Unifying Theories of Undefinedness in UTP

Jim Woodcock and Victor Bandur

The University of York

Abstract. In previous work, based on an original idea due to Saaltink, we proposed a unifying theory of undefined expressions in logics used for formally specifying software systems. In our current paper, we instantiate these ideas in Hoare and He's Unifying Theories of Programming, with each different treatment of undefinedness formalized as a UTP theory. In this setting, we show how to use classical logic to prove facts in a monotonic partial logic with guards, and we describe the guards for several different UTP theories. We show how classical logic can be used to prove semi-classical facts. We apply these ideas to the COMPASS Modelling Language (*CML*), which is an integration of VDM and CSP in the *Circus* tradition. We link *CML*, which uses McCarthy's left-to-right expression evaluation, and to VDM, which uses Jones's three-valued Logic of Partial Functions.

1 Introduction

We consider the problem of potentially undefined expressions, which arise from two language constructs: partial function application and definite description.

A simple example of the problem is in the expression $(y = 1/0)$. Here, the division operator is a partial function that is not defined for a zero divisor: it is being applied outside its domain of definition. So what should we make of the expression $(1/0)$? Does it denote a value? If so, then which value? If not, then what do we make of the containing predicate $(y = 1/0)$? Is this defined? Does it denote a truth value or not?

More generally, if we choose a specific treatment of undefined expressions, then is it possible to use verification tools that employ different treatments? For example, there are two different treatments of undefined expressions for VDM: Jones's VDM uses the Logic of Partial Functions (LPF), which has been implemented in Isabelle [1], whilst Larsen's VDM in Overture uses left-to-right evaluation [8]. What is the relationship between these? Formal verification in the increasingly popular setting of heterogeneous systems of systems demands an answer to this question. In our own context of the COMPASS Modelling Language (*CML*), the nature of the language suggests the use of several verification tools, where the treatment of undefinedness must be taken into account. For example, in the FDR implementation of CSPM [5], undefinedness is handled by a combination of arithmetic overflow and boolean short-circuit expressions. In the *Circus* tools, undefinedness is handled through the use of classical logic and arbitrary undefined values. Furthermore, if *CML* is used for a system of systems

B. Wolff, M.-C. Gaudel, A. Feliachi (Eds.): UTP 2012, LNCS 7681, pp. 1–22, 2013.

with heterogeneous constituents using different formalisms with different solutions to the undefined problem, this combination of tools must be able to cope with these differences, for the sake of correctness as well as efficiency.

One possible solution to all these problems is to adopt a single treatment of undefinedness, such as the one used in UTP [7], where the basic relational calculus is classical: there is no undefinedness and every expression denotes a value. There is an outline of a more specific treatment of undefined expressions in UTP, but this is explored only briefly in the book by Hoare & He [7, Sect. 9.3]. There are several other possible treatments, and in this section we describe some of them. In doing so we develop a unifying theory for monotonic partial logics (we explain this term fully below).

This work is based on original ideas due to Mark Saaltink in his underpinnings for the Z/Eves theorem prover [10]. Together we have published joint papers at Marktoberdorf [11] and ICECCS 2007 [12].

In Sect. 2, we augment UTP's alphabetised relational calculus with a basic treatment of three-valued logic with possibly undefined expressions and predicates. In Sect. 3, we give a treatment of first-order theories for monotonic partial logics and prove a theorem about construct monotonicity (Theorem 1). In Sect. 4, we formalise three theories of undefinedness: strict logic, McCarthy's left-to-right logic, and Kleene's three-valued logic. In Sect. 5, we describe a theory of guard systems for generating verification conditions for the definedness of expressions and predicates. We present our main theorem that allows us to trade theorems between different logics by proving facts about the guard in a stronger system and guaranteeing that the construct is defined in a weaker logic (Theorem 2). We also present a guard system for the definite McCarthy logic and state its soundness (Theorem 3). Finally in Sect. 6, we draw some conclusions and plan future work.

2 Three-Valued Logic in UTP

In this section we illustrate our approach to undefinedness by describing a restricted semi-classical three-valued logic in UTP. The logic has a distinguished semantic value for undefined expressions and predicates. Operators of the predicate calculus are Bochvar's strict internal operators [4], but equality is classical (Bochvar's external $\equiv$ operator), allowing a fine control of undefinedness.

2.1 Basic Sets and Constructors

The set of boolean values is $\mathbb{B} = \{true, false\}$. The universe of values, disjoint from $\mathbb{B}$, is $\mathbb{U}$. We introduce a specific semantic undefined value: $\bot$. Any set not already equipped with an undefined value can be lifted to include it: $X^\bot = X \cup \{\bot\}$. Notice that $\bot$ is neither a tuple, nor a function, nor is it a designated value from $\mathbb{B}$ or $\mathbb{U}$.

For k, a natural number, X^k is the set of k-tuples over X, with X^0 having the single element: the 0-tuple $()$. X^* is the union of all X^ks.

As usual, we have two kinds of function space: $X \to Y$, the set of total functions, and $X \nrightarrow Y$, the set of partial functions.

We take inspiration from Rose's standard encoding of three-valued logic [9], which is reminiscent of Hoare & He's UTP designs [7, Chap. 3], in modelling three logical values using just a pair of predicates: (P, Q). The intuitive meaning is that P describes the region where the predicate (P, Q) is true and Q describes the region where (P, Q) is defined. Just like Hoare & He designs, we can combine the pair of predicates into a single predicate by introducing an observational variable, in this case def: the observation that the predicate is defined. This gives us a model for the pair.

Definition 1 (TVL predicate pair). *The observation def is true exactly when the pair is defined (Q) and, providing it is defined, then P determines whether it is true or not.*

$$(P, Q) \mathrel{\widehat{=}} (def \Rightarrow P) \wedge (Q = def)$$

□

The next example demonstrates that this definition accounts for all three logical values.

Example 1 (TVL extreme points). Consider the four extreme points for the pair:

$$\begin{array}{lll} R = true & = & (true, true) \quad = \quad def \\ R = false & = & (false, true) \quad = \quad false \\ R = \bot & = & \left\{ \begin{array}{l} (true, false) \\ (false, false) \end{array} \right\} = \neg\, def \end{array}$$

□

It is noteworthy that if def and Q do not agree then the entire TVL predicate is false, as expected.

Two lemmas follow immediately from Definition 1. The first shows how we can make use of the definedness condition in the pair.

Lemma 1 (Definedness trading). *The definedness condition can be traded back and forth in a TVL predicate pair:*

$$(P \wedge Q, Q) = (P, Q)$$

Proof

$$\begin{array}{l} (P \wedge Q, Q) \\ = (def \Rightarrow P \wedge Q) \wedge (Q = def) \\ = (def \Rightarrow P) \wedge (Q = def) \\ = (P, Q) \end{array}$$

□

The second lemma shows that every three-valued predicate can be expressed as a TVL pair.

