

Übungen zu Grundlagen der Softwarezuverlässigkeit

Besprechung am 13.01.06

Aufgabe 5.1 CTL

Wir schreiben das Jahr 3000. Auf Grund stetig wachsender Bevölkerungszahlen und Unartigkeit kommt Santa Claus nicht mehr ohne Hilfe aus. Insbesondere die Anzahl der unartigen Menschen ist in den letzten Jahren gestiegen. Als friedfertiger Zeitgenosse verschenkt Santa Claus lieber Geschenke als Tadel zu verteilen. Daher plant er den unangenehmen Teil seiner Arbeit mit Hilfe modernster Technologie zu erleichtern. Von einer nicht näher bekannten, in Kalifornien beheimateten Firma wurde ihm hierfür der SantaRobot3000^(tm) verkauft.



Nach einer ersten Testphase (siehe Bild) entscheidet sich Santa Claus der Sicherheit wegen, das System Menschheit-SantaRobot auf gewisse Sicherheitseigenschaften hin zu testen:

- (a) Jeder Mensch hat immer die Möglichkeit, für alle Zeit artig zu sein.

Jeder Mensch hat also immer die Möglichkeit einen Zustand zu erreichen, von welchem ab er immer artig sein kann.

$$\mathbf{AG\ EF\ EG\ artig.}$$

- (b) Jede als unartig geltende Person soll auch irgendwann getadelt werden.

$$\mathbf{AG(\neg artig \rightarrow AF\ tadel)}$$

- (c) Jedem Tadel geht eine Unartigkeit voraus.

1. Interpretation: Es soll keinen Pfad geben, entlang welchem eine Person als artig gilt, jedoch schließlich getadelt wird.

$$\neg(\text{artig } \mathbf{EU}(\text{artig} \wedge \text{tadel}))$$

Dann könnte SantaRobot jedoch eine einmal unartig gewesene Person immer wieder tadeln, daher

2. Interpretation: SantaRobot darf nur den tadeln, der im Zeitschritt davor als unartig gegolten hat:

$$\mathbf{AG(\text{artig} \rightarrow \mathbf{AX}\ \neg\text{tadel})}$$

- (d) Eine Person, welche gerade als artig gilt, wird momentan nicht getadelt.

$$\mathbf{AG(\text{artig} \rightarrow \neg\text{tadel})}$$

(e) Eine Person, welche gerade getadelt wird, wird irgendwann nicht mehr getadelt.

$$\mathbf{AG}(\text{tadel} \rightarrow \mathbf{AF} \neg \text{tadel})$$

(f) Zwischen zwei Tadeln gilt die betroffene Person wenigstens kurzzeitig als artig.

Wir müssen Zustände abpassen, in welchen ein Tadel aufhört. Von solch einem Zustand aus sind dann nur noch Pfade zulässig, entlang welcher - die Person nie wieder getadelt wird - oder die Person erst wieder getadelt wird, nachdem ein Zustand besucht wurde, in welchem sie als artig galt: D.h. von einem solchen Zustand aus, darf kein Pfad existieren, entlang welchem die Person die ganze Zeit als unartig gilt und schließlich getadelt wird.

$$\neg \mathbf{EF}(\text{tadel} \wedge \mathbf{EX}(\neg \text{tadel} \wedge (\neg \text{artig} \mathbf{EU}(\neg \text{artig} \wedge \text{tadel}))))$$

oder äquivalent:

$$\mathbf{AG}(\text{tadel} \rightarrow \mathbf{AX}(\neg \text{tadel} \rightarrow \neg(\neg \text{artig} \mathbf{EU}(\neg \text{artig} \wedge \text{tadel}))))$$

(g) SantaRobot tadelt nie mehrere Personen gleichzeitig.

$$\mathbf{AG}(\text{tadel}_i \rightarrow \bigwedge_{j \neq i} \neg \text{tadel}_j)$$

(h) Solange SantaRobot eine Person tadelt, gilt diese als unartig.

$$\mathbf{AG}(\text{tadel} \rightarrow \neg \text{artig})$$

Formulieren Sie hierfür diese Eigenschaften in CTL. Die aussagenlogische Variable u_i soll dabei wahr sein, falls Mensch i unartig gewesen ist seit dem letzten Tadel durch SantaRobot. Entsprechend soll t_i wahr sein, falls SantaRobot gerade Mensch i tadelt. Verwenden Sie diese Variablen zum Formulieren der CTL-Formeln. (*Hinweis*: Gilt t_i über mehrere aufeinanderfolgende Zustände, dann wird dies als ein (langer) Tadel gewertet.)

