Electronic Communications of the EASST
Volume 66 (2013)

Proceedings of the Automated Verification of Critical Systems (AVoCS 2013)

From System Requirements to Software Requirements in the Four-Variable Model

Lucian M. Patcas, Mark Lawford and Tom Maibaum

15 pages

Guest Editors: Steve Schneider, Helen Treharne
Managing Editors: Tiziana Margaria, Julia Padberg, Gabriele Taentzer
ECEASST Home Page: http://www.easst.org/eceasst/ ISSN 1863-2122

From System Requirements to Software Requirements in the Four-Variable Model

Lucian M. Patcas[1], Mark Lawford[2] and Tom Maibaum[3]

[1]patcaslm@mcmaster.ca, [2]lawford@mcmaster.ca, [3]maibaum@mcmaster.ca
Department of Computing and Software, McMaster University, Canada

Abstract: The four-variable model of software-controlled embedded systems originally proposed by Parnas and Madey has been used successfully in the development of safety-critical applications in various industries. The model does not explicitly specify the software requirements, but rather bounds them by specifying the system requirements and the input and output hardware interfaces of the system. The software engineers are left with the problem of how to construct software that satisfies the system requirements and hardware interfacing constraints. After formalizing the properties of acceptable system and software implementations using the demonic calculus of relations, we provide (i) a necessary and sufficient condition for the existence of an acceptable software implementation and (ii) a mathematical characterization of the software requirements in terms of their weakest specification.

Keywords: safety-critical, requirements, four-variable model, specification and refinement, demonic calculus of relations, verified system development

1 Introduction

Many safety-critical systems monitor and control physical processes. Such systems typically are embedded systems that have sensors, actuators, and a controller. Based on the measured values of the physical quantities of interest obtained from the sensors, the controller commands the actuators with the purpose of maintaining certain properties in the physical process. The controller usually has a software implementation that runs on a digital computer. The four-variable model proposed by Parnas and Madey [PM95] (Figure 1) helps to clarify the behaviour of, and the boundaries between, the physical process, sensors, actuators, and control software. The model was used as early as 1978 as part of the Software Cost Reduction (SCR) program of the Naval Research Laboratory for specifying the flight software of the U.S. Navy's A-7 aircraft [VS90]. The ideas from SCR were later integrated in the Consortium Requirements Engineering (CoRE) methodology, which was used for specifying the avionics system of the C-130J aircraft in the 1980s [FFK+94]. Another significant example of a successful use of the four-variable model is the redesign of the software in the shutdown system of the Darlington nuclear power plant in Ontario, Canada in the 1990s [LMFM00, WL03, WL06]. More recently, in 2009, the four-variable model was used extensively in the Requirements Engineering Handbook [LM09], put together at the request of the Federal Aviation Administration in the United States.

In the four-variable model (Figure 1), *REQ* models the *system requirements*. At the system requirements level, a system is seen as a black-box that relates physical quantities measured by

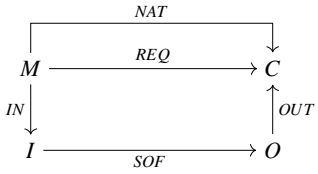

Figure 1: The four-variable model.

the system, called *monitored variables*, to physical quantities controlled by the system, called *controlled variables*. For example, monitored variables might be the pressure and temperature inside a nuclear reactor while controlled variables might be visual and audible alarms, as well as the trip signal that initiates a reactor shutdown; whenever the temperature or pressure reach abnormal values, the alarms go off and the shutdown procedure is initiated. The sets of the possible values of the monitored and controlled variables are denoted by M and C, respectively. If tolerances are allowed on the system requirements, then *REQ* will be a relation, as opposed to a function, because there will be values in M for which more than one value in C will be acceptable. The environmental constraints on the system are described by the relation *NAT* (from "nature"), which restricts the possible values of the monitored and controlled variables. An environmental constraint might be the maximum rate of climb of an aircraft in the case of an avionics system, for instance. As an extreme example, if everything was possibile in the physical environment, then *NAT* would be the universal relation between M and C.

The possible *system designs* are modelled by a sequential composition of *IN*, *SOF*, and *OUT*. Here, *IN* models the *input hardware interface* (sensors and analog-to-digital converters) and relates values of monitored variables to values of *input variables* in the software. The input variables model the information about the environment that is available to the software. For example, *IN* might model a pressure sensor that converts temperature values to analog voltages; these voltages are then converted via an A/D converter to integer values stored in a register accesible to the software. The *output hardware interface* (digital-to-analog converters and actuators) is modelled by *OUT*, which relates values of the *output variables* of the software to values of controlled variables. An output variable might be, for instance, a boolean variable set by the software with the understanding that the value `true` indicates that a reactor shutdown should occur and the value `false` indicates the opposite. The sets of the possible values of the input and output variables are denoted by I and O, respectively. To account for the inaccuracies introduced by the hardware interfaces, *IN* and *OUT* are in general relations and not functions. For example, assume that *IN* models an A/D converter that converts analog voltages in the range 0–5V with an accuracy of $\pm$0.5V; then, for an actual monitored voltage of 2.5V, the value of the corresponding input variable in the software can be any of the digital representations that correspond to 2V, 2.5V, and 3V, hence *IN* is a relation and not a function. Relating values of input variables to values of output variables is *SOF*, which models the *control software*. If we want to capture all the possible implementations of the control software, then *SOF* will typically have to be a relation.

The relations *NAT* and *REQ* are described by domain experts. The system designers allocate the system requirements between hardware and software, and describe *IN* and *OUT*. The software engineers must determine *SOF* and verify whether it meets the constraints imposed by *NAT*, *REQ*, *IN*, and *OUT*. Considering that changes in the specifications of the system requirements and hardware interfaces arise often in the early stages of system development, the task

of determining *SOF* is repetitive and, thus, even more demanding. An important question for software engineers to ask themselves before investing resources to develop and verify a detailed implementation is whether an acceptable software implementation exists at all given the system requirements and hardware interfaces as currently specified. A positive answer to this question would allow the software engineers to proceed with a software design having the confidence that their efforts are not destined to fail from the start. In the case of a negative answer, the next step would be to determine what needs to be changed in *REQ*, *IN*, or *OUT* in order for a software implementation to be possible.

Another question to consider is what are the software requirements? The four-variable model does not explicitly specify the software requirements, but rather bounds them by specifying the system requirements and the input and output hardware interfaces of the system. Extracting the software requirements from these specifications is "often an exercise in frustration" [MT01] and an automated method would be a significant advantage. Moreover, a mathematical characterization of the software requirements would offer a sound starting point for the software design process.

