UNIVERSITÄT ZU LÜBECK
INSTITUT FÜR INFORMATIONSSYSTEME

Evaluation eines Algorithmus zur Absicherung von Datenbanken vor probabilistischer Inferenz

Evaluation of an Algorithm for Securing Databases from Probabilistic Inference

Bachelorarbeit

verfasst am
Institut für Informationssysteme

im Rahmen des Studiengangs
IT-Sicherheit
der Universität zu Lübeck

vorgelegt von
Kevin Schmelzer

ausgegeben und betreut von
Prof. Dr. Ralf Möller

mit Unterstützung von
Simon Schiff

Lübeck, den 18. August 2020

Eidesstattliche Erklärung

Ich erkläre hiermit an Eides statt, dass ich diese Arbeit selbständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe.

_____

Kevin Schmelzer

Zusammenfassung

Die Vertraulichkeit sensibler Daten erfordert einen besonderen Schutz, insbesondere im medizinischen Bereich in dem mit Patientendaten gearbeitet wird. Dafür werden Zugangskontrollmechanismen verwendet, um Anfragen an sensible Daten zu verweigern. Diese Mechanismen beachten jedoch nicht, dass durch Kombinierung von statistischen Auswertungen und die Ergebnisse klug gewählter Anfragen auf unsensible Daten, trotzdem Informationen zu sensiblen Daten hergeleitet werden können. Eine Lösung, um solche Angriffe zu verhindern, ist die Database Inference Control (DBIC). Ein DBIC-Mechanismus ist der Prototyp Angerona, der anhand von Definitionen von probabilistischen Abhängigkeitsregeln, den Zugriff auf sensible Daten verweigert. In dieser Arbeit wird Angerona evaluiert, da dieser bisher nicht noch nicht auf echten Daten empirisch evaluiert wurde. Dafür wird die Sicherheit getestet und die Laufzeit gemessen. Dabei wurde festgestellt, dass die Laufzeit für praxisnahe Anwendungsfälle zwar möglich ist, jedoch zu hohe Laufzeiten für Anfragen aufweist, um diese in echten Systemen effektiv nutzen zu können.

Abstract

Englische Abstract

Inhaltsverzeichnis

1

Einleitung

Der Schutz von sensiblen Daten ist für viele Unternehmen und andere Einrichtungen wichtig, die persönliche Daten, wie Herkunft, Religion, Alter usw., erfassen, verarbeiten und speichern. Die Relevanz des Themas Privatsphäre und Digitalisierung ist spätestens seit der Wirksamkeit der Europäischen Datenschutz-Grundverordnung (DSGVO) [4] im Mai 2018 bemerkbar, da die dadurch entstandenen Herausforderungen auf Unternehmensseite diese vor rechtliche und insbesondere auch technische Probleme stellt, weshalb DSGVO-konforme Software-Lösungen entwickelt wurden, um die Arbeit zu erleichtern [10].
In dieser Arbeit wird ein besonderer Fokus auf die Privatsphäre in Krankenhäusern gelegt, da in Krankenhäusern viele sensible und persönliche Daten über Patienten erfasst werden. Dabei gab es schon vor der Einführung der DSGVO eine andere Regelung zum Schutz der Privatsphäre speziell im medizinischen Bereich und zwar die Health Insurance Portability and Accountability (HIPAA) aus dem Jahr 1996 aus den USA. Hierbei wird für jeden der Gesundheitsdaten verarbeitet oder speichert, durch Anordnungen und Regelungen zur Verarbeitung von Daten vorgeschrieben, die Vertraulichkeit der Daten der Patienten durch z.B. Zugangskontrolle, Anonymisierung, richtige Datenspeicherung usw. zu schützen [8].
Bisher wurde die Vertraulichkeit von sensiblen Daten geschützt, indem der Zugang durch Zugangskontrollmechanismen verboten wurde [11]. Jedoch existieren noch weitere Möglichkeiten, um Informationen zu sensiblen Daten zu erhalten. Daher erfordert der Schutz der Vertraulichkeit von sensiblen Daten einen Schutz vor direktem und indirektem Zugriff auf eine Datenbank. Der direkte Zugriff beschreibt dabei den Zugang zu Daten aus einer Datenbank und wird durch Zugangskontrollen geschützt, indem Zugriffsrechte auf bestimmte Daten oder Tabellen gewährt werden. Beim indirekten Zugriff hingegen, wird versucht durch statistische Auswertungen von externen Informationen und klug gewählten Anfragen an die Datenbank, an sensible Daten zu kommen. Um den direkten Zugriff zu verhindern gibt es einige neue Ansätze und eine davon ist die Database Inference Control (DBIC) [19]. Der DBIC-Mechanismus, der in dieser Arbeit evaluiert wird ist Angerona [7], wovon erstmals nur ein Prototyp existiert.
Der DBIC-Mechanismus Angerona sichert die Datenbank ab, indem eine probabilistische logische Programmiersprache (`Problog`) verwendet wird, um das Vorwissen eines Angreifers darzustellen. Dafür wird die Sprache `ATKLOG` entwickelt, die auf `Problog` basiert

und durch probabilistische Abhängigkeitsregeln das initiale Vorwissen des Angreifers modelliert. Ein Angreifermodell wird benötigt, um das initiale Vorwissen von jedem Benutzer zu speichern. Um ein Datenbankmodell nach Angerona zu übertragen, werden zuerst alle probabilistischen Abhängigkeitsregeln in ein Bayes-Netz modelliert. Anschließend wird aus dem Bayes-Netz ein Angreifermodell erstellt. Mithilfe von Sicherheitsregeln wird dann ein Schwellwert definiert, der den Zugang zum Datenbanksystem nur zulässt, wenn das Vorwissen des Angreifers unter dem Schwellwert liegt.
Damit DBIC-Mechanismen wie Angerona auch effektiv den indirekten Zugang schützen, müssen diese eine große Menge von probabilistischen Abhängigkeiten abdecken können, um viele verschiedene Angreifermodelle darzustellen und es muss eine angemessene Laufzeit aufweisen, um diese auch auf reale und große Datenbanken anwenden zu können.
Die Laufzeit wird dabei in *Online-* und *Offlinezeiten* unterteilt, wobei die `Onlinezeit` das Intervall zwischen dem Start der Anfrage und der Antwort ist und die `Offlinezeit` vom Start des Systems bis das System bereit für eine Eingabe ist.
Der Unterschied zu bisherigen DBIC-Mechanismen zu Angerona ist, dass die bisher nur eine begrenzte Anzahl von probabilistischen Abhängigkeiten erlaubt haben und somit nicht für reale Datenbanken tauglich sind. Angerona hingegen soll auch bei komplexen probabilistischen Abhängigkeiten eine angemessene Laufzeit haben und somit in der Praxis nutzbar sein [7].

1.1 Beiträge dieser Arbeit

In dieser Arbeit wird ein Verfahren vorgestellt, dass ein Bayes-Netz aus einem Datenbankmodell erstellt und in ein Angreifermodell aus `ATKLOG` überträgt und es somit mit Angerona ausführbar macht. Außerdem werden die Datenbanken MIMIC-III und eICU für die Krankheit Krebs durch Angerona abgesichert und gezeigt, dass für das MIMIC-III-Beispiel eine Offlinelaufzeit von 77 Minuten und für eICU 89 Minuten und die Onlinezeiten für MIMIC-III 36.4 Sekunden und für eICU 23.6 Sekunden besitzt und somit nicht effektiv in der Praxis abgesichert werden können. Anschließend wird gezeigt, dass die Offlinezeit abhängig ist von der Größe der zu initialisierenden Datenbank und die Onlinezeit von der Größe und Komplexität vom Angreifermodell.

1.2 Verwandte Arbeiten

Es existieren viele Arbeiten, die auch versuchen den indirekten Zugang zu Datenbanken zu verhindern. Es gibt einige Ansätze die auch implementierte DBIC-Mechanismen evaluiert haben und zum Ergebnis kommen, dass diese sicher und effizient sind , jedoch wurden diese nicht in realen Anwendungsfällen getestet [17].
Außerdem unterstützen die meisten DBIC-Mechanismen keine probabilistische Abhängigkeiten sondern nur einfache funktionale Abhängigkeiten.

1.3 Aufbau dieser Arbeit

In Kapitel 2 wird die Funktionsweise von Angerona beschrieben und aus einem Bayes-Netz ein Angreifermodell modelliert und mit Angerona initialisiert. Anschließend werden in Kapitel 3 die genutzten pseudonymisierten und synthetischen Datenbanken vorgestellt und dazu geeignete Angreifermodelle modelliert. Zum Schluss wird in Kapitel 4 die Sicherheit und die Online- und Offlinelaufzeit von Angerona geprüft und Kriterien analysiert, die sich auf die Laufzeit auswirken.

2

Grundlagen von Angerona

2.1 Grundlegendes

Systemmodell

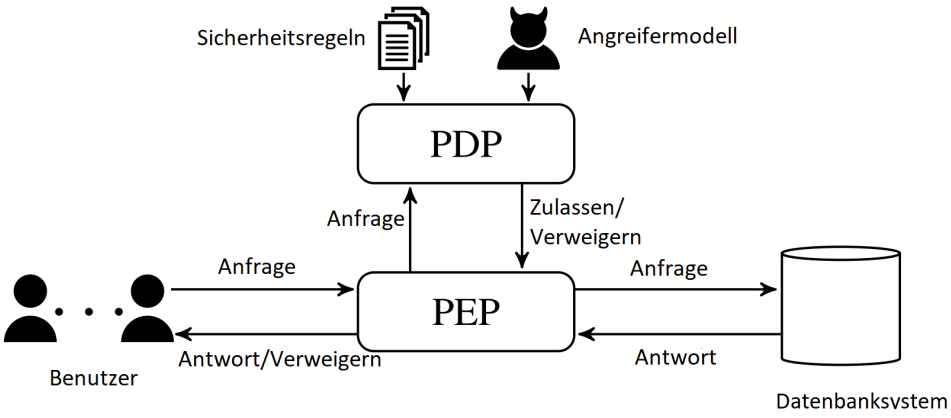


Abbildung 2.1: Systemmodell

Abbildung 2.1 zeigt das von Angerona genutzte Systemmodell. Dabei interagiert der Benutzer mit dem Inferenzkontrollsystem, welches aus den zwei Komponenten Policy Decision Point (PDP) und dem Policy Enforcement Point (PEP) besteht. Das Inferenzkontrollsystem entscheidet Anhand von vordefinierten Sicherheitsregeln und einem Angreifermodell, ob die Anfrage des Benutzers an das Datenbanksystem übergeben wird. Dabei wird davon ausgegangen, dass alle Anfragen und Antworten aus dem Systemmodell über sichere Kommunikationskanäle laufen. Außerdem gilt als Voraussetzung, dass jeder Benutzer das Datenbankschema und die Sicherheitsregeln kennt.

Datenbanksystem Das Datenbanksystem verwaltet alle Daten und antwortet auf Anfragen, um die Daten auszugeben.
Benutzer Jeder Benutzer hat ein eigenes Konto um Informationen zu erhalten, indem `SELECT` Anfragen an das Inferenzkontrollsystem gestellt werden. Dabei hat jeder Benut-

zer nur Leserechte und kann somit den Datenbankzustand nicht verändern. Jede Anfrage wird vom Inferenzkontrollsystem geprüft und wird nur ausgeführt, wenn diese von den Sicherheitsregeln autorisiert wird.

Sicherheitsregeln Die Sicherheitsregeln bestehen aus einer Menge von Regeln, die definieren, welche Informationen geheim gehalten werden sollen. Diese Regeln definieren die Erwartungen jedes Benutzers über den Inhalt der Datenbank als Wahrscheinlichkeitsverteilung. Die Regeln werden formalisiert durch ein Kommando in der Form `SECRET` q `FOR` u `THRESHOLD` l, wobei q die Anfrage, u den Benutzer und l den Grenzwert darstellt, bei der die Anfrage genehmigt werden darf,mit $0 \leq l \leq 1$. Eine Sicherheitsregel „Der Benutzer u ist nicht autorisiert die Antwort von der Anfrage q zu erfahren“ wird ausgedrückt mit `SECRET` q `FOR` u `THRESHOLD 1`. Außerdem kann die Regel „Für alle Benutzer $u \notin \{u_1, \ldots, u_n\}$, muss die Erwartung von u bei der Anfrage q kleiner als l sein“ mit dem Kommando „`SECRET` q `FOR USERS NOT IN` $\{u_1, \ldots, u_n\}$ `THRESHOLD 1`“ ausgedrückt werden.

Angreifer Ein potentieller Angreifer ist jeder Benutzer des Datenbanksystems mit einem Benutzerkonto. Das Ziel eines Angreifers ist es die Sicherheitsregeln zu verletzen indem er mindestens auf ein `SECRET` q schließen kann mit einer Wahrscheinlichkeit unter dem dazugehörigen `THRESHOLD` l.

Der Angreifer interagiert mit dem Inferenzkontrollsystem, indem er Anfragen stellt und somit neue Informationen aus den Antworten erhält und Daten die in Beziehung zueinander stehen feststellen kann. Ein Beispiel für eine Beziehung zwischen Daten ist zum Beispiel, dass wenn ein Patient raucht, dass die Wahrscheinlichkeit für Krebs für Ihn erhöht ist.

Datenbankzustand Der Datenbankzustand beschreibt die Belegung der Datenelemente in der Datenbank.

Angreifermodell Das Angreifermodell definiert das Vorwissen des Benutzers über den aktuellen Datenbankzustand. Im Angreifermodell werden auch die probabilistischen Abhängigkeitsregeln definiert, die ein Angreifer benötigt, um Informationen zu sensiblen Daten zu erhalten. Dies führt auch dazu, dass das Angreifermodell von Angerona aktualisiert wird, sobald sich das Vorwissen über eine Abhängigkeit verändert, wenn der Angreifer mit der Datenbank interagiert.

Inferenzkontrollsystem Das Inferenzkontrollsystem schützt die Vertraulichkeit der Daten in der Datenbank. Es besteht aus dem PEP und dem PDP und wird mit Sicherheitsregeln und einem Angreifermodell konfiguriert. Die PEP beobachtet dabei für jeden Benutzer das Vorwissen aus dem Angreifermodell und fängt alle Anfragen vom Benutzer ab und leitet diese an den PDP weiter. Der PDP entscheidet dann, ob der Benutzer autorisiert ist die Anfrage auszuführen. Dafür wird geprüft, ob die Anfrage die Sicherheitsregeln erfüllt und wenn das der Fall ist, dann wird dem PEP mitgeteilt, dass die Anfrage zugelassen ist. Anschließend leitet der PEP die Anfrage an das Datenbanksystem weiter und schickt die Antwort an den Benutzer.