Die Aussage, dass jeder Mensch immer wieder unartig sein kann, könnte dann durch

$$\bigwedge_{i=1}^{\#\text{Menschen}} \mathbf{AG} \mathbf{EF} u_i$$

ausgedrückt werden.

Bekanntlich muss Santa Claus ungefähr Lichtgeschwindigkeit erreichen, um alle Menschen an Weihnachten zu erreichen. Leider ist SantaRobot etwas schwer geraten, so dass die Beschleunigung von SantaRobot samt Wagen auf beinahe Lichtgeschwindigkeit Unmengen an Energie verschwendet. Es ist daher für Santa Claus kosteneffizienter, mehrere SantaRobots S_1, \dots, S_k anzuschaffen, um alle nötigen 'Besuche' an Weihnachten zu erledigen. Dabei darf natürlich nicht die Illusion zerstört werden, dass es nur einen Santa Claus gibt.

Es ist daher entscheidend, dass nie mehrere SantaRobots dieselbe Person gleichzeitig tadeln.

Geben Sie auch hierfür eine CTL-Formel an, wobei t_i durch t_i^k ersetzt wird, und t_i^k genau dann wahr ist, falls Mensch i gerade durch SantaRobot k getadelt wird.

Passen Sie entsprechend auch die CTL-Formeln für die Aussagen aus der letzten Teilaufgabe für das Heer von SantaRobots an.

Begleitend zu dieser Aufgabe finden Sie auf der Übungsseite ein Zip-Archiv. Darin enthalten sind die Dateien `SantaRobot(1|2).smv`, welche jeweils einen bzw. zwei SantaRobot(s) und die Menschheit als Prozesse beschreiben. Das Tool `smv` ermöglicht es über die Befehlszeile (`./smv ./SantaRobot*.smv`), die verschiedenen Fassungen von SantaRobot auf CTL-Eigenschaften zu testen. Dabei finden sich die CTL-Eigenschaften jeweils am Ende der `*.smv`-Dateien eingeleitet durch `SPEC`.

Ergänzen Sie diese Dateien um die CTL-Formeln aus den vorangegangenen Teilaufgaben und überprüfen Sie mit Hilfe von `smv`, ob die beiden Beschreibungen diese Eigenschaften erfüllen. Versuchen Sie notfalls die Beschreibungen so abzuändern, dass die Eigenschaften erfüllt sind.

Näheres zu `smv` finden Sie auf der Übungsseite und entnehmen Sie insbesondere den Kommentaren in beiden Dateien.

