UNIVERSITY OF TWENTE.

VerCors

CONTRACTS FOR CONCURRENT SOFTWARE WITH VERCORS

MARIEKE HUISMAN

UNIVERSITY OF TWENTE, NETHERLANDS

SAVES

CHEOPS
VERIFIED CONSTRUCTION OF CORRECT AND OPTIMISED PARALLEL SOFTWARE

Mercedes
MAXIMAL RELIABILITY OF CONCURRENT AND DISTRIBUTED SOFTWARE

Pallas

PLAN FOR TODAY

- Permission annotations for shared memory
- Fork-join concurrency
- Parallel blocks
- Reasoning about locks
- Reasoning about atomics


app.wooclap.com/events/ZIXONH/live-session
Exit
Neem deel aan dit Wooclap-evenement
1 Ga naar wooclap.com
2 Voer de code van het evenement in de bovenste banner in
Evenementcode
ZIXONH
Antwoorden per sms inschakelen
Deelnamelink kopiëren
wooclap
Stemmen / 1
Berichten
100%
0

WHY PERMISSION ANNOTATIONS?

- Concurrent programs: data on the heap is shared by threads
- Data races should be avoided (multiple threads accessing and updating the heap at the same moment)
- Permission annotations capture this:
  - Perm(x.f, 1) or Perm(x.f, write) – exclusive access
  - Perm(x.f, 1\2) or Perm(x.f, read) -- shared read-only access
- Permissions can be split and combined
- Soundness of the logic: total number of permissions to a location never more than 1

RGB.java

```
// r, g and b are heap locations.

class RGB{

    int r;
    int g;
    int b;

    void clear() {
        r = 0;
        g = 0;
        b = 0;
    }
}
```

-:--- RGB.java All (13,0) (Java/l Abbrev)
Beginning of buffer

RGB.java

2025-10-08-examples — -zsh — 82×27

```
m.huisman@UT181313 2025-10-08-examples % ~/o/Software/vercors-2.0.0/bin/vercors RGB.java
[INFO] Starting verification
========================================
At RGB.java:10:9:
----------------------------------------
    8
    9       void clear() {
               [-----
   10           r = 0;
                -----]
   11           g = 0;
   12           b = 0;
----------------------------------------
Insufficient permission to assign to field. (https://utwente.nl/vercors#assignFieldFailed)
========================================
m.huisman@UT181313 2025-10-08-examples %
```

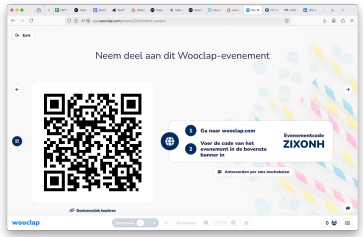
RGB-solution.java

```
// Permissions for ownership/access accounting

class RGB{

    int r;
    int g;
    int b;

    /*@
      @ requires Perm(r, write);
      @ requires Perm(g, write);
      @ requires Perm(b, write);
      @*/
    void clear() {
        r = 0;
        g = 0;
        b = 0;
    }
}
```

-:--- RGB-solution.java All (3,9) (Java/l Abbrev)

Wrote /Users/m.huisman/ownCloud/Documents/Research/VerCors/presentations/2025-10-08-examples/RGB-solution.java

Which permissions are needed here?



RGB-use.java

New Open Recent Revert Save Print Undo Redo Cut Copy Paste Search

RGB-use.java | RGB-use-solution.java | RGB-use-alt-solution.java

```
// Giving up and returning permissions


class RGB{

    int r;
    int g;
    int b;

    void clear() {
        r = 0;
        g = 0;
        b = 0;
    }

    void main() {

        clear();
        r = 255;
    }
```

-:--- RGB-use.java Top (10,18) (Java/l Abbrev)

Wrote /Users/m.huisman/ownCloud/Documents/Research/VerCors/presentations/2025-10-08-examples/RGB-use.java

