Refining Exceptions in Four-Valued Logic ⋆

Susumu NISHIMURA

Dept. of Mathematics, Graduate School of Science, Kyoto University
Sakyo-ku, Kyoto 606-8502, JAPAN
susumu@math.kyoto-u.ac.jp

December 25, 2009

Abstract. This paper discusses refinement of programs that may raise and catch exceptions. We show that exceptions are expressed by a class of predicate transformers built on Arieli and Avron's four-valued logic and develop a refinement framework for the four-valued predicate transformers. The resulting framework enjoys several refinement laws that are useful for stepwise refinement of programs involving exception handling and partial predicates. We demonstrate some typical usages of the refinement laws in the proposed framework by a few examples of program transformation.

1 Introduction

Program refinement has been intensively studied in the framework of refinement calculus [BvW98, Mor94]. Refinement calculus identifies each program with a predicate transformer and formally justifies refinement of programs by means of the so-called refinement relation that is induced from the logical entailment. Although refinement calculus is successfully applied to a certain extension of Dijkstra's guarded command language [Dij76], fundamental difficulties arise when we try to extend the language with *exceptions*.

First, since exceptional termination is not discriminated from non-termination in the predicate transformer semantics, a construct that catches exceptions would also catch non-termination, which is counter-intuitive from the operational point of view. Second, exceptions are not only raised explicitly by a command but also implicitly by a failure of computation (e.g., division by zero). In this paper, we argue the latter type of exceptions that are raised by *partial predicates*, whose truth value may not be defined. Partiality poses a foundational issue in developing the theory of refinement based on the classical logic, in which partiality is ruled out. For example, in Dijkstra's predicate transformer semantics, the weakest pre-condition of the conditional statement **if** p **then** S **else** T is specified by a formula $(p \Rightarrow S(\varphi)) \wedge (\neg p \Rightarrow T(\varphi))$ for any post-condition φ, but this formula is nonsensical in the classical logic when p is undefined.

King and Morgan [KM95] proposed a solution to the first problem by developing an extension to the traditional predicate transformer semantics for a language in which

⋆ This is a full version of the paper to appear in the post-proceedings of 19th International Symposium on Logic-Based Program Synthesis and Transformation (LOPSTR 2009), LNCS.

exceptions are explicitly raised by the command **exit** and are caught by the exception block construct **try** S **catch** T[1]. They specified the input of each predicate transformer by a pair of post-conditions $\langle \varphi_n, \varphi_e \rangle$, rather than by a single post-condition, where they write $wp(S, \varphi_n, \varphi_e)$ for the weakest pre-condition that guarantees the program S either to normally terminate establishing φ_n or to exceptionally terminate establishing φ_e. The weakest pre-conditions for **exit** and the exception block are given as below:

$$wp(\mathbf{exit}, \varphi_n, \varphi_e) = \varphi_e, \qquad wp(\mathbf{try}\ S\ \mathbf{catch}\ T, \varphi_n, \varphi_e) = wp(S, \varphi_n, wp(T, \varphi_n, \varphi_e)).$$

The intuition behind these specifications are explained as follows. The **exit** command immediately causes an exceptional termination. Thus, the command is guaranteed to terminate (exceptionally) establishing the pre-condition φ_e. The exception block **try** S **catch** T executes S and terminates normally, if no exception is raised; If an exception is ever raised, the raised exception is caught and then processed by T to resume normal execution. Therefore, for the exception block to terminate establishing the pair $\langle \varphi_n, \varphi_e \rangle$ of post-conditions, S is either to normally terminate establishing φ_n or to exceptionally terminate establishing $wp(T, \varphi_n, \varphi_e)$, which guarantees T to terminate establishing the pair of conditions $\langle \varphi_n, \varphi_e \rangle$.

In this paper, we propose a refinement calculus for a language that may raise and catch exceptions, where exceptions can be raised not only by the **exit** command explicitly but also by the evaluation of partial predicates implicitly. For this, we develop our theory of program refinement in a predicate transformer semantics based on Arieli and Avron's four-valued logic [AA96, AA98].

The four-valued predicate transformer semantics can be easily derived from King and Morgan's, in the following way. First, we identify each statement S by a predicate transformer that maps a pair of (classical) predicates $\langle \varphi_n, \varphi_e \rangle$ to another pair of predicates $\langle \varphi'_n, \varphi_e \rangle$, where φ'_n is the weakest pre-condition computed by King and Morgan's predicate transformer wp. This definition is intended to guarantee the program S either to normally terminate establishing φ_n or to exceptionally terminate establishing φ_e, whenever the preceding statement normally terminates establishing φ'_n or exceptionally terminates establishing φ_e. Notice that the condition φ_e for exceptional termination is left unchanged by the transformer because no statement can cancel exceptional termination caused by the preceding statements.

Next, let us designate a classical predicate by a total function from the set of states to $\{0, 1\}$, where 0 and 1 designates the two classical truth values (i.e., *false* and *true*, respectively). Then we identify each pair of predicates $\langle \varphi_n, \varphi_e \rangle$ by a single *four-valued predicate* φ such that $\varphi(\sigma) = \langle \varphi_n(\sigma), \varphi_e(\sigma) \rangle$ for every state σ. The range of the four-valued predicate is $\{\langle 1, 0 \rangle, \langle 0, 1 \rangle, \langle 0, 0 \rangle, \langle 1, 1 \rangle\}$, which we designate by $\mathbf{t}$, $\mathbf{f}$, $\bot$, and $\top$, respectively. This structure with four truth values gives rise to the so called Belnap's four-valued logic [Bel77], which has been studied by Ginsberg in the generalized setting of bilattices [Gin88] and was further examined by Fitting [Fit94]. Arieli and Avron [AA96, AA98] introduced the notion of logical bilattices and developed the corresponding proof system.

The four-valued logic provides a firm logical basis for refining exceptions, as the original refinement calculus does for refining the guarded command language. The

[1] This extends the exception block construct proposed in [KM95] with exception handling.

constructs for exceptions and others as well are concisely specified by the formulas of four-valued logic. The conditional control via partial predicates can be translated into a predicate transformer, where the undefinedness of partial predicates is denoted by the truth value $\perp$. The refinement relation is induced from the logical entailment (in the sense of four-valued logic), i.e., $S \sqsubseteq T$ iff $S(\varphi)$ entails $T(\varphi)$ for any post-condition φ.

We emphasize that we use the four-valued logic in two different ways. In the predicate transformer semantics, it is used for discriminating the possible termination behaviors (either, both, or none of normal termination and exceptional termination), while in modelling partial predicates, it is used as a many-valued logic that allows undefinedness. Although a three-valued logic would be sufficient for the latter purpose, we stick to the four-valued logic in developing the theory of refinement in order to achieve a smooth translation of conditional controls via partial predicates into four-valued predicate transformers. For a more neat characterization of partial predicates that adheres to the operational intuition, we also consider partial predicates in a three-valued sublogic, whose truth values are confined to **f**, **t**, and $\perp$. In later sections we exploit the properties of partial predicates in this three-valued sublogic.

Related work. It seems that there has been no attempt to formulate a predicate transformer semantics that gives a unifying account for both exceptions and partial predicates. The exception mechanism was formulated in terms of predicate transformers in King and Morgan's refinement calculus [KM95], which was further elaborated in [Wat02]. Partial predicates are out of their concern, however. (If partial predicates are ignored at all, the refinement calculus of theirs and that of ours are essentially the same.)

Partial predicates in program logic have been intensively studied in the context of three-valued logic. For instance, the VDM specification language deals with undefinedness in a logic called LPF [Jon86, JM94]; Bono et al. [BK06] formulated a Hoare logic with a third truth value denoting 'crash' of execution. Many other variants of three-valued logic have been proposed for the sake of a better treatment of partiality [Owe93, JM94, MB99]. The three-valued logic, however, is not suitable for describing a predicate transformer semantics for exceptions, because the underlying predicate logic must be able to discriminate the four different status of termination. Hähnle [Häh05] discussed that partiality should be dealt by underspecification, rather than by a value representing undefinedness in a many-valued logic. His argument is, however, about predicates in specification statements and does not consider exception catching.

Huisman and Jacobs [HJ00] extended Hoare logic to deal with abrupt (exceptional) termination in Java programming language. They also formulated the mechanism of catching exceptions in their program logic by representing several different modes of exceptional termination by different forms of Hoare triple. In contrast to theirs, ours simply supports a single mode of exceptional termination. This does not imply ours are less expressive than theirs. Ours can simulate different modes of exceptional termination by introducing a special variable indicating the mode of termination.

Outline. The rest of the paper is organized as follows. Section 2 introduces the notion of bilattices and the four-valued logic. Section 3 specifies a set of program statements as four-valued predicate transformers and we identify the class of predicate transformers. The statements involve **exit**, exceptions blocks, and conditional controls via partial

predicates. The logical connectives for partial predicates are also discussed. In Section 4, we investigate a set of refinement laws that hold for these statements and logical connectives. In Section 5, we apply the refinement laws to carry out some program transformations. Finally, Section 6 concludes the paper.

