Available online at www.sciencedirect.com
SCIENCE DIRECT®

Electronic Notes in Theoretical Computer Science

ELSEVIER Electronic Notes in Theoretical Computer Science 137 (2005) 111–129
www.elsevier.com/locate/entcs

The Specification Logic νZ

Martin C. Henson[1] Besnik Kajtazi[2]

Department of Computer Science
University of Essex
Wivenhoe Park, Colchester, Essex, CO4 3SQ, UK

Abstract

This paper introduces a wide-spectrum specification logic νZ. The minimal core logic is extended to a more expressive specification logic which includes a schema calculus similar (but not equivalent) to Z, some new additional schema operators and extensions to a programming and program development logic.

Keywords: Specification Language, Wide Spectrum Language, Specification Logic, Operation Refinement, Program Development.

1 Introduction

In this paper we introduce a wide-spectrum logic νZ. This is a very small specification logic based on a total correctness relational semantics with refinement as its fundamental relation.

The language which underlies the logic is Z-like, that is to say, we have schemas and schema operators. A significant difference is that operation schemas have two predicates, so resemble more the specification statements of the refinement calculus (*e.g.* [1] and [6]). This is, in fact, a fairly trivial difference, and the language could easily be set up using single predicate schemas if preferred. On the other hand, there are several significant differences between νZ and Z:

[1] Email: hensm@essex.ac.uk
[2] Email: bkajta@essex.ac.uk

doi:10.1016/j.entcs.2005.04.027

- Z is based on a *partial*-correctness semantics; νZ is based on a *total*-correctness semantics.
- Z permits refinement of over–specifications; νZ does not.
- Z schema operators are not monotonic; νZ schema operators are monotonic (anti-monotonic).
- Z is based on *equality*; νZ is based on *refinement*.
- Z is a *specification* language; νZ is *wide-spectrum*.
- Z is relatively inflexible; νZ is extensible.
- Z is a *language*; νZ is a *logic*.

In this introductory paper we concentrate entirely on the system itself, its mathematical basis and methodologies for extending the core framework with additional features for specification, for programming and for program development. In future publications we will explore more pragmatic issues, providing techniques and examples to demonstrate how to effectively specify, refine and implement systems within νZ.

2 Core νZ

νZ is interpreted within the logic $\mathcal{Z}_{\mathcal{C}}^{\perp}$, the extension of $\mathcal{Z}_{\mathcal{C}}$ introduced in [3] which includes $\perp$ elements in all types. We assume familiarity with this theory (and notational conventions); all this is also covered in [4].

2.1 Syntax of νZ

The syntax of the core νZ framework is minimal. The type of an *operation schema*, U, is $\mathbb{P}\, T$ (written $U^{\mathbb{P}\, T}$) where T is a schema type which has the form $\Delta V = V \curlywedge V'$. Generally we will, as is usual in Z, write ΔV for $V \curlywedge V'$. We will write $U(v)$ to indicate that variable v may appear free in the schema expression U.[3]

[3] When the variable has the type $\mathbb{P}\, T$ and T is a schema type (that is: it is a variable over schemas) we shall write it in $\mathcal{Z}_{\mathcal{C}}^{\perp}$, as we do in νZ, in upper-case.

Definition 2.1

$$
\begin{array}{lll}
U^{\mathbb{P}\,T} ::= & & \\
\quad X^{\mathbb{P}\,T} & & \text{– schema variable} \\
\quad [T \mid P \mid Q] & & \text{– atomic specifications} \\
\quad \neg\, U^{\mathbb{P}\,T} & & \text{– negation} \\
\quad U_0^{\mathbb{P}\,T_0} \vee\, U_1^{\mathbb{P}\,T_1} & (T = T_0 \curlyvee T_1) & \text{– disjunction} \\
\quad \exists\, \mathtt{x}^{T_{\mathtt{x}}} \bullet\, U_0^{\mathbb{P}\,T_0} & (T = T_0 - T_{\mathtt{x}}) & \text{– existential hiding} \\
\quad \mu X^{\mathbb{P}\,T} \bullet\, U(X)^{\mathbb{P}\,T} & & \text{– recursive schemas}
\end{array}
$$

2.2 Semantics of $\nu\mathsf{Z}$

We first need to define refinement. In this framework it is simply containment.

Definition 2.2

$$U_0^{\mathbb{P}\,T} \sqsupseteq U_1^{\mathbb{P}\,T} =_{df} [\![U_0]\!] \subseteq_T [\![U_1]\!]$$

We also need to specify the universe of specification models for a given type. Part (ii) is based on [2] section 8.1.

Definition 2.3

$$
\begin{array}{lll}
(i)\ \ magic^{\mathbb{P}\,T} & =_{df} & [T \mid true \mid false] \\
(ii)\ \ W_T & =_{df} & \{[\![U^{\mathbb{P}\,T}]\!] \mid magic^{\mathbb{P}\,T} \sqsupseteq U \wedge U \fatsemi magic^{\mathbb{P}(\Delta T^{out})} \sqsupseteq U\}
\end{array}
$$

Now we have the semantics of specifications.

