1 Games with reachability objectives

Recall that a model in the context of games with reachability objectives is a marked graph $G = ((S, E), (S_1, S_2))$, where $S$ is the finite set of states, $E$ the set of edges and $S_1$ and $S_2$ partition $S$. We abbreviate $n = |S|$ and $m = |E|$. For any $s$ in $S$, let $E(s) = \{s' \in S \mid (s, s') \in E\}$ and assume that for all $s$ in $S$, $E(s) \neq \emptyset$. $T \subseteq S$ denotes the set of target states. Intuitively\(^1\) player 1 and player 2 play the following game starting at some $s$ in $S$: Repeatedly player 1 and player 2 perform steps in the following way: if $s$ in $S_1$ ($S_2$), player 1 (2) chooses a successor state from $E(s)$, from where on the next step is started. Player 1 wins the game if she manages to reach some $s$ in $T$. Conversely, player 2 wins the game if she manages to stay in $S \setminus T$ forever.

In the last lecture it was shown that

$$\langle\langle 1 \rangle\rangle(\Diamond T) = S \setminus \langle\langle 2 \rangle\rangle(\Box F),$$

(1)

where $F = S \setminus T$, and $\langle\langle 1 \rangle\rangle(\Diamond T)$ (respectively $\langle\langle 2 \rangle\rangle(\Diamond T)$) denotes the set of initial states from where on player 1 (respectively 2) has a winning strategy. The set is called winning set of player 1 (respectively 2).

Further a non optimal (with respect to time complexity) algorithm was presented to compute $\langle\langle 1 \rangle\rangle(\Diamond T)$ for model $G$ and set $T$. Contrary, the algorithm presented in this lecture is optimal. Its pseudocode is stated in Figure 1 on the next page.

To ease the presentation of the algorithm, we might assume that for each $s$ in $S$, both the sets

$$In(s) = \{s' \mid (s', s) \in E\} \text{ and } Out(s) = \{s' \mid (s, s') \in E\}$$

are already given and do not need to be computed.

The algorithm maintains a set of counters $c[s]$ for each $s$ in $S$ which decrement from $|Out(s)|$ to 0. A Boolean array $Done[s]$ for each $s \in S$ is used to remember which states already have been visited. A queue $Q$ stores the states $s$ whose predecessor states $In(s)$ still need to be processed as well as a set $P$ containing all processed states.

The idea of the algorithm is to search backward, starting from the states in $T$ for states $s$ from where on player 1 has a winning strategy, marking them as $Done[s] = true$ and adding them to the set $P$. For a state $s$ in $P$, which is a subset of the winning set of player 1, the algorithm looks backward at the predecessor states $t$ in $In(s)$ (line 12) and distinguishes between whether (i) $t$ in $S_1$ (lines 13–15) or (ii) $t$ in $S_2$ (lines 17–20): (i) Clearly those states

\(^1\)For a rigorous definition consult the the scribe note of lecture 2.
Initially
1: for $s \in S$ do
2: $c[s] \leftarrow |\text{Out}(s)|$
3: $\text{Done}[s] \leftarrow false$
4: $Q \leftarrow ()$
5: $P \leftarrow \emptyset$
6: for $s \in T$ do
7: $Q \leftarrow \text{Enqueue}(Q, s)$
8: $\text{Done}[s] \leftarrow true$

Main
9: while $Q \neq ()$ do
10: $s \leftarrow \text{Dequeue}(Q)$
11: $P \leftarrow P \cup \{s\}$
12: for $t \in \text{In}(s)$ do
13: if $t \in S_1$ and $\text{Done}[t] = false$ then
14: $Q \leftarrow \text{Enqueue}(Q, t)$
15: $\text{Done}[t] \leftarrow true$
16: if $t \in S_2$ and $\text{Done}[t] = false$ then
17: $c[t] \leftarrow c[t] - 1$
18: if $c[t] = 0$ then
19: $Q \leftarrow \text{Enqueue}(Q, t)$
20: $\text{Done}[t] \leftarrow true$
21: return $P$