Lemma 2 (Canonical form of TVL predicates). *Every three-valued predicate has a canonical form:*

$$R = (R^t, \neg\, R^f), \quad \textit{where } R^b = R[b/def], \textit{ and } t \textit{ and } f \textit{ abbreviate true and false, respectively.}$$

Proof

$$\begin{aligned}
&((P, Q)^t, \neg\, (P, Q)^f) \\
&= (((def \Rightarrow P) \wedge (Q = def))^t, \neg\, ((def \Rightarrow P) \wedge (Q = def))^f) \\
&= ((true \Rightarrow P) \wedge (Q = true), \neg\, ((false \Rightarrow P) \wedge (Q = false))) \\
&= (P \wedge Q, \neg\, (true \wedge \neg\, Q)) \\
&= (P \wedge Q, Q)
\end{aligned}$$

□

Example 2 (Definedness of partial expressions). Consider the predicate $(z = x/y)$ interpreted as a three-valued predicate. It is defined exactly when $(y \neq 0)$, and when it is defined, it is true when $(x = y * z)$, where $(\_ * \_)$ is the total multiplication operator. So the three-valued predicate $(z = x/y)$ is modelled by the pair:

$$((x = y * z), (y \neq 0))$$

We can consider three examples with specific values for x, y, and z.

$$\begin{aligned}
&(3 = 6/2) \\
&= ((6 = 2 * 3), (2 \neq 0)) \\
&= (true, true) \\
&= def
\end{aligned}
\qquad
\begin{aligned}
&(2 = 6/2) \\
&= ((6 = 2 * 2), (2 \neq 0)) \\
&= (false, true) \\
&= false
\end{aligned}
\qquad
\begin{aligned}
&(2 = 6/0) \\
&= ((6 = 0 * 2), (0 \neq 0)) \\
&= (false, false) \\
&= \neg\, def
\end{aligned}$$

□

The model that we have chosen for three-valued predicates is not closed under any of the propositional operators, so we must choose particular definitions for them. There are plenty of choices: for two operands of three values, there are nine possible results, each of three values, making a total of: $3^9 \simeq 20,000$ combinations, although as Bergstra *et al.* point out, only a very small number of these are desirable [3]. In this section we choose strict interpretations of each operator.

2.2 Conjunction

The strict conjunction of two three-valued predicates is defined as follows.

Definition 2 (TVL conjunction). *$T \wedge_3 U$ is defined exactly when both T and U are defined; it is true exactly when both T and U are true.*

$$(P, Q) \wedge_3 (R, S) \mathrel{\widehat{=}} (P \wedge R, Q \wedge S)$$

It is useful to see the truth table for conjunction:

$\wedge_3$	def	$\neg\ def$	$false$
def	def	$\neg\ def$	$false$
$\neg\ def$	$\neg\ def$	$\neg\ def$	$\neg\ def$
$false$	$false$	$\neg\ def$	$false$

This truth table looks a little better if we replace the values in the model by the three truth values themselves:

$\wedge_3$	$true_3$	$\bot$	$false_3$
$true_3$	$true_3$	$\bot$	$false_3$
$\bot$	$\bot$	$\bot$	$\bot$
$false_3$	$false_3$	$\bot$	$false_3$

□

An example of the conjunction of the two three-valued predicates $(y = 3)$ and $(z = x/y)$ helps clarify the separation between a predicate's truth and definedness conditions.

Example 3 (Partial conjunction).

$$\begin{aligned}
&(y = 3) \wedge_3 (z = x/y) \\
&= ((y = 3), true) \wedge_3 ((x = y * z), (y \neq 0)) \\
&= ((y = 3) \wedge (x = y * z), true \wedge (y \neq 0)) \\
&= ((y = 3) \wedge (x = 3 * z), (y \neq 0))
\end{aligned}$$

□

2.3 Negation

The strict negation of a three-valued predicate is defined as follows.

Definition 3 (TVL negation). *The negation of a three-valued predicate R is defined exactly when R is defined, and is true exactly when R is false:*

$$\neg_3 (P, Q) = (\neg\ P, Q)$$

The truth table is:

$\neg_3$	
def	$false$
$\neg\ def$	$\neg\ def$
$false$	def

$\neg_3$	
$true_3$	$false_3$
$\bot$	$\bot$
$false_3$	$true_3$

□

An example illustrates how the negation of a three-valued predicate can be simply pushed into the underlying representation.

Example 4 (Partial negation).

$$\begin{aligned}
&\neg_3\,(z = x/y)\\
&= \neg_3\,((x = y * z), (y \neq 0))\\
&= ((x \neq y * z), (y \neq 0))
\end{aligned}$$

□

2.4 Disjunction

The strict disjunction of two three-valued predicates is defined as follows.

Definition 4 (TVL disjunction). *The disjunction of two three-valued predicates $T \vee_3 U$ is defined exactly when both T and U are defined; it is true when either of them is true.*

$$(P, Q) \vee_3 (R, S) \;\widehat{=}\; (P \vee R, Q \wedge S)$$

The truth tables are:

$\vee_3$	def	$\neg\ def$	$false$
def	def	$\neg\ def$	def
$\neg\ def$	$\neg\ def$	$\neg\ def$	$\neg\ def$
$false$	def	$\neg\ def$	$false$

$\vee_3$	$true_3$	$\bot$	$false_3$
$true_3$	$true_3$	$\bot$	$true_3$
$\bot$	$\bot$	$\bot$	$\bot$
$false_3$	$true_3$	$\bot$	$false_3$

□

Example 5 (Partial disjunction). Define $P \Rightarrow_3 Q$ as $\neg_3\,(P \vee_3 Q)$. Now suppose that f is a partial function symbol, such that

$$(y = f(x)) = ((y = f(x)), x \in \mathrm{dom}\, f)$$

Now consider the predicate $(x \in \mathrm{dom}\, f \Rightarrow_3 (y = f(x)))$, which is reminiscent of a precondition guarding the application of the partial function f. When interpreted as a three-valued predicate we have,

$$\begin{aligned}
&x \in \mathrm{dom}\, f \Rightarrow_3 (y = f(x))\\
&= \neg_3\,(x \in \mathrm{dom}\, f) \vee_3 (y = f(x))\\
&= \neg_3\,(x \in \mathrm{dom}\, f, \mathit{true}) \vee_3 (y = f(x))\\
&= (x \notin \mathrm{dom}\, f, \mathit{true}) \vee_3 (y = f(x))\\
&= (x \notin \mathrm{dom}\, f, \mathit{true}) \vee_3 ((y = f(x)), x \in \mathrm{dom}\, f)\\
&= (x \notin \mathrm{dom}\, f \vee (y = f(x)), \mathit{true} \wedge x \in \mathrm{dom}\, f)\\
&= (x \in \mathrm{dom}\, f \Rightarrow (y = f(x)), x \in \mathrm{dom}\, f)\\
&= ((y = f(x)), x \in \mathrm{dom}\, f)
\end{aligned}$$

It is defined exactly when $(x \in \mathrm{dom}\, f)$, and when it is defined, it is true exactly when $(y = f(x))$. □

2.5 Equality

There is nothing special about equality in our treatment of undefined values: it is just the existing classical equality in UTP. So, two three-valued predicates are equal exactly when their representation as pairs are equal. This is the symmetric closure of the following rules:

$$\begin{array}{rr} (def =_3 \neg\, def) = false & (true_3 =_3 \bot) = false \\ (def =_3 false) = false & (true_3 =_3 false_3) = false \\ (\neg\, def =_3 false) = false & (\bot =_3 false_3) = false \end{array}$$

Equality over the lifted domain $\mathbb{U}^\bot$ behaves similarly.

