KCSE 2020
The 22nd Korea Conference on Software Engineering

A Model Projection Technique for Compositional Verification using Model Checking

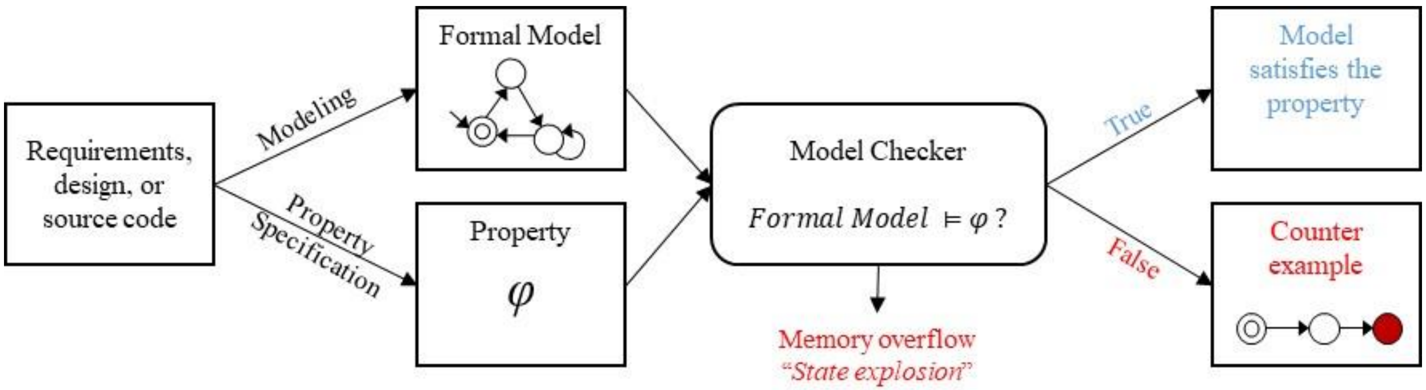
Dong-Ah Lee and Junbeom Yoo
KONKUK University

KONKUK UNIVERSITY | SOFTWARE ENGINEERING SOCIETY | DEPENDABLE SOFTWARE LABORATORY

Contents

Introduction

- Model checking

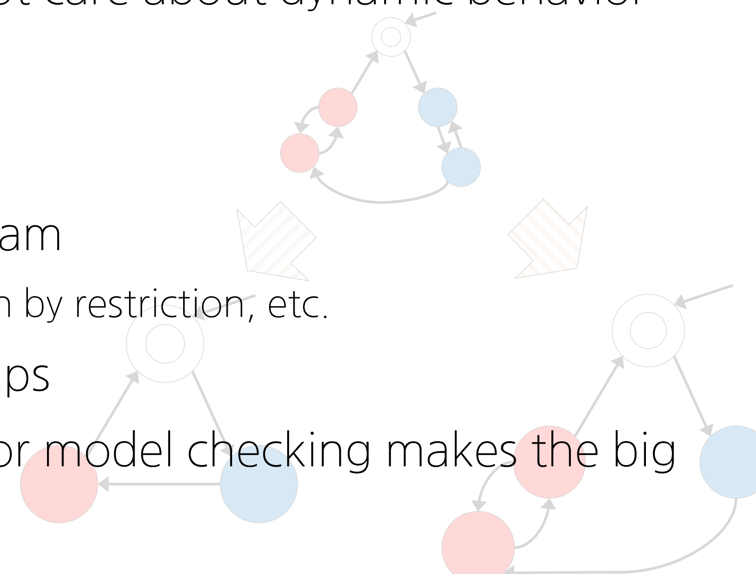


  - Checking whether a model meets a specification exhaustively and automatically
  - Demonstrating function correctness of hardware or software systems

Current Approaches & Limitations

- Code slicing (program slicing)
  - Program decomposition by analyzing their features
  - Slicing by static analysis
    - Data flow, control flow, etc.
  - Independent checking of the sliced programs, but no overall views
  - Code slicing by static analysis does not care about dynamic behavior

