Sistema di allarme.

Ogni stanza ha il suo allarme, c'è la persona anziana che è a letto e se ha bisogno può premere un pulsante di allarme.

Questo pulsante chiude un circuito, che andrà a far suonare una campanella ed accendere una lampadina presso la stanza degli infermieri. (semplice implementazione analogica)

Problema: le stanze potrebbero essere di gran numero, per esempio 50.

Ipotizziamo che un anziano suona l'allarme e che la lampadina sia una sola:

Questa lampadina sta suonando e continua a suonare per un minuto, poi due minuti, fino a quando l'infermiera non silenzia quell'allarme, premendo un bottone (quindi lo prende in carico e va dall'anziano).

PROBLEMA: se nei minuti in cui l'allarme suonava qualcun altro avesse chiamato l'allarme, cosa accadrebbe?

- ⇨ È una mutua esclusione sull'insieme lampadina-campanella. (Suona e si accende)
- ⇨ Mentre è utilizzata da qualcuno, non può essere utilizzata da un altro, altrimenti quella richiesta di utilizzo verrebbe perduta. (*In un sistema senza memoria*)

Se fosse un puro circuito, *quindi senza memoria*, quando la caposala/infermiera annulla l'allarme e prende in carico quel paziente, a quel punto un altro potrebbe far suonare l'allarme.

Si può vedere come un sistema a stati:

- *Allarme suona;*
- *Allarme non suona;*
- *Pulsante premuto;*
- *Pulsante non premuto;*

Basta fare delle simulazioni con SPIN dove uno preme l'allarme e prima ancora che la capo sala abbia disattivato l'allarme, qualcun altro suona l'allarme. Cosa succede?

- ⇨ Nel momento in cui la caposala disattiva l'allarme per servire il primo che ha chiamato, la chiamata dell'altro viene perduta.
- ⇨ La domanda è: **come risolvere questo**?
- ⇨ Si può pensare di introdurre un vettore, dove ogni processo va a lavorare solo su un elemento del vettore. Un altro processo va a lavorare su un altro elemento del vettore!
- ⇨ Ogni anziano di una stanza ha un elemento del vettore che equivale ad una lampadina. In questo modo non si ha più memoria condivisa.
- ⇨ Ci sono tante stanze e tante lampadine.

Fino a qui era ovvio anche senza SPIN che si ha bisogno di n stanze con n lampadine: andando avanti però, aggiungendo funzioni al sistema, il modello diventa più complesso e l'utilizzo di SPIN diventa essenziale per capire se si potrà arrivare ad uno stato indesiderato.

Possono esserci diversi livelli di allarme, che portano il sistema a crescere di complessità.

Obiettivo

Progettare un sistema d'allarme, partendo dal caso più banale e aggiungendo correzioni per aumentare la complessità e robustezza del modello.

Calcolo probabilistico con SPIN

Si può pensare di calcolare la probabilità di arrivare in un certo stato indesiderato.

Bisogna calcolare qual è la probabilità che un certo percorso casuale porti ad un nodo indesiderato.

- ⇨ Entra il gioco il simulatore per fare valutazioni di probabilità.

Si può adattare SPIN (anche se è più un compito da un model checker come PRISM).

Dato l'esempio già proposto del modello in PROMELA:

```
byte    n = 0;
proctype P() {
    byte temp, i;
    for (i: 1..5) {
        temp = n;
        n = temp + 1
    }
}
init {
    atomic {
        run P();
        run P()
    }
    (_nr_pr == 1); // forza init ad attendere gli altri due
    printf("Il valore è %d\n", n);
    assert (n > 2) // per cercare il caso peggiore
}
```

Si potrebbe calcolare la probabilità che la variabile n abbia un certo valore alla fine. (*Considerando almeno **10** iterazioni*)

- ⇨ Viene lanciato SPIN tramite uno script n volte automaticamente.
- ⇨ Si va a vedere su 1000 volte quante volte è stato dato assertion violation.
- ⇨ Facendo il rapporto si vede quanta è la probabilità che quell'evento accada.
- ⇨ Il simulatore in questo caso viene utilizzato non per fare una verifica di deadlock, starvation o violazione di mutua esclusione, ma viene usato per fare valutazioni più **generiche** e meno drammatiche. (*Deadlock e starvation sono le situazioni peggiori che si vogliono evitare*)