νZ – a wide-spectrum logic

RefineNet

10 January 2005

Martin Henson

Department of Computer Science

University of Essex

Some (of my personal) history ...

- *Program development in constructive set theories (–1997)*
  - $a : A$ – "Propositions as types" (Martin-Löf; Feferman)
- *Constructive Z (FMP 1998)*
  - Programs from (constructive) proofs
- *Z logics (FACJ, JLC, 1999–2000)*
  - $\mathcal{Z}_{\mathcal{C}}$ and $\mathcal{Z}_{\mathcal{C}}^{\perp}$ – Classical logics for Z
- *Classical "sets of implementations" (FACJ 2003)*
  - $f \Subset U$ – Classical, non-Z ...
- *Theories of refinement (JIGPAL 2003)*
  - total correctness refinement for partial relation semantics
- νZ*– wide-spectrum logic*
  - $U_0 \sqsupseteq U_1$ – Classical, non-Z, relational ...

What is νZ ...?

The framework νZ is a modification of the specification language Z. The differences are as follows:

- 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*.

What is νZ ...?

Core language of specifications:

$$
\begin{array}{lll}
U ::= & & \\
& X & \text{– schema variable} \\
& [T \mid P \mid Q] & \text{– atomic schemas} \\
& \neg U & \text{– negation schemas} \\
& U_0 \vee U_1 & \text{– disjunction schemas} \\
& \exists\, x : T \bullet U_0 & \text{– existential hiding schemas} \\
& \mu\, X \bullet U(X) & \text{– recursive schemas (positive } X \text{ only)}
\end{array}
$$

What is νZ ...?

The language of νZ is (at present) interpreted within $\mathcal{Z}_{\mathcal{C}}^{\perp}$, a conservative extension of $\mathcal{Z}_{\mathcal{C}}$, the Z logic developed by Steve Reeves and me between 1997 and 2000.

Example semantics:

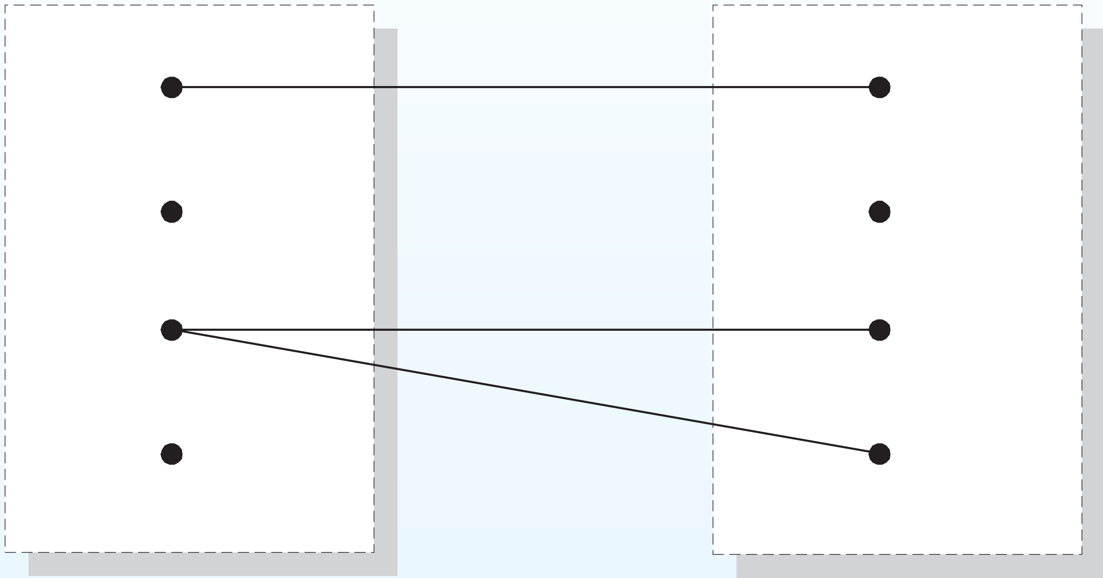
$$[\![U_0 \vee U_1]\!] =_{df} \{ z \in T^* \mid z \dot{\in} [\![U_0]\!] \vee z \dot{\in} [\![U_1]\!] \}$$

Z *versus* νZ semantics ...

