Dolce far niente?

Refined versions of *skip*, etc.

(Procs.: "Perspicuity and Granularity in Refinement")

Eerke A. Boiten

Refine 2011, 20 June, Limerick

Thanks to Carroll Morgan.

AS

$$b : \mathrm{bag}\,\mathbb{N}$$

$AInit$

$$AS'$$

$$b' = [\![\,]\!]$$

Ain

$$\Delta AS$$

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

$$b' = b \uplus [\![x?]\!]$$

$Aout$

$$\Delta AS$$

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

$$b \neq [\![\,]\!]$$

$$b = b' \uplus [\![x!]\!]$$

$$x! = min(b)$$

Retrieve relation: $b = items(s)$

CS

$s : \mathrm{seq}\,\mathbb{N}$

CInit

CS'

$s' = \langle\,\rangle$

Cin

$\ldots$

Cout

$\ldots$

$\ldots$

$\ldots$

Cycle

ΔCS

$(s = \langle\,\rangle \land s' = \langle\,\rangle) \lor$
$s' = (tail\ s) \frown \langle head\ s \rangle$

Retrieve relation: $b = items(s)$

$$
\begin{array}{|l}
\hline
\textbf{CS} \\
s : \mathrm{seq}\,\mathbb{N} \\
\hline
\end{array}
$$

$$
\begin{array}{|l}
\hline
\textbf{CInit} \\
CS' \\
\hline
s' = \langle\,\rangle \\
\hline
\end{array}
$$

$$
\begin{array}{|l}
\hline
\textbf{Cin} \\
\ldots \\
\hline
\end{array}
\qquad
\begin{array}{|l}
\hline
\textbf{Cout} \\
\ldots \\
\hline
\end{array}
$$

$$
\begin{array}{|l}
\hline
\textbf{Sort} \\
\Delta CS \\
\hline
items(s) = items(s') \\
sorted(s') \\
\hline
\end{array}
$$

$$
\begin{array}{|l}
\multicolumn{1}{l}{\textit{Cout}} \\
\Delta CS \\
x! : \mathbb{N} \\
\hline
s \neq \langle\, \rangle \\
sorted(s) \\
s = \langle x! \rangle \frown s'
\end{array}
$$

$$
\begin{array}{|l}
\multicolumn{1}{l}{\textit{Sort}} \\
\Delta CS \\
\hline
items(s) = items(s') \\
sorted(s')
\end{array}
$$

$SortOut == Sort \fatsemi Cout$, i.e.,

$$
\begin{array}{|l}
\multicolumn{1}{l}{\textit{SortOut}} \\
\Delta CS \\
x! : \mathbb{N} \\
\hline
s \neq \langle\, \rangle \\
\exists\, s'' : \mathrm{seq}\,\mathbb{N} \bullet items(s) = items(s'') \land sorted(s'') \\
\qquad\qquad\qquad \land\ s'' = \langle x! \rangle \frown s'
\end{array}
$$

Uncontroversial: *SortOut* refines *AOut* under the retrieve relation $b = \mathit{items}(s)$. (*Sort* and *Cout* are only used to define *SortOut*)

(More precisely: the data types containing . . . , it is a data refinement!)

(Aside: $\mathit{Cin} \mathbin{⨾} \mathit{Sort}$ also makes sense but needs stronger retrieve.)

Less simple: "does *Cout* refine *Aout*"?

The answer depends on the prevalent notion of refinement, and (!) on the status of *Sort*.

“Prevalent Notion of Refinement

(1) Consistency The effect of the concrete is allowed by the abstract.

(2) Enabledness When operations can be invoked in the abstract state, they can be invoked in the concrete state as well.

(3) Restricted consistency Where the abstract is enabled, the effect of the concrete is allowed by the abstract.

(1) or (3) is the essence: client spots no difference; (2) preserves client experiments.

(1) implies a converse of (2)

(3) without (2) makes no sense: not transitive

Traditional Z: (2)+(3)

Trace refinement: (1)

Event-B or Action Systems: (1)+ weaker (2) ("global deadlock" or "termination")

Failures-based: (1)+(2), with subtle differences in details of (2)

Adding Operations in Refinement

First the simple cases:

Alphabet Extension Just add more ops: fine in (3), odd in (1). Semantics: consider only traces over “old” alphabet.

Alphabet Translation 1-*n* map abstract to concrete *alphabets*. (Event-B “splitting”). Semantics: translate traces.

Moving on to harder cases. . .

Adding Operations in Refinement

Perspicuous Operations

"nothing" happens "most of the time".
(Stuttering steps)

Concrete events which refine abstract "skip". Call such events *perspicuous*. Refers to "this" refinement step only. Semantics: cross out perspicuous events from concrete trace, then compare with abstract.

("internal" events later: more requirements.)

Event-B: "refinements of modelling"

In example: *Sort* and *Cycle* are candidates

Perspicuous Operations: Divergence

Event-B: no additional deadlock due to new perspicuous events.

Semantics view: "crossing out" guaranteed to terminate, introduction of perspicuous events does not turn finite traces into infinite ones.

Proof of non-divergence: through variants etc. Preserved subsequently if (1) rather than (3).

Unfortunately both *Sort* and *Cycle* diverge. (How to fix . . .)

Butler (IFM'09): "*The new events introduced in a refinement step can be viewed as hidden events not visible to the environment of a system and are thus outside the control of the environment*"

So are Event-B new events just perspicuous?

Internal Operations

An internal operation is (?) perspicuous, with a special status:assumed to be invisible to the environment, under internal control of the system only.

Inspiration: process algebras (encapsulating internal communication, encoding internal choice, . . .)

Semantics (assuming no abstract internal ops): take joint behaviour of *all* concrete traces that match when internals crossed out, and compare. (As in LTS $\rightarrow$ replaced by $\Rightarrow$ for "weak" . . .)

Consequences:

- internal actions have a special status which remains

- if internal actions are necessary for progress, they will "eventually" happen, so external operations are viewed as "enabled" if their before-state is reachable through internal behaviour;

- no need for independent refinement conditions for internal operations: all internal behaviour is viewed in the context of its composition with external behaviour. Thus, internal operations need *not* be refinements of *skip*.

B [Butler] and Z [Derrick/Boiten] "weak" refinement: *prevention* of divergence. More general: *reduce* [Boiten/Derrick/Schellhorn 2009].

Adding Operations in Refinement

Action Refinement

No special status actions, just a translation that matches 1 abstract action to a sequence of concrete ones.

Like ASM 1-n diagrams.

Special case $n = 2$: like introducing "$;$" in refinement calculus: find the intermediate state. Problem: interference in intermediate state. (Exists whatever approach.)

How to Reduce Granularity

Three semantic models for reducing the granularity of actions in refinement:

- perspicuous actions that take on some of the "work" ;
- internal actions to the same effect – either as perspicuous actions, or more general "weak"
- explicit decompositions of actions where all parts have same status.

Combining with Prevalent Notion

Consider $n = 2$, first step is preparatory, second is real work *AW* vs *Prep* and *CW*.

Using perspicuous actions: *Prep* refines *skip*, *CW* refines *AW*. Now cannot have (2) (explain: two reasons).

Using internal actions, same rule: same problems.

Using weak refinement rules: this works with (2), and can be done with either (1) or (3).

Using explicit action refinement: fine.

A Conclusion for Event-B

Two entangled design decisions:

- to have essentially a trace semantics with only global deadlock prevention;

- to use stuttering step refinements for reducing granularity.

Advantage: simple refinement obligations from both.

Disadvantage: cannot strengthen basic refinement without a very significant cost.