Applied Mathematical Sciences, Vol. 2, 2008, no. 18, 861 - 881

Demonic Operational and Denotational Semantics

Fairouz Tchier

Mathematics Department, King Saud University
P.O. Box 22452, Riyadh 11495, Saudi Arabia
ftchier@hotmail.com

Abstract

Over the years, different approaches to give semantics of programming and specification language have been put forward. We restrict ourselves to the operational and the denotational approach, two main streams in the field of semantics. We use relational formalism and we consider the worst execution of the program i.e we suppose that the program behaves as badly as possible it's the *demonic relational semantics* . The principal aim of this paper is to show that relational demonic operational semantics of a nondeterministic program is equal to its demonic denotational semantics.

Keywords: Operational semantics; Denotational semantics; Demonic semantics, Relational semantics; Nondeterminism; Nontermination

Our mathematical tool is abstract relation algebra [17, 37, 39].

(1) **Definition.** A *(homogeneous) relation algebra* is a structure $(\mathcal{R}, \cup, \cap, \overline{\ }, \breve{\ }, \circ)$ over a non-empty set $\mathcal{R}$ of elements, called *relations*. The following conditions are satisfied.

- $(\mathcal{R}, \cup, \cap, \overline{\ })$ is a complete atomic Boolean algebra, with *zero* element $\emptyset$, *universal* element L and ordering $\subseteq$.
- *Composition*, denoted by $(\circ)$, is associative and has an identity element, denoted by I.
- The Schröder rule is satisfied: $P \circ Q \subseteq R \Leftrightarrow P^{-1} \circ \overline{R} \subseteq \overline{Q} \Leftrightarrow \overline{R} \circ Q^{-1} \subseteq \overline{P}$.
- $L \circ R \circ L = L \Leftrightarrow R \neq \emptyset$ (Tarski rule).

The precedence of the relational operators from highest to lowest is the following: $\overline{\ }$ and $\breve{\ }$ bind equally, followed by $\circ$, then by $\cap$, and finally by $\cup$. From now on, the composition operator symbol $\circ$ will be omitted (that is, we write QR for $Q \circ R$).

We now give useful definitions. For more details (see, e.g., [9, 17, 37]).

(2) **Definition.**

(a) A relation R is *functional* iff $R^{-1}R \subseteq I$.

(b) A relation v is a *vector* [37] iff $v = vL$.

(c) A relation a is a *partial identity* [41] iff $aa^{-1} \subseteq I$.

(d) A relation x is a *point* [37] iff $x = xL$ and $xx^{-1} \subseteq I$.

In our work we need to define an operator called *relative implication*. In previous work, we used the monotype and residual operators see [44, 45, 46, 47]

(e) A binary operator $\triangleleft$, called *relative implication* [41], is defined as follows :

$$Q \triangleleft R := \overline{Q\overline{R}}.$$

The operator $\triangleleft$ binds less than $(\circ)$ and more than $\cap$ and $\cup$.

We need to define the notions of direct sum and the direct product.

(3) **Definition.**

(a) A pair (σ_1, σ_2) of relations is called *direct sum*[7, 8, 19] iff :

$\sigma_1\sigma_1^{-1} = I, \quad \sigma_2\sigma_2^{-1} = I, \quad \sigma_1\sigma_2^{-1} = \emptyset, \quad \sigma_1^{-1}\sigma_1 \cup \sigma_2^{-1}\sigma_2 = I.$

Relations σ_1 and σ_2 are called *injections.*

The next definition introduces the notion of the disjoint union of two relations.

(b) Let (σ_1, σ_2) be a direct sum. The relation $\langle R_1, R_2\rangle := \sigma_1^{-1}R_1\sigma_1 \cup \sigma_2^{-1}R_2\sigma_2$ is called the *disjoint union* of relations R_1 and R_2 with respect to (σ_1, σ_2).

(c) A pair (π_1, π_2) of relations is called *direct product* iff [7, 8, 19]

$\pi_1^{-1}\pi_1 = I, \ \ \pi_2^{-1}\pi_2 = I, \ \ \pi_1\pi_2^{-1} = L, \ \ \pi_1\pi_1^{-1} \cap \pi_2\pi_2^{-1} = I.$

Relations π_1 and π_2 are called *projections.*

In what follows, we will define the cartesian product of relations. For more details see [19].

(d) Let (π_1, π_2) be a direct product and R_i, $1 \leq i \leq 2$ relations. The cartesian product of relations R_i with respect to (π_1, π_2) is :

$[R_1, R_2] := \pi_1 R_1 \pi_1^{-1} \cap \pi_2 R_2 \pi_2^{-1}$.

(e) Another operation that occurs in the definition of the while program semantics is the *reflexive transitive closure*. The *reflexive transitive closure* is an unary operation denoted $*$ and defined for every relation R by : $R^* = \bigcup_{i \geq 0} R^i$, where $R^0 = I$ and $R^{i+1} = RR^i$.

The unary operations $*$, $^{-1}$ and $\bar{}$ bind equally.

In the following, we describe notions that are useful for the description of the set of initial states of a program for which termination is guaranteed. These notions are *initial part* and the *progressive finiteness* of a relation. The algebraic definitions are

(f) The *initial part* [37] of a relation R, denoted $\mathcal{I}(R)$, is given by :

$$\mathcal{I}(R) := \bigcap \{x \mid R \triangleleft x = x\},$$

where x takes its value in the set of the vectors (by Definition 2(b), $\mathcal{I}(R)$ is a vector). See [36, 37]); in other words, $\mathcal{I}(R)$ is the least fixed point of the $\subseteq$-monotonic function $g(x) := R \triangleleft x$, where x is a vector (the least fixed point of g exists since the set of vectors is also a complete lattice [37]).

(g) A relation R is said to be *progressively finite* iff $\mathcal{I}(R) = L$, in other words if there is no infinite path by R. Progressive finiteness of a relation R is the same as well-foundedness of R^{-1}.

(Mnemonics : $\mathcal{I}(R)$ represents the set of states from which no infinite loop is possible.)

1 Relational diagrams

In the following, we will give the formal definition of a *diagram* and different types of diagrams. For more details, see [47, 45]

(4) **Definition.** Let $\mathcal{A}$ be a homogeneous relational algebra.

(a) A quadruple $\mathcal{P} = (P, C, \varepsilon, \xi)$ is a *diagram* on $\mathcal{A}$ iff,

- P is a relation of $\mathcal{A}$, called the *associated relation* of diagram $\mathcal{P}$,
- C is a set of partial identities (Definition (2)c) disjoint from each other, verifying the condition : $(\cup C)P(\cup C) = P$,

- ε (*entry*) and ξ (*sortie*,which means exit in french) are partial identities
  $(\varepsilon, \xi \in C)$ called respectively the *input relation* and the *output relation* of the diagram $\mathcal{P}$.

(b) A diagram $\mathcal{P}_1 = (P_1, C, \varepsilon_1, \xi_1)$ is a *sub-diagram* of diagram $\mathcal{P} = (P, C, \varepsilon, \xi)$ iff : $P_1 \subseteq P$ and $((\cap C) \cap \overline{\varepsilon_1 \cup \xi_1})P = P((\cup C) \cap \overline{\varepsilon_1 \cup \xi_1}) = \emptyset$.