Figure 1: Algorithm to compute $(\langle 1 \rangle)(\Diamond T)$

t in $S_1$, from where on a state $s$ in $P$ can be reached (line 13) are in the winning set of player 1, too and thus added to $Q$ (line 14), which results in them being added to $P$ during a later pass off the while-loop (line 11). (ii) Contrary, for those states $t$ in $S_2$, the counter $c[t]$ is decremented. If it holds that $c[t]$ can only be decremented once by each successor state$^2$, then $c[t] = 0$ implies, that it has only successor states in $P$. It thus can be safely added to $Q$ (line 18–20) and again is added to $P$ during a later pass of the while-loop.

We will next analyze the correctness and time complexity of the algorithm.

Correctness proof

Proposition 1. Every vertex $s$ in $S$ is enqueued at most once.

Proof. Consider some $s$ in $S$. Assume that $\text{Enqueue}(Q, s)$ is executed the first time for $s$. Then $\text{Done}[s] \leftarrow true$ is executed afterwards. From the algorithm, $\text{Enqueue}(Q, s)$ is executed only if $\text{Done}[s] = false$ holds. Thus $\text{Enqueue}(Q, s)$ cannot be executed a second time. ∎

Proposition 2. For every $s$ in $S$,

(i) if $s \in S_2$ and $s$ is added to $P$, then $E(s) \subseteq P$ or $s \in T$ must have held.

(ii) if $s \in S_1$ and $s$ is added to $P$, then $E(s) \cap P \neq \emptyset$ or $s \in T$ must have held.

Proof. ad (i) Assume that $s \in S_2$ and $s$ is added to $P$ (line 11), then it must have been enqueued before in the algorithm’s execution. In case $s \in T$ the proposition follows. If $s$ not

$^2$We will prove this correct later on.
Lemma 1. $P \subseteq \langle\langle 1\rangle\rangle(\sqcap T)$.

Proof. The proof is by induction on $P$’s size, which grows during an execution of the algorithm.

Begin ($|P| = 1$) Initially, a vertex $s$ from $T$ is added to $P$. Trivially player 1 has a strategy to reach $T$ from $s$.

Step ($k \to k + 1$) Assume the lemma holds for $|P| = k \geq 1$. We will next show that it holds, too, after the next vertex $s$ from $S$ is added to $P$. We distinguish between two cases for $s$: (i) $s$ in $S_2$ and (ii) $s$ in $S_1$.

ad (i) Because of Proposition 2, $E(s) \subseteq P$ or $s \in T$ must have held, before adding $s$ to $P$. In case $s \in T$, the lemma trivially follows. If $s$ not in $T$, by the induction hypothesis, player 1 has a winning strategy for all $s'$ in $P$. Since, $s$ in $S_2$, player 2 can chose an arbitrary successor state of $s$. However, all successor states of $s$, namely $E(s)$ are in the winning set of player 1. Thus the lemma follows.

ad (ii) Because of Proposition 2, $E(s) \cap P \neq \emptyset$ or $s \in T$ must have held, before adding $s$ to $P$. In case $s \in T$, the lemma trivially follows. If $s$ not in $T$, by the induction hypothesis, player 1 has a winning strategy for all $s'$ in $P$. Since, $s$ in $S_1$, player 1 can chose an arbitrary successor state of $s$. Let player 1 choose an arbitrary successor state of $s$ from $E(s) \cap P$, which, all are in the winning set of player 1. Thus the lemma follows.

Proposition 3. For every $s$ in $S$,

(i) if $s \in S_2$ and $E(s) \subseteq P$, $s$ is enqueued and added to $P$.

(ii) if $s \in S_1$ and $E(s) \cap P \neq \emptyset$, $s$ is enqueued and added to $P$.

