Transformational Design of Digital Systems based on Graph Rewriting

Corrie Huijs
University of Twente, P.O. Box 217, 7500 AE Enschede, The Netherlands
Tel.: +31 53 4893697, fax: +31 53 4894590, e-mail: chuijs@cs.utwente.nl

Abstract

Transformational design integrates design and verification. It combines "correctness by construction" and design creativity by the use of pre-proven behaviour preserving transformations as design steps. Transformational design is a formal design methodology in which formal aspects are hidden for the designer. Formal aspects of transformational design as a methodology for high-level synthesis, are discussed in this paper. Moreover graph rewriting theory is shown to be useful as a formal framework for transformational design. Transformations are defined as graph rewritings. Graph rewriting theory does not cover semantic aspects of graphs, which are useful as design representations because they visualise design information. A compositional formal semantics of design representations is essential in transformational design in order to prove the correctness of the transformations. This paper presents an extension of graph rewriting to attributed graphs as a suitable way to include semantical aspects. The used attribute algebra, table algebra, is a relation algebra derived from database theory. The combination of graph rewriting, table algebra and transformational design is new. Also new is the use of PVS (a theorem prover of SRI) to assist in the correctness proofs of the transformations.

Keywords: specification and verification, transformational design, graph rewriting, behaviour preserving transformations

I. Introduction

High-level synthesis deals with the derivation of RTL (Register Transfer Level) implementations from behavioural specifications. The correctness of high-level synthesis is of importance and needs to be guaranteed in order to eliminate costly design iterations. A design methodology based on "correctness by construction" is, because of the complexity of designs, preferred above design methodologies in which either simulation or verification is used to guarantee the design correctness. In high-level synthesis the creativity of the designer is of great influence and therefore exploitation of the designer's experience and insight needs to be possible. Transformational design incorporates *correctness by construction* and *interactive design* and therefore it not only leaves room for the designer's creativity but even stimulates it. C*orrectness by construction* in transformational design is achieved by building up the design process out of small design steps, transformations, which are proven to be correct previously. The designer gets a large set of correctness preserving transformations and selects, possibly supported by a transformational design system, which transformations will be used and in what order. In this design approach the decisions and their alternatives become clear which increases the insight of the designer [1] and stimulates his creativity. The feasibility of transformational design, especially for DSP (Digital Signal Processing) design applications, is shown [1, 2]. Transformational design is used for area, time as well as power optimisation objectives until now.

Transformational design is assumed to be formally well founded. Often more attention seems to be given to show its feasibility than to its formal aspects [3]. Correctness is related to the semantics of design representations, the specified behaviour. In order to prove transformations to be behaviour preserving the representation(s) on which the transformations are defined need to have a formal semantics. Only a few of the commonly used specification languages have such a formal semantics. The best approach for transformational design seems to be to define transformations on an intermediate design representation to which different specification languages can be converted [4]. This gives the opportunity to combine the use of different specification languages all having their own advantages and disadvantages. Graph representa-

tions are useful as intermediate design representations [4, 5, 6] because of their visualisation of design information. A formal semantic model for such graph representations, based on a relation algebra, is discussed in [7] and is used here.

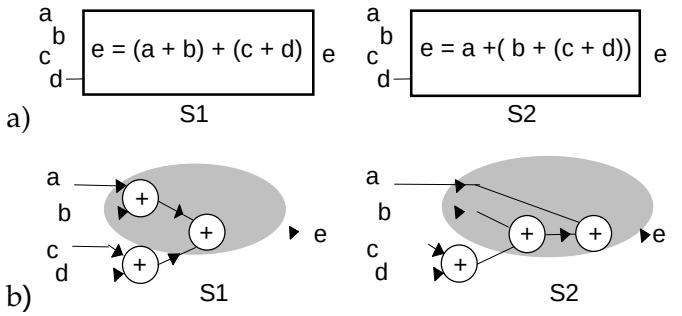


Fig.1: a) Two equivalent specifications of a simple system by the use of different algorithms
b) Graph representations of the algorithms

Transformations on graph representations are just replacements of selected subgraphs by new subgraphs. The algorithms of figure 1 can be transformed into each other by interchanging the subgraphs in the shaded areas. Graph rewriting is a mathematical theory developed during the last twenty years to describe the allowed replacements of subgraphs. It describes not only which subgraphs can be interchanged but also how they can be interchanged. In this paper the benefits are discussed of relating the definitions and formal aspects of transformational design to graph rewriting theory.