2 The bilattice *FOUR* and the Four-Valued Logic

2.1 The bilattice *FOUR* of four truth values

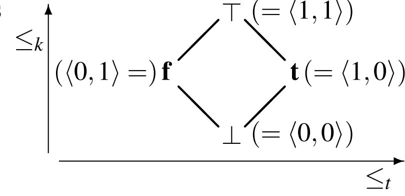

Fig. 1. The bilattice of four truth values

Let *TWO* be the lattice of classical truth values of 0, 1 with the trivial order $0 < 1$. The bilattice *FOUR* is a structure obtained by a product construction $\mathit{TWO} \odot \mathit{TWO}$: it consists of four elements $\langle 1,0\rangle$, $\langle 0,1\rangle$, $\langle 0,0\rangle$, and $\langle 1,1\rangle$, which are alternatively written $\mathbf{t}$, $\mathbf{f}$, $\bot$, and $\top$, respectively. The bilattice has two lattice structures simultaneously (see the double Hasse diagram of Figure 1), each characterized by the partial orders $\leq_t$ and $\leq_k$ defined below.[2]

$$\langle x_1,y_1\rangle \leq_t \langle x_2,y_2\rangle \text{ iff } x_1 \leq x_2 \text{ and } y_2 \leq y_1,$$
$$\langle x_1,y_1\rangle \leq_k \langle x_2,y_2\rangle \text{ iff } x_1 \leq x_2 \text{ and } y_1 \leq y_2.$$

The $\leq_t$ order (resp. $\leq_k$ order) induces the meet $\wedge$ and join $\vee$ operators (resp. meet $\otimes$ and join $\oplus$ operators) The definitions are given below, where $\sqcap$ and $\sqcup$ stand for the meet and join in *TWO*, respectively.

$$\langle x_1,y_1\rangle \wedge \langle x_2,y_2\rangle = \langle x_1 \sqcap x_2, y_1 \sqcup y_2\rangle, \qquad \langle x_1,y_1\rangle \vee \langle x_2,y_2\rangle = \langle x_1 \sqcup x_2, y_1 \sqcap y_2\rangle,$$
$$\langle x_1,y_1\rangle \otimes \langle x_2,y_2\rangle = \langle x_1 \sqcap x_2, y_1 \sqcap y_2\rangle, \qquad \langle x_1,y_1\rangle \oplus \langle x_2,y_2\rangle = \langle x_1 \sqcup x_2, y_1 \sqcup y_2\rangle.$$

In addition, negation $\neg$ is defined by $\neg\langle x,y\rangle = \langle y,x\rangle$ as an operator that inverts the $\leq_t$ order but keeps the $\leq_k$ order.

In *FOUR*, the operations $\vee$ and $\wedge$ are De Morgan dual of each other, i.e., $\neg(x \vee y) = \neg x \wedge \neg y$ and $\neg(x \wedge y) = \neg x \vee \neg y$, while $\oplus$ and $\otimes$ are De Morgan self-dual, i.e., $\neg(x \oplus y) = \neg x \oplus \neg y$ and $\neg(x \otimes y) = \neg x \otimes \neg y$. The four values are related with each other by means of $\vee$, $\wedge$, $\oplus$, and $\otimes$, e.g., $\bot \vee \mathbf{f} = \bot$, $\mathbf{t} \oplus \mathbf{f} = \top$, $x \vee \bot = x \otimes \mathbf{t}$.

The bilattice *FOUR* is *distributive*, i.e., the four lattice operations $\wedge$, $\vee$, $\otimes$, and $\oplus$ distribute over each other, e.g., $x \oplus (y \wedge z) = (x \oplus y) \wedge (x \oplus z)$. A distributive bilattice is also *interlaced*, that is, each of the four lattice operations is monotonic with respect to both $\leq_t$ and $\leq_k$, e.g., $y \leq_t z$ implies $x \otimes y \leq_t x \otimes z$.

The bilattice structure can be made into a *logical bilattice* that provides suitable notions of implications in four-valued logic [AA96]. With $\mathcal{D} = \{\mathbf{t}, \top\}$ being the set of

[2] In the literature, $\leq_t$ is often regarded as the *degree of truth* and $\leq_k$ as the *amount of information*. Given a product $\langle x,y\rangle$ of classical truth values, x represents the amount of evidence *for* an assertion, while y represents the amount of evidence *against* it. However, one should refrain from sticking to this particular interpretation, when the four-valued logic is used for discriminating the possible termination behaviors in the predicate transformer semantics.

designated truth values, which are the values recognized as (at least) known to be true, the bilattice *FOUR* is made into a logical bilattice with two implication connectives, called *weak implication* $\supset$ and *strong implication* $\rightarrow$, which are defined as below:

$$x \supset y \triangleq \begin{cases} \mathbf{t} \ (x \notin \mathcal{D}) \\ y \ \text{(otherwise)} \end{cases}, \qquad x \rightarrow y \triangleq (x \supset y) \land (\neg y \supset \neg x).$$

Using strong implication, we define the equivalence $x \leftrightarrow y$ by $(x \rightarrow y) \land (y \rightarrow x)$.

2.2 The four-valued predicate logic

We give a four-valued first-order predicate logic, based on the Arieli and Avron's four-valued propositional system. (Extension to the predicate logic is straightforward, as mentioned in [AA96].) We assume the set Value of program values (integers, etc.) and the set Var of program variables. Let us define State to be the set of total functions from Var to Value. Given $\sigma \in \mathsf{State}$ and $X \in \mathsf{Var}$, $\sigma(X)$ denotes the value that is assigned to the program variable X in the state σ.

Four-valued predicates, denoted by p, q, etc., are total functions from State to the four truth values in *FOUR*. The four-valued predicates form a bilattice, where the two partial orders $\leq_t$ and $\leq_k$ and logical connectives $\land$, $\lor$, $\otimes$, $\oplus$, $\neg$, $\supset$, $\rightarrow$, $\leftrightarrow$ are accordingly defined in the pointwise way. That is, for every state σ, $p \leq_t q$ (resp. $p \leq_k q$) holds iff $p(\sigma) \leq_t q(\sigma)$ (resp. $p(\sigma) \leq_k q(\sigma)$), and also logical connectives are defined by $(p \lor q)(\sigma) \triangleq p(\sigma) \lor q(\sigma)$, $(\neg p)(\sigma) \triangleq \neg p(\sigma)$, etc. In abuse of notations, we will also denote a constant predicate by the constant itself. That is, we write $\mathbf{t}$ for a predicate p such that $p(\sigma) = \mathbf{t}$ for every state σ; Similarly for $\mathbf{f}, \bot$, and $\top$.

It is easy to verify that the bilattice of the four-valued predicates is distributive, interlaced, bounded, and complete. (A bilattice is *complete*, if the two lattices induced by the partial orders $\leq_t$ and $\leq_k$ are both complete.) The completeness indicates that we may also define quantification by means of the infinite join or meet. Given a family of predicates $\{p(i) \mid i \in \mathsf{Value}\}$, we define the universal quantification (resp. existential quantification) over i of predicate $p(i)$ by $\forall i.p(i) \triangleq \bigwedge_i p(i)$ (resp. $\exists i.p(i) \triangleq \bigvee_i p(i)$)

The above mentioned structure of logical bilattice induces a four-valued predicate logic [AA96], which has a Gentzen-style proof system for sequents of the form $p_1, \cdots, p_n \vdash q_1, \cdots, q_m$ $(n, m \geq 0)$. The sequent corresponds to the consequence relation $p_1, \cdots, p_n \models q_1, \cdots, q_m$, which means, for any state σ, if $p_i(\sigma) \in \mathcal{D}$ for all i, then $q_j(\sigma) \in \mathcal{D}$ for some j. We say a predicate p is *valid* iff $\models p$ holds (i.e., $p(\sigma) \in \mathcal{D}$ for any state σ).

Notice that the four-valued logic is a non-classical logic. In particular it is paraconsistent and does not admit the law of the excluded middle, that is, we have neither $\vdash p \lor \neg p$ nor $p \land \neg p \vdash q$. The connectives $\supset$, $\rightarrow$, and $\leftrightarrow$ are a logical implication or an equivalence in the following sense: $\models p \supset q$ iff $p \models q$; $\models p \rightarrow q$ iff $p \leq_t q$; $\models p \leftrightarrow q$ iff $p = q$. Furthermore the logical equivalence $\leftrightarrow$ is a congruence: $\models p \leftrightarrow q$ implies $\models \Theta(p) \leftrightarrow \Theta(q)$ for any formula scheme Θ. For further details of the proof system and logical properties of the four-valued logic, see [AA96, AA98].

Throughout the paper, we follow the convention that the negation and quantifications bind most tightly, while implications do least tightly and associate to right. We do not impose any particular precedence between $\lor$, $\land$, $\oplus$, and $\otimes$.

Finally, let us introduce some notations that are related to states. A *program expression* e is a total function from State to Value. We write $\sigma[X\backslash v]$ for the state obtained by updating the value assigned to the program variable X in the state σ by the value v. Similarly, we write $\sigma[X\backslash e]$ for an update of variable X with the value of expression e, that is, $\sigma[X\backslash e(\sigma)]$. Given a four-valued predicate p, we also write $p[X\backslash v]$ (resp. $p[X\backslash e]$) for the predicate q such that $q(\sigma) = p(\sigma[X\backslash v])$ (resp. $q(\sigma) = p(\sigma[X\backslash e])$) In particular, a predicate $p[X\backslash v]$ can be recognized as a predicate indexed by v ranging over Value. In abuse of notations, we may often confuse a program variable X with an expression e such that $e(\sigma) = \sigma(X)$. More generally, we may confuse numerical expressions and predicates with their pointwise extensions. For example, when we write $X+1 \geq Y$, it denotes a predicate q such that $q(\sigma) = \big(\sigma(X)+1 > \sigma(Y)\big)$, where $+$ is the binary integer addition and $\geq$ is the binary predicate such that $(v \geq v') = \mathbf{t}$ if v is greater than or equal to v' but $(v \geq v') = \mathbf{f}$ otherwise.

