# SPIN/Promela

EE W382V - Parallel Algorithms

Ari Bruck
Andy Yang

# Parallel Program Checking

Two main approaches:

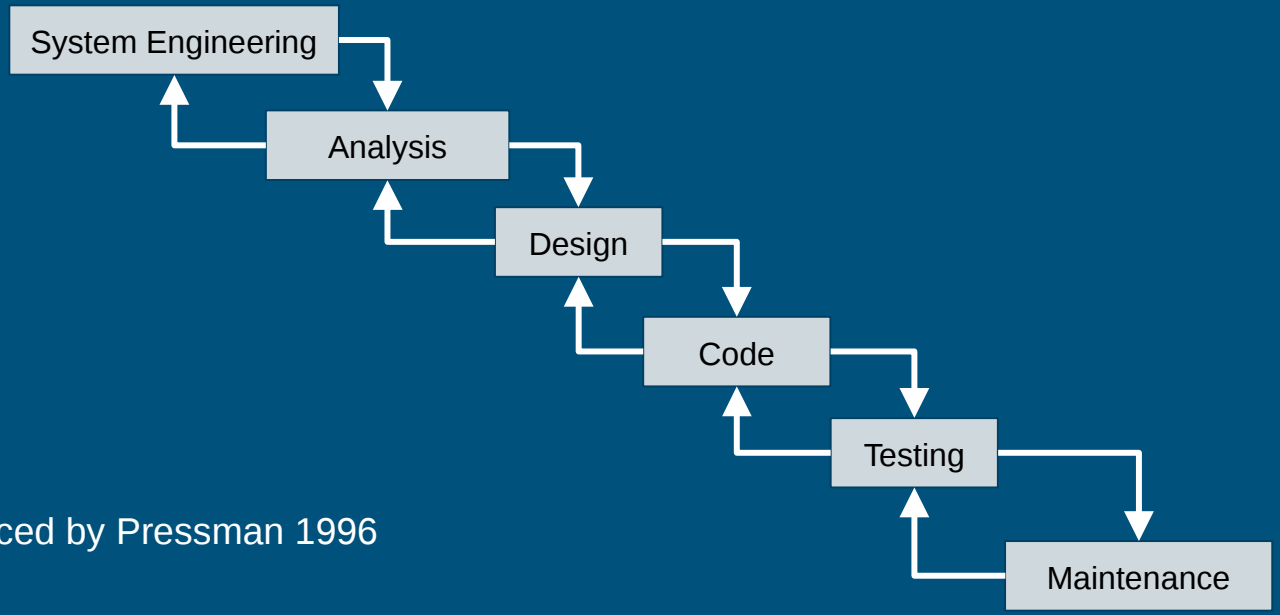
Debugging Approach: Try to find errors in the model

Verification Approach: Tries to prove correctness of a detailed model of the system under validation

The most effective approach is one that combines both

Automatic verification is not about proving correctness but rather finding bugs earlier in the system

# Classic Waterfall Model of System Development




Introduced by Pressman 1996

# Common Design Flaws in Parallel Programs

- Deadlock
- Livelock/Starvation
- Underspecification
  - Unexpected reception of messages
- Overspecification
  - “Dead” Code
- Violation of Constraints
  - Buffer Overrun

# SPIN/Promela

SPIN (**S**imple **P**romela **In**terpreter):

is a tool for analysing the logical consistency of concurrent systems, specifically of data communication protocols.

= state-of-the-art model checker, used by >2000 users

Concurrent systems are described in the modelling language called Promela.

Promela (= **Pr**otocol/**Pr**ocess **Me**ta **La**nguage)

Allows for the dynamic creation of concurrent processes.

Communication via message channels can be defined to be:

Synchronous (i.e. rendezvous), or

# How to Use It?

To verify a design, a formal model is built using PROMELA, Spin's input language.

PROMELA is a non-deterministic language, loosely based on Dijkstra's guarded command language notation and borrowing the notation for I/O operations from Hoare's CSP language.

# SPIN Modes

Spin can be used in four main modes:

1. as a simulator, allowing for rapid prototyping with a random, guided, or interactive simulations
2. as an exhaustive verifier, capable of rigorously proving the validity of user specified correctness requirements
3. as proof approximation system that can validate even very large system models with maximal coverage of the state space.
4. as a driver for swarm verification (a new form of swarm computing that can exploit cloud networks of arbitrary size), which can make optimal use of large numbers of available compute cores to leverage parallelism and search diversification techniques, which increases the chance of locating defects in very large verification models.