RGB-use-solution.java

New Open Recent Revert Save Print Undo Redo Cut Copy Paste Search

RGB-use.java 4 | RGB-use-solution.java 5 | RGB-use-alt-solution.java 6

```
// Giving up and returning permissions

class RGB{

    int r;
    int g;
    int b;


    /*@ requires Perm(r, write);
      @ requires Perm(g, write);
      @ requires Perm(b, write);
      @ ensures Perm(r, write);
      @ ensures Perm(g, write);
      @ ensures Perm(b, write);
      @*/
    void clear() {
        r = 0;
        g = 0;
        b = 0;
    }

    /*@ requires Perm(r, write);
      @ requires Perm(g, write);
      @ requires Perm(b, write);
      @ ensures Perm(r, write);
      @ ensures Perm(g, write);
      @ ensures Perm(b, write);
      @*/
    void main() {
        clear();

        r = 255;
    }
}
```

-:--- RGB-use-solution.java All (30,17) (Java/l Abbrev)

WHAT ARE PERMISSIONS?

- Permissions are “resources”
- When you call a method, you give up the permissions specified in its requirements
- When you come back from a call, you get back the permissions specified by the postcondition
- Permissions for objects are created in its constructor

Often permission annotations are the same in pre- and postcondition:
Context keyword

Functional properties can only be given about state that you can access (i.e. have permission for)

RGB-use-alt-solution.java

```
// Giving up and returning permissions

class RGB{

    int r;
    int g;
    int b;


    /*@ context Perm(r, write);
      @ context Perm(g, write);
      @ context Perm(b, write);
      @ ensures r == 0 && g == 0 & b == 0;
      @*/
    void clear() {
        r = 0;
        g = 0;
        b = 0;
    }

    /*@ context Perm(r, write);
      @ context Perm(g, write);
      @ context Perm(b, write);
      @ ensures r == 255 & g == 0 & b == 0;
      @*/
    void main() {
        clear();

        r = 255;
    }
}
```

Changing context to requires makes the functional postconditions fail to verify

HOW TO SPECIFY THIS FUNCTION?

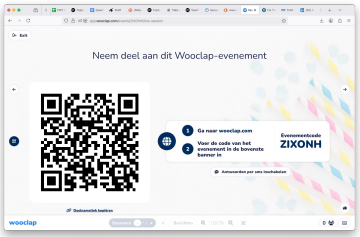
increase-annotated.java

New Open Recent Revert Save Print Undo Redo Cut Copy Paste Search Preferences Help

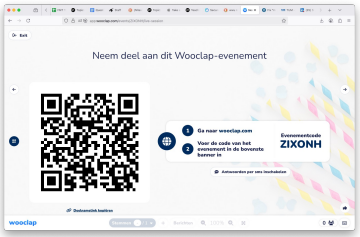
example5.java 7 | demo1.pvl 8 | increase.java 9 | increase-annotated.java

```
class Counter {
        int count;
    /*@
      @ requires Perm(count, 1);
      @ ensures Perm(count, 1\2);
      @ ensures count == \old(count) + n;
      @ */
        void incr(int n) {
                count = count + n;
        }

}
```

-:--- increase-annotated.java All (7,10) (Java/l Abbrev)



What will happen when we try to verify this?

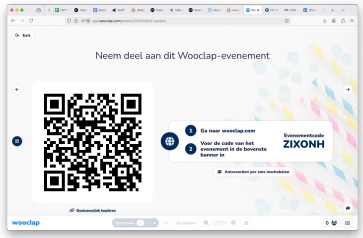


increase-twice.java

```
class Counter {
        int count;
    /*@
      @ requires Perm(count, 1);
      @ ensures Perm(count, 1\2);
      @ ensures count == \old(count) + n;
      @ */
        void incr(int n) {
                count = count + n;
        }

    /*@
      @ requires Perm(count, 1);
      @ ensures Perm(count, 1\2);
      @ ensures count == \old(count) + 2 * n;
      @ */
        void incr2(int n) {
                incr(n);
                incr(n);
        }

}
```

-:--- increase-twice.java All (19,25) (Java/l Abbrev)

What will happen when we try to verify this?