An operation schema:

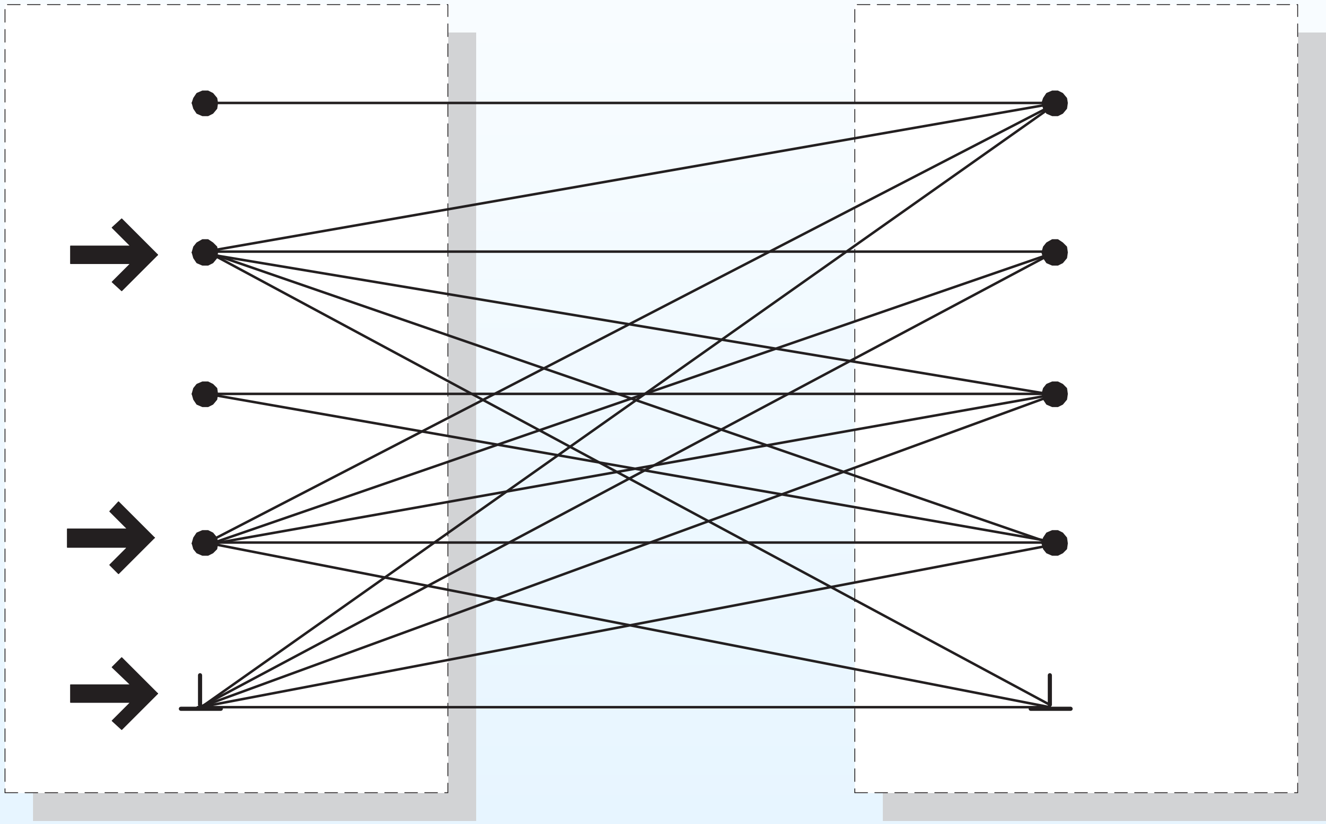
$$
\begin{array}{|l}
\multicolumn{1}{l}{U} \\
\hline
\mathsf{x}, \mathsf{x}' \in \{0, 1, 2, 3\} \\
\hline
(\mathsf{x} = 0 \wedge \mathsf{x}' = 0) \vee \\
(\mathsf{x} = 2 \wedge \mathsf{x}' = 2) \vee \\
(\mathsf{x} = 2 \wedge \mathsf{x}' = 3) \\
\hline
\end{array}
$$

Z semantics ...