1.1 Related work

Methods for asserting the existence of a software implementation in the four-variable model have not received much attention in the literature. Of the few examples, Lawford et al. [LMFM00] give, without proof, a necessary condition for the existence of SOF in a functional variant of the four-variable model. In the context of real-time systems, Hu et al. [HLW09] address in a functional four-variable model the ability of a software implementation to meet continuous-time requirements, such as the detection of physical events that have been enabled for a predefined amount of time; necessary and sufficient existence conditions for SOF are given for different assumptions made about the access of the software to the time of the environment. In this paper we address the need for existence conditions of a software implementation in the general, relational case of the four-variable model. The relational case is more realistic as it can model hardware inaccuracies and tolerances on requirements.

Techniques for deriving the software requirements from the specifications of the system requirements and input/output hardware interfaces in the four-variable model are mentioned in [BH00, HT00, MT01, WL06]. These techniques are variations of the same idea of manually decomposing *SOF* into three subrelations that model the input device drivers, output device drivers, and, respectively, the system functionality required by *REQ*; the latter subrelation of *SOF* closely resembles *REQ* and is regarded as the software requirements. In this paper, we do not decompose *SOF* and present a method for "calculating" the weakest (i.e, least restrictive) specification of *SOF* that still satisfies the constraints imposed by *NAT*, *REQ*, *IN*, and *OUT*. We regard this weakest specification of *SOF* as the software requirements specification.

Gunter et al. [GGJZ00] pointed out that the system and software acceptability conditions given in Parnas and Madey [PM95] are too weak and allow undesirable system and software specifications, in particular specifications that are inconsistent with the physical environment. We fix the acceptability conditions by giving a new semantics for the four-variable model based on the demonic calculus of relations [DMN97, Kah03]. In Section 2 we introduce the demonic calculus of relations and prove necessary and sufficient conditions for the existence of demonic left

and right factors which will later be used to prove a necessary and sufficient condition for the existence of an acceptable software implementation. In Section 3 we formalize in the demonic calculus of relations the properties of acceptable system and software implementations, giving a stronger definition for the angelic acceptability notion proposed by Parnas and Madey [PM95]; this stronger definition explicitly rejects system behaviours that are not physically meaningful. A necessary and sufficient condition for the existence of an acceptable software implementation is given in Section 4, along with a mathematical characterization of the software requirements in terms of their weakest specification.

2 Relations

A relation R from a set A to a set B is a subset of the cartesian product $A \times B$. In other words, R is a subset of the set of all ordered pairs (a,b), where $a \in A$ and $b \in B$. As customary in higher-order logic, a relation can also be viewed as a binary predicate, that is, as a function of type $A \to B \to bool$. This allows us to use currying and write $R\,a\,b$ to denote $(a,b) \in R$, and $R\,a = \{b \in B \mid R\,a\,b\}$ to denote the *image set* of a under the relation R.

Some elementary operations involving a relation $R \subseteq A \times B$ are:

- *domain* of R: $\mathsf{dom}\,(R) = \{a \in A \mid \exists b \in B.\ R\,a\,b\}$
- *range* of R: $\mathsf{ran}\,(R) = \{b \in B \mid \exists a \in A.\ R\,a\,b\}$
- *converse* of R: $R^{\smile} = \{(b,a) \in B \times A \mid R\,a\,b\}$
- *complement* of R: $\overline{R} = \{(a,b) \in A \times B \mid \neg R\,a\,b\}$

For any two sets A and B, the relation $\top_{A,B} = \{(a,b) \in A \times B \mid \mathsf{true}\}$ is the *universal relation* between A and B, while $\bot_{A,B} = \{(a,b) \in A \times B \mid \mathsf{false}\}$ is the *empty relation* between A and B.

2.1 Angelic operators

The *intersection* of two relations $P \subseteq A \times B$ and $Q \subseteq A \times B$ is the relation $P \cap Q = \{(a,b) \in A \times B \mid P\,a\,b \wedge Q\,a\,b\}$. Their *union* is $P \cup Q = \{(a,b) \in A \times B \mid P\,a\,b \vee Q\,a\,b\}$. The relation P is *contained* in the relation Q, written $P \subseteq Q$, if and only if $\forall a \in A.\ \forall b \in B.\ P\,a\,b \Longrightarrow Q\,a\,b$. Relational containment (or inclusion) is a partial order.

The *composition* of two relations $P \subseteq A \times B$ and $Q \subseteq B \times C$ is the relation:

$$P ; Q = \{(a,c) \in A \times C \mid \exists b \in B.\ P\,a\,b \wedge Q\,b\,c\}$$

The precedence of the relational operators introduced so far is as follows: the unary operators $^{\smile}$ and $\overline{\ }$ are evaluated first; the binary operator $;$ is evaluated next; and the binary operators $\cap$ and $\cup$ are evaluated last.

The following subrelations of a relation $P \subseteq A \times B$ are obtained by restricting the domain or range of P to the domain or range of another relation:

- the subrelation of P obtained by restricting the domain of P to the domain of a relation $R \subseteq A \times C$ is $P\big|_{\mathsf{dom}(R)} = P \cap R ; \top_{C,B} = \{(a,b) \in A \times B \mid R\,a\,b \wedge a \in \mathsf{dom}\,(R)\}$

- the subrelation of P obtained by restricting the domain of P to the range of a relation $R \subseteq C \times A$ is: $P\big|_{\mathsf{ran}(R)} = P \cap R^{\smile} ; \mathbb{T}_{C,B} = \{(a,b) \in A \times B \mid Rab \wedge a \in \mathsf{ran}\,(R)\}$
- the subrelation of P obtained by restricting the range of P to the domain of a relation $Q \subseteq B \times C$ is $P\big|^{\mathsf{dom}(Q)} = P \cap \mathbb{T}_{A,C} ; Q^{\smile} = \{(a,b) \in A \times B \mid Rab \wedge b \in \mathsf{dom}\,(Q)\}$

The relational composition and inclusion operations induce two residuation operations: the left and right residuals [SS93, Fra95, BKS97, Kah03]. Assuming two relations $R \subseteq A \times C$ and $Q \subseteq B \times C$, the *left residual* of R by Q, denoted R/Q, is the largest solution of the inequality $X ; Q \subseteq R$, where $X \subseteq A \times B$ is the unknown:

$$X ; Q \subseteq R \iff X \subseteq R/Q$$

The value of the left residual of R by Q is:

$$R/Q = \overline{\overline{R} ; Q^{\smile}} = \{(a,b) \in A \times B \mid \forall c \in C.\, Qbc \Longrightarrow Rac\} = \{(a,b) \in A \times B \mid Qb \subseteq Ra\}$$

