Wenn's drauf ankommt: Code korrekt beweisen

Mike Sperber

"Executable Specification"

Stopped

Runs: 7/7 Errors: 7 Failures: 0

- net.serenitybdd.demos.todos.cucumber.MaintainTodos [Runner: JUnit 4] (319.868 s)
  - Feature: Filtering todos (319.868 s)
    - View **only completed items** #123 (46.975 s)
    - Viewing items **by status** (45.242 s)
    - Viewing items **by status** (45.210 s)
    - Viewing items **by status** (45.205 s)
    - Viewing items **by status** (45.199 s)
    - Viewing items **by status** (45.204 s)
    - Viewing items **by status** (46.798 s)

Failure Trace

```
@cucumber
Feature: Filtering todos

  In order to make me feel **a sense of accomplishment**
  As a forgetful person
  I want to be to _view all of things I have completed_

  Scenario: View **only completed items** #123
    Given that Jane has a todo list containing Buy some milk, Walk the dog
    And she has completed the task called 'Walk the dog'
    When she filters her list to show only Completed tasks
    Then her todo list should contain Walk the dog

  Scenario Outline: Viewing items **by status**
    Given that Jane has a todo list containing <tasks>
    And she has completed the task called 'Walk the dog'
    When she filters her list to show only <filter> tasks
    Then her todo list should contain <expected>

    Examples: Example 1
      | tasks                      | filter    | expected      |
      | Buy some milk, Walk the dog | Active    | Buy some milk |
      | Buy some milk, Walk the dog | Completed | Walk the dog  |

    Examples: Example 2
      | tasks                      | filter    | expected      |
      | Buy some milk, Walk the dog | Active    | Buy some milk |
      | Buy some milk, Walk the dog | Completed | Walk the dog  |

    Examples: Example 3
      | tasks                      | filter    | expected      |
      | Buy some milk, Walk the dog | Active    | Buy some milk |
      | Buy some milk, Walk the dog | Completed | Walk the dog  |
```

2022 ACM Software System Award

Gernot Heiser, University of New South Wales; **Gerwin Klein**, Proofcraft; **Harvey Tuch**, Google; **Kevin Elphinstone**, University of New South Wales; **June Andronick**, Proofcraft; **David Cock**, ETH Zurich; **Philip Derrin**, Qualcomm; **Dhammika Elkaduwe**, University of Peradeniya; **Kai Engelhardt**; **Toby Murray**, University of Melbourne; **Rafal Kolanski**, Proofcraft; **Michael Norrish**, Australian National University; **Thomas Sewell**, University of Cambridge; and **Simon Winwood**, Galois, receive the **ACM Software System Award** for the development of the first industrial-strength, high-performance operating system to have been the subject of a complete, mechanically-checked proof of full functional correctness.

