Refinement with Approximations

John Derrick and Eerke Boiten

Ex: Specify a Garbage Collector

"Collects all dead cells immediately"
– *too often*

"Collects some dead cells, periodically"
– *possibly none, ever*

"Collects ≥ 17 dead cells every ≤ 0.2 seconds"
– *not abstract*

Solution

"Collects n dead cells every $1/m$ seconds" for $n, m \to \infty$.

Refine, retaining n and m

Compromise: pick specific values of n and m

Late compromise – why?

Example 1 - bounded buffer

Buffer

$$cont : \mathrm{seq}\,\mathbb{Z}$$

EmptyBuf

$$Buffer'$$
$$cont' = \langle\,\rangle$$

Remove

$$\Delta Buffer$$
$$out! : \mathbb{Z}$$
$$cont = \langle out! \rangle \frown cont'$$

$Insert_\infty$

$$\Delta Buffer$$
$$in? : \mathbb{Z}$$
$$cont' = cont \frown \langle in? \rangle$$

Implement by a bounded buffer of a particular size, e.g., $n = 256$. $Insert_\infty$ is replaced by $Insert_n$:

$Insert_n$

$$\Delta Buffer$$
$$in? : \mathbb{Z}$$
$$(\#cont < n \land cont' = cont \frown \langle in? \rangle)$$
$$\lor (\#cont >= n \land cont' = cont)$$

Example 2 - add

Start with Add_∞, replace it by Add_n, refine that to $ModAdd_n$ and finally instantiate n to `maxint`.

Add_∞

$$\Delta\left[x : \mathbb{N}\right];\ add? : \mathbb{N}$$

$$x' = x + add?$$

Add_n

$$\Delta\left[x : \mathbb{N}\right];\ add? : \mathbb{N}$$

$$x' = x + add? \land x' < n$$

$ModAdd_n$

$$\Delta\left[x : \mathbb{Z} \mid -n \leq x < n\right]$$

$$add? : \mathbb{N}$$

$$x' \bmod (2 * n) = (x + add?) \bmod (2 * n)$$

Central idea

If $\lim_{n\to\infty} S_n \equiv S$, then equate $(S_n)_{n\in\mathbb{N}}$ with S.

Define metrics and limits with respect to different criteria.

Development Process - Four kinds of steps:

Element-wise refine
apply refinement uniformly to each of (S_n)

Introduce sequence
replace S with (S_n) such that S is the limit of (S_n).

Replace sequence
replace (S_n) by (T_n) if both have same limit S.

Compromise
replace a sequence (S_n) with one of its elements S_n

Example 1

Initial spec Add_∞

(Add_n) (sequence introduction),

$(ModAdd_n)$ (element-wise refinement) where

$$
\begin{array}{|l}
\hline
ModAdd_n \\
\Delta \left[x : \mathbb{Z} \;\middle|\; -n \leq x < n \right] \\
add? : \mathbb{N} \\
\hline
x' \bmod (2 * n) = (x + add?) \bmod (2 * n) \\
\hline
\end{array}
$$

instantiate n to `maxint` − compromise

Example 2 - unbounded buffer

Initial spec Buf,

replace by (Buf_n) (sequence introduction)

replace by $(Buf_{2*n})_{n\in\mathbb{N}}$ (sequence replacement, same limit)

composition of two copies of Buf_n (element-wise refinement)

instantiate n to 256 (compromise).

Soundness etc

Theorem
The sequence (S_n) with limit S_∞ is refined by the sequence (T_n) with limit T_∞ iff
S_∞ is refined by T_∞.

(by definition)

But: element-wise refinement does *not* necessarily lead to refinement between sequences.

(T_n) must have a limit . . .

Metrics and limits

Idea: define a distance between the limit and each sequence element (ie approximation).

A metric on A is a function $d : A \times A \to \mathbb{R}$ such that $\forall\, x, y, z \in A$:

$d(x, y) \geq 0$
$d(x, y) = d(y, x)$
$d(x, y) = 0$ iff $x = y$
$d(x, y) \leq d(x, z) + d(z, y)$

The limit of a sequence is the defined as the point of convergence with respect to a metric.