Given two relations $R \subseteq A \times C$ and $P \subseteq A \times B$, the *right residual* of R by P, denoted $P\backslash R$, is the largest solution of the inequality $P ; X \subseteq R$, where $X \subseteq B \times C$ is the unknown:

$$P ; X \subseteq R \iff X \subseteq P\backslash R$$

The value of the right residual of R by P is:

$$P\backslash R = \overline{P^{\smile} ; \overline{R}} = \{(b,c) \in B \times C \mid \forall a \in A.\, Pab \Longrightarrow Rac\} = \left\{(b,c) \in B \times C \mid P^{\smile} b \subseteq R^{\smile} c\right\}$$

The precedence of $/$ and $\backslash$ is the same as the precedence of the relational composition. The residuation operations are loosely analogous to division of natural numbers and the values of the residuals are a form of quotient. The left residual R/Q can be understood as what remains on the left of R after R is "divided" by Q on the right. Dually, the right residual $P\backslash R$ is what remains on the right of R after "dividing" R by P on the left. Hoare and He [HH86] were among the first to advocate the importance of the relational residuals to software development. They called the left residual R/Q the *weakest prespecification* of program Q to achieve specification R; the right residual $P\backslash R$ was called the *weakest postspecification* of program P to achieve specification R.

The relational operators introduced so far have been used in the literature to formalize so called *angelic semantics*. In such semantics, specifications that allow bad behaviours are acceptable as long as good behaviours are also possible. In contrast, in a *demonic semantics* a specification is rejected if bad behaviours are possible. When developing safety-critical systems it is always wise to plan for the worst, therefore the demonic approach is more suitable.

2.2 Demonic operators

We now present the demonic relational operators that will be used in the paper and prove necessary and sufficient conditions for the existence of demonic left and right factors. These conditions will be used in Section 4 to prove a necessary and sufficient existence condition for an acceptable software implementation in the four-variable model. Overviews of the demonic calculus of relations can be found in [DMN97, Kah03].

A relation $P \subseteq A \times B$ is a *demonic refinement* of a relation $R \subseteq A \times B$, written $P \sqsubseteq R$, if and only if $P\big|_{\mathsf{dom}(R)} \subseteq R$ and $\mathsf{dom}(R) \subseteq \mathsf{dom}(P)$. The demonic refinement is a partial order. Consider the relations $R = \{(a_1,b_1),(a_1,b_2),(a_2,b_2)\}$, $P = \{(a_1,b_1),(a_2,b_2),(a_3,b_2),(a_3,b_3)\}$, $Q = \{(a_1,b_1),(a_2,b_1),(a_3,b_3)\}$, and $S = \{(a_2,b_2)\}$. Here, P refines R, Q does not refine R because $(a_2,b_1) \notin R$, and S does not refine R because $\mathsf{dom}(R) \not\subseteq \mathsf{dom}(P)$.

The *demonic composition* of relations $P \subseteq A \times B$ and $Q \subseteq B \times C$ is defined as:

$$P \boxdot Q = P ; Q \cap \overline{P ; \overline{Q ; \top_{C,C}}} = \{(a,c) \in A \times C \mid (a,c) \in P ; Q \wedge Pa \subseteq \mathsf{dom}(Q)\}$$

Demonic composition is the same as the angelic composition when P is univalent (i.e., a total or partial function) or when Q is a total relation. As an example, consider the following relations: $P = \{(a_1,b_1),(a_1,b_2)\}$ and $Q = \{(b_1,c_1)\}$; then $P ; Q = \{(a_1,c_1)\}$, whereas $P \boxdot Q = \bot\!\!\!\bot_{A,C}$. Specifications $P ; Q$ where dead ends such as (a_1,b_2) are possible essentially allow potential implementations to get stuck and not terminate for some inputs as long as they produce expected results for some other inputs; such specifications are not suitable for safety-critical systems.

As was the case with angelic composition and inclusion, demonic composition and demonic refinement induce two residuation operations. The *demonic left residual* of a relation $R \subseteq A \times C$ by a relation $Q \subseteq B \times C$, denoted $R \mathbin{/\!\!/} Q$, is defined as the largest solution with respect to $\sqsubseteq$ of the inequation $X \boxdot Q \sqsubseteq R$, where $X \subseteq A \times B$ is the unknown:

$$X \boxdot Q \sqsubseteq R \iff X \sqsubseteq R \mathbin{/\!\!/} Q$$

A solution X, also called *demonic left factor* of R through Q, does not always exist. As such, the demonic left residual $R \mathbin{/\!\!/} Q$ is not always defined. For example, if $Q = \bot\!\!\!\bot_{B,C}$ and $R \neq \bot\!\!\!\bot_{A,C}$, then $X \boxdot Q = \bot\!\!\!\bot_{A,C}$ for any $X \subseteq A \times B$, in which case $X \boxdot Q$ is not a demonic refinement of R and $R \mathbin{/\!\!/} Q$ is undefined. If the demonic left residual is defined, then its value is:

$$R \mathbin{/\!\!/} Q = R/Q \cap \top_{A,C} ; Q^{\smile} = (R/Q)\big|^{\mathsf{dom}(Q)}$$

We now state and prove a necessary and sufficient condition for the existence of a demonic left factor and, therefore, for the definedness of the demonic left residual.

Lemma 1 *Given two relations $R \subseteq A \times C$ and $Q \subseteq B \times C$, there exists a demonic left factor $X \subseteq A \times B$ such that $X \boxdot Q \sqsubseteq R$ if and only if $\forall a \in \mathsf{dom}(R).\ \exists b \in \mathsf{dom}(Q).\ Qb \subseteq Ra$.*

Proof. If direction: $(\exists X.\ X \boxdot Q \sqsubseteq R) \Longrightarrow \forall a \in \mathsf{dom}(R).\ \exists b \in \mathsf{dom}(Q).\ Qb \subseteq Ra$

$$\begin{aligned}
&\exists X.\ X \boxdot Q \sqsubseteq R \\
\Longrightarrow\ &\langle \text{by definition, if } X \boxdot Q \sqsubseteq R \text{ admits a solution, then } R \mathbin{/\!\!/} Q \text{ also is a solution} \rangle \\
&(R \mathbin{/\!\!/} Q) \boxdot Q \sqsubseteq R \\
\Longrightarrow\ &\langle \text{by definition of } \sqsubseteq \rangle \\
&\mathsf{dom}(R) \subseteq \mathsf{dom}((R \mathbin{/\!\!/} Q) \boxdot Q) \\
\Longrightarrow\ &\mathsf{dom}(R) \subseteq \mathsf{dom}(R \mathbin{/\!\!/} Q) \\
\Longrightarrow\ &\forall a \in \mathsf{dom}(R).\ \exists b \in B.\ (a,b) \in R \mathbin{/\!\!/} Q \\
\Longrightarrow\ &\langle \text{by value of } R \mathbin{/\!\!/} Q \rangle \\
&\forall a \in \mathsf{dom}(R).\ \exists b \in \mathsf{dom}(Q).\ Qb \subseteq Ra
\end{aligned}$$