- Abstraction
  - Making an abstract model of a program
    - Abstraction on the variables , Abstraction by restriction, etc.
  - The bigger abstraction, the bigger gaps
  - Abstraction of large-scale software for model checking makes the big gap

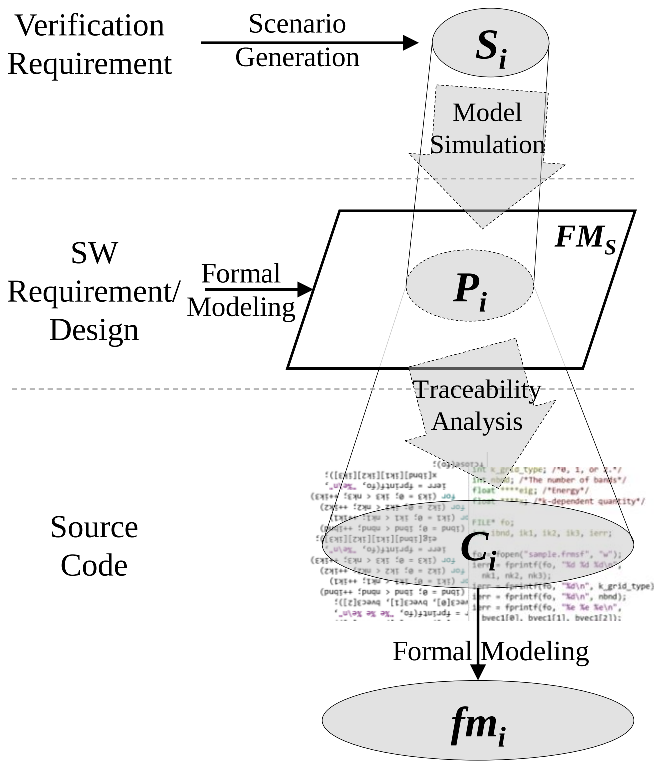


The Model Projection Technique

A technical process to identify relevant parts of source code with a verification purpose by executing a model of software

Identification of relevant parts by dynamic analysis

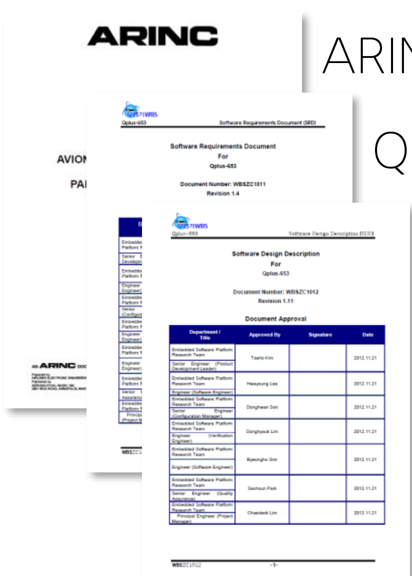
More concrete models derived from source code



1. Formal modeling at system-level (FM_S)
2. Generating simulation scenarios (S_i)
   Selecting the model checkers of fm_i
3. Simulating system-level model
4. Identifying running parts (P_1)
5. Tracing the parts to source code (C_i)
6. Model checking of the parts (fm_i)

