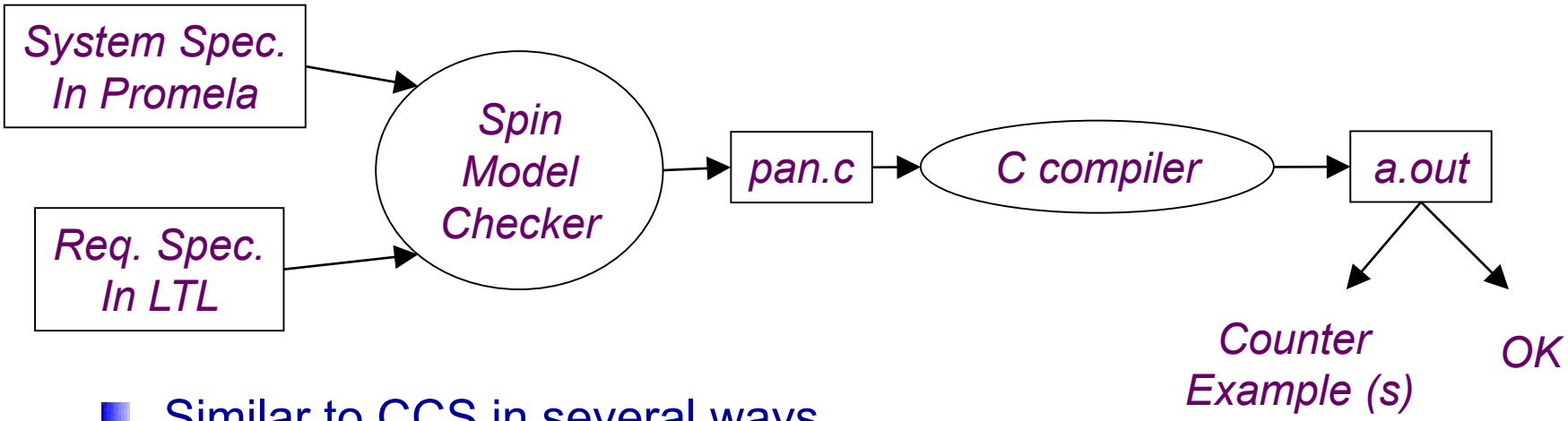
The Spin Model Checker : Part I

Overview of the Spin Architecture



- Similar to CCS in several ways
  - Promela allows a finite state model only
  - Asynchronous execution
  - Interleaving semantics for concurrency
  - 2-way process communication
  - Non-determinism
- Difference
  - Promela uses a special requirement language such as LTL while CCS uses CCS as both system spec lang and req. spec. lang
  - Promela provides (comparatively) rich set of constructs such as variables and message passing, dynamic creation of processes, etc

Overview of the Promela

```
byte x;
chan ch1= [3] of {byte};
```

Global variables (including channels)

```
active[2] proctype A() {
   byte z;
   printf("x=%d\n",x);
   z=x+1;
   ch1!z
}
```

Process (thread) definition and creation

```
proctype B(byte y) {
   byte z;
   ch1?z;
}
```

Another process definition

```
Init {
      run B(2);
}
```

System initialization

- Similar to C syntax but simplified
  - No pointer
  - No real datatype such as float or real
  - No functions
- Processes are communicating with each other using
  - Global variables
  - Message channels
- Process can be dynamically created
- Scheduler executes one process at a time using interleaving semantics

Process Creation Example

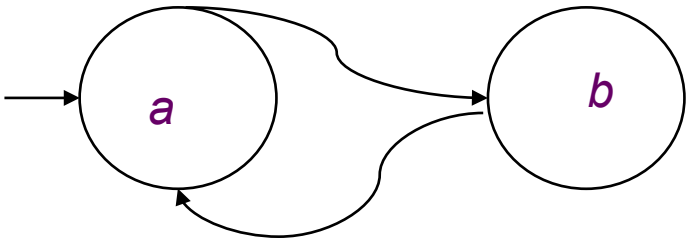
```
active[2] proctype A() {
   byte x;
   printf("A%d is starting\n");
}

proctype B() {
 printf("B is starting\n");
}

Init {
    run B();
}
```

- run() operator creates a process and returns a newly created process ID
- There are 6 possible outcomes due to non-deterministic scheduling
  - A0.A1.B, A0.B.A1
  - A1.A0.B, A1.B.A0
  - B.A0.A1, B.A1.A0
- In other words, process creation may not immediately start process execution

Variables and Types

- Basic types
  - bit
  - bool
  - Byte (8 bit unsigned integer)
  - short (16 bits signed integer)
  - Int (32 bits signed integer)
- Arrays
  - bool x[10];
- Records
  - typedef R { bit x; byte y;}
- Default initial value of variables is 0
- Most arithmetic (e.g.,+,-), relational (e.g. >,==) and logical operators of C are supported
  - bitshift operators are supported too.

- Promela spec generates only a finite state model because
  - Max # of active process <= 255
  - Each process has only finite length of codes
  - Each variable is of finite datatype
  - All message channels have bounded capability <= 255



a
ab
aba
abab
ababa
ababab
...
*/*buchi-automaton*/*

Basic Statements

- Each Promela statement is either
  - executable:
  - Blocked
- There are six types of statement
  - Assignment: always executable
    - Ex. `x=3+x`, `x=run A()`
  - Print: always executable
    - Ex. `printf("Process %d is created.\n",_pid);`
  - Assertion: always executable
    - Ex. `assert( x + y == z)`
  - Expression: depends on its value
    - Ex. `x+3>0`, `0`, `1`, `2`
    - Ex. `skip`, `true`
  - Send: depends on buffer status
    - Ex. `ch1!m` is executable only if `ch1` is not full
  - Receive: depends on buffer status
    - Ex. `ch1?m` is executable only if `ch1` is not empty

Expression Statements

- An expression is also a statement
  - It is executable if it evaluates to non-zero
  - 1 : always executable
  - 1<2:always executable
  - x<0: executable only when x < 0
  - x-1:executable only when x !=0
- If an expression statement in blocked, it remains blocked until other process changes the condition
  - an expression e is equivalent to while(!e); in C

- assert(expr)
  - assert is always executable
  - If expr is 0, SPIN detects this violation
  - assert is most frequently used checking method, especially as a form of invariance
    - ex. active proctype inv() { assert(x== 0);}
      - Note that inv() is equivalent to [] (x==0) in LTL with thanks to interleaving semantics

Program Execution Control

- Promela provides low-level control mechanism, i.e., goto and label as well as if and do
- Note that non-deterministic selection is supported
- else is predefined variable which becomes true if all guards are false; false otherwise

```
proctype A() {
    byte x;
    starting:
    x= x+1;
    goto starting;
}
```

```
proctype A() {
    byte x;
    if
    :: x <= 0 -> x=x+1
    :: x == 0 -> x=1
    fi
}
```

```
proctype A() {
    byte x;
    do
    :: x <= 0 -> x=x+1;
    :: x == 0 -> x=1;
    :: else -> break
    od
}
```