Example 6 (Partial equality). One of the definitions that we use later is a conditional containing five equations between three-valued predicates and expressions:

$$(f(x,y) =_3 \bot) \lhd (x = \bot) \vee (y = \bot) \rhd (f(x,y) =_3 (x = y))$$

Each equation is by definition either true or false: it cannot be undefined. In this way, UTP equality contains the use of three-valued logic. We also restrict our use of quantifiers to avoid undefinedness. □

A very simple lemma is a consequence of these definitions.

Lemma 3 (Normality of TVL operators). *When they are defined, the TVL propositional operators behave exactly like their classical counterparts (sometimes called "normality" [2]).*

1. $Q \Rightarrow (\neg_3\, (P, Q) = \neg\, P)$
2. $Q \wedge S \Rightarrow ((P, Q) \wedge_3 (R, S) = P \wedge R)$
3. $Q \wedge S \Rightarrow ((P, Q) \vee_3 (R, S) = P \vee R)$

□

This justifies UTP with three-valued logic, and allows the definition of theories in which definedness is elegantly available as a predicate, rather than appealing to obtrusive comparison with an explicitly designated undefined value. What is more, we will not introduce definite description or partial functions as fundamental concepts, so that it remains impossible to manufacture undefined values at the level of the logical calculus of the UTP. But we can build logics that do have these features, as we show later.

3 First-Order Theories

With the groundwork laid for a three-valued logical landscape in UTP, in this section we develop theories for various types of three-valued logics encountered in the literature and in the field. These theories depart from the classical world of UTP by making use of the lifted domain $\mathbb{U}^\bot$ and the lifted set of boolean values $\mathbb{B}^\bot$, soundly admitting the undefined value $\bot$ through the approach presented above.

3.1 Contexts for First-Order Theories

We introduce a context theory ***CXT*** for our first-order theories, which will all be subtheories of it. The alphabet of ***CXT*** contains two observational variables:

$$\begin{array}{rcl} PShape & : & \mathbb{P}((\mathbb{U}^\perp)^* \pfun \mathbb{B}^\perp) \\ FShape & : & \mathbb{P}((\mathbb{U}^\perp)^* \pfun \mathbb{U}^\perp) \end{array}$$

and its signature is:

$$\begin{array}{rcl} =_3 & : & \mathbb{U}^\perp \times \mathbb{U}^\perp \to \mathbb{B}^\perp \\ \neg_3 & : & \mathbb{B}^\perp \to \mathbb{B}^\perp \\ \vee_3 & : & \mathbb{B}^\perp \times \mathbb{B}^\perp \to \mathbb{B}^\perp \\ \forall_3 & : & (\mathbb{U} \pfun \mathbb{B}^\perp) \to \mathbb{B}^\perp \\ \iota_3 & : & (\mathbb{U} \pfun \mathbb{B}^\perp) \to \mathbb{U}^\perp \end{array}$$

PShape describes all the possible denotations for the predicate symbols of this theory. Every denotation is a partial function from some number of parameters, each of which could be drawn from $\mathbb{U}$ or could be undefined, to a boolean result, which could also be undefined. The purpose of *PShape* is to constrain all the theory's predicate symbols in a uniform way. *FShape* does the same job as *PShape*, except that it describes the possible denotations of function symbols. The operators $=_3$, $\neg_3$, and $\vee_3$ give the syntax for equality, negation, and disjunction, respectively. Since they operate over lifted domains, they allow the construction of TVL predicates.

The $\forall_3$ function takes as its argument a function $\mathbb{U} \pfun \mathbb{B}^\perp$ that describes a binding for the universal quantifier that characterises the predicate that must be universally true. The function considers each element of its domain in turn and assigns to it one of the three logical values. The $\forall_3$ function takes this binding function and decides whether the universally quantified predicate is true, false, or undefined. Notice that the binding function ranges only over defined values. This means that we are excluding logics where bound variables may be undefined, as is the case in LCF [6].

The ι_3 function also takes a binding function as its argument. It decides whether this binding is a definite description of a value in $\mathbb{U}$ or is undefined. Once more, the bound variable must be everywhere defined.

We add a single healthiness condition to constrain the definite description function:

$$\mathbf{CXT}(P) = P \land (\forall f : \mathbb{U} \pfun \mathbb{B}^\perp \bullet f \neq \emptyset \Rightarrow \iota_3(f) \in \operatorname{dom} f^\perp)$$

This requires that the definite description of a non-empty binding function returns either an undefined value or an element from the domain of the binding. We require this result in Lemma 6, where we prove that theories are closed under constructs over their signature.

Example 7 (Context). Consider a context with no predicate symbols and only monadic and dyadic function symbols.

$$\mathbf{X1}(P) = P \land (PShape = \emptyset) \land (FShape = (\mathbb{U}^\perp \cup (\mathbb{U}^\perp)^2 \pfun \mathbb{U}^\perp))$$

PShape and *FShape* are used to add type information: we use them to restrict how predicate and function symbols behave, particularly, as we shall see later, with respect to undefinedness. □

3.2 First-Order Theories

A first-order theory (FOT) is an enrichment of a particular context and acts as its model. We add to the context six more alphabetical variables and three healthiness conditions. The set of names A is partitioned into three sets: variables, predicate symbols, and function symbols.

$$\langle Var, Pred, Fun \rangle \text{ partition } A$$

The set $Dom : \mathbb{P}\,\mathbb{U}$ describes the domain of values for the first-order theory. Finally, the rank function $\rho : Pred \cup Fun \to \mathbb{N}$ describes the number of parameters that each predicate and function symbol can take.

The first healthiness condition requires that every variable is defined and has a value drawn from *Dom*:

$$\mathbf{DV}(P) = P \land (\forall\, v : Var \bullet v \in Dom)$$

The second and third healthiness conditions require that every predicate and function symbol ranges over arguments taken from $Dom^\perp$ and produces results in $\mathbb{B}^\perp$ and $\mathbb{U}^\perp$, respectively:

$$\mathbf{DP}(P) = P \land (\forall\, p : Pred \bullet p \in ((Dom^\perp)^{\rho(p)} \to \mathbb{B}^\perp) \cap PShape)$$
$$\mathbf{DF}(P) = P \land (\forall\, f : Fun \bullet f \in ((Dom^\perp)^{\rho(f)} \to \mathbb{U}^\perp) \cap FShape)$$

Example 8 (First-order theory). Consider a theory **T1** with context **X1** that has just a single function symbol for integer division:

$$\begin{aligned}
&\mathbf{T1}(P) = \\
&\quad \mathbf{X1}(P) \\
&\quad \land\ Var = \emptyset \\
&\quad \land\ Pred = \emptyset \\
&\quad \land\ Fun = \{_/_\} \\
&\quad \land\ Dom = \mathbb{N} \\
&\quad \land\ \rho = \{_/_ \mapsto 2\} \\
&\quad \land\ _/_ \in (\mathbb{N}^\perp \times \mathbb{N}^\perp \to \mathbb{N}^\perp) \cap FShape
\end{aligned}$$

□

3.3 Information-Theoretic Ordering

Our whole approach to unifying the treatment of undefinedness in different logics is built on a rather flat information-theoretic ordering. The goal is to allow for the comparison of logics that are more or less discriminant in the presence of undefinedness. This means that the undefined value is considered to be worse

than every other value; these other values are themselves incomparable with each other in this sense. The notion is captured below.