Only if direction: $(\forall a \in \mathsf{dom}(R).\ \exists b \in \mathsf{dom}(Q).\ Qb \subseteq Ra) \Longrightarrow \exists X.\ X \mathbin{\square} Q \sqsubseteq R$

$$
\begin{aligned}
& \forall a \in \mathsf{dom}(R).\ \exists b \in \mathsf{dom}(Q).\ Qb \subseteq Ra \\
\Longleftrightarrow\ & \langle \text{by value of } R/Q \text{ \& value of } R /\!\!/ Q \rangle \\
& \forall a \in \mathsf{dom}(R).\ \exists b \in \mathsf{dom}(Q).\ (a,b) \in R /\!\!/ Q \\
\Longleftrightarrow\ & \forall a \in \mathsf{dom}(R).\ \exists b \in B.\ (a,b) \in R /\!\!/ Q \wedge b \in \mathsf{dom}(Q) \\
\Longleftrightarrow\ & \langle \text{by definition of } \mathbin{\square} \rangle \\
& \forall a \in \mathsf{dom}(R).\ \exists c \in C.\ ((R /\!\!/ Q) \mathbin{\square} Q)\ ac \\
\Longrightarrow\ & \mathsf{dom}(R) \subseteq \mathsf{dom}((R /\!\!/ Q) \mathbin{\square} Q)
\end{aligned}
\tag{1}
$$

$$
\begin{aligned}
& ((R /\!\!/ Q) \mathbin{\square} Q)\big|_{\mathsf{dom}(R)} \subseteq R \\
\Longleftrightarrow\ & \forall a \in \mathsf{dom}(R).\ ((R /\!\!/ Q) \mathbin{\square} Q)\ a \subseteq Ra \\
\Longleftrightarrow\ & \langle \text{by unfolding } \subseteq \rangle \\
& \forall a \in \mathsf{dom}(R).\ \forall c \in C.\ ((R /\!\!/ Q) \mathbin{\square} Q)\ ac \Longrightarrow Rac \\
\Longleftrightarrow\ & \langle \text{by definition of } \mathbin{\square} \text{ and } ; \rangle \\
& \forall a \in \mathsf{dom}(R).\ \forall c \in C.\ (\exists b \in \mathsf{dom}(Q).\ ((R /\!\!/ Q)ab \wedge Qbc)) \Longrightarrow Rac \\
\Longleftrightarrow\ & \langle \text{by value of } R /\!\!/ Q \rangle \\
& \forall a \in \mathsf{dom}(R).\ \forall c \in C.\ (\exists b \in \mathsf{dom}(Q).\ ((Qbc) \Longrightarrow Rac) \wedge Qbc) \Longrightarrow Rac \\
\Longleftrightarrow\ & \langle \text{by modus ponens} \rangle \\
& \forall a \in \mathsf{dom}(R).\ \forall c \in C.\ Rac \Longrightarrow Rac \\
\Longleftrightarrow\ & \mathsf{true}
\end{aligned}
\tag{2}
$$

$$
\begin{aligned}
& \forall a \in \mathsf{dom}(R).\ \exists b \in \mathsf{dom}(Q).\ Qb \subseteq Ra \\
\Longrightarrow\ & \langle \text{by (1) \& (2) \& definition of } \sqsubseteq \rangle \\
& (R /\!\!/ Q) \mathbin{\square} Q \sqsubseteq R \\
\Longrightarrow\ & \exists X.\ X \mathbin{\square} Q \sqsubseteq R
\end{aligned}
$$

□

Several conditions for the definedness of the demonic left residual can be found in the literature, such as, if brought to our notation: $\mathsf{dom}(R) \subseteq \mathsf{dom}\left((R/Q)\big|^{\mathsf{dom}(Q)}\right)$ in [DBS+95, Fra95]; and, $\mathsf{dom}(R) \subseteq \mathsf{dom}((R/Q);Q)$ in [DMN97, Kah03]. It can be shown that these conditions are equivalent to our condition in Lemma 1.

The *demonic right residual* of a relation $R \subseteq A \times C$ by a relation $P \subseteq A \times B$, denoted $P \setminus\!\!\setminus R$, is defined as the largest solution with respect to $\sqsubseteq$ of the inequation $P \mathbin{\square} X \sqsubseteq R$, where $X \subseteq B \times C$ is the unknown:

$$P \mathbin{\square} X \sqsubseteq R \iff X \sqsubseteq P \setminus\!\!\setminus R$$

A solution X to this inequation, called a *demonic right factor* of R through P, does not always exists. Therefore, the demonic right residual $P \setminus\!\!\setminus R$ is not always defined. If $P \setminus\!\!\setminus R$ is defined, then

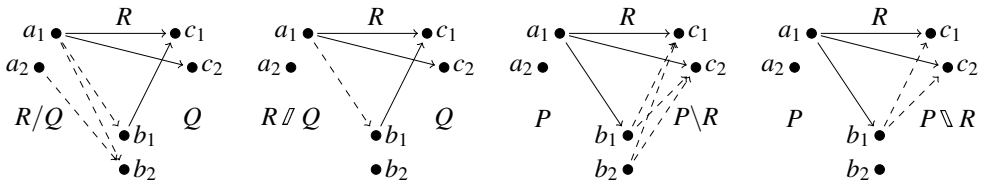

Figure 2: Examples of angelic and demonic residuals.

its value is:

$$P \mathbin{\backslash\!\!\backslash} R = (P \cap R ; \mathbb{T}_{C,B}) \backslash R \cap (P \cap R ; \mathbb{T}_{C,B})^{\smile} ; \mathbb{T}_{A,C} = \left(P|_{\mathsf{dom}(R)} \Big\backslash R \right) \Big|_{\mathsf{ran}\left(P|_{\mathsf{dom}(R)}\right)}$$

We now give a necessary and sufficient condition for the existence of a demonic right factor and, therefore, for the definedness of the demonic right residual.