Aufgabe 5.2 CTL Model Checker

Auf der Webseite zur Übung finden Sie ein Archiv mit einem Programmgerüst für einen *sehr* einfachen CTL-Model-Checker. Ihre Aufgabe ist es, den Model Checker um die temporalen Operatoren zu ergänzen. Details hierzu entnehmen Sie bitte der README, welche dem Programmgerüst beigelegt ist.

```
#include "ctl.h"

namespace vsctlmc {

    BitSet
    EX( BitSet phi, const Kripke& K ) {
        // [EX phi] ist die Menge aller Zustände von K,
        // welche einen Nachfolger in der Menge [phi]
        // besitzen, d.h. wir brauchen einfach
        // die Menge aller Vorgänger von [phi]:
        return K.getPredecessorsOfBitSet( phi );
    }

    BitSet
    EG( BitSet phi, const Kripke& K ) {
        // [ EG phi ] = [ phi \wedge EX EG phi ]
        // -> X = [phi] \cap \delta^{-1}( X )
        //
        // Für EG müssen wir den globalen Fixpunkt entsprechend
        // der obigen Fixpunktgleichung berechnen.
        // Entsprechend beginnt die Fixpunktapproximation
        // mit allen Zuständen der Kripkestruktur,
        // siehe folgenden Befehl
        BitSet X = K.getAllStates();

        // Nachfolgende Schleife entspricht nun der Fixpunktiteration
        while( true ) {
            // Für C++-Unerfahrene 'bitset::' sagt dem Compiler,
            // dass er die Funktion 'intersect' im Namensbereich 'bitset'
            // findet - bzw. dort danach schauen soll.
            // bitset::intersect bildet dabei die Schnittmengen,
            // bitset::merge die Vereinigung,
            // bitset::complement das Komplement
            BitSet X_ = bitset::intersect( phi, EX( X, K ) );
            if( X_ == X ) break;
            X = X_;
        }

        return X;
    }

    // Die nachfolgenden Operatoren müssen von Ihnen noch
    // implementiert werden; bis jetzt geben Sie einfach die
    // leere Menge zurück.

    BitSet
    EU( BitSet phi0, BitSet phi1, const Kripke& K ) {
        BitSet X = phi1;

        while( true ) {
            // p EU q = q \vee p \wedge EX ( p EU q )
            // -> X = [q] \cup [p] \cap \delta^{-1}(X)
            BitSet X_ = bitset::merge( phi1, bitset::intersect( phi0, EX( X, K ) ) );
            if( X_ == X ) break;
            X = X_;
        }

        return X;
    }

    BitSet
    EF( BitSet phi, const Kripke& K ) {
        // EF p = p \vee EX EF p
        // -> X = [p] \cup \delta^{-1}( X )
        BitSet X = phi;

        while( true ) {
            BitSet X_ = bitset::merge( phi, EX( X, K ) );

```

```

    if( X_ == X ) break;
    X = X_;
}
return X;
}

BitSet
AX( BitSet phi, const Kripke& K ) {
    // AX p = - EX -p
    return bitset::complement( EX( bitset::complement( phi ), K ) );
}

BitSet
AG( BitSet phi, const Kripke& K ) {
    // AG p = p \wedge AX AG p
    // -> X = [p] \cap [AX( X )]
    //
    // Test: AG p = - EF -p = -
    BitSet X = K.getAllStates();

    while( true ) {
        BitSet X_ = bitset::intersect( phi, AX( X, K ) );
        if( X_ == X ) break;
        X = X_;
    }
    return X;
}

BitSet
AU( BitSet phi1, BitSet phi2, const Kripke& K ) {
    // p AU q = q \vee p \wedge \AX ( p AU q )
    // -> X = [q] \cup ( [p] \cap [ AX( X ) ] )
    BitSet X = phi2;

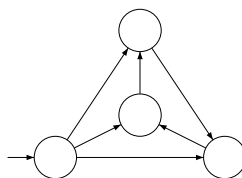
    while( true ) {
        BitSet X_ = bitset::merge( phi2, bitset::intersect( phi1, AX( X, K ) ) );
        if( X == X_ ) break;
        X = X_;
    }
    return X;
}

BitSet
AF( BitSet phi, const Kripke& K ) {
    // AF p = p \vee \AX \AF p
    // -> X = [p] \cup [AX( X )]
    BitSet X = phi;

    while( true ) {
        BitSet X_ = bitset::merge( phi, AX( X, K ) );
        if( X_ == X ) break;
        X = X_;
    }
    return X;
}
}
}

```

Aufgabe 5.3 CTL



Oben abgebildet finden Sie eine 'leere' Kripkestruktur. Sie sollen *jeweils* für jede Teilaufgabe angeben, in welchem Zustand welche atomare Proposition erfüllt ist, so dass die folgenden Formeln im Startzustand gelten, jedoch in

mindestens einem Zustand der jeweiligen Kripkestruktur nicht gelten.

Sollte dies nicht möglich sein, so begründen Sie warum.