Case Study

- Qplus-AIR
  - a RTOS for avionics complying the ARINC 653 by ETRI

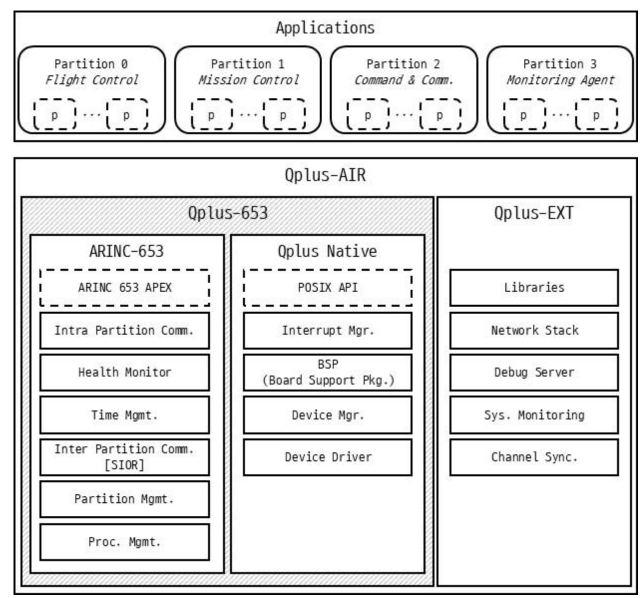


ARINC 653 Specification

Qplus-AIR SRD

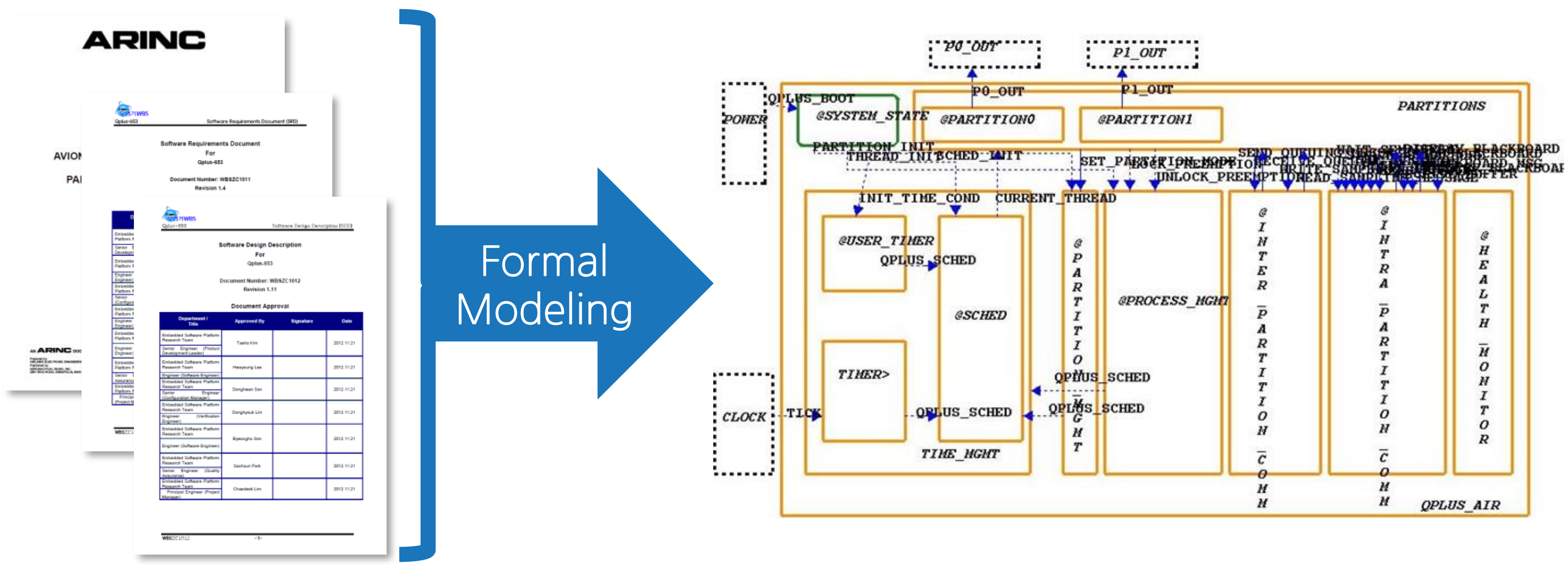
Qplus-AIR SDD

Qplus-AIR Architecture



Case Study

- Formal modeling at system-level using *Statemate*



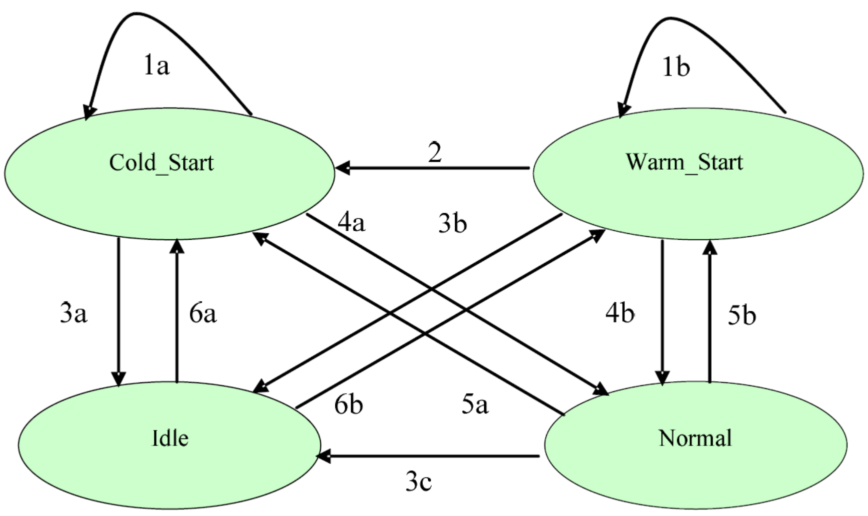
Case Study

- Simulation Scenarios
