Formalization of Visual Mathematical Notations

Bernd Meyer
bernd.meyer@acm.org
Ludwig Maximilians Universität
Insitut für Informatik
Oettingenstr. 67
D-80538 München, Germany

Abstract

This paper discusses picture logic, a visual language for the specification of diagrams and diagram transformations. Formal specification techniques for diagrammatic or visual languages have previously mainly been targeted towards static diagrammatic languages. For reasoning about certain types of diagrams, however, formalizing a notion of change is inevitable. This is particularly true of visual mathematical notations whose evaluation rules or consequence relations correspond to visual or graphical transformations. The paper presents constraint-based extensions of picture logic which render it suitable for the specification of such diagram notations and the required transformations.

Diagrammatic Reasoning and Formalizations

In *computational* diagrammatic reasoning, we can distinguish between reasoning *about* diagrams and reasoning *with* diagrams. We can either use non-visual computational methods to reason about diagrams or we can use computational reasoning methods that are themselves diagrammatic in nature but not necessarily applied to a diagrammatic domain. Of course, both directions can also be integrated into reasoning about diagrammatic domains with visual computational methods. This seems to be a very natural and promising extension, for if we aim at using diagrams as reasoning tools one of their natural places should be where the domain of reasoning is diagrammatic itself. Such an integration of reasoning with diagrams about diagrams will be discussed in this paper. We will present a diagrammatic logic language for the formalization and animation of diagrammatic notations.

There has been considerable discussion in the visual language community, whether a strict formalization of diagrammatic languages is really necessary or even useful at all. While the supporters of a formal approach generally maintain that only formal definitions of visual languages allow the creation of flexible tools like parsers and compilers, the arguments most often heard against such formalizations criticise the inflexibility and inefficiency of meta-environments that are directly based on formal language definitions. No matter which of these position one supports, it is clear that within the realm of diagrammatic reasoning some areas cannot live without a proper formalization of diagram languages. Our chief witness are visual mathematical notations, such as Venn Diagrams and Euler Circles, Peirce's α-β-Calculus, various visual notations for Church's λ-calculus (Citrin, Hall, & Zorn 1995; Keenan 1995) and boundary logic (Bricken 1988). Several technical notations, e.g. State Charts (Harel 1988), and declarative visual programming languages, e.g. Pictorial Janus (Kahn & Saraswat 1990), also fall into this category. It is perfectly clear that if we want such notations to be fully valid theoretical tools they have to be adequately formalized.

Despite the fact that visual reasoning and intuition is often one of the major factors in mathematical discovery, the conviction of most mathematicians is that visual methods can only serve as inspiration for discovery and as illustrations for proofs but are not permissible as proofs themselves, because they are not based on a well-defined, closed set of reasoning methods.[1] Nevertheless, the potential that visual expression bears for mathematical language and even mathematical proofs has been realized and has been demonstrated several times. A superb collection of examples is (Nelsen 1993), but even the introduction to this book states that "of course, 'proofs without words' are not really proofs".

In recent years some ground breaking approaches have attempted to establish the status of diagrammatic notations as fully valid mathematical reasoning devices by rigorously formalizing them (Hammer & Danner 1993; Hammer 1993; Shin 1995; Barwise 1993). While

[1] For a discussion see (Allwein & Barwise 1993; Shin 1995).

representing a big leap towards establishing diagrammatic notations as valid mathematical reasoning systems, they all were aimed at some particular diagrammatic system and the formalization methods chosen were targeted towards this specific system. A general framework or meta-language for the definition of diagrammatic mathematical notations has not yet been established and most of these approaches do not give rise to a computational implementation.

Another branch of diagrammatic reasoning that would benefit from the existence of a formal meta-language from a more practical point of view is found where typical AI reasoning techniques are applied to diagrammatic notations. Examples are the interpretation of classroom-style physics or geometry diagrams (e.g. (Chandrasekaran, Narayanan, & Iwasaki 1993)) and reasoning about board games (e.g. (Anderson 1996)). While a large variety of reasoning tools is readily available, the integration with visual domains normally has to be hand-crafted. A diagram meta-language which can be integrated with reasoning engines would greatly facilitate such work.

A logic-programming based approach seems an excellent candidate for such an integration and at the same time offers a basis for formally well-defined diagram specification and manipulation. Logic-based specification methods have already proven their suitability for linguistic representations and their principles can be extended into the domain of diagrams by making pictures a new domain for logic programming. The current paper explores the possibilities of such a logic-based framework. It presents a visual logic programming language for diagram handling, which is a diagrammatic language itself, and discusses its application to the definition and evaluation of visual mathematical notations.

A Method for the Formalization of Static Diagram Notations

Extensions of logic grammars have been used before to specify visual languages (e.g. in (Tanaka 1991; Marriott 1994; Ferrucci *et al.* 1991; Bolognesi & Latella 1989)). Our approach extends this idea and provides a more general integration of logic programming with visual expressions by integrating diagrammatic structures as first class data into a full constraint logic programming framework. This integration, called *picture logic*, has undergone several revisions in the last years. It was born as a visual set rewriting mechanism integrated into Prolog (Meyer 1992) based on picture matching instead of unification. Full picture unification (Meyer 1993; 1994) was added later to allow a tighter coupling with logic programming. Experiences with this formalism have shown that plain logic programming is only sufficient as the basis of the framework, when its usage is restricted to syntactic diagram specification. Used for the transformation of diagrams two shortcomings are revealed: (1) For flexible geometry handling a tighter coupling with arithmetics is required. (2) Dynamic changes in pictures can lead to temporary inconsistencies in a picture description which can only be resolved at a later derivation stage. Unfortunately, a deduction technique based on simple resolution refutation fails as soon as these inconsistencies occur and does not allow to resolve them at a later derivation stage. To solve these problems two extensions were made to the framework which are presented in this paper: (1) A transition from logic programming to constraint logic programming permits a close integration with arithmetics and thus allows to handle geometric properties properly. (2) A meta-programming technique is used to support transient inconsistencies which can be resolved at later processing stages.