We distinguish two types of diagrams : *elementary* and *compound* diagrams.

(a) A diagram $\mathcal{P} = (P, C, \varepsilon, \xi)$ is *atomic* iff $C = \{\varepsilon, \xi\}$ and $P = \varepsilon P \xi$.

An atomic diagram consists of a unique atomic step *i.e.* the transition between the input node and the output node is formed in exactly one step.

(b) A diagram $\mathcal{P} = (R, C, \varepsilon, \xi)$ is a *sequence diagram* iff :

$P = P_1 \cup P_2, \quad C = \{\varepsilon, a, \xi\}, \quad P = \varepsilon P_1 a$ *and* $P_2 = a P_2 \xi$.

(c) A diagram $\mathcal{P} = (R, C, \varepsilon, \xi)$ is a *branching diagram* iff

$R = G_1 \cup P \cup G_2 \cup Q, \quad C = \{\varepsilon, a, b, \xi\}, \quad G_1 = \varepsilon G_1 a, \quad P = aP\xi,$
$G_2 = \varepsilon G_2 b$ and $Q = bQ\xi$.

(d) A diagram $\mathcal{W} = (W, C, e, s)$ is a *loop diagram* iff

$W = P \cup Q, \quad P = \varepsilon P \varepsilon, \quad Q = \varepsilon Q \xi$ and $PL \cap QL = \emptyset$.

The graphs and the matrices representing the different diagrams $\mathcal{P}$ can be found in [41, 43]

In a loop diagram, P is applied until Q can be applied.

(e) A diagram is *elementary* if it is an atomic, a sequence, branching or a loop diagram.

(f) A diagram is *compound* if it is not elementary.

(5) **Lemma.** *Let $\mathcal{P} = (P, C, \varepsilon, \xi)$ is a diagram and σ a relation such that $\sigma\sigma^{-1} = I$ and $\sigma^{-1}\sigma \subseteq I$. Then $\mathcal{P}_\sigma := (\sigma^{-1}P\sigma, \sigma^{-1}C\sigma, \sigma^{-1}\varepsilon\sigma, \sigma^{-1}\xi\sigma)$, where $\sigma^{-1}C\sigma := \{\sigma^{-1}c\sigma : c \in C\}$, is a diagram.*

For more details see [21, 27, 41].

2 A demonic refinement ordering

We now define the refinement ordering we will be using in the sequel. This ordering induces a complete join semilattice, called a *demonic semilattice*. The associated operations are demonic join ($\sqcup$), demonic meet ($\sqcap$) and demonic composition ($\diamond$). We give the definitions and needed properties of these operations. For more details on relational demonic semantics and demonic operators, see [6, 7, 8, 9, 20, 21, 41].

(6) **Definition.** Let Q and R be relations. We have,

(a) We say that a relation Q *refines* a relation R [28, 29], denoted by $Q \sqsubseteq R$, iff
$$Q \cap RL \subseteq R \quad \wedge \quad RL \subseteq QL.$$

(b) The greatest lower bound (wrt $\sqsubseteq$) of relations Q and R is $Q \sqcup R = (Q \cup R) \cap QL \cap RL$.

(c) If Q and R satisfy the condition $QL \cap RL = (Q \cap R)L$, their least upper bound is $Q \sqcap R = (Q \cap R) \cup \overline{QL} \cap R \cup Q \cap \overline{RL}$, otherwise, the least upper bound does not exist.

(d) We will introduce a certain operation, related to the usual relational composition, the so-called *demonic composition*. Its definition is
$$Q \diamond R := QR \cap Q \triangleleft RL.$$

Note that we assign to $\diamond$ the same binding power as that of $\circ$.

3 Demonic input-output relation

During the execution of a program in an input state, by considering a demonic point of view (if there is a possibility for the program not to terminate normally then it will not terminate normally), three cases may happen : normal termination, abnormal termination and infinite loops. As our goal is to define formally the input-output relation of a diagram by supposing its worst execution, we have to consider these three previous cases together at the same time. Let us give the relational expressions that formalize these notions : the normal termination, the abnormal termination and the infinite loops.

Let $\mathcal{P} = (P, C, \varepsilon, \xi)$ be a diagram. The input-output relation of a diagram $\mathcal{P}$ is given by a relation $\mathcal{E}(\mathcal{P})$ where $\mathcal{E}$ is a function from the set of diagrams to a relational algebra, which associates to each diagram $\mathcal{P}$ the relation $\mathcal{E}(\mathcal{P})$ given by :
$$\mathcal{E}(\mathcal{P}) = \varepsilon \diamond (T \cap \mathcal{I}(P)) \diamond \xi, \quad \text{with } T := P^* \cap \overline{PL}^{\smile}.$$
For more details see [6, 7, 8, 9, 21].

4 Demonic denotational semantics

The demonic semantics of a nondeterministic program p is given by a relation $\mathcal{D}[\![p]\!]$, where $\mathcal{D}$ is a function from the set of programs $\mathcal{P}$ to a certain relational algebra (see Section 1). As, we are interested to imperative programs, this algebra is in general a complete algebra of the form $\mathrm{Rel}(X)$. The constructors considered are affectation, sequence, guarded commands and the while loop. To treat the last case, we need to suppose that the algebra is complete. We will present the syntax before introducing the denotational semantics.

4.1 Syntax

We will use the word *programs* and *instructions* in an interchangeable manner. In reality, we consider programs fragments. Each program uses a certain number of variables $x_0, \ldots, x_n$. We don't precise the syntax of the admissible expressions. Let i be an instruction.

- **Affectation** :

  $x_i := f(x)$, where $f(x)$ is an expression that depends on x_0, ..., x_n.

- **Sequence** :

  $i_1; i_2$, where i_1 and i_2 are instructions.

- **Guarded commands** : **if** $c_1 \to i_1 [\!] \; c_2 \to i_2$ **fi**, where each c_i is a boolean expression called *guarded command* where i_1 and i_2 are instructions. This can be generalized to an arbitrary number of guarded commands.

- **While loop** :

  do $c \to i$ **od**, where c is a boolean expression (guarded command) and i is an instruction.

4.2 Semantics

We will give the demonic denotational semantics of each constructor. These results are from [21].

(7) **Remark.** In our examples, we will use sets to define the program spaces. The space of the program is defined by the variables of the program and their type. Hence, the sets that we are interested in cartesian product of predefined sets.

Let R be a relation defined on a set X which is the cartesian product of sets X_0, ..., X_n. An element $x \in X$ has the form $x = (x_0, \ldots, x_n)$, where $x_i \in X_i$, so, the notation (x, x') is an abbreviation of $((x_0, \ldots, x_n), (x'_0, \ldots, x'_n))$.

We will begin by the basic case; affectation.