- **AG** p .
- **(AG** p) \wedge **(AG** q)
- **EF**(p **AU** q)

Die ersten beiden Formeln erzwingen jeweils, dass jeder Zustand der Kripkestruktur mit p bzw. p und q beschriftet ist, da jeder Zustand vom Startzustand aus erreichbar ist. **AG** ϕ verlangt aber, dass ϕ entlang jedem Pfad zu jedem Zeitpunkt erfüllt ist.

Für die letzte Formel lassen sich die vorgegebenen Bedingungen erfüllen, indem einfach der Startzustand mit q beschriftet wird, während die restlichen Zustände unbeschriftet bleiben.

Aufgabe 5.4 **Vollständige Verbände - nicht prüfungsrelevant**

Hinweis:

Eine Aufgabe von diesem Typ wird nicht in der Prüfung vorkommen - also bitte nicht wegen dieser Aufgabe nicht mehr in die Vorlesung kommen ... ☹. Lernziel ist zum einen die Wiederholung der formalen Definitionen von partieller Ordnung, vollständigen Verbänden, Fixpunkten wie sie beim CTL-Model-Checking aufgetreten sind und gleichzeitige die Vorbereitung für die später in der Vorlesung behandelte Abstraktion, in welcher nochmals vollständige Verbände benutzt werden. Sich bitte auch nicht von der Länge der Aufgabenstellung 'einschüchtern' lassen. Die Länge resultiert maßgeblich daraus, dass die benötigten Definitionen bereits in den Aufgabentext eingebunden sind.

Und jetzt die Aufgabe:

Sei V eine Menge. Eine *partielle Ordnung* \sqsubseteq auf V ist eine Relation $\sqsubseteq \subseteq V \times V$ mit

$$\begin{aligned} \forall a \in V \quad a &\sqsubseteq a && \text{(Reflexivität)} \\ \forall a, b \in V \quad (a &\sqsubseteq b \wedge b \sqsubseteq a) \Rightarrow a = b && \text{(Antisymmetrie)} \\ \forall a, b, c \in V \quad (a &\sqsubseteq b \wedge b \sqsubseteq c) \Rightarrow a \sqsubseteq c && \text{(Transitivität)} \end{aligned}$$

(V, \sqsubseteq) wird dann als partiell geordnete Menge bezeichnet. Elemente $a, b \in V$ heißen (bzgl. \sqsubseteq) *unvergleichbar*, falls weder $a \sqsubseteq b$ noch $b \sqsubseteq a$ gilt.

Man mache sich klar, dass für jede beliebige nicht-leere Menge M $(2^M, \subseteq)$ eine partiell geordnete Menge ist und gebe ein Beispiel für bzgl. \subseteq unvergleichbare Elemente. (Hinweis: $2^M := \{T \mid T \subseteq M\}$.)

Als Beispiel: $M = \{a, b\}$, damit $2^M = \{\emptyset, \{a\}, \{b\}, \{a, b\}\}$, wobei $\{a\}$ und $\{b\}$ unvergleichbar sind.

Für eine Teilmenge $T \subseteq V$ heißt $o \in V$ *obere Schranke* von T , falls

$$\forall t \in T : t \sqsubseteq o,$$

das bedeutet somit auch, dass o mit allen Elementen aus T vergleichbar sein muss - wir schreiben hierfür kurz $T \sqsubseteq o$. Entsprechend heißt u *untere Schranke* von T , falls

$$\forall t \in T : u \sqsubseteq t,$$

oder kurz $u \sqsubseteq T$.

s heißt dann *Supremum* (kleinste obere Schranke) von T , falls

$$T \sqsubseteq s \wedge \forall o \in V : (T \sqsubseteq o \rightarrow s \sqsubseteq o)$$

gilt - soweit für T ein Supremum existiert, wird dieses mit $\sqcup T$ bezeichnet.

Entsprechend heißt i *Infimum* (größte untere Schranke) von T , falls

$$i \sqsubseteq T \wedge \forall u \in V : (u \sqsubseteq T \rightarrow u \sqsubseteq i).$$

Existiert das Infimum von T , so wird es mit $\sqcap T$ bezeichnet.