We first review the basic framework. The general idea is to introduce a new kind of term structure for the description of diagrams into logic programming. These new terms, called *picture terms*, are diagrams themselves and have to be regarded as partially specified example pictures much like normal terms in logic programming are partially specified terms. Embedding these terms into a logic programming language we obtain a diagrammatic logic language for the specification of diagrams.

Picture terms consist of visual constants (picture objects) and visual variables (for picture objects).[2] Objects and variables in a picture term are in implicit spatial relationships that can be inferred from the depiction of the term. An underlying formal model for picture terms are graphs with typed object nodes and typed relationship edges according to the definition of a picture vocabulary $V = (OT, RT)$ of graphical object types OT and spatial relation types RT. Additionally a background variable (depicted as a solid frame around the term) can be used to denote an unknown context analogously to a list rest in normal logic programming. A second context variable, called frame (depicted as a dashed frame around the term) is used to denote the set of spatial relations between objects in the background and foreground objects. Assuming a vocabulary defined as $(\{circle, line, label\}, \{touches : circle \times line,\ attached : label \times line\})$, Figure 1 shows a picture term and the picture term graph to which it corresponds.

[2] We adopt the usual convention that variable names start with uppercase letters, constants with lowercase letters.

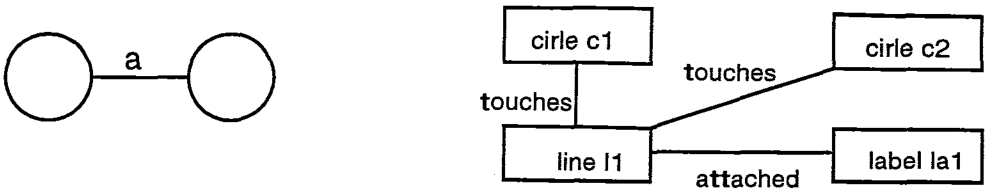

Figure 1: A Picture Term Graph

Every picture object can have an arbitrary number of attributes. A *circle* c, e.g., could have attributes $c.center \rightarrow point^2$ and $c.radius \rightarrow real$. Attributes can be any (possibly partial) data structure carrying additional information for geometry handling or interpretation purposes.

Picture terms are integrated with logic programming by defining a second kind of unification that is applied to picture terms. This unification performs a subgraph unification by finding a variable substitution π (called projection) for picture object variables that makes two picture term graphs identical up to some context which is contained in the background variable. It essentially solves the following simplified equation for π:

$$\pi(P \oplus B \oplus F) = \pi(P' \oplus B' \oplus F')$$

where $< P, B, F >$ and $< P', B', F' >$ are the picture terms to be unified. P (P') is the graph corresponding to the explicitly given picture objects and their relations (the foreground), B (B') is the graph corresponding to the background, the frame F (F') is the set of relation edges connecting nodes in P (P') with nodes in B (B'), π is the variable projection, and $\oplus$ is a merge operation for graphs. It is important to note that both context variables, background and frame, are partial data structures, i.e., an existent background or frame can be extended by new objects or relations during unification.

With picture unification, picture terms can be used in a logic program anywhere a normal term can be used. The rule in Figure 2, for example, is taken from a specification that defines the language accepted by a nondeterministic finite state automaton solely by applying visual transformations.

One advantage of such a specification is that it can be used for animation of diagrams without extra costs. If, for example, the instantiation of the first argument of *accept* is visualized for each inference step, then an animated execution of an NFA results.

Here is the point where geometry and thus arithmetic constraints come into play: When the above rule is applied, which moves the marker from one state to the next state, the actual coordinates of the new marker position are unknown. Since formally the transformation of the picture is only given by a transformation of the corresponding picture term graph, we only know that the point P is no longer inside of $C1$ but now inside of $C2$. Picture term graphs define only abstract spatial relations; nothing is known about the absolute coordinates which are needed if the term has to be visualized. Of course, in this trivial case it would be easy to make an assignment to the new coordinates of P by simply computing the center of $C2$, if its coordinates are known. However, in general this is not a good idea for two reasons: (1) Arithmetics is poorly integrated with normal logic programming and conflicts with backtracking, unbound variables, etc. (2) In more complicated cases it is not always possible to calculate geometric properties on the spot. In the above case, e.g., we might only know that P has to be inside of $C2$ but not where exactly it is located within $C2$. Putting it simply into the center might generate a layout that conflicts with other objects which could be put into $C2$ later. Therefore a more general form of integration with geometry is required.

