Formal Methods in Software Engineering

Assignment 3 (Rodin and Refinement)

(Due on Fri 06 Mar 2026)

1. Recall that in the Z-notion of refinement done in class, an ADT $\mathcal{B}$ was said to refine an ADT $\mathcal{A}$ precisely when $L(\mathcal{B}^+) \subseteq L(\mathcal{A}^+)$. An alternate way of defining the totalized version of an ADT was proposed, let us call it $\mathcal{A}^\oplus$, wherein $\mathcal{A}^\oplus$ was over the same ADT type as $\mathcal{A}$ (in particular the set of outputs O_n for an operation n is *not* augmented with a new "$\perp$" value). The rest of the definition of $\mathcal{A}^\oplus$ remains the same as the definition given in class (Slide 19 of ADT-refinement slide deck).

   Argue that this way of defining totalization does not suffice, in that there are ADTs $\mathcal{A}$ and $\mathcal{B}$ of the same type, such that $L(\mathcal{B}^\oplus) \subseteq L(\mathcal{A}^\oplus)$ but $\mathcal{B}$ does not "refine" $\mathcal{A}$ (in that a client which is "happy" with $\mathcal{A}$ may not be "happy" with $\mathcal{B}$, in the sense discussed in class).

   (*Hint*: Consider a simple ADT type $(\{n\}, \{a\}, \{b\})$, and the reason why the range set had to be augmented with "$\perp$" in the notion of relational refinement.)

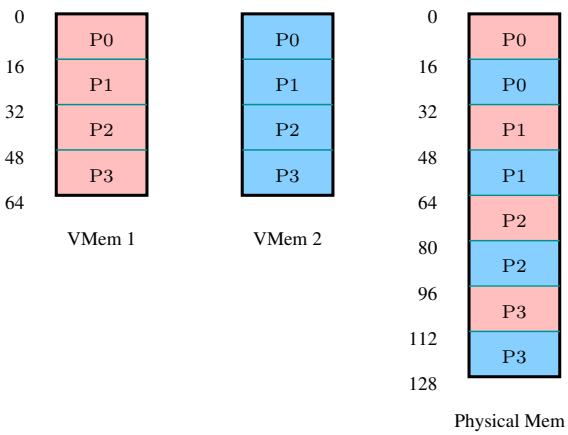
2. This question asks you to use Rodin to reason about the correctness of a virtual memory system.

   Notes:

   (a) Please go through the Rodin User's Handbook ("rodin-doc.pdf") on the course page. In particular Section 3.3 on "Mathematical Notation" in Rodin will be helpful.

   (b) Export your project as a zipped archive and submit on Teams.

   You are required to verify the functional correctness of a page table based virtual memory scheme implemented by an operating system. In a virtual memory system, each process sees a "virtual" address space (in this case of 64 Bytes, with each byte being an addressable word of memory), even though there is a common physical memory shared by the processes. The operating system and the processor (through its Memory Management Unit or MMU) orchestrate this view by setting up a page table for each process, which maps its virtual memory addresses to physical memory addresses. The read and write operations now read and write the physical memory, while translating addresses using the appropriate page table for the process accessing memory. See Sec. 18.1 of the chapter "Introduction to Paging" in the book *Operating Systems: Three Easy Pieces* (`https://pages.cs.wisc.edu/~remzi/OSTEP/vm-paging.pdf`) for more details on the proposed paging scheme.



In the present problem, we have two processes P_1 and P_2 each with a virtual address space of 64 B, and a physical memory of 128 B. The paging scheme used is the following. The virtual address space is divided into four pages of 16 B each, while the physical memory is divided into eight pages (called "frames") also of size 16 B each. Each address in a process's address space is a 6-bit address. The first two (i.e. most-significant) bits are used to obtain the virtual page number (VPN), while the last four bits are used to obtain the offset into the corresponding page. Each process has a page table ($ptab_1$ and $ptab_2$ which maps two-bit numbers (representing a VPN) to 3-bit numbers (representing the physical frame number (PFN)). The OS initially sets the page table of P_1 to map all addresses to even-numbered addresses in physical memory, and similarly for P2 to all odd-numbered addresses in physical memory, as shown in Fig. 2.

Your job is to verify the correctness of this virtual memory system by:

(a) Specifying an abstract system (or machine) called `memory`,

(b) Specifying a concrete system called `vmemory` which models the virtual memory system; and

(c) Proving that the concrete system refines the abstract.

The abstract system `memory` has two processes, each with its dedicated virtual memory space `vmem1` and `vmem2` respectively. The operation `read1` (for the read operation from process P1) takes an address in 0..63 in the form of a (page-number, offset) pair, and returns the integer value stored at that address in `vmem1`. Similarly the operation `write1` takes a pair (pn, off) representing an address in 0..63, and an integer value `val`, and replaces the contents at that address in `vmem1` by `val`.

The concrete system `vmemory` uses page tables to translate virtual addresses to physical ones. It has a physical memory `pmem` of size 128 B.

Each process has a page table (`ptab1` and `ptab2`) which maps the virtual page numbers to physical frame numbers. The operations "read1", "write1", etc now read and write the physical memory using the page tables to translate addresses.

Use Rodin to show that "vmemory" refines "memory". You can use the built-in Event-B notion of refinement in Rodin.

In particular you must do the following steps. The given zip file `vmemory.zip` (linked on the course page) contains two machines `memory` and `vmemory`.

(a) Complete the two models to complete/add the `read1`, `write1`, `read2` and `write2` operations.

(b) Add any additional state invariants you may need in the two machines.

(c) Add the gluing relation in `vmemory` (call the invariants `glue1`, `glue2`, etc).

(d) Discharge the resulting proof obligations in Rodin. If you cannot discharge them automatically, give a justification for why you think are logically valid.