information about their relative position to each other and influence their layouts accordingly.

Because of the existence of several adjacent sublayouts (we use the metaphor of the pictures in a photo-collage), changes in one sublayout do not affect other sublayouts—except, of course when there is a direct dependence between them. This intrinsic property allows incremental layout without loss of context.

IV. PROOF VISUALIZATION

In SEAMLESS, requirement specifications of programs are expressed as logical formulas. These are constructively proven with respect to a domain theory by generic methods employing a three-valued logic. Specifications are broken down into disjunctively and conjunctively connected subspecifications until axioms are reached and subsolutions are composed to yield constructive proofs. Since we are using a three-valued logic, three outcomes are possible: there exists a program for computing the stated goals, the goal is not provable, there could exist a proof, assuming further axioms are true. Results of constructive proof searches are being implemented as typed graphterms (cf. subsection II-B), named *proof terms*. Nodes are assigned specifications and they are being typed by its truth values. Edges denote relationships among the provability of specifications and subspecifications, eg. variable bindings or case divisions.

Proofs may become large or complicated. Mathematicians impose some structure on them by separating proofs into lemmas. Lemmas can shorten proofs if lemmas are used more than once and they provide more insight. Reflecting this human procedure, SEAMLESS takes an evolutionary approach to synthesis, where proof processes are analyzed to yield lemmatas and specifications are being proven by incremental adjusting or re-using already derived known proofs to subspecifications. We called this augmented “Divide-and-Conquer” principle *Knowledge-based Divide-and-Conquer*. Hence, first proofs may contain shared subproofs. Sharing reflects re-use of proofs. The size may shrink exponentially as it has been pointed out in [GP94]. Secondly, already derived subproofs are assigned names, they are stored in the knowledge base for re-use and can be accessed by their names or their specifications.

a) Visualization: of arbitrary synthesis artifacts which are implemented as typed DAG’s is realized through calling the following complex components of the SEAMLESS-CoGLay system.

- *Extraction:* Only those aspects that are relevant for visualization are selected and mapped onto a reduced term.
- *Layout type analysis:* Context-dependent proof types are annotated with the CoGLay graph classes and interaction mode relevant for their layouts.
- *Transformation:* Term-based SEAMLESS structures are transformed into the string-based CoGLay syntax. This procedure involves determining the parts of a graph which shall be considered as CoGLay subgraphs and accessing the SEAMLESS knowledge-base in order to substitute references by object level expressions.
- *Layout communication:* The CogLay server is asked to visualize the submitted string. On user-demand subgraphs are incrementally refined by invoking the CoGLay operations *unfolding* and *folding* of subgraphs.

The SEAMLESS system was applied to constructively prove the (non)-existence of complexity-bounded programs for sorting and searching over partially ordered sets. Some decisions for the layout of proofs of type *true* were made. Instead of

representing proofs as DAG’s with nodes being horn clauses (cf. [SL91]), the usual mathematical notation, ie. extended decision trees, was chosen (Step (A1)). Proof terms of type *true* are being recognized as directed graphs whose nodes have maximal degree two (Step (A2)). Decision graphs constructed in the re-use oriented proof mode include references for named subgraphs (‘lemmas’) and they share subgraphs. A natural visualization would be to consider lemmata as CoGLay subgraphs which can be annotated with *fold*, *unfold* modes (Step (A3)). Currently CoGLay can handle and layout only inter-graph-edges between graphs with a direct $<_{sub}$ relation between them. This restricts the set of lemmata which can be visualized as folded.

Resulting experimental layouts will be demonstrated on small examples. Figure 3 shows a decision graph representing the solution for selecting the 4th element of a 8-poset with at most 6 comparisons. Figure 4 shows the decision graph for the same problem. However, the problem has been solved in a re-use and knowledge-based mode. At a first glance, the user will see what has been derived for new during the proof search. The same graph with one unfolded subsolution is shown in figure 5.