The dots on the left correspond to the possible input states, written $\langle\!|\ x \Rrightarrow n\ |\!\rangle$ for the various of n. The dots on the right to the possible output states, written $\langle\!|\ x' \Rrightarrow n\ |\!\rangle$.

νZ semantics ...



... the *lifted-totalised completion* of the original relation.

What is a (wide-spectrum) logic ...?

Everything is characterised by introduction and elimination rules.
For example, atomic operation schemas:

$$\frac{z_0.P \vdash z_0.z_1'.Q}{z_0 \star z_1' \in [T \mid P \mid Q]} \ (U^+) \qquad \frac{z_0 \star z_1' \in [T \mid P \mid Q] \quad z_0.P}{z_0.z_1'.Q} \ (U^-)$$

For example, refinement:

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

For example, schema hiding:

$$\frac{z \in U}{z \mathrel{\dot{\in}} \exists\, \mathsf{x} : V \bullet U} \ (U_{\exists}^{+}) \qquad \frac{z \in \exists\, \mathsf{x} : V \bullet U \quad z \star \langle\!|\, \mathsf{x} \Rrightarrow y \,|\!\rangle \in U \vdash P}{P} \ (U_{\exists}^{-})$$

What is νZ ...?

Note that negation in νZ 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. An alternative characterisation of the semantics is available using a combination of relational inverse, disjunction and lifting.

$$\mathit{abort} = \{z_0 \star z_1' \in T^* \mid z_0 = \perp\}$$

Then:

$$\neg U =_{\mathit{df}} U^{-1} \vee \mathit{abort}$$

The rules for negation are:

$$\frac{t \notin U}{t \in \neg U} \qquad \frac{t_0 = \perp}{t_0 \star t_1' \in \neg U} \qquad \frac{t_0 \star t_1' \in \neg U \quad t_0 \star t_1' \notin U \vdash P \quad t_0 = \perp \ \vdash P}{P}$$

What is νZ ...?

Properties of negation include:

Double negation: $U = \neg\neg U$

Excluded middle: $chaos = U \vee \neg U$.

Operator definitions are used to extend the core language:

$$\mathtt{OP}(\cdots X_i \cdots) =_{df} U(\cdots X_i \cdots)$$

Specification in νZ ...

First specifications are to provide new ways to specify ...

We can define conjunction in terms of disjunction and negation, using the usual de Morgan definition:

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

The usual rules are derivable.

$$\frac{z \in U_0 \wedge U_1}{z \mathrel{\dot{\in}} U_i} \qquad \frac{z \mathrel{\dot{\in}} U_0 \quad z \mathrel{\dot{\in}} U_1}{z \in U_0 \wedge U_1}$$

Specification in νZ ...

Schema implication can be defined in the standard way.

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

With the rules:

$$\frac{z \mathrel{\dot{\in}} U_0 \vdash z \mathrel{\dot{\in}} U_1}{z \in U_0 \Rightarrow U_1} \qquad \frac{z \in U_0 \Rightarrow U_1 \quad z \mathrel{\dot{\in}} U_0}{z \mathrel{\dot{\in}} U_1}$$

Specification in νZ ...

Universal hiding can be specified using existential hiding and negation:

$$\forall \mathsf{x} : T \bullet U =_{df} \neg \exists \mathsf{x} : T \bullet \neg U$$

With the rules:

$$\frac{t \star \langle\!| \mathsf{x} \Rrightarrow z |\!\rangle \in U}{t \in \forall \mathsf{x} : T \bullet U} \qquad \frac{t \in \forall \mathsf{x} : T \bullet U \quad v \in T}{t \star \langle\!| \mathsf{x} \Rrightarrow v |\!\rangle \in U}$$

Specification in νZ ...