2025-10-08-examples — -zsh — 89×30

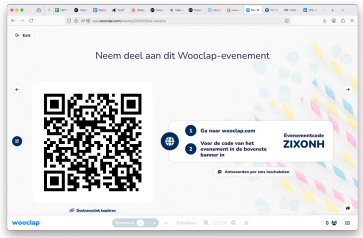
```
m.huisman@UT181313 2025-10-08-examples % ~/o/Software/vercors-2.0.0/bin/vercors increase-
twice.java
[INFO] Starting verification
========================================
 At increase-twice.java:19:3:
----------------------------------------
   17   void incr2(int n) {
   18           incr(n);
        [-------
   19           incr(n);
                -------]
   20   }
   21
----------------------------------------
[1/2] Precondition may not hold, since ...
----------------------------------------
 At increase-twice.java:4:18:
----------------------------------------
    2   int count;
    3      /*@
                        [--------------
    4        @  requires Perm(count, 1);
                         --------------]
    5        @  ensures Perm(count, 1\2);
    6        @  ensures count == \old(count) + n;
----------------------------------------
[2/2] ... there might not be enough permission to exhale this amount (https://utwente.nl/
vercors#preFailed:perm)
========================================
m.huisman@UT181313 2025-10-08-examples %
```

Clear-for.java

```
class Clear {
    /*@
      @ requires A != null;
      @ ensures A != null;
      @ context (\forall* int j; 0 <= j && j < A.length; Perm(A[j], write));
      @ ensures  (\forall int j; 0 <= j && j < A.length; A[j] == 0);
      @ */
  void clear(int[] A) {
      /*@
        @ loop_invariant A != null && 0 <= i && i <= A.length;
        @ loop_invariant (\forall int j; 0 <= j && j < i; A[j] == 0);
        @ */
    for (int i = 0; i < A.length; i++) {
      A[i] = 0;
    }
  }



}
```

-:--- Clear-for.java All (12,0) (Java/l Abbrev)

Which permissions need to be added to loop invariants?



Clear-for.java

2025-10-08-examples — -zsh — 89×30

```
m.huisman@UT181313 2025-10-08-examples % ~/o/Software/vercors-2.0.0/bin/vercors Clear-for
.java
[INFO] Starting verification
========================================
At Clear-for.java:12:59:
----------------------------------------
   10        /*@
   11          @ loop_invariant A != null && 0 <= i && i <= A.length;
                                                                  [----
   12          @ loop_invariant (\forall int j; 0 <= j && j < i; A[j] == 0);
                                                                   ----]
   13          @ */
   14           for (int i = 0; i < A.length; i++) {
----------------------------------------
There may be insufficient permission to access the array. (https://utwente.nl/vercors#arr
ayPerm)
========================================
m.huisman@UT181313 2025-10-08-examples %
```

Common pattern:
Context_everywhere

Clear-for.java

```
class Clear {
    /*@
      @ context_everywhere A != null;
      @ context_everywhere (\forall* int j; 0 <= j && j < A.length; Perm(A[j], write));
      @ ensures  (\forall int j; 0 <= j && j < A.length; A[j] == 0);
      @ */
  void clear(int[] A) {
      /*@
        @ loop_invariant A != null && 0 <= i && i <= A.length;
        @ loop_invariant (\forall int j; 0 <= j && j < i; A[j] == 0);
        @ */
    for (int i = 0; i < A.length; i++) {
      A[i] = 0;
    }
  }


}
```

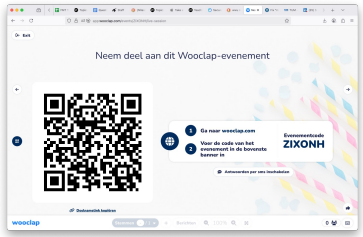
-:--- Clear-for.java All (11,1) (Java/l Abbrev)

FORK-JOIN CONCURRENCY

- Threads give away permissions when forking a thread
- When joining a thread, permissions are handed back to the joining thread
- Specified as contract of the run method
- If multiple threads can join, extra bookkeeping needed to ensure that permissions do not get duplicated (by means of join-token, which describes the share that the current thread will obtain)