# Promela Model

Goal: to model finite systems

Constructs:

- Process declaration
- Channel declaration
- Variable declaration
  - arithmetic, relational, and logical operators similar to C and Java
- Type declaration
  - bit, bool, byte, short, int
  - Arrays

# Process

Executes concurrently with other processes

Communicates with other processes via channels or global variables

Each process keeps its own local variables

Two ways to create processes:

```
proctype Foo (int i) {..}          OR        active [2] proctype Foo (5) {..}

init { run Foo(5); }
```

# Statements

Each process consists of statements that are either executable or blocked

Inspired by Dijkstra's guarded command language

run Foo() will be blocked if process Foo() can not be created

An expression 2 < 3 will always execute

x < 20  will be checked

If the above condition evaluates to be false, the execution flow for that process will be blocked until other processes make X to be less than 20

# if-Statement and do-statement

when multiple choices evaluate to true, SPIN randomly chooses one choice

do-statement will loop and re-evaluate the choices until a *break* statement

```
if
:: choice1 -> statements
:: choice2 -> statements
fi;

do
:: choice1 -> statements
:: choice2 -> statements
od;
```

```
if
:: (x >=10) -> y =0
:: (x<= 100) -> y=0
:: (x<10) -> y =1
fi
```

# Channels

Communication between processes through channels:

- Synchronization
- send/receive messages

Channels are FIFO buffer

Sending - ch ! <expr1>, <expr2>,..

- Send executed when buffer is not full, else blocked

Receiving - ch ? <var1>, <var2>,..

- Parts of message received are stored into these variables

# Assert-statement

Useful for checking variables or conditions in Promela Model

When assertion expression evaluates to false, SPIN will exit with error message showing that an expression has been violated

```
proctype monitor()  { assert ( x < 99) }

proctype receiver () { assert (msg != ERROR)}
```

# Check for Mutual Exclusion

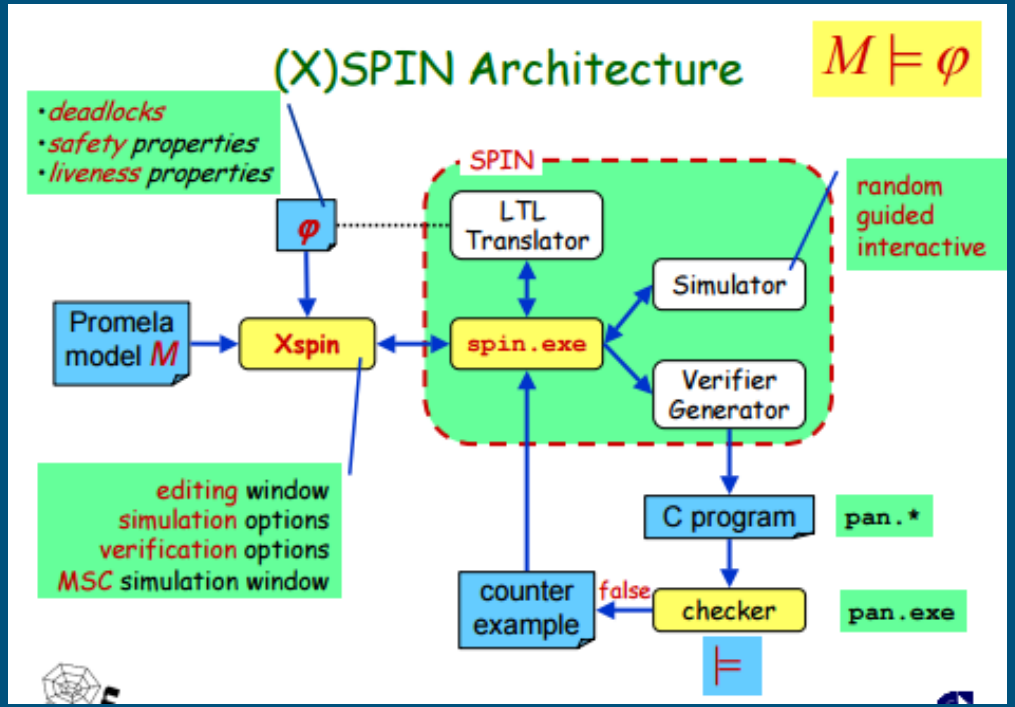
```
bit flag;
byte mutex;

active proctype P0() {
        flag !=1;
        flag =1;
        mutex++;
        mutex--;
        X =0;
}

active proctype P1() {
        flag!=1;
        flag=1;
        x == 0;
        mutex++;
        mutex--;
        y = 0;
}

active proctype monitor(){
        assert (mutex !=2); //WHEN BOTH PROCESSES GET TO CRITICAL SECTION, ASSERT EXPRESSION BECOMES  FALSE
}
```

# SPIN Architecture



# Promela in Action: DEMO