Further Examples:

$$
\begin{array}{lll l}
\mathit{abort} & =_{df} & [T \mid \mathit{false} \mid \mathit{false}] & \text{– abort} \\
\mathit{chaos} & =_{df} & [T \mid \mathit{false} \mid \mathit{true}] & \text{– chaos} \\
\mathit{chaos}_P & =_{df} & [T \mid \neg P \mid \mathit{false}] & \text{– } P\text{–chaos} \\
U[\mathrm{x} \Rrightarrow E] & =_{df} & \mathit{chaos}_{\mathtt{x}=\mathtt{E}} \wedge U & \text{– schema specialisation} \\
U \uparrow P & =_{df} & \mathit{chaos}_P \Rightarrow U & \text{– strengthened preconditions} \\
\Xi U & =_{df} & [\Delta U \mid \mathit{true} \mid \theta U = \theta' U] & \text{– } \Xi\text{-schemas} \\
U \diamond T & =_{df} & U \wedge \Xi T & \text{– } \mathtt{skip}\text{–extension}
\end{array}
$$

Specification in νZ ...

Composition can be specified using a modification of the standard approach:

$$U_0 \fatsemi U_1 =_{df} \exists \overline{\mathtt{t}} \bullet (U_0 \diamond T_L)[\alpha_0/\overline{\mathtt{t}}] \wedge (U_1 \diamond T_R)[\alpha_1/\overline{\mathtt{t}}]$$

With the rules:

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

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

Specification in νZ ...

Can also specify a programming language ...

(i) $\texttt{skip} =_{df} [\Delta T \mid true \mid \theta T = \theta' T]$

(ii) $\texttt{x:=}\, E =_{df} [x, x' : T \mid true \mid x' = E] \diamond T_E$

(iii) $\texttt{if}\ D\ \texttt{then}\ X_0\ \texttt{else}\ X_1 =_{df} X_0 \upharpoonright D \wedge X_1 \upharpoonright \neg D$

(iv) $\texttt{begin var x;}X\ \texttt{end} =_{df} \exists \mathtt{x}, \mathtt{x}' \bullet X$

(v) $\texttt{proc}\ f(\mathtt{x})\ \texttt{cases x in}\ 0 : X_0;\ \mathtt{m}+1 : X_1(f(\mathtt{m}))\ \texttt{endcases} =_{df}$
$\forall \mathtt{x} : \mathbb{N} \bullet \mu X \bullet X_0 \upharpoonright \mathtt{x} = 0 \wedge \exists \mathtt{m} \bullet X_1(X[\mathtt{x} \Rrightarrow \mathtt{m}]) \upharpoonright \mathtt{x} = \mathtt{m}+1$

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

Program logic ...

Simple example, conditionals:

$$\texttt{if}\ D\ \texttt{then}\ U_0\ \texttt{else}\ U_1 =_{df} U_0 \upharpoonright D \wedge U_1 \upharpoonright \neg D$$

Rules:

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

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

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

Inequational (refinement) logic ...

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

Proof:

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

Refinement logic ...

Preconditions and postconditions:

$$\frac{P_1 \vdash P_0}{[T \mid P_0 \mid Q] \sqsupseteq [T \mid P_1 \mid Q]} \ (\sqsupseteq^+_{pre}) \qquad \frac{Q_0 \vdash Q_1}{[T \mid P \mid Q_0] \sqsupseteq [T \mid P \mid Q_1]} \ (\sqsupseteq^+_{post})$$

Composition:

$$\frac{P_2 \vdash P_0 \wedge \forall \overline{v} \bullet Q_0[\overline{x'}/\overline{v}] \Rightarrow P_1[\overline{x'}/\overline{v}] \qquad \exists \overline{u} \bullet Q_0[\overline{x'}/\overline{v}] \wedge Q_1[\overline{x'}/\overline{v}] \vdash Q_2}{[T_0 \mid P_0 \mid Q_0] \mathbin{;} [T_1 \mid P_1 \mid Q_1] \sqsupseteq [T_0 \curlyvee T_1 \mid P_2 \mid Q_2]}$$

Conjunction:

$$\frac{P_2 \vdash P_0 \wedge P_1 \qquad Q_0 \vee Q_1 \vdash Q_2}{[T_0 \mid P_0 \mid Q_0] \wedge [T_1 \mid P_1 \mid Q_1] \sqsupseteq [T_0 \curlyvee T_1 \mid P_2 \mid Q_2]} \ (\sqsupseteq_\wedge)$$

Refinement logic ...