Man mache sich anhand der Definitionen klar, dass eine Menge nicht mehrere Suprema bzw. Infima bzgl. \subseteq haben kann.

Sind s und s' nach Definition Suprema von einer Menge $T \subseteq V$, dann sind beide nach Definition auch obere Schranken, so dass wiederum nach Definition auch $s \subseteq s'$ und $s' \subseteq s$ gelten muss. Da \subseteq als partielle Ordnung vorausgesetzt wurde, folgt aus $s \subseteq s' \wedge s' \subseteq s$ aber $s = s'$.

Entsprechend folgt, dass auch das Infimum eindeutig definiert ist - soweit es existiert.

Für $(2^M, \subseteq)$ und $T \subseteq 2^M$ (! Die Elemente von T sind Teilmengen von M) gebe man alle oberen und unteren Schranken an und zeige, dass $\sqcap T = \cap T$ und $\sqcup T = \cup T$ gilt (für $\subseteq := \subseteq$). Dabei sei $\cap T := \bigcap_{a \in T} a$ der Durchschnitt aller Elemente von T , entsprechend $\cup T := \bigcup_{a \in T} a$ die Vereinigung aller Elemente von T .

o ist eine obere Schranke von $T \subseteq 2^M$ genau dann, wenn für alle $a \in T : a \subseteq o$ gilt, wenn somit $\cup T \subseteq o$. Damit ist $\cup T$ selbst eine obere Schranke und auch kleiner gleich jeder anderen oberen Schranke, nach Definition also das Supremum $\sqcup T$.

Entsprechend für u untere Schranke von $T \Leftrightarrow \forall a \in T : u \subseteq a \Leftrightarrow u \subseteq \cap T$. Also ist $\cap T$ ebenfalls eine untere Schranke und offensichtlich bzgl. \subseteq die grösste, also $\cap T = \sqcap T$.

Ein vollständiger Verband (V, \subseteq) ist nun eine Struktur mit

- (a) (V, \subseteq) ist eine partiell geordnete Menge.
- (b) Jede Teilmenge T von V besitzt ein Supremum $\sqcup T$ in V .

Zeigen Sie, dass in einem vollständigen Verband jede Teilmenge $T \subseteq V$ auch bereits ein Infimum besitzt, welches durch $\sqcap T = \sqcup \{a \in V \mid a \subseteq T\}$ gegeben ist.

Zur Abkürzung sei $U := \{a \in V \mid a \subseteq T\}$. U ist also die Menge der unteren Schranken von T . Wir müssen $\sqcap T = \sqcup U$ zeigen, d.h. das Supremum der unteren Schranken von T ist gleich dem Infimum von T , d.h. nach Definition des Supremums müssen wir zeigen:

- $\sqcup U$ ist eine untere Schranke von $T : \forall t \in T : \sqcup U \subseteq t$.

Das folgt einfach daraus, dass jedes $t \in T$ eine obere Schranke von U ist, da jedes $u \in U$ kleiner gleich bzgl. \subseteq jedem $t \in T$ ist.

- $\sqcup U$ ist größer gleich jeder anderen unteren Schranke von T .

Jede untere Schranke von T ist aber in U enthalten und $\sqcup U$ ist größer gleich jedem Element aus U .

Wie Sie oben gezeigt haben, ist $(2^M, \subseteq)$ damit auch ein vollständiger Verband.

In einem vollständigen Verband existieren nach Definition auch die Elemente $\perp := \sqcup \emptyset$ und $\top := \sqcup V$. Zeigen Sie, dass für alle $a \in V$ $\perp \subseteq a \subseteq \top$ gilt mit dieser Definition. Welche Elemente sind durch $\sqcap \emptyset$ und $\sqcap V$ gegeben?

Für \top folgt nach Definition von $\sqcup V$:

$$(V \subseteq \top) \wedge \forall o \in V : (V \subseteq o) \rightarrow \top \subseteq o.$$

Insbesondere also auch $V \subseteq \top$, was zu zeigen war.

