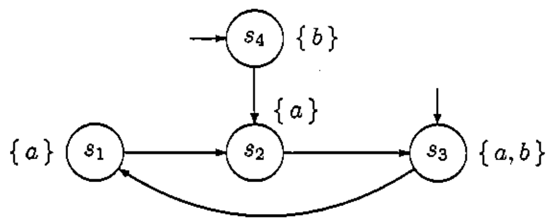
AUTOMATED REASONING, 2012/2013 1B: EXAM (OPEN BOOK), JAN 22, 2013

DOINA BUCUR, RUG

[**(P1) Write LTL from informal specifications**] Given atomic propositions $\{a, b, c, d\}$, write an LTL formula for each of the properties below, and characterize each into safety or liveness:
(a) a should never occur at the same time as b,
(b) any occurrence of c should eventually be followed by d,
(c) a should occur exactly once.

[15%]

[**(P2) LTL checking on states**] Consider the following Kripke structure over the set of atomic propositions $\{a, b\}$:



For each of the following LTL formulae f, state whether the formula holds on all computational paths, $\mathbf{A}f$, and—if the formula is violated—give a **minimal counterexample**:
(a) $f := \mathbf{G}(b\mathbf{U}a)$
(b) $f := \mathbf{G}\neg b$
(c) $f := \mathbf{GF}\neg a$
(d) $f := \mathbf{FG}a$
(e) $f := \mathbf{XX}b$

[15%]

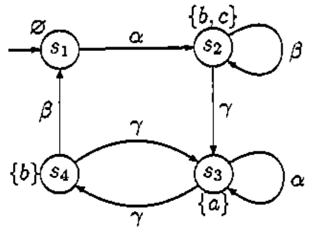
[**(P3) Equivalences of LTL formulas**] Which of the following LTL equivalences are correct? Either prove each equivalence or provide a counterexample. If you need to use other known LTL equivalences in a proof, prove those also; otherwise, simply use the LTL induction rules.

(1) $\mathbf{G}(f \vee g) \Leftrightarrow \mathbf{G}f \vee \mathbf{G}g$
(2) $\mathbf{XF}f \Leftrightarrow \mathbf{FX}f$
(3) $(\mathbf{F}f) \vee (\mathbf{XG}f) \Leftrightarrow \mathbf{XG}f$
(4) $\mathbf{F}(f \wedge g) \Leftrightarrow \mathbf{F}f \wedge \mathbf{F}g$

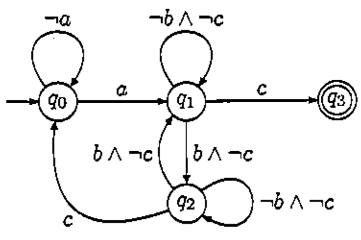
[20%]

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

M:



$\overline{S}$:



Does the property hold on this system? If not, give a counterexample.

(Note: the notation for state labels in M is such that only the positive form of atomic propositions is explicitly written; thus, a state label $\{a\}$ implicitly means $\{a, \neg b, \neg c\}$.)

[15%]

[(P5) Minimal counterexample] Sketch an algorithm (in pseudocode) for checking **invariants** over Kripke structures, such that in case the invariant is violated, the counterexample returned by the algorithm is of minimal length.

[15%]

[(P6) Extend LTL with past-time operators] You know the LTL temporal operators **X**, **G**, **F**, **U**; these are called "future-time" temporal operators, and they describe the future of an execution starting from the initial state. Extend LTL with "past-time" temporal operators, which describe the past of an execution from any state of that execution:

"Previously": $\mathbf{X}^{-1}f$. In the previous state on the path, f held.
"Always in the past": $\mathbf{G}^{-1}f$. Always in the past, f held.
"Eventually in the past": $\mathbf{F}^{-1}f$. Sometime in the past, f held.
"Since": $f\mathbf{U}^{-1}g$. Sometime in the past, g held, and ever since that point, f held.

(1) Write an induction rule for checking formulas written with each new temporal operator on an execution path π. You may use the usual notation π^k (with $k \geq 0$) to describe the fragment of π which starts in the k-th state; if you need new notation, define it.

(2) You know a method for runtime verification of future-time LTL which uses a monitor in automaton form, and which has constant computational complexity: it needs only a constant amount of time to verify the specification at each step in the system execution.

Sketch a new method to do runtime verification of a system execution against past-time LTL, with the same complexity, yet using only induction rules upon past-time LTL formulas. Your method should:

(i) take any given past-time LTL specification;
(ii) also take a linear system execution, which starts in an initial state and increases in length;
(iii) have an algorithm which outputs at each state in the system execution a (current) conclusion about the truth value of the specification.

(1) [12%]
(2) [8%]