Completely general transformation of conjunction to composition:

$$\frac{P_2 \vee P_3 \vdash P_0 \wedge \forall \overline{v} \bullet Q_0[\overline{x'}/\overline{v}] \Rightarrow P_1[\overline{x'}/\overline{v}] \qquad \exists \overline{v} \bullet Q_0[\overline{x'}/\overline{v}] \wedge Q_1[\overline{x'}/\overline{v}] \vdash Q_2 \wedge Q_3}{[T_0 \mid P_0 \mid Q_0] \fatsemi [T_1 \mid P_1 \mid Q_1] \sqsupseteq [T_2 \mid P_2 \mid Q_2] \wedge [T_3 \mid P_3 \mid Q_3]}$$

Monotone inductive schemas ...

The $\mu X \bullet U(X)$ are *recursive schemas*.
The schema algebra is *monotonic* (but not ω-continuous). Thus:

$$[\![\mu X \bullet U(X)]\!] =_{df} \bigsqcap \{X \in W \mid [\![U(X)]\!] \sqsupseteq X\}$$

satisfies:

$$\mu X \bullet U(X) = U(\mu X \bullet U(X)) \qquad \textbf{(}\mu\textbf{)}$$

It can be characterised as follows:

$$\bigsqcup_{i<\kappa} U^i(chaos) = \mu X \bullet U(X)$$

for some suitably f**k-off-huge ordinal κ.

Primitive recursive procedures ...

It takes 6 schema operators to specify primitive recursion over the natural numbers:

$$\mathtt{proc}\ f(\mathtt{x})\ \mathtt{cases}\ \mathtt{x}\ \mathtt{in}\ 0 : X_0;\ \mathtt{m}+1 : X_1(f(\mathtt{m}))\ \mathtt{endcases} =_{df}$$
$$\forall\, \mathtt{x} : \mathbb{N} \bullet \mu\, X \bullet X_0 \uparrow \mathtt{x} = 0 \wedge \exists\, \mathtt{m} \bullet X_1(X[\mathtt{x} \Rrightarrow \mathtt{m}]) \uparrow \mathtt{x} = \mathtt{m}+1$$

Introduction rule (proof requires μ):

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

Elimination rules:

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

Primitive recursive procedures ...

The rules for $\mathbb{N}$ are as follows:

$$\frac{}{0 \in \mathbb{N}}\ (\mathbb{N}_0^+) \qquad \frac{n \in \mathbb{N}}{n+1 \in \mathbb{N}}\ (\mathbb{N}_1^+) \qquad \frac{n \in \mathbb{N}}{0 \neq n+1}$$

$$\frac{n+1 = m+1}{n = m} \qquad \frac{P(0) \quad m \in \mathbb{N}, P(m) \vdash P(m+1)}{n \in \mathbb{N} \vdash P(n)}\ (\mathbb{N}^-)$$

Primitive recursive procedures ...

Characterising a procedure in terms of (all) its invocations:

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

Proof:

$$\dfrac{\dfrac{\dfrac{\dfrac{}{z \in f}\,(1) \qquad \dfrac{}{z.X = z.X}}{z \in f(z.X)} \qquad \begin{array}{c} \dfrac{}{z.X \in \mathbb{N}} \\ \vdots \\ f(z.X) \sqsupseteq U[z.X] \end{array}}{z \in U[z.X]}}{\dfrac{z \in U}{f \sqsupseteq U}\,(1)}$$

Primitive recursive procedures ...

The rule for *recursive synthesis*:

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

Proof:

$$\dfrac{\dfrac{\begin{array}{c}\vdots \\ f(0) \sqsupseteq U[0]\end{array} \quad \begin{array}{c}\overline{f(\mathtt{m}) \sqsupseteq U[\mathtt{m}]} \\ \vdots \\ f(\mathtt{m}+1) \sqsupseteq U[\mathtt{m}+1]\end{array}}{f(n) \sqsupseteq U[n]}\ \mathbb{N}^-}{f \sqsupseteq U}\ (inv_{\mathbb{N}})$$

Primitive recursive procedures ...

Base case:

$$\dfrac{\dfrac{U_0 \sqsupseteq U[0] \qquad \dfrac{\dfrac{\dfrac{}{z \in f(0)}\,(2)}{z.\mathsf{x} = 0} \qquad \dfrac{\dfrac{}{z \in f(0)}\,(2)}{z \in f}}{z \in U_0}}{z \in U[0]}}{f(0) \sqsupseteq U[0]}\,(2)$$

Induction case:

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

Other types ...