3 Predicate Transformers and Refinement

3.1 The lattice of predicate transformers

As we have argued earlier, a predicate transformer should be a function that maps a pair of predicates $\langle \varphi_n, \varphi_e \rangle$ to another pair $\langle \varphi'_n, \varphi_e \rangle$. We also require every predicate transformer to be *monotonic*.

Definition 3.1. *A pair of four-valued predicates p and p' is called an* exception matching pair *if $\mathbf{t} \oplus p = \mathbf{t} \oplus p'$ holds.*

A predicate transformer S over four-valued predicates is monotonic *if $S(\varphi) \leq_k S(\varphi')$ holds for every exception matching pair φ and φ' such that $\varphi \leq_k \varphi'$. S is* exception stable *if φ and $S(\varphi)$ are an exception matching pair, for every φ.*

Let PTran be the set of predicate transformers of four-valued predicates that are monotonic and exception stable. Then PTran is made into a bounded complete lattice as follows.

Theorem 3.1. *Let PTran be lattice induced by the partial order $\sqsubseteq$ by:*

$$S \sqsubseteq T \quad \textit{iff} \quad S(\varphi) \leq_k T(\varphi) \textit{ for any } \varphi,$$

where the join $\oplus$ and meet $\otimes$ operators are a pointwise extension of the corresponding logical connectives, i.e., $(S \oplus T)(\varphi) = S(\varphi) \oplus T(\varphi)$ and $(S \otimes T)(\varphi) = S(\varphi) \otimes T(\varphi)$. Then PTran is a bounded complete lattice.

The class PTran of predicate transformers are also closed under function composition, where we write $S;T$ to mean $(S;T)(\varphi) = S(T(\varphi))$ and intend a sequential execution of S followed by T. The meet $S \otimes T$ and join $S \oplus T$ in PTran, called *demonic choice* and *angelic choice*, respectively, are intended a non-deterministic choice between S and T: The demonic choice represents the least possible non-deterministic execution that the two statements agree, while the angelic choice represents the greatest possible one.

In order to verify that a refinement relation $S \sqsubseteq T$ holds, we need to show $S(\varphi) \leq_k T(\varphi)$ holds for every φ. There are several different ways to verify this.

$$
\begin{aligned}
\mathbf{skip}(\varphi) &\triangleq \varphi && \text{(skip)}\\
\big(X := e\big)(\varphi) &\triangleq (\mathbf{f}\oplus\varphi[X\backslash e])\otimes(\mathbf{t}\oplus\varphi) && \text{(assignment)}\\
\mathbf{abort}(\varphi) &\triangleq \mathbf{f}\otimes\varphi && \text{(non-termination)}\\
\mathbf{magic}(\varphi) &\triangleq \mathbf{t}\oplus\varphi && \text{(miracle)}\\
\mathbf{exit}(\varphi) &\triangleq (\mathbf{t}\oplus\varphi)\otimes\neg(\mathbf{t}\oplus\varphi) && \text{(exit)}\\
\mathbf{try}\ S\ \mathbf{catch}\ T &\triangleq \big(\mathbf{f}\oplus S\big((\mathbf{f}\oplus\varphi)\otimes\neg(\mathbf{f}\oplus T(\varphi))\big)\big)\otimes(\mathbf{t}\oplus\varphi) && \text{(exception handling)}\\
\{p\}(\varphi) &\triangleq \neg(p\supset\top)\otimes\varphi && \text{(assertion)}\\
[p](\varphi) &\triangleq (p\supset\bot)\oplus\varphi && \text{(assumption)}\\
\langle p\rangle(\varphi) &\triangleq ((p\supset\bot)\oplus\varphi)\otimes\neg((p\supset\top)\oplus\varphi) && \text{(conditional exit)}
\end{aligned}
$$

Fig. 2. Four-valued predicate transformers for program statements

Proposition 3.1. *For any* $S,T\in\mathsf{PTran}$ *and any four-valued predicate* φ, $S(\varphi)\leq_k T(\varphi)$ *iff* $S(\varphi)\leq_t T(\varphi)$ *iff* $\models S(\varphi)\to T(\varphi)$ *iff* $S(\varphi)\models T(\varphi)$ *iff* $S(\varphi)\vdash T(\varphi)$.

Thus we may verify $S\sqsubseteq T$ by checking the validity of $S(\varphi)\to T(\varphi)$ in the model of bilattice, which will be effective for the propositional cases. In case quantifiers are involved, we may resort to a formal proof deriving the sequent of the form $S(\varphi)\vdash T(\varphi)$. Further discussions on these alternative ways for validating refinement laws are found in Appendix A.

3.2 Predicate transformers for basic statements

Let us write $\langle\varphi_n,\varphi_e\rangle$ for the pair of predicates that a four-valued predicate φ encodes as we have argued in the introduction. When we define a predicate transformer in PTran, we often need to operate on each component of the pair separately. This can be easily expressed by the four-valued logic formulas. For example, given four-valued predicates p and q, we can express the pair $\langle p_n,q_e\rangle$ by the formula $(\mathbf{f}\oplus p)\otimes(\mathbf{t}\oplus q)$.[3] A simple calculation verifies this as follows:

$$(\mathbf{f}\oplus p)\otimes(\mathbf{t}\oplus q)=(\langle 0,1\rangle\oplus\langle p_n,p_e\rangle)\otimes(\langle 1,0\rangle\oplus\langle q_n,q_e\rangle)=\langle p_n,1\rangle\otimes\langle 1,q_e\rangle=\langle p_n,q_e\rangle.$$

In a similar way, we can verify that $(\mathbf{t}\oplus p)\otimes\neg(\mathbf{t}\oplus p)$ calculates $\langle p_e,p_e\rangle$ and $(\mathbf{f}\oplus p)\otimes\neg(\mathbf{f}\oplus p)$ does $\langle p_n,p_n\rangle$.

In Figure 2, we give the definitions of four-valued predicate transformers for a set of basic statements. (It is easy to verify that all of them are a member of PTran.)

- **skip** is the idle statement. It is an identity function and hence is a neutral element for the sequential composition, i.e., $\mathbf{skip};S=S;\mathbf{skip}=S$.

---

[3] There are different ways of expressing the same operation, e.g., $(\mathbf{t}\otimes p)\oplus(\mathbf{f}\otimes q)$.

- $X := e$ is the assignment statement. Given a post-condition $\langle \varphi_n, \varphi_e \rangle$, it calculates the weakest pre-condition $\varphi_n[X \backslash e]$ for normal termination and keeps the condition φ_e for exceptional termination unchanged. Note that this assignment is total and deterministic, that is, it always successfully assigns a unique value to the program variable. We will discuss partial assignments in Section 5.2.
- **abort** and **magic** are extremal elements, that is, the least and greatest elements of PTran, respectively. **abort**[4] represents a statement that is not guaranteed to terminate normally. On the other hand, **magic** represents a miraculous statement that always terminates normally, establishing any required post-condition (even falsity). They are a left-zero element of sequential composition, that is, **abort**$;S =$ **abort** and **magic**$;S =$ **magic**.
- **exit** is the statement that raises an exception. As we discussed earlier, it is characterized by a function that transforms every post-condition $\langle \varphi_n, \varphi_e \rangle$ into $\langle \varphi_e, \varphi_e \rangle$. Again **exit** is a left-zero element, i.e., **exit**$;S =$ **exit**.
- **try** S **catch** T is the exception handling statement. The statement calculates the weakest post-condition for normal termination given by King and Morgan's *wp* function and combines it with the condition for exceptional termination, using the formulas discussed above.
- $\{p\}$, $[p]$, and $\langle p \rangle$, which are called *assertion*, *assumption*, and *conditional exit*, respectively, are primitive forms of conditional controls, which decide how to continue the execution, depending on the value of the four-valued predicate p, which is called a *guard predicate*. They are all equivalent to **skip**, if the predicate p has a designated truth value (i.e., either **t** or $\top$); otherwise, $\{p\}$, $[p]$, $\langle p \rangle$ are equivalent to **abort**, **magic**, **exit**, respectively.[5]

The basic statements above can be combined to form a more complicated statement. A conditional statement **if** p **then** S **else** T, which may raise an exception when a partial predicate p evaluates to $\bot$, can be defined as follows:

$$\textbf{if } p \textbf{ then } S \textbf{ else } T \triangleq \langle p \vee \neg p \rangle ; (([p];S) \otimes ([p \supset \bot];T)).$$

The partiality of predicate p is first tested by the prepended $\langle p \vee \neg p \rangle$, which acts like **exit** if p has the value $\bot$ but like **skip** otherwise. Then, a demonic choice is made between the two branches, each prepended by an assumption statement. (The assumption statement in the unselected branch becomes **magic**, which is dismissed by the outer demonic choice.)

3.3 Logical connectives for partial predicates

In the above definition of conditional statements, we interpret $\top$ as an indication of true on the ground that $\top$ is a designated value in the four-valued logic, but this sometimes

[4] The name 'abort' is historical and is not necessarily adequate in the context of this paper, but we keep using it for compatibility.