The transition to a constraint logic framework enables us to achieve this by giving the spatial relations geometric, i.e. arithmetic, interpretations defined on relevant attributes of the involved objects. Every relation type in the vocabulary can either have an interpretation or be uninterpreted (in which case it is handled as a purely abstract relation object like above). An interpretation is defined by a normal CLP-clause involving arithmetic constraints. The *inside* : *point* × *circle* relation is, e.g., interpreted as:

```
interpretation(inside(P, C)) :-
    distance(P.center, C.center, D)
    & D + P.radius < C.radius.
```

This interpretation of *inside* enforces the fact that the point is completely contained in the circle, but does not give it a definite position. Interpretations can also be symbolic, not directly involving any constraints:

```
interpretation(left_touching(X,Y)) :-
    touching(X,Y) & left_of(X,Y).
```

During the unification of pictures the interpretations of relations are enforced. For every relation in a picture

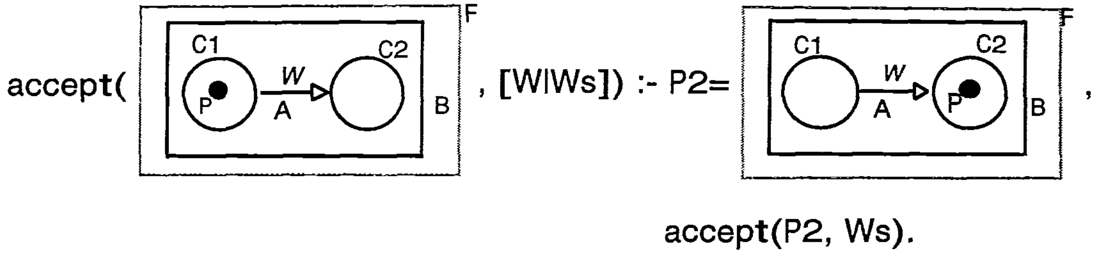

Figure 2: A Picture Logic Rule

that is the result of some unification the interpretation attached to this relation is evaluated. By this the unification is effectively constraining the values of the attribute variables of the involved objects. Our language is based on a logic programming language that directly handles constraints over real intervals, CLP(RI) (Older & Vellino 1990), so that we can leave the actual constraint solving entirely to the underlying LP language.

We now have introduced a new, additional source why a unification can fail: if the interpretations of some relations in the picture are not compatible, the underlying constraint solver will detect the inconsistency and reject the unification. For example, without interpretations, a picture term could well contain the relations $inside(X, Y)$ and $outside(X, Y)$ at the same time, because they are handled as "meaningless", unrelated predicates. With an appropriate interpretation the contradiction will be detected. In short, if every relation has an appropriate geometric interpretation no geometrically inconsistent pictures can occur.

Of course, the designer of a specification has to take care not to produce an over-constrained system of variables, since the derivation would then simply fail, as is the normal behaviour of a CLP-deduction. For an over-constrained system to produce a solution, some of the conflicting constraints have to be (automatically) relaxed. This can, for example, be achieved by extending the CLP-paradigm with hierarchical constraints such as in the HCLP-framework (Borning *et al.* 1989; Wilson & Borning 1993). As yet we have not investigated methods to support over-constrained specifications in our framework. A good overview of recent research into over-constrained systems can be found in (Jampel, Freuder, & Maher 1996).

Under-constrained systems, on the other hand, are of no harm to the derivation, but they do not produce sufficiently instantiated geometric variables for a concrete layout of the picture. If an under-constrained picture has to be displayed, the geometric attributes have to be instantiated with concrete values first. In the simplest case the underlying constraint solver can be forced to instantiate the geometric attributes with concrete values. Since the CLP(RI) solver works on real intervals, such a function can readily be achieved by an interval splitting algorithm and is indeed part of the CLP(RI) system. Of course, this will only produce some "random" solution that is consistent with the given constraints. There is no control over which layout is produced if several are possible. For more complicated cases specialized domain-specific layout modules have to be used that generate a "good" layout for an under-constrained picture. Such layout techniques are well investigated for some application areas, in particular for graphs. Since they are typically based on highly specialized algorithms, they can not readily be fully integrated with the basic framework and their use has to be confined to a dedicated output phase.

Problems in the Formalization of Visual Evaluation

We now take a closer look at the constraint-based extensions in the context of picture transformations. We have already said that a picture in which all relations are interpreted can no longer contain any inconsistencies. Unfortunately, there are cases where *temporary inconsistencies* are unavoidable or even desirable. These occur, for example, during picture transformations that are related to the evaluation of visual mathematical expressions.

When formalizing evaluation rules or consequence relations for visual mathematical notations, we often have to manipulate several graphical objects in a single step. In a rule-based model it is quite common for such transformations to be expressed in several rules each of which transforms only a part of the object set concerned. Intermediate transformation results, in which some of the objects are already modified, while others still have their original form, may well be meaningless, undefined, or even contradictory. In consequence, some kind of transaction concept is required, where the application of some set of transformation rules is regarded as an atomic transaction which may produce incorrect diagrams in intermediate steps but has to restore a well-defined state at its end.

We will use the visual λ-calculus VEX to illustrate this point. A complete discussion of VEX is beyond the scope of this paper and can be found in (Citrin, Hall, & Zorn 1995). A VEX-expression consists only of circles and lines. Textual labels may be used to improve readability, but they do not have any semantic meaning. In VEX an isolated circle represents a variable. A functional abstraction is represented by enclosing its body completely in another circle which does not touch any circles that belong to the body. The formal parameters of an abstraction are given as circles touching the abstraction circle from the inside.[3] Lines which connect circles are used to declare the identity of two graphical objects. A special rule for variables says that their corresponding circle has to be depicted on the level of graphical inclusion on which the variable is free. This graphical instance is called the root of a variable. Copies of the variable may appear in arbitrary positions of the expression and are connected to the root by identity lines. A function application is depicted by letting the applied function and the operand touch from the outside and drawing an arrow from the function abstraction to the actual parameter. The VEX expression in Figure 3 therefore represents the expression $(\lambda x.x)y$. A complete translation of VEX into textual λ-calculus can be specified in only eight simple picture logic clauses.