Definition 5 (Information-theoretic ordering). *The information-theoretic ordering $\sqsubseteq$ is defined as follows.*
Elements: for any set X with $a, b \in X$

$$a \sqsubseteq b \;\widehat{=}\; (a \neq \bot) \Rightarrow (a = b)$$

Pointwise extension to tuples: for $x, y \in X^k$

$$x \sqsubseteq y \;\widehat{=}\; \forall\, i : 1 \,..\, k \bullet x_i \sqsubseteq y_i$$

Pointwise extension to functions: for $f, g \in X \to Y$

$$f \sqsubseteq g \;\widehat{=}\; (\operatorname{dom} f = \operatorname{dom} g) \wedge (\forall\, x : \operatorname{dom} f \bullet f(x) \sqsubseteq g(x))$$

Comparing sets of functions: for $A, B : \mathbb{P}\, X$, the Hoare preorder is defined:

$$A \sqsubseteq_H B \;\widehat{=}\; \forall\, a : A \bullet \exists\, b : B \bullet a \sqsubseteq b$$

□

These definitions are illustrated in the following set of examples.

Example 9 (Ordering).

1. On elements:

$$\begin{aligned} \bot &\sqsubseteq 1 \\ 1 &\sqsubseteq 1 \\ \neg\, (1 &\sqsubseteq 2) \end{aligned}$$

2. On tuples:

$$\begin{aligned} (0, \bot, 2) &\sqsubseteq (0, 1, 2) \\ () &\sqsubseteq () \\ (1, 2) &\sqsubseteq (1, 2) \\ \neg\, ((1, 2) &\sqsubseteq (2, 2)) \end{aligned}$$

3. On functions:

$$(\lambda\, x, y : \mathbb{N} \bullet \bot \lhd (y = 0) \rhd x/y) \sqsubseteq (\lambda\, x, y : \mathbb{N} \bullet 0 \lhd (y = 0) \rhd x/y)$$
$$(\lambda\, n : \mathbb{N} \bullet \bot \lhd (n \bmod 2 = 0) \rhd n) \sqsubseteq (\lambda\, n : \mathbb{N} \bullet n)$$

4. On sets of functions:

$$\begin{aligned} &\{(\lambda\, x, y : \mathbb{N} \bullet \bot \lhd (y = 0) \rhd x/y), \\ &\;(\lambda\, n : \mathbb{N} \bullet \bot \lhd (n \bmod 2 = 0) \rhd n), \\ &\;(\lambda\, n : \mathbb{N} \bullet n)\} \\ \sqsubseteq_H \\ &\{(\lambda\, x, y : \mathbb{N} \bullet 0 \lhd (y = 0) \rhd x/y), \\ &\;(\lambda\, n : \mathbb{N} \bullet n)\} \end{aligned}$$

□

We further generalise the ordering by lifting it to contexts.

Definition 6 (Ordering on contexts).

$$\boldsymbol{S} \sqsubseteq_H \boldsymbol{T} = \forall\, P : \boldsymbol{S};\ Q : \boldsymbol{T} \bullet P \sqsubseteq_H Q$$

where

$$P \sqsubseteq_H Q =$$
$$PShape_{\boldsymbol{S}} \sqsubseteq_H PShape_{\boldsymbol{T}}$$
$$\land\ FShape_{\boldsymbol{S}} \sqsubseteq_H FShape_{\boldsymbol{T}}$$
$$\land\ (=_{\boldsymbol{S}}) \sqsubseteq (=_{\boldsymbol{T}})$$
$$\land\ (\neg_{\boldsymbol{S}}) \sqsubseteq (\neg_{\boldsymbol{T}})$$
$$\land\ (\vee_{\boldsymbol{S}}) \sqsubseteq (\vee_{\boldsymbol{T}})$$
$$\land\ (\forall_{\boldsymbol{S}}) \sqsubseteq (\forall_{\boldsymbol{T}})$$
$$\land\ (\iota_{\boldsymbol{S}}) \sqsubseteq (\iota_{\boldsymbol{T}})$$

□

Intuitively it can be seen that all functions and predicates admissible in $\boldsymbol{S}$ are less discriminant of the undefined value than those admissible in $\boldsymbol{T}$. We say that undefinedness is more *contagious* in $\boldsymbol{S}$.

Example 10 (Subtheory). Consider $\boldsymbol{X2}$, a subtheory of $\boldsymbol{X1}$, where the following holds:

$$\forall\, f : FShape_{\boldsymbol{X1}} \bullet zero \circ f \in FShape_{\boldsymbol{X2}}$$

and where the total function *zero* is defined:

$$zero(x) \ \widehat{=}\ (0 \lhd (x = \bot) \rhd x)$$

All other components remain unchanged. Then $P_{\boldsymbol{X1}} \sqsubseteq_H P_{\boldsymbol{X2}}$, since

$$f \sqsubseteq zero \circ f$$
$$= (\mathrm{dom}\, f = \mathrm{dom}(zero \circ f)) \land \forall\, x : \mathrm{dom}\, f \bullet f(x) \sqsubseteq zero \circ f(x)$$

and so we have $FShape_{X1} \sqsubseteq_H FShape_{X2}$.

□

In the following sections, we introduce the three important notions of strictness, definiteness, and monotonicity.

3.4 Strictness

The notion of strictness is a familiar one from the definition of programming languages. A function f is strict if $f(\bot) = \bot$, and it is usually used to denote that a function loops forever or performs an illegal operation, such as division by zero. Generally no distinction is made if the function in fact delivers a useable result before this happens. We can interpret a strict function operationally as one that always evaluates all of its arguments. A restricted notion considers functions that are strict in one or more arguments.

Definition 7 (Strict). *Function* $f : (X^\bot)^{\rho(f)} \to Y^\bot$ *is strict if, whenever at least one of its arguments is undefined, then the result is undefined:*

$$\textbf{\textit{strict}}(f) = \forall\, x : (X^\bot)^{\rho(f)} \bullet (\exists\, i : 1 \,..\, \rho(f) \bullet (x_i = \bot)) \Rightarrow (f(x) = \bot)$$

□

Example 11 (Strict function). Suppose that $\_ * \_$ is the standard multiplication operator on natural numbers: $\_ * \_ : \mathbb{N} \times \mathbb{N} \to \mathbb{N}$. We can define a strict version of the operator:

$$\begin{array}{l} \_ *_3 \_ : \mathbb{N}^\bot \times \mathbb{N}^\bot \to \mathbb{N}^\bot \\ x *_3 y = \bot \lhd (x = \bot) \lor (y = \bot) \rhd x * y \end{array}$$

□

We can extend the notion of strictness to a context, where every predicate has only strict denotations for its predicate and function symbols. We find it useful to define a healthiness function ***strict***() that is applied to a context (which of course is a set of predicates).