[5] Some programming languages provide a feature called 'assertion', which is used for exceptionally terminating the execution when some critical violation of condition is detected. Note the difference from the assertion $\{p\}$, which is non-terminating when the test on p is false. The name 'assertion' is thus somewhat confusing but we keep using it for historical reason.

leads to a result that runs counter to the operational intuition. (For example, some of the laws given in Section 5.1 do not hold for arbitrary four-valued guard predicates.)

In order to obtain a more precise modelling of partial predicates that adheres to the operational intuition, let us consider *consistent* predicates [Fit94]: A four-valued predicate p is called consistent if $p(\sigma) \in \{\mathbf{t}, \mathbf{f} \bot\}$ for any σ. The class of consistent predicates forms a three-valued sublogic, whose logical operators $\wedge$ and $\vee$, a.k.a. strong Kleene connectives, are non-strict operators that avoid $\bot$ whenever possible. (For instance, both $\mathbf{f} \wedge \bot$ and $\bot \wedge \mathbf{f}$ are interpreted $\mathbf{f}$ rather than $\bot$.) Non-strictness implies that the strong Kleene connectives cannot be implemented in real programming languages.

We can define logical operators that are found in practical programming languages in the three-valued sublogic as follows. Following [Fit94], let us write $p : q$ for $((p \otimes t) \oplus \neg(p \otimes t)) \otimes q$. This derived formula $p : q$ has $\bot$ if p has $\mathbf{f}$ or $\bot$; otherwise, it has the value of q.

We can define a 'sequential' disjunction $\vec{\vee}$ and conjunction $\vec{\wedge}$ for any pair of consistent predicates p and q, as follows.

$$p \vec{\wedge} q \triangleq\ p \wedge (p : q) \qquad\qquad p \vec{\vee} q \triangleq\ p \vee (\neg p : q)$$

These operators are strict and evaluated sequentially from left to right: it becomes $\bot$ as soon as the left subformula p evaluates to $\bot$.

We can also define the weak Kleene connectives $\vee^w$ and $\wedge^w$ as the consensus of the corresponding two sequential connectives of opposite directions.

$$p \wedge^w q \triangleq\ (p \vec{\wedge} q) \otimes (q \vec{\wedge} p) \qquad\qquad p \vee^w q \triangleq\ (p \vec{\vee} q) \otimes (q \vec{\vee} p)$$

In contrast to the strong Kleene connectives, the value of these connectives is defined only if both of the subformulas are defined.

The strong Kleene connectives $\wedge$ and $\vee$, the sequential connectives $\vec{\wedge}$ and $\vec{\vee}$, and also the weak Kleene connectives $\wedge^w$ and $\vee^w$ are all De Morgan dual for each.

4 Refinement Laws for Statements

In the rest of this paper, we assume that guard predicates occurring in control statements are four-valued, unless explicitly stated otherwise. We will indicate wherever a guard predicate is required to be consistent. We further assume that, unless it is explicitly stated otherwise, numerical predicates (which we mentioned in the last paragraph of Section 2.2) are classical, that is, $p(\sigma) \in \{\mathbf{t}, \mathbf{f}\}$ for any σ. The class of classical predicates in the four-valued logic forms a classical sublogic, where the connectives $\vee$, $\wedge$, and $\neg$ substitute for the classical connectives of disjunction, conjunction, and negation, respectively, and implications $\supset$ and $\rightarrow$ substitute for the material implication. We may resort to the standard classical logical reasoning in this sublogic.

Let us first examine some basic refinement laws. From the distributivity of logical connectives, we can derive several distribution laws for demonic choice. The sequencing operator admits the left distribution law, i.e., $(S_1 \otimes S_2); T = (S_1; T) \otimes (S_2; T)$. (The right distribution law does not hold in general, though.) The exception handling statement also admits a distribution law $\mathbf{try}\ S_1 \otimes S_2\ \mathbf{catch}\ T = (\mathbf{try}\ S_1\ \mathbf{catch}\ T) \otimes (\mathbf{try}\ S_2\ \mathbf{catch}\ T)$.

By the interlaced property of logical connectives, all the statements introduced in the previous section are monotonic with respect to refinement of its substatements. For instance, $S_1 \otimes T_1 \sqsubseteq S_2 \otimes T_2$ holds if $S_1 \sqsubseteq S_2$ and $T_1 \sqsubseteq T_2$.

4.1 Refinement of conditional controls

The statement **skip** and the three conditional control statements are ordered by $\sqsubseteq$ as below.

$$\{p\} \sqsubseteq \textbf{skip} \sqsubseteq [p] \qquad (4.1) \qquad\qquad \{p\} \sqsubseteq \langle p \rangle \sqsubseteq [p] \qquad (4.2)$$

Further, the assertion (resp. the assumption) is monotonic (resp. anti-monotonic) with respect to the $\leq_t$ order over guard predicates. That is, if $p \to q$ is valid (or equivalently, $p \models q$), we have:

$$\{p\} \sqsubseteq \{q\} \qquad (4.3) \qquad\qquad [q] \sqsubseteq [p] \qquad (4.4)$$

In contrast, the conditional exit has no such particular (anti-)monotonicity property.

Provided that $p \to q$ is valid, we have:

$$\{p\} = \{p\};\{q\} = \{p\};[q] = \{p\};\langle q \rangle \qquad (4.5)$$
$$[p] = [p];\{q\} = [p];[q] = [p];\langle q \rangle \qquad (4.6)$$
$$\langle p \rangle = \langle p \rangle;\{q\} = \langle p \rangle;[q] = \langle p \rangle;\langle q \rangle \qquad (4.7)$$

The following laws indicate that successive conditional control statements of the same kind can be substituted with a single control statement which combines the guard formulas in the original statements by either $\wedge$, $\vec{\wedge}$, or $\wedge^w$.

$$\{p\};\{q\} = \{q\};\{p\} = \{p \wedge q\} = \{p \vec{\wedge} q\} = \{p \wedge^w q\} \qquad (4.8)$$
$$[p];[q] = [q];[p] = [p \wedge q] = [p \vec{\wedge} q] = [p \wedge^w q] \qquad (4.9)$$
$$\langle p \rangle;\langle q \rangle = \langle q \rangle;\langle p \rangle = \langle p \wedge q \rangle = \langle p \vec{\wedge} q \rangle = \langle p \wedge^w q \rangle \qquad (4.10)$$

Combining the laws (4.5) through (4.10), we can propagate a copy of a conditional control statement past one or more successive control statements (of possibly different kinds), e.g., $[p];\{q\};\langle r \rangle = [p];\{q\};\langle r \rangle;[p]$.

The disjunction in the guard of an assertion or an assumption can be substituted with an appropriate non-deterministic choice.

$$\{p \vee q\} = \{p\} \oplus \{q\} \qquad (4.11) \qquad\qquad [p \vee q] = [p] \otimes [q] \qquad (4.12)$$

From the fact that exactly one of the formulas p and $p \supset \bot$ can have a designated truth value at once, we obtain the following laws.

$$[p] \otimes [p \supset \bot] = \textbf{skip} \qquad (4.13) \qquad\qquad \langle p \rangle;\langle p \supset \bot \rangle = \textbf{exit} \qquad (4.15)$$
$$[p];[p \supset \bot] = \textbf{magic} \qquad (4.14)$$

Recall that we have used $\langle p \vee \neg p \rangle$ for testing partiality of the predicate p in the definition of conditional branch statement in Section 3.2. We will later make use of the following rules in order to exploit the implicit control structure indicated by sequential and weak Kleene connectives occurring in the test predicate.

$$\langle (p \vec{\wedge} q) \vee \neg (p \vec{\wedge} q) \rangle = \langle p \vee \neg p \rangle ; \langle \neg p \vee q \vee \neg q \rangle \tag{4.16}$$

$$\langle (p \vec{\vee} q) \vee \neg (p \vec{\vee} q) \rangle = \langle p \vee \neg p \rangle ; \langle p \vee q \vee \neg q \rangle \tag{4.17}$$

$$\langle (p \wedge^w q) \vee \neg (p \wedge^w q) \rangle = \langle (p \vee^w q) \vee \neg (p \vee^w q) \rangle = \langle p \vee \neg p \rangle ; \langle q \vee \neg q \rangle \tag{4.18}$$

When the predicate p is classical, the following laws hold.

$$\{ p \supset \bot \} = \{ \neg p \} \quad (4.19) \qquad [p \supset \bot] = [\neg p] \quad (4.20) \qquad \langle p \vee \neg p \rangle = \mathbf{skip} \quad (4.21)$$

4.2 Refinement of exceptions

The following refinement laws hold for exception statements.

$$\mathbf{exit}; S = \mathbf{exit} \quad (4.22) \qquad \mathbf{try}\ S; \langle p \rangle\ \mathbf{catch\ skip} = \mathbf{try}\ S\ \mathbf{catch\ skip} \quad (4.23)$$

An interesting subclass of PTran is the one that never raise exceptions. We would say that a transformer S never raises exceptions under any program context, if $\psi_n = \psi'_n$ holds whenever $\langle \psi_n, \varphi_e \rangle = S(\langle \varphi_n, \varphi_e \rangle)$ and $\langle \psi'_n, \varphi'_e \rangle = S(\langle \varphi_n, \varphi'_e \rangle)$. This is formally specified in terms of four-valued logic as follows.