Wenn der Benutzer nicht autorisiert ist, weil die Anfrage nicht die Sicherheitsregeln erfüllt, wird eine `security exception` ausgelöst und die Anfrage wird verweigert [16].

Bayes-Netze

Für jede Implementierung, die in Angerona vorgenommen wird, hilft es die probabilistischen Abhängigkeiten mit einem Bayes-Netz graphisch darzustellen. Dann kann das Bayes-Netz direkt in ein Angreifermodell für Angerona übersetzt werden. In einem Angreifermodell von Angerona ist es ohne Bayes-Netz schwer die Übersicht über die probabilistischen Abhängigkeiten zu behalten.

Ein Bayes-Netz ist ein direkter, azyklischer Graph in dem jeder Knoten die gemeinsamen Wahrscheinlichkeitsverteilungen seiner Zufallsvariable enthält. Dabei hat jeder Knoten V_i selbst eine Wahrscheinlichkeitsverteilung $P(V_i \mid Parent(V_i))$. Die Elternbeziehung eines Knotens gibt dabei an, dass $Parent(V_i)$ einen direkten Einfluss auf V_i hat und wird mit einem gerichteten Pfeil dargestellt [15].
Ein Beispiel für solch ein Bayes-Netz ist in Abbildung 2.2 zu sehen, indem der Start des Motors davon abhängig ist, ob die Batterie OK ist und ob genug Benzin vorhanden ist. Die Abhängigkeit kann formal beschrieben werden durch $P(m \mid b, e)$. Die Wahrscheinlichkeit, dass der Motor startet ist dabei 0.99, wenn genug Benzin und die Batterie in Ordnung ist. Zum Beispiel die Wahrscheinlichkeit, dass der Motor startet, obwohl nicht genug Benzin vorhanden ist und die Batterie nicht in Ordnung ist beträgt 0.03.
Eine zusätzliche Voraussetzung die Angerona vorschreibt ist, dass die Bayes-Netze die Eigenschaften eines Polytree erfüllen müssen. Die Eigenschaften dafür sind in Theorem 2.1 definiert.

Definition 2.1 (Polytree). Ein Polytree ist ein direkter, azyklischer Graph, der azyklisch bleibt, selbst wenn alle gerichtete Kanten durch ungerichtete Kanten ersetzt werden.

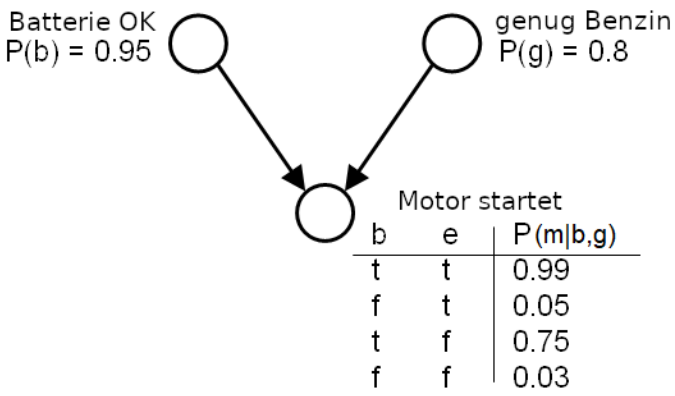


Abbildung 2.2: Bayes-Netz-Beispiel [6]

Problog

`Problog` ist eine probabilistische Erweiterung von `Prolog` (PROgramming in LOGic), wobei `Prolog` eine logische Programmiersprache ist, die aus verschiedenen Klauseln k_i

und aus logischen Verknüpfungen ein Ergebnis berechnet. `Problog` erweitert jedes k_i mit einer Wahrscheinlichkeit p_i, wodurch dies Wahrscheinlichkeiten ausgibt, mit welcher die k_i eintreten, wohingegen `Prolog` nur ein Ergebnis nach logischen Berechnungen liefert und somit keine Angaben zu Wahrscheinlichkeiten unterstützt.
Ein Probabilistischer Fakt lässt sich durch $p_i :: k_i$ definieren und weißt einer Klausel die Wahrscheinlichkeit zu. Eine logische Verknüpfung $\wedge$ wird in `Problog` mit einem „,“ dargestellt und eine Negation mit „\+“. Ein Beispiel für ein `Problog` Programm ist in Programmtext 2.1 zu sehen.

Programmtext 2.1: `Problog` Programm-Beispiel

```
1 0.1::erdbeben.
2 0.9::alarm_bei_erdbeben.
3 alarm :- erdbeben, alarm_bei_erdbeben.
```

Ein `Problog` Programm $T = p_1 :: k_1, \ldots, p_n :: k_n$ gibt somit eine probabilistische Verteilung über die einzelnen Klauseln an und diese können anschließend zu einer beliebigen logischen Verknüpfung aus $K = k_1, \ldots, k_n$ verknüpft werden. In Beispiel 1 sieht man, dass der Fakt *erdbeben* mit Wahrscheinlichkeit $P(erdbeben) = 0.1$ wahr ist und 0.9 falsch ist. Andersherum für den Fakt alarm_bei_erdbeben ist die Wahrscheinlichkeit 0.9, dass der Alarm auslöst und eine Gegenwahrscheinlichkeit von 0.1, dass dieser nicht auslöst. Die Aussagen aus Zeile 1 und 2 sind probabilistische Fakten.
In Beispiel 1 gilt somit $P(alarm) = P(erdbeben) * P(alarm_bei_erdbeben) = 0.9 * 0.1 = 0.09$ [12][13].
Noch ein Konstrukt, das von `Problog` geliefert wird, ist die *annotated disjunction*, die es möglich macht Klauseln zu definieren, die mehr als nur zwei Werte annehmen können. Dafür werden alle möglichen Werte die eintreffen können als probabilistischen Fakt dargestellt, jedoch wird diesen ein Tupel $t = (v, c)$ übergeben, wobei v eine festgelegte Variable ist und $c \in \mathbb{N}$ eine Konstante die den zutreffenden Wert repräsentiert, und mit einem logischen $\vee$ verknüpft, das in `Problog` mit einem „;“ dargestellt wird. Ein Beispiel dafür ist in Programmtext 2.2 zu sehen, dass einen Alarm beschreibt, der die Werte $P(A_1) = 0.2$ (an), $P(A_2) = 0.7$ (aus) oder $P(A_3) = 0.1$ (defekt) annehmen kann [13].

Programmtext 2.2: `Problog`-Beispiel *annotated disjunction*

```
1 2/10::alarm(X, 1); 7/10::alarm(X, 2); 1/10::alarm(X, 3).
```

ATKLOG

Angerona verwendet `ATKLOG`, um ein Angreifermodell zu modellieren. `ATKLOG` ist eine Sprache die auf `Problog` basiert und entworfen wurde, um das initiale Vorwissen eines Benutzers über den Datenbankzustand darzustellen. Das Vorwissen des Benutzers wird als Wahrscheinlichkeitsverteilung definiert. Außerdem verändert sich das Vorwissen eines Benutzers in `ATKLOG`, wenn dieser eine Anfrage stellt, da das Ergebnis dem Benutzer mehr Informationen über den aktuellen Datenbankzustand verrät. Beachtet wird auch, dass verweigerte Anfragen das Vorwissen des Benutzers auch beeinflussen [7].

Ein syntaktischer Unterschied von ATKLOG zu Problog ist, dass eine Negation mit „*NOT*“ dargestellt wird. Außerdem werden *annotated disjunctions* anders dargestellt, weil ATKLOG nur Variablenzuweisung oder probabilistische Fakten erlaubt.
Umgewandelt wird eine *annotated disjunction* $v_1 :: a_1, ..., v_n :: a_n$ in ein ATKLOG konformes Schema, indem das „;“ entfernt wird und für jeden Wert den die Variable annehmen kann eine neue Klausel erstellt wird. Anschließend wird jede Klausel mit einer neuen Variable, die im Folgenden $sw_1, ... sw_n$ genannt werden, konjugiert und die dazugehörige Wahrscheinlichkeit mit v_i für $1 \leq i < n$ wird für jeder dieser neu erstellen Variable ein probabilistischer Fakt hinzugefügt. Jedes p_i wird dabei neu berechnet mit $v_i = p_i \cdot (1 - \sum_{1 \leq j < i} p_i)^{-1}$, wobei $p_1, \ldots, p_{n-1}, p_n$ die Wahrscheinlichkeiten für das eintreffen der Werte sind. Außerdem muss für jeden möglichen eintreffenden Wert ein Variablenname zugeordnet werden der als *name* definiert wird und die Tupel, die durch die *annotated disjunction* übergeben werden als $\bar{t}_1, \ldots, \bar{t}_n$. Der Ausdruck $v :: sw(\_)$ ist eine Abkürzung dafür, dass ein probabilistischer Wert für $v :: sw(\bar{t})$ für jedes Tupel $\bar{t}$ existiert. Ein Muster für die Vorgehensweise bei *annotated disjunction* ist in Programmtext 2.3 zu sehen.

Programmtext 2.3: Vorgehensweise bei *annotated disjunction*

```
1 name(t̄_1) :- sw_1(t̄_1)
2 ...
3 name(t̄_n) :- NOT sw_1(t̄_1), ..., NOT sw_{n-1}(t̄_{n-1}), sw_n(t̄_n)
4
5 v_1 :: sw_1(_)
6 ...
7 v_n :: sw_n(_)
```

Angerona

Angerona ist ein DBIC-Mechanismus [19] der Datenbanken gegen probabilistische Inferenzen absichert. Es existiert zur Zeit nur ein Prototyp von Angerona, der nur boolesche Anfragen unterstützt. Es wurde zwar eine Lösung für nicht-boolesche Anfragen im Paper vorgestellt, jedoch wurde diese nicht im Prototypen implementiert. Der Angerona Algorithmus aus 1 erhält folgende Eingaben:

- Den Systemzustand $s = \langle db, U, P \rangle$ der den aktuellen Systemzustand beschreibt, wobei db der aktuelle Datenbankzustand ist, U die Menge der Benutzer und P die Sicherheitsregeln enthält.
- Die Historie H, die aus einer Sequenz $h = \langle \langle u, q \rangle, d, a \rangle$ besteht, die für jeden Benutzer alle bereits getätigten Anfragen speichert. Jeder Eintrag in der Historie speichert den Benutzer u, die Anfrage q, die Entscheidung $d \in \{\top, \bot\}$, ob die Anfrage genehmigt wurde und die Antwort aus dem Datenbanksystem $a \in \{\top, \bot, \dagger\}$, wobei $\dagger$ die Antwort ist, wenn die Anfrage nicht genehmigt wurde.
- Die Aktionen $\langle u, q \rangle$ die alle Anfragen an Angerona stellen.
- Die Systemkonfiguration C, die aus $\langle D, \Gamma \rangle$ besteht, wobei D das Datenbankschema darstellt und Γ die Integritätsbedingungen.

Algorithm 1 Angerona Algorithmus

Input Systemzustand $s = \langle db, U, P \rangle$, Historie h, Aktion $\langle u, q \rangle$, Systemkonfiguration C und ein ATKLOG Modell ATK
Output Die Sicherheitsentscheidung ob die Anfrage q zugelassen wird in $\{\top, \bot\}$
for $\langle u, \psi, l \rangle \in$ secrets(P, u) **do**
    if $secure(C, ATK, h, \langle u, \psi, l \rangle)$ **then**
        if $pox(C, ATK, h, \langle u, q \rangle)$ **then**
            $h' \leftarrow h \cdot \langle \langle u, q \rangle, \top, \top \rangle$
            if $\neg secure(C, ATK, h', \langle u, \psi, l \rangle$ **then**
                return $\bot$
        if $pox(C, ATK, h, \langle u, \neg q \rangle)$ **then**
            $h' \leftarrow h \cdot \langle \langle u, q \rangle, \top, \bot \rangle$
            if $\neg secure(C, ATK, h', \langle u, \psi, l \rangle$ **then**
                return $\bot$
return $\top$

function SECURE($\langle D, \Gamma \rangle$, ATK, h, $\langle u, \psi, l \rangle$)
    $p \leftarrow ATK(u)$
    for $\phi \in knowledge(h, u)$ **do**
        $p \leftarrow p \cup PL(\phi) \cup \{evidence(head(\phi), true)\}$
    $p \leftarrow p \cup PL(\psi)$
    return $[\![p]\!]_D(head(\psi)) < l$

function $pox(\langle D, \Gamma \rangle, ATK, h, \langle u, \psi \rangle)$
    $p \leftarrow ATK(u)$
    for $\phi \in knowledge(h, u)$ **do**
        $p \leftarrow p \cup PL(\phi) \cup \{evidence(head(\phi), true)\}$
    $p \leftarrow p \cup PL(\psi)$
    return $[\![p]\!]_D(head(\psi)) > 0$

– Das *ATKLOG* Modell *ATK*, dass das Angreifermodell für jeden Benutzer u über das aktuelle Datenbankschema darstellt.

Angerona verwendet außerdem noch einige Funktionen innerhalb der `secure` und `pox` Funktionen. Diese sind zum einen die Funktion `knowledge`, die aus der Historie h alle raus extrahiert, die zum Benutzer u gehören.
Die Funktion `evidence` ist aus `Problog` und wird verwendet, um Informationen als Wahr oder Falsch anzunehmen, was dadurch die Wahrscheinlichkeitswerte für andere probabilistische Abhängigkeitsregeln verändert.
Die Funktion `PL`, die eine Anfrage in logische Programmierregeln umwandelt und somit für `Problog` lesbar macht [7].