Lists:

$$\begin{array}{lll} \texttt{proc} & f(\texttt{x})\ \texttt{cases x in} & \\ & \texttt{Nil} & : U_0; \\ & \texttt{Cons}\ \texttt{m}_0\ \texttt{m}_1 & : U_1(f(\texttt{m}_1))\ \texttt{endcases} =_{df} \end{array}$$

$$\forall\,\texttt{x} : \mathit{List} \bullet \mu X \bullet \quad \begin{array}{l} U_0 \uparrow \texttt{x} = \texttt{Nil} \wedge \\ \exists\,\texttt{m}_0, \texttt{m}_1 \bullet U_1(X[\texttt{x} \Rrightarrow \texttt{m}_1]) \uparrow \texttt{x} = \texttt{Cons}\ \texttt{m}_0\ \texttt{m}_1 \end{array}$$

Other types ...

Trees:

$$\begin{array}{lll} \texttt{proc} & f(\texttt{x})\ \texttt{cases}\ \texttt{x}\ \texttt{in} & \\ & \texttt{Leaf}\ \texttt{m}_0 & : U_0; \\ & \texttt{Node}\ \texttt{m}_1\ \texttt{m}_2 & : U_1(f(\texttt{m}_1), f(\texttt{m}_2))\ \texttt{endcases} =_{df} \end{array}$$

$$\begin{array}{ll} \forall\, \texttt{x} : \mathit{Tree} \bullet \mu X \bullet & \exists\, \texttt{m}_0 \bullet U_0 \uparrow \texttt{x} = \texttt{Leaf}\ \texttt{m}_0 \wedge \\ & \exists\, \texttt{m}_1, \texttt{m}_2 \bullet U_1(X[\texttt{x} \Rrightarrow \texttt{m}_1], X[\texttt{x} \Rrightarrow \texttt{m}_2]) \uparrow \texttt{x} = \texttt{Node}\ \texttt{m}_1\ \texttt{m}_2 \end{array}$$

The “frame problem” ...

What does a specification say about behaviour outside the precondition, and outside the frame?

Dec

$$\mathsf{x}, \mathsf{x}' : \mathbb{N}$$

$$\mathsf{x} > 0$$

$$\mathsf{x}' = \mathsf{x} - 1$$

Solutions:

– Refinement calculus: `skip` outside the frame.
– Henson-Reeves FACJ: *Chaos* outside the frame.
– νZ: *silent* outside the frame.

A worked example

The Fibonacci numbers are, as usual, specified as follows:

$$fib \in \mathbb{N} \to \mathbb{N}$$

$$fib(0) = 1$$
$$fib(1) = 1$$
$$fib(n+2) = fib(n+1) + fib(n)$$

We shall take subtraction, over the natural numbers, to satisfy:

$$n - m = 0 \text{ whenever } m > n$$

A worked example

The initial specification is:

$$\begin{array}{|l}
\multicolumn{1}{l}{\textit{Fib}} \\ \hline
y, y' \in \mathbb{N} \\
z? \in \mathbb{N} \\ \hline
y' = \textit{fib}(z?) \\ \hline
\end{array}$$

First Stage - frame expansion

The following variation *expands the frame* to include a new observation, and *strengthens the postcondition*. Both these transformations are *refinements* of the original specification.

ExFib

$$x, x', y, y' \in \mathbb{N}$$
$$z? \in \mathbb{N}$$

$$y' = \mathit{fib}(z?)$$
$$x' = \mathit{fib}(z? - 1)$$

First Stage - frame expansion

We have the *refinement*:

$$\mathit{ExFib} \sqsupseteq \mathit{Fib}$$

In particular: *every implementation of ExFib is an implementation of Fib*.

Second stage - schema valued functions

We specialise with respect to the input observation $z?$.

$ExFib[z? \Rrightarrow n] =$

$$\begin{array}{|l}
\hline
x, x', y, y' \in \mathbb{N} \\
\hline
y' = fib(n) \\
x' = fib(n-1) \\
\hline
\end{array}$$

We are preparing to derive a simply recursive procedure over the input.