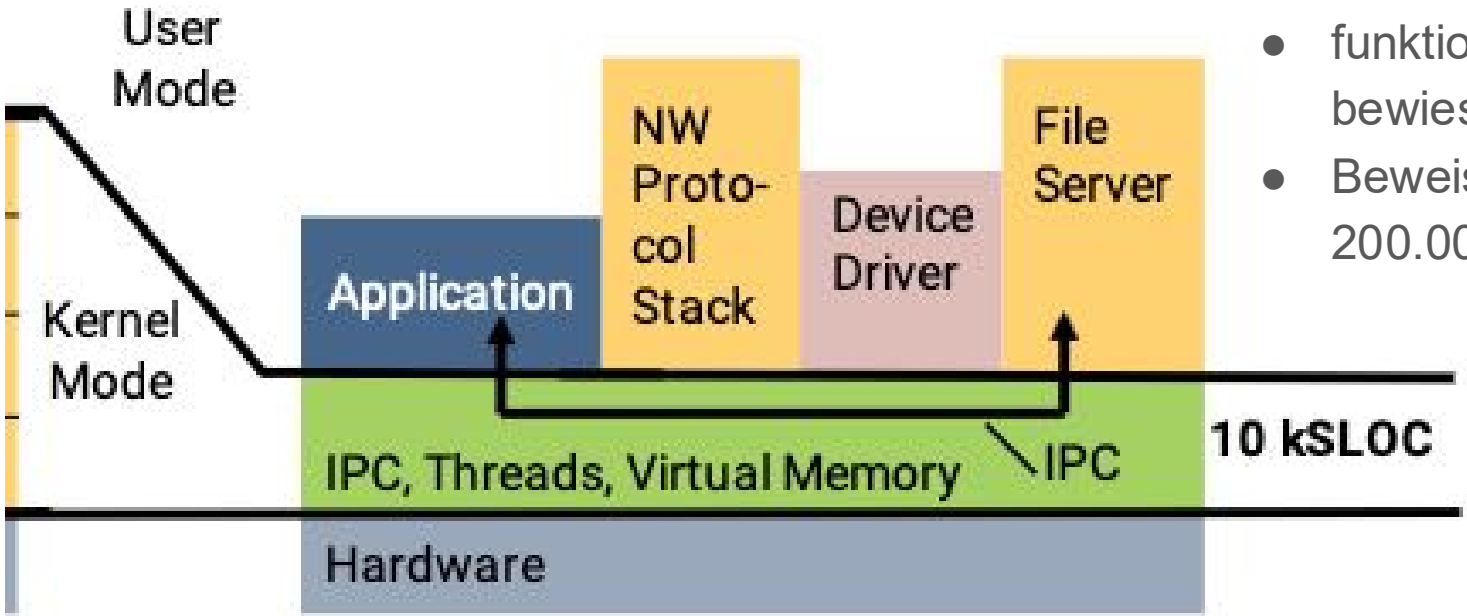
AWARDS & RECOGNITION

Software System Award Goes to Fourteen for the Development of Groundbreaking High-Performance Operating System

Gernot Heiser, University of New South Wales; **Gerwin Klein**, Proofcraft; **Harvey Tuch**, Google; **Kevin Elphinstone**, University of New South Wales; **June Andronick**, Proofcraft; **David Cock**, ETH Zurich; **Philip Derrin**, Qualcomm; **Dhammika Elkaduwe**, University of Peradeniya; **Kai Engelhardt**; **Toby Murray**, University of Melbourne; **Rafal Kolanski**, Proofcraft; **Michael Norrish**, Australian National University; **Thomas Sewell**, University of Cambridge; and **Simon Winwood**, Galois, receive the **ACM Software System Award** for the development of the first industrial-strength, high-performance operating system to have been the subject of a complete, mechanically-checked proof of full functional correctness.