  - Change partition modes

2.3.1.4 Partition Modes

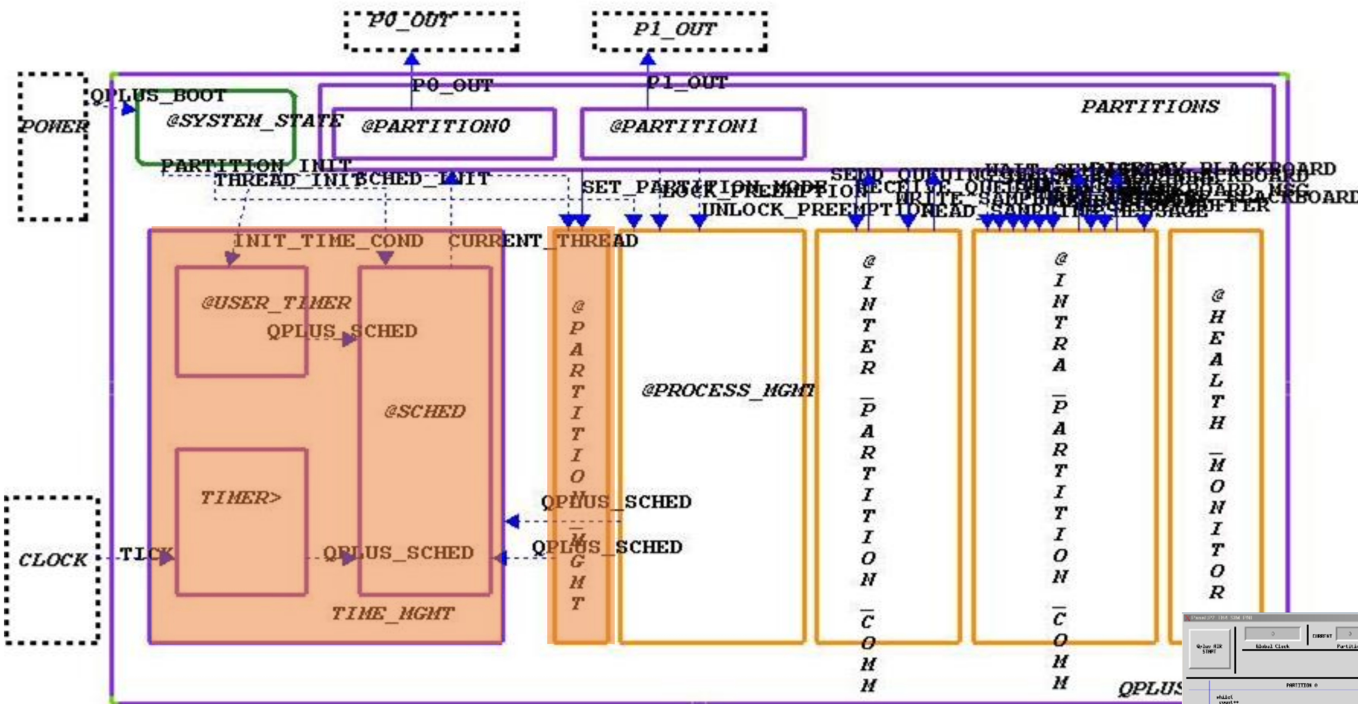
The SET_PARTITION_MODE service allows the partition to request a change to its operating mode. The Health Monitor, through its health monitoring configuration data, can also instigate mode changes. The current mode of the partition is available with the GET_PARTITION_STATUS service.

