MODULAR REASONING ABOUT CONCURRENT HIGHER-ORDER IMPERATIVE PROGRAMS

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

THE LOGIC AND SEMANTICS GROUP

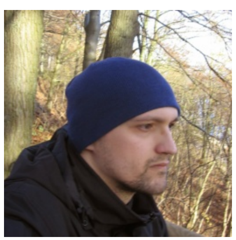
Lars Birkedal
Professor

Ranald Clouston
Postdoc

Thomas Dinsdale-Young
Postdoc

Filip Sieczkowski
Postdoc

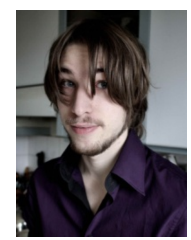
Kasper Svendsen
Postdoc



Aleš Bizjak
PhD student

Hans Bugge Grathwohl
PhD student

Morten Krogh-Jespersen
PhD student



Yannick Zakowski
Intern


Lars Birkedal
Professor


Ranald Clouston
Postdoc


Thomas Dinsdale-Young
Postdoc


Filip Sieczkowski
Postdoc


Kasper Svendsen
Postdoc


Ales Bizjak
PhD student


Hans Bugge Grathwohl
PhD student


Morten Krogh-Jespersen
PhD student


Yannick Zakowski
Intern

TROLL 2

Lars Birkedal
Professor

Ranald Clouston
Postdoc

Thomas Dinsdale-Young
Postdoc

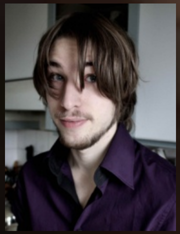
Filip Sieczkowski
Postdoc

Kasper Svendsen
Postdoc

Aleš Bizjak
PhD student

Hans Bugge Grathwohl
PhD student

Morten Krogh-Jespersen
PhD student



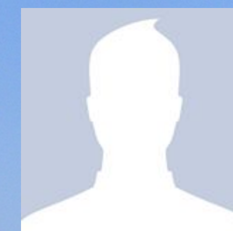
Yannick Zakowski
Intern

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

MODURES

- 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

APPROACH

- 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

$$T \cong \blacktriangleright((\mathbb{N} \rightharpoonup_{fin} T) \rightarrow_{mon} P(V))$$

- 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

```
tmp = *C; *C = tmp+1; 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

```
while (true) { tmp = *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_t)
    node = new_node()
    node->next.ptr = NULL
    Q->Head = Q->Tail = node
```

```
enqueue(Q: pointer to queue_t, value: data type)
    node = new_node()
    node->value = value
    node->next.ptr = NULL
    loop
        tail = Q->Tail
        next = tail.ptr->next
        if tail == Q->Tail
            if next.ptr == NULL
                if CAS(&tail.ptr->next, next, <node, next.count+1>)
                    break
                endif
            else
                CAS(&Q->Tail, tail, <next.ptr, tail.count+1>)
            endif
        endif
    endloop
    CAS(&Q->Tail, tail, <node, tail.count+1>)
```

```
dequeue(Q: pointer to queue_t, pvalue: pointer to data type): boolean
    loop
        head = Q->Head
        tail = Q->Tail
        next = head->next
        if head == Q->Head
            if head.ptr == tail.ptr
                if next.ptr == NULL
                    return FALSE
                endif
                CAS(&Q->Tail, tail, <next.ptr, tail.count+1>)
            else
                # Read value before CAS, otherwise another dequeue
                *pvalue = next.ptr->value
                if CAS(&Q->Head, head, <next.ptr, head.count+1>)
                    break
                endif
            endif
        endif
    endloop
    free(head.ptr)
    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