Definition 4.1. *A predicate transformer $S \in \mathsf{PTran}$ is called* non-exceptional*, if $S(\varphi) = S(\varphi')$ holds whenever $\mathbf{f} \oplus \varphi = \mathbf{f} \oplus \varphi'$.*

It is easy to verify that all the statements introduced in Section 3, except for **exit** and $\langle p \rangle$, are non-exceptional if so are their substatements.

For any non-exceptional statement S, the following laws hold.

$$\mathbf{try}\ S\ \mathbf{catch}\ T = S \quad (4.24) \qquad \mathbf{try}\ S; \mathbf{exit}\ \mathbf{catch}\ T = S; T \quad (4.25)$$

5 Examples of Program Transformation by Stepwise Refinement

We will apply the refinement laws developed in the previous section to transformation of programs that involve exceptions and partial predicates.

5.1 Translating conjunctions and disjunctions into explicit controls

Programs often contain implicit controls by partial predicates. For example, a single conditional statement **if** $p \vec{\wedge} q$ **then** S **else** T contains several implicit information for control: The predicate $p \vec{\wedge} q$ evaluates from left to right; As soon as p evaluates to $\mathbf{f}$, the **else** clause is selected; exception is raised as soon as p evaluates to $\bot$; q is examined only if p evaluates to $\mathbf{t}$.

We justify this operational intuition via refinement by showing that the above conditional statement is equivalent to the nested conditional statement **if** p **then** (**if** q **then** S **else** T) **else** T. Let us first give a few subsidiary refinement laws.

$$[p \vec{\wedge} q \supset \bot] = [p \wedge q \supset \bot] = [p \supset \bot] \otimes [q \supset \bot]. \tag{5.1}$$

$$[p]; \langle \neg p \vee q \rangle = [p]; \langle q \rangle \quad \text{if } p \text{ is consistent} \tag{5.2}$$

$$\langle p \vee \neg p \rangle; [p \supset \bot] = \langle p \vee \neg p \rangle; [p \supset \bot]; \langle \neg p \vee q \vee \neg q \rangle \quad \text{if } p \text{ is consistent} \tag{5.3}$$

Then we can carry out the following derivation, provided p is consistent.

$$\begin{aligned}
&\textbf{if } p \vec{\wedge} q \textbf{ then } S \textbf{ else } T = \langle (p \vec{\wedge} q) \vee \neg (p \vec{\wedge} q) \rangle; ([p \vec{\wedge} q]; S \otimes [p \vec{\wedge} q \supset \bot]; T) \\
&= \langle p \vee \neg p \rangle; \langle \neg p \vee q \vee \neg q \rangle; ([p]; [q]; S \otimes [p \supset \bot]; T \otimes [q \supset \bot]; T)) \\
&\qquad \text{— by (4.16), (4.9), (5.1), and distributivity} \\
&= \langle p \vee \neg p \rangle; [p]; \langle \neg p \vee q \vee \neg q \rangle; ([p]; [q]; S \otimes [p \supset \bot]; T \otimes [q \supset \bot]; T) \\
&\qquad \otimes \langle p \vee \neg p \rangle; [p \supset \bot]; \langle \neg p \vee q \vee \neg q \rangle; ([p]; [q]; S \otimes [p \supset \bot]; T \otimes [q \supset \bot]; T) \\
&\qquad \text{— by (4.13) and distributivity} \\
&= \langle p \vee \neg p \rangle; \Big([p]; \langle q \vee \neg q \rangle; ([q]; S \otimes [q \supset \bot]; T) \otimes [p \supset \bot]; T \otimes [p \supset \bot]; [q \supset \bot]; T \Big) \\
&\qquad \text{— by (5.2), (5.3), (4.6), (4.7), (4.9), (4.10), (4.14), and distributivity} \\
&= \langle p \vee \neg p \rangle; \Big([p]; \langle q \vee \neg q \rangle; ([q]; S \otimes [q \supset \bot]; T) \otimes [p \supset \bot]; T \Big) \quad \text{— by (4.9), (4.4)} \\
&= \textbf{if } p \textbf{ then } (\textbf{if } q \textbf{ then } S \textbf{ else } T) \textbf{ else } T.
\end{aligned}$$

We can also derive a law for the sequential disjunction:

$$\textbf{if } p \vec{\vee} q \textbf{ then } S \textbf{ else } T = \textbf{if } p \textbf{ then } S \textbf{ else } (\textbf{if } q \textbf{ then } S \textbf{ else } T),$$

where p is consistent. For the weak Kleene connectives, we have similar laws:

$$\textbf{if } p \wedge^w q \textbf{ then } S \textbf{ else } T = \textbf{if } p \textbf{ then } (\textbf{if } q \textbf{ then } S \textbf{ else } T) \textbf{ else } \langle q \vee \neg q \rangle; T \quad \text{and}$$
$$\textbf{if } p \vee^w q \textbf{ then } S \textbf{ else } T = \textbf{if } p \textbf{ then } \langle q \vee \neg q \rangle; S \textbf{ else } (\textbf{if } q \textbf{ then } S \textbf{ else } T),$$

where p need not be consistent.

5.2 Refining exception handling

Let us apply our refinement laws to a larger program. In the development, we will make use of the technique that propagates context information via the assertion statement [LvW97, Gro00]. Below we list several non-trivial laws for propagating context information.

$$\{p\}; X := e \sqsubseteq X := e; \{\exists v. (p[X \backslash v] \wedge X = e[X \backslash v])\} \tag{5.4}$$

$$\{p\}; [q] \sqsubseteq [q]; \{p \wedge q\} \tag{5.5}$$

$$\{p\}; \langle q \rangle \sqsubseteq \langle q \rangle; \{p \wedge q\} \tag{5.6}$$

$$\{p\};\textbf{if } q \textbf{ then } S \textbf{ else } T \sqsubseteq \textbf{if } q \textbf{ then } (\{p \wedge q\};S) \textbf{ else } (\{p \wedge (q \supset \bot)\};T) \quad (5.7)$$

$$\textbf{if } q \textbf{ then } (S;\{p\}) \textbf{ else } (T;\{q\}) \sqsubseteq (\textbf{if } q \textbf{ then } S \textbf{ else } T);\{p \vee q\} \quad (5.8)$$

$$\{p\};\textbf{try } S \textbf{ catch } T \sqsubseteq \textbf{try } \{p\};S \textbf{ catch } T \quad (5.9)$$

$$\textbf{try } S;\{p\} \textbf{ catch } (T;\{q\}) \sqsubseteq (\textbf{try } S \textbf{ catch } T);\{p \vee q\}; \quad (5.10)$$

Let us consider the following program S_0 that implements a numerical algorithm.

$$S_0 \triangleq X := N;\textbf{try repeat } Y := X; X := (Y \times Y + N) \div (2 \times Y) \textbf{ until } X \geq Y \textbf{ catch skip}.$$

This program computes the integral value of $\sqrt{N}$ for non-negative integer N, based on the Newton-Raphson method [PTVF07], and assigns the answer to the variable Y. In the **repeat** $\cdots$ **until** loop, the integer division operator $\div$ may raise an exception due to division-by-zero, in which case, however, the exception is caught and the execution normally terminates with a correct answer.

Since PTran is a bounded complete lattice, each loop statement is specified by the least fixpoint $\mu.\mathcal{F}$ of a function $\mathcal{F} \in \mathsf{PTran} \to \mathsf{PTran}$ that is monotonic w.r.t. refinement order $\sqsubseteq$ [BvW98]. The loop statement in S_0 is given by the least fixpoint of the function:

$$\mathcal{F}(T) \triangleq Y := X; X := (Y \times Y + N) \div (2 \times Y); \textbf{if } X \geq Y \textbf{ then skip else } T.$$

In order to express the partial assignment $X := (Y \times Y + N) \div (2 \times Y)$, which may raise exception due to division-by-zero, we interpret it by the compound statement $\langle \neg(Y = 0) \rangle; X := (Y \times Y + N) \div' (2 \times Y)$, where $\div'$ is a total extension of $\div$ such that division by zero yields a fixed constant value (say, 0) instead of being undefined.