Der Algorthmus prüft dabei für die Anfrage q, welche der Benutzer u gestellt hat aus der Aktion $\langle u, q \rangle$, ob diese eine Sicherheitsregel aus $secrets(P, u)$ verletzt und gibt $\top$ zu-

rück wenn die Anfrage genehmigt werden soll oder $\bot$ wenn diese verweigert werden soll. Dafür wird über alle $secrets(P,u) = \{\langle u,\phi,l\rangle \mid \langle u,\phi,l\rangle \in P \wedge u \in U\}$ iteriert, wobei ϕ die Anfrage und l den Schwellwert definiert, bei der die Anfrage verweigert werden soll. In jeder Iteration für jede Sicherheitsregel wird zuerst geprüft, ob die Sicherheitsregel bereits verletzt wurde, indem zuerst die Funktion `secure` ausgeführt wird.
Die Funktion `secure` erweitert dafür das aktuelle Angreifermodell ATK, indem alle bereits getätigten Anfragen aus der Historie h von dem Benutzer u hinzufügt werden und als `evidence` markiert werden. Anschließend wird mit der Funktion `PL` die Anfrage aus der Sicherheitsregel noch in `Problog` definiert und dem Angreifermodell hinzugefügt. Der Ausdruck $[\![p]\!]_D(head(\psi)$ prüft dabei für alle Datenbankzustände in welcher die Anfrage ψ erfüllt ist und berechnet die Wahrscheinlichkeit durch (Menge der Datenbankzustände in den ψ erfüllt ist)/(Menge aller Datenbankzustände). Wenn diese Wahrscheinlichkeit kleiner dem Schwellwert l ist, dann gilt die Sicherheitsregel als nicht verletzt und es wird ein $\top$ ausgegeben.
Anschließend wird sichergestellt, dass die Sicherheitsentscheidung selbst keine Informationen verrät, indem das Vorwissen vom Benutzer u unter dem Schwellwert l liegen muss für die Fälle, dass q genehmigt wurde und das q verweigert wurde. Dafür wird mit der Funktion *pox* geprüft , ob ein Datenbankzustand für das Datenbankmodell existiert in dem q erfüllt ist. Dabei geht die Funktion *pox* die selben Schritte wie die Funktion *secure* durch, vergleicht am Ende jedoch die Wahrscheinlichkeit der erfüllenden Datenbankzustände nicht mit dem Schwellwert sondern prüft nur ob diese größer 0 ist. Wenn *pox* ein $\top$ zurückgibt und somit ein Datenbankzustand existiert indem q erfüllt ist, wird eine neue Historie h' erstellt, die um die Anfrage q erweitert ist. Anschließend wird die `secure` Funktion mit der neuen Historie h' ausgeführt und geprüft, ob die Sicherheitsregel verletzt wird, wenn die Anfrage q genehmigt wurde. Damit wird verhindert, dass eine genehmigte Anfrage eine andere Sicherheitsregel verletzen würde. Wenn die erweiterte Historie h' die Sicherheitsregel verletzt, dann verweigert Angerona die Anfrage q gibt somit ein $\bot$ als Output.
Danach werden die selben Schritte für die Anfrage $\neg q$ durchgeführt, die beschreibt, dass die Anfrage nicht genehmigt wurde. Dafür wird wieder mit der Funktion *pox* geprüft, ob ein Datenbankzustand existiert, indem $\neg q$ gilt. Wenn dies der Fall ist, dann wird die Historie h' um die Anfrage $\neg q$ erweitert und es wird geprüft, ob durch die Verweigerung der Anfrage q die Sicherheitsregel verletzt wird. Wenn dies nicht der Fall ist, wird von Angerona ein $\top$ ausgegeben, ansonsten ein $\bot$.

2.2 Praktisches Setup

Der Prototyp von Angerona kann auf der Webseite von Marco Guarnieri [16] heruntergeladen werden. Angerona lässt sich in zwei verschiedenen Modis starten :

1. **Experiment**: Hiermit können vorgefertigte Beispiele aus dem Paper „Securing Databases from Probabilistic Inference“ [7] reproduziert werden. Dadurch werden keine größeren Initialisierungsschritte benötigt und das Programm übernimmt die Gene-

rierung vom Angreifermodell und die dazugehörige Datenbank. Anschließend werden zufällig generierte Anfragen an die Datenbank gestellt und von Angerona geprüft. Als Ergebnis vom Programm erhält man die Zeitmessung für die Ausführung der Anfrage, die Ausführungszeit von Angerona und die gesamte Zeit in einer CSV Datei.

2. **Manual**: Erlaubt es mit Angerona eine Datenbank abzusichern und Anfragen über Angerona an diese zu stellen. In diesem Modus erwartet Angerona jeweils den Pfad zu den Dateien `beliefProgram.pbl`, `initStatements.txt` und `template.cpt` als Argument bei Programmstart.

Im folgenden wird ausschließlich der Manual Modus verwendet. Das Problem jedoch ist, dass es noch keine Dokumentation zu Angerona gibt und nur eine mitgelieferte `README` in der beschrieben wird wie sich Angerona starten lässt. Jedoch wird nicht beschrieben wie die drei mitgelieferten Dateien definiert werden können. Deshalb wurde anhand von Beispielen im Code und Inhalten aus dem Paper hergeleitet, wie die Dateien für den Input erstellt werden können und wie Anfragen über Angerona gestellt werden können.

Angreifermodell

Das Angreifermodell wird in der Datei `beliefProgram.pbl` beschrieben. Hier wird das Vorwissen des Angreifers mithilfe von `ATKLOG` beschrieben, dass auf `Problog` basiert. Dabei übernimmt das Angreifermodell die Aufgabe, das Vorwissen jedes Benutzers für jeden Patienten in der Datenbank zu modellieren.
Hierfür wird zuerst ein Bayes-Netz für die abzusichernden sensiblen Daten und deren Abhängigkeiten mit den probabilistischen Wahrscheinlichkeiten modelliert. Anschließend wird das Bayes-Netz in zwei Schritten in das Angreifermodell überführt. Im ersten Schritt wird das Bayes-Netz in ein `Problog`-Programm überführt. Das `Problog`-Programm enthält noch Syntaktischen Zucker, wie zum Beispiel die *annotated disjunctions*, der von Angerona nicht unterstützt wird. Entsprechend wird wie in Abschnitt 2.1 beschrieben das `Problog`-Programm in ein `ATKLOG`-konformes Schema überführt. Diese Schritte müssen Manuell durchgeführt werden.
Dann werden alle Knoten $\{v_0, \ldots, v_n\} \in V$ aus dem Bayes-Netz in das Angreifermodell übertragen, die **keine Abhängigkeit** besitzen. Diese sind definiert als $\{np_i \in V \mid Parent(np_i) = \varnothing\}$, wobei das Attribut *name* dem vorher definierten Namen des Knotens entspricht. Dafür werden zuerst die Knoten einer Variable zugeordnet und anschließend die Wahrscheinlichkeiten für die Variable als probabilistischen Fakt für jeden Patienten $p \in \mathbb{N}$ in das Angreifermodell hinzugefügt, wie im Programmtext 2.4 zu sehen ist.

Programmtext 2.4: `beliefProgram.pbl` für Knoten ohne Abhängigkeiten

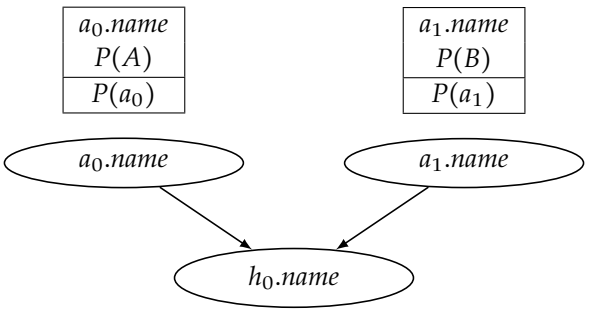
$np_0.name(X) : -p\_np_0.name(X).$
$np_1.name(X) : -p\_np_1.name(X).$
...
$np_n.name(X) : -p\_np_n.name(X).$

$P(np_0) :: p\_np_0.name(p_0).$

$P(np_1) :: p\_np_1.name(p_0).$
...
$P(np_n) :: p\_np_n.name(p_0).$
$P(np_0) :: p\_np_n.name(p_1).$
...
$P(np_0) :: p\_np_0.name(p_n).$
$P(np_1) :: p\_np_1.name(p_n).$
...
$P(np_n) :: p\_np_n.name(p_n).$

Für die Knoten **mit Abhängigkeiten**, also für alle Knoten V für die gilt $H = \{v \in V \mid |Parent(v)| \geq 1\}$ werden die Klauseln für jede mögliche Welt von den Abhängigkeiten $a_0, ...a_n$ eingefügt und verknüpft mit einer Konjunktion, die in `ATKLOG` mit einem "," dargestellt wird. Anschließend wird jeder Ausdruck mit einer selbst definierten Variable versehen und mit diesem konjugiert, die am Ende durch einen probabilistischen Fakt die dazugehörige Wahrscheinlichkeit für diese Welt zugeordnet wird. Ein Beispiel für ein Bayes-Netz mit zwei Abhängigkeiten ist in Abbildung 2.3 und die dazugehörige `beliefProgram.pbl` in Programmtext 2.5 zu sehen.

Abbildung 2.3: Bayes-Netz mit 2 Abhängigkeiten



$h_0.name$		
A	B	$P(C\|A,B)$
T	T	$P(h_0)$
T	F	$P(h_1)$
F	T	$P(h_2)$
F	F	$P(h_3)$

Programmtext 2.5: `beliefProgram.pbl` für Knoten mit Abhängigkeiten

$H(X) :- a_0.name(X)\ ,\ a_1.name(X), variable11(X).$
$H(X) :- a_0.name(X)\ ,\ NOTa_1.name(X), variable10(X).$
$H(X) :- NOT\ a_0.name(X)\ ,\ a_1.name(X), variable01(X).$
$H(X) :- NOT\ a_0.name(X)\ ,\ NOTa_1.name(X), variable00(X).$

$P(h_0) :: variable11(p_0)$
$P(h_1) :: variable10(p_0)$

$P(h_2) :: variable01(p_0)$
$P(h_n) :: variable00(p_0)$
...
$P(h_0) :: variable11(p_n)$
$P(h_1) :: variable10(p_n)$
$P(h_2) :: variable01(p_n)$
$P(h_n) :: variable00(p_n)$

Eine **mehrwertige Variable** kann anstatt zwei Wahrheitswerten `true` und `false` auch mehr Werte annehmen. Um diese im Angreifermodell auszudrücken, wird der mehrwertigen Variable in eine *annotated disjunction* eingeteilt. Zum Beispiel das Alter einer Person kann mehrere Werte annehmen. Im folgenden Beispiel wird das Alter einer Person in jung, mittel(alt) und alt eingeteilt. Dabei sind die Wahrscheinlichkeiten dafür, dass eine Person jung ist 40%, mittel 25% und alt 35%. Dann würde im `beliefProgram.pbl` die Klauseln wie in Programmtext 2.6 zu sehen ist angeordnet werden und die Wahrscheinlichkeiten p_i jeweils berechnet werden durch :

- $0.4 \cdot (1 - 0)^{-1} = 0.4$
- $0.25 \cdot (1 - 0.4)^{-1} = 0.4166 = 5/12$
- $0.35 \cdot (1 - 0.35 - 0.4)^{-1} = 1$

Programmtext 2.6: Beispiel für mehrwertige Variablen

$alter(X,1) :- p\_jung(X).$
$alter(X,2) :- NOT\ p\_jung(X), p\_mittel(X).$
$alter(X,3) :- NOT\ p\_jung(X)\ , NOT\ p\_mittel(X)\ , p\_alt(X).$

$4/10 :: p\_jung(id).$
$5/12 :: p\_mittel(id).$
$1/1 :: p\_alt(id).$

Initialisierung und Sicherheitsregeln

Die `initStatements.txt` definiert das Datenbankschema und füllt die Datenbank mit Daten. Außerdem werden hier die Sicherheitsregeln definiert, die den Schwellwert festlegen, bei dem ein Benutzer auf die Daten der Datenbank zugreifen darf..
Die `initStatements.txt` wird in vier Schritte initialisiert:

1. Die **Tabellen** werden mit dem Kommando *AS admin : CREATE TABLE tablename(attribute)* initialisiert.
2. Die **Benutzer** werden mit dem Kommando *AS admin : ADD USER benutzer* initialisiert.
3. Die **Sicherheitsregeln** werden mit dem Kommando *AS admin : SECRET anfrage('id') FOR benutzer THRESHOLD schwellwert* initalisiert.
4. Das **Füllen** der Datenbank wird mit dem Kommando *AS admin : INSERT IN tabelle ['id']* erledigt.

Template

Die `template.cpt` ist die Vorlage für Angerona. In dieser werden die Tabellen aus den `initStatements.txt` und die definierten Fakten aus dem `beliefProgram.pbl` initialisiert.

2.3 Beispiel

In diesem Beispiel wird eine medizinische Datenbank eines Krankenhauses betrachtet, die das Rauchverhalten und die Elternbeziehung von Patienten speichert und ob diese Krebs haben. Die Datenbank hat dabei die Tabellen *Patienten, Raucher, Krebs, MutterHatKrebs* und *VaterhatKrebs*. In der medizinischen Datenbank sind die Patienten Alice mit der *id* 1, Bob mit der *id* 2 und Carl mit der *id* 3 die Patienten, wobei Alice und Bob die Eltern von Carl sind. Alice raucht nicht, aber Bob und Carl rauchen. Alice und Bob haben Krebs und Carl hat kein Krebs.
Der Benutzer für das System ist Mallory mit dem Benutzernamen *mallory*. Außerdem wird vom folgendem probabilistischen Modell ausgegangen:

1. Jeder Patient entwickelt mit einer Wahrscheinlichkeit von 5% Krebs
2. Für jeden Elternteil der Krebs hat, steigt die Wahrscheinlichkeit für das Kind Krebs zu bekommen um 15%
3. Wenn ein Patient raucht, dann steigt die Wahrscheinlichkeit, dass dieser Krebs bekommt, um 25%

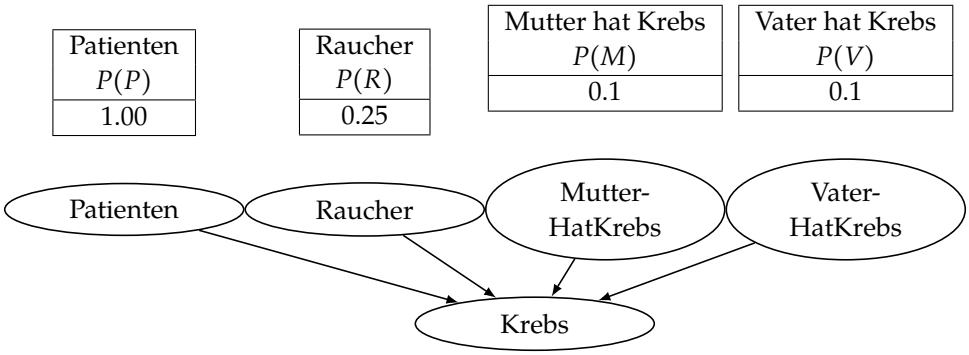
Zu Beginn wird das probabilistische Modell in ein Bayes-Netz übertragen. Dabei wird davon ausgegangen, dass die Wahrscheinlichkeit dafür, dass ein Patient raucht 25% und das jeweils ein Elternteil Krebs hat 10% entspricht [5]. Ein Patient ist in diesem Fall zu 100% ein Patient, da eine medizinische Datenbank betrachtet wird [7]. Dadurch ergibt sich das Bayes-Netz aus Abbildung 2.4
Die Fälle mit *P=false* sind bewusst nicht in der Tabelle für Krebs gelistet, da diese Fälle nicht eintreffen können. Die Wahrscheinlichkeiten dafür, dass ein Patient Krebs hat wird berechnet, indem die oben genannten Wahrscheinlichkeiten aufsummiert werden, wenn dieser Fakt auf den Patienten zutrifft. Zum Beispiel der Wert in der ersten Tabellenzeile ergibt sich aus 0.05 + 0.15 + 0.15 + 0.25 = 0.6.