Definition 2.4 In what follows, $T^\star =_{df} V_\perp \star V'_\perp$. The types are omitted here, but are taken to be as specified in the syntax above.

$$
\begin{array}{lll}
[\![X]\!] & =_{df} & X \\
[\![[T \mid P \mid Q]]\!] & =_{df} & \{z_0 \star z'_1 \in T^\star \mid z_0.P \Rightarrow z_0.z'_1.Q\} \\
[\![\neg U]\!] & =_{df} & \{z \in T^\star \mid z =_V \perp \vee\, z \notin U\} \\
[\![U_0 \vee U_1]\!] & =_{df} & \{z \in T^\star \mid z \dot{\in} [\![U_0]\!] \vee z \dot{\in} [\![U_1]\!]\} \\
[\![\exists\, \mathtt{x} \bullet U_0]\!] & =_{df} & \{z \in T^\star \mid \exists\, y \in T_0^\star \bullet y \in [\![U_0]\!] \wedge z = y \upharpoonright T\} \\
[\![\mu X \bullet U(X)]\!] & =_{df} & \bigsqcap\{X \in W_T \mid [\![U(X)]\!] \sqsupseteq X\}
\end{array}
$$

In the case of recursion, the schema variable X must appear in a *positive* position in U. That is: this is monotone recursion. The notation $t.P$ indicates the usual distribution of the binding t through the proposition P so that its component observations $\mathtt{x}$ are replaced by $t.\mathtt{x}$. Note that $\bot.P = \mathit{false}$ for all P ($\bot$ satisfies nothing, in particular it is outside every precondition).

2.3 Logic of $\nu\mathtt{Z}$

The semantics induces a logic for the constructs, as follows. In this introductory paper we omit the proofs.

2.3.1 Refinement

The rules for operation refinement in $\nu\mathtt{Z}$ are as follows:

Proposition 2.5 *Let z be a fresh variable.*

$$\frac{z \in U_0 \vdash z \in U_1}{U_0 \sqsupseteq U_1}\ (\sqsupseteq^+) \qquad \frac{U_0 \sqsupseteq U_1 \quad t \in U_0}{t \in U_1}\ (\sqsupseteq^-)$$

2.3.2 Atomic operation schemas

The rules for atomic operation schema in $\nu\mathtt{Z}$ are as follows:

Proposition 2.6

$$\frac{t_0.P \vdash t_0.t_1'.Q}{t_0 \star t_1' \in [T \mid P \mid Q]}\ (U^+) \qquad \frac{t_0 \star t_1' \in [T \mid P \mid Q] \quad t_0.P}{t_1'.Q}\ (U^-)$$

2.3.3 Negated schemas

Note that negation in $\nu\mathtt{Z}$ is not the relational inverse: it is well-known that the universe of total-correctness relations in this model is not closed under that operation (see *e.g.* [2]). An alternative characterisation of the semantics is available using a combination of relational inverse, disjunction and magic.

Definition 2.7

$$\neg U = U^{-1} \vee \mathit{magic}$$

In any event, the rules for negation are derivable:

Proposition 2.8

$$\frac{t \notin U}{t \in \neg U}\ (U^+_{\neg_0}) \qquad \frac{t_0 = \bot}{t_0 \star t_1' \in \neg U}\ (U^+_{\neg_1})$$

$$\frac{t_0 \star t_1' \in \neg U \quad t_0 \star t_1' \notin U \vdash P \quad t_0 = \bot \vdash P}{P}\ (U_\neg^-)$$

The notion satisfies double negation and excluded middle.

Proposition 2.9

$$\frac{t \in U}{t \in \neg\neg U} \qquad \frac{t \in \neg\neg U}{t \in U} \qquad \frac{}{t \in \neg U \vee U}$$

2.3.4 Disjunction schemas

The rules for disjunction schemas in $\nu\mathtt{Z}$ are derivable, as follows:

Proposition 2.10 *Let $i \in 2$.*

$$\frac{t \dot\in U_i}{t \in U_0 \vee U_1}\ (U_{\vee_i}^+) \qquad \frac{t \in U_0 \vee U_1 \quad t \dot\in U_0 \vdash P \quad t \dot\in U_1 \vdash P}{P}\ (U_\vee^-)$$

2.3.5 Existential hiding schemas

The rules for existential hiding schemas in $\nu\mathtt{Z}$ are derivable, as follows:

Proposition 2.11

$$\frac{t \in U}{t \dot\in \exists\, \mathtt{x}^{T_\mathtt{x}} \bullet U}\ (U_\exists^+) \qquad \frac{t \in \exists\, \mathtt{x}^{T_\mathtt{x}} \bullet U \quad t \star \langle\!|\, \mathtt{x} \Rrightarrow y \,|\!\rangle \in U \vdash P}{P}\ (U_\exists^-)$$

2.3.6 Recursive schemas

The rules for recursive schemas in $\nu\mathtt{Z}$ are derivable, as follows:

Proposition 2.12

$$\frac{t \in U(\mu X \bullet U(X))}{t \in \mu X \bullet U(X)}\ (\mu^+) \qquad \frac{t \in \mu X \bullet U(X)}{t \in U(\mu X \bullet U(X))}\ (\mu^-)$$

3 Specifying a specification language in $\nu\mathtt{Z}$

The principles on which $\nu\mathtt{Z}$ is based include *economy* (the core system begin so small) and *extensibility* (the ease with which the core system can be made more expressive). Since the core system is so inexpressive, a first ambition will be to provide additional infrastructure which provides for a considerably

more expressive specification language. We cover some aspects of this in this section, beginning with extensions providing other standard schema operators.

Some of the operators which we consider here are familiar from Z (though, because the semantics is differnt, the logic of these operators departs from that in Z). In addition there will be variations on familiar operators, such as composition: in this section we provide a notion of composition which allows *arbitrary* schemas to be composed, even when those schemas do not match for type. Finally, we introduce a range of quite new operators, unfamiliar in Z, which we will see have some use when we turn to the topic of programming languages and program development logics in later sections.