Evaluation of VEX expressions is defined by graphical transformations. Informally speaking, the graphical β-reduction rule for VEX is defined in the following way: (1) The arrow is redirected from the functional abstraction to the formal parameter, (2) the circle representing the functional abstraction is removed (3) the formal parameter and the actual parameter are merged and variable links are shrinked accordingly. The order of these steps is irrelevant. The complete β-reduction of $(\lambda x.x)y$ is therefore given in Figure 4.

Here we can observe two facts: (1) When performing the steps of a β-reduction separately, the intermediate states are not *semantically* well-defined diagrams and (2) intermediate steps obtained by the manipulation of single objects may even generate *geometrically* ill-defined pictures. Let us look at the second case more closely. Assume we are merging the formal parameter with the actual parameter before removing the circle of the functional abstraction. This is basically done by removing the formal parameter and substituting the actual parameter in its place. A naive approach to solve this problem is given by the rule in Figure 5. Remember that the two rectangles labeled B and F are not picture objects but the context variables for background and frame. We apply this rule to the VEX diagram given in Figure 3. Since picture unification automatically maintains the context relations between picture elements, the line from the root node y to the actual parameter will automatically follow the merging, because the actual parameter $C2$ to which it is connected is moved. Thus the pictures before and after the application of this rule are given by Figure 6. What happens to the line connecting the root node y and the actual parameter? Call this line l. A reasonable picture vocabulary for VEX contains the spatial relations *touches* : *line* $\times$ *circle*, *inside* : *line* $\times$ *circle*, *inside* : *circle* $\times$ *circle*, *outside* : *line* $\times$ *circle* and *outside* : *circle* $\times$ *circle*. Given this vocabulary, the picture before the rule application will contain the relations $outside(l, C1) \wedge touches(l, C2)$. After the rule application it will still contain both these relations, since l has not been changed explicitly. But in addition it will now contain $inside(C1, C2)$. The entire set of relations thus has become geometrically inconsistent: $outside(l, C1) \wedge touches(l, C2) \wedge inside(C1, C2)$. If the spatial relations are interpreted geometrically, and we have argued above that they have to be interpreted, this geometric inconsistency will be detected by the unification and the derivation will be rejected. The rule is therefore not applicable.

Note that this is only a local conflict that would be resolved at the end of the transformation once the functional abstraction circle is removed.

While one can find workarounds to avoid generating this simple inconsistency, this is not always possible in the general case. More importantly, the above rule expresses quite precisely and directly the transformation we have in mind so that it would be preferable if we can extend our framework to accommodate such temporary inconsistencies as long as a consistent state is restored at the end of a transformation.

A Method for the Resolution of Temporary Conflicts

We will now present an extension to picture logic that allows to detect and handle temporary inconsistencies. For the sake of a more concise discussion we will use a simplified example diagram language *boxes* in the following. The sentences of *boxes* consist of rectangles and circles. Circles are connected to a single "parent" rectangle by a line, and circles sharing the same parent may be interconnected by arrows. The desired transformation of *boxes* is to move all circles into their parent boxes while maintaining their interconnections. Thus the picture on the left hand side of

[3] We are using a pure version of VEX here that allows only abstractions over single parameters.

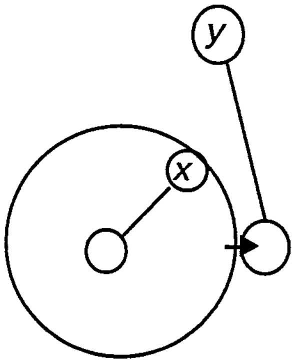

Figure 3: The VEX Expression $(\lambda x.x)y$

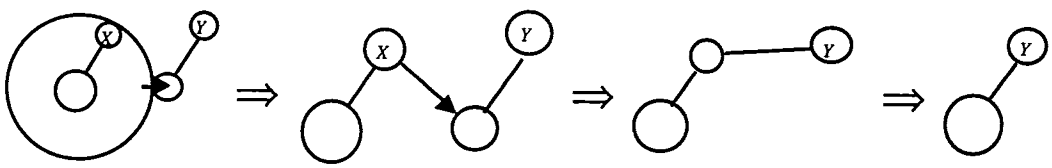

Figure 4: VEX β-Reduction

Figure 7 is a sentence of *boxes* and the right hand side is the desired transformation of this sentence. Like above a naive approach to formalize the transformation consists only of the two clauses in Figure 8 with the vocabulary $(\{circle, rect, arrow, line\}, \{startsat : arrow \times circle, endsat : arrow \times circle, inside : O_1 \times rect, outside : O_1 \times rect, attached : line \times O_2\})$, where $O_1 = circle \cup arrow \cup line, O_2 = rect \cup circle$. Since context relations are maintained automatically we do not need to worry about handling the arrows at all. Just swapping the boxes inside of their parent box, the arrows are following them automatically, since the *startsat* and *endsat* relations are maintained. However, we are facing the same problem as above. When moving the circles one by one the first rule application generates the state given in Figure 9. Thus the picture contains the original relations $outside(A, R) \wedge endsat(A, C)$ and the new relation $inside(C, R)$ which are contradictory.