In the following derivation, we refine the original program S_0, with the assumption $N \geq 0$, into a program that makes no uses of exceptional statements.

$$\begin{aligned}\{N \geq 0\};S_0 &\sqsubseteq X := N; \{X = N \wedge N \geq 0\}; \textbf{try } \mu.\mathcal{F} \textbf{ catch skip} && \text{— by (5.4)}\\ &\sqsubseteq X := N; ((\textbf{try } [\neg(X = 0)]; \{0 < X \leq N\}; \mu.\mathcal{F} \textbf{ catch skip}) \otimes \\ &\qquad (\textbf{try } [\neg(X = 0) \supset \bot]; \{\neg(X = 0) \supset \bot\}; \mu.\mathcal{F} \textbf{ catch skip}))\\ &\qquad\qquad \text{— by (5.9), (4.13), (4.3), (4.1), (4.6), and distributivity.}\end{aligned}$$

The left substatement of the demonic choice is refined as follows. (We defer proofs of some lemmas to Appendix. Lemma B.1 indicates that $0 < X \leq N$ is a loop invariant and lemma B.2 says that the fixpoint operator on PTran preserves non-exceptionality.)

$$\begin{aligned}&\textbf{try } [\neg(X = 0)]; \{0 < X \leq N\}; \mu.\mathcal{F} \textbf{ catch skip}\\ \sqsubseteq\ &\textbf{try } [\neg(X = 0)]; \textbf{repeat } \{0 < X \leq N\}; Y := X; X := (Y \times Y + N) \div' (2 \times Y)\\ &\qquad \textbf{until } X \geq Y \textbf{ catch skip} \qquad \text{— by lemma B.1}\\ =\ &[\neg(X = 0)]; \textbf{repeat } \{0 < X \leq N\}; Y := X; X := (Y \times Y + N) \div' (2 \times Y)\\ &\qquad \textbf{until } X \geq Y \textbf{ catch skip} \qquad \text{— by (4.24) and non-exceptionality from lemma B.2}\\ =\ &[\neg(X = 0)]; \textbf{repeat } Y := X; \{\neg(Y = 0)\}; \langle \neg(Y = 0) \rangle; X := (Y \times Y + N) \div' (2 \times Y)\\ &\qquad \textbf{until } X \geq Y \qquad \text{— by (5.4), (4.3), and (4.5)}\\ =\ &[\neg(X = 0)]; \textbf{repeat } Y := X; X := (Y \times Y + N) \div (2 \times Y) \textbf{ until } X \geq Y \quad \text{— by (4.1)}\end{aligned}$$

For the other substatement of the choice, we derive:

$$\begin{aligned}
& \textbf{try } \big[\neg(X=0) \supset \bot\big];\big\{\neg(X=0) \supset \bot\big\};\mu.\mathcal{F} \textbf{ catch skip} \\
\sqsubseteq\ & \textbf{try } \big(\big[\neg(X=0) \supset \bot\big];Y := X;\big\{\neg(Y=0) \supset \bot\big\};\big\langle\neg(Y=0)\big\rangle; \\
& \quad X := (Y \times Y + N) \div' (2 \times Y);\textbf{if } X \geq Y \textbf{ then skip else } \mu.\mathcal{F}\big) \textbf{ catch skip} \\
& \qquad\qquad\qquad\qquad\qquad\qquad \text{— fixpoint; by (5.4)} \\
=\ & \textbf{try } \big[\neg(X=0) \supset \bot\big];Y := X;\big\{\neg(Y=0) \supset \bot\big\};\textbf{exit catch skip} \\
& \qquad\qquad\qquad\qquad\qquad\qquad \text{— by (4.6), (4.10), and (4.15)} \\
\sqsubseteq\ & \big[\neg(X=0) \supset \bot\big];Y := X \qquad\qquad \text{— by (4.25) and (4.1).}
\end{aligned}$$

Therefore the derivation ends up with:

$$\begin{aligned}
S_0 \sqsubseteq\ & X := N;\big\langle\neg(X=0) \vee \neg\neg(X=0)\big\rangle;\big(\big[\neg(X=0)\big];\mu.\mathcal{F} \otimes \big[\neg(X=0) \supset \bot\big];Y := X\big) \\
& \qquad\qquad\qquad\qquad\qquad\qquad \text{— by (4.21)} \\
=\ & X := N;\textbf{if } \neg(X=0) \textbf{ then repeat } Y := X;X := (Y \times Y + N) \div (2 \times Y) \textbf{ until } X \geq Y \\
& \qquad\qquad \textbf{else } Y := X.
\end{aligned}$$

6 Conclusion and Future Work

We proposed a refinement calculus for refining exceptions in programs. In order to model the normal termination as well as the exceptional termination in a single unified platform, we developed a four-valued predicate transformer semantics, which is based on Arieli and Avron's four-valued logic [AA96]. The programming constructs for raising and catching exceptions can be concisely expressed by the formulas of four-valued logic in this framework. In particular, we allow partial predicates in the conditional control statements in order to model exceptions that are raised implicitly when the predicate in a conditional statement is undefined. The four-valued logic provides a fruitful field for justifying refinement of programs that involve both explicit and implicit controls by exceptions.

This paper, with a few deviations, dealt with concrete program statements such as assignment, (conditional) exit, etc. Future research will concern abstract statements such as non-deterministic (possibly partial) assignment and general specification statement (which allows the uses of partial pre- and post-conditions) and also the methodology for deriving concrete programs from these abstract statements.

Acknowledgment I thank anonymous reviewers for their suggestions and comments. This work was supported by JSPS KAKENHI(20500011).

References

[AA96] Ofer Arieli and Arnon Avron. Reasoning with logical bilattices. *Journal of Logic, Language, and Information*, 5(1):25–63, 1996.

[AA98] Ofer Arieli and Arnon Avron. The value of four values. *Artificial Intelligence*, 102(1):97–141, 1998.

[Bel77] N. D. Belnap. A useful four-valued logic. In G. Epstein and J. M. Dunn, editors, *Modern Uses of Multiple-Valued Logic*, pages 7–37. Reidel Publishing Company, 1977.

[BK06] Viviana Bono and Manfred Kerber. Extending Hoare calculus to deal with crash. Technical Report CSR-06-08, School of Computer Science, The University of Birmingham, 2006.

[BvW98] Ralph-Johan Back and Joakim von Wright. *Refinement Calculus: A Systematic Introduction*. Springer-Verlag, 1998. Graduate Texts in Computer Science.

[Dij76] Edsger W. Dijkstra. *A Discipline of Programming*. Prentice-Hall, 1976.

[Fit94] Melvin Fitting. Kleene's three-valued logics and their children. *Fundamenta Informaticae*, 20(1/2/3):113–131, 1994.

[Gin88] Matthew L. Ginsberg. Multivalued logics: A uniform approach to reasoning in artificial intelligence. *Computational Intelligence*, 4:265–316, 1988.

[Gro00] Lindsay John Groves. *Evolutionary Software Development in the Refinement Calculus*. PhD thesis, Victoria University of Wellington, 2000.

[Häh05] Reiner Hähnle. Many-valued logic, partiality, and abstraction in formal specification languages. *Logic Journal of the IGPL*, 13(4):415–433, 2005.

[HJ00] Marieke Huisman and Bart Jacobs. Java program verification via a Hoare logic with abrupt termination. In *Fundamental Approaches to Software Engineering (FASE 2000)*, volume 1783 of *LNCS*, pages 284–303, 2000.

[JM94] Cliff B. Jones and C. A. Middelburg. A typed logic of partial functions reconstructed classically. *Acta Informatica*, 31(5):399–430, 1994.

[Jon86] Cliff B. Jones. *Systematic Software Development Using VDM*. Prentice-Hall International Series in Computer Science. Prentice-Hall International, 1986.

[KM95] Steve King and Carroll Morgan. Exits in the refinement calculus. *Formal Aspects of Computing*, 7(1):54–76, 1995.

[LvW97] Linas Laibinis and Joakim von Wright. Context handling in the refinement calculus framework. Technical Report 118, TUCS Technical Report, 1997.

[MB99] Joseph M. Morris and Alexander Bunkenburg. E3: A logic for reasoning equationally in the presence of partiality. *Science of Computer Programming*, 34(2):141–158, 1999.

[Mor94] Carroll Morgan. *Programming from specifications*. Prentice-Hall International Series in Computer Science. Prentice-Hall International, 2nd edition, 1994.

[Owe93] Olaf Owe. Partial logics reconsidered: A conservative approach. *Formal Aspects of Computing*, 5(3):208–223, 1993.

[PTVF07] William H. Press, Saul A. Teukolsky, William T. Vetterling, and Brian P. Flannery. *Numerical Recipes: The Art of Scientific Computing*. Cambridge University Press, 3rd edition, 2007.

[Wat02] Geoffrey Watson. Refining exceptions using King and Morgan's exit construct. In *9th Asia-Pacific Software Engineering Conference (APSEC 2002)*, pages 43–51. IEEE Computer Society, 2002.

Appendix

A Proving Refinement Laws in Four-Valued Logic

As we have seen in Section 3.1, a refinement relation $S \sqsubseteq T$ can be verified either by showing the validity of $S(\varphi) \rightarrow T(\varphi)$ or by deriving the sequent of the form $S(\varphi) \vdash T(\varphi)$ by a formal proof. The set of inference rules of Arieli and Avron's four-valued logic [AA96] is given in Figure 3 for reference purpose.

The former method can be applied to most of the refinement laws given in this paper, since they are propositional (i.e., the formula $S(\varphi) \rightarrow T(\varphi)$ is quantifier-free). The task of checking the validity for propositional formulas can be mechanized.

The latter method can be used for justifying refinement relations in a more human understandable way. For example, the following derivation proves $\{p\};\{q\} \sqsubseteq \{p \wedge q\}$. (The proof of the other direction of refinement is similar.)

1. $p, \neg\top, q, \neg\top, \varphi \vdash p$	Initial sequent
2. $p, \neg\top, q, \neg\top, \varphi \vdash q$	Initial sequent
3. $p, \neg\top, q, \neg\top, \varphi \vdash p \wedge q$	$1, 2, [\vdash \wedge]$
4. $p, \neg\top, q, \neg\top, \varphi \vdash \neg\top$	Initial sequent
5. $p, \neg\top, q, \neg\top, \varphi \vdash \neg(p \wedge q \supset \top)$	$3, 4, [\vdash \neg \supset]$
6. $\neg(p \supset \top), \neg(q \supset \top), \varphi \vdash \neg(p \wedge q \supset \top)$	$5, [\neg \supset \vdash]$
7. $\neg(p \supset \top), \neg(q \supset \top), \varphi \vdash \varphi$	Initial sequent
8. $\neg(p \supset \top), \neg(q \supset \top), \varphi \vdash \neg(p \wedge q \supset \top) \otimes \varphi$	$6, 7, [\vdash \otimes]$
9. $\neg(p \supset \top) \otimes \neg(q \supset \top) \otimes \varphi \vdash \neg(p \wedge q \supset \top) \otimes \varphi$	$8, [\otimes \vdash]$

In the above, the derivation is shown in a linear format: Each line contains a sequent and indicates the set of premises and the inference rule that are used for deriving the sequent.