View the full list of ACM Awards

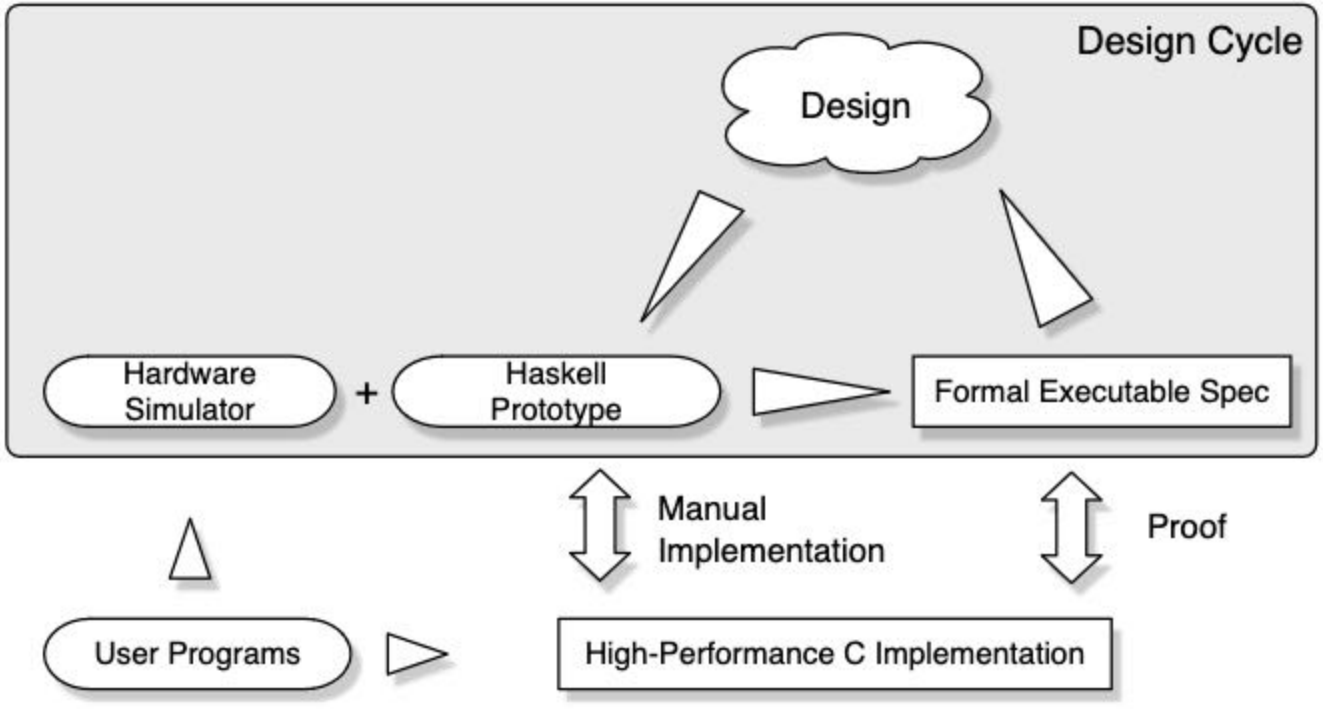
seL4



- Mikrokernel
- Varianten in Qualcomm-Modems und iPhone-Security-Enklave
- funktionale Korrektheit bewiesen (2009)
- Beweis 200.000loc

Gernot Heiser: The seL4 Microkernel - an Introduction

seL4: Entwicklungsprozess



Klein et al: seL4: Formal Verification of an OS Kernel

Klein et al: seL4: Formal Verification of an OS Kernel

Abstrakte Spezifikation

```
schedule ≡ do
    threads ← all_active_tcbs;
    ...
    switch_to_thread thread
od OR switch_to_idle_thread
```

Isabelle/HOL

Ausführbare Spezifikation

```
schedule = do
  action <- getSchedulerAction
  case action of
    ChooseNewThread -> do
      chooseThread
      setSchedulerAction ResumeCurrentThread
      ...
chooseThread = do
    r <- findM ... [minBound .. maxBound])
    when (r == ...) ...
chooseThread' prio = do
     q <- getQueue prio
     liftM isJust $ findM chooseThread'' q
chooseThread'' thread = do
     runnable <- isRunnable thread
     if not runnable then do
             tcbSchedDequeue thread
             return False
     else do
             switchToThread thread
             return True
```