- **Affectation** :

  Suppose that $x_0, \ldots, x_n$ are the program variables and their values are in $X_0, \ldots, X_n$, respectively, and that f is a certain executable function. The demonic semantics of the affectation $x_i := f(x)$, where $0 \leq i \leq n$, $x = (x_0, \ldots, x_n)$ and $x' = (x'_0, \ldots, x'_n)$, is the relation

  $$\mathcal{D}[\![x_i := f(x)]\!] := \{(x, x') \mid (\forall j : 0 \leq j \leq n : x_j \in X_j \wedge x'_j \in X_j) \wedge x'_i = f(x) \\ \wedge (\forall j : 0 \leq j \leq n \wedge j \neq i : x'_j = x_j)\}.$$

- **Sequence** :

  The demonic semantics of the sequence $p; q$ of programs p and q is :

  $$(8) \qquad \mathcal{D}[\![p; q]\!] := \mathcal{D}[\![p]\!] \circ \mathcal{D}[\![q]\!].$$

- **Guarded commands** :

  The demonic semantics of the guarded commands $\textbf{if } g_1 \rightarrow p_1 [\!] \; g_2 \rightarrow p_2 \textbf{ fi}$ is

  $$(9) \qquad \mathcal{D}[\![\textbf{ if } g_1 \rightarrow p_1 [\!] \; g_2 \rightarrow p_2 \textbf{ fi }]\!] := \mathcal{G}[\![g_1]\!] \circ \mathcal{G}[\![g_2]\!]^{\sim} \circ \mathcal{D}[\![p_1]\!] \; \sqcap \; \mathcal{G}[\![g_1]\!]^{\sim} \circ \mathcal{G}[\![g_2]\!] \circ \mathcal{D}[\![p_2]\!] \\ \sqcap \; \mathcal{G}[\![g_1]\!] \circ \mathcal{G}[\![g_2]\!] \circ (\mathcal{D}[\![p_1]\!] \sqcup \mathcal{D}[\![p_2]\!]),$$

  where $\mathcal{G}[\![g_i]\!]$ is the semantics of the guarded commands g_i, $i = 1, 2$. The relation $\mathcal{G}[\![g_i]\!]$ is a partial identity such that its domain satisfies the guarded command condition. Notice that $\mathcal{G}$ is applied to boolean expressions but $\mathcal{D}$ is applied to instructions and for this raison we will use two different symbols.

  As the relations $\mathcal{G}[\![g_i]\!]$ are partial identities, by certain properties (for more details see [41]), the last expression is given with angelic operators as follows :

  $$\mathcal{D}[\![\textbf{if } g_1 \rightarrow p_1 [\!] \; g_2 \rightarrow p_2 \textbf{ fi}]\!] = \mathcal{G}[\![g_1]\!]\mathcal{G}[\![g_2]\!]^{\sim}\mathcal{D}[\![p_1]\!] \; \cup \; \mathcal{G}[\![g_1]\!]^{\sim}\mathcal{G}[\![g_2]\!]\mathcal{D}[\![p_2]\!] \\ \cup \; \mathcal{G}[\![g_1]\!]\mathcal{G}[\![g_2]\!](\mathcal{D}[\![p_1]\!] \sqcup \mathcal{D}[\![p_2]\!]).$$

  This expression is inductively explained as follows:

  If g_1 is true and g_2 is false, execute p_1 ; if g_1 is false and g_2 is true, execute p_2 ; if both are true, so do a demonic choice between p_1 and p_2 ($\sqcup$).

  In the case where the conditions are mutually exclusive, we will use certain rules and Equation 9 is reduced to: $\mathcal{D}[\![\textbf{if } g_1 \rightarrow p_1 [\!] \; g_2 \rightarrow p_2 \textbf{ fi}]\!] = \mathcal{G}[\![g_1]\!] \circ \mathcal{D}[\![p_1]\!] \sqcap \mathcal{G}[\![g_2]\!] \circ \mathcal{D}[\![p_2]\!]$.

  In the case of many commands, we have,

  $\mathcal{D}[\![\textbf{if } [\!] \; {}_{i=1}^{n} g_i \rightarrow p_i \textbf{ fi}]\!] := \sqcap_X g_X \circ g_{\widetilde{X}} \circ p_X$, where

- $n \geq 0$,
- $\emptyset \neq X \subseteq \{0, \ldots, n\}$
- $\overline{X}$ is the complement of X with respect to $\{1, \ldots, n-1\}$,
- g_X is the demonic composition of partial identities $\mathcal{G}(g_i)$, for $i \in X$,
- $g_{\overline{X}}^{\sim}$ the demonic composition of partial identities $\mathcal{G}(g_i)^{\sim}$, for $i \in \overline{X}$ (as $\mathcal{G}(g_i)$ and $\mathcal{G}(g_i)^{\sim}$ are partial identities, the order of composition is not important; for $\overline{X} = \emptyset$, we define $g_X^{\sim} = I$,
- $p_X = \bigsqcup_{i \in X} \mathcal{D}[\![p_i]\!]$.

For $n = 2$, we will find Equation (9).

- **While loop**

We treat the while loop case; $W :=$ **do** $g \rightarrow p$ **od**, where g is the loop body condition and p is the loop body.

The demonic semantics of the while loop W is the greatest fixed point with respect to $\sqsubseteq$ of the semantics function $W_d(X) := \mathcal{G}[\![g]\!]^{\sim} \sqcap \mathcal{G}[\![g]\!] \circ \mathcal{D}[\![p]\!] \circ X$ (d recall $d$emonic), where $\mathcal{G}[\![g]\!]$ is the semantics of the condition g ; as for the guarded commands, $\mathcal{G}[\![g]\!]$ is a partial identity whose domain verifies the condition is a partial identity where the domain satisfies the condition. Formally,

$$\mathcal{D}[\![W]\!] = \bigsqcup\{X \mid X = \mathcal{G}[\![g]\!]^{\sim} \sqcap \mathcal{G}[\![g]\!] \circ \mathcal{D}[\![p]\!] \circ X\}. \tag{10}$$

As the domains of two terms of $\sqcap$ are disjoint, and that the partial identities are (deterministic) and that $\circ$ is associative, we have :

$W_d(X) = \mathcal{G}[\![g]\!]^{\sim} \cup (\mathcal{G}[\![g]\!]\mathcal{D}[\![p]\!]) \circ X$,

which is a familiar form of the while loop definition. By fixed point property, we have also,

$$\mathcal{D}[\![W]\!] = \bigsqcup\{X \mid X \sqsubseteq W_d(X)\}.$$

By Remark 14, $\mathcal{D}[\![W]\!]$ exists and is well defined. Expression 10 is the demonic semantics of the while loop given in previous work [21, 41, 43]. By choosing the greatest fixed point (wrt $\sqsubseteq$) means that it is the fixed point with the least domain which has been chosen, which is conform with demonic point of view. Other similar definitions of the demonic semantics of the while loop can be found in [32, 35].

$$\mathcal{E}(\mathcal{P}'') = (G\mathcal{E}(\mathcal{P}))^* Q \cap \mathcal{A}(G\mathcal{E}(\mathcal{P}), Q) \cap \mathcal{I}(G\mathcal{E}(\mathcal{P})). \tag{11}$$

We remark that for the instructions (affectation, sequence, guarded commands and while loop), the semantics of each instruction was given intuitively from the behavior of the instruction. For the while loop, $\mathcal{D}[\![W]\!]$, is the greatest fixed point of a certain function. We will show its existence but it is difficult to calculate it. To solve this problem, when the loop is deterministic, it is possible to use a theorem known as *the Mills while loop verification rule* [30, 31]. We generalized this theorem to a nondeterministic context [40, 41, 42, 43].