The basic idea of how the problem of temporary conflicts can be solved is to define an extension of picture unification which can succeed despite of conflicts. For the unification to remain a consistent operation the conflicts have to be identified and removed from the picture term. Unification then has to yield the conflict set explicitly so that it can be handled by the remainder of the specification.

In order to achieve this we extend the notation of picture terms. Let P be some picture term, then the term $P\&C$ denotes the picture term and its associated conflict set C. Unification of two picture terms without conflict sets remains the same as above, but it behaves differently with conflict sets. First we have to analyze where conflicts can be detected: (1) Conflicts can only be detected during the evaluation of relation interpretations. (2) No conflicts can arise only from the evaluation of relations directly given in the foreground of some picture term. Since the foreground has a consistent geometric interpretation (it was given as a picture in the first place!), the relations in the foreground must have a consistent spatial interpretation. (3) Conflicts therefore have to be detected during the interpretation of relations in the background or frame.

If a conflict between two relations $r_1 \wedge r_2$ occurs and neither of them is in the foreground we have to define which of them is preferred, i.e., is kept in the picture term. The less preferred relation will be put into the conflict set. This is achieved by defining a partial order over the relation types in the vocabulary so that r_1 is preferred over r_2 if $r_1 < r_2$. For the purpose of our example it is sufficient to define *startsat, endsat* < *inside, outside*.

We can now extend unification in the following way: To evaluate $unif(PT, PT')$ for picture terms $PT =< P, B, F > \&C1$ and $PT' =< P', B', F' > \&C2$ the subsequent steps are performed:

1. Set the current conflict set $C := C1$

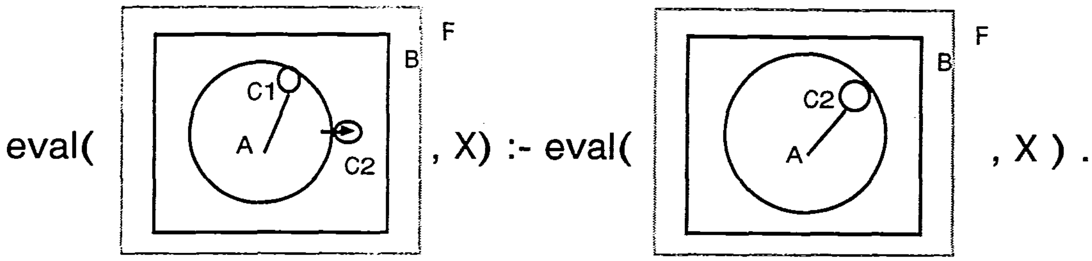

Figure 5: A Naive Approach to Define β-Reduction

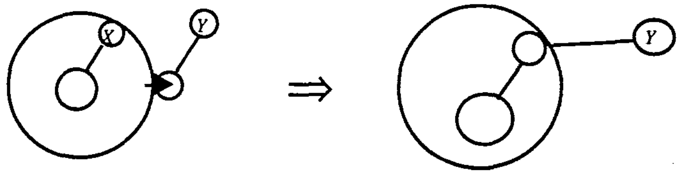

Figure 6: Diagram Instances before and after Rule Application

2. Find a projection π such that

$$\pi(P \oplus B \oplus F) = \pi(P' \oplus B' \oplus F')$$

   This is the usual picture unification without constraints.

3. Evaluate the interpretations attached to all the relations in P'.

4. For every interpreted relation r in B', in the sequence defined by their partial order, evaluate its interpretation. If the evaluation fails, a conflict is detected. In this case let $B' := B' - \{r\}$ and $C := C + \{r\}$.

5. Repeat the last step with F' in place of B'.

6. If $C2$ is a variable or $C = C2$ then succeed with $C2 := C$.

7. Otherwise fail.

Note that unification with conflict sets has become a directed operation. New conflicts can only be introduced in PT' but not in PT. The conflict set is an aggregative structure and $C2$ will contain all conflicts from $C1$ plus the new conflicts that were caused by the unification. A conflict-free picture can still be enforced by using an empty conflict set for the second argument. We now have simple means to realize the required "transaction" concept: during the transaction temporary conflicts can be gathered in a conflict set and at the end of the transaction a unification without conflict set is used to ensure that a consistent picture state has been restored.

Let us now look at the transformation of our example language *boxes* again. If we extend the first of the above transformation clauses with conflict sets like shown in Figure 10, it is easy to verify that after it has exhaustively been applied, the required transformation is achieved, but the conflict set contains the relations $outside(A, B)$ for all arrows A and their respective parent boxes B. Thus the original second clause, which has an empty conflict set, is not applicable. It would be possible to simply drop the conflict set when calling the second clause, but any control would be lost over whether the conflict set indeed contains only the anticipated conflicts. We therefore introduce a special predicate *resolve* for controlled conflict resolution. Its first and fourth arguments are picture terms together with their conflict sets, the second argument is a picture term describing a conflict item, and the third argument is a picture term describing how the conflict is resolved. *resolve* matches all relations which are given in the second argument with the conflict set of the first argument. If the match is successful it removes the matched relations from the conflict set of the first argument and adds all relations given in the third argument to the picture term of the first argument. The modified picture term and the modified conflict set are returned in the fourth argument. *resolve* fails, if the match of the relations in the second argument with the conflict set of the first argument fails. Using this predicate we can replace the original productions as shown