3.1 Schema conjunction and implication

We can define these in terms of disjunction and negation, using the usual de Morgan definitions. We omit the proofs, which are a little more involved than usual, due to the more complex notion of negation we are obliged to use.

Definition 3.1

$$U_0 \wedge U_1 =_{df} \neg(\neg U_0 \vee \neg U_1)$$

The usual rules are derivable.

Proposition 3.2 *Let $i \in 2$.*

$$\frac{t \in U_0 \quad t \in U_1}{t \in U_0 \wedge U_1}\ (U^+_\wedge) \qquad \frac{t \in U_0 \wedge U_1}{t \in U_i}\ (U^-_{\wedge_i})$$

Definition 3.3

$$U_0 \Rightarrow U_1 =_{df} \neg U_0 \vee U_1$$

With the obvious rules derivable:

Proposition 3.4

$$\frac{z \in U_0 \vdash z \in U_1}{z \in U_0 \Rightarrow U_1}\ (U^+_\Rightarrow) \qquad \frac{t \in U_0 \Rightarrow U_1 \quad t \in U_0}{t \in U_1}\ (U^-_\Rightarrow)$$

3.2 Universal Hiding

Universal hiding is defined in terms of existential hiding and negation, using the standard de Morgan definition. We provide the proofs in detail, for illustration.

Definition 3.5

$$\forall \mathtt{x}^{T_\mathtt{x}} \bullet U =_{df} \neg \exists \mathtt{x}^{T_\mathtt{x}} \bullet \neg U$$

And then the usual introduction and elimination rules are derivable.

Proposition 3.6 *Let z be a fresh variable. We assume that t has the form $t_0 \star t_1'$.*

$$\frac{t \star \langle\!| \, \mathtt{x} \Rrightarrow z \, |\!\rangle \in U}{t \in \forall \mathtt{x}^{T_{\mathtt{x}}} \bullet U}$$

Proof.

$$\dfrac{\dfrac{}{t_0 = \bot \vee\ t_0 \neq \bot} \quad \dfrac{\dfrac{}{t_0 = \bot}\,(0)}{t \in \neg \exists \mathtt{x}^{T_{\mathtt{x}}} \bullet \neg U} \quad \begin{array}{c}\delta_0\\ \vdots \\ t \in \neg \exists \mathtt{x}^{T_{\mathtt{x}}} \bullet \neg U\end{array}}{t \in \neg \exists \mathtt{x}^{T_{\mathtt{x}}} \bullet \neg U}\,(0)$$

where δ_0 is:

$$\dfrac{\dfrac{}{t \in \exists \mathtt{x}^{T_{\mathtt{x}}} \bullet \neg U}\,(1) \quad \begin{array}{c}\delta_1\\ \vdots \\ \mathit{false}\end{array}}{\dfrac{\dfrac{\mathit{false}}{t \notin \exists \mathtt{x}^{T_{\mathtt{x}}} \bullet \neg U}\,(1)}{t \in \neg \exists \mathtt{x}^{T_{\mathtt{x}}} \bullet \neg U}}\,(2)$$

and where δ_1 is:

$$\dfrac{\dfrac{}{t \star \langle\!| \, \mathtt{x} \Rrightarrow z \, |\!\rangle \in \neg U}\,(2) \quad \dfrac{\dfrac{}{t \star \langle\!| \, \mathtt{x} \Rrightarrow z \, |\!\rangle \notin U}\,(3) \quad t \star \langle\!| \, \mathtt{x} \Rrightarrow z \, |\!\rangle \in U}{\mathit{false}} \quad \dfrac{\dfrac{}{t_0 = \bot}\,(3) \quad \dfrac{}{t_0 \neq \bot}\,(0)}{\mathit{false}}}{\mathit{false}}\,(3)$$

□

Proposition 3.7 *Let t have the form $t_0 \star t_1'$.*

$$\frac{t \in \forall \mathtt{x}^{T_{\mathtt{x}}} \bullet U \quad v \in T_{\mathtt{x}}}{t \star \langle\!| \, \mathtt{x} \Rrightarrow v \, |\!\rangle \in U}$$

Proof.

$$\dfrac{\dfrac{}{t_0 = \bot \vee\ t_0 \neq \bot}\,(LEM) \quad \begin{array}{c}\delta_0\\ \vdots \\ t \star \langle\!| \, \mathtt{x} \Rrightarrow v \, |\!\rangle \in U\end{array} \quad \begin{array}{c}\delta_1\\ \vdots \\ t \star \langle\!| \, \mathtt{x} \Rrightarrow v \, |\!\rangle \in U\end{array}}{t \star \langle\!| \, \mathtt{x} \Rrightarrow v \, |\!\rangle \in U}\,(0)$$

where δ_0 is:

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{t_0 \star \langle\!|\, \mathtt{x} \Rrightarrow v \,|\!\rangle \notin U}\,(1)}{t_0 \star \langle\!|\, \mathtt{x} \Rrightarrow v \,|\!\rangle \in \neg U}}{t_0 \star \langle\!|\, \mathtt{x} \Rrightarrow v \,|\!\rangle \in T^\star}}{t_0 \star \langle\!|\, \mathtt{x} \Rrightarrow v \,|\!\rangle \in T_{0\perp}} \qquad \dfrac{\dfrac{}{t_0 = \perp}\,(0)}{\dfrac{}{t_0 \neq \perp}} \qquad \dfrac{\dfrac{}{t_0 \star \langle\!|\, \mathtt{x} \Rrightarrow v \,|\!\rangle \in T_0}\,(2)}{t_0 \dot{\in} T_0} \qquad \dfrac{\dfrac{\dfrac{}{t_0 \star \langle\!|\, \mathtt{x} \Rrightarrow v \,|\!\rangle = \perp}\,(2)}{\langle\!|\, \mathtt{x} \Rrightarrow v \,|\!\rangle = \perp}}{v = \perp} \qquad \dfrac{}{v \neq \perp}}{\dfrac{\dfrac{}{false}}{t \star \langle\!|\, \mathtt{x} \Rrightarrow v \,|\!\rangle \in U}\,(1)}\,(2)$$