Proof. ad (i) Assume that $s \in S_2$ and $E(s) \subseteq P$ hold at some time $\tau$ in the algorithm’s execution. Then all $s$’s successors, $E(s) = Out(s)$, have been added to $P$. Thus all vertices in $Out(s)$ were enqueued before (see lines 10–11) and each of them has decremented $c[s]$ even before (line 16 must have held, if $s$ has not already been enqueued before). It follows that $c[s] = 0$ must have held after some decrement of $c[s]$. Thus lines 19–20 are executed, i.e., $s$ is enqueued.

ad (ii) Assume that $s \in S_1$ and $E(s) \cap P \neq \emptyset$ holds at some time $\tau$. Then at least one vertex in $E(s) = Out(s)$, say $s'$, has been added to $P$ (line 11). Since $s$ in $In(s')$ (line 12) and $s \in S_1$ (line 13), $s$ is enqueued (line 14), if it has not already been enqueued before.
Lemma 2. \( S \setminus P \subseteq \langle \langle 2 \rangle \rangle (\Box F) \).

Proof. Assume that the algorithm returns \( P \) and let \( s \) be from \( S \setminus P \). We distinguish between two cases: (i) \( s \) in \( S_2 \) and (ii) \( s \) in \( S_1 \).

ad (i) By Proposition 3, \((E(s) \subseteq P)\) must hold, since otherwise \( s \) would have been added to \( P \). Thus, there is an edge from \( s \) to \( S \setminus P \), player 2 can choose, and thereby stay in \( S \setminus P \).

ad (ii) By Proposition 3, \((E(s) \cap P \neq \emptyset)\) must hold, since otherwise \( s \) would have been added to \( P \). Thus, there are only edges from \( s \) to \( S \setminus P \), player 1 can choose, and thereby must stay in \( S \setminus P \).

It follows that
\[
\Box (S \setminus P) \text{ since } T \subseteq P \text{ and thus } S \setminus P \subseteq S \setminus T \Rightarrow \Box (S \setminus T) = \Box F.
\]

We are now ready to state the correctness of the algorithm.

Theorem 1. It holds that \( P = \langle \langle 1 \rangle \rangle (\Diamond T) \) and \( S \setminus P = \langle \langle 2 \rangle \rangle (\Box F) \).

Proof. From Lemma 1,
\[
\begin{align*}
P &\subseteq \langle \langle 1 \rangle \rangle (\Diamond T) \Rightarrow \\
S \setminus P &\supseteq S \setminus \langle \langle 1 \rangle \rangle (\Diamond T) \text{ by (1)} \\
S \setminus P &\supseteq \langle \langle 2 \rangle \rangle (\Box F) \text{ by Lemma 2} \\
S \setminus P &\supseteq \langle \langle 2 \rangle \rangle (\Box F) \text{ and} \\
P &\supseteq \langle \langle 1 \rangle \rangle (\Diamond T).
\end{align*}
\]

Time Complexity The initial part of the algorithm (lines 1–8) can be done in time \( O(n) \).

Since every \( s \) in \( S \) loops through all its predecessors \( t \) in \( \text{In}(s) \) at most once, it follows that the time complexity of the main part is bounded by \( O(\sum_{s \in S} |\text{In}(s)|) = O(m) \). We therefore obtain an overall linear time complexity bound of \( O(m + n) \). Since, the problem of finding the winning states for player 1 was shown to be \( \text{PTIME} \) complete, the given algorithm is optimal with respect to time complexity.

2 Monotonic functions and fixpoints

The aim of the first homework was to find a proper framework of definitions such that the objectives stated hold. For example, a classical definition of decreasing monotonicity for a function \( f : 2^S \to 2^S \), where \( S \) is a finite set, namely, for any two sets \( A, B \), \((A \subseteq B) \Rightarrow (f(A) \supseteq f(B))\), did not yield the desired result: any monotonically decreasing function \( f \) has a greatest fixpoint.\(^3\) An alternative definition, turning out more fruitful, is:

\(^3\)As a counter example consider \( S = \{1, 2\} \) and \( f(S) = \emptyset, f(\emptyset) = S, f(\{1\}) = \{2\} \) and \( f(\{2\}) = \{1\} \)
Definition 1. Given a finite set $S$, a function $f : 2^S \to 2^S$ is monotonically increasing (respectively decreasing) iff properties (A) and (B1) (respectively (A) and (B2)) hold, where

$$\forall A, B \subseteq S : (A \subseteq B) \Rightarrow (f(A) \subseteq f(B)) \quad \text{(A)}$$

$$\forall A \subseteq S : A \subseteq f(A) \quad \text{(B1)}$$

$$\forall A \subseteq S : A \supseteq f(A). \quad \text{(B2)}$$

Consider the algorithm given in Figure 2.

