Examination Algorithms for Model Checking (2IW55)

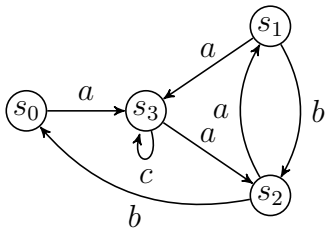
6 April, 2011, 14:00 – 17:00

Important notes:

- The exam consists of four questions.
- Weighting: 1: **30**, 2: **25**, 3: **25**, 4: **20**.
- Carefully read and answer the questions. The book, the course notes and other written material may be used during this examination.

---

1. Consider the following Kripke Structure K:



Consider the formula ϕ: $\nu X.\mu Y.((\langle a\rangle X \wedge \nu Z.\langle c\rangle Z) \vee (\langle a\rangle Y \wedge \mu W.[c]W))$.

(a) Compute the *nesting depth*, the *alternation depth* and the *dependent-alternation depth* of ϕ.

Solution:

- *Nesting depth:*

$$\begin{aligned} & ND(\phi) \\ = \;& 1 + ND(\mu Y.((\langle a\rangle X \wedge \nu Z.\langle c\rangle Z) \vee (\langle a\rangle Y \wedge \mu W.[c]W))) \\ = \;& 1 + 1 + ND((\langle a\rangle X \wedge \nu Z.\langle c\rangle Z) \vee (\langle a\rangle Y \wedge \mu W.[c]W)) \\ = \;& 1 + 1 + (ND(\nu Z.\langle c\rangle Z) \max ND(\mu W.[c]W)) \\ = \;& 1 + 1 + (1 \max 1) \\ = \;& 3 \end{aligned}$$

- *Alternation depth:*

$$\begin{aligned} & AD(\phi) \\ = \;& 1 + AD(\mu Y.((\langle a\rangle X \wedge \nu Z.\langle c\rangle Z) \vee (\langle a\rangle Y \wedge \mu W.[c]W))) \\ = \;& 1 + 1 + AD((\langle a\rangle X \wedge \nu Z.\langle c\rangle Z) \vee (\langle a\rangle Y \wedge \mu W.[c]W)) \\ = \;& 1 + 1 + \max\{AD(f) \mid f \textit{ is a } \nu\textit{-subformula of} \\ & \qquad (\langle a\rangle X \wedge \nu Z.\langle c\rangle Z) \vee (\langle a\rangle Y \wedge \mu W.[c]W)\} \\ = \;& 1 + 1 + AD(\nu Z.\langle c\rangle Z) \\ = \;& 1 + 1 + 1 \\ = \;& 3 \end{aligned}$$

- *Dependent alternation depth:*

$$\begin{aligned}
& dAD(\phi) \\
= \;& 1 + dAD(\mu Y.((\langle a\rangle X \wedge \nu Z.\langle c\rangle Z) \vee (\langle a\rangle Y \wedge \mu W.[c]W))) \\
= \;& 1 + \max(dAD((\langle a\rangle X \wedge \nu Z.\langle c\rangle Z) \vee (\langle a\rangle Y \wedge \mu W.[c]W)), \\
& \quad 1 + \max\{dAD(f) \mid f \textit{ is a } \nu\textit{-subformula of} \\
& \qquad\qquad (\langle a\rangle X \wedge \nu Z.\langle c\rangle Z) \vee (\langle a\rangle Y \wedge \mu W.[c]W), \textit{ and,} \\
& \qquad\qquad Y \textit{ occurs in } f \;\} \\
= \;& 1 + \max(1, 1 + 0) \\
= \;& 2
\end{aligned}$$

(b) Compute the set of states of K that satisfy ϕ, using the **Emerson-Lei** algorithm. Show the intermediate steps in all your computations.