Lemma 2 *Given two relations $R \subseteq A \times C$ and $P \subseteq A \times B$, there exists a demonic right factor $X \subseteq B \times C$ such that $P \square X \sqsubseteq R$ if and only if $\forall b \in \mathsf{ran}\left(P|_{\mathsf{dom}(R)}\right). \exists c \in C.\ \left(P|_{\mathsf{dom}(R)}\right)^{\smile} b \subseteq R^{\smile} c$ and $\mathsf{dom}(R) \subseteq \mathsf{dom}(P)$.*

Proof. Similar to the proof of Lemma 1. □

The definedness conditions for the demonic right residual presented in the literature [Fra95, DBS+95, Kah03] can all be brought to the following common form in our notation: $\mathsf{dom}(R) \subseteq \mathsf{dom}(P)$ and $\mathbb{T}_{B,C} \subseteq \left(P|_{\mathsf{dom}(R)} \Big\backslash R \right) ; \mathbb{T}_{C,C}$. These conditions are stated only as sufficient, but it can be shown that they are equivalent to our condition in Lemma 2. The advantage of our conditions for the definedness of demonic left and right residuals when used by engineers is that the formulations in first-order logic give a better insight into their meaning compared to the abstract relation algebraic formulations found in the literature.

Examples of angelic and demonic residuals are depicted in Figure 2. Here, if seen as specifications, the angelic residual R/Q allows the dead end (a_1, b_2) where an implementation could get stuck, whereas the demonic residual $R \mathbin{/\!\!/} Q$ does not allow any dead ends. Moreover, both demonic residuals in the figure are less restrictive than their angelic counterparts without breaking refinement; for example, $P\backslash R$, but not $P \mathbin{\backslash\!\!\backslash} R$, asks its implementations to produce an output for b_2, which is not something of interest for R.

Two relations $P \subseteq A \times B$ and $Q \subseteq A \times B$ are *compatible* if and only if $\mathsf{dom}(P) \cap \mathsf{dom}(Q) \subseteq \mathsf{dom}(P \cap Q)$. The demonic refinement partial order induces a complete join-semilattice [DMN97, Kah03]. In this paper we will be using the meet operation of this semilattice, also called the *demonic intersection*. The demonic intersection of two relations $P \subseteq A \times B$ and $Q \subseteq A \times B$ is defined if P and Q are compatible. If the demonic intersection of P and Q is defined, its value is:

$$P \sqcap\!\!\sqcap Q = (P \cap Q) \cup \left(P \cap \overline{Q ; \mathbb{T}_{B,B}}\right) \cup \left(\overline{P ; \mathbb{T}_{B,B}} \cap Q\right) = (P \cap Q) \cup P|_{\overline{\mathsf{dom}(Q)}} \cup Q|_{\overline{\mathsf{dom}(R)}}$$

For example, let $P = \{(a_1, b_1), (a_2, b_1), (a_2, b_3)\}$ and $Q = \{(a_2, b_2), (a_2, b_3), (a_3, b_3)\}$; in this case, $P \sqcap\!\!\sqcap Q = \{(a_2, b_3), (a_1, b_1), (a_3, b_3)\}$.

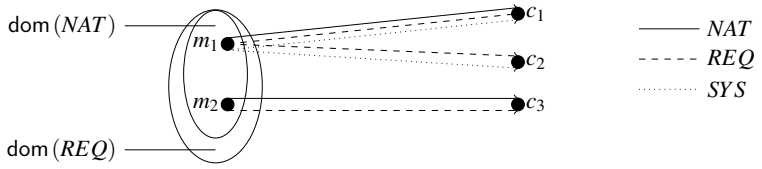

Figure 3: Problems with acceptability conditions in [PM95].

3 System and software acceptability

What conditions must a software implementation satisfy to be acceptable? A first intuition is that an acceptable software implementation must be part of an acceptable system implementation. In this section we formalize the properties of acceptable system and software implementations. The starting point is system requirements that are consistent with the physical laws of the environment.

In Parnas and Madey [PM95], the system requirements *REQ* are *feasible* with respect to *NAT* if and only if the following conditions are both satisfied:

$$\mathsf{dom}\,(NAT) \subseteq \mathsf{dom}\,(REQ) \tag{3}$$

$$\mathsf{dom}\,(NAT \cap REQ) = \mathsf{dom}\,(NAT) \cap \mathsf{dom}\,(REQ) \tag{4}$$

These conditions do not imply computability, nor that an implementation of the system requirements is even practical. The intention of the authors was for the feasibility conditions to allow only required behaviours (as described by *REQ*) that are allowed by the natural environment (as described by *NAT*). Condition (3) means that the system requirements should specify behaviour for all monitored values that can arise in the environment. It can be assumed that the inputs that are outside the domain of *NAT* will never occur. Condition (4) says that for each input in the common subset of their domains, *NAT* and *REQ* should agree on at least one output. Together, conditions (3) and (4) postulate that, for every input possible in the environment, the system requirements should ask the system to produce at least one output that is physically meaningful. If (3) and (4) are not satisfied, then there will be no realizable system implementation. For the rest of the paper, *REQ* will be assumed to be feasible unless otherwise stated. Another assumption is that the system requirements are consistent [HJL96].

Parnas and Madey [PM95] define acceptability of software using the following condition:

$$NAT \cap (IN ; SOF ; OUT) \subseteq REQ \tag{5}$$

A system implementation $SYS = IN ; SOF ; OUT$ is then acceptable if and only if it satisfies:

$$NAT \cap SYS \subseteq REQ \tag{6}$$

These acceptability conditions are, however, not strong enough. Let us consider the relations *NAT*, *REQ*, and *SYS* in Figure 3, where m_1, m_2 are values of monitored variables in the set M and c_1, c_2, c_3 are values of controlled variables in the set C. These relations satisfy the conditions (3), (4), and (6). Therefore, according to [PM95], the system requirements *REQ* are feasible with respect to *NAT* and the system implementation *SYS* is acceptable although:

- $(m_2, c_3) \in NAT$, $(m_2, c_3) \in REQ$, and $(m_2, c_3) \notin SYS$. A system implementation SYS that does not deal with all the inputs in $\mathrm{dom}(NAT)$ is deemed acceptable. It is mentioned elsewhere in [PM95] that $\mathrm{dom}(NAT) \subseteq \mathrm{dom}(IN)$; this, however, does still not ensure that $\mathrm{dom}(NAT) \subseteq \mathrm{dom}(IN \fatsemi SOF \fatsemi OUT)$.

- $(m_1, c_2) \in SYS$ and $(m_1, c_2) \notin NAT$. A system specification SYS that asks an implementation to produce outputs not physically possible is deemed acceptable. From an engineering perspective, such systems are not realizable and it is important to reject early specifications that allow them. A similar problem with the acceptability condition in [PM95] was pointed out in Gunter et al. [GGJZ00].