A sequence s_n converges to s, denoted $s_n \to s$, whenever:

$$\forall\, \epsilon > 0 \;\; \exists\, N \;\; \forall\, n > N \;\; d(s_n, s) \leq \epsilon$$

Program length

Data refinement asks for consistency of observations for all programs.

Define a metric by assigning a distance to specifications which agree on observations up to a certain length.

Define the metric d_l by

$$d_l(A, C) = \begin{cases} 0 & \text{if } A =_{data} C \\ 2^{-n} & \text{if } n = min\{m : \mathbb{N} \mid \mathsf{p_C} \neq \mathsf{p_A} \\ & \qquad \wedge \#\mathsf{p} = \mathsf{m}\} \end{cases}$$

where the length of the program is the number of operations plus one (for the initialisation).

Two specifications are close if it takes a long time to tell them apart, where a 'long time' is length of program before the difference is observed.

Buffer example

The shortest program that can observe that Buf_n is different from Buf has size $2n+2$:

first $n+1$ elements are inserted (the last of which is the first one to be ignored),

then n *Remove* operations are successful, and the next *Remove* operation fails.

Thus $d_I(Buf_n, Buf) = 2^{-(2n+3)}$ and so $Buf_n \to Buf$.

If we change the finite buffer specification, we get a different distance,

but still convergence to the unbounded buffer.

For example, if we insert and remove from the same end then we can observe the difference quicker since we don't have to remove all the elements first.

The distance between the two buffers is then $2^{-(n+1)}$, and thus tends to zero.

Add example

Need observer:

Obs

$out! : \mathbb{N}$

$out! = x$

Then $d_I(A_n, A_\infty) = 1/2^3$ since the sequence *Init*; *Add*; *Obs* will observe a difference for any input bigger than n.

So, with respect to this metric, we don't get convergence.

This is despite the sequence $(Add_n)_n$ being ordered by refinement:

$$\text{pre}\, Add_n \Rightarrow \text{pre}\, Add_{n+1}$$

and

$$\text{pre}\, Add_n \wedge Add_{n+1} \equiv Add_n$$

Although $d_I(A_n, A_\infty) = 1/2^3$, the behaviour is correct for some inputs.

This metric stresses quantification over programs at the expense of quantification over inputs/outputs.

Consider, for example,

Op_N

$\Delta [x : \mathbb{N}]$

$x? : \mathbb{N}$

$y! : \mathbb{B}$

$x? \neq N \Leftrightarrow y!$

Op

$\Delta [x : \mathbb{N}]$

$x? : \mathbb{N}$

$y! : \mathbb{B}$

$y!$

Then $d_I(Op_N, Op) = 1/4$ despite the fact that Op and Op_N have identical behaviour for all but one input.

Input/output metrics - Bounded data types

Add_n

$$\Delta[x : 0..m]$$
$$add? : 0..m$$
$$x' = add? \land x' < n$$

Add_∞

$$\Delta[x : 0..m]$$
$$add? : 0..m$$
$$x' = add?$$

Obs

$$y! : \mathbb{N}$$
$$y! = x$$

We want to define a metric which counts the values for which $(Init, Add_n, Obs)$ differs from $(Init, Add_\infty, Obs)$.

This time will base our metric on the simulation rules

Not sufficient to just count the outputs: Obs in the concrete is clearly a correct refinement of Obs in the abstract.

Thus will need to consider both inputs and outputs - and thus both applicability and correctness.

Define d in terms of the maximum distance between the constituent operations:

$$d(A, C) = max_{i \in I} d(AOp_i, COp_i)$$

and the distance between two operations in terms of an asymmetrical distance based on applicability and correctness:

$$d(AOp, COp) = max\{\rho(AOp, COp), \rho(COp, AOp)\}$$
$$\rho(AOp, COp) = \rho_a(AOp, COp) + \rho_c(AOp, COp)$$

ρ_a will measure distance in preconditions, and ρ_c distance in correctness.

Both will count values where failure occurs, and return the ratio of this to the size of the input/output domain as the distance.