Solution: There are a total of four different formal variable in ϕ: W, X, Y, Z. Let A be an array that keeps track of the values for these variables during the approximations. We initialise A as follows:

$$\begin{aligned}
A[W] &:= \emptyset (= \mathit{false}) \\
A[X] &:= S (= \mathit{true}) \\
A[Y] &:= \emptyset (= \mathit{false}) \\
A[Z] &:= S (= \mathit{true})
\end{aligned}$$

We next analyse ϕ to see which fixed-point computations should be performed; we do this to see if there are sub-expressions that can be analysed in isolation. Formula ϕ has four distinct fixed-point expressions; for each, we state which values in A will have to be reset before applying function $\mathsf{eval}()$ *on them:*

- $\mathsf{eval}(\nu X.\mu Y.((\langle a\rangle X \wedge \nu Z.\langle c\rangle Z) \vee (\langle a\rangle Y \wedge \mu W.[c]W)))$*: no resets required, as there are no surrounding binders;*
- $\mathsf{eval}(\mu Y.((\langle a\rangle X \wedge \nu Z.\langle c\rangle Z) \vee (\langle a\rangle Y \wedge \mu W.[c]W)))$*: reset $A[Y]$ to $\emptyset$;*
- $\mathsf{eval}(\nu Z.\langle c\rangle Z)$*: surrounding binder is μ, but Z is closed, so no reset required;*
- $\mathsf{eval}(\mu W.[c]W)$*: surrounding binder is μ, but Z is closed, so no resets.*

Since the latter two fixed-point expressions are closed and do not reset any of the variables occurring in them, we can compute these in isolation. We here show the computations required for these:

- *Given $A[Z] = S$, we compute* $\mathsf{eval}(\nu Z.\langle c\rangle Z)$*. As observed no resets are*

required. We thus obtain the following sequence of computations:

$$\begin{array}{ll} Z_{old} & := S \\ A[Z] & := \mathsf{eval}(\langle c \rangle Z) \\ & = \{s \in S \mid \exists s' \in S : s \xrightarrow{c} s' \wedge s' \in \mathsf{eval}(Z)\} \\ & = \{s \in S \mid \exists s' \in S : s \xrightarrow{c} s' \wedge s' \in S\} \\ & = \{s_3\} \\ Z_{old} & := \{s_3\} \\ A[Z] & := \mathsf{eval}(\langle c \rangle Z) \\ & = \{s \in S \mid \exists s' \in S : s \xrightarrow{c} s' \wedge s' \in \{s_3\}\} \\ & = \{s_3\} \end{array}$$

We therefore find that $\mathsf{eval}(\nu Z.\langle c \rangle Z)$ *changes* $A[Z] = S$ *to* $A[Z] = \{s_3\}$*, and returns* $\{s_3\}$*. Any subsequent call to* $\mathsf{eval}(\nu Z.\langle c \rangle Z)$*, given that* $A[Z] = \{s_3\}$ *will return* $\{s_3\}$ *almost immediately.*

- *Given* $A[W] = \emptyset$*, we compute* $\mathsf{eval}(\mu W.[c]W)$*. As observed, no resets are performed; we thus have the following computation:*

$$\begin{array}{ll} W_{old} & := \emptyset \\ A[W] & := \mathsf{eval}([c]W) \\ & = \{s \in S \mid \forall s' \in S : s \xrightarrow{c} s' \Rightarrow s' \in \mathsf{eval}(W)\} \\ & = \{s \in S \mid \forall s' \in S : s \xrightarrow{c} s' \Rightarrow s' \in \emptyset\} \\ & = S \setminus \{s_3\} \\ W_{old} & := S \setminus \{s_3\} \\ A[W] & := \mathsf{eval}([c]W) \\ & = \{s \in S \mid \forall s' \in S : s \xrightarrow{c} s' \Rightarrow s' \in S \setminus \{s_3\}\} \\ & = S \setminus \{s_3\} \end{array}$$

Thus, $\mathsf{eval}(\mu W.[c]W)$ *changes* $A[W] = \emptyset$ *to* $A[W] = S \setminus \{s_3\}$ *and returns* $S \setminus \{s_3\}$*. Again, given* $A[W] = S \setminus \{s_3\}$*, invoking* $\mathsf{eval}(\mu W.[c]W)$ *will almost instantly return* $S \setminus \{s_3\}$*.*

We introduce the following abbreviation:

$$\phi' := \mu Y.((\langle a \rangle X \wedge \nu Z.\langle c \rangle Z) \vee (\langle a \rangle Y \wedge \mu W.[c]W))$$