and δ_1 is:

$$\dfrac{t \in \neg \exists\, \mathtt{x}^{T_\mathtt{x}} \bullet \neg U \qquad \dfrac{\dfrac{\dfrac{}{t \star \langle\!|\, \mathtt{x} \Rrightarrow v \,|\!\rangle \notin U}\,(4)}{t \star \langle\!|\, \mathtt{x} \Rrightarrow v \,|\!\rangle \in \neg U}}{t \in \exists\, \mathtt{x}^{T_\mathtt{x}} \bullet \neg U} \qquad \dfrac{\dfrac{}{t_0 \star \langle\!|\, \mathtt{x} \Rrightarrow\ \perp \,|\!\rangle \neq \perp}\,(0)}{t_0 \neq \perp}}{\dfrac{false}{t \star \langle\!|\, \mathtt{x} \Rrightarrow v \,|\!\rangle \in U}\,(4)}$$

□

3.3 Ξ-schemas

We have the usual idea of Ξ-schemas:

Definition 3.8

$$\Xi T =_{df} [\Delta T \mid true \mid \theta T = \theta' T]$$

The rules are straightforward:

Proposition 3.9

$$\frac{}{t \star t' \in \Xi T} \qquad \frac{t_0 \star t_1' \in \Xi T}{t_0 = t_1}$$

3.4 The `skip`*-extension*

We use this to define the `skip`-extension of a schema:

Definition 3.10 When T_0 and T_1 are disjoint, we define:

$$U^{\mathbb{P}\, T_0} \diamond T_1 =_{df} U \wedge \Xi T_1$$

Naturally this is well-defined even when the types are not disjoint, but the purpose of this is, as described, to extend a schema with `skip` and the definition has pathological effects in other circumstances.

The rules are straightforward:

Proposition 3.11

$$\frac{t_0 \star t_1' \mathrel{\dot\in} U \quad t_0 =_T t_1}{t_0 \star t_1' \in U \diamond T}\ (U^+_\diamond)$$

$$\frac{t \in U \diamond T}{t \mathrel{\dot\in} U}\ (U^-_{\diamond_0}) \qquad \frac{t_0 \star t_1' \in U \diamond T}{t_0 =_T t_1}\ (U^-_{\diamond_1})$$

3.5 Schema Composition

In $\nu\mathtt{Z}$ we wish to compose *arbitrary* specifications; even when the types of the operations do not match. In this regard $\nu\mathtt{Z}$ differs from Z. For such compositions to make sense, it is necessary to match incompatible types and to ensure that operations do not arbitrarily adjust bindings in the process. The definition of schema composition in $\nu\mathtt{Z}$ is, therefore, a little more complex than in Z. Nevertheless, it is possible to specify composition in the core theory, using the `skip`-extension operator.

Definition 3.12 Let $T_L = T_1 - T_0$ with the form $\Delta T_L = T_L^{in} \curlyvee T_L^{out'}$ and let Let $T_R = T_0 - T_1$ with the form $\Delta T_R = T_R^{in} \curlyvee T_R^{out'}$. Let $\overline{\mathtt{t}}$ be a vector of fresh observations with the size of the alphabet of $T_0^{out'} \curlyvee T_L^{out'}$ (equivalently: $T_1^{in} \curlyvee T_R^{in}$).

$$U_0^{\mathbb{P}(T_0^{in} \curlyvee T_0^{out'})} \fatsemi U_1^{\mathbb{P}(T_1^{in} \curlyvee T_1^{out'})} =_{df} \exists \overline{\mathtt{t}} \bullet (U_0 \diamond T_L)[\alpha(T_0^{out'} \curlyvee T_L^{out'})/\overline{\mathtt{t}}] \wedge (U_1 \diamond T_R)[\alpha(T_1^{in} \curlyvee T_R^{in})/\overline{\mathtt{t}}]$$

The following introduction and elimination rules are derivable for schema composition:

Proposition 3.13

$$\frac{t_0 \star t_2' \mathrel{\dot\in} U_0 \quad t_0 =_{T_L} t_2 \quad t_2 \star t_1' \mathrel{\dot\in} U_1 \quad t_2 =_{T_R} t_1}{t_0 \star t_1' \in U_0 \fatsemi U_1}\ (U^+_\fatsemi)$$

Proposition 3.14

$$\frac{t_0 \star t_1' \in U_0 \fatsemi U_1 \quad t_0 \star t_2' \mathrel{\dot\in} U_0, t_0 =_{T_L} t_2, t_2 \star t_1' \mathrel{\dot\in} U_1, t_2 =_{T_R} t_1 \vdash P}{P}\ (U^-_\fatsemi)$$

3.6 Restricted chaos

This definition introduces a restricted form of *chaos*: outside P this schema blocks.

Definition 3.15

$$chaos_P =_{df} [T \mid \neg P \mid false]$$

This leads to the following logical rules.

Proposition 3.16

$$\frac{t_0.P}{t_0 \star t_1' \in chaos_P}\ (chaos_P^+) \qquad \frac{t_0 \star t_1' \in chaos_P \quad \neg t_0.P}{false}\ (chaos_P^-)$$

3.7 Schema specialisation

We use restricted chaos to introduce the specialisation of a schema at a particular observation (it blocks elsewhere).

