Formal Construction of a Non-blocking Concurrent Queue Algorithm
(a case study in concurrency and atomicity)

by J-R. Abrial and D. Cansell

February 2004
Background

- The following study (proposed by Richard Bornat)

Simple, Fast, and Practical Non-Blocking and Blocking Concurrent Queue Algorithms

- by M.M. Michael and M.L. Scott
This Presentation

- Informal and partial construction of the algorithm :-)

- Formal construction :-|

- Final Algorithm and conclusion :-)
A Non-empty Queue (Ready for Dequeue)
A Successful Dequeue
A Queue (Ready for Enqueue)
A Successful Enqueue
A Queue (Not Ready for Enqueue but still for Dequeue)
A Successful Adapt
Summary of Non-concurrent Actions

Dequeue ≜
if $Head \neq Tail$ then
    $Head := Next(Head)$;
    return(true)
else
    return(false)
end

Enqueue ≜
nde := new_node;
Next(nde) := null;
if Next(Tail) = null then
    Next(Tail) := nde
end

Adapt ≜
if Next(Tail) \neq null then
    Tail := Next(Tail)
end
Concurrency Assumptions

- Several processes can perform these actions “simultaneously”

- Assumption: No critical section around the queue

- Global variables
  - The nodes
  - The connections between them (pointers)

- Local variables (within processes)
  - Local pointers to Head and Tail (might be WRONG)
  - Local pointers to candidates new nodes to be enqueued
The Rules of the Game: Allowable Atomic Actions

- Reading

\[ local\_variable \ := \ global\_variable \]

- Compare_and_Swap

\[
\text{if } \ global\_variable \ = \ local\_variable \ \text{then} \\
\text{global}\_variable \ := \ local\_value \\
\text{end}
\]
Development Strategy: First Informal

- Merge *Adapt* within *Dequeque* and *Enqueue*

- **Transform** the 2 remaining primitives
  - Into a number of allowable *atomic actions*
  - **Sequentially executed** by each process
Development Strategy: and then Formal

- Formalize atomic actions as events (guard + action)

- Introduce and refine them gradually

- Prove each refinement step
Dequeue with Adapt (still non-concurrent)

if $Head \neq Tail$ then
  $Head := \text{Next}(Head)$;
  return(true)
else
  return(false)
end

loop
if $Head \neq Tail$ then
  $Head := \text{Next}(Head)$;
  return(true)
elsif $\text{Next}(Tail) \neq \text{null}$ then
  $Tail := \text{Next}(Tail)$
else
  return(false)
end
end

Adapt $\equiv$

if $\text{Next}(Tail) \neq \text{null}$ then
  $Tail := \text{Next}(Tail)$
end
Enqueue with Adapt (still non-concurrent)

\[
\text{nde} := \text{new\_node};
\]

\[
\text{Next}(\text{nde}) := \text{null};
\]

\[
\text{loop}
\]

\[
\text{if } \text{Next}(\text{Tail}) \neq \text{null} \text{ then}
\]

\[
\text{Tail} := \text{Next}(\text{Tail})
\]

\[
\text{else}
\]

\[
\text{Next}(\text{Tail}) := \text{nde};
\]

\[
\text{break}
\]

\[
\text{end}
\]

\[
\text{end}
\]

\[
\text{Adapt} \equiv
\]

\[
\text{if } \text{Next}(\text{Tail}) \neq \text{null} \text{ then}
\]

\[
\text{Tail} := \text{Next}(\text{Tail})
\]

\[
\text{end}
\]
Dequeue with Adapt and Local Variables (still non-concurrent)

```plaintext
loop
  if Head ≠ Tail then
    Head := Next(Head);
    return(true)
  else
    if Next(Head) ≠ null then
      Tail := Next(Head)
      return(true)
    else
      return(false)
  end
end
```

```plaintext
loop
  hdd := Head;
  tld := Tail;
  nxd := Next(hdd);
  if hdd ≠ tld then
    Head := nxd;
    return(true)
  else
    if nxd ≠ null then
      Tail := nxd
    else
      return(false)
  end
end
```
Enqueue with Adapt and Local Variables (still non-concurrent)