Second stage – continued

- If $c_0 \sqsupseteq U[z? \Rrightarrow 0]$
- and $c_1 \sqsupseteq U[z? \Rrightarrow m+1]$, assuming that $p[m] \sqsupseteq U[z? \Rrightarrow m]$
- then:

$$
\begin{array}{ll}
\texttt{proc}\ p[z?] & \texttt{cases}\ z?\ \texttt{in} \\
& \quad 0: \quad\ c_0, \\
& \quad m+1: c_1 \\
& \texttt{endcases} \\
\sqsupseteq U &
\end{array}
$$

Third stage – simplification

$U[z? \Rrightarrow 0]$ simplifies to:

$$\begin{array}{|l}\hline x, x', y, y' \in \mathbb{N} \\ \hline y' = 1 \wedge x' = 1 \\ \hline\end{array}$$

and this can be refined to the simultaneous assignment:

$$x, y := 1, 1$$

Third stage – simplification

$U[z? \Rrightarrow m + 1]$ simplifies to:

$$\begin{array}{|l}
x, x', y, y' \in \mathbb{N} \\
\hline
y' = fib(m+1) \\
x' = fib(m)
\end{array}$$

Fourth stage – conjunction

We can express $U[z?\Rrightarrow m+1]$ as a *conjunction* by splitting the precondition:

$$U_0[m] \wedge U_1[m] \sqsupseteq U[z?\Rrightarrow m+1]$$

Simplifying, we get:

$$\begin{array}{|l}
\hline
U_0[m] \\
x, x', y, y' \in \mathbb{N} \\
\hline
m = 0 \\
y' = 1 \\
x' = 1 \\
\hline
\end{array}$$

Fourth stage – conjunction

$$\begin{array}{|l}
\hline
U_0[m] \\
x, x', y, y' \in \mathbb{N} \\
\hline
m = 0 \\
y' = 1 \\
x' = 1 \\
\hline
\end{array}$$

can, after *weakening the precondition* be refined to the simultaneous assignment:

$$x, y := 1, 1$$

Fifth stage – composition

$$
\begin{array}{|l}
\multicolumn{1}{l}{U_1[m]} \\
\hline
x, x', y, y' \in \mathbb{N} \\
\hline
\exists n \in \mathbb{N} \bullet m = n + 1 \\
y' = \mathit{fib}(m+1) \\
x' = \mathit{fib}(m) \\
\hline
\end{array}
$$

A little analysis in the *inequational logic* (refinement rules) permits us to express this as a *composition*:

$$U_1[m] =_{df} U_3[m] \fatsemi \mathit{Step}$$

Fifth stage – continued

$$\begin{array}{|l}
\multicolumn{1}{l}{U_3[m]} \\ \hline
x, x', y, y' \in \mathbb{N} \\ \hline
\exists\, n \bullet m = n + 1 \\
y' = \mathit{fib}(m) \\
x' = \mathit{fib}(m - 1) \\ \hline
\end{array}$$

By weakening the precondition:

$$U[z? \Rrightarrow m] \sqsupseteq U_3[m]$$

and this can be refined to the *recursive call*:

$$p[m]$$

Fifth stage – continued

Step

$$x, x', y, y' \in \mathbb{N}$$

$$x' = y$$

$$y' = x + y$$

This can be refined to a simultaneous assignment:

$$x, y := y, x + y$$

The derived program

$$
\begin{array}{ll}
\texttt{proc}\; p[z?] & \texttt{cases}\; z?\; \texttt{in} \\
 & \quad 0: \qquad x, y := 1, 1 \\
 & \quad m+1: \; \texttt{if}\; m == 0 \\
 & \qquad\qquad\quad \texttt{then}\; x, y := 1, 1 \\
 & \qquad\qquad\quad \texttt{else}\; p[m];\; x, y := y, x+y \\
 & \texttt{endcases}
\end{array}
$$

and this is a refinement of the original specification *Fib*.

Structuring derivations

Consider the following specification:

$$\begin{array}{|l} \textit{Inc} \\ \hline n, n' \in \mathbb{N} \\ \hline n' = n + 1 \\ \hline \end{array}$$

We wish to consider this as a *local operation* over the *local state* $\mathbb{N}$. It is trivially implemented by:

$$n := n + 1$$

Promotion ...