Definition 8 (Strict contexts). *We make a context* $\boldsymbol{T}$ *strict:*

$$\begin{array}{l} \textbf{\textit{strict}}(\boldsymbol{T}) = \{\, P : \boldsymbol{T} \bullet \textbf{\textit{strict}}(P) \,\} \\ \textbf{\textit{where strict}}(P) = \exists\, PShape_0, FShape_0 \bullet \\ \qquad\qquad PShape = \{\, p : PShape_0 \mid \textbf{\textit{strict}}(p) \,\} \\ \qquad\qquad \land\ FShape = \{\, f : FShape_0 \mid \textbf{\textit{strict}}(f) \,\} \\ \qquad\qquad \land\ P[PShape_0, FShape_0 / PShape, FShape] \end{array}$$

□

3.5 Definiteness

Definiteness is, in a sense, a dual notion to strictness. If a function is definite, then it cannot manufacture undefinedness. That is, if the function produces an undefined result, then it must have had an undefined argument.

Definition 9 (Definite). *Function* $f : (X^\bot)^{\rho(f)} \to Y^\bot$ *is definite:*

$$\textbf{\textit{definite}}(f) = \forall\, x : (X^\bot)^{\rho(f)} \bullet (f(x) = \bot) \Rightarrow (\exists\, i : 1 \,..\, \rho(f) \bullet (x_i = \bot))$$

□

Example 12 (Definite function). The function $\_ *_3 \_$ above is definite. □

As for strictness, we define a healthiness function for contexts.

Definition 10 (Definite contexts). *Making a context definite:*

$$\begin{array}{l} \textbf{\textit{definite}}(\boldsymbol{T}) = \{\, P : \boldsymbol{T} \bullet \textbf{\textit{definite}}(P) \,\} \\ \textbf{\textit{where definite}}(P) = \\ \qquad \exists\, PShape_0, FShape_0 \bullet \\ \qquad\qquad PShape = \{\, p : PShape_0 \mid \textbf{\textit{definite}}(p) \,\} \\ \qquad\qquad \land\ FShape = \{\, f : FShape_0 \mid \textbf{\textit{definite}}(f) \,\} \\ \qquad\qquad \land\ P[PShape_0, FShape_0 / PShape, FShape] \end{array}$$

□

3.6 Monotonicity

A monotonic function on ordered sets is one that preserves that order. In our unifying theory, we are interested in defined-monotonic functions, that is, ones that preserve the definedness ordering.

Definition 11 (Monotonic). *Function* $f : (X^\perp)^{\rho(f)} \to Y^\perp$ *is monotonic:*

$$\mathbf{monotonic}(f) = \forall\, x_1, x_2 : (X^\perp)^{\rho(f)} \bullet x_1 \sqsubseteq x_2 \Rightarrow f(x_1) \sqsubseteq f(x_2)$$

□

Example 13 (Monotonic operator). The TVL negation operator $\neg_3$ from Sect. 3 is monotonic:

$\neg_3$	
$true_3$	$false_3$
$\perp$	$\perp$
$false_3$	$true_3$

□

Here also it is convenient to define a predicate that is true if a context is monotonic.

Definition 12 (Monotonic contexts). $\boldsymbol{T}$ *is a monotonic context:*

$$\begin{aligned}&\mathbf{monotonic}(\boldsymbol{T}) = \forall\, P : \boldsymbol{T} \bullet \mathbf{monotonic}(P)\\ &\mathbf{where}\ \mathbf{monotonic}(P) =\\ &\qquad (\forall\, p : Pred_{\boldsymbol{T}} \bullet \mathbf{monotonic}(p))\\ &\qquad \wedge\ (\forall\, f : Fun_{\boldsymbol{T}} \bullet \mathbf{monotonic}(f))\\ &\qquad \wedge\ \mathbf{monotonic}(=_{\boldsymbol{T}})\\ &\qquad \wedge\ \mathbf{monotonic}(\neg_{\boldsymbol{T}})\\ &\qquad \wedge\ \mathbf{monotonic}(\vee_{\boldsymbol{T}})\\ &\qquad \wedge\ \mathbf{monotonic}(\forall_{\boldsymbol{T}})\\ &\qquad \wedge\ \mathbf{monotonic}(\iota_{\boldsymbol{T}})\end{aligned}$$

□

The following simple lemma is useful.

Lemma 4 (Strict monotonic). *Every strict function is monotonic.* □

3.7 Comparing First-Order Theories

In Definition 6, we lifted our information-theoretic ordering up to contexts; now we lift it to first-order theories. This makes sense only if the two FOTs in question have the same domain of values.

Definition 13 (Comparing FOTs). *Comparing FOTs* $\boldsymbol{U}$ *and* $\boldsymbol{V}$*: for* $P : \boldsymbol{U}$ *and* $Q : \boldsymbol{V}$

$$P \sqsubseteq_H Q = Dom_{\boldsymbol{U}} = Dom_{\boldsymbol{V}} \wedge Pred_{\boldsymbol{U}} \sqsubseteq_H Pred_{\boldsymbol{V}} \wedge Fun_{\boldsymbol{U}} \sqsubseteq_H Fun_{\boldsymbol{V}}$$

□

Using this definition, we can state an important lemma. If $\boldsymbol{S}$ and $\boldsymbol{T}$ are two contexts, such that $\boldsymbol{S}$ is less defined than (or equal to) $\boldsymbol{T}$, and we have a FOT that models $\boldsymbol{S}$, then there will also be a FOT that models $\boldsymbol{T}$.

Lemma 5 (Models). *Suppose that we have two CXTs* $\boldsymbol{S}$ *and* $\boldsymbol{T}$*, where* $\boldsymbol{S} \sqsubseteq_H \boldsymbol{T}$*. Suppose further that* $\boldsymbol{U}$ *is a FOT extending* $\boldsymbol{S}$*. Then there is a FOT* $\boldsymbol{V}$ *extending* $\boldsymbol{T}$ *such that* $\boldsymbol{U} \sqsubseteq \boldsymbol{V}$*.* □

The proof of this lemma is quite straightforward. The relationship between $\boldsymbol{S}$ and $\boldsymbol{T}$ shows where undefined values in the former have been replaced by defined values in the latter. This is used as a guide to construct an appropriate model.

Example 14 (Application of Models lemma). Suppose that we have two contexts $\boldsymbol{S}$ and $\boldsymbol{T}$. Suppose further that $\boldsymbol{S}$ has only a single monadic function symbol $inc : \mathbb{U}^\perp \nrightarrow \mathbb{U}^\perp$. Define a simple model $\boldsymbol{U}$ for $\boldsymbol{S}$ that instantiates inc as a rather trivial increment operation on binary digits. This operation is easy to define on the argument 0, it returns the result 1. It is undefined otherwise. The context $\boldsymbol{T}$, on the other hand produces only defined results $inc : \mathbb{U}^\perp \nrightarrow \mathbb{U}$. There must be a model $\boldsymbol{V}$ for $\boldsymbol{T}$, such that $\boldsymbol{U} \sqsubseteq \boldsymbol{V}$. This is easy to construct. The domain of values has to be the same as for $\boldsymbol{U}$. The inc can return an arbitrary value for any argument that returns $\perp$. Note that this makes it non-strict: it must produce a defined value for the argument $\perp$. All this is summarised in the following table:

	$\boldsymbol{S}$	$\boldsymbol{T}$
$PShape$	$\emptyset$	$\emptyset$
$FShape$	$\textbf{strict}(\mathbb{U}^\perp \nrightarrow \mathbb{U}^\perp)$	$\mathbb{U}^\perp \nrightarrow \mathbb{U}$
	$\boldsymbol{U}$	$\boldsymbol{V}$
Dom	$\{0, 1\}$	$\{0, 1\}$
ρ	$\{inc \mapsto 1\}$	$\{inc \mapsto 1\}$
A	$inc(\perp) = \perp$	$inc(\perp) = 0$
	$inc(0) = 1$	$inc(0) = 1$
	$inc(1) = \perp$	$inc(1) = 1$

□

We state another important lemma about the closure of a FOT under the syntax of expressions.

