Modelling Concurrency

Uri Abraham
Departments of Mathematics and Computer Science,
Ben-Gurion University, Be'er-Sheva, Israel
Visiting Department of Mathematical Sciences, CMU.

September 16, 2003

1 Correctness proof

We define here the predicate calculus language, the axioms, and then required correctness conditions, all formulated in the language.

1.0.1 Higher-level signature

The higher-level events are described in a language called DCC (for Directory Cache Coherence). The signature of this language is the following. There are three sorts, *HL* (for higher-level events), *Address* for the address space (set of all memory locations), and *Data* for the type of objects loaded and stored. We have the following predicates defined on the events *HL*.

1. Unary predicates: MemoryAccess, MemoryBarrier, Commit.
2. Unary predicates *Load* and *Store*.
3. Unary predicates P_i for $i \in \{1, \ldots, N\}$.
4. Binary predicates $<_i$ for $i \in \{1, \ldots, N\}$. And binary predicate $<_{Directory}$.
5. Unary function *Value* is from the events *HL* into *Data*.
6. Unary functions μ and χ are from *HL* into *HL*.
7. Unary function *source* is from *HL* to *HL*.

8. unary function *address* from *HL* to *Address*.

1.0.2 Higher-level axioms

We have the following axioms (or properties) formulated in DCC.

Events: The set of events defined by the predicates $\mathsf{MemoryAccess}$, $\mathsf{MemoryBarrier}$, and Commit are pairwise disjoint. The sets *Load* and *Store* are disjoint. For every x, $\mathsf{MemoryAccess}(x)$ iff $\mathit{Load}(x) \vee \mathit{Store}(x)$. The events $P_0, \ldots, P_N$ are pairwise disjoint. Each $\mathsf{MemoryAccess}$ and $\mathsf{MemoryBarrier}$ is in some P_i. If $P_i(x)$ then $\mathsf{MemoryAccess}(x) \vee \mathsf{MemoryBarrier}(x)$.

Orderings Each $<_i$ is a linear ordering of P_i. Every event e in P_i has only a finite number of events a in P_i such that $a <_i e$. Relation $<_{Directory}$ is a linear ordering of the set of $\mathsf{MemoryBarrier}$ events. Again, each $\mathsf{MemoryBarrier}$ event e has only a finite number of $\mathsf{MemoryBarrier}$ events a such that $a <_{Directory} e$.

μ properties: The function μ is defined on the $\mathsf{MemoryAccess}$ events. For every event m such that $\mathsf{MemoryAccess}(m)$, $\mathsf{Commit}(\mu(m))$. For every $\mathsf{MemoryAccess}$ event m, $\mathit{address}(m) = \mathit{address}(\mu(m))$ and $\mathit{Value}(m) = \mathit{Value}(\mu(m))$. If $\mu(m_1) = \mu(m_1)$, then m_1 and m_2 belong to the same process. That is, for some i $P_i(m_1)$ and $P_i(m_2)$.

For every $\mathsf{MemoryAccess}$ events m_1 and m_2 in P_i, if $m_1 <_i m_2$, then $\mu(m_1) \leq_{Dirctory} \mu(m_2)$.

χ and $\Leftarrow$: For every *Load*/*Store* event h by P_i:

$$\chi(h) = max_{<_{Directory}}(\{\mu(h)\} \cup \{\mu(h') : \ \mathit{Store}(h') \vee \mathit{Load}(h') \text{ and } \exists mb \in \mathsf{MemoryBarrier}\ h' <_i mb <_i h\})$$

For every events h_1 and h_2 that are $\mathsf{MemoryAccess}$ events we define:

$$h_1 \Leftarrow h_2 \text{ iff } \chi(h_1) <_{Directory} \chi(h_2) \vee (\chi(h_1) = \chi(h_2) \wedge \exists i\ h_1 <_i h_2).$$

Source: Let h be any *Load* event in P_i. There exists a Commit event c with $address(c) = address(h)$ such that $c \leq_{Dirctory} \mu(h)$ and there is a *Store* event h' such that $\mu(h') = c$. Moreover, let c_0 denote the latest such Commit in the $\leq_{Dirctory}$ ordering. Then either $c_0 <_{Directory} \mu(h)$ or else $c_0 = \mu(h)$ and then there exists a *Store* event h' in P_i such that $h' <_i h$ and $\mu(h') = c_0$. Accordingly, there are two cases in the definition of $source(h)$.

1. In the first case $c_0 = \mu(h)$, and then $source(h) = h'$ for the $<_i$ latest *Store* event h' such that $h' <_i h$ and $\mu(h') = c_0$.
2. In the second case $c_0 <_{Directory} \mu(h)$, and $source(h) = h'$ where h' is a *Store* event such that $\mu(h') = c_0$ and $h'' <_j h'$ for any other *Store* event in P_j (the process of h') such that $\mu(h'') = c_0$.

Value: If h is a *Load* event then $Value(h) = Value(source(h))$.

Inversion: Let h be a *Store*/*Load* event. h is called an **inversion** iff

$$\mu(h) <_{Directory} \chi(h).$$

1. Let h be a write memory access, such that h is an inversion. Then there does not exist a commit event c such that $\mu(h) <_{Directory} c <_{Directory} \chi(h)$ and $address(c) = address(h)$.
2. Let h be a memory access such that h is an inversion. Then there does not exist a *Store* memory access event w, such that

$$\mu(h) <_{Directory} \mu(w) <_{Directory} \chi(h) \quad \text{and } address(w) = address(h).$$

1.0.3 Correctness conditions

We must prove the following.

1. The *beforeOrder* relation $\Leftarrow$ is a linear ordering on the *Store*/*Load* events.
2. If $a <_i b$ and a, b access the same address, then $a \Leftarrow b$.
3. If a and b are *Store*/*Load* events in P_i and there exists a memory barrier event mb such that $a <_i mb <_i b$, then $a \Leftarrow b$.
4. For every *Load* event e:

(a) $source(e)$ is a $Store$ event. $address(e) = address(source(e))$ and $Value(e) = Value(source(e))$.

(b) $source(e) \Leftarrow e$.

(c) There is no $Store$ event w with $address(w) = address(e)$ such that $source(e) \Leftarrow w \Leftarrow e$.

Theorem 1.1 *The axioms of 1.0.2 imply the correctness conditions 1.0.3.*

References

[1] U. Abraham, *Models for Concurrency,* Gordon and Breach, 1999.

[2] J. Barwise, *Admissible Sets and Structures.* Springer 1975.

[3] L. Lamport. Specifying concurrent systems with TLA$^+$. In Manfred Broy and Ralf Steinbrüggen, editors, *Calculational system Design,* pages 183-247, Amsterdam, 1999. IOS Press.

[4] L. Lamport, M. Sharma, M. Tuttle, and Y. Yu. The Wildfire Challenge Problem. 2001.