FIBONACCI

- Which permissions for the run method?
- Which permissions for the constructor?
- Functional behaviour



fibonacci-bare.pvl — 2025-10-08-examples (git: master)

fibonacci.pvl | fibonacci-bare.pvl

```
class Fib {
  int input, output;

  run {
    if (input<2) {
      output = 1;
    } else {
      Fib f1 = new Fib(input-1);
      Fib f2 = new Fib(input-2);
      fork f1;  fork f2;
      join f1;  join f2;
      output = f1.output + f2.output;
    }
  }

  constructor(int n){
    input = n;
  }
}
```

Line: 22 | Vercors PVL | Tab Size: 4 | join

fibonacci.pvl — 2025-10-08-examples (git: master)

fibonacci.pvl | fibonacci-bare.pvl

```
pure int fib(int n)=n<2?1:fib(n-1)+fib(n-2);

class Fib {
  int input, output;

  requires Perm(input,read) ** Perm(output,write);
  ensures  Perm(input,read) ** Perm(output,write) ** output==fib(input);
  run {
    if (input<2) {
      output = 1;
    } else {
      Fib f1 = new Fib(input-1);
      Fib f2 = new Fib(input-2);
      fork f1;  fork f2;
      join f1;  join f2;
      output = f1.output + f2.output;
    }
  }

  ensures Perm(input,write) ** Perm(output,write) ** input==n;
  constructor(int n){
    input = n;
  }
}
```

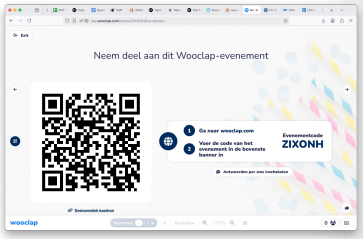
Line: 26 | Vercors PVL | Tab Size: 4 | join

PARALLEL BLOCKS

- Create n parallel threads, each executing the same code on their own data
- Massive parallelism (as in GPU's)
- Permissions should be distributed over the different threads
- Barriers for synchronization
- Barriers redistribute permissions

PARALLEL CLEAR

What is the contract per thread?



clear-bare.pvl — 2025-10-08-examples (git: master)

fibonacci.pvl | clear-par.pvl | clear-bare.pvl | fibonacci-bare.pvl

```
    context_everywhere A != null;
    context (\forall* int j; 0 <= j && j < A.length; Perm(A[j], write));
    ensures (\forall int j; 0 <= j && j < A.length; A[j] == 0);
    void clearPar(int[] A) {
        par (int tid = 0 .. A.length)
        {
            A[tid] = 0;
        }
    }

}
```

Line: 12 | Vercors PVL | Tab Size: 4 | par

PARALLEL CLEAR

clear-par.pvl — 2025-10-08-examples (git: master)

fibonacci.pvl | clear-bare.pvl | clear-par.pvl | fibonacci-bare.pvl

```
context_everywhere A != null;
context (\forall* int j; 0 <= j && j < A.length; Perm(A[j], write));
ensures (\forall int j; 0 <= j && j < A.length; A[j] == 0);
void clearPar(int[] A) {
    par (int tid = 0 .. A.length)
        requires Perm(A[tid], write);
        ensures Perm(A[tid], write);
        ensures A[tid] == 0;
    {
        A[tid] = 0;
    }
}
```

Line: 14 | Vercors PVL | Tab Size: 4 | par

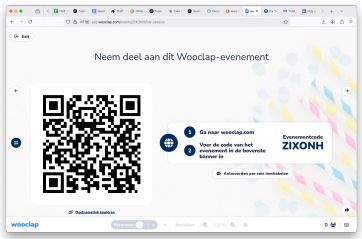
barrier-bare.pvl — 2025-10-08-examples (git: master)