Mithilfe des Bayes-Netzes wird das Angreifermodell definiert, indem zuerst jeder Knoten der keine Abhängigkeiten besitzt definiert wird, wie in Kapitel Abschnitt 2.2 beschrieben. Im Beispiel sind das die Knoten Patient, Raucher, MutterHatKrebs und VaterHatKrebs. Dadurch ergibt sich für die `beliefProgram.pbl` der Code aus Programmtext 2.7

Programmtext 2.7: Beispiel für Knoten ohne Abhängigkeiten

```
1 patient(X) :- p_patient(X).
2 raucher(X) :- p_raucher(X).
3 mutterhatkrebs(X) :- p_mutterhatkrebs(X).
4 vaterhatkrebs(X) :- p_vaterhatkrebs(X).
```

Abbildung 2.4: Bayes-Netz



Krebs				
P	R	M	V	$P(K\|P,R,M,V)$
T	T	T	T	0.6
T	T	T	F	0.45
T	T	F	T	0.45
T	T	F	F	0.3
T	F	T	T	0.35
T	F	T	F	0.2
T	F	F	T	0.2
T	F	F	F	0.05

```
5
6  1/1 :: p_patient(1).
7  25/100 :: p_raucher(1).
8  1/10 :: p_mutterhatkrebs(1).
9  1/10 :: p_vaterhatkrebs(1).
10 1/1 :: p_patient(2).
11 25/100 :: p_raucher(2).
12 1/10 :: p_mutterhatkrebs(2).
13 1/10 :: p_vaterhatkrebs(2).
14 1/1 :: p_patient(3).
15 25/100 :: p_raucher(3).
16 1/10 :: p_mutterhatkrebs(3).
17 1/10 :: p_vaterhatkrebs(3).
```

Anschließend können die Knoten mit Abhängigkeiten definiert werden. Im Beispiel ist das nur der Knoten Krebs. Für den Knoten Krebs wird dafür für jede mögliche Welt in Abhängigkeiten von den Elternknoten eine Klausel initialisiert und mit einer selbst definiert Variable versehen, die im Folgenden die Form $v0000, v0001, \ldots, v1111$ hat. Damit wird das `beliefProgral.pbl` erweitert um den Code aus Programmtext 2.8

Programmtext 2.8: Beispiel für Knoten mit Abhängigkeiten

```
1 krebs(X) :- patient(X), raucher(X), mutterhatkrebs(X), vaterhatkrebs(X), v1111(X).
```

2 $krebs(X) :- patient(X), raucher(X), mutterhatkrebs(X), NOT\ vaterhatkrebs(X), v1110(X).$
3 $krebs(X) :- patient(X), raucher(X), NOT\ mutterhatkrebs(X), vaterhatkrebs(X), v1101(X).$
4 $krebs(X) :- patient(X), raucher(X), NOT\ mutterhatkrebs(X), NOT\ vaterhatkrebs(X), v1100(X).$
5 $krebs(X) :- patient(X), NOT\ raucher(X), mutterhatkrebs(X), vaterhatkrebs(X), v1011(X).$
6 $krebs(X) :- patient(X), NOT\ raucher(X), mutterhatkrebs(X), NOT\ vaterhatkrebs(X), v1010(X).$
7 $krebs(X) :- patient(X), NOT\ raucher(X), NOT\ mutterhatkrebs(X), vaterhatkrebs(X), v1001(X).$
8 $krebs(X) :- patient(X), NOT\ raucher(X), NOT\ mutterhatkrebs(X), NOT\ vaterhatkrebs(X), v1000(X).$
9
10 $6/10 :: v1111(1).$
11 $45/100 :: v1110(1).$
12 $45/100 :: v1101(1).$
13 $3/10 :: v1100(1).$
14 $35/100 :: v1011(1).$
15 $2/10 :: v1010(1).$
16 $2/10 :: v1001(1).$
17 $5/100 :: v1000(1).$
18 $6/10 :: v1111(2).$
19 $45/100 :: v1110(2).$
20 $45/100 :: v1101(2).$
21 $3/10 :: v1100(2).$
22 $35/100 :: v1011(2).$
23 $2/10 :: v1010(2).$
24 $2/10 :: v1001(2).$
25 $5/100 :: v1000(2).$
26 $6/10 :: v1111(3).$
27 $45/100 :: v1110(3).$
28 $45/100 :: v1101(3).$
29 $3/10 :: v1100(3).$
30 $35/100 :: v1011(3).$
31 $2/10 :: v1010(3).$
32 $2/10 :: v1001(3).$
33 $5/100 :: v1000(3).$

Damit wäre das Angreifermodell mit dem *beliefProgram.pbl* vollständig initialisiert.

Die `initStatements.txt` werden nach den vorher angegeben vier Schritten initialisiert.

1. Für jeden Knoten aus dem Bayes-Netz wird eine **Tabelle** angelegt:

   $AS\ admin : CREATE\ TABLE\ patient(id)$
   $AS\ admin : CREATE\ TABLE\ raucher(id)$
   $AS\ admin : CREATE\ TABLE\ mutterhatkrebs(id)$

AS admin : *CREATE TABLE vaterhatkrebs*(*id*)
AS admin : *CREATE TABLE krebs*(*id*)

2. Der **Benutzer** Mallory wird folgendermaßen angelegt:

AS admin : *ADD USER mallory*

3. Die **Sicherheitsregeln** können beliebig gewählt werden. Im folgenden darf Mallory auf die Daten nur zugreifen, wenn sie mit einer Wahrscheinlichkeit von unter 50% weiß, dass ein beliebiger Patient Krebs hat.

AS admin : *SECRET krebs*('1') *FOR mallory THRESHOLD* 1/2
AS admin : *SECRET krebs*('2') *FOR mallory THRESHOLD* 1/2
AS admin : *SECRET krebs*('3') *FOR mallory THRESHOLD* 1/2

4. **Gefüllt** wird die Datenbank mit den oben gegeben Werten dann folgendermaßen:

AS admin : *INSERT IN patient*['1']
AS admin : *INSERT IN patient*['2']
AS admin : *INSERT IN patient*['3']
AS admin : *INSERT IN raucher*['2']
AS admin : *INSERT IN raucher*['3']
AS admin : *INSERT IN mutterhatkrebs*['1']
AS admin : *INSERT IN vaterhatkrebs*['1']
AS admin : *INSERT IN krebs*['1']
AS admin : *INSERT IN krebs*['2']

In der `tempalte.cpt` werden alle verwendeten Variablen aus der `beliefProgram.pbl` und `initStatements.txt` wie in Programmtext 2.9 initalisiert.

Programmtext 2.9: `tempalte.cpt` vom Beispiel

```
1  patient: []
2  raucher: []
3  mutterhatkrebs: []
4  vaterhatkrebs: []
5  p_patient: []
6  p_raucher: []
7  p_mutterhatkrebs: []
8  p_vaterhatkrebs: []
9  krebs: []
10 v1111: []
11 v1110: []
12 v1101: []
13 v1100: []
14 v1011: []
15 v1010: []
16 v1001: []
```

```
17 v1000: [ ]
```

Anschließend wird Angerona mit der `prototype.jar` ausgeführt und mit dem Befehl `java -Xmx8192m -jar prototype.jar angeronaManual 1 models/BAExample/beliefProgram.pbl models/BAExample/initStatements.txt models/BAExample/template.cpt` gestartet. Anschließend ist es möglich mit Angerona Anfragen zu stellen in der Form : `AS [user] : [tabelle]('[id]')`. Ein Beispiel für eine Anfrage von Mallory, ob Alice Krebs hat, ist in Programmtext 2.10 zu sehen.

Programmtext 2.10: Angerona: Anfrage von Mallory ob Alice Krebs hat

```
1 Write a command. Write 'Stop' to conclude.
2 AS mallory:krebs('1')
3 ACCESS CONTROL RESULT
4 Acc.Ctrl.: false Cause: mallory : krebs ('1') is not authorized
```

3

Medizinische Datenbanken und Angreifermodellierung

In diesem Kapitel wird der Prototyp von Angerona auf drei verschiedene medizinische Datenbanken angewendet. Betrachtet werden die Datenbanken MIMIC-III, eICU und der synthetische Patientengenerator Synthea.

3.1 MIMIC-III

MIMIC-III (Medical Information Mart for Intensive Care) III ist eine für Forschungszwecke frei zugängliche Datenbank, die pseudonymisierte medizinische Daten und dazugehörige klinische Daten, die in Intensivstationen in einem Spezialkrankenhaus eingewiesen wurden, speichert. Die Daten stammen dabei aus dem *Beth Israel Deaconess Medical Center* in Boston, Massachusetts und wurden in dem Zeitraum vom Juni 2001 bis Oktober 2012 erfasst. Die Datenbank enthält 58976 Krankenhauseinweisungen für 38645 Erwachsene und 7875 Neugeborene. Gespeichert werden Daten wie Vitalparameter, Medikamente, Labormessungen, Beobachtungen, Notizen vom Personal, Flüssigkeitsbilanzen, Verfahrenscodes, Diagnosecodes, Aufenthaltsdauer, Überlebensdaten und mehr [14].

Im folgenden wird für jeden Patienten in der MIMIC-III-Datenbank die Information, dass dieser Krebs hat, abgesichert. Die Wahrscheinlichkeitswerte des Vorwissens vom Angreifermodell wurden aus der MIMIC-III-Datenbank ausgelesen, indem zum Beispiel die Wahrscheinlichkeit für das Vorwissen, dass eine Person Krebs hat berechnet wird durch $P(krebs) = \frac{\text{Anzahl der Patienten die Krebs haben}}{\text{Gesamtanzahl der Patienten}}$.
Risikofaktoren und somit Abhängigkeiten für Krebs sind in dem Fall Alter, Geschlecht und Rauchverhalten des Patienten [1, 2]. Um diese Daten aus der MIMIC-III-Datenbank zu erhalten werden die folgenden Tabellen benötigt :

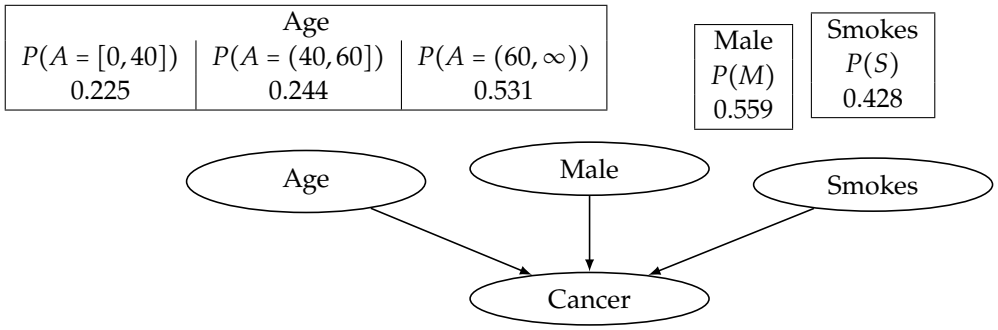
1. Die Tabelle *ADMISSIONS* enthält Informationen über die Einweisung ins Krankenhaus. Dabei ist jeder Krankenhausaufenthalt einer eindeutigen *HADM_ID* zugeordnet. Die Tabelle enthält Informationen zu demographischen Daten, Ein- und Ausweisungszeiten und erste Einweisungsinformationen.

2. Die Tabelle *PATIENTS* enthält Informationen über jeden Patienten. Dabei ist jeder Patient einer eindeutigen *SUBJECT_ID* zugeordnet und kann mehrere *HADM_ID* zugeordnet werden, weil ein Patient mehrmals ins Krankenhaus eingewiesen werden kann. Die Tabelle enthält Informationen über das Geschlecht, Geburtsdatum und falls vorhanden Todesdatum, .
3. Die Tabelle *DIAGNOSES_ICD* enthält pro *HADM_ID* und *SUBJECT_ID* die dazugehörige Diagnose als `icd9-code`(International Classification of Diseases). `Icd9-codes` sind standardisierte Codes, die verwendet werden um Krankheiten , Verletzungen oder sonstige Diagnosen International einheitlich zu speichern [3].
4. Die Tabelle *NOTEEVENTS* enthält alle Notizen zu den Patienten, die vom Krankenhauspersonal dokumentiert sind.