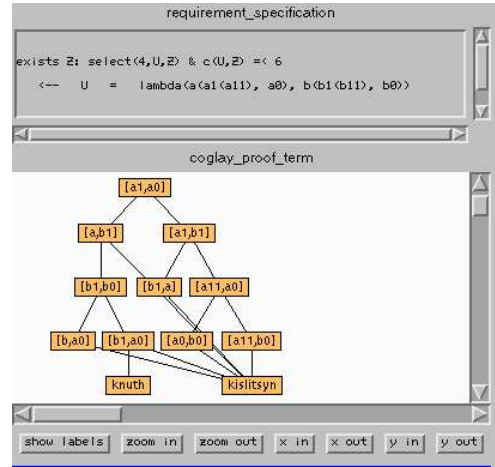


Fig. 3. Structured decision graph

V. CONCLUSIONS

In this paper we described a new distributed and open generic architecture which integrates the SEAMLESS system for evolutionary synthesis and the CoGLay graph layout server. This approach differs significantly from previous work since it allows to re-use previously developed proofs and to visualize proofs at different levels of abstraction in an incremental and composite way.

There is a growing interest in the incorporation of visualization technics into logic programming and synthesis systems which are based on formal specifications and deductive methods, mainly for assisting requirement specification processes and based on uniform tools (cf. IOSS [HSZ95], SPECWARETM [SJ95]). Compositional layouts is a rather new field. A conscious effort to combine declarative and

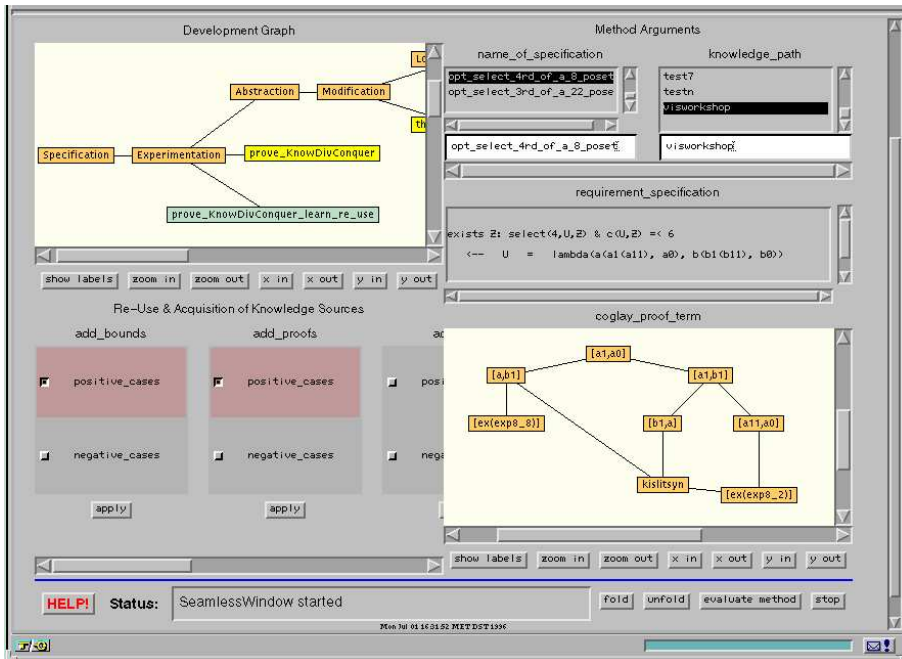


Fig. 4. Structured decision graph with re-use

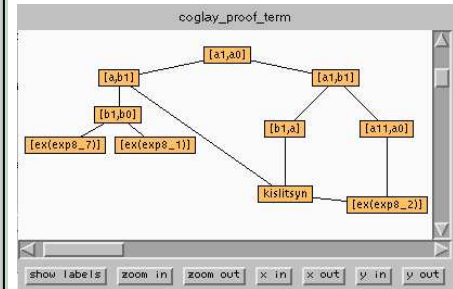


Fig. 5. Unfolded decision graph with re-use

algorithmic approaches undertake Lin and Eades in [LE94], being on an interesting question formulating stage. Dynamic algorithms for various classes are investigated in [CBTT94]. KGB [Ben95] uses graph abstraction for dealing with different graphs the same way. It is most desirable to combine the advantages of these systems, i.e. using various algorithms in a dynamical, and incremental environment for presenting combined, heterogenous data according to its semantic in a transparent way.