Partition modes and their state transitions are shown in the following diagram:

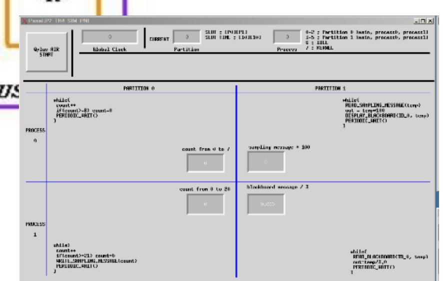


Case Study

- Simulation of the System-level Model with the Scenarios

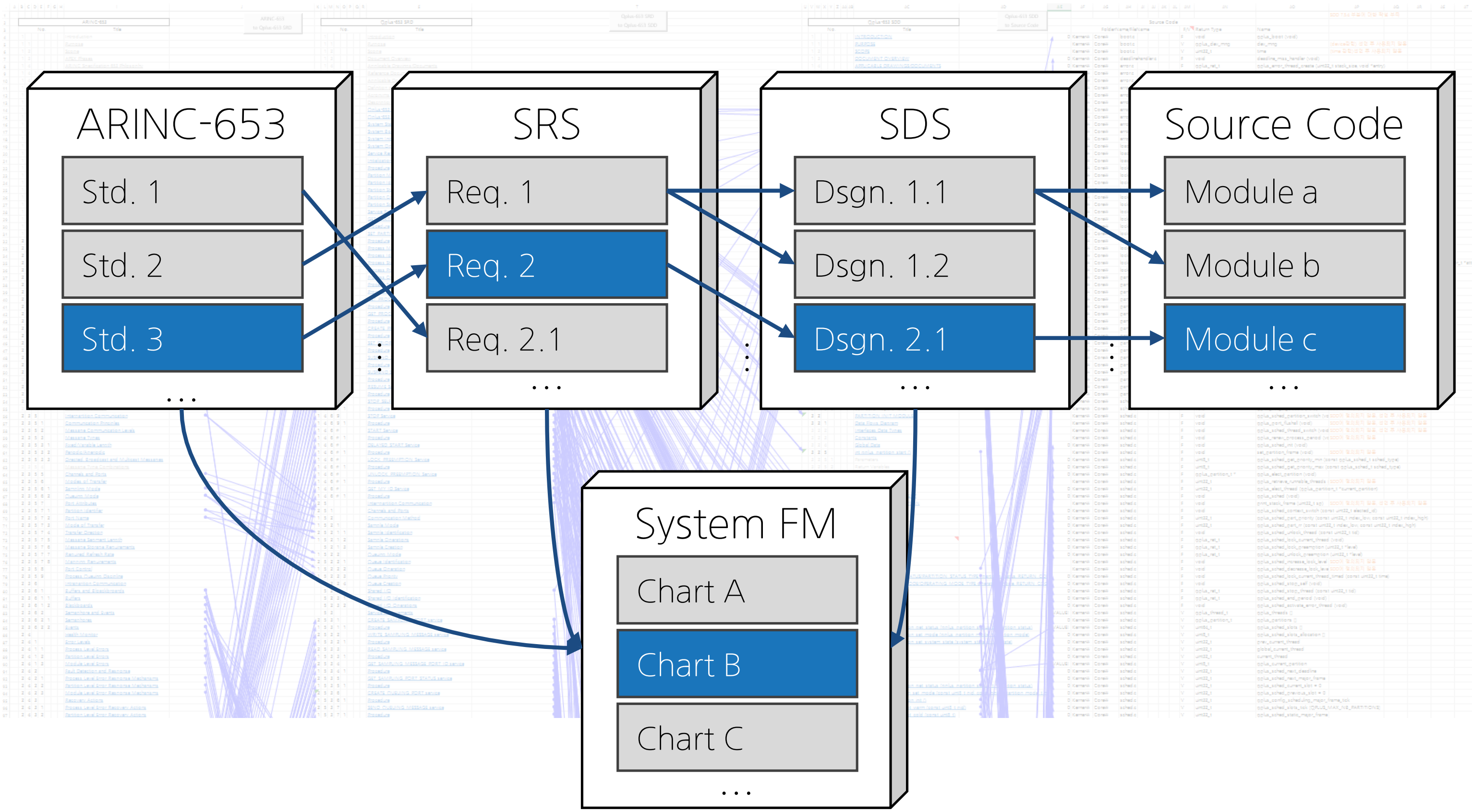


Simulation Panel



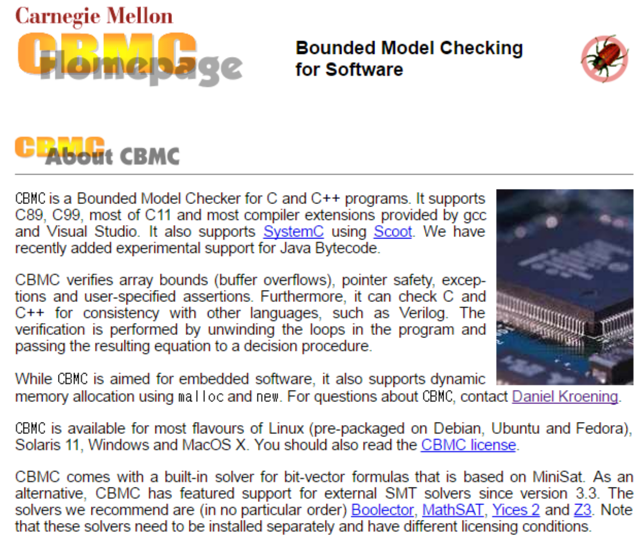
Case Study

- Traceability Analysis from Model to Source Code



Case Study

- CBMC
  - Bounded model checking for ANSI-C programs