In the remainder of the section, the conditions (5) and (6) will be strengthened using the demonic calculus of relations, introduced in Subsection 2.2. Condition (4) of the system requirements feasibility is in fact equivalent to the compatibility condition (Subsection 2.2) of NAT and REQ because $\mathrm{dom}(NAT \cap REQ) \subseteq \mathrm{dom}(NAT) \cap \mathrm{dom}(REQ)$ is a tautology. Therefore, if REQ is feasible with respect to NAT, then $NAT \barwedge REQ$ is defined. In Figure 3, if we implement $NAT \barwedge REQ = \{(m_1, c_1), (m_2, c_3)\}$ instead of REQ, then $(m1, c2)$ will not be allowed to be part of SYS and $(m2, c3)$ will be required to belong to SYS. That is, $NAT \barwedge REQ$ captures only that part of the system requirements that is physically meaningful.

We need to clarify at this point what it means for an implementation to satisfy a specification. As a satisfaction concept, we choose the demonic refinement of relations, also known as "total correctness" in the relation algebra literature and introduced in Subsection 2.2. The interpretation of the demonic refinement is twofold. First, if a specification is defined for some input, then any implementation must produce a result allowed by the specification. Second, if the specification is not defined for some input, then producing arbitrary results or producing no result whatsoever are both allowed for that input. We now redefine the acceptability notion of Parnas and Madey [PM95] in the demonic calculus of relations.

Definition 1 A system implementation SYS is acceptable if and only if $SYS \sqsubseteq NAT \barwedge REQ$.

For an acceptable system implementation SYS, the demonic refinement, demonic intersection, and feasibility of system requirements ensure that $\mathrm{dom}(NAT) \subseteq \mathrm{dom}(NAT \barwedge REQ) \subseteq \mathrm{dom}(REQ) \subseteq \mathrm{dom}(SYS)$. Consequently, an acceptable system implementation will sense all the inputs that are possible in the environment. For these inputs, the system is asked to produce only outputs allowed by the physical environment. The inputs outside the domain of NAT, but in the domain of REQ, can be assumed to never happen under normal circumstances; these inputs can be used for specifying fault-tolerant behaviour for abnormal circumstances. Allowing arbitrary behaviour outside the domain of REQ should present no danger as it is assumed that, for a final product, hazard analyses have been conducted and all the inputs that could lead to hazardous system behaviour have been added to the domain of REQ as additional safety requirements.

In Parnas and Madey [PM95], a system implementation is given as $SYS = IN \fatsemi SOF \fatsemi OUT$. As seen in Subsection 2.2, the angelic composition allows dead ends between the composed relations. We argue that this may allow undesirable system behaviours. For example, let us consider a relation IN that models an 8-bit resolution A/D converter such that a monitored analog voltage $m = 0\mathrm{V}$ is mapped to a digital value $i = 0$, $m = 2.49\mathrm{V}$ is mapped to $i = 127$, and $m = 4.99\mathrm{V}$

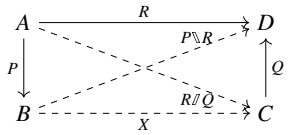

Figure 4: Existence of a demonic mid factor.

is mapped to $i = 255$ (i.e., $i = \lfloor m * 2^8/5 \rfloor$). The requirements ask the system to produce at the output the double of the input. Because the relation *NAT* says that the monitored voltages will be in the range 0–2.49V, it is decided that 8-bit unsigned integers will be used to represent the values of the output variable $o = 2 * i$ set by the software. If the converter has an accuracy of ± 0.03V, the following situation can occur: a monitored voltage of $m = 2.49$V can be mapped to any of the software inputs between $i = 125$ ($m = 2.46$V) and $i = 129$ ($m = 2.52$V); in the cases when the software input i is greater than or equal to 128, the corresponding software output $o = 2 * i$ will be greater than 255 and will not fit in the 8-bit variable. In practice this means an overflow that can cause a fatal runtime error, in which case the system will not produce an expected result. At the specification level, this situation manifests itself as a dead end between *SOF* and *OUT*. Angelic composition allows such system behaviours because, even if the implementation fails to produce expected results when $i \geq 128$, expected results will be produced when $125 \leq i \leq 127$. Demonic composition will discard the behaviours that correspond to the system input $m = 2.49V$ altogether since some of them ($i \geq 128$) are prone to failure. Therefore, we redefine the description of a system implementation to be $SYS = IN \boxdot SOF \boxdot OUT$.

Definition 2 A software implementation *SOF* is *acceptable* if and only if $IN \boxdot SOF \boxdot OUT \sqsubseteq NAT \boxplus REQ$.

4 Existence of an acceptable software implementation

In the previous section we formalized what it means for a software implementation to be acceptable, but does such a software implementation exist at all? The mathematical question we ask is, given relations *NAT*, *REQ*, *IN*, and *OUT* in the four-variable model, does a relation *SOF* exist such that $IN \boxdot SOF \boxdot OUT \sqsubseteq NAT \boxplus REQ$?

To reduce notational verbosity, we switch momentarily to using R to denote $NAT \boxplus REQ$, P for *IN*, Q for *OUT*, and X for *SOF* (Figure 4). We are interested in necessary and sufficient conditions for the existence of a demonic factor X such that $P \boxdot X \boxdot Q \sqsubseteq R$. Demonic composition is an associative operation [BW93]: $P \boxdot (X \boxdot Q) = (P \boxdot X) \boxdot Q$. Associativity of $\boxdot$ indicates that both diagonals AC and BD are necessary for X to exist. This suggests that the existence of the diagonals might also be a sufficient condition. As it turns out, this is not the case. By applying Lemma 2 in $\triangle A, B, D$, the necessary and sufficient condition for diagonal BD to exist is:

$$\mathsf{dom}(R) \subseteq \mathsf{dom}(P) \wedge \forall b \in \mathsf{ran}\left(P|_{\mathsf{dom}(R)}\right). \exists d \in D.\ \left(P|_{\mathsf{dom}(R)}\right)^{\smile} b \subseteq R^{\smile} d \tag{7}$$

By definition, the largest relation, with respect to $\sqsubseteq$, for the diagonal BD is the demonic right residual $P \backslash R$. The necessary and sufficient condition for the existence of diagonal AC is obtained by applying Lemma 1 in $\triangle A, C, D$:

$$\forall a \in \mathsf{dom}(R). \exists c \in \mathsf{dom}(Q).\ Q\,c \subseteq R\,a \tag{8}$$

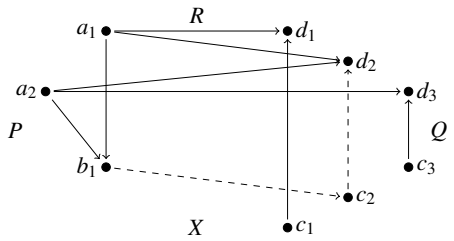

Figure 5: Diagonals are not sufficient for a demonic mid factor.

