Hunting Superfluous Locks with Model Checking

Viet-Anh Nguyen[1], Wendelin Serwe[2], Radu Mateescu[2], and Eric Jenn[1]

[1] IRT Saint Exupéry, Toulouse

[2] CONVECS, Inria / LIG, Grenoble

Stefania's Inspiring Work

- **ACTL** (*Action-based Computation Tree Logic*) [De Nicola-Fantechi-Gnesi-Ristori-93]
  - Translation into modal μ-calculus [Stefania-et-al-92]
  - On-the-fly model checking for μ-ACTL (**FMC**)
  - Extensions with state and data-aware operators (**UMC**)
- Action-based logics and bisimulations
  - μ-ACTL characteristic formulas [Stefania-et-al-96]
  - Adequacy of μ-ACTL fragments w.r.t. branching bisimulation [Stefania-et-al-94]

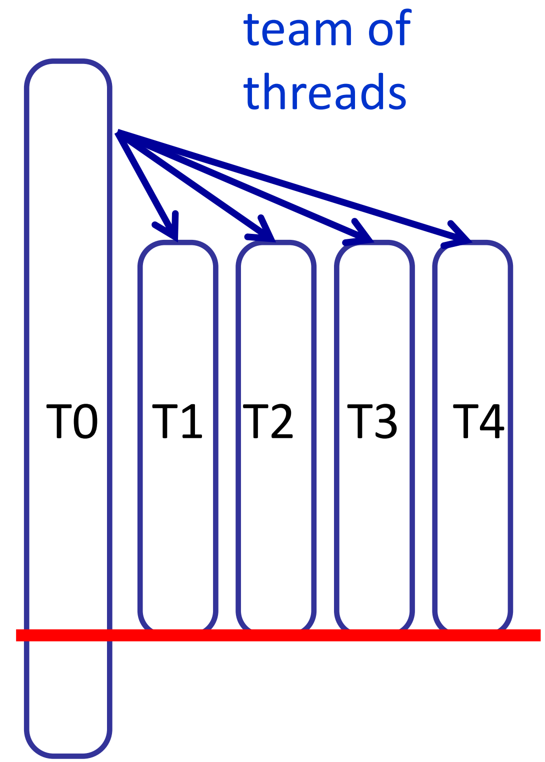
Context and Motivation

- Parallelization of software applications
  - Increase performance (many-core hardware architectures)
- Avionics domain
  - Safety-critical applications
  - Legacy code (sequential, optimized, safe)
- CAPHCA project (IRT Saint Exupéry, PIA)
  - Critical applications on predictable HPC architectures
- OpenMP: “lightweight” parallelization approach
  - Annotate sequential code with parallelization constructs
  - Parallel implementation by compiler and execution framework

Does not ensure absence of errors (data races, deadlocks, …)

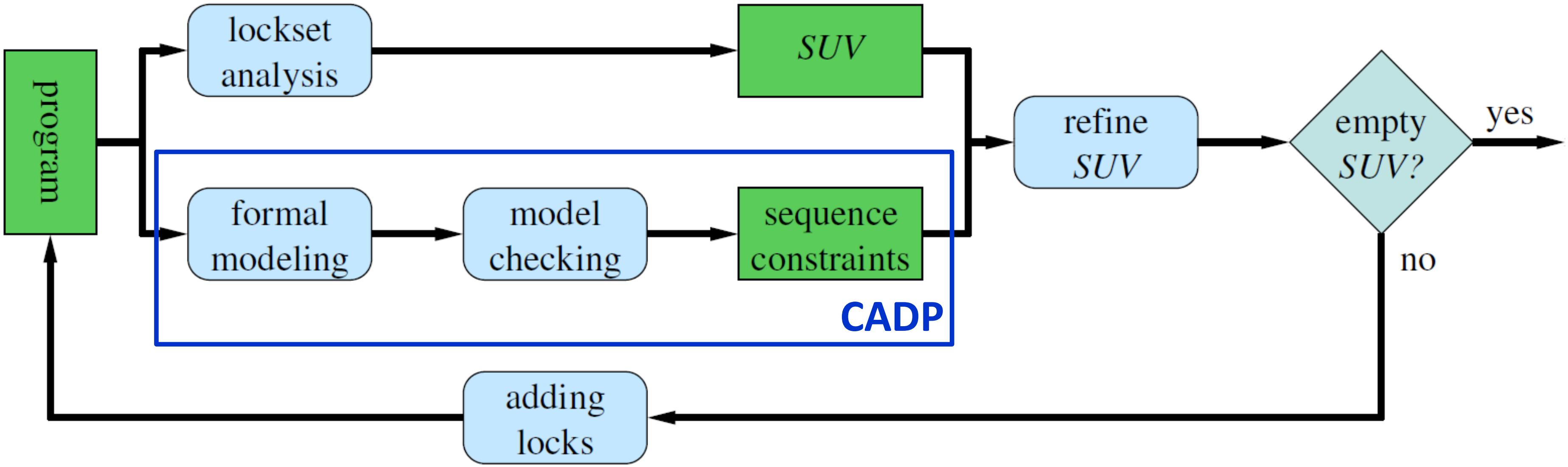
OpenMP by Example

```c
int a[5] = {2, 3, 4, 5, 6};
int main()
{
    int i, sum = 0; // work unit WU0
    #pragma omp parallel
    {
      #pragma omp for schedule (static, 1)
      for (i = 0; i < 5; i++)
      { // work units WU1 to WU5
        a[i] = a[i] * a[i];
        sum += a[i];
      }
      #pragma omp single
      sum += a[4]; // work unit WU6
    }
    return 0;
}
```