II. Formal aspects of Transformational design

As discussed by McFarland [4], the formal aspects of transformational design methodologies for digital systems are not always well defined and the correctness of transformations is often based on intuition instead of proofs. Proving the correctness of transformations is an essential element of the transformational design approach. These proofs not only ensure correctness of transformations but also support the understanding of the transformational design process as well as of the applicability of the various transformations. In order to prove the correctness of transformations a formal framework for transformational design needs to be defined. Formal aspects that have to be included in such a framework are:

- the system model used
- the correctness definition used
- formal semantics of design representation(s)
- formal definitions of transformations
- correctness proofs for transformations

These formal aspects and their relation with graphs and graph rewriting will be discussed.

III. The system model

System models are used to describe and/or analyse the essential characteristics of systems. These models are simplifications of reality and directed at and restricted to certain system characteristics. Correctness can only be discussed in relation to a certain model and therefore is always limited to the characteristics described by this specific model. A guarantee of correctness therefore only is meaningful if also the model to which it is related is defined.

Here we concentrate on modelling functionality for systems that can be implemented as deterministic synchronous digital systems. A combination is used of a RTL model and a black box specification model. In the RTL model structure is described by *signal flow graphs* and behaviour is a derivative. In the specification behaviour is essential but structure can be of importance too. Specifications are given by algorithms which inherently include structure that can be used as implementation suggestion (see figure 1).

Essential assumptions at the specification level are related to the communication with the environment. Only one input stream and one output stream are assumed, of which the first is independent of the interaction between the system and its environment. Note that several input streams can be combined into one in case the streams are synchronised.

A specification is supposed to prescribe the *observable behaviour*. The *observable behaviour* is the relation between the values of elements of the stream of inputs and elements of the stream of outputs. Each element of the output stream is allowed to depend on the history determined by a finite (statically determined) part of the stream of inputs. The *history dependence* is modelled by state variables like in FSM models.

Determinism is related to an initial state of the system. Each time when starting from this initial state the same stream of inputs is given also the same stream of outputs has to be delivered by the system. Determinism corresponds with monotonicity: if an input stream is the head of another input stream its output stream is the head of the other output stream. Although the system has to be de-

terministic we allow the specification of the system to be non-deterministic. Non-determinism is used to describe design freedom.

Concurrency is a hardware characteristic that is important in the RTL model and which is nicely modelled by data flow graphs. Data flow graphs model operations on data and communication of data between operations. Operations correspond with combinational components and the communication with wires in RTL descriptions. Data flow graphs can be defined *hierarchically* which is of importance to handle complexity. For the modelling of synchronous systems we use a variation on data flow graphs: *signal flow graphs*. The reason for this is that data flow graphs model asynchronous systems by using tokens to model activity. In synchronous systems a more abstract model of activity can be used: all components can be said to be active exactly once each clock period. This is based on the fundamental assumption that the system clock of a synchronous system is chosen such that all values in the system stabilise each clock period (obligation for the designer). The stabilisation of values is modelled by a *single assignment model* for *signal values* which replaces the token model of data flow graphs. Signals are streams of values related to a certain clock representing the stable values and modelled by functions on natural numbers. Signal flow graphs model operations on signals; operations are repeated every clock period. Control flow is integrated in our model by the use of conditional execution of operations.

IV. The Correctness Definition

Observable behaviour, the data relation between inputs and outputs, needs to be preserved in the design process. A non-deterministic specification of this behaviour has to be transformed into a system with deterministic behaviour. As a consequence the definition of correctness of transformations needs to include behaviour equivalence of design representations as well as behaviour implication of one design representation to another. In this paper the concepts of our approach are illustrated for correctness based on behaviour equivalence only.

In transformational design it is essential that the behaviour of a system is a composition of the behaviours of its subsystems. Only then replacing a subsystem by another with equivalent behaviour can guarantee not to influence the behaviour of the total system. This is called *compositionality* and delivers a constraint for the formal semantics.

Note that compositionality of behaviour also means that behaviour may not depend on the environment.

V. Transformations as Graph Rewritings

In this section a short and informal introduction to graph rewriting theory is given. Transformations are defined as graph rewritings and preconditions for their syntactical correctness are presented. First of all a suitable definition of graphs is discussed.