(12) **Lemma.** *Let $\mathcal{P} = (P, C, \varepsilon, \xi)$ be a diagram and σ a relation such that $\sigma\sigma^{-1} = I$ and $\sigma^{-1}\sigma \subseteq I$. Then $\mathcal{E}(\mathcal{P}_\sigma) = \sigma^{-1}\mathcal{E}(\mathcal{P})\sigma$.*

5 Demonic operational semantics

In the following, we will define the demonic operational semantics of a program.

Let $\mathrm{Rel}(\mathbf{N} \times S)$ be a relational algebra, where $\mathbf{N}$ is the input set of naturals. The input represent the edges of graph representing the program p and S is the set of the states of the program.

First, we will introduce certain notions to define the operational semantics.

(a) The projections $\pi_1 : \mathbf{N} \times S \to \mathbf{N}$ and $\pi_2 : \mathbf{N} \times S \to S$. The pair (π_1, π_2) forms a direct product. Along this paper, the cartesian products are defined relatively to (π_1, π_2).

(b) The function $\mathcal{Q}$, from the set of programs to the set of associated diagrams; associates to each program p the diagram $\mathcal{Q}[\![p]\!] = (P, C, e, s)$ where P is an element of the algebra $\mathrm{Rel}(\mathbf{N} \times S)$.

(c) The function $\mathcal{F} : \mathrm{Rel}(\mathbf{N} \times S) \to \mathrm{Rel}(S)$ associates to each relation X the relation $\mathcal{F}(X) = \pi_2^{-1} X \pi_2$. The function $\mathcal{F}$ associates to each relation on $\mathbf{N} \times S$ its projection on S. Let $\mathcal{O} := \mathcal{F} \circ \mathcal{E} \circ \mathcal{Q}$. The demonic operational semantics of a program p is given by the relation $\mathcal{O}[\![p]\!] = \mathcal{F}(\mathcal{E}(\mathcal{Q}[\![p]\!]))$.

By consequent, $\mathcal{O}$ is a function from the set of programs to the relations algebra $\mathrm{Rel}(S)$.

6 Comparison of semantics

In this section, we want to compare operational and denotational semantics. Exactly, we will show their equality, in other words, we will prove that each program p verifies

(13) $\mathcal{D}[\![p]\!] = \mathcal{O}[\![p]\!]$.

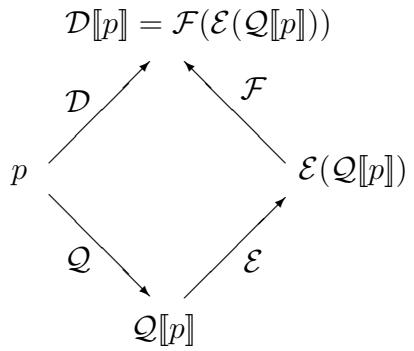

Figure 1: $\mathcal{O}[\![p]\!] := \mathcal{F}(\mathcal{E}(\mathcal{Q}[\![p]\!])) = \mathcal{D}[\![p]\!]$

This is equivalent to prove that this diagram (in the usual meaning) of the figure 1 commutes.

We have to show that each program p is associated to a diagram $\mathcal{Q}[\![p]\!]$ and the projection by a function $\mathcal{F}$ of the input/output relation $\mathcal{E}(\mathcal{Q}[\![p]\!])$ of the diagram $\mathcal{Q}[\![p]\!]$ is equal to the demonic denotational semantics $\mathcal{D}[\![p]\!]$ of the program p.

(14) **Remark.** As we consider the relational algebra $\mathrm{Rel}(\mathbf{N} \times S)$, where $\mathbf{N}$ is the set of natural states and S is the set of the program states, P is a relation on $\mathbf{N} \times S$ and the elements of the set C are of the form $[xx^{-1}, I]$ where x is a relation on $\mathbf{N}$ that represents a state of the program p. For example, the relation $0 := \{(0, n) \mid n \in \mathbf{N}\}$ is the relation that represents the state 0 of the program. This relation is a point (see Definition 2(d).

It is not difficult to prove that the elements of C are partial identities disjoint to each other. The relation $[xy^{-1}, Q]$ on $\mathbf{N} \times S$ indicates that there is a transition from the state x to the state y and that the state variation that follows the transition from x to y described by the relation Q. The relations ε and ξ are given by $[xx^{-1}, I]$ and $[yy^{-1}, I]$ where x and y are respectively the initial state and the final state of program p.

Let x and y be states on $\mathbf{N}$, such that $e = [xx^{-1}, I]$ and $s = [yy^{-1}, I]$. It is easy to prove that

$$\mathcal{E}(\mathcal{Q}[\![p]\!]) = [xy^{-1}, \mathcal{D}[\![p]\!]] \tag{15}$$

implies Equation 13.

Consequently, instead of proving Equation 13, it suffices to prove Equation 15 and show that is verified by each program p (in other words, it exists points x, y such that the equality is verified). As, affectations are atomic programs, and as the sequence, the guarded commands and the loop can be used to construct complex programs, we will define the affectation to satisfy Equation

15. After this, we will define the function $\mathcal{Q}[\![p]\!]$ inductively and we will show that if p is obtained by using constructors to programs that satisfy Equation 15, then p satisfies Equation 15.

- **Affectation**

  We consider the affectation given in Section 4, $x_i := f(x)$, where $0 \leq i \leq n$, $x = (x_0, \ldots, x_n)$. It is easy to see that the diagram associated to the affectation is atomic. The integers 0 and 1 represent the vertices of the graph and $\mathcal{D}[\![x_i := f(x)]\!]$ is the relation that shows the variation from 0 to 1. By using the direct product definition (see c) we have $\mathcal{E}(\mathcal{Q}[\![x_i := f(x)]\!]) = [01^{-1}, \mathcal{D}[\![x_i := f(x)]\!]]$, where 0 and 1 are respectively the initial input state and the final point state of the affectation.

  Before treating the other cases, we describe the method we will use to show that the other constructors verify the induction hypothesis 15. In the beginning, we use the direct sum, we combine together the diagrams associated to programs given by hypotheses. In the following, we show that we obtain a diagram. The third step consists to calculate the input/output relation of the new diagram. Finally, we suppose that the induction Hypotheses 15 is verified by the given diagrams and we verify that the new diagram satisfies also Equation 15.

  In the following, we treat the sequence case.

- **Sequence**

  Let p_1, p_2 be two programs and $\mathcal{Q}[\![p_1]\!] = (P_1, C_1, \varepsilon_1, \xi_1)$,
  $\mathcal{Q}[\![p_2]\!] = (P_2, C_2, \varepsilon_2, \xi_2)$ diagrams associated respectively to these programs. The input relations ε_1 and ε_2 and the output relations ξ_1 and ξ_2 of diagrams $\mathcal{Q}[\![p_1]\!]$ and $\mathcal{Q}[\![p_2]\!]$ are respectively (see Remark 14) :

  $$(16) \quad \varepsilon_1 = [x_1 x_1^{-1}, I], \quad \xi_1 = [y_1 y_1^{-1}, I], \quad \varepsilon_2 = [x_2 x_2^{-1}, I], \quad \xi_2 = [y_2 y_2^{-1}, I],$$

  where x_1, y_1, x_2 and y_2 are points on $\mathbf{N}$ representing respectively the initial states and final states of the programs p_1 and p_2.