Let $x? : Y$, and $y! : Z$.

Define $\rho_a(AOp, COp) = \frac{(\#Y - \#T)}{\#Y}$

where T is the largest set $T \subseteq Y$ such that

$$\forall x? : T \bullet \forall State \bullet \mathrm{pre}\, AOp \Rightarrow \mathrm{pre}\, COp$$

Define $\rho_c(AOp, COp) = \frac{(\#Z - \#T)}{\#Z}$

where T is the largest set $T \subseteq Z$ such that

$$\forall y! : T \bullet \forall State;\ State';\ x? : Y \bullet \\ \mathrm{pre}\, AOp \wedge COp \Rightarrow AOp$$

Example - Add

Distance between $A_n = (Init, Add_n, Obs)$ and $A_\infty = (Init, Add_\infty, Obs)$:

$$
\begin{aligned}
& d(A_n, A_\infty) = d(Add_n, Add_\infty) \\
& = \rho(Add_\infty, Add_n) \\
& = \rho_a(Add_\infty, Add_n) \\
& = (m - \#T)/m
\end{aligned}
$$

where T is the largest $T \subseteq 0..m$ for which $add? : T \bullet add? < n$, hence

$$
d(A_n, A) = \begin{cases} (m - (n-1))/m & \text{if } n < m \\ 0 & \text{otherwise} \end{cases}
$$

and so $d(A_n, A) \to 0$.

Proposition 1 *d is a metric on the set of equivalence classes (with respect to refinement) of specifications.*

Proposition 2 *If* $S_i \sqsubseteq S_{i+1}$ *and S is the least upper bound in the refinement ordering (i.e.,* $S_i \sqsubseteq S$ *and no other* $S_i \sqsubseteq T \sqsubseteq S$ *with* $T \neq_{data} S$*), then* $d(S_i, S) \to 0$.

Non-finite state

$State_1$

$x : 0..m$

$State_2$

$x : \mathbb{N}$

$Add_n[S]$

ΔS

$add? : 0..m$

$x' = x + add? \land x' < n$

$Add_\infty[S]$

ΔS

$add? : 0..m$

$x' = x + add?$

Then $d(Add_n[State_1], Add_\infty[State_1]) = 1 - \#T/m$, where

$$T = \begin{cases} \varnothing & \text{if } n \leq m \\ 0..m & \text{if } n > 2m \\ 0..(n-m) & \text{if } m < n \leq 2m \end{cases}$$

so $d(Add_n[State_1], Add_\infty[State_1]) \to 0$.

And $d(Add_n[State_2], Add_\infty[State_2]) = 1 - \#T/m$, but $T = \varnothing$ since for no input values can we guarantee that $x' < n$ for all state.

This sequence does not converge, since we can never force Add_n to behave like Add_∞ no matter what input values are chosen.

Similarly, in *Insert*, the value of the input chosen is immaterial thus $d(Buf_n, Buf) = 1$ since it is the size of the state that forces convergence or otherwise.

Arbitrary input/output types

Avoid the problem - recognise the nature of approximation of implementation.

In any real implementation approximations will be made, e.g.,$\mathbb{N}$ will usually be implemented as $0..maxint$ and so forth.

We thus use the same definition for d and ρ, and adapt the definition of ρ_a nd ρ_c to take into account the implementation range.

Adapt definition and take Y_{imp} to be the actual implementation of Y, and then find $\frac{(\#Y_{imp} - \#T)}{\#Y_{imp}}$

where T is the largest set $T \subseteq Y_{imp}$ such that

$$\forall x? : T \bullet \forall State \bullet \mathrm{pre}\, AOp \Rightarrow \mathrm{pre}\, COp$$

and similarly for ρ_c.

With this metric should be a description of how each infinite type has been implemented, e.g., $\mathbb{N}$ as $0..maxint$.

Open questions

What is the relationship of these metrics and the metric space approaches to semantics?

What are the topological characteristics of the metrics?

What is their basis in terms of data refinement?

What alternative metrics are there?

Work in progress

Defining meaning of 'uniform' in *Element-wise refine - apply refinement uniformly to each of* (S_n).