Our future plans are the visualization of further synthesis artifacts. One specific goal is a layout method and an appropriate data structure which would enable inter-graph-edges between arbitrary graphs parts. CoGLay provides an open interface for the embedding of further layout algorithms and layout knowledge. Given an arbitrary formal entity which is encoded as directed graph, its layout is not predetermined. The visualisation of SEAMLESS artifacts through CoGLay is based on flexible decision rules.

The presented open architecture has further revolutionary advantages for cooperative work and communication of heterogenous systems. It suggests a general way how heterogenous systems may interact to solve specific tasks in a cooperative way. Computing environments can be called world-wide independent of available platforms. Users don't have to worry about system specific installation tasks.

Acknowledgements: The implementation of the SchedCom-Prolog protocol by Birgit Wendholt and the advice of Adrian Müller in the use of SchedCom are gratefully acknowledged.

REFERENCES

[Ben95] H. Benz. KGB a customizable graph browser. In *Symposium on Graph Drawing, GD 95*, Lecture Notes in Computer Science, pages 20–23. Springer Verlag, September 1995.

[CBTT94] R. Cohen, G. Di Battista, R. Tamassia, and I. Tollis. Dynamic graph drawings: Trees, series-parallel digraphs, and planar st-digraphs. accessible through ftp, 1994.

[Eus95] J. Eusterbrock. Efficient reasoning with transitive DAG's. In *Proceedings of the International KRUSE Symposium, Knowledge Retrieval, Use and Storage for Efficiency*, pages 97–101, 1995.

[Eus96] J. Eusterbrock. A multi-layer architecture for knowledge-based system synthesis. In Zbigniew W. Ras and Maciej Michalewicz, editors, *Foundations of Intelligent Systems, 9th International Symposium, ISMIS'96*, volume LNCS 1079, pages 582–592. Springer Verlag, 1996.

[GFG95] A. Goldberg, R. Furst, and C. Green. A refinement approach to visualization. Technical report, Kestrel Institute, 1995.

[GP94] J. Gobault and J. Posegga. Bdds and automated deduction. In Z. W. Ras and M. Zemankova, editors, *Methodologies for Intelligent Systems, Proc. 8th International Symposium, ISMIS'94*, pages 541–550. Springer Verlag, 1994.

[HSZ95] M. Heissel, T. Santen, and D. Zimmermann. A generic system architecture for strategy-based software development. Technical report, Technische Universität Berlin, 8 1995.

[LE94] T. Lin and P. Eades. Intergration of declarative and algorithmic approaches for layout creation. In *Proceedings of DIMACS International Workshop, GD '94, on Graph Drawing*, pages 376–387, 1994.

[Mil53] Miller. The magical number seven plus or minus two: some limits on our capacity to process information. *Psychological Review*, 63:81–97, 1953.

[Mül96] A. Müller. SchedCom - a c-library for scheduled http-process communication on unix machines. Technical Report forthcoming, GMD, 1996.

[Nic95] M. Nicolaides. Rekursive Graph Layout mit Deklarativen und Algorithmischen Ansätzen. Master's thesis, Technical University of Darmstadt, Darmstadt, Germany, May 1995.

[SJ95] Y.V. Srinivas and R. Jülig. SPECWARE (TM): Formal support for composing software. In *Proceedings of the International Conference on the Mathematics of Program Construction*. Pitman Publ., 1995.

[SL91] H. Senay and S.G. Lazzeri. Graphical representation of logic programs and their behaviour. In *IEEE Workshop on Visual Languages*, pages 25–31. IEEE Computer Society Press, 1991.