barrier-bare.pvl | barrier-example.pvl

```
context_everywhere a != null && b != null && c != null;
context_everywhere tcount >= 0;
  context_everywhere a.length == tcount && b.length == tcount && c.length == tcount;
requires (\forall* int i; 0 <= i && i < tcount; Perm(a[i], write));
requires (\forall* int i; 0 <= i && i < tcount; Perm(b[i], write));
requires (\forall* int i; 0 <= i && i < tcount; Perm(c[i], 1\4)); // Perm(c[i], read) also possible (1)
void main(int tcount, int[] a, int[] b, int[] c) {
  par fwd (int tid=0..tcount)
    requires Perm(a[tid], write);
    requires Perm(b[tid], write);
    requires Perm(c[tid], 1\4); // Perm(c[tid], read) also possible (2)
  {
    b[tid]=c[tid];
    barrier(fwd) {
        // Barrier specification needed
    }
    if(tid>0) {
      a[tid]=b[tid-1]+b[tid];
    } else {
      a[tid]=b[tid];
    }
  }
}
```

Line: 23:4 | Vercors PVL | Tab Size: 4 | else

How to specify the effect of the barrier?



barrier-example.pvl — 2025-10-08-examples (git: master)

barrier-bare.pvl | barrier-example.pvl

```
context_everywhere a != null && b != null && c != null;
context_everywhere tcount >= 0;
  context_everywhere a.length == tcount && b.length == tcount && c.length == tcount;
requires (\forall* int i; 0 <= i && i < tcount; Perm(a[i], write));
requires (\forall* int i; 0 <= i && i < tcount; Perm(b[i], write));
requires (\forall* int i; 0 <= i && i < tcount; Perm(c[i], 1\4)); // Perm(c[i], read) also possible (1)
void main(int tcount, int[] a, int[] b, int[] c) {
  par fwd (int tid=0..tcount)
    requires Perm(a[tid], write);
    requires Perm(b[tid], write);
    requires Perm(c[tid], 1\4); // Perm(c[tid], read) also possible (2)
  {
    b[tid]=c[tid];
    barrier(fwd) {
      context 0 <= tid && tid < tcount;
      // losing all permissions:
      context  Perm(a[tid], write);
      requires Perm(b[tid], write);
      requires Perm(c[tid], 1\4); // Perm(c[tid], read) also possible (3)
      ensures  Perm(b[tid], 1\4); // Perm(b[tid], read) also possible (1)
      ensures  tid>0 ==> Perm(b[tid-1], 1\4); // Perm(b[tid-1], read) also possible (2)
    }
    if(tid>0) {
      a[tid]=b[tid-1]+b[tid];
    } else {
      a[tid]=b[tid];
    }
  }
}
```

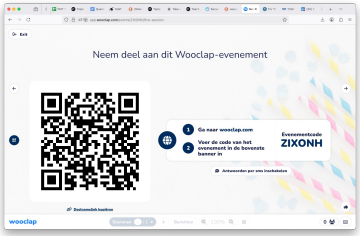
Line: 15:3 | Vercors PVL | Tab Size: 4 | barrier

REASONING ABOUT LOCKS

- Shared state protected by a lock
- Lock invariant specifies which shared state is protected by the lock
- Lock invariant has to be explicitly “committed” (typically done in the constructor)
- Acquiring the lock transfers the lock invariant to the owner of the lock
- Releasing the lock transfer the lock invariant back to the “neutral” state
- Releasing the lock means that you do not know anything about the shared state anymore
- Reentrant locks: special care needed to reason about when lock invariant is (not) transferred

PARALLEL SUM

- Notice the committed annotations
- Which data is protected by the lock?
- What is the lock invariant?
- Do we need other permission annotations?



parallel-sum-locks-bare.pvl — 2025-10-08-examp...

barrier-bare.pvl | barrier-...mple.pvl | parallel...are.pvl

```
class ArraySum {
    int sum;

    ensures committed(this);
    constructor() {
      commit this;
    }

    requires committed(this);
    context_everywhere A != null;
    void sum(int[] A) {
        par (int tid = 0 .. A.length)
        {
            lock this;
            sum = sum + A[tid];
            unlock this;
        }
    }
}
```