  Our aim here is to join into sequence the diagrams $\mathcal{Q}[\![p_1]\!]$ and $\mathcal{Q}[\![p_2]\!]$ in such a way that the output state of the program p_1 coincides with the input state of the program p_2. This will be done by giving new labels to the states of both diagrams in a way that the final state of the program p_1 has to be given the same name as the initial state of the program p_2. This will be done by using the injections σ_1 and σ_2. The idea is very simple, even the details seem to be complicated.

  The relations σ_1 and σ_2 verify the following conditions :

$$(17)\quad \sigma_1\sigma_1^{-1} = I, \qquad \sigma_2\sigma_2^{-1} = I, \qquad \sigma_1\sigma_2^{-1} = y_1x_2^{-1}, \qquad \sigma_1^{-1}\sigma_1 \subseteq I, \qquad \sigma_2^{-1}\sigma_2 \subseteq I,$$

which means that relations σ_1 and σ_2 are total injections.

For example, the state x_1 of the program p_1 is named into $\sigma_1^{-1}x_1$. The condition $\sigma_1\sigma_2^{-1} = y_1x_2^{-1}$ means that the state y_1 (final state of the program p_1) coincides (is connected) with the state x_2 (initial state of program p_2).

The elements of C_1 and C_2 verify the following propriety : $c_1[\sigma_1\sigma_2^{-1}, I] = \emptyset$, where $c_1 \in C_1$ and $c_1 \neq \xi_1$.

In a similar way, we have

$[\sigma_1\sigma_2^{-1}, I]c_2 = \emptyset$, where $c_2 \in C_2$ and $c_2 \neq \varepsilon_2$.

Let the program be $p := p_1; p_2$ which is the sequence of programs p_1 and p_2. In the following, by using the relations σ_1 and σ_2 also the associated to programs p_1 and p_2, we will define a quadruplet $\mathcal{Q}[\![p]\!]$ and show in the following that this last one is effectively a diagram.

Let, $\mathcal{Q}[\![p]\!] := (P, C, e, s)$, where

$$(18)\quad \begin{aligned} P &:= [\sigma_1^{-1}, I]P_1[\sigma_1, I] \cup [\sigma_2^{-1}, I]P_2[\sigma_2, I],\\ C &:= \{[\sigma_i^{-1}, I]c[\sigma_i, I] \mid c \in C_i,\ i = 1, 2, \},\\ \varepsilon &:= [\sigma_1^{-1}, I]\varepsilon_1[\sigma_1, I],\\ \xi &:= [\sigma_2^{-1}, I]\xi_2[\sigma_2, I]. \end{aligned}$$

(19) **Remark.** If σ is a one to one application, $\sigma^{-1}\sigma \subseteq I$ and $\sigma\sigma^{-1} = I$, it is easy to see that the relation $[\sigma, I]$ is also a one to one application. As, (σ_1, σ_2) are one to one applications, $[\sigma_1, I]$ and $[\sigma_2, I]$ are also.

By Remark 19 and Lemma 5, the quadruplets

$\mathcal{Q}_1[\![p_1]\!] := ([\sigma_1^{-1}, I]P_1[\sigma_1, I], [\sigma_1^{-1}, I]C_1[\sigma_1, I], [\sigma_1^{-1}, I]\varepsilon_1[\sigma_1, I], [\sigma_1^{-1}, I]\xi_1[\sigma_1, I])$,

and $\mathcal{Q}_2[\![p_2]\!] := ([\sigma_2^{-1}, I]P_2[\sigma_2, I], [\sigma_2^{-1}, I]C_2[\sigma_2, I], [\sigma_2^{-1}, I]\varepsilon_2[\sigma_2, I], [\sigma_2^{-1}, I]\xi_1[\sigma_2, I])$

are diagrams.

Before proving that $\mathcal{Q}[\![p]\!]$ is effectively a diagram, let us show that the output relation of the diagram $\mathcal{Q}_1[\![p_1]\!]$ is equal to the input relation of the diagram $\mathcal{Q}_2[\![p_2]\!]$, in other words,

$$(20)\quad [\sigma_1^{-1}, I]\xi_1[\sigma_1, I] = [\sigma_2^{-1}, I]\varepsilon_2[\sigma_2, I].$$

By relpacing ξ_1 and ε_2 in Equation 20 by their values (16), we obtain

(21) $[\sigma_1^{-1} y_1 y_1^{-1} \sigma_1, I] = [\sigma_2^{-1} x_2 x_2^{-1} \sigma_2, I].$

It is easy to see that Equation 21 comes from $\sigma_1^{-1} y_1 = \sigma_2^{-1} x_2$. In a similar manner, we show that $\sigma_2^{-1} x_2 \subseteq \sigma_1^{-1} y_1$.

We are now ready to prove that $\mathcal{Q}[\![p]\!] = (P, C, e, s)$ (18) is effectively a diagram (Definition 4). In other words, we have to verify that each element of C is a partial identity, that the elements (different) of C are pairwise disjoint, that $e, s \in C$ and finally that $(\bigcup C)P(\bigcup C) = P$.

- By Remark 19 and the fact that the elements of C_1 and C_2 are partial identities, we will deduce that the elements of C (18) are also partial identities.
- Let c and c' two arbitrary different partial identities from the set C. By considering the structure of the set C(18), three cases are possible :
  - $c = [\sigma_1^{-1}, I] c_1 [\sigma_1, I]$ and $c' = [\sigma_1^{-1}, I] c_1' [\sigma_1, I]$, where c_1, $c_1' \in C_1$. By Remark 19 and Lemma 5, we deduce that $cc' = \emptyset$.
  - $c = [\sigma_2^{-1}, I] c_2 [\sigma_2, I]$ and $c' = [\sigma_2^{-1}, I] c_2' [\sigma_2, I]$, where$c_2$, $c_2' \in C_2$. This case can be treated as the precedent case.
  - $c = [\sigma_1^{-1}, I] c_1 [\sigma_1, I]$ with $c_1 \in C_1$ and $c' = [\sigma_2^{-1}, I] c_2 [\sigma_2, I]$ with $c_2 \in C_2$ (or the symmetric case). If $c_1 = sortie_1$ and $c_2 = \varepsilon_2$, we have $c = c'$. As $c \neq c'$, by Hypotheses, we must have $c_1 \neq \xi_1$ or $c_2 \neq \varepsilon_2$.

  We conclude that the elements of C are disjoint from each other.
- It is easy to see that the relations $[\sigma_1^{-1}, I] \varepsilon_1 [\sigma_1, I]$ and $[\sigma_2^{-1}, I] \xi_2 [\sigma_2, I]$ are elements of the set C (Equation 18). This is from $\varepsilon_1 \in C_1$ and $\xi_2 \in C_2$.
- Finally, let us show this $(\bigcup C)P(\bigcup C) = P$. As $(\bigcup C) \subseteq I$, it suffices to show that $P \subseteq (\bigcup C)P(\bigcup C)$. As $(\circ)$ is distributive with respect to $\cup$, we have $\bigcup C = [\sigma_1^{-1}, I](\bigcup C_1)[\sigma_1, I] \cup [\sigma_2^{-1}, I](\bigcup C_2)[\sigma_2, I]$.