Definition 3.17 Let E_T be the schema type corresponding to the observations contained in E. Let $\Delta[\mathtt{x}^{T_\mathtt{x}}] \preceq T$.

$$U^{\mathbb{P}\,T}[\mathtt{x} \Rrightarrow E^{E_T}] =_{df} chaos^{\mathbb{P}\,\Delta([\mathtt{x}^{T_\mathtt{x}}] \curlyvee \mathtt{E}_\mathtt{T})}_{(\mathtt{x}=\mathtt{E})} \wedge U$$

This induces the following rules:

Proposition 3.18

$$\frac{t \dot{\in} U \quad t.\mathtt{x} = z.E}{t \in U[\mathtt{x} \Rrightarrow E]} \qquad \frac{t \in U[\mathtt{x} \Rrightarrow E]}{t \dot{\in} U} \qquad \frac{t \in U[\mathtt{x} \Rrightarrow E]}{t.\mathtt{x} = t.E}$$

3.8 Strengthening preconditions

This operator has the effect of (in general) strengthening the precondition of a schema U by stipulating an additional condition P.

Definition 3.19 Let $T_P \preceq T$.

$$U^{\mathbb{P}\,T} \upharpoonright P^{T_P} =_{df} chaos_P \Rightarrow U$$

The operator is governed by induced logical rules.

Proposition 3.20

$$\frac{t.P \vdash t \in U}{t \in U \upharpoonright P} \qquad \frac{t \in U \upharpoonright P \quad t.P}{t \in U}$$

4 Specifying a programming language in $\nu\mathtt{Z}$

It is central to the methodology of $\nu\mathtt{Z}$ that it smoothly integrates specification and programming, and that it is possible to develop programs from specifications. This is achieved by firstly *specifying* a programming language in $\nu\mathtt{Z}$ and then inducing a corresponding program logic: refinement then automatically permits development from specifications to programs. We will develop such a language incrementally in this section.

4.1 `skip`

Definition 4.1

$$\mathtt{skip}^{\mathbb{P}(\Delta T)} =_{df} \Xi T$$

Rules for `skip`:

Proposition 4.2

$$\frac{}{t \star t' \in \mathtt{skip}}\ (\mathtt{skip}^{+}) \qquad \frac{t_0 \star t_1' \in \mathtt{skip}}{t_0 = t_1}\ (\mathtt{skip}^{-})$$

Inequation:

Proposition 4.3

$$\frac{\theta T = \theta' T \vdash Q}{\mathtt{skip} \sqsupseteq [T \mid P \mid Q]}$$

4.2 Assignment

Let $V = T_E - [x^T]$

Definition 4.4

$$\mathtt{x:=}\ E =_{df} [\Delta[x^T] \mid true \mid x' = E] \wedge \Xi V$$

Rules for assignment:

Proposition 4.5

$$\frac{}{t \star t'[\mathtt{x}'/t.E] \in \mathtt{x:=}\ E}\ (\mathtt{:=}^{+}) \qquad \frac{t_0 \star t_1' \in \mathtt{x:=}\ E}{t_0[\mathtt{x}/t_0.E] = t_1}\ (\mathtt{:=}^{-})$$

Rule for assignment:

$$\frac{z.P \ \vdash\ z.z'[\mathtt{x}'/z.E].Q}{\mathtt{x:=}E \sqsupseteq [\mathtt{x}^T;\ V \mid P \mid Q]}$$

4.3 Conditional

We define a new operator, a conditional schema, in terms of conjunction and strengthening of preconditions:

Definition 4.6 Let $T_D \preceq T_0 \curlywedge T_1$.

$$\texttt{if } D \texttt{ then } U_0^{\mathbb{P}\,T_0} \texttt{ else } U_1^{\mathbb{P}\,T_1} =_{df} U_0 \uparrow D \wedge U_1 \uparrow \neg D$$

Rules for the conditional:

Proposition 4.7

$$\frac{t.D \vdash z \mathrel{\dot\in} U_0 \quad \neg t.D \vdash t \mathrel{\dot\in} U_1}{t \in \texttt{if } D \texttt{ then } U_0 \texttt{ else } U_1}\ (\texttt{if}^+)$$

$$\frac{t \in \texttt{if } D \texttt{ then } U_0 \texttt{ else } U_1 \quad t.D}{t \mathrel{\dot\in} U_0}\ (\texttt{if}_0^-)$$

$$\frac{t \in \texttt{if } D \texttt{ then } U_0 \texttt{ else } U_1 \quad t.(\neg D)}{t \mathrel{\dot\in} U_1}\ (\texttt{if}_1^-)$$

Equations and inequations:

Proposition 4.8

$$\texttt{if } \mathit{true} \texttt{ then } U_0 \texttt{ else } U_1 \doteq U_0$$

Proof. Follow from specialisations of the introduction rule and the first elimination rule:

$$\dfrac{\dfrac{\dfrac{}{\mathit{false}}\,(1)}{z \mathrel{\dot\in} U_0 \quad z \mathrel{\dot\in} U_1}\,(1)}{\dfrac{z \in \texttt{if } D \texttt{ then } U_0 \texttt{ else } U_1 \quad z \in \texttt{if } D \texttt{ then } U_0 \texttt{ else } U_1}{z \mathrel{\dot\in} U_0}}$$

□

Proposition 4.9

$$\texttt{if } \mathit{false} \texttt{ then } U_0 \texttt{ else } U_1 \doteq U_1$$