A. Definition of Graphs

Graph rewriting theory is based on mathematical directed graphs. In order to benefit from this theory graphical design representations therefore are related to these mathematical directed graphs. A mathematical graph is a composition of two kinds of objects: nodes (or vertices) and edges (or arcs). Graph representations suitable as intermediate representations in high level synthesis are flow graphs, compositions of operators and data communications. Operations are defined on tuples, ordered sets, of data values and can not be represented by elements of simple mathematical graphs.

Directed hypergraphs are directed graphs in which the edges, called hyperedges, are allowed to have more than one source and more than one target. Sources and targets of a hyperedge are given by lists of nodes. The ordering of sources and targets is explicitly part of the structure of hypergraphs. This corresponds nicely with operand tuples of operations. Therefore hyperedges can be usefully related to the operators in flow graphs. Where different operators appear in flow graphs the possibility to distinguish different kinds of hyperedges is needed. Labelling is used for this purpose. Labels relate hyperedges to their semantics.

A graphical representation of the edge labelled directed hypergraphs is shown in figure 2. It is important to notice that the *hyperedges* in this paper correspond with what are often called "*nodes*" in flow graph representations. Because of this choice useful transformations on flow graphs can be related to edge replacements [9] in graph rewriting theory.

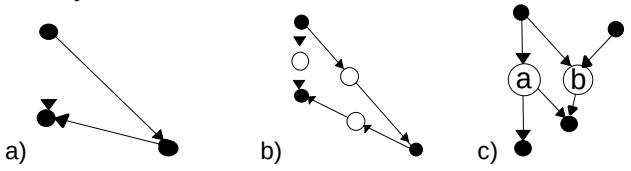


Figure 2:

a) "Usual" representation of a simple mathematical graph (● = node; → = edge)

b) Representation of the same graph defined as directed hypergraph (● = node; O=hyperedge; → source and target relation)

c) A "real" and labelled hypergraph : an edge, labelled *b*, with more than one source and an edge, labelled *a*, with more than one target.

Definitions:

1. When X is a set, X^* is the set of all finite tuples (lists) of elements of X
2. An ***edge labelled directed hypergraph*** is a 5-tuple $G =< N, E, s, t , lab>$ in which
   N is the set of nodes and
   E the set of directed hyperedges
   $s : \quad E \rightarrow N^*$ a function that gives lists of sources of the hyperedges
   $t : \quad E \rightarrow N^*$ a function that gives lists of targets of the hyperedges
   $lab: \quad E \rightarrow LABS$ a function that gives the labels of the hyperedges
3. A ***graph morfism*** of an edge labelled directed hypergraph G to an edge labelled directed hypergraph G' is a tuple $m= <m_N, m_E>$ such that $m_N : N \rightarrow N'$, $m_E : E \rightarrow E'$ and (see figure 3):
   $s' \circ m_E = m_N \circ s$, $t' \circ m_E = m_N \circ t$, $lab' \circ m_E = lab$

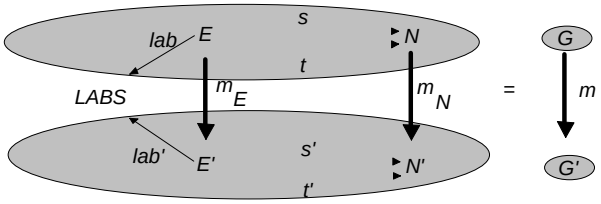


Fig. 3. Commutative diagram for a graphmorfism

B. Graph Rewritings

Graph morfisms are used to formally define the selection of a copy of a (hyper)graph in another (hyper)graph. Graph morfisms preserve structural properties including labels: edges are mapped into edges and nodes into nodes. Moreover the sources of the image of an hyperedge are the images of the sources of the hyperedge and similar for targets. This structure preservation is said to correspond with the condition that the diagram in figure 3 has to be commutative.

Transformation rules, productions in graph rewriting theory, are defined by 3 graphs *L, K, R* and 2 graph morfisms *l* and *r*. *L* corresponds with the graph to be replaced and is called *left hand side. R* corresponds with the replacing graph and is called *right hand side. K* is called the *interface graph* and corresponds to the "boundary" along which the copy of graph *L* is cut out of a graph and along which the copy of *R* is glued into

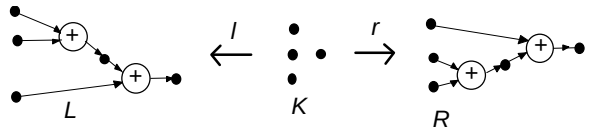


Fig. 4. Graphical representation of a transformation rule describing associativity of addition.

the same graph. Figure 4 gives an example of the graphical representation of such a transformation rule: the transformation rule describing associativity of addition.