  We consider the fact that $(\bigcup C_1)P_1(\bigcup C_1) = P_1$ and $(\bigcup C_2)P_2(\bigcup C_2) = P_2$ and that $P = [\sigma_1^{-1}, I] P_1 [\sigma_1, I] \cup [\sigma_2^{-1}, I] P_2 [\sigma_2, I]$, it is easy to deduce that $P = (\bigcup C)P(\bigcup C)$.

So, we have showed that $\mathcal{Q}[\![p]\!] = (P, C, e, s)$ is effectively a diagram. We will calculate the input/output relation of diagram $\mathcal{Q}[\![p]\!]$ and prove that the induction hypothesis 15 is verified by the program $p = p_1; p_2$.

So, the diagram $\mathcal{Q}[\![p]\!]$ is constructed from two diagrams $\mathcal{Q}_1[\![p_1]\!]$ and $\mathcal{Q}_2[\![p_2]\!]$ such that the output relation of the first one coincides with the

input relation of the second one (par 20). Consequently, Equation 8 is applied to diagram $\mathcal{Q}[\![p]\!]$ and we have $\mathcal{E}(\mathcal{Q}[\![p]\!]) = (\mathcal{E}(\mathcal{Q}_1[\![p_1]\!]) \triangleright \mathcal{E}(\mathcal{Q}_2[\![p_2]\!])$.

By Remark 19 and Lemma 5(b), this is equivalent to

$\mathcal{E}(\mathcal{Q}[\![p]\!]) = ([\sigma_1^{-1}, I]\mathcal{E}(\mathcal{Q}[\![p_1]\!])[\sigma_1, I]) \triangleright ([\sigma_2^{-1}, I]\mathcal{E}(\mathcal{Q}[\![p_2]\!])[\sigma_2, I])$.

By the induction Hypothesis 15, we have

$\mathcal{E}(\mathcal{Q}[\![p_1]\!]) = [x_1 y_1^{-1}, \mathcal{D}[\![p_1]\!]]$. Similarly, we have $\mathcal{E}(\mathcal{Q}[\![p_2]\!]) = [x_2 y_2^{-1}, \mathcal{D}[\![p_2]\!]]$. We have now necessary notions to show the induction Hypothesis 15 is satisfied by the program $p = p_1; p_2$, in other words,

$$\mathcal{E}(\mathcal{Q}[\![p_1; p_2]\!]) = [\sigma_1^{-1} x_1 y_2^{-1} \sigma_2, \mathcal{D}[\![p_1; p_2]\!]]. \tag{22}$$

It is not difficult to show that $\sigma_1^{-1} x_1$ and $\sigma_2^{-1} y_2$ are des points on $\mathbf{N}$, verifying $e = [\sigma_1^{-1} x_1 x_1^{-1} \sigma_1, I]$ and $s = [\sigma_2^{-1} y_2 y_2^{-1} \sigma_2, I]$ (this can be deduced from 16, 17.

- **Guarded Commands** Let p_1 and p_2 two programs, g_1 and g_2 two guarded commands. We aim to verify if the program **if** $g_1 \to p_1 [\!] \; g_2 \to p_2$ **fi** (see diagram c) satisfies Equation 15. The treatment of guarded commands is the same as the sequence and in a certain manner as the while loop case that we will present in the following. Consequently, we give directly the result.

  Equation 9 implies that :

$$\begin{aligned}\mathcal{E}(\mathcal{Q}[\![\textbf{ if } g_1 \to p_1 [\!] \; g_2 \to p_2 \textbf{ fi}]\!]) = \\ [xy^{-1}, \mathcal{D}[\![\textbf{ if } g_1 \to p_1 [\!] \; g_2 \to p_2 \textbf{ fi }]\!]],\end{aligned} \tag{23}$$

  where x and y are respectively the initial and the final state of the program **if** $g_1 \to p_1 [\!] \; g_2 \to p_2$ **fi** .

  We deduce that the induction hypotheses is verified 15 for guarded commands.

- **While Loop**

  Finally, we treat the while loop case.

  Let p be a program and $\mathcal{Q}[\![p]\!] = (P, C_1, \varepsilon_1, \xi_1)$ the diagram associated to program p where $\varepsilon_1 := [x_1 x_1^{-1}, I]$ and $s := [y_1 y_1^{-1}, I]$. Let the program $w :=$ **do** $g \to p$ **od** (see d for the definition of the associated diagram).

We seek to prove that the induction hypothesis 15 is verified by the program w. We will construct diagram $\mathcal{Q}[\![w]\!] = (W, C, \xi_1, s)$. It suffices to add a unique state. Let $s := [yy^{-1}, I]$, where y is such that $s(\bigcup C_1) = \emptyset$. In other words, y is a state on the naturals numbers that was not already used as intermediate state in $\mathcal{Q}[\![p]\!]$. Let,

- $C := C_1 \cup \{s\}$,
- $W := [y_1 x_1^{-1}, \mathcal{G}(g)] \cup P \cup [y_1 y^{-1}, \mathcal{G}(g)^{\sim}]$.

Let us show that $\mathcal{Q}[\![w]\!]$ is a diagram (Definition 4). Il is suffisant to show that $W \subseteq (\bigcup C)W(\bigcup C)$, as the other properties are easy to verify. By using Remark 14, the fact that $[x_1^{-1}x_1, I]$ and $[y_1^{-1}y_1, I]$ are elements of C_1 (recall that the elements of C_1 are pairwise disjoints) and that $s = [yy^{-1}, I]$ and verifies $s(\bigcup C_1) = \emptyset$, it is not difficult to show that the following properties are verified :

- $(\bigcup C_1)[y_1 x_1^{-1}, \mathcal{G}(g)] = [y_1 x_1^{-1}, \mathcal{G}(g)]$,
- $s[y_1 x_1^{-1}, \mathcal{G}(g)] = \emptyset$,
- $(\bigcup C_1)[y_1 y^{-1}$,
- $\mathcal{G}(g)^{\sim}] = [y_1 y^{-1}, \mathcal{G}(g)^{\sim}]$,
- $s[y_1 y^{-1}, \mathcal{G}(g)^{\sim}] = \emptyset$.

In a similar way, we have:

- $[y_1 x_1^{-1}, \mathcal{G}(g)](\bigcup C_1) = [y_1 x_1^{-1}, \mathcal{G}(g)]$,
- $[y_1 x_1^{-1}, \mathcal{G}(g)]s = \emptyset$,
- $[y_1 y^{-1}, \mathcal{G}(g)^{\sim}](\bigcup C_1) = \emptyset$,
- $[y_1 y^{-1}, \mathcal{G}(g)^{\sim}]s = [y_1 y^{-1}, \mathcal{G}(g)^{\sim}]$,

we obtain: $W \subseteq (\bigcup C)W(\bigcup C)$. So, $\mathcal{Q}[\![p]\!]$ is effectively a diagram. In what follows, we will calculate the input/output relation.