Line: 20 Verc... Tab Size: 4 unlock

parallel-sum-locks.pvl — 2025-10-08-examples (git: master)

barrier-bare.pvl | barrier-example.pvl | demo3a-func.pvl | parallel-...locks.pvl

```
lock_invariant Perm(this.sum, 1);
class ArraySum {
    int sum;

    ensures committed(this);
    constructor() {
      commit this;
    }

    requires committed(this);
    context_everywhere A != null;
    context (\forall* int i; 0 <= i && i < A.length; Perm(A[i], 1\2));
    void sum(int[] A) {
        par (int tid = 0 .. A.length)
            requires Perm(A[tid], 1\2);
            ensures Perm(A[tid], 1\2);
        {
            lock this;
            sum = sum + A[tid];
            unlock this;
        }
    }
}
```

Line: 23:2 | Vercors PVL | Tab Size: 4 | unlock

REASONING ABOUT ATOMICS

- Similar ideas to locks
- Atomic code blocks: invariant dynamically associated to it

parallel-sum-atomic-bare.pvl — 2025-10-08-examples (git: master)

barrier-bare.pvl | barrier-example.pvl | demo3a-func.pvl | parallel-s...c-bare.pvl

```
class ArraySum {
    int sum;

    context_everywhere A != null;
    context (\forall* int i; 0 <= i && i < A.length; Perm(A[i], 1\2));
    context Perm(this.sum, write);
    void sum(int[] A) {
        invariant inv(Perm(this.sum, write)) {
            par (int tid = 0 .. A.length)
                requires Perm(A[tid], 1\2);
                ensures Perm(A[tid], 1\2);
            {
                atomic(inv) {
                    sum = sum + A[tid];
                }
            }
        }
    }
}
```

Line: 20 | Vercors PVL | Tab Size: 4 | void

Can we also give a functional specification for this function?

parallel-sum-func.pvl — 2025-10-08-examples (git: master)

barrier-bare.pvl | barrier-...ple.pvl | parallel...mic.pvl | parallel...bare.pvl | parallel...cks.pvl | ● parallel...bare.pvl | paralle...nc.pvl

```
class ArraySum {
    int sum;

    requires A != null;
    requires 0 <= i && i <= A.length;
    requires (\forall* int j; 0 <= j && j < A.length; Perm(A[j], 1\2));
    pure int sum_contrib(int[] A, int i) =
      (i == A.length) ? 0 : A[i] + sum_contrib(A, i + 1);

    given int[A.length] contrib;
    context_everywhere A != null;
    context_everywhere contrib != null;
    context_everywhere contrib.length == A.length;
    context (\forall* int i; 0 <= i && i < A.length; Perm(A[i], 1\2));
    context (\forall* int i; 0 <= i && i < A.length; Perm(contrib[i], 1));
    context Perm(this.sum, write);
    requires (\forall* int i; 0 <= i && i < A.length; contrib[i] == 0);
    void sum(int[] A) {

        invariant inv(
                Perm(this.sum, write) **
                (\forall* int i; 0 <= i && i < contrib.length; Perm(contrib[i], 1\2)) **
                this.sum == sum_contrib(contrib, 0))
                {
            par (int tid = 0 .. A.length)
                requires Perm(A[tid], 1\2);
                requires Perm(contrib[tid], 1\2) ** contrib[tid] == 0;
                ensures Perm(A[tid], 1\2);
                ensures Perm(contrib[tid], 1\2) ** contrib[tid] == A[tid];
            {
                atomic(inv) {
                    sum = sum + A[tid];
                    contrib[tid] = A[tid];
                }
            }
        }
    }
}
```