  - Checking pointer safety, array bound, overflow, divided by zero, etc.
  - User defined assertion checking


Carnegie Mellon

CBMC Homepage

Bounded Model Checking for Software

About CBMC

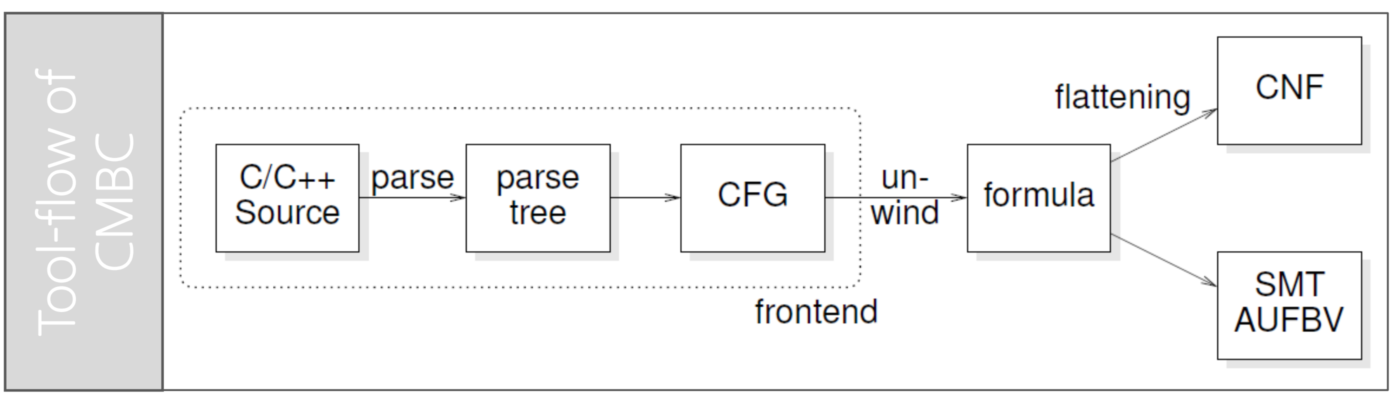
CBMC is a Bounded Model Checker for C and C++ programs. It supports C89, C99, most of C11 and most compiler extensions provided by gcc and Visual Studio. It also supports SystemC using Scoot. We have recently added experimental support for Java Bytecode.