Im folgenden wird die Angreifermodellierung für die MIMIC-III-Datenbank anhand der Krankenhauseinweisungen betrachtet. Das bedeutet, dass die *HADM_ID* als *id* genutzt wird, um die Patienten nach Angerona zu übertragen. Daher kann es auch vorkommen, dass ein Patient mehrmals auftauchen kann, wenn dieser mehrmals eingewiesen wurde. Dabei wurde das Alter in die Intervalle $[0,40]$, $(40,60]$ und $(60,\infty)$ eingeteilt, da dies in [1] als Altersgrenze definiert wurde, in denen die Risiken für Krebs unterschiedlich sind und mit höherem Alter ein erhöhtes Krebsrisiko existiert. Krankenhauseinweisungen für Patienten mit einem Alter im Intervall $[0,40]$ existieren 13265, für $(40,60]$ 14382 und $(60,\infty)$ 31329. Dies lässt sich berechnen, indem die Differenz zwischen dem Geburtsdatum aus der Tabelle *PATIENTS* und dem Einweisungsdatum aus der Tabelle *ADMISSIONS* berechnet wird. Das Geschlecht jedes Patienten kann aus der Tabelle *PATIENTS* unter dem Attribut *gender* ausgelesen werden und es gibt 32950 Männer und 26026 Frauen, wobei Männer eine höhere Wahrscheinlichkeit haben Krebs zu bekommen als Frauen. Das Rauchverhalten wurde aus der Tabelle *NOTEEVENTS* ausgelesen, indem nach den Schlagwörtern „smoke“ und „cigarette“ gesucht wurde. Somit erhält man 25237 Raucher, wobei Raucher ein erhöhtes Risiko haben Krebs zu bekommen.
Um die Krankenhausaufenthalte zu erhalten, für die Patienten bei den Krebs diagnostiziert wurde, werden die `icd9-codes` für Krebs [9] mit den aus der *DIAGNOSES_ICD* verglichen und bei Gleichheit hinzugefügt. Dafür wurde ein Skript geschrieben, dass alle `icd9-codes` aus einer Liste, in der `icd9-codes` für Krebs notiert sind, extrahiert und in an das `icd9-code` Format von der MIMIC-III-Datenbank umwandelt und anschließend diese in eine extra angefertigte Tabelle kopiert. Somit kann man diese Tabellen direkt miteinander vergleichen und erhält 13658 Krebspatienten. Diese Informationen werden mit einem Bayes-Netz grafisch dargestellt, dass in Abbildung 3.1 zu sehen ist.
Das Bayes-Netz wird in ein Angreifermodell nach dem Schema aus 2.2 übertragen und als `beliefProgram.pbl` gespeichert. Für das Alter wird dabei die *annotated disjunction* verwendet, da diese durch die drei Intervalle mehr als zwei Werte annehmen können.

Die *initStatements.txt* wird nach dem Schema von 2.2 initialisiert, indem die Tabellen jeweils ein Knoten aus dem Bayes-Netz darstellen. Die Benutzer und die Sicherheitsregeln für Krebs können hierbei beliebig erstellt werden. Für das Füllen der Tabellen werden die *HADM_ID*s verwendet. Dafür wurde ein Bash-Skript geschrieben, dass alle *HADM_ID*'s aus der Datenbank filtert auf die das Attribut in der Tabelle zutrifft und anschließend in

Abbildung 3.1: MIMIC-III Bayes-netz



Cancer			
Age	Smokes	Male	$P(C \mid A,M,S)$
<= 40	T	T	0.0247
<= 40	T	F	0,027
<= 40	F	T	0.022
<= 40	F	F	0.025
40 – 60	T	T	0.182
40 – 60	T	F	0.242
40 – 60	F	T	0.15
40 – 60	F	F	0.2
> 60	T	T	0.322
> 60	T	F	0.3
> 60	F	T	0.271
> 60	F	F	0.237

der Form „`AS admin : INSERT IN tabelle ['HADM_ID']`“ die Daten speichert. Zum Beispiel werden die Männer modelliert, indem alle *HADM_ID*'s angefragt werden die Männlich sind und für jede Männliche Person ein Eintrag in die `initStatements.txt` in der Form „`AS admin : INSERT IN male ['HADM_ID']`“ eingefügt.
Anschließend können beliebig viele Sicherheitsregeln hinzugefügt werden für die einzelnen `HADM_IDs`, indem in der `initStatements.txt` die Sicherheitsregeln erweitert werden durch den Befehl „`AS admin : SECRET cancer('HADM_ID') FOR u1 THRESHOLD 0/1`“. Anschließend muss im Angreifermodell, also in der Datei `beliefProgram.pbl` für diese HADM_ID das Vorwissen definiert werden. Ein Beispiel der `beliefProgram.pbl` in der ein Patient mit der HADM_ID 165315 abgesichert wurde ist in Programmtext 3.1 zu sehen.

Programmtext 3.1: *beliefProgram.pbl* für Krebs in MIMIC-III

```
1 male(X) :- p_male(X).
2 smokes(X) :- p_smokes(X).
3
4 age(X,100000) :- p_young(X).
5 age(X,100001) :- NOT p_young(X), p_middle(X).
```

```
6  age(X,100002) :- NOT p_young(X) , NOT p_middle(X) , p_old(X).
7
8  cancer(X) :- age(X,100000), smokes(X), male(X), p_young_smokes_male(X).
9  cancer(X) :- age(X,100000), smokes(X), NOT male(X), p_young_smokes_not_male(X).
10 cancer(X) :- age(X,100000), NOT smokes(X), male(X), p_young_not_smokes_male(X).
11 cancer(X) :- age(X,100000), NOT smokes(X), NOT male(X),
       p_young_not_smokes_not_male(X).
12
13 cancer(X) :- age(X,100001), smokes(X), male(X), p_middle_smokes_male(X).
14 cancer(X) :- age(X,100001), smokes(X), NOT male(X), p_middle_smokes_not_male(X).
15 cancer(X) :- age(X,100001), NOT smokes(X), male(X), p_middle_not_smokes_male(X).
16 cancer(X) :- age(X,100001), NOT smokes(X), NOT male(X),
       p_middle_not_smokes_not_male(X).
17
18 cancer(X) :- age(X,100002), smokes(X), male(X), p_old_smokes_male(X).
19 cancer(X) :- age(X,100002), smokes(X), NOT male(X), p_old_smokes_not_male(X).
20 cancer(X) :- age(X,100002), NOT smokes(X), male(X), p_old_not_smokes_male(X).
21 cancer(X) :- age(X,100002), NOT smokes(X), NOT male(X),
       p_old_not_smokes_not_male(X).
22
23 559/1000 :: p_male(165315).
24 696/1000 :: p_smokes(165315).
25 225/1000 :: p_young(165315).
26 244/775 :: p_middle(165315).
27 1/1 :: p_old(165315).
28 44/1000 :: p_young_smokes_male(165315).
29 48/1000 :: p_young_smokes_not_male(165315).
30 19/1000 :: p_young_not_smokes_male(165315).
31 23/1000 :: p_young_not_smokes_not_male(165315).
32 201/1000 :: p_middle_smokes_male(165315).
33 262/1000 :: p_middle_smokes_not_male(165315).
34 178/1000 :: p_middle_not_smokes_male(165315).
35 233/1000 :: p_middle_not_smokes_not_male(165315).
36 365/1000 :: p_old_smokes_male(165315).
37 327/1000 :: p_old_smokes_not_male(165315).
38 324/1000 :: p_old_not_smokes_male(165315).
39 279/1000 :: p_old_not_smokes_not_male(165315).
```

3.2 eICU

eICU ist eine Datenbank, die aus einer großen Anzahl von Daten aus verschiedenen Krankenhäusern der USA besteht. Dabei ist MIMIC-III nicht Teil der eICU, wodurch dies einen unabhängigen Datensatz darstellt. Alle Tabellen wurden so pseudonymisiert, dass diese dem HIPAA Standard entsprechen. Dadurch kommt es auch, dass Patienten mit einem

Alter über 89 in der Tabelle als „>89“ dargestellt werden. Die Daten stammen aus dem Jahr 2014 bis 2015 und wurden dabei zufällig aus verschiedenen Krankenhäusern der USA gewählt und anschließend wurde jedem Krankenhausaufenthalt und Patient eine eindeutige Identifikationsnummer zugeordnet.
Die Datenbank enthält 200859 Krankenhauseinweisungen für 139367 Patienten aus 208 verschiedenen Krankenhäusern. Zu den Daten gehören Vitalparameter, Messungen, Pflegepläne, Art und Schweregrad der Krankheit, Diagnoseinformationen, Behandlungsinformationen und mehr.

Im folgenden wird für jeden Patienten in der eICU Datenbank die Information, dass dieser Krebs hat, abgesichert. Die Wahrscheinlichkeitswerte des Vorwissens vom Angreifermodell werden nach demselben Prinzip wie bei MIMIC-III aus der eICU Datenbank ausgelesen.
Risikofaktoren und somit Abhängigkeiten für Krebs sind in dem Fall Alter, Geschlecht und eine Chemotherapie oder andere Onkologische Maßnahmen [1, 2]. Um diese Daten aus der eICU Datenbank zu erhalten werden die folgenden Tabellen benötigt :

1. Die Tabelle *ADMISSIONDX* stellt für jede Station in der eine Diagnose gestellt wurde, die Diagnoseinformationen bereit. Die Tabelle enthält Informationen zu demographischen Daten, Ein- und Ausweisungszeiten und erste Einweisungsinformationen.
2. Die Tabelle *PATIENT* enthält genauere Informationen über einen Patienten. Dabei ist jeder Patient einer eindeutigen *patientUnitStayID* zugeordnet, wodurch Patienten mehrfach auftauchen können, wenn diese mehrfach eingewiesen wurden. Dies ist immer die Ausgangstabelle, da alle Krankenhausaufenthalte betrachtet werden. Zu den Daten gehören Informationen über das Geschlecht, Alter, ethnische Zugehörigkeit und mehr.
3. Die Tabelle *DIAGNOSIS* enthält Diagnosedaten zu jedem Krankenhausaufenthalt. Diese werden als `icd9-codes` gespeichert.
4. Die Tabelle *TREATMENT* enthält Behandlungsinformationen über den Krankenhausaufenthalt.

Die Intervalle für das Alter wurden wie bei MIMIC-III gewählt und das Alter kann aus der Tabelle *PATIENT* aus dem Attribut *age* ausgelesen werden. Somit erhält man für die Intervalle $(0, 40] = 23091$, $(40, 60] = 57065$ und $(60, 89] = 120608$ Patienten. Dabei wurden 95 Patienten nicht berücksichtigt, weil das Attribut *age* leer war.
Nach demselben Prinzip wie beim Alter kann auch das Geschlecht ausgelesen werden, indem das Attribut *gender* aus der Tabelle *PATIENT* ausgelesen wird. Somit erhält man eine Verteilung von 92303 Frauen und 108379 Männern. Hierbei sind wieder 177 nicht berücksichtigt, weil das Attribut *gender* für diese leer war. Da die Anzahl der nicht berücksichtigten Patienten nur 272 beträgt, haben diese keine große Auswirkung auf die berechneten Wahrscheinlichkeiten für das Angreifermodell.
Ob ein Patient eine Chemotherapie oder andere Onkologische Maßnahmen hatte, lässt sich aus der Tabelle *TREATMENT* auslesen, indem man nach „*oncology*“ und „*chemotherapy*“sucht. Dabei wurde festgestellt, dass dies eine garantierte Aussage darüber ist, ob dieser Patient Krebs hat, weil jeder Patient Krebs hatte auf den dies zutrifft. In der eICU Datenbank gibt es 2387 Patienten, bei denen Krebs diagnostiziert wurde und diese

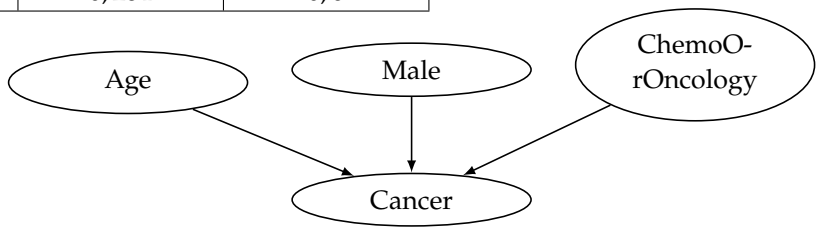
anschließend mit einer Chemotherapie oder anderen onkologischen Maßnahmen behandelt wurden.
Um die Krebspatienten zu erhalten, werden aus der *ADMISSIONDX* Tabelle die *icd9-codes* für Krebs gefiltert. Das führt dazu, dass 8179 Krebspatienten in der Datenbank enthalten sind. Daraus ergibt sich das Bayes-Netz aus Abbildung 3.2.
Die `initStatements.txt` und `beliefProgram.pbl` werden nach demselben Prinzip wie für MIMIC-III generiert. Ein Beispiel für einen Patienten mit der id 875123 ist Programmtext 3.2.

Abbildung 3.2: eICU III Bayes-Netz

Age		
$P(A = [0,40])$	$P(A = (40,60])$	$P(A = (60,89])$
0,116	0,284	0,6

Male
$P(M)$
0,46

ChemoOrOncology
$P(O)$
0,012



Cancer			
Age	Oncology	Male	$P(C \mid A, M, O)$
<= 40	T	T	1
<= 40	T	F	1
<= 40	F	T	0.016
<= 40	F	F	0.02
40 – 60	T	T	1
40 – 60	T	F	1
40 – 60	F	T	0.033
40 – 60	F	F	0.042
> 60	T	T	1
> 60	T	F	1
> 60	F	T	0.044
> 60	F	F	0.034

Programmtext 3.2: *beliefProgram.pbl* für Krebs in eICU

```
male(X) :- p_male(X).
oncology(X) :- p_oncology(X).

age(X,100000) :- p_young(X).
age(X,100001) :- NOT p_young(X), p_middle(X).
age(X,100002) :- NOT p_young(X) , NOT p_middle(X) , p_old(X).

cancer(X) :- age(X,100000), oncology(X), male(X), p_young_oncology_male(X).
cancer(X) :- age(X,100000), oncology(X), NOT male(X), p_young_oncology_not_male(X).
cancer(X) :- age(X,100000), NOT oncology(X), male(X), p_young_not_oncology_male(X).
cancer(X) :- age(X,100000), NOT oncology(X), NOT male(X),
    p_young_not_oncology_not_male(X).

cancer(X) :- age(X,100001), oncology(X), male(X), p_middle_oncology_male(X).
cancer(X) :- age(X,100001), oncology(X), NOT male(X), p_middle_oncology_not_male(X).
cancer(X) :- age(X,100001), NOT oncology(X), male(X), p_middle_not_oncology_male(X).
cancer(X) :- age(X,100001), NOT oncology(X), NOT male(X),
    p_middle_not_oncology_not_male(X).

cancer(X) :- age(X,100002), oncology(X), male(X), p_old_oncology_male(X).
cancer(X) :- age(X,100002), oncology(X), NOT male(X), p_old_oncology_not_male(X).
cancer(X) :- age(X,100002), NOT oncology(X), male(X), p_old_not_oncology_male(X).
cancer(X) :- age(X,100002), NOT oncology(X), NOT male(X),
    p_old_not_oncology_not_male(X).

46/100 :: p_male(875123).
12/1000 :: p_oncology(875123).
116/1000 :: p_young(875123).
71/221 :: p_middle(875123).
1/1 :: p_old(875123).
1/1 :: p_young_oncology_male(875123).
1/1 :: p_young_oncology_not_male(875123).
16/1000 :: p_young_not_oncology_male(875123).
20/1000 :: p_young_not_oncology_not_male(875123).
1/1 :: p_middle_oncology_male(875123).
1/1 :: p_middle_oncology_not_male(875123).
33/1000 :: p_middle_not_oncology_male(875123).
42/1000 :: p_middle_not_oncology_not_male(875123).
1/1 :: p_old_oncology_male(875123).
1/1 :: p_old_oncology_not_male(875123).
44/1000 :: p_old_not_oncology_male(875123).
34/1000 :: p_old_not_oncology_not_male(875123).
```