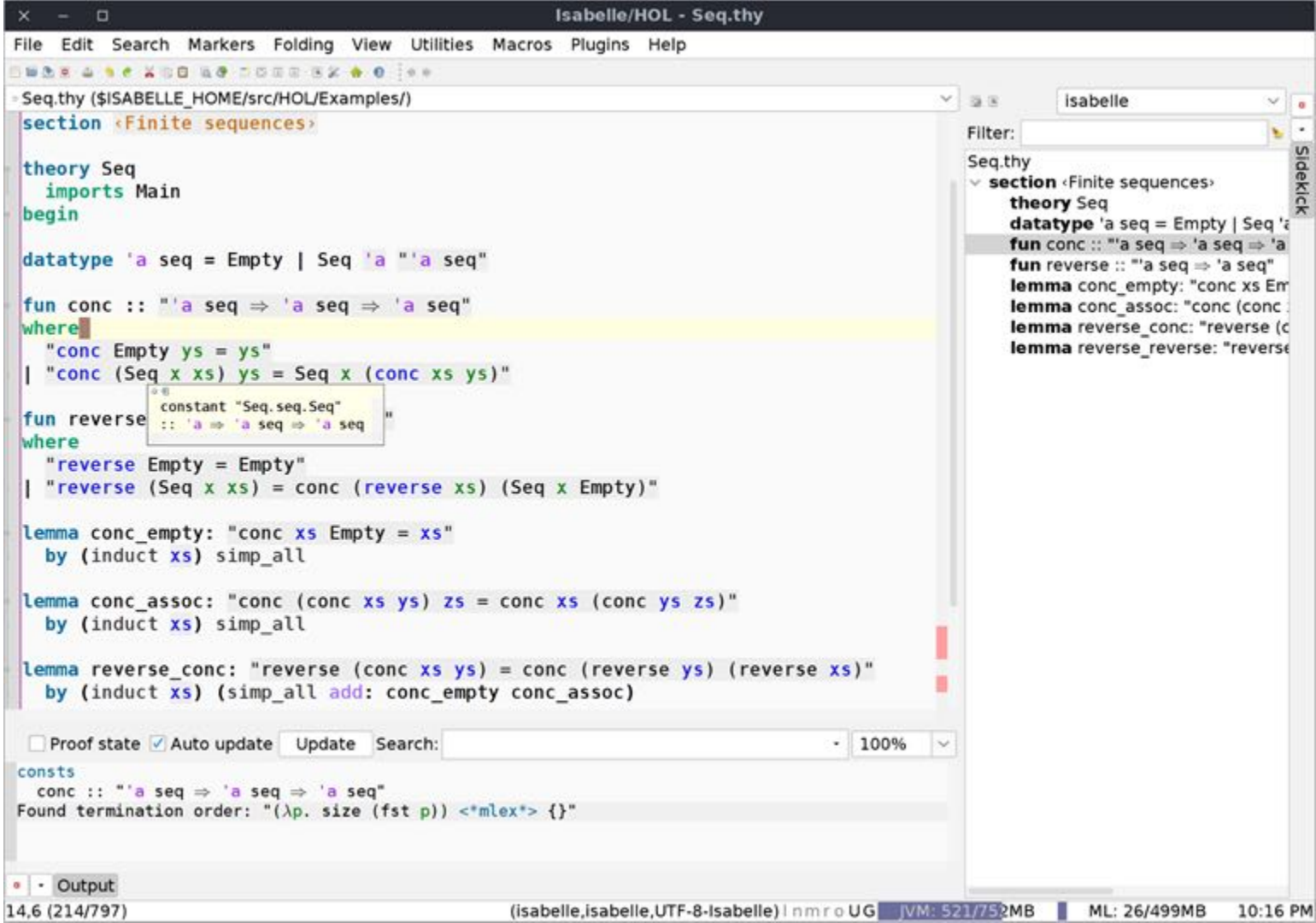
Haskell

Implementierung

```
void setPriority(tcb_t *tptr, prio_t prio) {
  prio_t oldprio;
  if(thread_state_get_tcbQueued(tptr->tcbState)) {
    oldprio = tptr->tcbPriority;
    ksReadyQueues[oldprio]
      tcbSchedDequeue(tptr, ksReadyQueues[oldprio]);
    if(isRunnable(tptr)) {
      ksReadyQueues[prio]
        tcbSchedEnqueue(tptr, ksReadyQueues[prio]);
    }
    else {
      thread_state_ptr_set_tcbQueued(&tptr->tcbState,
                                     false);
    }
  }
  tptr->tcbPriority = prio;
}
```