Line: 1 | Vercors PVL | Tab Size: 4 | class

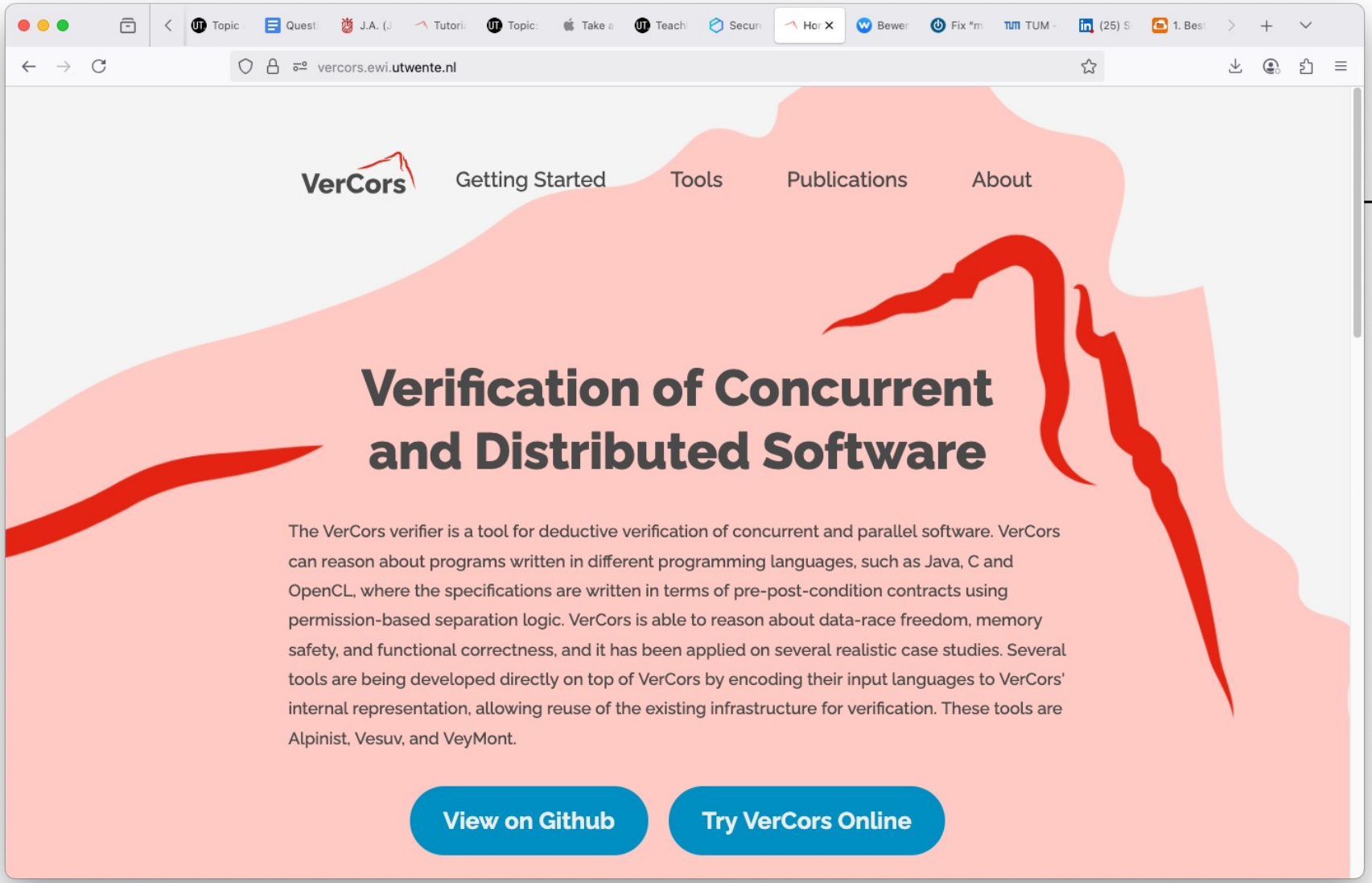
FORK-JOIN SUM

fork-join-sum.pvl — 2025-10-08-examples (git: master)

barrier-bare.pvl | barrier-...mple.pvl | parallel-...tomic.pvl | fork-joi...sum.pvl

```
class Worker {
    Main main;
    int val;

    requires m != null;
    ensures Perm(main, 1) ** main != null ** main == m;
    ensures Perm(val, 1) ** val == v;
    ensures idle(this);
    constructor(Main m, int v) {
        this.main = m;
        this.val = v;
    }

    context Perm(main, 1\2) ** main != null;
    requires committed(main);
    context Perm(val, 1\2);
    run {
        lock main;
        main.sum = main.sum + val;
        unlock main;
    }
}

lock_invariant Perm(this.sum, 1);
class Main {
    int sum;

    ensures committed(this);
    constructor() {
      commit this;
    }

    requires committed(this);
    void sum(seq<int> xs) {
        if (0 < |xs|) {
            Worker w = new Worker(this, xs.head);
            fork w;
            sum(xs.tail);
            join w;
        }
    }
}
```