The largest relation, with respect to $\sqsubseteq$, for the diagonal AC is the demonic left residual $R /\!\!/ Q$. While conditions (7) and (8) are both necessary for X, Figure 5 provides a counterexample to the sufficiency of their conjunction. Condition (7) is satisfied because $\mathsf{dom}(R) \subseteq \mathsf{dom}(P)$ and $\left(P|_{\mathsf{dom}(R)}\right)^{\smile} b_1 = \{a_1, a_2\} \subseteq R^{\smile} d_2 = \{a_1, a_2\}$. Condition (8) is also satisfied because $Q c_1 = \{d_1\} \subseteq R a_1 = \{d_1, d_2\}$ and $Q c_3 = \{d_3\} \subseteq R a_2 = \{d_2, d_3\}$. However, if $(b_1, c_1) \in X$, then a_2 can be connected to d_1 via $P \,\square\, X \,\square\, Q$ although $(a_2, d_1) \notin R$; similarly, if $(b_1, c_3) \in X$, then a_1 can reach d_3 via $P \,\square\, X \,\square\, Q$ although $(a_1, d_3) \notin R$. Consequently, there is no relation X such that $P \,\square\, X \,\square\, Q \sqsubseteq R$, although both (7) and (8) are satisfied. It is only when $(c_2, d_2) \in Q$ that there is an $X = \{(b_1, c_2)\}$ such that $P \,\square\, X \,\square\, Q \sqsubseteq R$. It can be seen in Figure 5 that d_2 enjoys a special property: the amount of "confusion" at the input of R to produce d_2 is at least the same as the amount of "confusion" at the input of P to produce b_1, that is, $\left(P|_{\mathsf{dom}(R)}\right)^{\smile} b_1 = \{a_1, a_2\} \subseteq R^{\smile} d_2 = \{a_1, a_2\}$. This suggests that Q reaching points similar to d_2 must be part of a necessary and sufficient condition for X to exist.

Lemma 3 *Given three relations $R \subseteq A \times D$, $P \subseteq A \times B$, and $Q \subseteq C \times D$, there exists a demonic factor $X \subseteq B \times C$ such that $P \,\square\, X \,\square\, Q \sqsubseteq R$ if and only if* $\mathsf{dom}(R) \subseteq \mathsf{dom}(P)$ *and* $\forall b \in \mathsf{ran}\left(P|_{\mathsf{dom}(R)}\right). \exists c \in \mathsf{dom}(Q). Q c \subseteq \left\{ d \in D \mid \left(P|_{\mathsf{dom}(R)}\right)^{\smile} b \subseteq R^{\smile} d \right\}$.

Proof. The geometrical interpretation (Figure 5) of the associativity of $\square$ is that it does not matter if we use the diagonal BD or the diagonal AC to arrive to the conditions for the existence of X. As such, it suffices to use the diagonal BD and show that the conditions in Lemma 3 are necessary and sufficient for X such that $P \,\square\, (X \,\square\, Q) \sqsubseteq R$.

$$
\begin{array}{ll}
 & \exists X.\, P \,\square\, (X \,\square\, Q) \sqsubseteq R \\
\iff & \langle \text{by definition of } \setminus\!\!\setminus \text{ \& Lemma 2 applied in } \triangle A, B, D \rangle \\
 & (\exists X.\, X \,\square\, Q \sqsubseteq P \setminus\!\!\setminus R) \wedge (7) \\
\iff & \langle \text{by Lemma 1 applied in } \triangle B, C, D \rangle \\
 & \forall b \in \mathsf{dom}(P \setminus\!\!\setminus R).\, \exists c \in \mathsf{dom}(Q).\, Q c \subseteq (P \setminus\!\!\setminus R) b \wedge (7) \\
\iff & \left\langle \mathsf{dom}(P \setminus\!\!\setminus R) = \mathsf{ran}\left(P|_{\mathsf{dom}(R)}\right) \;\&\; (P \setminus\!\!\setminus R) b = \left\{ d \in D \mid \left(P|_{\mathsf{dom}(R)}\right)^{\smile} b \subseteq R^{\smile} d \right\} \right\rangle \\
 & \mathsf{dom}(R) \subseteq \mathsf{dom}(P) \\
 & \wedge\, \forall b \in \mathsf{ran}\left(P|_{\mathsf{dom}(R)}\right). \exists c \in \mathsf{dom}(Q).\, Q c \subseteq \left\{ d \in D \mid \left(P|_{\mathsf{dom}(R)}\right)^{\smile} b \subseteq R^{\smile} d \right\}
\end{array}
$$

□

Lemma 4 *Given relations $R \subseteq A \times D$, $P \subseteq A \times B$, $X \subseteq B \times C$, and $Q \subseteq C \times D$, if $P \fatsemi X \fatsemi Q \sqsubseteq R$, then $X \sqsubseteq P \backslash\backslash R /\!/ Q$.*

Proof. For any X such that $P \fatsemi (X \fatsemi Q) \sqsubseteq R$ we have that $P \fatsemi (X \fatsemi Q) \sqsubseteq R \iff X \fatsemi Q \sqsubseteq P \backslash\backslash R \iff X \sqsubseteq (P \backslash\backslash R) /\!/ Q$ by using the definitions of $\backslash\backslash$ and $/\!/$, respectively. Considering that the demonic composition is associative, it is also the case that any X that satisfies $(P \fatsemi X) \fatsemi Q \sqsubseteq R$ is also a demonic refinement of the residual $P \backslash\backslash (R /\!/ Q)$ and that $(P \backslash\backslash R) /\!/ Q = P \backslash\backslash (R /\!/ Q)$. Therefore, we drop the parentheses and say that any solution of the inequality $P \fatsemi X \fatsemi Q \sqsubseteq R$, if it exists, is a demonic refinement of the residual $P \backslash\backslash R /\!/ Q$. In other words, this residual is the largest solution, with respect to $\sqsubseteq$, of $P \fatsemi X \fatsemi Q \sqsubseteq R$. □

At this point we are ready to present the main results of the paper. The following theorem states a necessary and sufficient condition for the existence of an acceptable software implementation in the four-variable model (Figure 1).

Theorem 1 *There exists an acceptable software implementation SOF if and only if for any software input $i \in \mathsf{ran}\left(IN|_{\mathsf{dom}(NAT \barwedge REQ)}\right)$ there exists a software output $o \in \mathsf{dom}(OUT)$ such that $OUT\, o \subseteq \left\{c \in C \mid \left(IN|_{\mathsf{dom}(NAT \barwedge REQ)}\right)^{\smile} i \subseteq (NAT \barwedge REQ)^{\smile} c\right\}$, and $\mathsf{dom}(NAT \barwedge REQ) \subseteq \mathsf{dom}(IN)$.*

Proof. Direct consequence of Lemma 3. □

Following from Definition 2 and Lemma 4, we have that any acceptable software implementation, if it exists, is a demonic refinement of the residual $IN \backslash\backslash (NAT \barwedge REQ) /\!/ OUT$. As a result, this residual is the least restrictive software specification, or the *weakest software specification*, as it leaves open most software design options. The weakest software specification describes all the possible acceptable software implementations and, in this sense, it describes the software requirements. An actual software implementation is a functional (i.e., deterministic) demonic refinement of the software requirements.