potential data races

Parallelization Workflow



Lockset Analysis

- Dynamic approach to detect potential data races
  - ERASER tool [Savage-Burrows-Nelson-97]
  - *Locking discipline*: every access to a shared variable is protected by (at least) one lock
  - Compute the candidate lockset *C* (*v*) for each program run
  - Safe (guarantees no data races)
  - Pessimistic (may report false data races)

Program	*locks_held*	*C(v)*
	{}	{mu1,mu2}
`lock(mu1);`		
	{mu1}	
`v := v+1;`		
		{mu1}
`unlock(mu1);`		
	{}	
`lock(mu2);`		
	{mu2}	
`v := v+1;`		
		{}
`unlock(mu2);`		
	{}	

Lockset Analysis (simple)

```
int a[5] = {2, 3, 4, 5, 6};
int main()
{
    int i, sum = 0; // work unit WU0
    #pragma omp parallel
    {
        #pragma omp for schedule (static, 1)
        for (i = 0; i < 5; i++)
        { // work units WU1 to WU5
            a[i] = a[i] * a[i];
            sum += a[i];
        }
        #pragma omp single
        sum += a[4]; // work unit WU6
    }
    return 0;
}
```

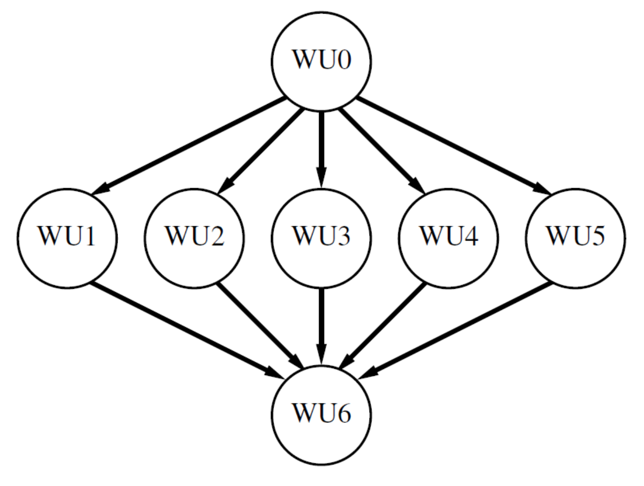
potential data races

➔ Detect spurious locks by model checking

OpenMP to LNT

- Work unit graph
  - Work unit: uninterruptible block of code
  - Static analysis of the OpenMP code (similar to control flow)
  - (Rough) abstraction of the OpenMP application
  - Encoded in LNT