By Taking

$$G := [y_1 x_1^{-1}, \mathcal{G}(g)], \quad P := P, \quad Q := [y_1 y^{-1}, \mathcal{G}(g)^{\sim}]. \tag{24}$$

So, Equation 11 is applied to the diagram $\mathcal{Q}[\![w]\!]$ and we have

$$\mathcal{E}(\mathcal{Q}[\![w]\!]) = (G\mathcal{E}(\mathcal{Q}[\![p]\!]))^* Q \cap \mathcal{A}(G\mathcal{E}(\mathcal{Q}[\![p]\!]), Q) \cap \mathcal{I}(G\mathcal{E}(\mathcal{Q}[\![p]\!])).$$

We remark that $QL \cap G\mathcal{E}(\mathcal{Q}[\![p]\!]L = \emptyset$. By fixed point property, we have

$$\mathcal{E}(\mathcal{Q}[\![w]\!]) = \bigsqcup\{X \mid X = Q \cup G\mathcal{E}(\mathcal{Q}[\![p]\!]) \triangleleft X\}. \tag{25}$$

By applying the induction Hypothesis 15 to diagram $\mathcal{Q}[\![p]\!]$, we obtain :

$$\mathcal{E}(\mathcal{Q}[\![p]\!]) = [x_1 y_1^{-1}, \mathcal{D}[\![p]\!] \triangleleft X]. \tag{26}$$

By applying Equations 24, 26, Equation 25 becomes

$$\mathcal{E}(\mathcal{Q}[\![w]\!]) = \bigsqcup\{X \mid X = [y_1 y^{-1}, \mathcal{G}(g)^{\sim}] \cup [y_1 y_1^{-1}, \mathcal{G}(g)\mathcal{D}[\![p]\!]] \triangleleft X\}. \tag{27}$$

To simplify the notations we adopt the next abbreviations :

$$\begin{aligned} &\textbf{Abbreviation.}\ A := y_1 y_1^{-1}, \qquad B := y_1 y^{-1}, \\ &f(X) := [B, \mathcal{G}(g)^{\sim}] \cup [A, \mathcal{G}(g)\mathcal{D}[\![p]\!]] \triangleleft X, \\ &g(X) := \mathcal{G}(g)^{\sim} \cup \mathcal{G}(g)\mathcal{D}[\![p]\!] \triangleleft X. \end{aligned} \tag{28}$$

By fixed point property [38], Abbreviations 28 and Equation 27, let us show that

$$\nu(f) = [B, \nu(g)]. \tag{29}$$

It is easy to see that the relations A and B verify the following properties :

$$AB = B, \qquad AL = BL, \qquad A \subseteq I.$$

Before we prove the Equation 29, we will give the following.

(30) **Lemma.** *Let A, B be relations given in Abbreviation 28, and P and Q be relations*

(a) $[A, P]^*[B, Q] = [B, P^*Q]$,

(b) $\mathcal{A}([A, P], [B, Q]) = [BL, \mathcal{A}(P, Q)]$.

Now, we will Equation 29. Consider f, B and g as given in (abbreviation 28) Equation 29, we find the following equation :

$$\mathcal{E}(\mathcal{Q}[\![w]\!]) = [y_1 y^{-1}, \bigsqcup\{X \mid X = \mathcal{G}(g)^{\sim} \cup \mathcal{G}(g)\mathcal{D}[\![p]\!] \triangleleft X\}]. \tag{31}$$

By certain rules the precedent equation becomes

$$\mathcal{E}(\mathcal{Q}[\![w]\!]) = [y_1 y^{-1}, \mathcal{D}[\![w]\!]]. \tag{32}$$

Then, we have proved that the induction hypothesis 15 is verified by the program :

$w :=$ **do** $g \to p$ **od**.

So, by induction, we have proved the induction hypotheses 15 is verified by the sequence (22), by guarded commands (23) and finally by the while loop (31). As, we consider imperative programs whose constructors are sequence, guarded commands and the while loop, so we can affirm that the program p verifies $\mathcal{O}[\![p]\!] = \mathcal{D}[\![p]\!]$, in other words the demonic operational semantics of a nondeterministic program p is equal to the demonic denotational semantics of this program.

7 Conclusion

In this paper, we have defined the notion of the demonic operational semantics of a nondeterministic p. This semantics was given from the notion of diagram and the input/output relation of diagram. By proceeding by induction on constructors, we have showed that the demonic operational of a program is equal to the demonic denotational semantics of this program. As, an intermediate result, we have also showed how to combine the diagrams as sequence and as a loop to obtain a new diagram. Finally, we have showed the equality between the demonic operational semantics and the demonic denotational semantics. This implies the Figure 1 commutes.

The approach to demonic input-output relation presented here is not the only possible one. In [23, 24, 25], the infinite looping has been treated by adding to the state space a fictitious state $\perp$ to denote nontermination. In [9, 16, 27, 33], the demonic input-output relation is given as a pair (relation,set). The relation describes the input-output behavior of the program, whereas the set component represents the domain of guaranteed termination.

We note that the preponderant formalism employed until now for the description of demonic input-output relation is the wp-calculus. For more details see [2, 3, 11, 14, 34, 48].

Acknowledgements. This work was supported by King Saud University Deanship of Academic Research, Women Sections Research Center.

References

[1] C. Aarts, R. Backhouse, P. Hoogendijk, E. Voermans, and J. van der Woude. A relational theory of datatypes. Department of Computing Science, Eindhoven University of Technology, 1992. http://www.win.tue.nl/win/cs/wp/papers.

[2] R. J. R. Back. On the correctness of refinement in program development. Thesis, Department of Computer Science, University of Helsinki, 1978.

[3] R. J. R. Back. Combining angels, demons and miracles in program specifications. *Theoretical Computer Science*, 100:365–383, 1992.

[4] R. J. R. Back. A continuous semantics for unbounded nondeterminism, *Theoretical Computer Science*, 23:187–210, 1983.

[5] R. C. Backhouse and H. Doornbos. Mathematical induction made calculational. Computing Science Note 94/16, Dept. of Mathematics and

Computer Science, Eindhoven University of Technology, The Netherlands, 1994.

[6] R. C. Backhouse and J. van der Woude. Demonic operators and monotype factors. *Mathematical Structures in Computer Science*, 3(4):417–433, December 1993. Also: Computing Science Note 92/11, Dept. of Mathematics and Computer Science, Eindhoven University of Technology, The Netherlands, 1992.

[7] R. Berghammer. Relational specification of data types and programs. Technical report 9109, Fakultät für Informatik, Universität der Bundeswehr München, Germany, September 1991.

[8] R. Berghammer and G. Schmidt. Relational specifications. In C. Rauszer (ed.), *Algebraic Logic, Banach Center Publications*, 28, Polish Academy of Sciences, 1993.

[9] R. Berghammer and H. Zierer. Relational algebraic semantics of deterministic and nondeterministic programs. *Theoretical Computer Science*, 43:123–147, 1986.

[10] C. Böhm. On a family of Turing machines and the related programming languages. *ICC Bull*, 3:187–194, 1964.