Lemma 6 (Expression consistency). *Suppose that* e *is an expression over a FOT* $\boldsymbol{U}$*, then every* $\boldsymbol{U}$*-healthy predicate* P *ensures:*

$$P \Rightarrow e \in Dom_{\boldsymbol{U}}^\perp$$

□

This lemma is proved by syntactic induction.

A third important result is the following theorem that states that constructs (expressions or predicates) are monotonic.

Theorem 1 (Construct monotonicity). *Suppose* $\mathbf{S} \sqsubseteq_H \mathbf{T}$*, that* $\mathbf{U}$ *extends* $\mathbf{S}$*,* $\mathbf{V}$ *extends* $\mathbf{T}$*, and that either* $\mathbf{S}$ *or* $\mathbf{T}$ *is monotonic. Then, for any construct* c*, we have*

$$c_{\mathbf{U}} \sqsubseteq c_{\mathbf{V}}$$

Proof (Construct monotonicity). The proof of the theorem is by induction on the syntax of the construct c*. To illustrate the proof, we consider only the second induction case: application of a function symbol to actual parameters. This is enough to demonstrate the role of monotonicity in one of the two contexts.*

The induction hypothesis is that $x_{\mathbf{S}} \sqsubseteq x_{\mathbf{T}}$*.*

Case 2.1: $\mathbf{S}$ *is monotonic*

$$\begin{array}{ll}
(f(x))_{\mathbf{U}} & \{\ \textit{interpretation}\ \} \\
= f_{\mathbf{U}}(x_{\mathbf{U}}) & \{\ \textit{hypothesis } x_{\mathbf{U}} \sqsubseteq x_{\mathbf{V}} + \mathbf{S} \textit{ monotonic, and so } f_{\mathbf{U}} \textit{ is monotonic}\ \} \\
\sqsubseteq f_{\mathbf{U}}(x_{\mathbf{V}}) & \{\ \textit{assumption: } P_{\mathbf{U}} \sqsubseteq Q_{\mathbf{V}}\textit{, and so } Fun_{\mathbf{U}} \sqsubseteq Fun_{\mathbf{V}} \textit{ and so } f_{\mathbf{U}} \sqsubseteq f_{\mathbf{V}}\ \} \\
\sqsubseteq f_{\mathbf{V}}(x_{\mathbf{V}}) & \{\ \textit{interpretation}\ \} \\
= (f(x))_{\mathbf{V}} &
\end{array}$$

Case 2.2: $\mathbf{T}$ *is monotonic*

$$\begin{array}{ll}
(f(x))_{\mathbf{U}} & \{\ \textit{interpretation}\ \} \\
= f_{\mathbf{U}}(x_{\mathbf{U}}) & \{\ \textit{assumption: } P_{\mathbf{S}} \sqsubseteq P_{\mathbf{T}}\ \} \\
\sqsubseteq f_{\mathbf{V}}(x_{\mathbf{U}}) & \{\ \textit{hypothesis} + \mathbf{V} \textit{ monotonic}\ \} \\
\sqsubseteq f_{\mathbf{V}}(x_{\mathbf{V}}) & \{\ \textit{interpretation}\ \} \\
= (f(x))_{\mathbf{V}} &
\end{array}$$

□

4 Specific First-Order Theories

In this section we consider three different theories of logic with undefinedness: strict logic, McCarthy's logic and Kleene's logic. In our definitions, we demonstrate the differences between these three; in our theorems, we demonstrate the similarities.

4.1 Strict Logic

Strict logic treats undefinedness as extremely contagious: whenever an undefined value appears in an expression or predicate, the overall construct collapses to become undefined. As we saw in Definition 7, this is strictness. First of all, every predicate in this theory is strict (see Definition 8). This means that *PShape* and *FShape* both contain only strict denotations.

$$\mathbf{S1}(P) = \mathbf{strict}(P)$$

Next, equality is strict:

$$(=_s(x, y) =_3 \bot) \lhd (x = \bot) \lor (y = \bot) \rhd (=_s(x, y) =_3 (x = y))$$

Recall Example 6 for an explanation of the definedness of this definition. If either argument is undefined, then the equality is undefined: otherwise, strict equality depends on the underlying UTP equality.

Definite description is strict:

$$(\iota_s(f) = x) \lhd \bot \notin \operatorname{ran} f \land (\operatorname{dom}(f \rhd \{true\}) = \{x\}) \rhd (\iota_s(f) = \bot)$$

The argument to ι_s is a function f that binds elements of its domain to one of three truth values. If this binding is everywhere defined and there is only one element of f's domain that satisfies f's characteristic predicate, then the definite description is exactly this element. Otherwise, it is undefined.

The universal quantifier is strict. Once more, the argument to $\forall_s$ is a binding. If this binding is anywhere undefined, then the universal quantifier is itself undefined. Otherwise, it depends on whether every element evaluates to true or not.

$$(\forall_s(f) =_3 \bot) \lhd \bot \in \operatorname{ran} f \rhd (\forall_s(f) = (\operatorname{ran} f =_3 \{true\}))$$

Negation is strict and is modelled by the underlying strict UTP operator:

$$\neg_s(P) = \neg_3\ P$$

Similarly, disjunction is strict and is modelled by the underlying UTP strict operator:

$$\vee_s\ (P, Q) = P \vee_3 Q$$

These are the two operators introduced in Sect. 2. Their definitions are perhaps more appealing as truth tables.

$\neg_s$	
$true_3$	$false_3$
$\bot$	$\bot$
$false_3$	$true_3$

$\vee_s$	$true_3$	$\bot$	$false_3$
$true_3$	$true_3$	$\bot$	$true_3$
$\bot$	$\bot$	$\bot$	$\bot$
$false_3$	$true_3$	$\bot$	$false_3$

4.2 Kleene System

Kleene's system makes the logical connectives as defined as possible, whilst still being monotonic. So, every function is monotonic:

$$\mathbf{K1}(P) = P \land (\forall f : PShape_k \cup FShape_k \bullet \mathbf{monotonic}(f))$$

Equality and definite description are both strict:

$$(=_k) = (=_s)$$
$$(\iota_k) = (\iota_s)$$

If the binding function f for the universal quantifier evaluates anywhere to *false*, then this is enough information to constitute a counterexample, and so $\forall_k(f)$ is also *false*. Otherwise, if it evaluates everywhere to *true*, then clearly it is universally satisfied. Otherwise, it is undefined.

$$((\forall_k(f) =_3 false_3) \lhd false \in \operatorname{ran} f \rhd \\ ((\forall_k(f) =_3 true_3) \lhd (\operatorname{ran} f = \{true\}) \rhd (\forall_k(f) =_3 \bot)))$$