3.3 Synthea

Synthea [22, 18] ist eine open-source Software, mit der medizinische Datenbanken generiert werden können. Die demographischen Daten der Patienten werden dabei anhand von öffentlich zugänglichen demographischen Daten vom US Census Bureau [21] generiert. Synthea unterstützt dabei die Generierung der Datenbank in den Formaten ccda, fhir, text und CSV. Im folgendem Fall werden ausschließlich CSV-Dateien generiert.
Um Synthea für Angerona kompatibel zu machen, müssen die generierten Patientenid's von UUID zu einer eindeutig identifizierbaren id, die nur aus Zahlen besteht umgewandelt werden, weil Angerona als ids nur Zahlen erlaubt und die UUID noch Sonderzeichen und Buchstaben enthält. Dies kann implementiert werden, indem im Quellcode von Synthea in der LifeCycleModule.java die Zeile 136 durch folgende ersetzt wird:

```
attributes.put(Person.ID, String.format("\%040d", new
    BigInteger(UUID.randomUUID().toString().replace("-", ""), 16)));
```

Anschließend kann man mit dem Kommando `.\run_synthea -p [anzahl] -o false` die CSV-Dateien generieren und in eine beliebige Datenbank übertragen. Das -o steht für *overflowPopulation* und sorgt dafür, dass genau die Anzahl an Patienten generiert wird, die übergeben wird. Ansonsten generiert Synthea solange Patienten, bis es `anzahl` viele lebende Patienten gibt. Die Gesamtanzahl an Patienten in der generierten Datenbank beträgt dann nicht `anzahl`, sondern `anzahl` plus die Anzahl bereits gestorbener Patienten.

Um die Daten für das Vorwissen des Angreifers zu modellieren, wird das `Generic Module Framework` von Synthea verwendet. Dadurch lassen sich die Abhängigkeiten für Krankheiten oder sonstige Einweisungsgründe ablesen, indem man in der `Module Gallery` [18] eine beliebige öffnet. Im folgenden wird das Angreifermodell für die Krankheit Osteoporose definiert. Aus der `Module Gallery` in Abbildung 3.4 geht hervor, dass Osteoporose abhängig ist von folgenden Faktoren :

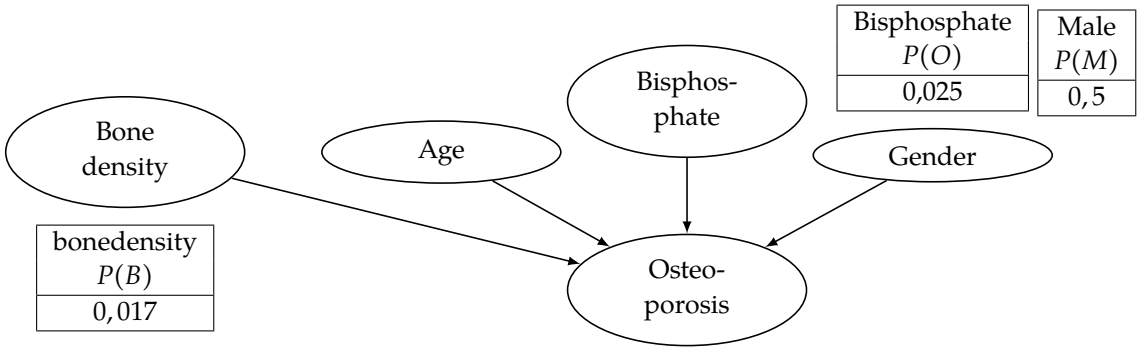
1. **Geschlecht**: Frauen haben ein erhöhtes Risiko an Osteoporose zu erkranken als Männer, wobei Männer eine 0.6-Fache geringere Wahrscheinlichkeit als Frauen haben.
2. **Alter**: Eine Person kann erst an Osteoporose erkranken, wenn diese mindestens 60 Jahre alt ist. Dann steigt die Wahrscheinlichkeit mit dem Alter in 10er Schritten immer mehr an bis zum Alter 90, wodurch sich ein Intervall von $[0, 60]$, $(60, 70]$, $(70, 80]$, $(80, 90]$ und $(90, \infty)$ ergibt.
3. **Knochendichte** (**bone-density**): Wenn die Knochendichte zwischen −3.8 und −2.5 liegt, ist dies ein garantiertes Indiz für Osteoporose.
4. **Medikament** Bisphosphonat : Wenn ein Patient das Medikament Bisphosphonat verschrieben bekommen hat, dann ist dies ebenfalls ein garantiertes Indiz für Osteoporose.

Das Vorwissen für das Geschlecht wurde aus der demographics.csv im Synthea Verzeichnis ausgelesen, indem der Durchschnitt aller Städte gebildet wurde, wobei ein Wahrscheinlichkeitswert von 0.5 berechnet wurde. Die Daten für die Medikamente, Alter und die Knochendichte wurden aus einer Tabelle mit 100.000 Patienten gesampled.
Die Modellierung des Bayes-Netz ist in Abbildung 3.3 zu sehen. Die Wahrscheinlichkeitswerte für $P(A < 60)$ sind nicht gelistet, weil diese die Wahrscheinlichkeit 0 haben, da kein

Patient unter 60 Osteoporose haben kann nach dem Schema aus der `Module Gallery`. Ebenfalls werden die Wahrscheinlichkeiten für $P(O)$ in denen $B = TRUE$ oder $O = TRUE$ ist nicht gelistet, weil diese die Wahrscheinlichkeit 1 haben und die Tabelle ansonsten zu groß zum darstellen ist.
Ein erweitertes Angreifermodell mit mehr Abhängigkeiten wurde durch hinzufügen des Moduls `Injury` erreicht. Hierfür wurde das Angreifermodell mit dem Vorwissen für einen Knochenbruch erweitert. Ein Knochenbruch ist dabei Abhängig von Osteoporose und ein Knochenbruch wird unterteilt in Armbruch, Knöchelbruch, Gelenkbruch, Rippenbruch, Schlüsselbeinbruch und gebrochene Hüfte. Jedoch lässt sich dies nicht in einem für Angerona konformen Bayes-Netz modellieren, da dadurch die Bedingung verletzt wird, dass das Bayes-Netz ein Polytree sein muss. Durch die genannte Modellierung würde nämlich ein Zyklus entstehen zwischen den Knoten Osteoporose ↔Knochenbruch ↔Art des Bruches, weil z.B. ein Armbruch abhängig ist vom Knochenbruch und von Osteoporose. Deshalb wurde die Wahrscheinlichkeit Knochenbruch als Knoten ohne Abhängigkeit definiert und die Wahrscheinlichkeit für ein Knochenbruch aus einer generierten medizinischen Datenbank mit 100.000 Patienten gesampled. Die Wahrscheinlichkeiten für die einzelnen Brüche können aus der `Module Gallery` für `Injury` ausgelesen werden. Dabei sind die Wahrscheinlichkeitswerte für die spezifischen Knochenbrüche bei den *broken bone* den Wert *false* entspricht nicht gelistet, weil diese die Wahrscheinlichkeit 0 haben. Somit ergibt sich das Bayes-Netz Abbildung 3.5

Abbildung 3.3: Synthea Osteoporose Bayes-Netz

Age				
$P(A < 60)$	$P(A = (60,70]$	$P(A = (70,80])$	$P(A = (80,90])$	$P(A \geq 90)$
0.766	0.123	0,0654	0,025	0,021



Osteoporosis				
Age	Male	Medication	Bone density	$P(O \mid A, M, O, B)$
60 – 70	T	F	F	0,0714
60 – 70	F	F	F	0.1
70 – 80	T	F	F	0.0714
70 – 80	F	F	F	0.1
80 – 90	T	F	F	0.143
80 – 90	F	F	F	0.2

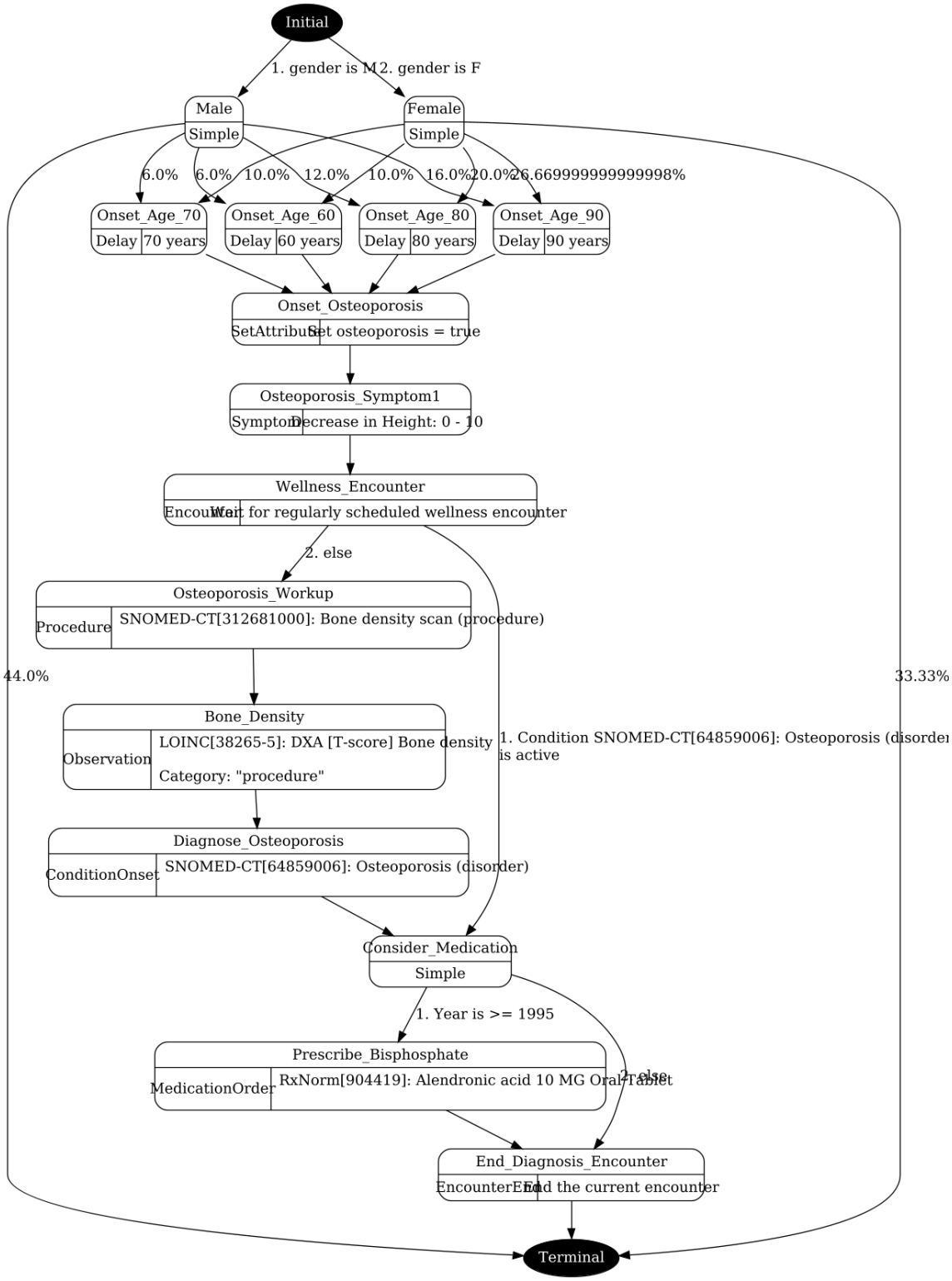
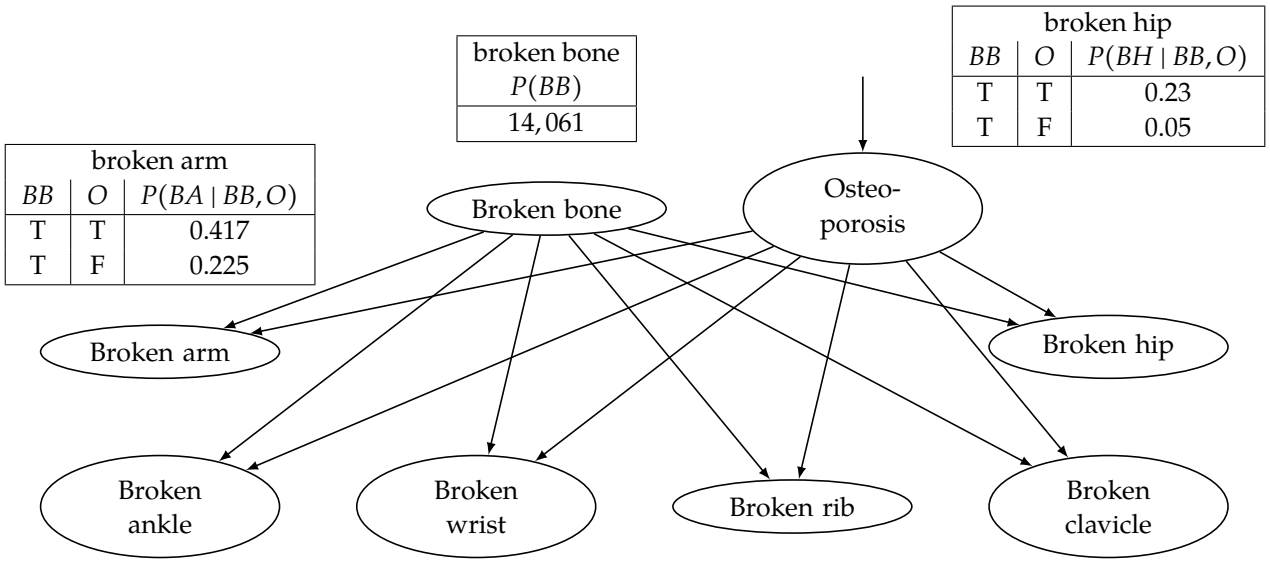


Abbildung 3.4: Synthea Osteoporose Modul [20]

Abbildung 3.5: Synthea Knochenbruch Bayes-Netz



4

Auswertungen der medizinischen Datenbanken

In diesem Kapitel wird die Sicherheit von Angerona an einem praxisnahen Anwendungsfall gezeigt Dafür wird zuerst geprüft, ob der definierte Schwellwert eingehalten wird, indem aus dem Vorwissen des Angreifermodells der Schwellwert berechnet wird. Anschließend wird gezeigt, dass bei einem größeren definierte Schwellwert als das Vorwissen, der Zugriff verweigert wird. Andersherum wird gezeigt, dass bei kleinerem definierten Schwellwert als das Vorwissen, der Zugriff gewährt wird.
Anschließend wird gezeigt, dass mithilfe der Historie erkannt wird, ob beim Zulassen einer Anfrage eine Sicherheitsregel verletzt werden würde.
Um die Laufzeit zu messen wird die Laufzeitmessung aus der Arbeit von Angerona genauer betrachtet und mit unseren Hardwarespezifikationen gemessen. Dann werden mit Synthea zwei verschieden große, synthetische, medizinischen Datenbanken generiert und für das einfache Osteoporose-Beispiel und für das komplexere Knochenbruch-Beispiel geprüft, wie sich die Laufzeiten bei verschieden komplexen Abhängigkeiten verhalten. Anschließend wird die Laufzeit von den medizinischen Datenbanken MIMIC-III und eICU bei wachsender Anzahl von Sicherheitsregeln getestet und geschaut wie sich die Laufzeit dabei verhält.

