MODULAR REASONING ABOUT CONCURRENT HIGHER-ORDER IMPERATIVE PROGRAMS

MORTEN KROGH-JESPERSEN
PHD-STUDENT, LOGIC AND SEMANTICS GROUP
THE LOGIC AND SEMANTICS GROUP

Lars Birkedal
Professor

Ranald Clouston
Postdoc

Thomas Dinsdale-Young
Postdoc

Filip Sieczkowski
Postdoc

Kasper Svendsen
Postdoc

Aleš Bizjak
PhD student

Hans Bugge Grathwohl
PhD student

Morten Krogh-Jespersen
PhD student

Yannick Zakowski
Intern
INTRODUCTION TO OUR WORK

› Software is a key part of infrastructure
› We rely on software to be bug-free
› We want more ‘efficient’ programs
› Scientifically rigorous evidence is expensive
› Tools to help software developers
Modern programming languages are imperative and higher-order (function pointers, interfaces, libraries, type-parametricity)

Some of them are even concurrent

Develop new mathematical models for modular reasoning for such modern programming languages
APPRAOCH

› Look at the operational semantics of a programming language

› Develop mathematical models and logic / type systems
› Not your ordinary math

› Experiment by testing on challenging case studies
› Specify and prove correctness by hand
› Develop tool support (Coq, Aqda) for larger studies
EXTENDING THE MATH TOOL-BOX

\[ T \cong \bigtriangledown((\mathbb{N} \rightarrow_{\text{fin}} T) \rightarrow_{\text{mon}} P(V)) \]

- The guard is pronounced 'later'
- Without it, no non-trivial sets exists satisfying the isomorphism
- New model that uses category theory / domain theory / metric spaces
A SIMPLE EXAMPLE

› Imagine a counter-module in C

\[ \text{tmp} = \ast C; \ast C = \text{tmp} + 1; \text{return tmp;} \]

› Some interleavings will compute the wrong result
› One could use locks - prevents all bad interleavings by preventing all interleavings
› A fine-grained concurrent pattern without locks using CAS

\[ \text{while (true) \{ tmp = \ast C; if (CAS(C, tmp, tmp+1)) return tmp; \}} \]
FINE-GRAINED CONCURRENT DATA STRUCTURES EXAMPLE (FGCDS)

› Stack and queues are simple data-structures. What about concurrent versions?

› FGCDS refrains from using locks and requires all clients to make progress. FGCDS are challenging!
MICHAEL-SCOTT QUEUE

initialize(Q: pointer to queue, node)
  node = new_node()
  node->next.ptr = NULL
  Q->Head = Q->Tail = node

enqueue(Q: pointer to queue, value: data type)
E1:  node = new_node()
E2:  node->value = value
E3:  node->next.ptr = NULL
E4:  loop
    E5:    tail = Q->Tail
    E6:    next = tail.ptr->next
    E7:    if tail == Q->Tail
    E8:        if next.ptr == NULL
    E9:            if CAS(&tail.ptr->next, next, <node, next.count+1>)
    E10:         break
    E11:     endif
    E12:   else
    E13:     CAS(&Q->Tail, tail, <next.ptr, tail.count+1>)
    E14:   endif
E15:  endloop
E16:  CAS(&Q->Tail, tail, <node, tail.count+1>)

dequeue(Q: pointer to queue, pvalue: pointer to data type): boolean
D1:  loop
D2:    head = Q->Head
D3:    tail = Q->Tail
D4:    next = head->next
D5:    if head == Q->Head
D6:        if head.ptr == tail.ptr
D7:            if next.ptr == NULL
D8:                return FALSE
D9:        endif
D10:     # Read value before CAS, otherwise another dequeue
D12:         *pvalue = next.ptr->value
D13:        if CAS(&Q->Head, head, <next.ptr, head.count+1>)
D14:            break
D15:        endif
D16:     endif
D17:     endloop
D18:     endloop
D19:     free(head.ptr)
D20:     return TRUE
CURRENT RESEARCH

› Ranald Clouston & Hans Bugge Grathwohl – Programming languages with guarded recursion

› Thomas Dinsdale-Young – Semi-automated verification of programs

› Filip Sieczkowski – Formalizing in Coq + Coq tutorial
CURRENT RESEARCH

› Kasper Svendsen – iCAP

› Aleš Bizjak – Models of probabilistic programming languages

› Morten Krogh-Jespersen – Verifying concurrent data structures in iCAP
HOW TO GET INVOLVED

› Opportunity to do interesting projects (PREP)

› Look at the Coq-tutorial

› Take the Semantics of Programming Language course (WARNING: Advanced!)

› Talk to us – 2nd floor of the Turing building