Wir betrachten $\perp = \sqcup \emptyset$, d.h. \perp erfüllt nach Definition von \sqcup :

$$(\emptyset \subseteq \perp) \wedge \forall o \in V : (\emptyset \subseteq o) \rightarrow \perp \subseteq o.$$

Nun ist $\emptyset \subseteq o$ als $\forall a \in V : (a \in \emptyset) \rightarrow a \subseteq o$ definiert. Da die Bedingung $a \in \emptyset$ nie erfüllt ist, ist die Implikation immer erfüllt, so dass für \perp aus obigem folgt:

$$\forall a \in V : \perp \subseteq a.$$

Zunächst gilt $\sqcap \emptyset = \sqcup \{a \in V \mid a \subseteq \emptyset\} = \sqcup V = \top$. Für $\sqcap V$ folgt wegen $\perp \in V$, dass $\sqcap V \subseteq \perp$ gilt, also $\sqcap V = \perp$.

\mathbb{N} seien die natürlichen Zahlen einschließlich der 0. Für $a, b \in \mathbb{N}$ gelte $a \mid b$ genau dann, falls a b teilt, d.h. falls ein $k \in \mathbb{N}$ existiert mit $a \cdot k = b$. Zeigen Sie, dass \mathbb{N} mit \mid so zunächst zwar eine partiell geordnete Menge ist, jedoch noch kein vollständiger Verband. Erweitern Sie \mathbb{N} um ein Element

(und \sqsubseteq entsprechend), so dass ein vollständiger Verband entsteht. Was ist dann das Infimum und das Supremum in diesem vollständigen Verband (insbesondere für nicht-leere endliche Teilmengen)?

Fehler in der Aufgabenstellung: es hätte "ohne die 0" heißen sollen; mit der 0 handelt es sich bereits um einen vollständigen Verband.

Aus $a|b$ und $b|a$ folgt $b = ka$ und $a = k'b$, also $a = kk'a$ oder $kk' = 1$, also über \mathbb{N} $k = k' = 1$ und damit $a = b$. Also ist $|$ antisymmetrisch. Weiter gilt mit $k = 1$ auch $a|a$, also ist $|$ reflexiv. Die Transitivität folgt entsprechend aus $a|b \Rightarrow b = ka$ und $b|c \Rightarrow c = lb$ mit $c = kla$, also $a|c$.

Damit ist $(\mathbb{N}, |)$ eine partiell geordnete Menge. Weiter folgt mit $k = b$ für alle $n \in \mathbb{N}$ $1|n$, also $\perp = 1$. Nach Definition von $|$ folgt weiter, dass für jedes $n \in \mathbb{N}$ $n|0$ gilt, d.h. $\top = 0$.

Für \emptyset folgt, dass jedes $n \in \mathbb{N}$ eine untere und obere Schranke ist, also $\sqcup \emptyset = 1$ und $\sqcap \emptyset = 0$.

Sei $M \subseteq \mathbb{N}$ eine nicht leere, endliche Menge, also $M = \{m_1, \dots, m_k\}$. Dann folgt dass $\sqcup M = \text{kgV}(M)$ und $\sqcap M = \text{ggT}(M)$ gilt.

Ist M dagegen nicht leer und unendlich, also $M = \{m_1, m_2, \dots\}$, so ist M unbeschränkt, so dass für jedes $s > 0$ folgt, dass ein $m \in M$ existiert mit $m > s$ und damit mit $m \not\leq s$. Also folgt $\sqcup M = 0$. Für das Infimum $\sqcap M$ betrachte man die Folge $g_i := \text{ggT}(m_1, \dots, m_i)$ mit $g_1 = m_1$. Dann gilt $g_{i+1}|g_i$, die Folge ist also bzgl. $|$ monoton fallend und nach unten durch 1 beschränkt, konvergiert somit bzw. wird stationär gleich einem $g \in \mathbb{N}$. (Es kann nur endlich viele k 's mit $g_{k+1}|g_k \wedge g_{k+1} \neq g_k$ geben.) Jede andere untere Schranke von M muss bereits die g_i 's und damit auch g teilen, so dass g das Infimum von M sein muss.