Die Datenbanken und der Server auf dem Angerona ausgeführt wird laufen auf einer virtuellen Maschine in einem PowerEdge R530 Server. Dieser Server hat zwei Intel Xeon E5-2620 v3 Prozessoren 2,4GHz mit jeweils 6 Core / 12 Threads und 64GB DDR4-SDRAM. Als Datenbanksystem wird PostgreSQL mit der Version 9.5.24 verwendet.

4.1 Sicherheit

Schwellwert

Betrachtet wird im Folgenden das Beispiel aus Abschnitt 2.3. Um zu prüfen, ob der Schwellwert eingehalten wird, wird das Beispiel vereinfacht. Es wird im Angreifermodell das Vorwissen, dafür das Alice Eltern Krebs haben, von Mallory für Alice (id 1) auf 100%

gesetzt. Dies wird umgesetzt, indem die Variablen *p_vaterhatkrebs* und *p_mutterhatkrebs* auf 1/1 gesetzt werden wie in Programmtext 4.1.

Programmtext 4.1: Mallorys Vorwissen

$1/1 :: p\_patient(1).$
$25/100 :: p\_raucher(1).$
$1/1 :: p\_mutterhatkrebs(1).$
$1/1 :: p\_vaterhatkrebs(1).$

Anschließend kann der Schwellwert berechnet werden, indem für jede mögliche Klausel $krebs(X)$ die noch eintreffen kann, die Wahrscheinlichkeiten aufaddiert werden. Diese ist in dem Fall folgende, weil nur noch die Abhängigkeit „Raucher“ variabel ist:

- `krebs(X) :- patient(X), raucher(X), mutterhatkrebs(X), vaterhatkrebs(X), v1111(X).`
- `krebs(X) :- patient(X), NOT raucher(X), mutterhatkrebs(X), vaterhatkrebs(X), v1011(X).`

Somit ist der Schwellwert für $krebs(X) = (\frac{1}{1} \cdot \frac{25}{100} \cdot \frac{1}{1} \cdot \frac{1}{1} \cdot \frac{6}{10}) + (\frac{1}{1} \cdot 1 - \frac{25}{100} \cdot \frac{1}{1} \cdot \frac{1}{1} \cdot \frac{35}{100}) = \frac{33}{80}$. Nun kann die Sicherheitsregel für krebs(X) für Mallory auf den Schwellwert gesetzt werden und es wird als Ergebnis von Angerona *true* ausgegeben, da auf den Wert zugegriffen werden darf, wie in Programmtext 4.2 zu sehen ist. Allerdings wird bei einem Schwellwert $> \frac{33}{80}$ der Zugriff verweigert, wie im Programmtext 4.3 zu sehen ist.

Programmtext 4.2: Angerona Ergebnis für Schwellwert=$\frac{33}{80}$

Write a command. Write 'Stop' to conclude.
AS mallory:krebs('1')
ACCESS CONTROL RESULT
    Acc.Ctrl.: true
ACTION RESULT
    Result: true Exception: false

Programmtext 4.3: Angerona Ergebnis für Schwellwert=$\frac{34}{80}$

Write a command. Write 'Stop' to conclude.
AS mallory:krebs('1')
ACCESS CONTROL RESULT
    Acc.Ctrl.: false Cause: mallory : krebs ('1') is not authorized

Damit wurde gezeigt, dass Angerona den Schwellwert richtig berechnet und den Zugriff erfolgreich verweigert, wenn das Vorwissen größer dem Schwellwert ist.

Historie

Im Folgenden wird das Beispiel von Abschnitt 4.1 fortgesetzt. Um die Historie zu prüfen, wird an Angerona die Anfrage gestellt, ob Alice eine Raucherin ist. Aber da die Information dafür, dass Alice Raucherin ist, den Schwellwert auf $\frac{6}{10}$ anhebt und somit das

Vorwissen ≥ Schwellwert ist, wird der Zugriff verweigert, wie in Programmtext 4.4 zu sehen ist.

Programmtext 4.4: Angerona Ergebnis für die Abfrage, ob Alice Raucherin ist

Write a command. Write 'Stop' to conclude.
AS mallory:raucher('1')
ACCESS CONTROL RESULT
Acc.Ctrl.: false Cause: mallory : raucher ('1') is not authorized

4.2 Laufzeit

Die Messung der Laufzeit wird in *Onlinezeit* und *Offlinezeit* unterteilt. Die *Onlinezeit* beschreibt die Zeit, die vergeht, zwischen dem Stellen einer Anfrage und dem Erhalten des Ergebnisses der Anfrage. Die *Offlinezeit* beschreibt die Zeit zwischen dem Start von Angerona bis zu dem Zeitpunkt an dem Angerona bereit für eine Eingabe ist.
Dabei ist bei der Onlinezeit besonders wichtig, dass diese nicht zu hoch ist, da sonst die Benutzer lange Wartezeiten für jede Anfrage haben. Die Offline Zeit hingegen kann auch länger dauern, da Angerona in der Praxis nur einmal gestartet wird.
Jedoch ist bereits beim Start der kleineren medizinischen Datenbank MIMIC-III die Offlinelaufzeit nach 24 Stunden zu keinem Ende gekommen. Deshalb wird im ersten Schritt mit Synthea getestet, wie groß eine Patientendatenbank werden kann, sodass diese noch in angemessener Zeit nutzbar ist. Anschließend wird gezeigt, wie Angerona sich bei wachsender Anzahl von Sicherheitsregeln verhält.
Dabei ist jede Onlinezeit das `worst-case` Szenario und bedeutet das der Schwellwert auf 0 gesetzt wird, damit die Überprüfung der Sicherheitsregel bis zum Schluss ausgeführt wird, die Anfrage nicht verweigert und somit abgebrochen wird. Es wurden dabei nicht wie im Paper nur 100 Sicherheitsregeln definiert, sondern jeweils immer für alle Patienten versucht, weil dies praxisnäher ist um die Sicherheit zu gewährleisten. Denn wenn nur die Patienten abgesichert werden, die zum Beispiel Krebs haben, dann könnte ein Angreifer ohne Historie für alle Patienten abfragen, ob diese Krebs haben und würde nur `false` oder `security exceptions` erhalten. Wenn der Angreifer dann durch statistische Recherche herausfindet, dass das Verhältnis der Anzahl der erhaltenen `security exceptions` zu der bei den `false` erhalten wurde ca. der Wahrscheinlichkeit entspricht das ein beliebiger Patient Krebs hat, kann der Angreifer daraus schließen, dass die Patienten für die eine `security exception` ausgegeben wurde mit einer sehr hohen Wahrscheinlichkeit Krebs haben bzw. vielleicht sogar alle Krebs haben.
Jedoch kann dies umgangen werden, indem einigen Patienten die kein Krebs haben auch eine Sicherheitsrichtlinie hinzugefügt wird, da dann der Angreifer alle möglichen Ergebnisse erhalten kann. Deshalb sollte mindestens für die Leute die Krebs haben eine Sicherheitsrichtlinie definiert werden, aber es ist von Vorteil noch mehr Sicherheitsregeln zu definieren.

Vorhandene Auswertungen

Die Laufzeit von Angerona wurde bereits in der Arbeit von Guarnieri et. al. [7] analysiert. Dabei wurde ein synthetisches `beliefProgram.pbl` für 1,000 bis 100,000 Patienten generiert, wovon ein Beispiel für 1 Patienten in Programmtext 4.5 zu sehen ist. Davon wurden 100 Patienten mit Sicherheitsregeln abgesichert und anschließend zufällig 100 Anfragen gestellt.

Programmtext 4.5: `beliefProgram.pbl` für das Beispiel aus [16]

$cancer(X) :- patient(X), sw1(X).$
$cancer(X) :- smokes(X), sw2(X).$
$cancer(Y) :- father(X,Y), mother(Z,Y), cancer(X), NOT\ cancer(Z), sw3(Y).$
$cancer(Y) :- father(X,Y), mother(Z,Y), NOT\ cancer(X), cancer(Z), sw3(Y).$
$cancer(Y) :- father(X,Y), mother(Z,Y), cancer(X), cancer(Z), sw3(Y), sw4(Y).$
$cancer(Y) :- father(X,Y), mother(Z,Y), cancer(X), cancer(Z), NOT\ sw3(Y), sw4(Y).$
$1/1 :: patient(0).$
$1/20 :: sw1(0).$
$5/19 :: sw2(0).$
$3/14 :: sw3(0).$
$3/7 :: sw4(0).$

Das Ergebnis für die Onlinezeit von Guarnieri et. al. [7] wird in Abbildung 4.1 dargestellt. Das Experiment wurde von uns für dieselbe Modellierung, aber unseren Hardware Spezifikationen vom Server wiederholt und eine graphische Modellierung des Ergebnisses für die Onlinezeiten ist in Abbildung 4.2 zu sehen. Die Onlinezeiten unterscheiden sich dabei nicht sehr stark, da diese bei unter 1 Sekunde bleibt bei beliebiger Größe der Datenbank bis 100,000 Patienten.
Die Offlinezeit hingegen unterscheidet sich minimal zu der angegebenen aus der Arbeit von Guarnieri et. al. [7], in der eine Maximale-Offlinezeit von 2,5 Minuten für beliebig große Datenbanken bis 100,000 Patienten gemessen wurde. Nach unseren Messungen wächst die Offlinezeit bei wenig definierten Sicherheitsregeln linear und hat bei einer Implementierung für 100,000 Patienten eine Offlinelaufzeit von fast 11 Minuten. Dies kann den Grund haben, dass bei unseren Messungen die Initialisierung von Angerona anhand der 3 Dateien `beliefProgram.pbl`, `initStatements.txt` und `template.cpt` geschieht und diese noch von Angerona eingelesen und die Daten an die Datenbank gesendet werden müssen. In dem Beispiel aus der Arbeit von Guarnieri et. al. [7] hingegen wurden die Beispiele direkt in den Code eingebaut, wodurch der Übersetzungsschritt der 3 Dateien wegfällt.

Komplexität des Angreifermodell

Um zu prüfen, wie Angeronas Laufzeit sich verhält bei wachsender Komplexität der Abhängigkeiten, wird das Osteoporose- und Knochenbruch-Beispiel aus der Synthea Datenbank in Angerona modelliert. Dafür wurden für das Osteoporose-Beispiel für alle Patien-

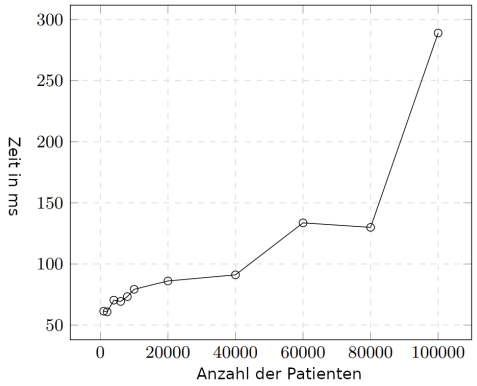


Abbildung 4.1: Onlinezeit aus [7]

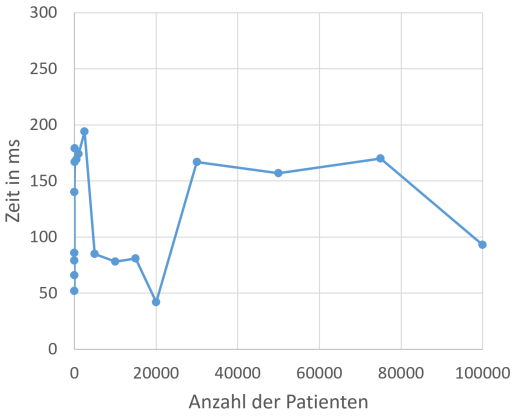


Abbildung 4.2: Onlinezeit Osteoporose

ten eine Sicherheitsregel für die Abfrage nach der Krankheit Osteoporose festgelegt mit einem Schwellwert von 0, damit der *worstcase* der Onlinelaufzeit betrachtet wird. Für das Knochenbruch-Beispiel wurde für jeden Patienten alle Arten von Knochenbrüche abgesichert. Eine graphische Modellierung für den Vergleich der Offlinezeit ist in Abbildung 4.3 und der Onlinezeit in Abbildung 4.4 zu sehen.
Dabei wächst für kleinere Datenbankgrößen die Laufzeit kaum und wird erst bei einer Datenbankgröße mit 3000 Patienten bemerkbar, da hierbei die Differenz der Onlinezeit vom Knochenbruch-Beispiel ca. 750 Millisekunden länger ist und die Offlinezeit ca. 12 Minuten länger als beim Osteoporose-Beispiel. Die Differenz der Komplexität steigt jedoch für komplexere Datenbanken konstant mit, sodass bereits bei einer Datenbankgröße mit 5000 Patienten die Differenz der Offlinezeit 23 Minuten und die Onlinezeitdifferenz 6 Sekunden beträgt. Gründe für die erhöhten Laufzeiten beim Knochenbruch-Beispiel sind dabei, dass Daten in die Datenbank eingefügt werden, die Angreifermodellierung komplexer und somit die `beliefProgram.pbl` größer ist und dass es mehr Abhängigkeiten gibt und somit mehr probabilistische Fakten je Patient modelliert werden.