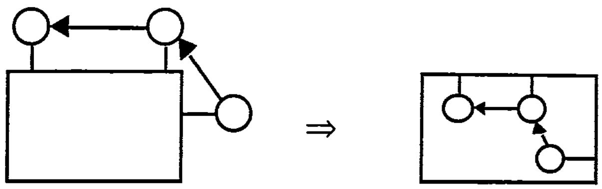

Figure 7: The Example Language *boxes*

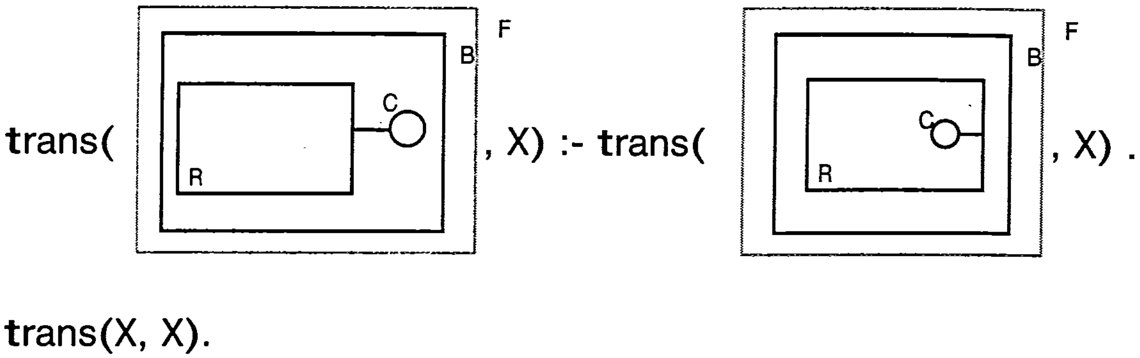

Figure 8: A First Approach to Formalize the *boxes* Transformation

in Figure 11.[4] *resolve* is used to move all arrows with *outside*-conflicts from the outside of the rectangle to the inside. The new production set will now perform the intended transformation and the derivation would still fail as required if additional conflicts would occur that have not been defined in the specification. In this way global consistency is still maintained.

As we can see from the abstracted example we now have the means to define diagram transformations like those that occur in the β-reduction of the visual λ-calculus VEX. The combination of constraint-based extensions and temporary conflict resolution therefore makes picture logic applicable to the definition and evaluation of the diagrammatic mathematical notations we were looking at.

Implementation

A complete implementation of basic picture logic as a fully interactive graphical system exists. The system consists of a Lisp-based frontend and a Prolog-based backend and serves as both, a compiler and a runtime environment. Diagrammatic input can be given with an object-oriented graphics editor or it can be sketched on a pen-tablet.

Since the underlying unification mechanism utilizes full graph unification (implemented by set partitioning), it is clear that high execution costs are involved. Particular attention has to be paid to the fact that picture unification is a non-deterministic operation, since it is impossible to define a unique most general unifier. While full picture unification is desirable for the use as a specification language, it is actually not utilized in most picture logic programs. In the cases we have explored, it is sufficient if unification deterministically yields a single unifier. Significant speed improvements have been achieved by introducing committed choice unification which implements this restriction.

Prototyped versions of the constraint-based extensions have been implemented in CLP(RI), but are not yet fully integrated with the interactive environment.

Since normal constraint logic programming languages are unable to delete constraints, we have to simulate this by meta-programming techniques which require frequent re-evaluation of constraints. We are currently exploring the integration of a constraint-solver that allows to delete constraints (Helm *et al.* 1995) into the system. Such an integration would promise significant performance improvements.

Related Work

There is a solid body of work on the formal specification of visual languages which mainly is based on grammars. A discussion is beyond the scope of this paper and can be found in (Marriott, Meyer, & Wittenburg 1997). In general, grammar-based methods appear un-

[4] The notation "P1:Picture-Term" used in the second clause introduces the variable *P1* as a shorthand for the depicted term.

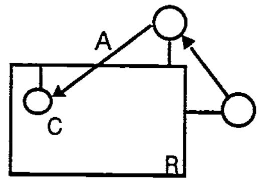

Figure 9: Intermediate State in the *boxes* Transformation

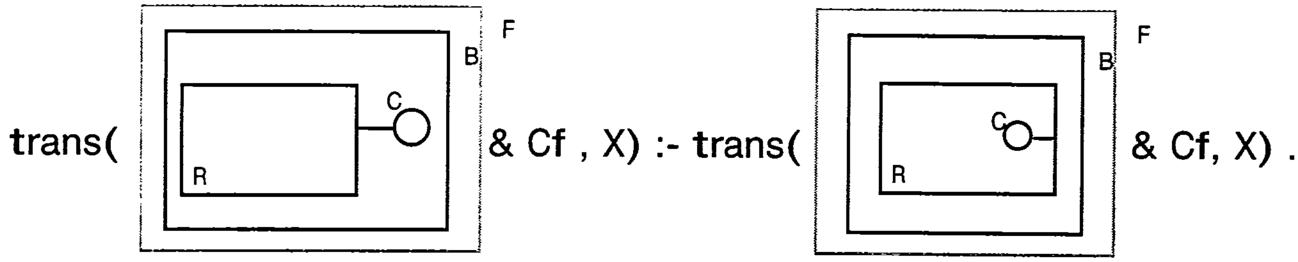

Figure 10: Revised Transformation Rule for *boxes*

suitable as a basis of reasoning, mainly for three reasons: (1) Grammars, especially context-free grammars, have limited expressiveness and are not suited for expressing reasoning processes directly. (2) Grammars are aimed at one-step processing tasks like parsing. Dynamic changes in diagrams are difficult to integrate. Such dynamic changes typically occur in diagrammatic reasoning when reasoning about sequences of diagrams like diagrammatic proofs or as a result of user interaction. (3) Grammars are difficult to integrate with general reasoning methods except for as isolated blackboxes which allow only very limited interaction with the other components.