A transformation is the application of a transformation rule (figure 5). The application of a transformation rule <*L,K,R,l,r*> on hypergraph *G* is given by specifying a graph morfism $g: L \rightarrow G$ which "selects" a copy of *L* in *G: g(L).* This copy of *L* has to be replaced by a copy of *R*. The interface graph *K* is used to prescribe how the replacement has to be done. *D* is the *context graph*, *G* after removing the copy of *L* but not the interface:

$$D = G - g(L-l(K))$$

Notice that *D* is a well-defined graph because of including the $(g \circ l)$-image of the interface graph *K*. There is a small but important difference between the context $G - g(L)$ and the context graph *D*. The interface graph is essential because of this.

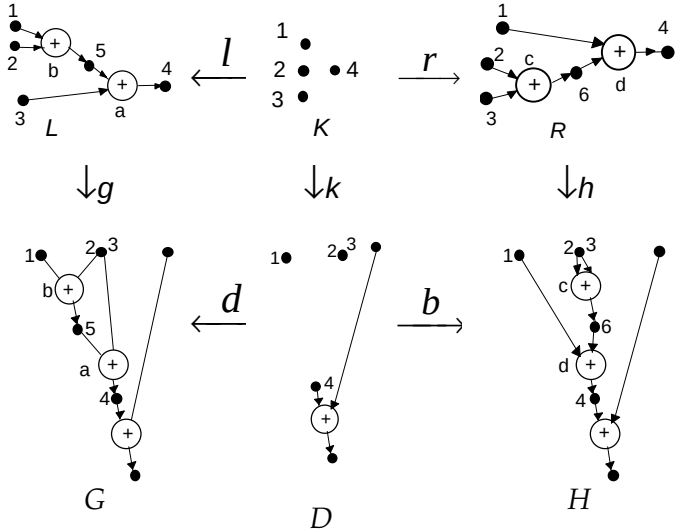


Fig. 5. A transformation defined as graph rewriting: (All graph morfisms are represented by numbers and characters. Nodes (edges) with the same number (character) are mapped onto each other).

C. Syntactical Correctness of Rewritings

The algebraic specification of graph rewriting as discussed above and defined by Ehrig [9] is useful in proofs. Of special importance is the *gluing condition* which defines preconditions for the syntactic correctness of the application of transformation rules [9]. Informally explained the gluing condi-

tion states that elements of *L* not belonging to *the gluing l(K)* may not have the same g-image as another element of *L*. It also states that an *internal node* of *L*, an element of *L-l(K)*, may not be a source or a target of an hyperedge in the context: *G-g(L)*. Figure 6 illustrates two syntactical incorrect applications (not satisfying the gluing condition) of the transformation rule of figure 4.

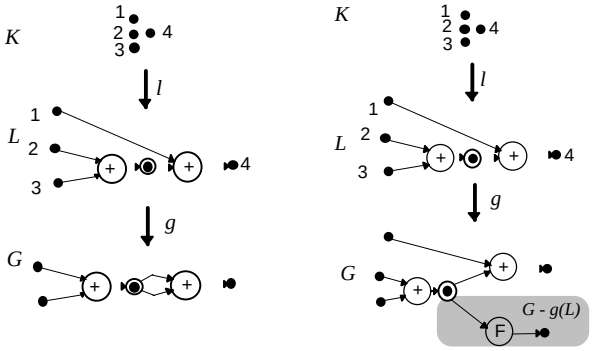


Fig. 6. Two applications of the transformation rule of figure 4 which do not satisfy the *gluing condition.* ◉ represent the elements of *L* causing problems and their *g*-images.

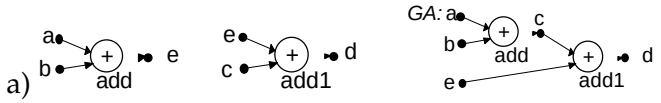
VI. FORMAL SEMANTICS OF (LABELLED) GRAPHS

The semantical aspects of the above used graphs are related to the labels. These labels are references to the behaviour of the hyperedges. This reference can be used in a formal semantics based on attributed graphs. Attributed graphs are graphs in which attributes are assigned to elements of the graphs:

Definition:

An ***attributed graph*** is a graph *G* together with one or more (attribute)functions of type $E_G \rightarrow \Sigma_1$ or type $N_G \rightarrow \Sigma_2$ for some algebra's Σ_1

In case the semantics of hyperedges is defined by the attributes assigned to them, the attribute function corresponds to a valuation function of a denotational semantics [10]. The attribute algebra becomes a semantic algebra and a formal semantics is determined. This means that a formal semantics can be based on attributed graphs together with a suitable attribute algebra.