\[
\text{nde} := \text{new\_node}; \\
\text{Next}(\text{nde}) := \text{null}; \\
\text{loop} \\
\text{if} \ \text{Next}(\text{Tail}) \neq \text{null} \ \text{then} \\
\quad \text{Tail} := \text{Next}(\text{Tail}) \\
\text{else} \\
\quad \text{Next}(\text{Tail}) := \text{nde}; \\
\quad \text{break} \\
\text{end} \\
\text{end}
\]
Dequeue: Preparing **Concurrency**

\[
\text{loop} \\
\quad hdd := \text{Head}; \\
\quad tld := \text{Tail}; \\
\quad nxd := \text{Next}(hdd); \\
\quad \text{if } hdd \neq tld \text{ then} \\
\quad \quad \text{Head} := nxd; \\
\quad \quad \text{return(true)} \\
\quad \text{else} \\
\quad \quad \text{if } nxd \neq \text{null} \text{ then} \\
\quad \quad \quad \text{Tail} := nxd \\
\quad \quad \text{else} \\
\quad \quad \quad \text{return(false)} \\
\quad \text{end} \\
\text{end} \\
\text{end}
\]

\[
\text{loop} \\
\quad hdd := \text{Head}; \\
\quad tld := \text{Tail}; \\
\quad nxd := \text{Next}(hdd); \\
\quad \text{if } hdd \neq tld \text{ then} \\
\quad \quad \text{if } \text{Head} = hdd \text{ then} \\
\quad \quad \quad \text{Head} := nxd; \\
\quad \quad \quad \text{return(true)} \\
\quad \quad \text{else} \\
\quad \quad \quad \text{elsif } \text{Tail} = tld \text{ then} \\
\quad \quad \quad \quad \text{if } nxd \neq \text{null} \text{ then} \\
\quad \quad \quad \quad \quad \text{Tail} := nxd \\
\quad \quad \quad \quad \text{else} \\
\quad \quad \quad \quad \quad \text{return(false)} \\
\quad \quad \text{else} \\
\quad \quad \quad \text{return(false)} \\
\quad \text{end} \\
\text{end} \\
\text{end}
\]
Enqueue: Preparing Concurrency

```
nde := new_node;
Next(nde) := null;
loop
  tle := Tail;
nxe := Next(tle);
  if nxe ≠ null then
    Tail := nxe
  else
    Next(tle) := nde;
    break
  end
end
```
Dequeue: Atomic Actions

loop
  hdd := Head; ...
  tld := Tail; ...
  nxd := Next(hdd); ...
  if hdd ≠ tld then
    if Head = hdd then
      Head := nxd;
      return(true)
    end
  elseif Tail = tld then
    if nxd ≠ null then
      Tail := nxd
    else
      return(false)
    end
  end
end
Dequeue: Showing Concurrent Events

loop

\[
\begin{align*}
    \text{hdd} & := \text{Head}; \\
    \text{tld} & := \text{Tail}; \\
    \text{nxd} & := \text{Next}(\text{hdd}); \\
\text{if} & \ hdd \neq tld \land \text{Head} = \text{hdd} \text{ then} \\
    & \quad \text{Head} := \text{nxd}; \\
    \quad \text{return}(\text{true}) \\
\text{end}
\end{align*}
\]

Deq1

Deq2

Deq3

\[
\begin{align*}
\text{if} & \ hdd = \text{tld} \land \text{nxd} \neq \text{null} \land \text{Tail} = \text{tld} \text{ then} \\
    & \quad \text{Tail} := \text{nxd} \\
\text{end}
\end{align*}
\]

Adapt_Deq

\[
\begin{align*}
\text{if} & \ hdd = \text{tld} \land \text{nxd} = \text{null} \land \text{Tail} = \text{tld} \text{ then} \\
    \quad \text{return}(\text{false}) \\
\text{end}
\end{align*}
\]