[11] Franck van Breugell. An introduction to metric semantics: operational and denotational models for programming and specification languages, *Theoretical Computer Science*, 258:1–98, 2001

[12] C. Brink, W. Kahl, and G. Schmidt, editors. *Relational Methods in Computer Science*. Springer, 1997.

[13] H. Doornbos, R. Backhouse, and J. van der Woude. A calculational approach to mathematical induction. *Theoretical Computer Science*, 179(1–2):103–135, 1997.

[14] E. W. Dijkstra. *A Discipline of Programming*. Prentice-Hall, 1976.

[15] H. Doornbos. Reductivity. *Science of Computer Programming*, 26:217–236, 1996.

[16] H. Doornbos. A relational model of programs without the restriction to Egli-Milner monotone constructs. *IFIP Transactions*, A-56:363–382. North-Holland, 1994.

[17] L. H. Chin and A. Tarski. Distributive and modular laws in the arithmetic of relation algebras. *University of California Publications*, 1:341–384, 1951.

[18] B. A. Davey and H. A. Priestley. *Introduction to Lattices and Order*. Cambridge Mathematical Textbooks, Cambridge University Press, Cambridge, 1990.

[19] J. Desharnais. *Abstract relational semantics*. School of Computer Science, Univ. McGill, Montréal, 1989. Ph D. thesis

[20] J. Desharnais, B. Möller, and F. Tchier. Kleene under a demonic star. *8th International Conference on Algebraic Methodology And Software Technology (AMAST 2000)*, May 2000, Iowa City, Iowa, USA, *Lecture Notes in Computer Science*, Vol. 1816, pages 355–370, Springer-Verlag, 2000.

[21] J. Desharnais, N. Belkhiter, S. B. M. Sghaier, F. Tchier, A. Jaoua, A. Mili and N. Zaguia. Embedding a demonic semilattice in a relation algebra. *Theoretical Computer Science*, 149(2):333–360, 1995.

[22] R. W. Floyd. Assigning Meanings to programs. *Proceedings AMS Symposium in Applied Mathematics*, 19:19–31, 1967.

[23] C. A. R. Hoare and J. He. The weakest prespecification. *Fundamenta Informaticae IX*, 1986, Part I: 51–84, 1986.

[24] C. A. R. Hoare and J. He. The weakest prespecification. *Fundamenta Informaticae IX*, 1986, Part II: 217–252, 1986.

[25] C. A. R. Hoare and al. Laws of programming. *Communications of the ACM*, 30:672–686, 1986.

[26] Z. Huibiao, J. P. Bowen and He Jifeng. Deriving Operational Semantics from Denotational Semantics for Verilog. *Proc. APSEC 2001: 8th Asia-Pacific Software Engineering Conference, IEEE Computer Society Press*, pp. 177–184, Macau, China, December 2001. http://citeseer.ist.psu.edu/zhu01deriving.html

[27] R. D. Maddux. Relation-algebraic semantics. *Theoretical Computer Science*, 160:1–85, 1996.

[28] A. Mili. A relational approach to the design of deterministic programs. *Acta Informatica*, 20:315–328, 1983.

[29] A. Mili, J. Desharnais and F. Mili. Relational heuristics for the design of deterministic programs. *Acta Informatica*, 24(3):239–276, 1987.

[30] H. D. Mills. The new math of computer programming. *Commun. ACM 18*, 1, January 1975, 43–48.

[31] H. D. Mills, V. R. Basili, J. D. Gannon and R. G. Hamlet. *Principles of Computer Programming. A Mathematical Approach.* Allyn and Bacon, Inc., 1987.

[32] Nguyen, T. T.: A Relational Model of Demonic Nondeterministic Programs. *Int. J. Foundations Comput. Sci.*, **2(2)**, 101–131 (1991).

[33] D. L. Parnas. A Generalized Control Structure and its Formal Definition *Communications of the ACM*, 26:572-581, 1983

[34] G.D. Plotkin. A Structural Approach to Operational Semantics. *Technical Report DIAMI FN-19, Computer Science Department, Aarhus University*, 1981.

[35] E. Sekerinski. A Calculus for Predicative Programming. *Second International Conf on the Mathematics of Program Construction*, R. S. Bird, C. C. Morgan, and J. C. P. Woodcock (eds), Oxford, June 1992, *Lecture Notes in Computer Science*, Vol. 669, Springer-Verlag, 1993.

[36] G. Schmidt. Programs as partial graphs I: Flow equivalence and correctness. *Theoretical Computer Science*, 15:1–25, 1981.

[37] G. Schmidt and T. Ströhlein. *Relations and Graphs.* EATCS Monographs in Computer Science, Springer-Verlag, Berlin, 1993.

[38] A. Tarski. A lattice-theoretical fixpoint theorem and its applications. *Pacific Journal of Mathematics*, 5:285–309, 1955.

[39] A. Tarski. On the calculus of relations. *J. Symbolic Logic*, 6(3):73–89, 1941.

[40] F. Tchier and J. Desharnais. A generalisation of a theorem of Mills. *Proceedings of the Tenth International Symposium on Computer and Information Sciences, ISCIS X*, pages 27–34, October 1995, Turkey.

[41] F. Tchier. Sémantiques relationnelles démoniaques et vérification de boucles non déterministes. Ph D. thesis, Département de mathématiques and de statistique, Université Laval, Canada, 1996. http://auguste.ift.ulaval.ca/ desharn/Theses/index.html.

[42] F. Tchier and J. Desharnais. Applying a generalization of a theorem of Mills to generalized looping structures. In *Science and Engineering in Software Development. A recognition of Harlan D. Mills' Legacy*, Los Angeles, CA, May 1999, pages 31–38, IEEE Computer Society Press, 1999.

[43] F. Tchier. La sémantique démoniaque relationnelle des diagrammes composés. Proc. 5th Seminar on Relational Methods in computer Science. (RelMICS'5). Université Laval, Canada, 9-14 January 2000.

[44] F. Tchier. Demonic relational semantics of compound diagrams. In: Jules Desharnais, Marc Frappier and Wendy MacCaull, editors. Relational Methods in computer Science: The Québec seminar, pages 117-140, Methods Publishers 2002.

[45] F. Tchier. While loop demonic relational semantics monotype/residual style.*2003 International Conference on Software Engineering Research and Practice (SERP03)*, Las Vegas, Nevada, USA, 23-26, June 2003.

[46] F. Tchier. Demonic semantics by monotypes. *International Arab conference on Information Technology* (Acit2002),University of Qatar, Qatar, 16-19 December 2002,

[47] F. Tchier. Demonic Semantics: using monotypes and residuals. *International Journal of Mathematics and Mathematical Sciences, IJMMS* 2004:3, 135-160, 2004.

[48] J. Tiuryn and M. Wand. Untyped Lambda-Calculus with Input-Output. *In H. Kirchner, editor, Trees in Algebra and Programming: CAAP'96, Proc. 21st International Colloquium, volume 1059 of Lecture Notes in Computer Science*, pages 317–329, Berlin, Heidelberg, and New York, April 1996.

[49] M. Wand and G. T. Sullivan. Denotational Semantics Using an Operationally-Based Term Model. *In Proceedings 23rd ACM Symposium on Programming Languages*, pages 386–399, 1997.