Proposition 4.10

$$\texttt{if } D \texttt{ then } [T \mid P \mid D \wedge Q] \texttt{ else } [T \mid P \mid \neg D \wedge Q] \sqsupseteq [T \mid P \mid Q]$$

Proof. In what follows we write ϕ for

$$z \in \texttt{if } D \texttt{ then } [T \mid P \mid D \wedge Q] \texttt{ else } [T \mid P \mid \neg D \wedge Q]$$

$$\dfrac{\dfrac{\overline{D \vee \neg D} \qquad \dfrac{\dfrac{\overline{z.P}\ (1) \quad \dfrac{\phi \quad \overline{z.D}\ (2)}{z \in [T \mid P \mid D \wedge Q]}}{z.(D \wedge Q)}}{z.Q} \qquad \dfrac{\dfrac{\overline{z.P}\ (1) \quad \dfrac{\phi \quad \overline{z.(\neg D)}\ (2)}{z \in [T \mid P \mid \neg D \wedge Q]}}{z.(\neg D \wedge Q)}}{z.Q}\ (2)}{z.Q}}{z \in [T \mid P \mid Q]}\ (1)$$

□

4.4 Cases

The previous section can easily be generalised to case commands:

We define a new operator, a case schema, in terms of conjunction and strengthening of preconditions:

Definition 4.11 Let $T = \{\cdots c_i \cdots\}$.

$$\texttt{cases } E^T \texttt{ in } c_0 : U_0^{\mathbb{P}\, T_0} \ \cdots \ c_n : U_n^{\mathbb{P}\, T_n} \texttt{ endcases} =_{def} U_0 \uparrow E = c_0 \wedge \cdots \wedge U_n \uparrow E = c_n$$

Rules and inequations are omitted here, but are obvious generalisations of those for the conditional.

4.5 Scope

Definition 4.12

$$\texttt{begin var x} : T_{\mathtt{x}}\texttt{; } U \texttt{ end} =_{df} \exists\, \mathtt{x}^{T_{\mathtt{x}}}, \mathtt{x}'^{T_{\mathtt{x}}} \bullet U$$

Proposition 4.13

$$\frac{t \in U}{t \mathrel{\dot\in} \texttt{begin var x} : T_{\mathtt{x}}\texttt{; } U \texttt{ end}}\ (\texttt{begin}^+)$$

$$\frac{t \in \texttt{begin var x} : T_{\mathtt{x}}\texttt{; } U \texttt{ end} \quad t \star \langle\!|\, \mathtt{x} \Rrightarrow y_0, \mathtt{x}' \Rrightarrow y_1 \,|\!\rangle \in U \vdash P}{P}\ (\texttt{begin}^-)$$

4.6 Procedure call

This and the interpretation of procedures themselves are mutually dependent. Suppose that f is a procedure (we will see an example in the next section), then procedure call is trivially defined:

$$f(E) =_{df} f[\mathtt{x} \Rrightarrow E]$$

This leads to inference rules:

$$\frac{t \mathrel{\dot\in} f \quad t.\mathtt{x} = t.E}{t \in f(E)} \qquad \frac{t \in f(E)}{t \mathrel{\dot\in} f} \qquad \frac{t \in f(E)}{t.\mathtt{x} = t.E}$$

It is necessary to analyse this in advance of procedures themselves, as it is implicated in the definition, as we will now see.

4.7 Primitive recursive procedures over numbers

We define a new schema operator, primitive recursion over the natural numbers, in terms of conjunction, strengthening of preconditions, existential hiding, schema application and recursive schemas.

Definition 4.14

$$\mathtt{proc}\ f(\mathtt{x})\ \mathtt{cases}\ \mathtt{x}\ \mathtt{in}\ 0 : U_0;\ \mathtt{m}+1 : U_1(f(\mathtt{m}))\ \mathtt{endcases} =_{df}$$
$$\mu X \bullet U_0 \uparrow \mathtt{x} = 0 \land \exists\, \mathtt{m} \bullet U_1(X[\mathtt{x} \Rrightarrow \mathtt{m}]) \uparrow \mathtt{x} = \mathtt{m} + 1$$

The idea is that U_1 is a schema whose alphabet includes $\mathtt{m}$ and which contains a free schema variable X whose type is the type of the entire procedure.

And the rules.

Proposition 4.15 *Introduction:*

$$\frac{t.\mathtt{x} = 0 \vdash t \mathrel{\dot\in} U_0 \quad t.\mathtt{x} = t.\mathtt{m} + 1 \vdash t \mathrel{\dot\in} U_1(f(\mathtt{m}))}{t \mathrel{\dot\in} f}$$

Consider the following derivation:

Proof.

$$\dfrac{\dfrac{\dfrac{\overline{t.\mathtt{x} = 0}\ (1)}{\vdots}}{\dfrac{t \mathrel{\dot\in} U_0}{t \mathrel{\dot\in} U_0 \uparrow \mathtt{x} = 0}\ (1)} \quad \dfrac{\dfrac{\dfrac{\overline{t.\mathtt{x} = t.\mathtt{m} + 1}\ (2)}{\vdots}}{t \mathrel{\dot\in} U_1[f(\mathtt{m})]}}{\dfrac{t \mathrel{\dot\in} U_1[f(\mathtt{m})] \uparrow \mathtt{x} = \mathtt{m} + 1}{t \mathrel{\dot\in} \exists \mathtt{m} \bullet U_1[f(\mathtt{m})] \uparrow \mathtt{x} = \mathtt{m} + 1}\ (2)}}{\dfrac{t \mathrel{\dot\in} U_0 \uparrow \mathtt{x} = 0 \wedge \exists \mathtt{m} \bullet U_1[f(\mathtt{m})] \uparrow \mathtt{x} = \mathtt{m} + 1}{t \mathrel{\dot\in} f}\ (\mu^+)}$$