Deq_false

end
Enqueue: Atomic Actions

\[
\begin{align*}
nde &:= \text{new}.\text{node}; \\
Next(nde) &:= \text{null}; \\
\end{align*}
\]

\[
\text{loop}
\]

\[
\begin{align*}
tle &:= \text{Tail}; \\
nxe &:= \text{Next}(tle); \\
\end{align*}
\]

\text{if } nxe \neq \text{null } \text{then}

\[
\begin{align*}
\text{if } \text{Tail} = tle &\text{ then} \\
\text{Tail} &:= nxe \\
\end{align*}
\]

\text{endif}

\[
\text{elsif } \text{Next}(tle) = nxe \text{ then}
\]

\[
\begin{align*}
\text{Next}(tle) &:= nde; \\
\text{break} \\
\end{align*}
\]

\text{endif}

\text{end}

\text{end}
Enqueue: Showing Concurrent Events

```plaintext
nde := new_node;
Next(nde) := null

... loop


tle := Tail

... nxe := Next(tle)

... if nxe ≠ null ∧ Tail = tle then

	Tail := nxe

end

... if nxe = null ∧ Next(tle) = nxe then

	Next(tle) := nde;

	break

end

... end

Enq1

Enq2

Enq3

Adapt_Enq

Enq
```
Level of Concurrency

- 7 Dequeue events:

  Deq1, Deq2, Deq3, Deq_true, Adapt_deq, Deq_false, Deq_loop

- 6 Enqueue events:

  Enq1, Enq2, Enq3, Adapt_enq, Enq, Enq_loop

- Supposing 20 concurrent processes:
  - 260 on-going concurrent events!
  - $13^{20}$ different situations!
Formal Abstract Specification: Serialization

- Defining constants (the nodes) and variables (elements of queue)

- Defining the invariant: the queue properties.

- Defining the three basic events: Dequeue, Enqueue, Adapt

- 45 proofs (40 automatic)
1st Refinement: Dequeue

- Introducing processes

- Introducing local process variables for dequeuing

- Refining Dequeue and Adapt

- Introducing extra invariants

- Introducing 4 more events: Deq1, Deq2, Deq3, Deq_loop

- Discovering (and solving) the ABA problem (proofs impossible)

- 79 proofs (64 automatic)
2nd Refinement: Waste Basket

- Introducing a waste basket

- Where dequeued nodes are thrown away for ever

- 22 proofs (21 automatic)
3rd Refinement: Enqueue

- Introducing *local* process variables for dequeuing
- Refining *Enqueue* and *Adapt*
- Introducing *extra invariants*
- Introducing *4 more events*: Enq1, Enq2, Enq3, Enq_loop
- *83 proofs* (71 automatic)
4th Refinement: Free List

- Introducing a **global free list**

- Introducing **extra invariants**

- Refining **existing events**

- **57 proofs** (56 automatic)
5th Refinement: Concrete Nodes

- Projecting abstract nodes on concrete nodes
- Stamping concrete nodes with numbers
- Introducing extra invariants
- Refining existing events
- 40 proofs (28 automatic)
6th Refinement: Final Implementation

- Removing abstract nodes (data refinement)

- Introducing extra invariants

- Refining existing events

- 30 proofs (12 automatic)
Summary of Development

- Formal Specification: Basic Dequeue, Enqueue, and Adapt (45,6)
- 1st refinement: Refining Dequeue (79,15)
- 2nd refinement: Introducing waste basket (22,1)
- 3rd refinement: Refining Enqueue (83,12)
- 4th refinement: Introducing a "free list" (57,1)
- 5th refinement: Making the nodes more concrete (40, 12)
- 6th refinement: Final Implementation (30,18)
- Total: 355,65 \( \sim \) 81% automatic
Conclusion: What We Have Learned

- Correct proved construction versus final validation

- Determining the atomic actions and mechanisms (CAS)

- Importance of abstraction where strange things are permitted

- Importance of small animations

- An interesting experience with model-checking (PRO_B)