On Context Switch Upper Bound for Checking Linearizability

Vadim Mutilin

Institute for System Programming, RAS

June 2, 2010

High Level View

- **Linearizability** – the parallel execution of operations is equivalent to sequential one
- Bound the number of **context switches** which are done by scheduler
- **How many** context switches do we need to be sure that a program is linearizable

The Specification of CELL Example

State of CELL is empty ($CELL = \emptyset$) or contains an integer ($CELL = \{i\}$)

bool insert(int i)

if($CELL = \emptyset$)

$CELL' = \{i\} \wedge RESULT = T$

else

$CELL' = CELL \wedge RESULT = F$

bool delete()

if($CELL = \emptyset$)

$CELL' = CELL \wedge RESULT = F$

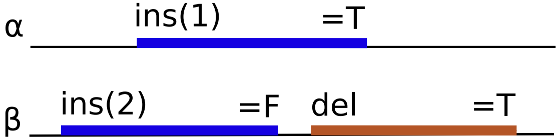
else

$CELL' = \emptyset \wedge RESULT = T$

The Notion of Linearizability

Implementation History (from empty state)

β:ins(2); α:ins(1); β:=F; β:del; α:=T; β:=F

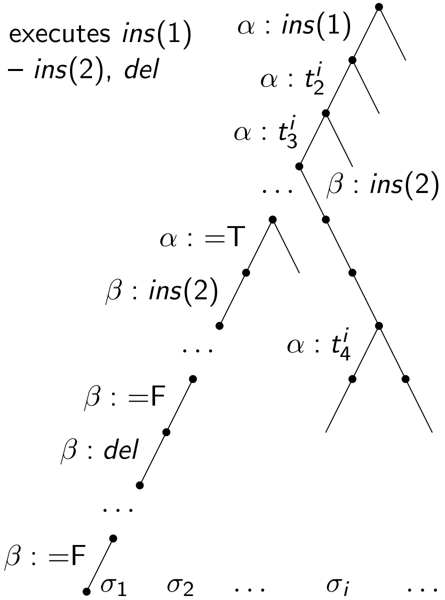


Linearizations

- ☑ Lin1: α:ins(1)=T; β:ins(2)=F; β:del=T
- ☒ Lin2: β:ins(2)=F; α:ins(1)=T; β:del=T
- ☒ Lin3: β:ins(2)=F; β:del=T; α:ins(1)=T

The Execution Traces

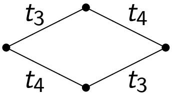
α executes $ins(1)$
β – $ins(2)$, del



1. At every point a **context switch** may occur
2. The size is exponential
3. $H(\sigma_i)$ – history of execution trace σ_i is a projection of σ_i on invocation and return transitions
4. The program is linerizable **iff** all $H(\sigma_i)$ are linearizable

$$\sigma_1 = \alpha : t_1 \quad \alpha : t_2 \quad \alpha : t_3 \quad \beta : t_1 \quad \beta : t_2 \quad \beta : t_3 \quad \alpha : t_4$$

$\Downarrow$



$\{(t_3, t_4)$ are independent, $H(\sigma_1) = H(\sigma_2)\} \Downarrow$

$$\sigma_2 = \alpha : t_1 \quad \alpha : t_2 \quad \alpha : t_3 \quad \beta : t_1 \quad \beta : t_2 \quad \alpha : t_4 \quad \beta : t_3$$

$\Downarrow \{(t_2, t_4)$ are independent, $H(\sigma_2) = H(\sigma_3)\} \Downarrow$

$$\sigma_3 = \alpha : t_1 \quad \alpha : t_2 \quad \alpha : t_3 \quad \beta : t_1 \quad \alpha : t_4 \quad \beta : t_2 \quad \beta : t_3$$

$\Downarrow \{(t_1, t_4)$ are independent, $H(\sigma_3) = H(\sigma_4)\} \Downarrow$

$$\sigma_4 = \alpha : t_1 \quad \alpha : t_2 \quad \alpha : t_3 \quad \alpha : t_4 \quad \beta : t_1 \quad \beta : t_2 \quad \beta : t_3$$

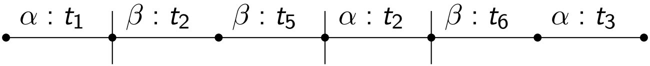
Zero context switches

Related Work

- Partial order reductions (Java PathFinder, Verisoft . . .)
  - Reduces the number of traces but still may be exponential
  - Algorithms do not fully exploit independencies
    - SleepSets – reduce only transitions
    - PersistentSet – requires independence with all successor transitions
- Iterative context switch bounding (CHESS)
  - A user provides the number of context switches
  - The number of traces is polynomial in the length of the trace

The Cycles

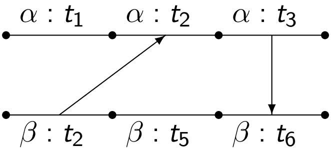
$\sigma =$



$csw(\sigma) = 3$

$\alpha =$

$\beta =$



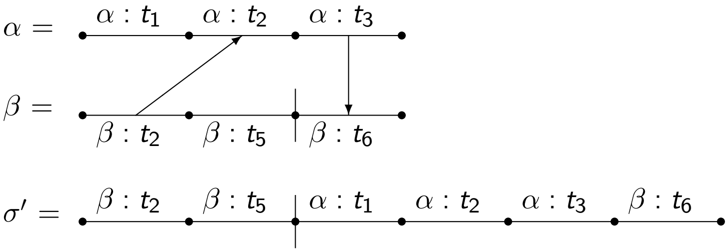
(t_2, t_2), (t_3, t_6) – dependent pairs,
all other are independent

The circle: $\beta : t_2 < \alpha : t_2 = \alpha : t_3 < \beta : t_6 = \beta : t_2$

An Upper Bound

Theorem

If the number of holding cycles in a trace σ is k then there exists a trace σ' such that $H(\sigma) = H(\sigma')$ and $csw(\sigma') \leq k$.



$$csw(\sigma') = 1$$

The CycleCount Algorithm

1. Make an over-approximation A_{op} of reachable execution traces E, such that every reachable trace is in A_{op} ($E \subseteq A_{op}$)
   - Construct the traces as a combination of operation traces
   - Remove unreachable ones
2. Find the minimum number of context switches on A_{op}
   min $cycles(\sigma) \mid \sigma \in A_{op}$, where $cycles(\sigma)$ – the number of cycles in the trace σ

Results

1. An upper bound theorem about the minimal number of context switches in an execution trace
2. An algorithm for finding the bound on context switches for the given threads and operations executing in them
3. Bounds for CELL example:
   α executes *insert*, β – *delete*, γ – *lookUp*

Threads	CycleCount	POR	Maximum
β, γ	0	1	6
α, γ	2	3	9
α, β, γ	3	3	11

Thank you!