5 Conclusions

A method for assessing the existence of an acceptable software implementation early in the development of a safety-critical system may save time and resources. We have addressed this need and proved a necessary and sufficient condition for the existence of an acceptable software implementation in a general, relational variant of the four-variable model in which inaccuracies of the sensing and actuating hardware as well as tolerances can be modelled. To be useful in practice, the existence check for acceptable software needs to be supported by tools. Satisfiability Modulo Theories (SMT) solving may be a fruitful direction for a completely automated check, although a point of concern is the existential quantifier that falls within the scope of the universal quantifier in our existence condition. When SMT solving does not work, verifying whether the existence condition is satisfied or not will still be possible in an interactive proof assistant.

In this paper we also have formalized the acceptability conditions for system and software implementations using the demonic calculus of relations and fixed the angelic conditions of Parnas and Madey [PM95]. In the process, we provided a mathematical characterization of the software requirements in terms of their weakest specification. The main advantage for having a relation algebraic characterization for the software requirements is the high potential for mechanization. Given the finite relations *NAT*, *REQ*, *IN*, and *OUT*, it is possible to calculate the residual $IN \setminus\!\setminus (NAT \boxminus REQ) /\!\!/ OUT$ by turning the relational calculus into matrix operations on the adjacency matrices of the graphs associated with the relations [SS93]. If this proves unfeasible for very large relations, or in the case of infinite relations, reasoning about relational specifications will still be possible in an interactive proof assistant.

For increased confidence in our results, we formalized and verified the mathematical development presented in this paper with the proof assistant Coq.

Acknowledgements: The authors would like to thank Dave Parnas for clarifications about the four-variable model. Thanks also go to Wolfram Kahl who has always found time to answer our questions about relation algebra.

Bibliography

[BH00] R. Bharadwaj, C. Heitmeyer. Developing High Assurance Avionics Systems with the SCR Requirements Method. In *Proceedings of the 19th Digital Avionics Systems Conference*. October 2000.

[BKS97] C. Brink, W. Kahl, G. Schmidt (eds.). *Relational Methods in Computer Science*. Advances in Computing. Springer-Verlag, 1997.

[BW93] R. C. Backhouse, J. van der Woude. Demonic Operators and Monotype Factors. *Mathematical Structures in Computer Science* 3(4):417–433, December 1993.

[DBS+95] J. Desharnais, N. Belkhiter, S. B. M. Sghaier, F. Tchier, A. Jaoua, A. Mili, N. Zaguia. Embedding a Demonic Semilattice in a Relation Algebra. *Theoretical Computer Science* 149(2):333–360, 1995.

[DMN97] J. Desharnais, A. Mili, T. Nguyen. *Refinement and Demonic Semantics*. In [BKS97], chapter 11, pp. 166–183, 1997.

[FFK+94] S. Faulk, J. Finneran, J. Kirby, S. Shash, J. Sutton. Experience Applying the CoRE Method to the Lockhead C-130J Software Requirements. In *Ninth Annual Conference on Computer Assurance*. Gaithersburg, Maryland, June 1994.

[Fra95] M. Frappier. *A Relational Basis for Program Construction by Parts*. PhD thesis, Computer Science Department, University of Ottawa, 1995.

[GGJZ00] C. A. Gunter, E. L. Gunter, M. Jackson, P. Zave. A Reference Model fo Requirements and Specifications. *IEEE Software* 17(3):37–43, May/June 2000.

[HH86] C. A. R. Hoare, J. He. The Weakest Prespecification. *Fundamenta Informaticae* 9(1):(Part I) 51–84, (Part II) 217–252, 1986.

[HJL96] C. L. Heitmeyer, R. D. Jeffords, B. G. Labaw. Automated Consistency Checking of Requirements Specifications. *ACM Transactions on Software Engineering and Methodology* 5(3):230–261, 1996.

[HLW09] X. Hu, M. Lawford, A. Wassyng. Formal Verification of the Implementability of Timing Requirements. In *Formal Methods for Industrial Critical Systems*. Lecture Notes in Computer Science 5596, pp. 119–134. Springer, 2009.

[HT00] M. Heimdahl, J. Thompson. Specification Based Prototyping of Control Systems. In *Proceedings of the 19th IEEE Digital Avionics Systems Conference*. October 2000.

[Kah03] W. Kahl. Refinement and Development of Programs from Relational Specifications. *Electronic Notes in Theoretical Computer Science (ENTCS)* 44(3):51–93, 2003.

[LM09] D. L. Lempia, S. P. Miller. Requirements Engineering Management Handbook. Technical report DOT/FAA/AR-08/32, U.S. Department of Transportation, Federal Aviation Administration, June 2009.

[LMFM00] M. Lawford, J. McDougall, P. Froebel, G. Moum. Practical Application of Functional and Relational Methods for the Specification and Verification of Safety Critical Software. In *Proceedings of Algebraic Methodology and Software Technology, AMAST*. Lecture Notes in Computer Science 1816, pp. 73–88. Springer, 2000.

[MT01] S. P. Miller, A. C. Tribble. Extending the Four-Variable Model to Bridge the System-Software Gap. In *Proceedings of the 20th IEEE Digital Avionics Systems Conference*. October 2001.

[PM95] D. L. Parnas, J. Madey. Functional Documents for Computer Systems. *Science of Computer Programming* 25(1):41–61, 1995.

[SS93] G. Schmidt, T. Ströhlein. *Relations and Graphs: Discrete Mathematics for Computer Scientists*. EATCS Monographs on Theoretical Computer Science. Springer-Verlag, 1993.

[VS90] A. Van Schouwen. The A-7 REquirements Model: Re-examination for Real-Time Systems and an Application to Monitoring Systems. Technical report 90-276, Queens University, Ontario, Canada, 1990.

[WL03] A. Wassyng, M. Lawford. Lessons Learned from a Successful Implementation of Formal Methods in an Industrial Project. In Araki et al. (eds.), *FME 2003*. Lecture Notes in Computer Science 2805, pp. 133–153. Springer, 2003.

[WL06] A. Wassyng, M. Lawford. Software Tools for Safety-Critical Software Development. *International Journal on Software Tools for Technology Transfer (STTT)* 8(4–5):337–354, August 2006.