Im Fall $0 \notin \mathbb{N}$ folgt - wie gerade gezeigt, dass keine unendliche Menge $M \subset \mathbb{N}$ ein Supremum $s > 0$ bzgl. $|$ besitzen kann. Üblicherweise würde man nun ∞ zu \mathbb{N} noch hinzunehmen mit der Definition, dass jedes $n \in \mathbb{N}$ ∞ teilt. ∞ fällt bzgl. $|$ aber mit 0 in diesem Fall zusammen.

Man überlege sich, wie sich aus zwei vollständigen Verbänden (V_1, \sqsubseteq_1) und (V_2, \sqsubseteq_2) auf natürliche Weise ein vollständiger Verband über $V_1 \times V_2$ ergibt.

Setze $V := V_1 \times V_2$ mit $(a, b) \sqsubseteq (c, d) :\Leftrightarrow a \sqsubseteq_1 c \wedge b \sqsubseteq_2 d$. Bezeichnet dann π_i die Projektion auf die i -te Komponente, so ergibt sich $\sqcup T = (\sqcup \pi_1(T), \sqcup \pi_2(T))$.

Bei der Berechnung von **EG** und **EU** sind ganz natürlich *monotone* Funktionen und *Fixpunkte* aufgetreten. Diese sollen noch etwas genauer untersucht werden:

Eine Abbildung $f : V \rightarrow V$ heißt *monoton*, falls $\forall a, b \in V : a \sqsubseteq b \Rightarrow f(a) \sqsubseteq f(b)$ gilt. Ein $a \in V$ heißt *Fixpunkt* einer (beliebigen) Abbildung $f : V \rightarrow V$, falls $f(a) = a$ gilt.

a heißt *kleinster Fixpunkt* von f , falls $f(a) = a \wedge \forall b \in V : b = f(b) \Rightarrow a \sqsubseteq b$.

Wir schreiben μf für den kleinsten Fixpunkt von f - soweit er existiert.

a heißt *größter Fixpunkt* von f , falls $f(a) = a \wedge \forall b \in V : b = f(b) \Rightarrow b \sqsubseteq a$.

Der größte Fixpunkt wird mit νf abgekürzt.

Es sei f monoton und (V, \sqsubseteq) ein vollständiger Verband. Zeigen Sie, dass dann

$$\mu f = \sqcap \{a \in V \mid f(a) \sqsubseteq a\}$$

und

$$\nu f = \sqcup \{a \in V \mid a \sqsubseteq f(a)\}.$$

gilt. [Knaster-Tarski-Theorem]

Hinweis: Die Menge $\text{Post}_f := \{a \mid f(a) \sqsubseteq a\}$ heißt die Menge der **Postfixpunkte** von f . Sei $i := \sqcap \text{Post}_f$. Offensichtlich ist jeder **Fixpunkt** von f auch ein **Postfixpunkt**, so dass i auf jeden Fall kleiner-gleich jedem **Fixpunkt** ist. Es ist also $f(i) = i$ zu zeigen bzw. äquivalent hierzu $i \sqsubseteq f(i)$ und $f(i) \sqsubseteq i$. Zeigen Sie zunächst $f(i) \sqsubseteq i$ oder äquivalent hierzu, dass i selbst ein **Postfixpunkt** ist. Verwenden Sie dies, um zu zeigen, dass $f(i)$ ebenfalls ein **Postfixpunkt** ist, und schließen Sie damit, dass $i \sqsubseteq f(i)$ gilt.

Gehen Sie entsprechend für νf vor. Die Menge $\text{Prä}_f := \{a \mid a \sqsubseteq f(a)\}$ heißt dabei entsprechend die Menge der **Präfixpunkte von f .**

Wir gehen entsprechend dem Hinweis vor und zeigen zunächst $f(i) \sqsubseteq i$.