Using the above pre-computations, we run $\mathsf{eval}(\phi)$*, which is* $\mathsf{eval}(\nu X.\phi')$*. We get:*

$$\begin{array}{ll} X_{old} & := S; \\ A[X] & := \mathsf{eval}(\phi') \\ & = S \text{ (see computation at † below)} \end{array}$$

So, the approximation for X *immediately stabilises to* S*. The intermediate computation used at † is detailed next. The computation* $\mathsf{eval}(\phi')$ *only resets*

$A[Y]$ *to* $\emptyset$ *as observed in our analysis. As a result, we find:*

$$
\begin{array}{ll}
Y_{old} & := \emptyset; \\
A[Y] & := \mathsf{eval}((\langle a\rangle X \wedge \nu Z.\langle c\rangle Z) \vee (\langle a\rangle Y \wedge \mu W.[c]W)) \\
& = (\{s \in S \mid \exists s' \in S : s \xrightarrow{a} s' \wedge s' \in \mathsf{eval}(X)\} \cap \mathsf{eval}(\nu Z.\langle c\rangle Z)) \cup \\
& \quad (\{s \in S \mid \exists s' \in S : s \xrightarrow{a} s' \wedge s' \in \mathsf{eval}(Y)\} \cap \mathsf{eval}(\mu W.[c]W)) \\
& = (\{s \in S \mid \exists s' \in S : s \xrightarrow{a} s' \wedge s' \in S\} \cap \{s_3\}) \cup \\
& \quad (\{s \in S \mid \exists s' \in S : s \xrightarrow{a} s' \wedge s' \in \emptyset\} \cap S \setminus \{s_3\}) \\
& = \{s_3\} \\
Y_{old} & := \{s_3\}; \\
A[Y] & := \mathsf{eval}((\langle a\rangle X \wedge \nu Z.\langle c\rangle Z) \vee (\langle a\rangle Y \wedge \mu W.[c]W)) \\
& = (\{s \in S \mid \exists s' \in S : s \xrightarrow{a} s' \wedge s' \in \mathsf{eval}(X)\} \cap \mathsf{eval}(\nu Z.\langle c\rangle Z)) \cup \\
& \quad (\{s \in S \mid \exists s' \in S : s \xrightarrow{a} s' \wedge s' \in \mathsf{eval}(Y)\} \cap \mathsf{eval}(\mu W.[c]W)) \\
& = (\{s \in S \mid \exists s' \in S : s \xrightarrow{a} s' \wedge s' \in S\} \cap \{s_3\}) \cup \\
& \quad (\{s \in S \mid \exists s' \in S : s \xrightarrow{a} s' \wedge s' \in \{s_3\}\} \cap S \setminus \{s_3\}) \\
& = \{s_0, s_1, s_3\} \\
Y_{old} & := \{s_0, s_1, s_3\}; \\
A[Y] & := \mathsf{eval}((\langle a\rangle X \wedge \nu Z.\langle c\rangle Z) \vee (\langle a\rangle Y \wedge \mu W.[c]W)) \\
& = (\{s \in S \mid \exists s' \in S : s \xrightarrow{a} s' \wedge s' \in \mathsf{eval}(X)\} \cap \mathsf{eval}(\nu Z.\langle c\rangle Z)) \cup \\
& \quad (\{s \in S \mid \exists s' \in S : s \xrightarrow{a} s' \wedge s' \in \mathsf{eval}(Y)\} \cap \mathsf{eval}(\mu W.[c]W)) \\
& = (\{s \in S \mid \exists s' \in S : s \xrightarrow{a} s' \wedge s' \in S\} \cap \{s_3\}) \cup \\
& \quad (\{s \in S \mid \exists s' \in S : s \xrightarrow{a} s' \wedge s' \in \{s_0, s_1, s_3\}\} \cap S \setminus \{s_3\}) \\
& = \{s_0, s_1, s_2, s_3\}
\end{array}
$$

At this point, $A[Y] = S$*, so no further approximation is required (*$A[Y]$ *cannot grow beyond* S*). Note that the computation of* $\mathsf{eval}(\phi')$ *requires the solutions to* $\mathsf{eval}(\nu Z.\langle c\rangle Z)$ *and* $\mathsf{eval}(\mu W.[c]W)$ *detailed earlier. In conclusion, we find that* ϕ *holds in* ***every*** *state in the given Kripke Structure.*

2. Consider the PBES $\mathcal{E}$ given below:

$$\begin{array}{ll}(\nu X(n{:}Nat) & = Y(n)) \\ (\mu Y(n{:}Nat) & = (even(n) \wedge Y(n)) \vee (odd(n) \wedge X(n+1)))\end{array}$$

(a) Compute the solution to $X(n)$, for arbitrary $n \in Nat$ for the PBES $\mathcal{E}$ given above. Show all steps in your computations.