We may simplify the verification task for a refinement relation that is accompanied with a side condition by appealing to the logical equivalence. (Recall that the logical equivalence in four-valued logic is a congruence.) For example, let us show the law (4.24). Suppose S is non-exceptional. Then it holds that $S\big(\mathbf{f} \oplus ((\mathbf{f} \oplus \varphi) \otimes \neg(\mathbf{f} \oplus T(\varphi)))\big) = S(\varphi)$, because we have:

$$
\begin{aligned}
& \mathbf{f} \oplus \big(\mathbf{f} \oplus \big((\mathbf{f} \oplus \varphi) \otimes \neg(\mathbf{f} \oplus T(\varphi))\big)\big) \\
&= (\mathbf{f} \oplus \varphi) \otimes \neg(\mathbf{t} \oplus \mathbf{f} \oplus T(\varphi)) && \text{— distributivity, De Morgan} \\
&= \mathbf{f} \oplus \varphi && \text{— by } \mathbf{t} \oplus \mathbf{f} = \top.
\end{aligned}
$$

Hence the following equational reasoning proves the law.

$$
\begin{aligned}
& \big(\mathbf{f} \oplus S\big((\mathbf{f} \oplus \varphi) \otimes \neg(\mathbf{f} \oplus T(\varphi))\big)\big) \otimes (\mathbf{t} \oplus \varphi) \\
&= (\mathbf{f} \oplus S(\varphi)) \otimes (\mathbf{t} \oplus \varphi) && \text{— by the above equation} \\
&= (\mathbf{f} \oplus S(\varphi)) \otimes (\mathbf{t} \oplus S(\varphi)) && \text{— } S \text{ is exception stable} \\
&= S(\varphi) && \text{— distributivity.}
\end{aligned}
$$

$$[\wedge\vdash]\frac{\Gamma,p,q\vdash\Delta}{\Gamma,p\wedge q\vdash\Delta}\qquad[\vdash\wedge]\frac{\Gamma\vdash p,\Delta\quad\Gamma\vdash q,\Delta}{\Gamma\vdash p\wedge q,\Delta}$$

$$[\neg\wedge\vdash]\frac{\Gamma,\neg p\vdash\Delta\quad\Gamma,\neg q\vdash\Delta}{\Gamma,\neg(p\wedge q)\vdash\Delta}\qquad[\vdash\neg\wedge]\frac{\Gamma\vdash\neg p,\neg q,\Delta}{\Gamma\vdash\neg(p\wedge q),\Delta}$$

$$[\vee\vdash]\frac{\Gamma,p\vdash\Delta\quad\Gamma,q\vdash\Delta}{\Gamma,p\vee q\vdash\Delta}\qquad[\vdash\vee]\frac{\Gamma\vdash p,q,\Delta}{\Gamma\vdash p\vee q,\Delta}$$

$$[\neg\vee\vdash]\frac{\Gamma,\neg p,\neg q\vdash\Delta}{\Gamma,\neg(p\vee q)\vdash\Delta}\qquad[\vdash\neg\vee]\frac{\Gamma\vdash\neg p,\Delta\quad\Gamma\vdash\neg q,\Delta}{\Gamma\vdash\neg(p\vee q),\Delta}$$

$$[\otimes\vdash]\frac{\Gamma,p,q\vdash\Delta}{\Gamma,p\otimes q\vdash\Delta}\qquad[\vdash\otimes]\frac{\Gamma\vdash p,\Delta\quad\Gamma\vdash q,\Delta}{\Gamma\vdash p\otimes q,\Delta}$$

$$[\neg\otimes\vdash]\frac{\Gamma,\neg p,\neg q\vdash\Delta}{\Gamma,\neg(p\otimes q)\vdash\Delta}\qquad[\vdash\neg\otimes]\frac{\Gamma\vdash\neg p,\Delta\quad\Gamma\vdash\neg q,\Delta}{\Gamma\vdash\neg(p\otimes q),\Delta}$$

$$[\oplus\vdash]\frac{\Gamma,p\vdash\Delta\quad\Gamma,q\vdash\Delta}{\Gamma,p\oplus q\vdash\Delta}\qquad[\vdash\oplus]\frac{\Gamma\vdash p,q,\Delta}{\Gamma\vdash p\oplus q,\Delta}$$

$$[\neg\oplus\vdash]\frac{\Gamma,\neg p\vdash\Delta\quad\Gamma,\neg q\vdash\Delta}{\Gamma,\neg(p\oplus q)\vdash\Delta}\qquad[\vdash\neg\oplus]\frac{\Gamma\vdash\neg p,\neg q,\Delta}{\Gamma\vdash\neg(p\oplus q),\Delta}$$

$$[\neg\neg\vdash]\frac{\Gamma,p\vdash\Delta}{\Gamma,\neg\neg p\vdash\Delta}\qquad[\vdash\neg\neg]\frac{\Gamma\vdash p,\Delta}{\Gamma\vdash\neg\neg p,\Delta}$$

$$[\supset\vdash]\frac{\Gamma\vdash p,\Delta\quad\Gamma,q\vdash\Delta}{\Gamma,p\supset q\vdash\Delta}\qquad[\vdash\supset]\frac{\Gamma,p\vdash q,\Delta}{\Gamma\vdash p\supset q,\Delta}$$

$$[\neg\supset\vdash]\frac{\Gamma,p,\neg q\vdash\Delta}{\Gamma,\neg(p\supset q)\vdash\Delta}\qquad[\vdash\neg\supset]\frac{\Gamma\vdash p,\Delta\quad\Gamma\vdash\neg q,\Delta}{\Gamma\vdash\neg(p\supset q),\Delta}$$

$$[\forall\vdash]\frac{\Gamma,p(e)\vdash\Delta}{\Gamma,\forall v.p(v)\vdash\Delta}\qquad[\vdash\forall]\frac{\Gamma\vdash p(u),\Delta}{\Gamma\vdash\forall v.p(v),\Delta}$$

$$[\neg\forall\vdash]\frac{\Gamma,\neg p(u)\vdash\Delta}{\Gamma,\neg\forall v.p(v)\vdash\Delta}\qquad[\vdash\neg\forall]\frac{\Gamma\vdash\neg p(e)}{\Gamma\vdash\neg\forall v.p(v),\Delta}$$

$$[\exists\vdash]\frac{\Gamma,p(u)\vdash\Delta}{\Gamma,\exists v.p(v)\vdash\Delta}\qquad[\vdash\exists]\frac{\Gamma\vdash p(e),\Delta}{\Gamma\vdash\exists v.p(v),\Delta}$$

$$[\neg\exists\vdash]\frac{\Gamma,\neg p(e)\vdash\Delta}{\Gamma,\neg\exists v.p(v)\vdash\Delta}\qquad[\vdash\neg\exists]\frac{\Gamma\vdash\neg p(u)}{\Gamma\vdash\neg\exists v.p(v),\Delta}$$

$$[\neg\mathbf{t}\vdash]\frac{}{\Gamma,\neg\mathbf{t}\vdash\Delta}\qquad[\vdash\mathbf{t}]\frac{}{\Gamma\vdash\mathbf{t},\Delta}$$

$$[\mathbf{f}\vdash]\frac{}{\Gamma,\mathbf{f}\vdash\Delta}\qquad[\vdash\neg\mathbf{f}]\frac{}{\Gamma\vdash\neg\mathbf{f},\Delta}$$

$$[\bot\vdash]\frac{}{\Gamma,\bot\vdash\Delta}\qquad[\vdash\top]\frac{}{\Gamma\vdash\top,\Delta}$$

$$[\neg\bot\vdash]\frac{}{\Gamma,\neg\bot\vdash\Delta}\qquad[\vdash\neg\top]\frac{}{\Gamma\vdash\neg\top,\Delta}$$

Fig. 3. The inference rules [AA96]

When quantifiers are involved in the formula, we need to resort to a formal proof. For example, the law (5.4) in Section 5.2

$$\{p\};X:=e\sqsubseteq X:=e;\{\exists v.(p[X\backslash v]\wedge X=e[X\backslash v])\}$$

is formally proved as below, where we write p' for $\neg(\exists v.(p[X\backslash v] \wedge X = e[X\backslash v] \supset \top)$ and indicate the use of equality axiom $\Gamma \vdash e = e, \Delta$ by [Equal].