1: $i \leftarrow 0$
2: $X_i \leftarrow \emptyset$
Do
3: $i \leftarrow i + 1$
4: $X_i \leftarrow f(X_{i-1})$
Until $X_i \neq X_{i-1}$
5: return $X_i$

Figure 2: Algorithm to obtain least fixpoint of monotonically increasing $f$

Clearly, in case $f$ is monotonically increasing, the algorithm terminates and returns a fixpoint, since otherwise the series $(X_i)_{i \geq 0}$ would grow ad infinitum contradicting the finiteness of $S$. Let us call the returned result $X^\ast$. We will next show that $X^\ast$ is a least fixpoint: Consider an arbitrary fixpoint $Y \subseteq S$. Trivially $X_0 \subseteq Y$ holds. Since (A) holds for $f$, $f(X_0) \subseteq f(Y) = Y$. By induction it can be shown that for any $k \geq 0$, $f^k(X_0) \subseteq f^k(Y) = Y$, holds, too; and thus $X^\ast \subseteq Y$, i.e., $X^\ast$ is smaller than or equal to any other fixpoint — it is the least fixpoint.

Definition 2. The complement of function $f : 2^S \to 2^S$ is defined as $\bar{f} : 2^S \to 2^S$ with $\bar{f} : X \mapsto S \setminus f(S \setminus X)$.

The following proposition directly follows from the definition:

Proposition 4. If $f : 2^S \to 2^S$ is monotonically increasing (decreasing), then $\bar{f}$ is monotonically decreasing (increasing).

Proof. Assume $f$ is monotonically increasing. We will first show that $\bar{f}$ fulfills (A): consider any two sets $X, Y \subseteq S$ with $X \subseteq Y$.

$$X \subseteq Y \Rightarrow$$

$$S \setminus X \supseteq S \setminus Y \quad \text{from (A)}$$

$$f(S \setminus X) \supseteq f(S \setminus Y)$$

$$S \setminus f(S \setminus X) \subseteq S \setminus f(S \setminus Y) \Rightarrow$$

$$\bar{f}(X) \subseteq \bar{f}(Y).$$

From (B1) it follows that

$$S \setminus X \subseteq f(S \setminus X) \Rightarrow$$

$$S \setminus (S \setminus X) \supseteq S \setminus (f(S \setminus X)) \Rightarrow$$

$$X \supseteq \bar{f}(X).$$

The proof for $f$ being monotonically decreasing is analogous. Thus the proposition follows. \qed
An interesting result is,

**Proposition 5.** The least fixpoint of a monotonically increasing function \( f : 2^S \to 2^S \) is the complement of the greatest fixpoint of \( \bar{f} \), where the complement of set \( X \), \( \bar{X} \) is defined as \( \bar{X} = S \setminus X \).

**Proof.** The proof is in two parts: (i) we show that, if \( X \) is a fixpoint of \( f \), then \( \bar{X} \) is a fixpoint of \( \bar{f} \) and (ii) we show that if \( X \) is the least fixpoint of \( f \), \( \bar{X} \) is the greatest fixpoint of \( \bar{f} \).

**ad (i)** Let \( X \) be a fixpoint of \( f \).

\[
\bar{f}(\bar{X}) = S \setminus f(S \setminus \bar{X})
= S \setminus (S \setminus \bar{X})
= \bar{X}.
\]

**ad (ii)** Let \( X \) be the least fixpoint of \( f \). Then \( \bar{X} \) is a fixpoint of \( \bar{f} \). Assume there is a fixpoint \( Y \) of \( \bar{f} \). To prove that \( \bar{X} \) is the greatest fixpoint of \( \bar{f} \), it remains to show that \( Y \subseteq \bar{X} \). From (i) we know that \( Y \) is fixpoint of \( f \). Since, \( X \) is the least fixpoint of \( f \), \( Y \supseteq X \Rightarrow Y \subseteq \bar{X} \).