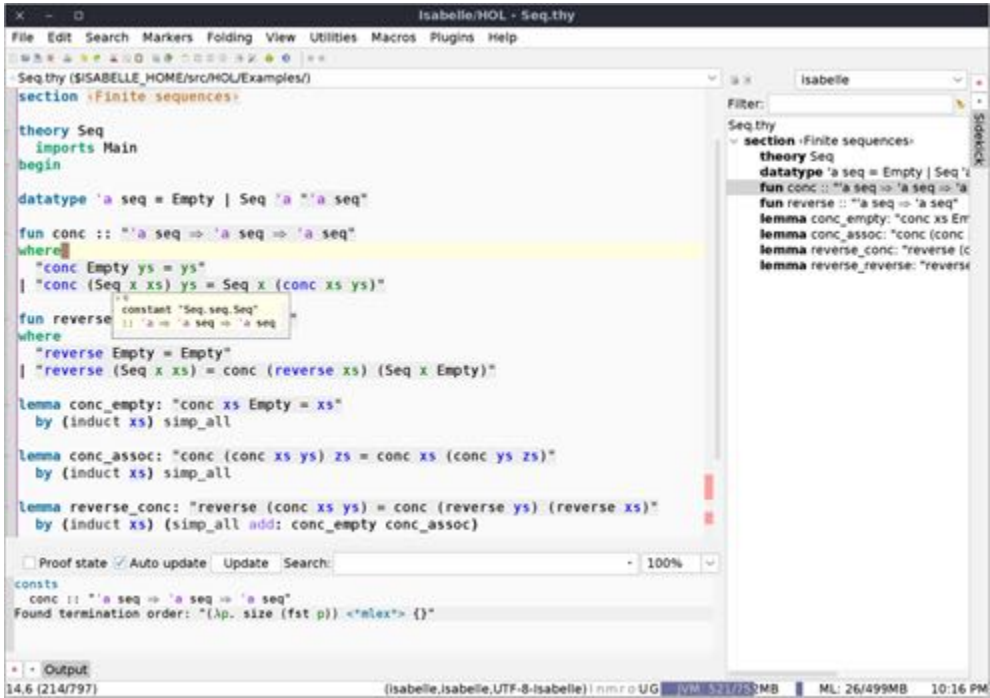
C

Isabelle


Isabelle/HOL - Seq.thy
File Edit Search Markers Folding View Utilities Macros Plugins Help
Seq.thy ($ISABELLE_HOME/src/HOL/Examples/)
section ‹Finite sequences›
theory Seq
imports Main
begin
datatype 'a seq = Empty | Seq 'a "'a seq"
fun conc :: "'a seq ⇒ 'a seq ⇒ 'a seq"
where
"conc Empty ys = ys"
| "conc (Seq x xs) ys = Seq x (conc xs ys)"
constant "Seq.seq.Seq"
:: 'a ⇒ 'a seq ⇒ 'a seq
fun reverse
where
"reverse Empty = Empty"
| "reverse (Seq x xs) = conc (reverse xs) (Seq x Empty)"
lemma conc_empty: "conc xs Empty = xs"
by (induct xs) simp_all
lemma conc_assoc: "conc (conc xs ys) zs = conc xs (conc ys zs)"
by (induct xs) simp_all
lemma reverse_conc: "reverse (conc xs ys) = conc (reverse ys) (reverse xs)"
by (induct xs) (simp_all add: conc_empty conc_assoc)
Proof state
Auto update
Update
Search:
100%
consts
conc :: "'a seq ⇒ 'a seq ⇒ 'a seq"
Found termination order: "(λp. size (fst p)) <*mlex*> {}"
Output
isabelle
Filter:
Seq.thy
section ‹Finite sequences›
theory Seq
datatype 'a seq = Empty | Seq 'a
fun conc :: "'a seq ⇒ 'a seq ⇒ 'a
fun reverse :: "'a seq ⇒ 'a seq"
lemma conc_empty: "conc xs Em
lemma conc_assoc: "conc (conc
lemma reverse_conc: "reverse (c
lemma reverse_reverse: "reverse
Sidekick
14,6 (214/797)
(isabelle,isabelle,UTF-8-Isabelle)
JVM: 521/752MB
ML: 26/499MB
10:16 PM

Isabelle

- “funktionale Programmiersprache + IDE”
- mathematische Syntax für Mengen, Algebra, Analysis, …
- Sprache für Beweise
- mathematisches Modulsystem
- unendlich erweiterbar



seL4: Aufwand

	Artefact	Effort (py)	Total (py)
Kernel Development	Haskell prototype	2.0	2.2
	C implementation	0.2	
Correctness Proof	Generic framework and tools	9.0	20.5
	Abstract specification	0.3	
	Executable specification	0.2	
	Refinement $\mathcal{M}_A \leftrightarrow \mathcal{M}_E$	8.0	
	Refinement $\mathcal{M}_E \leftrightarrow \mathcal{M}_C$	3.0	
Binary Verification	Verified Binary	2.0	2.0

Stevens: An Overview of the Verification of the seL4 Microkernel