Language Support for Concurrency

November 14, 2016

Classic concurrency constructs

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

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