Promela

Cheatsheet

Functions and Initialization

Promela provides three different abstractions for functions:

- `init`: Used to initialize a Promela model:

```
init {
    printf("Hello World\n")
}
```

- `proctype`: A simple process that can take parameters (except arrays)

```
proctype myProc(int p){...}
```

the process above must be run from e.g. an `init` function with `run myProc(42);`. The `run` command start the procedure and returns immediately.
Alternatively we can also start procedures right from the initial system state using the active keyword:

```
active proctype myProc(int p){...}
```

or start `N` processes of the type `myProc`.

```
active [N] proctype myProc(int p){...}
```

If we pass multiple variables we seperate them with a semicolon and not a comma!

- `inline`: A macro that gets copy-pasted. Doesn't do any simulation but just replaces text for a symbolic name with possible parameters:

```
inline swap(a, b) {
    int tmp;
    tmp = a;
    a = b;
    b = tmp;
}
```

Note that we do not have a return value and cannot use recursion as well as we do not have a new variable scope as we are literally copy-pasting.

Synchronization & Assertions

Atomic Block

To bundle multiple instructions together and have them execute atomically, we can bundle them as follows:

```
atomic{
    printf("hello ");
    printf("world\n");
}
```

Guards

We can suspend the execution until a condition is true. This is called a guard and is a boolean expression as a statement. The statement thereafter will only be executed if the guard evaluates to true:

```
time == 42;
printf("it is time\n");
```

Assertion

If we want to model check a program we can add assertions. The model checker will then check all combinations for a violation of the assertions:

```
proctype finish(){
    finished == 0; /* wait until finished */
    assert(n == 100);
}
```

Control Flow

If-Statement

One of the guard that is true is chosen non-deterministically. One (and only one) option can be `else` which is taken if no other match. If no conditions matched spin will block until a matching one could be found.

```
if
:: x > 0 -> x--
:: x < 0 -> x++
:: x > 100 -> x += 100
:: else  -> break
fi
```

Do-Statement

This loop construct executes an option non-deterministically until a `break` or `goto` is found.

```
do
:: count++
:: count--
:: (count == 0) -> break
od
```

Declarations

Constants

Define a constant to be replaced: The two are completely equivalent:

```
mtype = { ack, nak, err, next, accept };
```

```
#define ack 5
#define nak 4
#define err 3
#define next    2
#define accept  1
```

Structures

```
typedef vector {int x; int y};
```

Variable

Can be global or scoped:

```
bit counter;
byte counter2 = 2;
```

We can choose between `bit, bool, byte, short, int` as well as arrays which we create like in C: `int array[2]` **Built in Primitives**

`len(), empty(), nempty(), nfull(), full(), run eval() enabled() pcvalue()` are built in functions. Furthermore we have the standard C operators.
A special addition is the `_pid` variable giving me the process id of a single process an the `_nr_pr` giving me the total number of processes.

Channels

We can create a channel using

```
chan ch = [d] of {mtype, int}
```

That channel takes two parameters, an mtype and an int. The `d` is the buffer size. If it is unbuffered (d=0) we can use the channel for synchronization as both threads have to rendevouz to exchange a message. Otherwise we will just place it in the buffer.

- Receiving: `ch ? req, n;` receives a `req` mtype of arbitrary value `n`.

- Sending: `ch ! ack, 1;` send an `ack` mtype with value `1`.

Sources

1. ETH lecture slides by Prof. Peter Müller
2. SPIN documentation
3. A tutorial introduction to shared memory concurrency by K.V.S. Prasad (Chalmers)