Solution: A cursory inspection of the PBES indicates that it will not pay to run a redundant parameter elimination algorithm, as the only parameter n occurs significantly in the equation for Y. Since the parameter n of X influences the value of n in Y, all parameters are possibly relevant. The only viable option is, hence, to run Gauß Elimination, combined with symbolic approximation. We first approximate the solution to Y. The i-th approximant is denoted Y_i; since Y is a least-fixed point equation, approximation starts at false*:*

$$\begin{array}{ll}Y_0(n) & = \mathit{false} \\ Y_1(n) & = (even(n) \wedge Y_0(n)) \vee (odd(n) \wedge X(n+1)) \\ & = (even(n) \wedge \mathit{false}) \vee (odd(n) \wedge X(n+1)) \\ & = odd(n) \wedge X(n+1) \\ Y_2(n) & = (even(n) \wedge Y_1(n)) \vee (odd(n) \wedge X(n+1)) \\ & = (even(n) \wedge (odd(n) \wedge X(n+1))) \vee (odd(n) \wedge X(n+1)) \\ & = (odd(n) \wedge X(n+1))\end{array}$$

We thus find that $Y(n) = odd(n) \wedge X(n+1)$. We next substitute the solution to Y in the equation for X; this yields the following equation for X:

$$(\nu X(n{:}Nat) = odd(n) \wedge X(n+1))$$

We repeat the process of symbolic approximation, applied to the equation for X; since X is a greatest-fixed point equation, approximation starts at true*:*

$$\begin{array}{ll}X_0(n) & = \mathit{true} \\ X_1(n) & = odd(n) \wedge X_0(n+1) \\ & = odd(n) \wedge \mathit{true} \\ & = odd(n) \\ X_2(n) & = odd(n) \wedge X_1(n+1) \\ & = odd(n) \wedge odd(n+1) \\ & = \mathit{false}\end{array}$$

So, the solution to $X(n)$ is false *for all $n \in Nat$.*

(b) Consider the process described by the LPE P given below.

$$\begin{array}{rrrl}P(n{:}Nat) & = & even(n) & \rightarrow stable \cdot P(n) \\ & + & odd(n) & \rightarrow inc \cdot P(n+1) \\ & + & \sum m{:}Nat.true & \rightarrow \tau \cdot P(m)\end{array}$$

Let ϕ be the first-order modal μ-calculus formula $\nu X.\mu Y.([stable]Y \wedge [inc]X)$

Verify whether the PBES $\mathcal{E}$ can be the result (up-to logical equivalence of right-hand side formulae) of the transformation $\mathsf{E}\ (\phi)$ applied to P, by performing the transformation and indicating all (if any) differences between $\mathsf{E}\ (\phi)$ and $\mathcal{E}$.

Solution: No, the PBES cannot be the result of the transformation $\mathsf{E}\ (\phi)$*. We detail the transformation below.*

$$
\begin{array}{ll}
 & \mathsf{E}\ (\phi) \\
= & \mathsf{E}\ (\nu X.\mu Y.([stable]Y \wedge [inc]X)) \\
= & (\nu X(n{:}Nat) = \mathsf{RHS}(\mu Y.([stable]Y \wedge [inc]X))) \\
 & \mathsf{E}\ (\mu Y.([stable]Y \wedge [inc]X)) \\
= & (\nu X(n{:}Nat) = \mathsf{RHS}(\mu Y.([stable]Y \wedge [inc]X))) \\
 & (\mu Y(n{:}Nat) = \mathsf{RHS}([stable]Y \wedge [inc]X))
\end{array}
$$