The proposition follows. \( \square \)

In the sequel we will denote \( \bar{X} \) as \( \neg X \) and make use of the equality \( \bar{f}(\neg X) = S \setminus f(X) \). Further we denote the least (greatest) fixpoint of \( f \) by \( \mu X. f \) (\( \nu X. f \)). We can thus restate Proposition 5 as

**Theorem 2.** Let \( f : 2^S \to 2^S \) be a monotonically increasing function. Then the equality \( \mu X. f = S \setminus (\nu X. \bar{f}) \) holds.

### 3 Symbolic Algorithm

In Section 1 an algorithm was presented that, given a graph \( G \) and target set \( T \), computes the set \( \langle \langle 1 \rangle \rangle (\cap T) \). Although it was shown to be optimal with respect to time complexity it turns out to be not well suited for practical problems, since it is based on traversing \( G \) state by state and maintaining a counter as well as a boolean marking for each state.

In this section an alternative algorithm is presented, which is not optimal, however, in practice shows up to have better runtimes. It iteratively increases sets of states \( X \subseteq S \) and is based on finding fixpoints. We may thus apply techniques discussed in Section 2. In practice it turns out that handling a set of states, which can be stored and manipulated in terms of BDDs, is often more efficient in time complexity than handling data structures the algorithm presented in Section 1 makes use of.

Let us define the operators \( Pre_1(X) \) and \( Pre_2(X) \) for \( X \subseteq S \):

\[
Pre_1(X) = X \cup \left( \{s \in S_1 \mid E(s) \cap X \neq \emptyset \} \cup \{s \in S_2 \mid E(s) \subseteq X \} \right)
\]

\[
Pre_2(X) = X \cap \left( \{s \in S_2 \mid E(s) \cap X \neq \emptyset \} \cup \{s \in S_1 \mid E(s) \subseteq X \} \right).
\]

In that latter we will be concerned with computing fixpoints for \( Pre_1 \) and \( Pre_2 \) respectively. We first deduce that
Proposition 6. $\text{Pre}_1$ is monotonically increasing.

Proof. We show that both (A) and (B1) hold.

ad (A) Assume that $X, Y \subseteq S$ and $X \subseteq Y$. Thus for any $s$ in $S$,
\[
E(s) \cap X \neq \emptyset \quad \Rightarrow \quad E(s) \cap Y \neq \emptyset \quad \text{and} \\
E(s) \subseteq X \quad \Rightarrow \quad E(s) \subseteq Y,
\]
and it follows that $\text{Pre}_1(X) \subseteq \text{Pre}_1(Y)$.

ad (B1) Is trivially fulfilled.
Thus the proposition follows. \hfill \Box

Analogously we obtain

Proposition 7. $\text{Pre}_2$ is monotonically decreasing.

Proof. We show that both (A) and (B2) hold.

ad (A) Assume that $X, Y \subseteq S$ and $X \subseteq Y$. Let us abbreviate (2) by $\text{Pre}_2(X) = X \cap \text{Rest}(X)$. Again for any $s$ in $S$,
\[
E(s) \cap X \neq \emptyset \quad \Rightarrow \quad E(s) \cap Y \neq \emptyset \quad \text{and} \\
E(s) \subseteq X \quad \Rightarrow \quad E(s) \subseteq Y,
\]
and it follows that $\text{Rest}(X) \subseteq \text{Rest}(Y)$. Combination with $X \subseteq Y$ yields,
\[
X \cap \text{Rest}(X) \subseteq Y \cap \text{Rest}(Y) \Rightarrow \\
\text{Pre}_2(X) \subseteq \text{Pre}_2(Y).
\]