In (Gooday & Cohn 1996) a spatial logic based on Clarke's point calculus (Clarke 1981) is presented and a specification of the visual programming language Pictorial Janus is given. This approach is aimed at the specification of execution, too, but only single execution steps can be specified, since the framework does not have an underlying notion of state or sequence. No computational implementation is discussed and this approach does not lend itself easily to an implementation, since it is based on full first order logic and point sets.

Another logic approach to diagram specification is presented in (Haarslev 1997). It is based on description-logic extended by concrete geometric domains and arithmetic constraint solving. While this system presents a powerful method for the specification and analysis of diagrams, it is not aimed at transformation or animation.

Logic-based approaches to visualization are either oriented towards generating layouts for static pictures (Kamada & Kawai 1991) or based on procedural notions of algorithm animation (Takahashi *et al.* 1994), whereas we are aiming at animation as an automatic side effect of the declarative specification of diagram languages.

None of the approaches discussed in this section is a diagrammatic language itself. We feel that it is important to explore diagrammatic languages as specification languages for visual notations, because diagrams as a tool for formal reasoning should be particularly useful when applied to spatial domains.

Conclusions

We have presented picture logic, a visual logic language for handling diagrams. Its major distinguishing features are that it uses the expressive power of visualization within the specification formalism itself and that the same specification framework can be used for specification, translation, transformation, and animation of diagrams. In contrast to grammar-based approaches, picture logic, being a full logic programming language, offers good opportunities for the integration with other reasoning methods.

This paper has in particular looked at the evaluation and animation of visual mathematical expressions as a special application area and has defined constraint-based extension of picture logic that allow to handle this domain.

In the same way consequence relations in visual logic notations like Peirce's α-β-calculus or Shin's extended Venn Diagram Systems should be formalizable. We

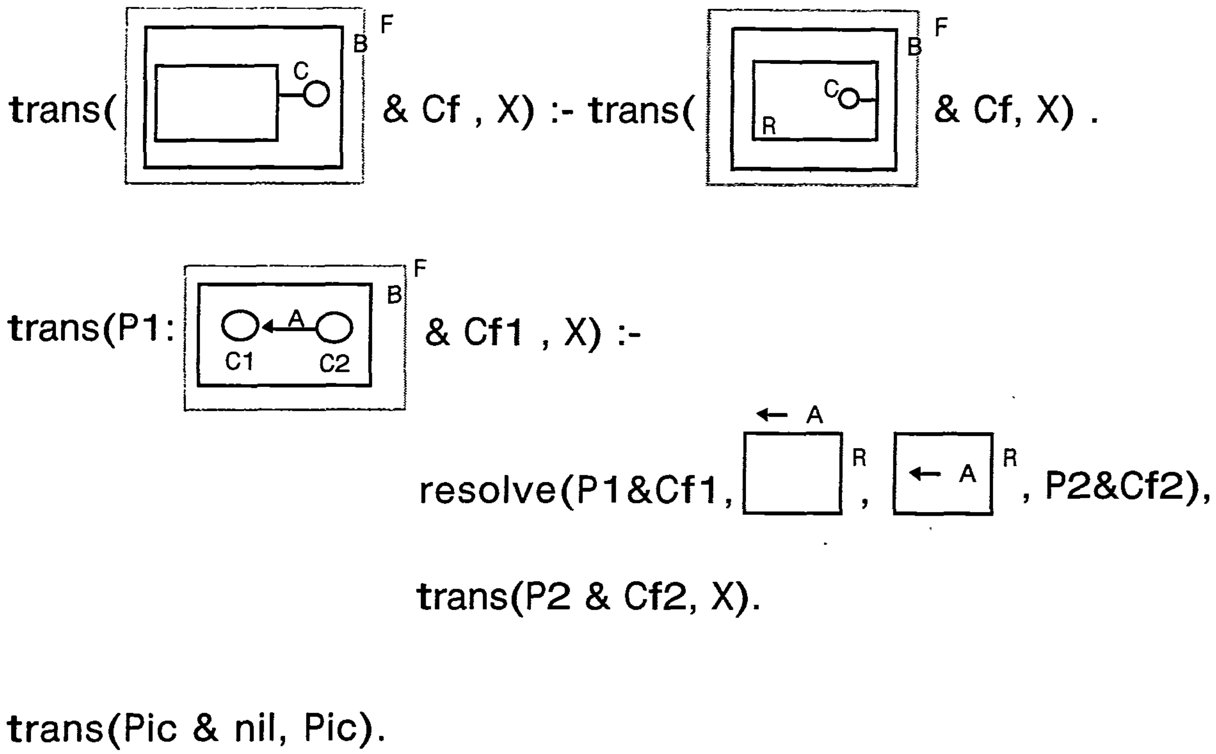

Figure 11: Resolving Anticipated Conflicts after the Transformation

will continue to investigate the usage of picture logic for the specification of such visual notations and hope that it will prove useful as a general meta-language.

Acknowledgments

The author gratefully acknowledges discussions with Kim Marriott during a stay at Monash University in which many ideas on diagram formalization were either born or substantially refined. During the time of writing the author was visiting the University of Colorado at Boulder supported by DFG Grant ME11/94.

