AUTOMATED REASONING, 2014/2015 1B: EXAM (OPEN BOOK), JAN 23, 2015

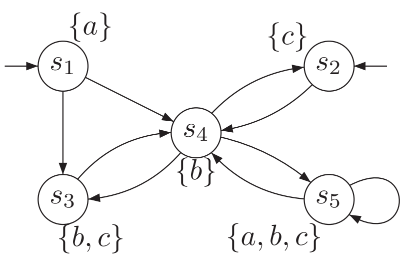
DOINA BUCUR, RUG

[**(P1) Types of specifications**] Take the (types of) properties below, where p and q are state formulas. Say whether they express **safety** or **liveness**, and explain why this is the case:

(1) a progress property;
(2) any invariant;
(3) the formula $p\mathbf{U}q$;
(4) a mutual exclusion property;
(5) a lack of starvation;
(6) a lack of deadlocks;
(7) the formula $\mathbf{XF}p$;
(8) the formula $!(\mathbf{G}p \wedge \mathbf{G}q)$;
(9) the LTL stability (or non-progress) pattern, $\mathbf{FG}p$;
(10) a property describing *fair* execution paths.

80

[**(P2) LTL checking on transition systems**] Take the following transition system M over the set of atomic propositions $\{a, b, c\}$.



For each LTL formula f below, decide whether $\mathbf{A}f$ ("for all computation paths, f") **holds** for M. When it does not, provide a path π in M on which $\pi \not\models f$.

(1) $\mathbf{G}a$
(2) $\mathbf{FG}c$
(3) $\mathbf{GF}c$
(4) $\mathbf{F}a$
(5) $(\mathbf{X}\neg c) \rightarrow \mathbf{XX}c$
(6) $a\mathbf{U}\mathbf{G}(b \vee c)$

80

[**(P3) New temporal sugar**] Take any LTL formulae f, g and h, and these informal descriptions for new temporal operators:

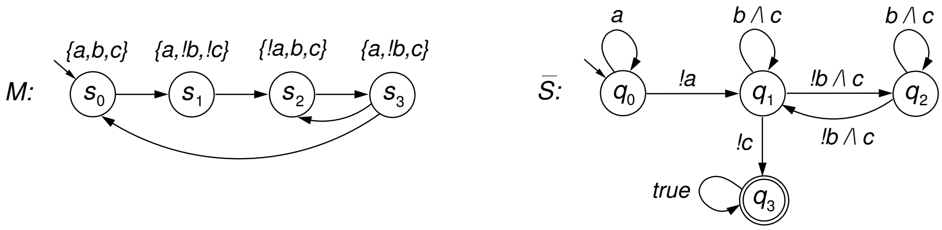
"Before": $f\mathbf{B}g$. If g holds sometime, then f holds at all times before that.
"After": $f\mathbf{A}g$. If g holds sometime, f does so at all times after that.
"From/to": $f\mathbf{F}g\mathbf{T}h$. From any point when g holds to some point when h holds, f holds all the way.

All other behaviour not described above should be left unspecified; for example, if g never holds, then the behaviour of f is irrelevant.

Formalize each operator by providing an equivalent (future-time) LTL formula using classical LTL operators.

80

[**(P4) Automata-based model checking**] Consider the following system model M and negated property in automaton form $\overline{S}$, both over the set of atomic propositions $\{a, b, c\}$:



Does the property hold on this system? If not, find the shortest counterexample.

80

[**(P5) Verifying pseudocode with pointers: an exploitable data race**] Have a computer system where all files (say, f and g) are held in small data structures:

```
struct {
    string  last_modified_by   // which process identifier last modified this file?
    string* pointer            // reference to a memory location where contents are stored
} f,g
```

Initially, both files f, g exist but are empty of content; f was last modified by process `"user0"`, and g by `"user1"`. Any process can open a file by calling function *open*, can then *write* into the pointer returned by *open*, can query which process last modified this file by calling *last_modified_by*, and can create a "soft" link from one file to another by calling *symlink*. Assume that the file data structures are passed by reference to these functions:

```
string* open(file, process) { atomic {
    file.last_modified_by := process
    return file.pointer
}}
string last_modified_by(file) { atomic {
    return file.last_modified_by
}}
void symlink (file_source, file_destination, process) { atomic {
    file_source.last_modified_by := process
    file_source.pointer := file_destination.pointer
}}
```

Take the two short concurrent processes below in pseudocode:

```
          (process "user0")                              (process "user1")
// initial values: safe := false, p := 0      // initial values: q: = 0, x := empty string

if (last_modified_by(f) == "user0")           symlink(f, g, "user1")
    safe := true                              atomic { q = open(g, "user1")
if (safe)                                              read from *q into x }
    atomic { p = open(f, "user0")
             write "user0-password" to *p }
```

(1) Write the Kripke structure for this concurrent system. How many locations of memory should you model?
(2) Then, verify on it the properties $\mathbf{F}$(`x==''user0-password''`) and !$\mathbf{F}$(`x==''user0-password''`).

(This is a known system bug: `http://en.wikipedia.org/wiki/Time_of_check_to_time_of_use`.)

80