The left-hand sides of the equations match with the left-hand sides of the equations in $\mathcal{E}$*. We next check whether the right-hand sides of the above equations will match those in* $\mathcal{E}$ *as well. For the equation for* X*, we obtain:*

$$
\begin{array}{ll}
 & \mathsf{RHS}(\mu Y.([stable]Y \wedge [inc]X)) \\
= & Y(n)
\end{array}
$$

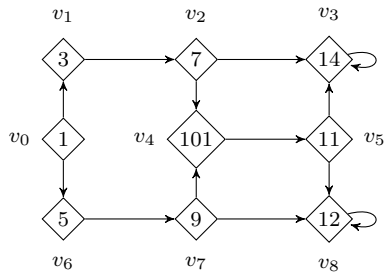
So, the equations for X *match. Next, consider the equation for* Y*; we perform on-the-fly matching with action names, to avoid computing superfluous expressions:*

$$
\begin{array}{ll}
 & \mathsf{RHS}([stable]Y \wedge [inc]X) \\
= & \mathsf{RHS}([stable]Y) \wedge \mathsf{RHS}([inc]X) \\
= & (even(n) \Rightarrow (\mathsf{RHS}(Y)[n := n])) \wedge \\
 & (odd(n) \Rightarrow (\mathsf{RHS}(X)[n := n+1])) \\
= & (even(n) \Rightarrow Y(n)) \wedge (odd(n) \Rightarrow X(n+1))
\end{array}
$$

The equation for Y*, as derived above clearly cannot be rewritten to match the equation for* Y *in* $\mathcal{E}$*. The translation* $\mathsf{E}\ (\phi)$ *results in:*

$$
\begin{array}{ll}
(\nu X(n{:}Nat) & = Y(n)) \\
(\mu Y(n{:}Nat) & = (even(n) \Rightarrow Y(n)) \wedge (odd(n) \Rightarrow X(n+1)))
\end{array}
$$

3. Solve the Parity Game, depicted below, using either the **Small Progress Measures** algorithm or the **Recursive** algorithm. State which of the two algorithms you choose to use, and **explicitly** mention which transformations you use on the Parity Game (if any) prior to solving the game, and **why** you choose to use them.

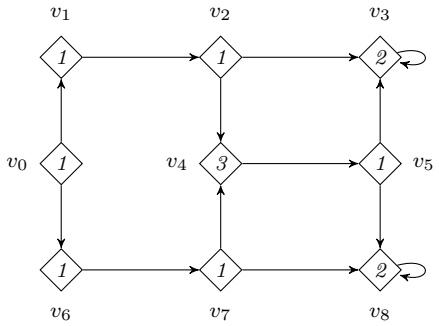


Solution: Observe that it is not advisable (though not impossible) to run either of the required algorithms on the original Parity Game:

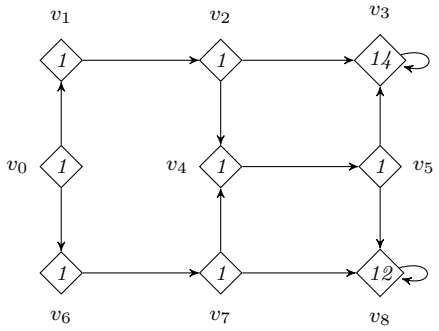
- *The Small Progress Measures algorithm requires excessively large progress measures tuples (due to priority 101);*
- *the Recursive algorithm has a large number of recursive calls due to the fact that in each recursive call, a graph of only a single vertex less is considered.*

We therefore first apply transformations on the Parity Game, thereby reducing the priorities that occur in the game, with the aim to reduce the progress measures vectors and possibly lower the number of recursive calls in the Recursive algorithm. We can either use priority compaction *or* priority propagation*; both have, in this case, similar effects.*

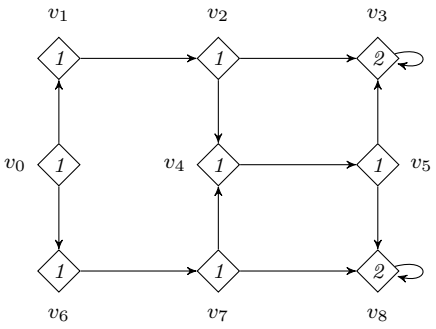
- *Priority compaction: Observe that priority 10 does not occur in the game, so we can replace all priorities 11 with 9. Since priority 8 does not occur, all priorities 9 can be replaced to 7. Since priority 6 does not occur, priority 7 changes to 5. As priority 4 does not occur, all vertices with priority 5 change to vertices with priority 3. Finally, we can change all vertices with priority 3 to vertices with priority 1. In a similar vein, we can then transform the vertices of priority 14 and 12 to 2. Finally, we can change priority 101 to 3, using the same reasoning. We are thus left with the following graph:*