Line: 34:25 | Vercors PVL | Tab Size: 4 | void


vercors.ewi.utwente.nl
VerCors
Getting Started
Tools
Publications
About
Verification of Concurrent and Distributed Software
The VerCors verifier is a tool for deductive verification of concurrent and parallel software. VerCors can reason about programs written in different programming languages, such as Java, C and OpenCL, where the specifications are written in terms of pre-post-condition contracts using permission-based separation logic. VerCors is able to reason about data-race freedom, memory safety, and functional correctness, and it has been applied on several realistic case studies. Several tools are being developed directly on top of VerCors by encoding their input languages to VerCors' internal representation, allowing reuse of the existing infrastructure for verification. These tools are Alpinist, Vesuv, and VeyMont.
View on Github
Try VerCors Online

VERIFYTHIS 2019: SPARSE MATRIX MULTIPLICATION

Sparse matrices represented using the coordinate list format: non-zero values of the matrix are stored in a sequence of triplets containing the row, the column, and the corresponding value

Algorithm which computes the multiplication of a vector of values (encoded as a sequence) with a sparse matrix

```
y <- (0, ..., 0)
for every element (r, c, v) of m do
  y (c) <- y (c) + x (r) * v
done
```

Verify that the concurrent algorithm does not exhibit concurrency issues (data-races, deadlocks, . . .).

sparse_matrix_mult.pvl — Assignment_8_rme (git: master)

```
/*
 * Performs a multiplication of sparse matrix a (M x N) with vector x (M) and stores the result in vector y (N).
 *
 * Dimensions:  |x| = m, |y| = n, |a| = (m, n)
 * Notation:    (row, column, value)
 */
context_everywhere \matrix(a, a.length, 3);             // Read-permission of 'a'
context_everywhere (\forall* int idx; 0 <= idx && idx < a.length; Perm(a[idx][0], read) ** Perm(a[idx][1], read) ** Perm(a[idx][2], read));
context_everywhere x != null && y != null;
context_everywhere Perm(x[*], read);                    // Read-permission on 'x'
context Perm(y[*], write);                              // Write-permission on 'y'
context_everywhere a.length > 0 && x.length > 0 && y.length > 0;
context_everywhere (\forall int i = 0 .. a.length; 0 <= a[i][0] && a[i][0] < x.length); // Well-formedness of 'a'
context_everywhere (\forall int i = 0 .. a.length; 0 <= a[i][1] && a[i][1] < y.length);
// ensures mat_mult_spec(a, x, y);
void vec_mat_mult_par(int[][] a, int[] x, int[] y) {

    // Create a thread for every element in the output-vector.
    par threads (int tid = 0 .. y.length)
        context Perm(y[tid], write);
        // ensures y[tid] == spec_elem(a, x, tid, a.length);
    {
        // Clear y[tid]
        y[tid] = 0;

        // assert y[tid] == spec_elem(a, x, tid, 0);

        // Iterate over all elements of the matrix
        loop_invariant 0 <= e && e <= a.length;
        loop_invariant Perm(y[tid], write);
        // loop_invariant y[tid] == spec_elem(a, x, tid, e);
        for (int e = 0; e < a.length; e++) {
            // Multiply and add all elements that contribute to y[tid] (i.e. all elements that belong to column 'tid' of the matrix).
            if (a[e][1] == tid) {
                y[tid] = y[tid] + (a[e][2] * x[a[e][0]]);
            }
            // assert y[tid] == spec_elem(a, x, tid, e+1);
        }
    }

    // assert (\forall int idx = 0 .. y.length; y[idx] == spec_elem(a, x, idx, a.length));

}
```

Line: 39:6 Vercors PVL Tab Size: 4 if