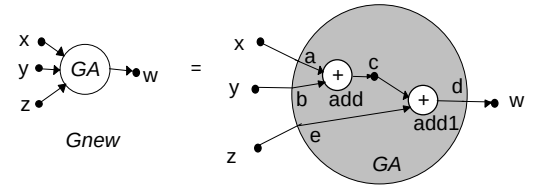


Fig. 7. a) graph *GA* as composition of two subgraphs
b) The use of GA to define an hierarchical hyperedge in Gnew

A.Formal Semantics based on Table Algebra

The observable behaviour specified by a hypergraph is a data-relation on its external nodes, its boundary. The formal semantics defines a mathematical representation of observable behaviour as well as its composition from the behaviour of objects in the hypergraph. *Tables* are defined as representations of data-relations and can be viewed as a generalisation of the truth table concept. Three operations are defined on tables: the *natural join* ⋈ as composition operator, the *restriction* ` as projection operator and the *renaming operator* ∞. The use of the obtained table algebra in the definition of a formal semantics is informally illustrated by discussing the derivation of the observable behaviour of hypergraph *Gnew* in figure 7.

A *table* is a set of functions with a common domain [11]. This common domain is called *the heading* of the table. If a table is used as representation of the behaviour of a hyperedge then the heading of the table consists of the nodes to which this hyperedge is connected. A table corresponds with a predicate on functions (specifying whether the function represents an allowed combination of data-values to be observed at the nodes). For example the behaviour of hyperedges instantiating addition operations, *add* and *add1* of figure 7, can be given by:

$$Table(add) = \{ f \mid [dom(f) = \{a,b,e\}] \wedge [f(a) \in [1,2]] \wedge [f(b) \in [1,2]] \wedge [f(e) \in INT] \wedge [f(e) = f(a) + f(b)] \}$$

$$Table(add1) = \{ g \mid [dom(g) = \{c,e,d\}] \wedge [g(c) \in [1,2]] \wedge [g(e) \in [1,2]] \wedge [g(d) \in INT] \wedge [g(d) = g(c) + g(e)] \}$$

These tables can also be given graphically which makes clear why these sets of functions are called tables. Each function in the set corresponds with a row in the graphical table:

Table(add)=

a	*b*	*c*
1	1	2
1	2	3
2	1	3
2	2	4

Table(add1)=

c	*e*	*d*
1	1	2
1	2	3
2	1	3
2	2	4

The behaviour of hypergraph *GA* is the composition of *Table(add)* and *Table(add1):*

$Table(GA) = Table(add) \bowtie Table(add1) =$

a	*b*	*c*	*e*	*d*
1	1	2	1	3
1	1	2	2	4

The natural join $\bowtie$, known from database theory [11], corresponds with the conjunction of predicates. The natural join combines function elements, rows, of tables into a function element, row, of a new table. Functions can only be combined if for shared domain elements the original functions deliver the same function value. In case of the natural join of *Table(add)* and *Table(add1)* the function value of *c* needs to be 2.

The observable behaviour of *GA* is the data relation on its external nodes: {*a, b, e, d*}. This observable behaviour is obtained by projection $\backslash$, restricting each element of *Table(GA)* to the external nodes:

observable behaviour of GA=

$Table(GA) \backslash \{a, b, e, d\} =$

a	*b*	*e*	*d*
1	1	1	3
1	1	2	4

GA is used as specification of the behaviour of a hyperedge. The attributes of the above given table, {*a, b, e, d*} become formal parameters and are renamed by ∞, into the sources and targets of the hyperedge:

Table(Gnew)=

$Table(GA) \infty \{<x,a>,<y,b>,<z,e>,<w,d>\} =$

x	*y*	*z*	*w*
1	1	1	3
1	1	2	4

VII. CORRECTNESS PROOFS OF TRANSFORMATIONS

The observable behaviour of the representation of a system may not be changed by transformations. Because of the compositionality of the formal semantics it is sufficient to prove that the behaviour of the new part is equivalent to the behaviour of the replaced part. The semantical correctness of the transformations can be proven to be based on semantical correctness of the transformation rules, that still need to be proven, and *the gluing condition*. Because of the large number of transformation rules a tool that supports the proofs of semantical correctness is needed. PVS (Prototype Verification System of SRI) [8] is shown to be useful in supporting correctness proofs of our transformation rules [12]. Because PVS is based on interactive proving it makes clear the problems in the proofs. Proofs that could not be given often resulted in more insight than those which could be proven easily.