ad (B2) Is trivially fulfilled.
Thus the proposition follows. \hfill \Box

From studying the definitions of $\text{Pre}_1$ and $\text{Pre}_2$ we observe: Consider an arbitrary set $X \subseteq S$ and $s$ from $S$. $s$ in $\text{Pre}_1(X)$ iff (i) it already is in $X$, or (ii) in case $s$ in $S_1$, it has at least one outgoing edge to a vertex in $X$; and in case $s$ in $S_2$, it has all its outgoing edges to a vertex in $X$. Conversely, $s$ in $\text{Pre}_2(\neg X)$ iff (i) it is in $S \setminus X$, and (ii) in case $s$ in $S_1$, it has all its outgoing edges to a vertex in $S \setminus X$; and in case $s$ in $S_2$, it has at least one of its outgoing edges to a vertex in $S \setminus X$. Thus
\[
\text{Pre}_2(\neg X) = S \setminus \text{Pre}_1(X). \quad (3)
\]

Given a set of target states $T$, we will next define the functions $f, f' : 2^S \to 2^S$ by
\[
f : X \mapsto T \cup \text{Pre}_1(X) \quad \text{and} \quad (4)
\]
\[
f' : X \mapsto \neg T \cap \text{Pre}_2(X). \quad (5)
\]

The following proposition holds:

Proposition 8. $f$ is monotonically increasing and $f' = f$. 

7
Proof. Because of Proposition 6, \( f \) is monotonically increasing.

It is next shown that \( f' = \bar{f} \). Consider an arbitrary \( X \subseteq S \). Then

\[
\bar{f}(X) = S \setminus f(S \setminus X) \\
= S \setminus (T \cup Pre_1(S \setminus X)) \\
= (S \setminus T) \cap (S \setminus Pre_1(S \setminus X)) \quad \text{and by (3)} \\
= \neg T \cap Pre_2(X) = f'(X).
\]

From Propositions 8 we deduce that we may apply Theorem 2 to the functions defined in (4) and (5), yielding

\[
\mu X.f = S \setminus \nu X.f'.
\]

We will next show that both, \( \mu X.f \) and \( \nu X.f' \) have an important meaning in terms of winning sets of player 1 and 2, respectively.

Since \( f \) is monotonically increasing, \( \mu X.f \) can be computed by the algorithm presented in Section 2, Figure 2, which obtains \( X_0 = \emptyset, X_1 = T, \ldots \) until it returns the fixpoint \( \mu X.f \). By definition of \( Pre_1 \), we further obtain: for all \( i \geq 0 \),

\[
X_i \subseteq \langle\langle 1 \rangle\rangle(\diamond T) \Rightarrow \\
\mu X.f \subseteq \langle\langle 1 \rangle\rangle(\diamond T).
\]

By definition of \( Pre_2 \), the fixpoint \( X^* = \nu X.\bar{f} \), must fulfill

\[
X^* = \neg T \cap Pre_2(X^*),
\]

i.e., it is possible for player 2 to stay in \( X^* \), which does not contain states form \( T \), forever: \( X^* \) only contains vertices \( s \) in \( S_1 \) with all successor states in \( X^* \) and vertices \( s \) in \( S_2 \) with at least one successor state in \( X^* \). Thus, with \( F = \neg T \),

\[
\nu X.\bar{f} \subseteq \langle\langle 2 \rangle\rangle(\Box F) \Rightarrow \\
S \setminus \nu X.\bar{f} \supseteq S \setminus \langle\langle 2 \rangle\rangle(\Box F) \quad \text{by Theorem 2 and (1)} \Rightarrow \\
\mu X.f \supseteq \langle\langle 1 \rangle\rangle(\diamond T) \quad \text{by (6)} \Rightarrow \\
\mu X.f = \langle\langle 1 \rangle\rangle(\diamond T).
\]

We finally obtain

**Theorem 3.** It holds that \( \mu X.f = \langle\langle 1 \rangle\rangle(\diamond T) \) and \( \nu X.\bar{f} = \langle\langle 2 \rangle\rangle(\Box F) \).

Thus the algorithm depicted in Figure 2, allows to compute \( \langle\langle 1 \rangle\rangle(\diamond T) \) and trivially terminates in at most \( O(n) \) symbolic steps.