CBMC verifies array bounds (buffer overflows), pointer safety, exceptions and user-specified assertions. Furthermore, it can check C and C++ for consistency with other languages, such as Verilog. The verification is performed by unwinding the loops in the program and passing the resulting equation to a decision procedure.

While CBMC is aimed for embedded software, it also supports dynamic memory allocation using malloc and new. For questions about CBMC, contact Daniel Kroening.

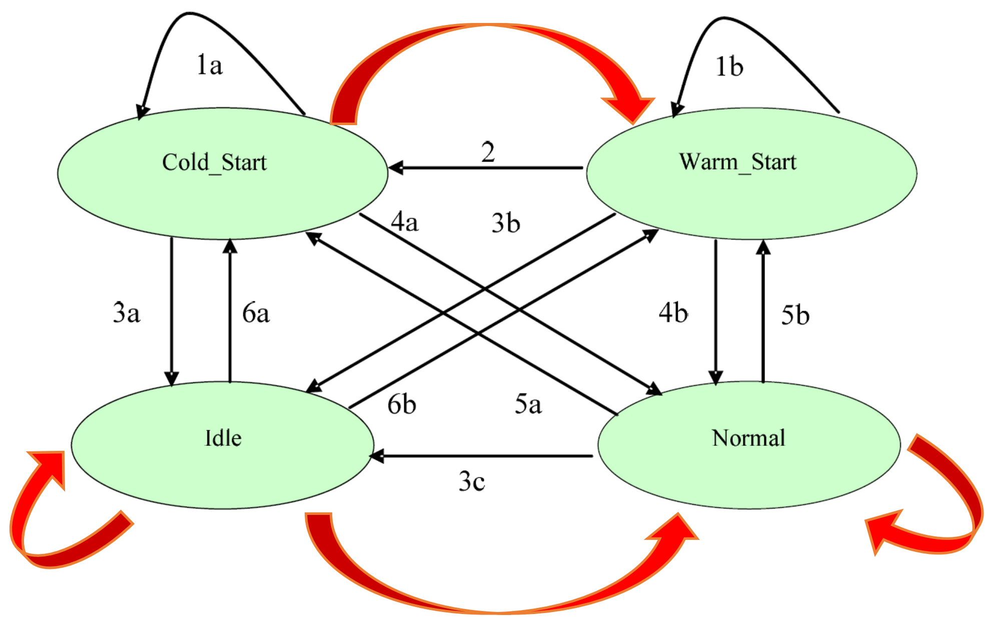
CBMC is available for most flavours of Linux (pre-packaged on Debian, Ubuntu and Fedora), Solaris 11, Windows and MacOS X. You should also read the CBMC license.

CBMC comes with a built-in solver for bit-vector formulas that is based on MiniSat. As an alternative, CBMC has featured support for external SMT solvers since version 3.3. The solvers we recommend are (in no particular order) Boolector, MathSAT, Yices 2 and Z3. Note that these solvers need to be installed separately and have different licensing conditions.