VIII. CONCLUSIONS

Graph rewriting is a suitable basis for a formal framework for transformational design on graphical design representations. A transformation defined as graph rewriting is split into a *transformation rule* and an application of this rule. Graphs describe structure. The extension of graph rewriting to attributed graphs is a useful way to integrate semantical aspects with structure described by graphs. The semantical correctness of transformations is nicely based on the combination of semantical correctness of the transformation rule and *syntactical correctness* of the application. The *gluing condition* from graph rewriting theory unifies constraints for syntactical correctness of transformation rule applications. The *interface graph* is essential in the definition of transformations as well as in the gluing condition. It prescribes the way subgraphs have to be connected (interfacing) with their environment.

Table algebra is suitable as attribute algebra and therefore also as semantic algebra of a denotational semantics of the design representation. It is nicely intuitive but also formally well defined.

IX. REFERENCES

[1] P.F.A. Middelhoek, G. E. Mekenkamp, E. Molenkamp, Th. Krol, *A Transformational Approach to VHDL and CDFG Based High-Level Synthesis: a Case Study*, Proc. of CICC 95, pp. 37-40, Santa Clara, Ca, May 1995. (http//wwwspa.cs.utwente.nl/aid/trades/trades.html)

[2] A. P Chandrakasan, M. Potkonjak, et al. *Optimizing power using transformations*, IEEE Trans. on CAD, Vol. 14. No.1 January 1995, pp. 12-31.

[3] M.C. McFarland SJ., *Formal Analysis of Correctness of Behavioral Transformations*, in: Formal Methods in System Design Vol. 2, Number 3, June 1993, pp 231-257.

[4] Th. Krol et al, *The SPRITE Input Language, An intermediate format for High Level Synthesis*, in: Proc. of EDAC 92, Brussels, 16-19 March 1992, pp. 186-192.

[5] R. Camposano, *Behaviour-preserving transformations for high-level synthesis*, in: M. Leeser and G. Brown *Hardware Specification, Verification and Synthesis: Mathematical Aspects*, LNCS 408, (Springer-Verlag, Berlin Heidelberg,1989) pp.106-127.

[6] Z.Peng and K. Kuchinski, *Automated Transformation of Algorithms into Register-Transfer Level Implementations*, IEEE Trans. on CAD Vol. 13, No. 2. Feb. 1994, pp. 150-166.

[7] C. Huijs and Th. Krol, *Relational Semantics for Flow Graph Representations as basis for Transformational Design of Digital Systems*, in: Proceedings of the workshop on Design Methodologies for Microelectronics and Signal Process-

ing, Gliwice-Cracow Poland, 20-23 October 1993, pp. 21-29.
(http//wwwspa.cs.utwente.nl/aid/trades/trades.html)

[8] S.Owre, N. Shankar and J.M. Rushby, *Userguide for the PVS specification and Verification system*, Computer Science Laboratory, SRI, Menlo Park, CA, Feb. 1993.

[9] H. Ehrig, M. Korff, M. Löwe, *Tutorial introduction to the algebraic approach of graph grammars based on double and single pushouts,* in H. Ehrig, H.-J. Kreowski, G. Rozenberg(Eds) *Graph-Grammars and their Application to Computer Science* , LNCS532, (Springer-Verlag, Berlin Heidelberg,1991) pp.24-37.

[10] P.D. Mosses, *Denotational Semantics*, in: J. van Leeuwen eds.,*Formal Models and Semantics, Handbook of Theoretical Comp. Sc. Volume B* (Elsevier Science Publishers, North-Holland,1990) pp. 577- 631.

[11] E.O. de Brock, *Foundations of Semantic Databases*, Prentice Hall, 1995.

[12] E.W. Prangsma, *Judging the Suitability of PVS for Proving Transformations on SIL*, Master Thesis Computer Science, May 1996, University of Twente, SPA-96-05.