1. $p, \neg\top \vdash p$	Initial sequent
2. $p, \neg\top \vdash e = e$	Axiom [Equal]
3. $p, \neg\top \vdash p[X\backslash X] \wedge e = e[X\backslash X]$	1, 2, rule $[\vdash \wedge]$
4. $p, \neg\top \vdash \exists v.(p[X\backslash v] \wedge e = e[X\backslash v])$	3, rule $[\vdash \exists]$
5. $p, \neg\top \vdash \neg\top$	Initial sequent
6. $p, \neg\top \vdash \neg(\exists v.(p[X\backslash v] \wedge e = e[X\backslash v]) \supset \top)$	4, 5, rule $[\vdash \neg \supset]$
7. $\neg(p \supset \top) \vdash \neg(\exists v.(p[X\backslash v] \wedge e = e[X\backslash v]) \supset \top)$	6, rule $[\neg \supset \vdash]$
8. $\neg(p \supset \top), \varphi[X\backslash e] \vdash \neg(\exists v.(p[X\backslash v] \wedge e = e[X\backslash v]) \supset \top)$	7, weakening
9. $\neg(p \supset \top), \varphi[X\backslash e] \vdash \varphi[X\backslash e]$	Initial sequent
10. $\neg(p \supset \top), \varphi[X\backslash e] \vdash \neg(\exists v.(p[X\backslash v] \wedge e = e[X\backslash v]) \supset \top) \otimes \varphi[X\backslash e]$	8, 9, rule $[\vdash \otimes]$
11. $\neg(p \supset \top), \mathbf{f} \vdash (p' \otimes \varphi)[X\backslash e]$	Axiom $[\mathbf{f} \vdash]$
12. $\neg(p \supset \top), \mathbf{f} \oplus \varphi[X\backslash e] \vdash (p' \otimes \varphi)[X\backslash e]$	10, 11, rule $[\oplus \vdash]$
13. $\neg(p \supset \top), \mathbf{f} \oplus \varphi[X\backslash e], \mathbf{t} \oplus \varphi \vdash \mathbf{f}, (p' \otimes \varphi)[X\backslash e]$	12, weakening
14. $\neg(p \supset \top), \mathbf{f} \oplus \varphi[X\backslash e], \mathbf{t} \oplus \varphi \vdash \mathbf{f} \oplus (p' \otimes \varphi)[X\backslash e]$	13, rule $[\vdash \oplus]$
15. $\neg(p \supset \top), \mathbf{f} \oplus \varphi[X\backslash e], \mathbf{t} \oplus \varphi \vdash \mathbf{t}, (p' \otimes \varphi)$	Axiom $[\vdash \mathbf{t}]$
16. $\neg(p \supset \top), \mathbf{f} \oplus \varphi[X\backslash e], \mathbf{t} \oplus \varphi \vdash \mathbf{t} \oplus (p' \otimes \varphi)$	15, rule $[\vdash \oplus]$
17. $\neg(p \supset \top), \mathbf{f} \oplus \varphi[X\backslash e], \mathbf{t} \oplus \varphi \vdash \big(\mathbf{f} \oplus (p' \otimes \varphi)[X\backslash e]\big) \otimes \big(\mathbf{t} \oplus (p' \otimes \varphi)\big)$	14, 16, rule $[\vdash \otimes]$
18. $\neg(p \supset \top) \otimes (\mathbf{f} \oplus \varphi[X\backslash e]) \otimes (\mathbf{t} \oplus \varphi) \vdash \big(\mathbf{f} \oplus (p' \otimes \varphi)[X\backslash e]\big) \otimes \big(\mathbf{t} \oplus (p' \otimes \varphi)\big)$	17, rule $[\otimes \vdash]$

B Supplementary Laws for the Development in Section 5.2

Lemma B.1.

$$\begin{aligned} & \{0 < X \leq N\}; \textbf{repeat } Y := X; X := (Y \times Y + N) \div (2 \times Y) \textbf{ until } X \geq Y \\ \sqsubseteq\ & \textbf{repeat } \{0 < X \leq N\}; Y := X; X := (Y \times Y + N) \div' (2 \times Y) \textbf{ until } X \geq Y \end{aligned}$$

Proof. Let us define $\mathcal{F}$ as in Section 5.2 and $\mathcal{F}'$ as follows.

$$\mathcal{F}'(T) \triangleq \{0 < X \leq N\}; Y := X; X := (Y \times Y + N) \div' (2 \times Y); \textbf{if } X \geq Y \textbf{ then skip else } T.$$

It is enough to show by transfinite induction that $\{0 < X \leq N\}; \mathcal{F}^{\alpha}(\textbf{abort}) \sqsubseteq \mu.\mathcal{F}'$ for any ordinal α.

In case α being a non-limit ordinal, suppose $\alpha = \beta + 1$ for some ordinal β.

$$\begin{aligned} & \{0 < X \leq N\}; \mathcal{F}^{\beta+1}(\textbf{abort}) \\ = & \{0 < X \leq N\}; Y := X; X := (Y \times Y + N) \div (2 \times Y); \textbf{if } X \geq Y \textbf{ then skip else } \mathcal{F}^{\beta}(\textbf{abort}) \\ \sqsubseteq & \{0 < X \leq N\}; Y := X; \{0 < Y \leq N \wedge Y = X\}; \langle \neg(Y = 0) \rangle; X := (Y \times Y + N) \div' (2 \times Y); \\ & \quad \textbf{if } X \geq Y \textbf{ then skip else } \mathcal{F}^{\beta}(\textbf{abort}) \qquad \text{— by (5.4) and (4.3)} \end{aligned}$$

$$
\begin{aligned}
&= \{0 < X \leq N\}; Y := X; \{0 < Y \leq N \wedge Y = X\}; X := (Y \times Y + N) \div' (2 \times Y); \\
&\qquad \textbf{if } X \geq Y \textbf{ then skip else } \mathcal{F}^{\beta}(\textbf{abort}) \quad \text{— by (4.5), since } 0 < Y \leq N \rightarrow \neg(Y = 0) \\
&\sqsubseteq \{0 < X \leq N\}; Y := X; X := (Y \times Y + N) \div' (2 \times Y); \\
&\qquad \textbf{if } X \geq Y \textbf{ then } \{0 < Y \leq N \wedge X = (Y \times Y + N) \div' (2 \times Y)\}; \textbf{skip} \\
&\qquad \textbf{else } \{0 < Y \leq N \wedge X = (Y \times Y + N) \div' (2 \times Y)\}; \mathcal{F}^{\beta}(\textbf{abort}) \\
&\qquad\qquad\qquad\qquad\qquad\qquad \text{— by (5.4), (5.7), and (4.3)} \\
&\sqsubseteq \{0 < X \leq N\}; Y := X; \{\neg(Y = 0)\}; X := (Y \times Y + N) \div' (2 \times Y); \\
&\qquad \textbf{if } X \geq Y \textbf{ then skip else } \{0 < X \leq N\}; \mathcal{F}^{\beta}(\textbf{abort}) \\
&\qquad\quad \text{— by (4.1) and (4.3) with an elementary number theoretical argument} \\
&\sqsubseteq \{0 < X \leq N\}; Y := X; X := (Y \times Y + N) \div' (2 \times Y); \textbf{if } X \geq Y \textbf{ then skip else } \mu.\mathcal{F}' \\
&\qquad\qquad\qquad\qquad\qquad\qquad \text{— by induction hypothesis} \\
&= \mu.\mathcal{F}' \qquad\qquad\qquad\qquad\qquad \text{— fixpoint.}
\end{aligned}
$$

If α is a limit ordinal, we have:

$$
\begin{aligned}
\{0 < X \leq N\}; \mathcal{F}^{\alpha}(\textbf{abort}) &= \{0 < X \leq N\}; \textstyle\bigoplus_{\beta<\alpha} \mathcal{F}^{\beta}(\textbf{abort}) \\
&= \textstyle\bigoplus_{\beta<\alpha} (\{0 < X \leq N\}; \mathcal{F}^{\beta}(\textbf{abort})) \quad \text{— by distributivity} \\
&\sqsubseteq \textstyle\bigoplus_{\beta<\alpha} \mu.\mathcal{F}' \qquad\qquad \text{— by induction hypothesis} \\
&= \mu.\mathcal{F}'.
\end{aligned}
$$

□

Lemma B.2. *Suppose $\mathcal{F} \in \mathsf{PTran} \rightarrow \mathsf{PTran}$ is a monotonic function. Then, $\mu.\mathcal{F}$ is non-exceptional, if $\mathcal{F}(S)$ is so for every non-exceptional S.*

Proof. We show $\mathcal{F}^{\alpha}(\textbf{abort})$ is non-exceptional for any ordinal α by transfinite induction. Suppose α is a non-limit ordinal such that $\alpha = \beta + 1$. By induction hypothesis, $\mathcal{F}^{\beta}(\textbf{abort})$ is non-exceptional. Therefore $\mathcal{F}^{\alpha}(\textbf{abort}) = \mathcal{F}(\mathcal{F}^{\beta}(\textbf{abort}))$ is non-exceptional. In case α is a limit ordinal, by induction hypothesis, $\mathcal{F}^{\beta}(\textbf{abort})$ is non-exceptional for any $\beta < \alpha$. This implies, for any φ and φ' such that $\mathbf{f} \oplus \varphi = \mathbf{f} \oplus \varphi'$, $\mathcal{F}^{\alpha}(\textbf{abort})(\varphi) = \bigoplus_{\beta<\alpha} \mathcal{F}^{\beta}(\textbf{abort})(\varphi) = \bigoplus_{\beta<\alpha} \mathcal{F}^{\beta}(\textbf{abort})(\varphi') = \mathcal{F}^{\alpha}(\textbf{abort})(\varphi')$. Therefore the limit $\mathcal{F}^{\alpha}(\textbf{abort})$ is also non-exceptional.