References

Allwein, G., and Barwise, J. 1993. Working papers on diagrams and logic. Technical Report IULG-93-24, Indiana University Logic Group, Visual Inference Laboratory.

Anderson, M. 1996. Diagrammatic reasoning and cases. In *Thirteenth National Conference on Artificial Intelligence*.

Barwise, J. 1993. Heterogeneous reasoning. In Allwein, G., and Barwise, J., eds., *Working Papers on Diagrams and Logic*. Bloomington: Indiana University. 251 ff.

Bolognesi, T., and Latella, D. 1989. Techniques for the formal definition of the g-lotos syntax. In *IEEE Workshop on Visual Languages*, 43 – 49. Rom: IEEE Computer Society Press.

Borning, A.; Maher, M.; Martindale, A.; and Wilson, M. 1989. Constraint hierarchies and logic programming. In Levi, G., and Martelli, M., eds., *International Conference on Logic Programming*. MIT Press. 149 – 164.

Bricken, W. M. 1988. An introduction to boundary logic with the losp deductive engine. Research report, University of Washington.

Chandrasekaran, B.; Narayanan, N. H.; and Iwasaki, Y. 1993. Reasoning with diagrammatic representations - a report on the spring symposium. *AI-Magazine* 14(2):49 – 56.

Citrin, W.; Hall, R.; and Zorn, B. 1995. Programming with visual expressions. In *IEEE Workshop on Visual Languages*, 294 – 301. Darmstadt, Germany: IEEE Computer Society Press.

Clarke, B. 1981. A calculus of individuals based on 'conncection'. *Notre Dame Journal of Formal Logic* 23(3):204–218.

Ferrucci, F.; Pacini, G.; Tortora, G.; Tucci, M.; and Vitiello, G. 1991. Efficient parsing of multidimensional structures. In *IEEE Workshop on Visual Languages*, 105 – 110. Kobe/Japan: IEEE Computer Society Press.

Gooday, J., and Cohn, A. 1996. Using spatial logic to describe visual programming languages. *Artificial Intelligence Review* 10:171 – 186.

Haarslev, V. 1997. A fully formalized theory for describing visual notations. In Marriott, K., and Meyer, B., eds., *Theory of Visual Languages*. Springer-Verlag. Forthcoming.

Hammer, E., and Danner, N. 1993. Towards a model theory of diagrams. In Allwein, G., and Barwise, J., eds., *Working Papers on Diagrams and Logic*. Bloomington: Indiana University.

Hammer, E. 1993. Representing relations diagrammatically. In Allwein, G., and Barwise, J., eds., *Working Papers on Diagrams and Logic*. Bloomington: Indiana University.

Harel, D. 1988. On visual formalisms. *Communications of the ACM* 31(5):514 – 530.

Helm, R.; Huynh, T.; Marriott, K.; and Vlissides, J. 1995. An object-oriented architecture for constraint-based graphical editing. In *Object-Oriented Programming for Graphics*. New York: Springer-Verlag.

Jampel, M.; Freuder, E.; and Maher, M., eds. 1996. *Over-Constrained Systems*. Springer.

Kahn, K. M., and Saraswat, V. A. 1990. Complete visualizations of concurrent programs and their execution. In *IEEE Workshop on Visual Languages*, 7 – 15. Skokie/IL: IEEE Computer Society Press.

Kamada, T., and Kawai, S. 1991. A general framework for visualizing abstract objects and relations. *ACM Transactions on Graphics* 10(1).

Keenan, D. 1995. To dissect a mockingbird: A graphical notation for the lambda calculus with animated reduction. Technical report, Smalltalk Computing.

Marriott, K.; Meyer, B.; and Wittenburg, K. 1997. A survey of visual language specification and recognition. In Marriott, K., and Meyer, B., eds., *Theory of Visual Languages*. Springer-Verlag. Forthcoming.

Marriott, K. 1994. Constraint multiset grammars. In *IEEE Symposium on Visual Languages*. St. Louis/MO: IEEE Computer Society Press.

Meyer, B. 1992. Pictures depicting pictures - on the specification of visual languages by visual grammars. In *IEEE Workshop on Visual Languages*, 41 – 47. Seattle/WA: IEEE Computer Society Press.

Meyer, B. 1993. Logic and the structure of space: Towards a visual logic for spatial reasoning. In Miller, D., ed., *International Symposium on Logic Programming*. Vancouver: MIT Press.

Meyer, B. 1994. *Visual Logic Languages for Spatial Information Handling (in German)*. Doctoral thesis, FernUni Hagen.

Nelsen, R. B. 1993. *Proofs without Words*. Washington/DC: The Mathematical Association of America.

Older, W., and Vellino, A. 1990. Extending prolog with constraint arithmetic on real intervals. In *Canadian Conference on Electrical and Computer Engineering*.

Shin, S.-J. 1995. *The Logical Status of Diagrams*. Cambridge/MA: Cambridge University Press.

Takahashi, S.; Miyashita, K.; Matsuoka, S.; and Yonezawa, A. 1994. A framework for constructing animations via declarative mapping rules. In *IEEE Symposium on Visual Languages*, 314 – 322.

Tanaka, T. 1991. Definite clause set grammars: A formalism for problem solving. *Journal of Logic Programming* 10:1–17.

Wilson, M., and Borning, A. 1993. Hierarchical constraint logic programming. *Journal of Logic Programming* 16(3 & 4):277 – 318.