Negation is strict:

$$\neg_k = \neg_s$$

If either operand of a disjunction is *true*, then the disjunction is also *true*, regardless of whether the other operand is defined or not. If both are false, then so is the disjunction. Otherwise the disjunction is undefined. We end up with the following refinement to the initial definition of strict disjunction.

$$((\vee_k(P, Q) =_3 true_3) \lhd (P =_3 true_3) \vee (Q =_3 true_3) \rhd \\ ((\vee_k(P, Q) =_3 false_3) \lhd (P =_3 false_3) \wedge (Q =_3 false_3) \rhd \\ (\vee_k(P, Q) =_3 \bot)))$$

As usual, the truth table paints a clearer picture:

$\vee_k$	$true_3$	$\bot$	$false_3$
$true_3$	$true_3$	$true_3$	$true_3$
$\bot$	$true_3$	$\bot$	$\bot$
$false_3$	$true_3$	$\bot$	$false_3$

4.3 McCarthy System

McCarthy's system is very operational in flavour: it is assumed that there is an interpreter working through the text of logical constructs from left to right. The left-hand operand is evaluated first. The right-hand operand is evaluated only if it is needed. Function and predicate symbols are monotonic, just like in Kleene's system.

$$M1 = K1$$

Equality and definite description are both strict.

$$(=_m) = (=_k)$$
$$\iota_m = \iota_k$$

In general, universal quantification in McCarthy's system is just the same as in Kleene's system. However, Overture [8] uses a variant of McCarthy logic where the binding function itself is executed from left to right, which distinguishes it from Kleene logic.

$$\forall_m = \forall_k$$

Negation is the same as Kleene's.

$$\neg_m = \neg_k$$

Finally, disjunction has a short-circuit semantics which induces the distinguishing left-to-right evaluation order:

$$\begin{aligned}((\vee_m (P, Q) =_3 true_3) \lhd (P =_3 true_3) \vee ((P =_3 false_3) \wedge (Q =_3 true_3)) \rhd \\ ((\vee_m (P, Q) =_3 \bot) \lhd (P =_3 \bot) \vee (Q =_3 \bot) \rhd (\vee_m (P, Q) =_3 false_3)))\end{aligned}$$

The truth table has the following structure:

$\vee_m$	$true_3$	$\bot$	$false_3$
$true_3$	$true_3$	$true_3$	$true_3$
$\bot$	$\bot$	$\bot$	$\bot$
$false_3$	$true_3$	$\bot$	$false_3$

All three systems are monotonic.

Lemma 7 (Strict-Kleene-McCarthy monotonicity).

1. *The strict system is monotonic*
2. *The Kleene system is monotonic*
3. *The McCarthy system is monotonic*

□

There exists an interesting definedness order between the three systems. It shows the relative resilience of the three logics to undefinedness:

Lemma 8 (Strict-McCarthy-Kleene ordering). *For* $\rho_s = \rho_m = \rho_k$ *and* $Dom_s = Dom_m = Dom_k$ *we have*

$$\mathbf{FOT}_s \sqsubseteq \mathbf{FOT}_m \sqsubseteq \mathbf{FOT}_k$$

□

This lemma allows us to relate theorems proved in the different systems. Suppose that P is a theorem in the strict system; then it would also be true in the McCarthy and Kleene systems. More concretely, if we prove a theorem in VDM in Overture, then it would still be a theorem if we interpreted it in LPF, since the former is a McCarthy system and the latter is a Kleene system.

5 Guard Systems

We turn our attention now to the proof obligations that different systems can use to demonstrate the definedness of constructs.

5.1 Validity

Suppose $\boldsymbol{T}$ is a $\boldsymbol{CXT}$ and P is a predicate. Then define P is *valid* in $\boldsymbol{T}$:

$$\boldsymbol{T} \models P \ \widehat{=}\ \text{for all } \boldsymbol{U}, \boldsymbol{T} \sqsubseteq_H \boldsymbol{U} \text{ implies } P_{\boldsymbol{U}} = \mathit{true}$$

That is, any construct that is valid in a given logical system will also be valid in a logical system that refines it in the definedness order.

5.2 Guards

Suppose that c is a construct. Then predicate G is a *guard* for c in $\boldsymbol{CXT}_{\boldsymbol{T}}$ (denoted by $G \rightsquigarrow_{\boldsymbol{T}} c$) iff for every $\boldsymbol{FOT}_{\boldsymbol{V}}$ that extends $\boldsymbol{CXT}_{\boldsymbol{T}}$ we have

1. $(G_{\boldsymbol{V}} \neq \bot)$
2. $(G_{\boldsymbol{V}} = \mathit{true}) \Rightarrow (c_{\boldsymbol{V}} \neq \bot)$

G is a *tight guard* if we also have

3. $(G_{\boldsymbol{V}} = \mathit{false}) \Rightarrow (c_{\boldsymbol{V}} = \bot)$

Now we are ready to state and prove our main result, which is due originally to Saaltink.

Theorem 2 (Main theorem (Saaltink)). *Suppose that* $\boldsymbol{CXT}_{\boldsymbol{S}} \sqsubseteq \boldsymbol{CXT}_{\boldsymbol{T}}$*, that either one is monotonic, and that* G *is a guard for* P *in* $\boldsymbol{CXT}_{\boldsymbol{S}}$*. Then, if* $(\boldsymbol{T} \models G)$ *and* $(\boldsymbol{T} \models P)$*, we have that* $(\boldsymbol{S} \models P)$*.* □

The significance of this result is in trading theorems between provers, as shown in the next example.

Example 15 (Trading theorems). Suppose that we want a proof of P in Larsen's VDM, as implemented in the Overture toolset [8], but the only theorem prover we have is for Jones's VDM. Overture uses a form of McCarthy's logic, whilst Jones's VDM uses LPF, a form of Kleene's logic. By Lemma 8, we have Overture $\sqsubseteq$ LPF. We could find a guard G for P in Overture (McCarthy logic), and then can carry out the proof of both G and P in Jones's logic (Kleene). Our Main Theorem then tells us that P is a theorem in Overture. All proofs are carried out in the stronger logic, but hold in weaker one. Perhaps more interestingly, a similar theorem holds for using classical logic instead of Kleene's logic. In this way, classical logic could be used to prove results in Overture. □

Proof (Main theorem).

1. From the Models Lemma 5, since $\boldsymbol{CXT}_{\boldsymbol{S}} \sqsubseteq \boldsymbol{CXT}_{\boldsymbol{T}}$ and $\boldsymbol{FOT}_{\boldsymbol{U}}$ extends $\boldsymbol{CXT}_{\boldsymbol{S}}$, then there exists $\boldsymbol{FOT}_{\boldsymbol{V}}$ that extends $\boldsymbol{CXT}_{\boldsymbol{T}}$ and for which we have $\boldsymbol{FOT}_{\boldsymbol{U}} \sqsubseteq \boldsymbol{FOT}_{\boldsymbol{V}}$.
2. Since $G \rightsquigarrow_{\boldsymbol{S}} P$, know that $(G_{\boldsymbol{U}} \neq \bot) \wedge ((G_{\boldsymbol{U}} = \mathit{true}) \Rightarrow (P_{\boldsymbol{U}} \neq \bot))$ from the definition of a guard.