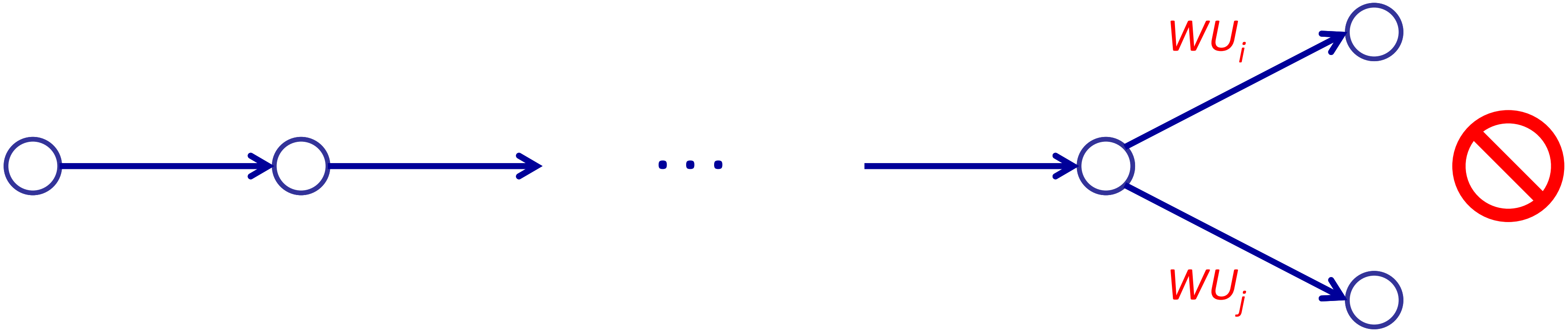
```
module OMP is
   process MAIN [WU0, WU1, WU2, WU3, WU4, WU5, WU6: none] is
      WU0;
      par
         WU1
      || WU2
      || WU3
      || WU4
      || WU5
      end par;
      WU6
   end process
end module
```



Sequentiality Detection

- Two working units WU_i and WU_j cannot execute concurrently (i.e., at the same time)
- ACTL formula (checked on the LTS of the WU graph):

Seq (WU_i, WU_j) = not **EF** $_{\text{true}}$ (**EX** $_{WUi}$ true and **EX** $_{WUj}$ true)



- On-the-fly verification using CADP / EVALUATOR
  - Use the ACTL translation to µ-calculus [Stefania et al 1992]

Insertion of Locks

```
int a[5] = {2, 3, 4, 5, 6};
int main()
{
    int i, sum = 0; // work unit WU0
    #pragma omp parallel
    {
        #pragma omp for schedule (static, 1)
        for (i = 0; i < 5; i++)
        { // work units WU1 to WU5
            #pragma omp critical
            {
                a[i] = a[i] * a[i];
                sum += a[i];
            }
        }
        #pragma omp single
        sum += a[4]; // work unit WU6
    }
```

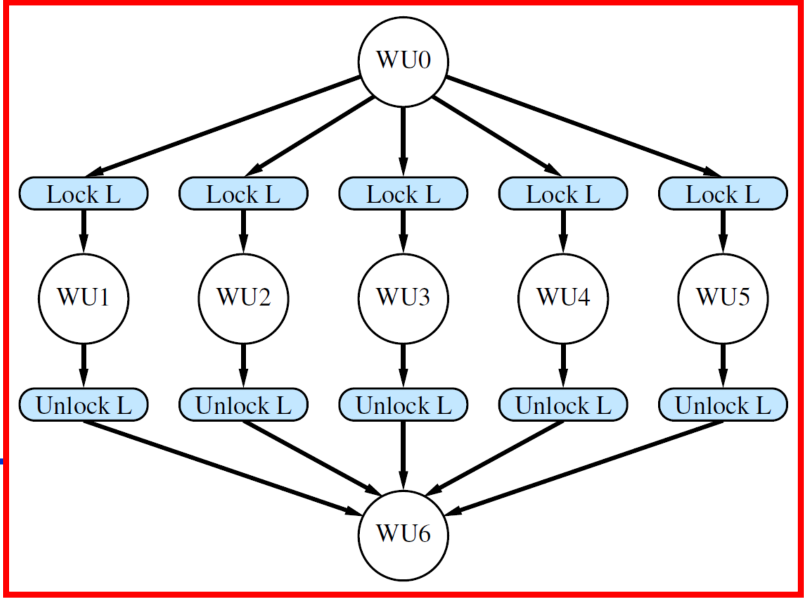
no need for lock

Seq (WU_0, WU_i)

Seq (WU_i, WU_6)

not ***Seq*** (WU_i, WU_j)

$i, j \in 1..5$



Conclusion and Perspectives

- Iterative method to ensure data race-free || programs
  - Combination of lockset analysis and model checking
  - OpenMP → work unit graph → LNT
  - Separation of concerns (parallelization and verification)
  - Tradeoff between quality of result and model checking cost
- Perspectives
  - Apply to other languages equipped with LNT translator (AADL)
  - Refinement and further analysis of LNT model (deadlocks, …)
  - Compositional verification of sequentiality property

    Seq (*WUi*, *WUj*): ACTL formula with strong and weak modalities
    [see the FM 2019 paper Lang-Mateescu-Mazzanti]