*Da $i \sqsubseteq a$ für $a \in \text{Post}_f$ und f monoton folgt $f(i) \sqsubseteq f(a)$. Da a ein **Postfixpunkt** ist, folgt weiter $f(i) \sqsubseteq f(a) \sqsubseteq a$. Also ist $f(i)$ eine untere Schranke von Post_f und damit $f(i) \sqsubseteq i$. Somit ist i selbst ein **Postfixpunkt** und damit auch der kleinste, also auch kleiner gleich jedem anderen **Fixpunkt**.*

Es bleibt $i \sqsubseteq f(i)$. Da $f(i) \sqsubseteq i$ gezeigt ist, folgt aus der Monotonie von f $f(f(i)) \sqsubseteq f(i)$. Also ist auch $f(i)$ ein Postfixpunkt, d.h. $f(i) \in \text{Post}_f$. Dann folgt aber sofort $i \sqsubseteq f(i)$, da nach Voraussetzung $i \sqsubseteq \text{Post}_f$.

Wegen der vorausgesetzten Antisymmetrie von \sqsubseteq folgt damit $i = f(i)$. Also ist i der kleinste Fixpunkt von f .

Ganz analog ergibt sich dann $\nu f = \sqcup\{a \in V \mid a \sqsubseteq f(a)\}$.

Vor allem besagt das Knaster-Tarski-Theorem also, dass jede monotone Funktion auf einem vollständigen Verband einen eindeutig bestimmten kleinsten und größten Fixpunkt besitzt (**Warum?**).

Nach Definition eines vollständigen Verbandes existieren das Supremum und Infimum auf der rechten Seite.

Bei der Berechnung der CTL-Operatoren haben Sie gesehen, dass sich $\mathbf{AG} \phi$ durch $\neg \mathbf{EF} \neg \phi$ berechnen lässt, d.h. der implizit in \mathbf{AG} vorkommende größte Fixpunkt wird durch den kleinsten Fixpunkt aus \mathbf{EF} ausgedrückt - d.h. größter und kleinster Fixpunkt scheinen *dual* zu einander zu sein.

Betrachten Sie nun den Potenzmengenverband über der Menge $A \neq \emptyset$ (im Fall einer Kripkestruktur wäre A also gerade die Menge der Zustände). Das Komplement einer Menge definiert dann eine Abbildung

$$c : 2^A \rightarrow 2^A : X \mapsto A \setminus X$$

auf 2^A (Nebenbemerkung: eine sogenannte fixpunktfreie Involution). Bekanntlich gilt $c \circ c = \text{Id}$ und $X \subseteq Y \Leftrightarrow c(X) \supseteq c(Y)$.

Für eine Abbildung $f : 2^A \rightarrow 2^A$ ist die *duale* Abbildung f' dann durch $f' := c \circ f \circ c$ definiert.

Zeigen Sie, für monotonen f ist auch f' monoton, und weiter, dass $\mu f = c(\nu f')$ und $\nu f = c(\mu f')$ gilt. Hinweis:

$$\begin{aligned} x \in c\left(\bigcap_{a \in T} a\right) & \\ \Leftrightarrow \neg(x \in \bigcap_{a \in T} a) & \\ \Leftrightarrow \neg(\forall a \in T : x \in a) & \\ \Leftrightarrow \exists a \in T : x \notin a & \\ \Leftrightarrow \exists a \in T : x \in c(a) & \\ \Leftrightarrow x \in \bigcup_{a \in T} c(a) & \end{aligned}$$

Sei $a, b \in 2^A$ mit $a \subseteq b$. Dann folgt $c(a) \supseteq c(b)$ und aus der Monotonie von f $f(c(a)) \supseteq f(c(b))$, also auch $c(f(c(a))) \subseteq c(f(c(b)))$ - also ist auch $f' = c \circ f \circ c$ monoton.

Man betrachte dann:

$$\begin{aligned} c(\nu f') &= c(\bigcup\{a \in 2^A \mid a \subseteq c \circ f \circ c(a)\}) \\ &= \bigcap\{c(a) \in 2^A \mid a \subseteq c \circ f \circ c(a)\} \\ &= \bigcap\{c(a) \in 2^A \mid c(a) \supseteq f(c(a))\} \\ &= \bigcap\{b \in 2^A \mid b \supseteq f(b)\} \\ &= \mu f \end{aligned}$$