Case Study

- Property specification
  - All the defined and undefined mode changes



Case Study

- Found violations among undefined mode changes

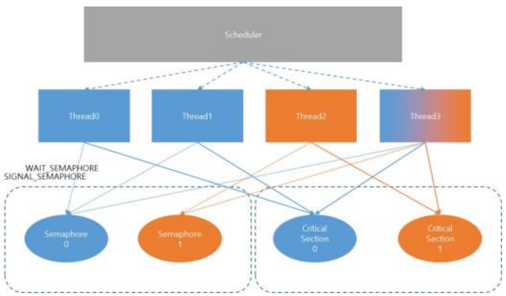
Verification Condition		Description
Predecessor Mode	Successor Mode	
COLD_START	WARM_START	Impossible
IDLE	IDLE	Possible
IDLE	NORMAL	Possible
NORMAL	NORMAL	Impossible

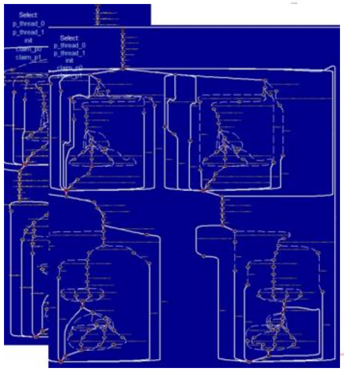
VIOLATION: IDLE → IDLE

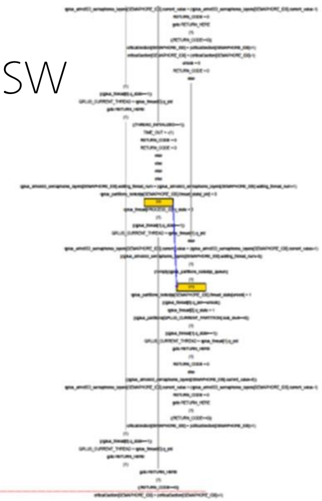
```
size of program expression: 352 steps
slicing removed 167 assignments
Generated 1 VCC(s), 1 remaining after simplification
Passing problem to propositional reduction
converting SSA
Running propositional reduction
Post-processing
Solving with MiniSAT 2.2.1 with simplifier
2554762 variables, 4046937 clauses
SAT checker inconsistent: instance is UNSATISFIABLE
Runtime decision procedure: 2.121s
VERIFICATION SUCCESSFUL
```

Other Case Studies

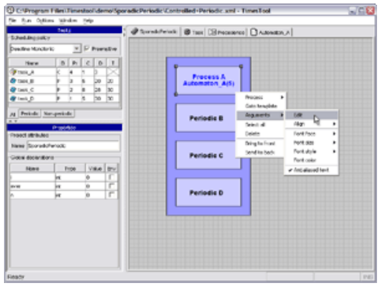
- Verification of communications
  - Model checker SPIN: the formal verification of multi-threaded SW applications

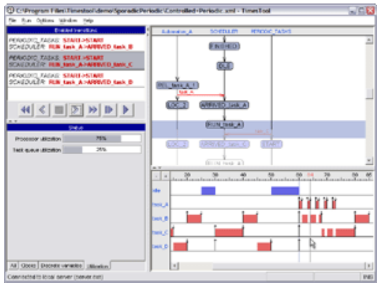


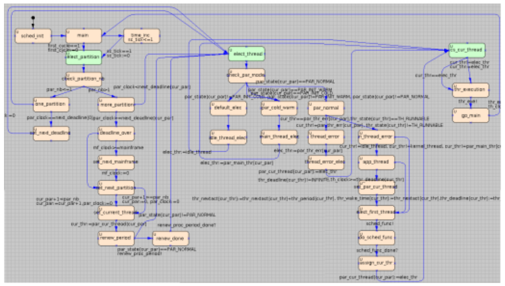




- Verification of a scheduler in Qplus-AIR
  - TIMES: Modeling, Verification, and Implementation of Embedded Systems







Conclusions

- The model projection technique to identify relevant parts with verification purposes
- Compositional verification with systematic analysis about relation and influence between components at a system level

FUTURE PLAN

- An elaborate model and a tool for traceability analysis to make projection much easier and quicker