Language Support for Concurrency

November 13, 2017

Classic concurrency constructs

- locks
- semaphores (binary, counting)
- monitors
- conditional critical regions

Process Algebras: CSP and CCS

Process calculi (process algebras): algebraic notation for describing *processes*, their sequential and parallel *composition* and *communication* over *channels*

Communicating Sequential Processes (Hoare, 1978)

alphabet of actions: $\alpha_V = \{in1p, in2p, small, large, out1p\}$

$V = (in2p \to (large \to V \Box small \to out1p \to V) \Box in1p \to small \to V)$
or formally, as *least fixpoint* (μ) of above equation system
$V = \mu X.(in2p \to (large \to X \Box small \to out1p \to X) \Box in1p \to small \to X)$

deterministic choice $\Box$, nondeterministic choice, interleaving, synchronization on an event, hiding events (e.g. synchronization)

Calculus of Communicating Systems (Milner, 1980)
later π-calculus, allowing communicating channel names $\Rightarrow$ mobility

1. Software Transactional Memory

based on Hoare's Conditional Critical Regions

```
public int get() {
  atomic (items != 0) {
    items --;
    return buffer[items];
  }
}
```

What's missing:

what is the data protected ?

when is a blocked thread released ?

What does STM offer ?

dynamically non-conflicting executions can operate concurrently

CCR conditions re-evaluated only on a shared update

non-blocking implementation (prevents deadlock, priority inversion)

Goals: minimal restrictions for code enclosed in `atomic`

low implementation overhead *outside* CCRs

Sample implementation [Harris,Fraser - OOPSLA03]

```
void STMStart()
void STMAbort()
boolean STMCommit()
boolean STMValidate()
void STMWait()
```

Sample implementation - Clojure refs

Clojure: dynamic language (Lisp dialect) compiled to Java bytecode

Refs allow shared use of mutable storage locations

- mutation of location allowed only in transaction

2. Persistent Data Structures

All values are immutable

including composite ones

change is actually a function that returns a new value

old value still exists and can be used

To change state:

construct new compound value

change the reference

$\Rightarrow$ can be done much easier

3. Actors

Everything is an actor.

Actors may
- send messages to other actors
- create new actors (a finite number)
- designate behavior for next message received

Similar to
- Smalltalk (send messages)
- process algebras

4. Dataflow

Examples in Oz [Wikipedia]

– Programs wait until variables bound to values

```
thread
   Z = X+Y     % waits until both X and Y are bound.
   {Browse Z}  % shows the value of Z.
end
thread X = 40 end
thread Y = 2 end
```

– immutable values (cannot change while bound)

5. Tuple Spaces

[after vanRoy and Haridi]

`out(T)` adds tuple `T` to the tuple space.

`in(T)` reads and removes tuple (based on pattern matching)

`rd(T)` reads nondistructively

`eval` creates a new process evaluating a tuple (used for IPC) can be implemented with a lock, a dictionary and a concurrent queue

Concurrent Queue in Linda

```
init() {
  out("head", 0);
  out("tail", 0);
}
put(elem) {
  in("tail", ?tail);
  out("elem", tail, elem);
  out("tail", tail+1);
}
take(elem) {
  in("head", ?head);
  out("head", head+1);
  in("elem", head, ?elem);
}
```

http://www.lindaspaces.com/teachingmaterial/LindaTutorial_Jan2006.pdf