- *Priority propagation: consider vertices v_1 and v_6; their only predecessor is vertex v_0, which has priority 1; as a result, we can reset their priority to 1 (the maximal priority of the predecessors if that is smaller than the current priority, which is the case here). This is sound, since any path that visits v_1 (resp. v_6), inevitably visits v_0. The same reasoning now applies to vertices v_2 and v_7, which can also be reassigned priority 1. Vertex v_4 has two predecessors, which now both have priority 1. As a result, we can change priority 101 to 1. Likewise, we can change priority 11 for vertex v_5 to 1. This results in the following graph:*



Both games have now become manageable for both solving using the recursive algorithm. We note that both transformations can also be combined by first applying one, and then the other. This results (regardless of the order in which the transformations are applied) in the following graph, which we will subsequently solve:



Denote the set of all vertices in the graph by V, and we refer to our graph as G.

- *Recursive algorithm: Start by identifying all vertices with priority 1. This is $V \setminus \{v_3, v_8\}$. Compute $Attr_\Box(G, V \setminus \{v_3, v_8\})$. Due to the self-loops in both v_3 and v_8, we find that $Attr_\Box(G, V \setminus \{v_3, v_8\}) = V \setminus \{v_3, v_8\}$.*

  We note that the Recursive algorithm has to be run on the subgraph G', which is G restricted to $\{v_3, v_8\}$. We therefore first solve this graph. The least priority occurring in this game graph is 2. We have $Attr_\Diamond(G', \{v_3, v_8\}) = \{v_3, v_8\}$. A subsequent recursive call to the empty graph, returns $(\emptyset, \emptyset)$. As a result, we find that player even *wins all vertices in G'.*

  We now apply consider the main computation again. We already found that $Attr_\Box(G, V \setminus \{v_3, v_8\}) = V \setminus \{v_3, v_8\}$, and $Recursive(G', \{v_3, v_8\}) = (\{v_3, v_8\}, \emptyset)$. Therefore, assign $(X_\Diamond, X_\Box) = (\{v_3, v_8\}, \emptyset)$. Since $X_\Diamond \neq \emptyset$, we assign $B = Attr_\Diamond(G, \{v_3, v_8\})$ (observe that the computation now is within graph G instead of G'):

$$\begin{aligned}
Attr^0_\Diamond(G, \{v_3, v_8\}) &= \{v_3, v_8\} \\
Attr^1_\Diamond(G, \{v_3, v_8\}) &= \{v_3, v_8\} \cup \{v_2, v_5, v_7\} \\
Attr^2_\Diamond(G, \{v_3, v_8\}) &= \{v_2, v_3, v_5, v_7, v_8\} \cup \{v_1, v_4, v_6\} \\
Attr^3_\Diamond(G, \{v_3, v_8\}) &= \{v_1, v_2, v_3, v_4, v_5, v_6, v_7, v_8\} \cup \{v_0\} \\
Attr^4_\Diamond(G, \{v_3, v_8\}) &= \{v_0, v_1, v_2, v_3, v_4, v_5, v_6, v_7, v_8\}
\end{aligned}$$

  As a result, $B = V$. A recursive call on the graph $G \setminus B$ (which is the empty game graph), results in the assignment $(Y_\Diamond, Y_\Box) = (\emptyset, \emptyset)$. We now set $W_\Box = Y_\Box$, and $W_\Diamond = Y_\Diamond \cup B$, i.e., $W_\Diamond = V$. As a result, the Recursive algorithm terminates with $W_\Diamond = V$ and $W_\Box = \emptyset$. Thus, the entire graph is won by player even.