3. Now, from construct monotonicity (since $\boldsymbol{S}$ is monotonic) we have that $G_U \sqsubseteq G_V$. But because $(G_U \neq \bot)$, it must be that $(G_U = G_V)$. We are assuming that G is valid in T $(\boldsymbol{T} \models G)$, so we have that $(G_V = true)$ and so $(G_U = true)$. Now, from the definition of a guard, we must have that $(P_U \neq \bot)$
4. We now repeat this argument for P. By construct monotonicity, ($\boldsymbol{S}$ monotonic), we have $P_U \sqsubseteq P_V$, therefore $(P_U = P_V)$. But $\boldsymbol{T} \models P$, so $(P_V = true)$ and therefore $(P_U = true)$.

□

5.3 Definedness Guards

Suppose that e is an expression. We use the notation $\mathcal{D}e$ to define the circumstances under which e is defined.

Example 16 (Definedness guard)

$$\mathcal{D}((x + y)/z) = z \neq 0$$

□

The definedness guards that we are interested in are all first order; that is, the guards themselves are always defined.

Definition 14 (First-order definedness). *The definedness function is first order:*

$$\boldsymbol{D1}(\mathcal{D}\Phi) \;\widehat{=}\; \mathcal{D}\Phi \wedge \mathcal{D}(\mathcal{D}\Phi)$$

□

If we define a system of guards for every construct in our language, then we can use this system inductively to generate verification conditions for the definedness of all constructs. In the next section we demonstrate this for the case of the definite McCarthy system.

5.4 Guards for Definite McCarthy System

Assuming we have a theory $\boldsymbol{T}$ of McCarthy logic, we can develop the following recursive definedness conditions for constructs c of that theory.

$$\begin{aligned}
\mathcal{D}_m x &= true \\
\mathcal{D}_m(p(e)) &= \forall\, i : 1 \,..\, \rho(P) \bullet \mathcal{D}_m e_i \\
\mathcal{D}_m(f(e)) &= \forall\, i : 1 \,..\, \rho(f) \bullet \mathcal{D}_m e_i \\
\mathcal{D}_m(e_1 = e_2) &= \mathcal{D}_m e_1 \wedge \mathcal{D}_m e_2 \\
\mathcal{D}_m(\neg\, P) &= \mathcal{D}_m P \\
\mathcal{D}_m(P \vee Q) &= \mathcal{D}_m P \wedge (P \vee \mathcal{D}_m Q) \\
\mathcal{D}_m(\forall\, x \bullet P) &= \forall\, x \bullet \mathcal{D}_m P \\
\mathcal{D}_m(\iota x \bullet P) &= (\forall\, x \bullet \mathcal{D}_m P) \wedge (\exists_1\, x \bullet P)
\end{aligned}$$

A theorem follows immediately, which has a Kleene analogue and (trivially) a strict analogue as well.

Theorem 3 (McCarthy guards). *If c is a construct, then $\mathcal{D}_m(c)$ is a guard for c in $\boldsymbol{definite}(\boldsymbol{T})$, and a tight guard for c in $\boldsymbol{strict}(\boldsymbol{definite}(\boldsymbol{T}))$.* □

No analogue of this theorem exists for indefinite systems, but the partitioning of predicates into TVL pairs (P, Q) allows us to extract the guard condition immediately from Q. The advantage is that Q may be tailored to be either a plain or a tight guard, depending on the application.

6 Conclusions

The notion of undefinedness has played a prominent role in the study of logic, and continues to be a relevant research problem. With tools emerging that employ more than simple classical logic, and their use being adopted for verification in the heterogeneous landscape of systems of systems, a treatment of the relationships among different logics becomes necessary. In this section we summarize our specific contributions and prospects in this direction.

6.1 Contributions

We have presented a unifying theory for monotonic partial logics with undefined expressions, as a foundation for exploring the formal basis for migrating theorems between tools and methods that employ different types of logic and treatments of undefinedness. The aim is to support the forthcoming COMPASS Modelling Language. Based closely on Saaltink's original work, but cast in Hoare & He's Unifying Theories of Programming, we have demonstrated an information-theoretic unification for three logical systems: strict, McCarthy, and Kleene. Other approaches are possible and are under investigation.

6.2 Future Work

In this paper we have told only part of the story, since *CML* is not restricted to definite constructs: precondition predicates are needed for handling indefinite expressions and predicates. Our next step will be to extend our work in this way, thus developing a comprehensive treatment of undefined expressions for *CML*.

Fortunately we see many avenues of research starting here. Can our unifying theory cope with every treatment of undefinedness, such as (i) the Alloy paradigm, where there is no function application; (ii) the logic of LCF, where quantifiers also range over undefined values; (iii) second-order undefinedness; (iv) logics with more than three values. These are all important contemporary logical treatments of undefinedness that can not be excluded from such an unification effort.

References

1. Agerholm, S., Frost, J.: An Isabelle-based Theorem Prover for VDM-SL. In: Gunter, E.L., Felty, A.P. (eds.) TPHOLs 1997. LNCS, vol. 1275, pp. 1–16. Springer, Heidelberg (1997)

2. Bergmann, M.: An Introduction to Many-Valued and Fuzzy Logic: Semantics, Algebras and Derivation Systems. Cambridge University Press (2008)
3. Bergstra, J.A., Bethke, I., Rodenburg, P.: A propositional logic with 4 values: true, false, divergent and meaningless. Journal of Applied NonClassical Logics 5, 199–217 (1995)
4. Bochvar, D.A., Bergmann, M.: On a three-valued logical calculus and its application to the analysis of the paradoxes of the classical extended functional calculus. History and Philosophy of Logic 2(1), 87–112 (1981)
5. Goldsmith, M.: FDR2 user's manual. Technical Report Version 2.82. Formal Systems (Europe) Ltd. (2005)
6. Gordon, M., Wadsworth, C.P., Milner, R.: Edinburgh LCF. LNCS, vol. 78. Springer, Heidelberg (1979)
7. Hoare, C.A.R., He, J.: Unifying Theories of Programming. Series in Computer Science. Prentice Hall (1998)
8. Larsen, P.G., Battle, N., Ferreira, M.A., Fitzgerald, J.S., Lausdahl, K., Verhoef, M.: The Overture initiative: integrating tools for VDM. ACM SIGSOFT Software Engineering Notes 35, 1–6 (2010)
9. Rose, A.: A lattice-theoretic characterisation of three-valued logic. Journal of the London Mathematical Society 25, 255–259 (1950)
10. Saaltink, M.: The Z/EVES System. In: Bowen, J.P., Hinchey, M.G., Till, D. (eds.) ZUM 1997. LNCS, vol. 1212, pp. 72–85. Springer, Heidelberg (1997)
11. Woodcock, J., Saaltink, M., Freitas, L.: Unifying theories of undefinedness. In: Summer School Marktoberdorf 2008: Engineering Methods and Tools for Software Safety and Security. NATO ASI Series F. IOS Press, Amsterdam (2009)
12. Woodcock, J., Freitas, L.: Linking VDM and Z. In: Hinchey, M. (ed.) ICECCS, pp. 143–152. IEEE Computer Society (2008)