□

Proposition 4.16 *Elimination:*

$$\dfrac{t \mathrel{\dot\in} f \quad t.\mathtt{x} = 0}{t \mathrel{\dot\in} U_0} \qquad \dfrac{t \mathrel{\dot\in} f \quad t.\mathtt{x} = \mathtt{m} + 1}{t \mathrel{\dot\in} U_1(f(\mathtt{m}))}$$

In what follows, we write $U[E]$ for $U[\mathtt{x} \Rrightarrow E]$, when $\mathtt{x}$ is understood.

Proposition 4.17 *The following rule is derivable:*

$$\dfrac{n \in \mathbb{N} \vdash f(n) \sqsupseteq U[n]}{f \sqsupseteq U}$$

Proof. Consider the following derivation:

$$\dfrac{\dfrac{\dfrac{\overline{z \in f}\ (1) \quad \overline{z.\mathtt{x} = z.\mathtt{x}}}{z \in f(z.\mathtt{x})} \quad \dfrac{\overline{z.\mathtt{x} \in \mathbb{N}}}{\vdots}}{\dfrac{f(z.\mathtt{x}) \sqsupseteq U[z.\mathtt{x}]}{z \in U[z.\mathtt{x}]}}}{\dfrac{z \in U}{f \sqsupseteq U}\ (1)}$$

□

And now, the key rule for program development for recursive programming: the rule for recursive synthesis:

Proposition 4.18 *The following rule is derivable:*

$$\dfrac{U_0 \sqsupseteq U[0] \quad f(\mathtt{m}) \sqsupseteq U[\mathtt{m}] \vdash U_1(f(\mathtt{m})) \sqsupseteq U[\mathtt{m} + 1]}{f \sqsupseteq U}$$

Proof. Consider the following derivation:

$$\dfrac{\dfrac{U_0 \sqsupseteq U[0] \quad \dfrac{\dfrac{\dfrac{}{z \in f(0)}\,(2)}{z.\mathtt{x} = 0} \quad \dfrac{\dfrac{}{z \in f(0)}\,(2)}{z \in f}}{z \in U_0}}{\dfrac{z \in U[0]}{f(0) \sqsupseteq U[0]}\,(2)} \quad \begin{array}{c}\delta \\ \vdots \\ f(\mathtt{m}+1) \sqsupseteq U[\mathtt{m}+1]\end{array}}{\dfrac{f(n) \sqsupseteq U[n]}{f \sqsupseteq U}\,(0)}\,(1)$$

where δ is:

$$\dfrac{\dfrac{\dfrac{}{z \in f(\mathtt{m}+1)}\,(3)}{z.\mathtt{x} = \mathtt{m}+1} \quad \dfrac{\dfrac{}{z \in f(\mathtt{m}+1)}\,(3)}{z \in f}}{z \in U_1(f(\mathtt{m}))} \quad \begin{array}{c}\dfrac{}{f(\mathtt{m}) \sqsupseteq U[\mathtt{m}]}\,(0) \\ \vdots \\ U_1(f(\mathtt{m})) \sqsupseteq U[\mathtt{m}+1]\end{array}$$
$$\dfrac{z \in U[\mathtt{m}+1]}{f(\mathtt{m}+1) \sqsupseteq U[\mathtt{m}+1]}\,(3)$$

□

4.8 Primitive recursion over lists

The technique is easy to generalise. For example:

Definition 4.19

$$\mathtt{proc}\ f(\mathtt{x})\ \mathtt{cases\ x\ in\ Nil}\ :\ U_0;\ \mathtt{Cons}\ \mathtt{m}_0\ \mathtt{m}_1\ :\ U_1(f(\mathtt{m}_1))\ \mathtt{endcases} =_{df}$$
$$\mu X \bullet U_0 \uparrow \mathtt{x} = \mathtt{Nil} \wedge \exists\, \mathtt{m}_0, \mathtt{m}_1 \bullet U_1(X[\mathtt{x} \Rrightarrow \mathtt{m}_1]) \uparrow \mathtt{x} = \mathtt{Cons}\ \mathtt{m}_0\ \mathtt{m}_1$$

The rule for recursive synthesis over lists:

Proposition 4.20 *The following rule is derivable:*

$$\dfrac{U_0 \sqsupseteq U[\mathtt{Nil}] \quad f(\mathtt{m}_1) \sqsupseteq U[\mathtt{m}_1] \vdash U_1(f(\mathtt{m}_1)) \sqsupseteq U[\mathtt{Cons}\ \mathtt{m}_0\ \mathtt{m}_1]}{f \sqsupseteq U}$$

4.9 Primitive recursion over trees

Similarly for trees:

Definition 4.21

$$\texttt{proc}\ f(\texttt{x})\ \texttt{cases x in Leaf m}_0 : U_0;\ \texttt{Node m}_1\ \texttt{m}_2 : U_1(f(\texttt{m}))\ \texttt{endcases} =_{df}$$
$$\mu X \bullet \exists \texttt{m}_0 \bullet U_0 \uparrow \texttt{x} = \texttt{Leaf}\ m_0 \wedge \exists \texttt{m}_1, \texttt{m}_2 \bullet U_1(X[\texttt{x} \Rrightarrow \texttt{m}_1], X[\texttt{x} \Rrightarrow \texttt{m}_2]) \uparrow \texttt{x} = \texttt{Node m}_1\ \texttt{m}_2$$

The rule for recursive synthesis over trees:

Proposition 4.22 *The following rule is derivable:*

$$\frac{U_0 \sqsupseteq U[\texttt{Leaf m}_0] \quad f(\texttt{m}_1) \sqsupseteq U[\texttt{m}_1], f(\texttt{m}_2) \sqsupseteq U[\texttt{m}_2] \vdash U_1(f(\texttt{m}_1, \texttt{m}_2)) \sqsupseteq U[\texttt{Node m}_1\ \texttt{m}_2]}{f \sqsupseteq U}$$

4.10 Primitive recursion over arbitrary free-types

All these special cases can be generalised to syntax-directed *free types*.