- *Small Progress Measures: We first identify the set $\mathbb{M}^\top_G$. Observe that there are 7 vertices with priority 1, and two vertices with priority 2. Thus:*

$$\mathbb{M}^\top_G = \{(0, i, 0) \mid 0 \leq i \leq 7\} \cup \{\top\}$$

  The initial progress measure is given by $\rho_0(v) = (0, 0, 0)$ for all $v \in V$. Lifting this progress measure can be performed in a number of ways. The most efficient strategy is to first lift v_5, and, from thereon, all its predecessors:

$$\begin{aligned}
\rho_1 &= Lift_{v_5} = \rho_0[v_5 := (0, 1, 0)] \\
\rho_2 &= Lift_{v_4} = \rho_1[v_4 := (0, 2, 0)] \\
\rho_3 &= Lift_{v_2} = \rho_2[v_2 := (0, 1, 0)] \\
\rho_4 &= Lift_{v_7} = \rho_3[v_7 := (0, 1, 0)] \\
\rho_5 &= Lift_{v_1} = \rho_4[v_1 := (0, 2, 0)] \\
\rho_6 &= Lift_{v_6} = \rho_5[v_6 := (0, 2, 0)] \\
\rho_7 &= Lift_{v_0} = \rho_6[v_0 := (0, 3, 0)]
\end{aligned}$$

  It is not hard to verify that the resulting progress measure ρ_7 is stable (i.e., no vertex can be lifted), and, therefore, is the outcome of applying the Small

Progress Measures algorithm. Since we have $\rho_7(v) \neq \top$ *for all* $v \in V$*, we find that all vertices are won by player* even. *Note that, apart from the notational inconveniences, the analysis does not change essentially by not first applying the transformations.*

4. Let $M = (S, R, S_0, L)$ be an arbitrary Kripke Structure over a set AP of atomic propositions; S is the set of states, R is the transition relation, $\emptyset \neq S_0 \subseteq S$ is the set of initial states and $L{:}S \to 2^{AP}$ is the labelling. Read the statements below carefully, and, for each statement, give either a short proof or a counterexample.

   (a) Let M', M'' be Kripke Structures over AP such that:

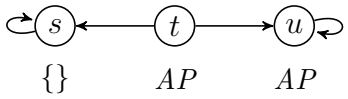
   - $M \sqsubseteq M'$, i.e., M' simulates M, and
   - $M' \equiv M''$, i.e., M' and M'' are bisimilar.

   Let f be an ACTL^* formula such that $M'' \models f$. Then also $M \models f$.

   Solution: This statement is true. Since $M'' \models f$ and $M' \equiv M''$, we have $M' \models f$ (bisimilarity preserves and reflects CTL^*). Since f is in ACTL^*, and $M \sqsubseteq M'$, we find that $M \models f$ (similarity is preserved by ACTL^*).

   (b) Let state $s \in S$ be such that $L(s) = \emptyset$. Assume that for all $s', s'' \in S \setminus \{s\}$, we have $L(s') = L(s'')$ and $L(s') \neq \emptyset$. That is, all states in M, save state s, have the same non-empty labelling. Then, the relation B_0^* on S, defined as $B_0^* = \{(s', s'') \in S \times S \mid L(s') = L(s'')\}$ is always a bisimulation relation.

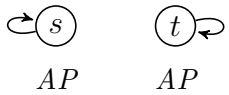
   Solution: This statement is false. We need at least three states to show this. Consider the Kripke Structure below:



   If B_0^* is a bisimulation relation, then t and u should be bisimilar. However, state u satisfies $\mathsf{A\,G}\ AP$, but state t does not satisfy $\mathsf{A\,G}\ AP$. This means they cannot be bisimilar.

   (c) Let $s, s' \in S$ be two distinct, bisimilar states. That is, there is a bisimulation relation $B \subseteq S \times S$ such that $(s, s') \in B$. Then there is no fairness constraint $\mathcal{F}$ and no CTL formula f, such that $s \models_F f$ and $s' \not\models_F f$ using constraint $\mathcal{F}$.

   Solution: This statement is false. Consider the following Kripke Structure:



   Observe that states s and t are (trivially) bisimilar. Let $\mathcal{F} = \{\{s\}\}$. Clearly, $s \models_F \mathsf{true}$, but not $t \models_F \mathsf{true}$, invalidating the statement.

□