Concurrent Systems

3D4 ⟷ CS2016

Operating Systems

Andrew Butterfield
ORI.G39, Andrew.Butterfield@scss.tcd.ie

Trinity College Dublin
Coláiste na Tríonóide, Baile Átha Cliath
The University of Dublin

with thanks to Mike Brady

Promela Variables

- Basic types
  - bit e.g. turn=1; range: [0..1]
  - bool e.g. flag; [0..1]
  - byte e.g. counter; [0..255]
  - short e.g. s; [-2^{15}.. $2^{15}-1$]
  - int e.g. msg; [-2^{31}.. $2^{31}-1$]
- Default initial value of basic variables (local and global) is 0.
- Most arithmetic, relational, and logical operators of C/Java are supported, including bitshift operators.

Promela Variables (2)

- Arrays
  - Zero-based indexing
- Records ("structs" in C/Java)
  - typedefs

```
typedef Record {
  short f1;
  byte f2;
}
```

Statements

- A statement is either
  - *executable*: the statement can be executed immediately.
  - *blocked*: the statement cannot be executed.
- An assignment is always executable.
- An expression is also a statement; it is executable if it evaluates to non-zero. E.g.
  - 2 < 3 always executable
  - x < 27 only executable if value of x is smaller 27
  - 3 + x executable if x is not equal to –3

Statements (2)

- The skip statement is always executable.
  - it "does nothing", only changes the process counter
- A run statement is only executable if a new process can be created (the total number of processes is bounded by 255).
- A printf statement is always executable (but ignored, i.e. not considered, during verification).

```
int x;
proctype Aap()
{
  int y=1;
  skip;
  run Noot();
  x=2;
  x>2 && y==1;
  skip;
}
```

Statements (3) — assert

- Format: assert(<expr>);
- The assert statement is always executable.
- If <expr> evaluates to zero, SPIN will exit with an error, as the *<expr> has been violated.*
- Often used within Promela models, to check whether certain properties are valid in a state.

If Statement

```
if
 :: (n % 2 != 0) -> n=1
 :: (n >= 0)     -> n=n-2
 :: (n % 3 == 0) -> n=3
 :: else         -> skip
fi
```

- Each :: introduces an alternative followed by convention with a guard statement
  - if the guard is executable, then the alternative is executable, and one executable alternative is non-deterministically chosen.
    - The optional else becomes executable if none of the other guards are executable.
- If no guard is executable, the if statement blocks.

Do Statement

- Same as the If statement, but it repeats the choice at the end.
- Use the break statement to move on to the next sequential statement.

```
do
 :: choice1 -> stat1.1; stat1.2; stat1.3; …
 :: choice2 -> stat2.1; stat2.2; stat2.3; …
 :: …
                :: choicej -> break;
 :: choicen -> statn.1; statn.2; statn.3; …
od;
```

Atomic Statement

```
atomic { stat1; stat2; ... statn }
```

- An `atomic{}` statement can be used to group statements into an atomic sequence;
- all statements are executed in a single sequence (no interleaving with statements of other processes), though each step is taken.
- The statement is executable if `stat1` is executable
- If a `stat i` (with i>1) is blocked, the "atomicity token" is temporarily lost and other processes may do a step.

d_step (d = deterministic?)

```
d_step { stat1; stat2; ... statn }
```

- A `d_step` is a more efficient version of atomic:
  - No intermediate states are generated and stored.
  - It may only contain deterministic steps.
- It is a run-time error if `stat i` (`i>1`) blocks.

Process Execution / Evaluation Semantics

- Promela processes execute concurrently.
  - Non-deterministic scheduling of the processes.
- Processes are interleaved (statements of different processes do not occur at the same time).
  - exception: rendezvous communication. (We're not using this).
- All statements are atomic; each statement is executed without interleaving with other processes.
- Each process may have several different possible actions enabled at each point of execution.
  - one choice is made, non-deterministically.

Model Checking

- Model checking tools automatically verify whether a property holds in a given (finite-state) model of a system.
- Safety Properties
  - "Something bad never happens."
    - SPIN tries to find a path to the bad thing; if not found, the property is satisfied.
- Liveness Properties
  - "Something good always happens eventually."
    - SPIN tries to find an infinite loop in which the good thing doesn't happen; if not found, the property is satisfied.

The Closed World Assumption

- The model seeks to establish the truth of something by trying exhaustively and failing to prove the negation of it.
  - e.g. to prove that mutex holds, it tries to prove that it doesn't hold. If it fails to do so, it is taken as proof that the mutex holds.
- This only corresponds to what we normally understand to be "proof" if it is assumed that the model checker knows everything (i.e. can prove everything provable) about the system -- it's called the "Closed World Assumption."

Promela Programs

- Of course, we can execute Promela programs
- An interpreter will generate a *scenario*.
- We can use this to help get an intuitive understanding of the system described.
- Not useful for proving properties of the system.
- For that, we need *verification*.