Types of the form Υ are the names of the free types and are given by equations of the form:

$$\Upsilon\ ::=\ \cdots \mid\ c_i\ \langle\!\langle \cdots \Upsilon_{ij} \cdots \rangle\!\rangle \mid \cdots$$

The terms of free-type:

$$t^{\Upsilon}\ ::=\ c_i \cdots t^{\Upsilon_{ij}} \cdots$$

The logic of free types permits the introduction of values in the type, equality reasoning and finally, elimination (generally by induction).

Proposition 4.23

$$\frac{\cdots z_{ij} \in \Upsilon_{ij} \cdots}{c_i \cdots z_{ij} \cdots \in \Upsilon}\ (\Upsilon^+) \qquad \frac{\cdots z_{ij} \in \Upsilon_{ij} \cdots \quad \cdots z_{kl} \in \Upsilon_{kl} \cdots}{c_i \cdots z_{ij} \cdots \neq c_k \cdots z_{kl} \cdots}\ (\Upsilon_{\neq})$$

$$\frac{c_i \cdots z_{ij} \cdots = c_i \cdots y_{ij} \cdots}{z_{ij} = y_{ij}}\ (\Upsilon_{=})$$

$$\frac{\cdots \quad \cdots z_{ij} \in \Upsilon_{ij} \cdots, \cdots P[z/y_k] \cdots \vdash\ P[z/c_i \cdots z_{ij} \cdots] \quad \cdots}{z \in \Upsilon \vdash\ P}\ (\Upsilon^-)$$

where the y_k are all those variables occurring in the z_{ij} with type Υ.

Given a general free type Υ, the corresponding recursive program scheme is:

Definition 4.24

$$\mathtt{proc}_{\Upsilon}\ f(x)\ \mathtt{cases\ x\ in}\ \cdots\ H_i\ \cdots\ \mathtt{endcases}$$

where the H_i are the component cases:

$$H_i =_{df}\ c_i \cdots\ \mathtt{m}_i\ \cdots\ :\ U_i(\cdots f(\mathtt{w}_k), \cdots)$$

where the $\mathtt{w}_k$ are those observations among the $\mathtt{m}_i$ with type Υ.

The semantics in the general case is given by:

Definition 4.25

$$\mathtt{proc}_{\Upsilon}\ f(x)\ \mathtt{cases\ x\ in}\ \cdots\ H_i\ \cdots\ \mathtt{endcases} =_{df} \mu X \bullet \cdots\ \wedge K_i(X) \wedge \cdots$$

where:

$$K_i(X) =_{df} \exists \cdots \mathtt{m}_i \cdots \bullet\ U_i(\cdots X[\mathtt{x} \Rrightarrow \mathtt{w}_k] \cdots) \uparrow \mathtt{x} = c_i \cdots\ \mathtt{m}_i\ \cdots$$

5 Conclusions and further work

As we mentioned in the introduction, this expository paper concentrates entirely on the theoretical basis of $\nu\mathtt{Z}$. We have showed how an extremely simple logic can be extended towards an expressive specification logic and a program (development) logic. One of the benefits of this approach is its flexibility: one is not constrained by any particular specification or programming language infrastructure. The ability to provide elegant rules for total correctness development of procedures is also a strength: these rules resemble those which proved so useful in program development within constructive theories (see, for example, [5]) but are here combined with the ability to synthesize imperative programs.

Much infrastructural and pragmatic work remains to be done, both at the level of specification and program development. At the pragmatic level in particular, much work is being undertaken by Kajtazi and this will be reported in his PhD thesis.

6 Acknowledgements

The authors are grateful to Steve Reeves, Lindsay Groves and especially Moshe Deutsch for numerous discussions and critical appraisal of this work. We would like to acknowledge the support of the EPSRC RefineNet Network Grant (Ref: GR/S69979/02) in the development of this work, including comments from

participants at the January 2004 meeting in Sheffield where an earlier version of this material was presented.

References

[1] R. J. R. Back and J. von Wright. *Refinement Calculus: A Systematic Introduction*. Springer, 1998.

[2] W. P. de Roever and K. Engelhardt. *Data Refinement: Model-Oriented Proof Methods and Their Comparison*. Prentice Hall International, 1998.

[3] M. Deutsch, M. C. Henson, and S. Reeves. An analysis of total correctness refinement models for partial relation semantics I. *Logic Journal of the IGPL*, 11(3):287–317, 2003.

[4] M. C. Henson and S. Reeves. Investigating Z. *Logic and Computation*, 10(1):43–73, 2000.

[5] P. Martin-Löf. Constructive mathematics and computer programming. In *Logic, Methodology and Philosophy of Science VI*, pages 153–175. North Holland, 1982.

[6] C. C. Morgan. *Programming from Specifications*. Prentice Hall International, 2nd edition, 1994.