In the global state we have two numbers, represented by the cartesian product $\mathbb{N} \times \mathbb{N}$. The global operation simply generalises the local operation by specifying *which* of the two values is to be altered. The *promotion schema* explains *how* the local and global state spaces are to be connected.

Promote

$$n, n' \in \mathbb{N}$$
$$p, p' \in \mathbb{N} \times \mathbb{N}$$
$$z? \in \{0, 1\}$$

$$(z? = 0 \land p.1 = n \land p'.1 = n' \land p'.2 = p.2) \lor$$
$$(z? = 1 \land p.2 = n \land p'.2 = n' \land p'.1 = p.1)$$

Derivation ...

Finally, the global operation is defined by hiding the local state changes:

$$GlobalInc =_{df} \exists\, n, n' \in \mathbb{N} \bullet Inc \wedge Promote$$

The first step in the derivation is to abstract with respect to the input observation $z?$ and then to expand the conjunction by splitting the precondition of the promotion schema. This leads to:

$$\exists\, n, n' \in \mathbb{N} \bullet (Inc \wedge P_0[m]) \vee (Inc \wedge P_1[m])$$

Derivation ...

Where, for example:

$$
\begin{array}{|l}
\hline
P_0[m] \\
n, n' \in \mathbb{N} \\
p, p' \in \mathbb{N} \times \mathbb{N} \\
\hline
m = 0 \\
p.1 = n \\
p'.1 = n' \\
p'.2 = p.2 \\
\hline
\end{array}
$$

Refinement – continued

We can refine $\mathit{Inc} \wedge P_0[m]$ to:

$$
\begin{array}{|l}
n, n' \in \mathbb{N} \\
p, p' \in \mathbb{N} \times \mathbb{N} \\
\hline
m = 0 \\
p.1 = n \\
n' = n + 1 \\
p'.1 = n' \\
p'.2 = p.2
\end{array}
$$

Refinement – continued

We can then express the schema as a composition of:

$$
\begin{array}{|l}
\hline
n, n' \in \mathbb{N} \\
p, p' \in \mathbb{N} \times \mathbb{N} \\
\hline
n' = p.1 + 1 \\
\hline
\end{array}
$$

with:

$$
\begin{array}{|l}
\hline
n, n' \in \mathbb{N} \\
p, p' \in \mathbb{N} \times \mathbb{N} \\
\hline
p'.1 = n \\
p'.2 = p.2 \\
\hline
\end{array}
$$

Refinement – continued

The former can be refined to a composition of:

$$n, n' \in \mathbb{N}$$
$$p, p' \in \mathbb{N} \times \mathbb{N}$$

$$n' = p.1$$

followed by:

Inc

$$n, n' \in \mathbb{N}$$

$$n' = n + 1$$

Refinement

These are refined to:

$n := p.1$ and the local operation $n := n + 1$

and:

$$\begin{array}{|l}
n, n' \in \mathbb{N} \\
p, p' \in \mathbb{N} \times \mathbb{N} \\
\hline
p'.1 = n \\
p'.2 = p.2
\end{array}$$

can be refined to an assignment:

$$p.1 := n$$

Refinement

The assignments now sequence to implement the relevant composed specifications and the split precondition leads to a conditional. The quantifed program can be refined into a block; and the entire specification of *GlobalInc* into a procedure:
In summary we have the following program:

$$
\begin{array}{l}
\texttt{proc}\ \mathit{globalinc}[z?] \\
\quad \texttt{begin}\ \texttt{var}\ n; \\
\qquad \texttt{if}\ \texttt{z}?\ \texttt{then}\ n := p.1;\ n := n + 1;\ p.1 := n \\
\qquad\qquad\quad \texttt{else}\ n := p.2;\ n := n + 1;\ p.2 := n \\
\quad \texttt{end}
\end{array}
$$

Conclusions ...

- νZ is very small and easy to understand.
- νZ is very adaptable for creating a more comprehensive specification language.
- νZ is very adaptable for integrating a programming language.
- νZ is completely formal.
- νZ is designed for reasoning about specifications.
- νZ is designed for reasoning about programs.
- νZ is designed for deriving programs from specifications.