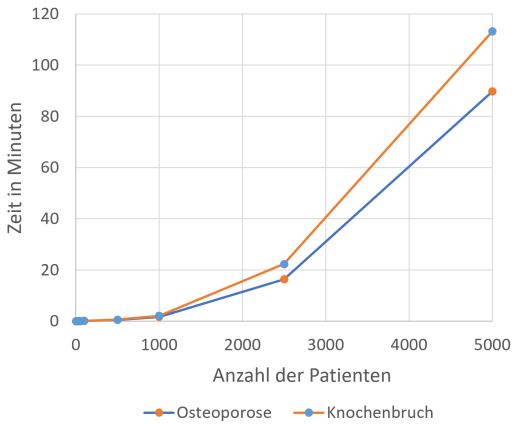


Abbildung 4.3: Offlinezeiten Synthea

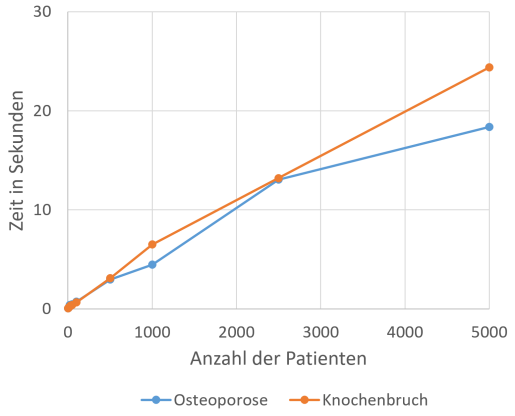


Abbildung 4.4: Onlinezeiten Synthea

MIMIC-III und eICU bei wachsenden Sicherheitsregeln

Die MIMIC-III und eICU Datenbank lässt sich nur in Angerona starten, wenn man die Sicherheitsregeln nur für die Patienten definiert, die Krebs haben. Wenn für alle Patienten einen Sicherheitsrichtlinie definiert werden soll, überschreitet die Offlinelaufzeit bereits 24 Stunden und lässt sich somit nicht ausführen. Deshalb wird im Folgenden geprüft, wie sich die Online- und Offlinelaufzeiten verhalten bei wachsenden Sicherheitsregeln für die Information, dass ein Patient Krebs hat. Zum Schluss wird die Laufzeit gezeigt, wenn man das Vorwissen des Angreifers und die Sicherheitsregeln nur für Patienten mit Krebs definiert.

Um zu testen wie Angerona sich bei wachsender Größe des Angreifermodells verhält, wird die MIMIC-III und eICU Datenbank verwendet. Dabei werden jeweils für die Datenbanken die Sicherheitsregeln für die Patienten in der `initStatements.txt` und das Vorwissen in der `beliefProgram.pbl` erweitert. Dadurch ergeben sich die Onlinelaufzeiten in Abbildung 4.6 und die Offlinelaufzeiten in Abbildung 4.5 .
Auffällig ist, dass die Offlinezeiten annähernd gleich steigen, jedoch verschiedene Startzeiten haben bei nur einer definierten Sicherheitsrichtlinie. Dies ist dadurch zu erklären, dass initial die benötigte Datenbank generiert wird und diese beim eICU-Beispiel fast vier mal größer ist, als beim MIMIC-III-Beispiel. Die Offlinelaufzeit ist jedoch bei nur einer definierten Sicherheitsrichtlinie nicht vier mal länger für MIMIC-III, denn zum Start ist die Offlinelaufzeit für eICU mit 48.8 Minuten knapp 8 mal länger als für das MIMIC-III-Beispiel mit einer Offlinelaufzeit von ca. 6.5 Minuten. Der Faktor verringert sich jedoch bei wachsender Anzahl der Sicherheitsregeln und ist bereits bei 10,000 definierten Sicherheitsregeln nur noch knapp 2 mal länger mit einer Offlinelaufzeit für eICU von ca. 105 Minuten und für MIMIC-III mit 65 Minuten.
Die Onlinelaufzeit wächst dabei für MIMIC-III und eICU annähernd linear mit der Anzahl der Sicherheitsregeln, wobei für größere Datenbanken wie bei eICU die Onlinelaufzeit generell etwas höher liegt. Zwar ist bei bis zu 1000 Sicherheitsregeln die Laufzeit identisch und bleibt bei unter 3 Sekunden für beide Fälle, jedoch ab 2000 Sicherheitsregeln ist die Laufzeit für größere Datenbanken (eICU) länger als für kleinere Datenbanken (MIMIC-III). Der Unterschied der Onlinelaufzeit bleibt jedoch annähernd konstant und ist zum Beispiel für die viermal größere Datenbank eICU ca. 4 Sekunden länger als für die MIMIC-III-Datenbank. Für das MIMIC-III-Beispiel wurde außerdem beim Absichern aller Patienten, die Krebs haben, was eine Menge von 13658 Sicherheitsregeln entspricht, eine Offlinelaufzeit von 77 Minuten und eine Onlinelaufzeit von 36.4 Sekunden gemessen. Außerdem ist dabei aufgefallen, dass während Angerona läuft 6.7 GB vom Arbeitsspeicher reserviert wurde.
Für das eICU-Beispiel wurde beim Absichern aller Patienten die Krebs haben, was eine Menge von 8179 Sicherheitsregeln entspricht, eine Offlinelaufzeit von 89 Minuten und eine Onlinelaufzeit von 23.6 Sekunden gemessen. Außerdem ist dabei aufgefallen, dass während Angerona läuft 5.8 GB vom Arbeitsspeicher reserviert wurde.
Hierbei sieht man, dass die Datenbankgröße, also die Größe der `initStatements.txt` eher eine Auswirkung auf die Offlinelaufzeit und die Anzahl der Sicherheitsregeln, also die Größe der `beliefProgram.pbl`, auf die Onlinelaufzeit hat.

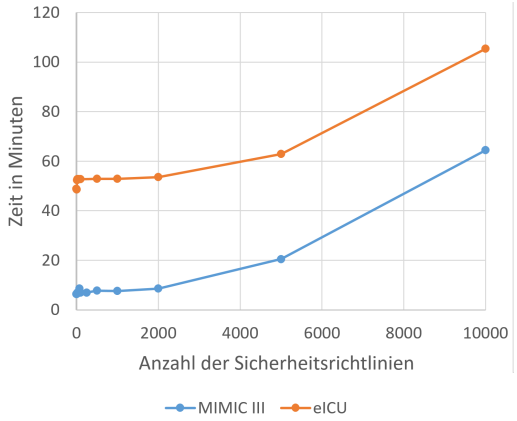


Abbildung 4.5: Offlinezeiten Sicherheitsregeln

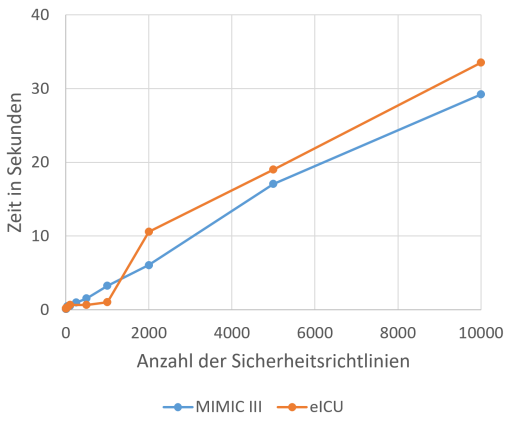


Abbildung 4.6: Onlinezeiten Sicherheitsregeln

Das Ergebnis der Evaluierung aus Guarnieri et. al. [7] wurde unter angepassten Voraussetzung für Angerona getestet, wodurch die Gute Laufzeit zustande kam. Wie man in den Auswertung dieser Arbeit sieht, ist die Nutzung von Angerona in realen Anwendungsfällen nicht möglich, weil die Offlinezeiten und Onlinezeiten zu hoch sind. Meiner Meinung nach wäre eine Offlinelaufzeit von maximal 30 Minuten und eine Onlinelaufzeit von maximal 5000 Millisekunden akzeptabel um solch ein System in einem realen Anwendungsfall nutzen zu können. Damit wäre Angerona bei einer Datenbankgröße mit bis zu 100.000 Patienten, nicht zu komplexen Abhängigkeitsregeln und ca. 1000 definierten Sicherheitsregeln effizient nutzbar. Diese Größen reichen für die meisten realen Anwendungsfälle jedoch nicht aus.
Ein weiteres Problem ist, dass bei jeder Datenbankänderung einer von Angerona abgesicherten Datenbank, die `initStatements.txt` neu generiert bzw. ergänzt werden muss um die geänderten Werte. Dadurch muss Angerona komplett neugestartet werden und solange ist die Datenbank die Dauer der Offlinezeit nicht erreichbar. Da dies der Fall ist, ist selbst die Offlinelaufzeit von 30 Minuten zu hoch und dürfte meiner Meinung nach 5 Minuten nicht überschreiten.
Unter diesen Bedingungen würde Angerona in kaum einem realen Anwendungsfall nutzbar sein, da selbst eine kleine Datenbank, wie MIMIC-III, nicht in einer effizienten Offlinezeit Nutzbar wäre.

5

Zusammenfassung und Ausblick

In dieser Arbeit wurde der Prototyp Angerona in ein praxisnahem Anwendungsfall angewendet und evaluiert. Dabei wurde festgestellt, dass die definierten Sicherheitsregeln von Angerona wie spezifiziert eingehalten wurden. Außerdem wird die Sicherheit für die Historie ebenfalls eingehalten, da Angerona erfolgreich prüft, ob eine Anfrage eine Sicherheitsrichtlinie verletzten würde, wenn diese genehmigt werden würde.
Angerona bietet jedoch für praxisnahe Anwendungsfälle keine ausreichende Laufzeit, weil die Onlinelaufzeit bereits für die kleinere Datenbank MIMIC-III mit 36.4 Sekunden zu hoch ist. Dies liegt daran, dass Angeronas Laufzeit stark durch die Anzahl der definierten Sicherheitsregeln und der dazugehörigen Angreifermodellierung für jeden Krankenhausaufenthalt beeinflusst wird. Ein anderer Faktor der die Laufzeit beeinflusst, ist die Komplexität der definierten Abhängigkeiten im Angreifermodell. Diese haben einen großen Einfluss auf die Laufzeit, welche bei wachsender Datenbankgröße immer länger benötigt, wie am Synthea-Beispiel gezeigt wurde.
Hinzu kommt, dass in der Praxis meist noch komplexere Modellierungen abgesichert werden müssen, wenn zum Beispiel noch mehr als nur eine Krankheit abgesichert werden soll. Wenn man davon ausgeht, dass alle Patienten mit allen Krankheiten abgesichert werden sollen, dann würde es nicht möglich sein, dies mit Angerona umzusetzen, weil die Laufzeit zu hoch wird und der Arbeitsspeicher nicht ausreichen wird.
Jedoch wurde in Guarnieri et. al. [7] bereits erwähnt, dass die Laufzeit durch Parallelisierung verbessert werden kann, da dieser momentan nur auf einem Kern läuft. Deshalb ist der Prototyp Angerona eine gute Grundlage, um diesen zu verbessern oder um sich für weitere Entwicklungen von DBIC-Mechanismen inspirieren zu lassen. Eine Erweiterung, die das Laufzeitverhalten ebenfalls beeinflussen könnte, wäre ein Gruppensystem für die Definition der probabilistischen Abhängigkeitsregeln, sodass es möglich ist das Vorwissen für mehrere Patienten gleichzeitig zu definieren. Außerdem wäre eine bessere Ausgabe von Fehlermeldungen hilfreich beim Interagieren mit Angerona, da Angerona sich bei falschen Anfragen beendet und unklare Fehlermeldungen auswirft.

Literatur

[1] Ahmed, K., Kawsar, A.-A., Kawsar, E., Emran, A.-A., Jesmin, T., Mukti, R. F., Rahman, M., Ahmed, F. u. a. Early detection of lung cancer risk using data mining. In: 2013.

[2] Barasch, A., Morse, D. E., Krutchkoff, D. J. und Eisenberg, E. Smoking, gender, and age as risk factors for site-specific intraoral squamous cell carcinoma. A case-series analysis. In: *Cancer* 73(3):509–513, 1994.

[3] Beutelspacher, A. *„Das ist o. B. d. A. trivial!": Tipps und Tricks zur Formulierung mathematischer Gedanken (Mathematik für Studienanfänger)*. Ninth, updated edition. Vieweg+Teubner Verlag, 2009.

[4] *Datenschutz-Grundverordnung(DSGVO)*. URL: https://dsgvo-gesetz.de/ (besucht am 22. 01. 2020).

[5] *Europäische Union: Anteil der Raucher an der Gesamtbevölkerung, aufgeschlüsselt nach Geschlecht und Mitgliedstaat im Jahr 2017*. Jan. 2020. URL: https://de.statista.com/statistik/daten/studie/1099197/umfrage/anteil-der-raucher-in-der-eu-nach-geschlecht/#professional (besucht am 12. 01. 2021).

[6] Golibrzuch, P., Möller, R. und Kaya, A. A study of the rough set approach for image understanding. In: 2021.

[7] Guarnieri, M., Marinovic, S. und Basin, D. Securing Databases from Probabilistic Inference. In: *Proceedings of the 30th IEEE Computer Security Foundations Symposium*. IEEE, 2017, S. 343–359.

[8] *HIPAA*. URL: https://www.hhs.gov/hipaa/for-professionals/security/laws-regulations/index.html (besucht am 30. 12. 2020).

[9] *ICD-9 Codes for Cancer*. URL: https://medicine.yale.edu/intmed/vacs/instruments/ (besucht am 22. 01. 2020).

[10] Ingelheim, A. Das erste Jahr DSGVO - Eine Bestandsaufnahme. In: Springer, 2019.

[11] Kučera, M., Tsankov, P., Gehr, T., Guarnieri, M. und Vechev, M. Synthesis of Probabilistic Privacy Enforcement. In: Association for Computing Machinery, 2017. URL: https://doi.org/10.1145/3133956.3134079%7D.

[12] Luc De Raedt∗, A. K. und Toivonen, H. *ProbLog: A Probabilistic Prolog and its Application in Link Discovery*. Techn. Ber. Machine Learning Lab, Albert-Ludwigs-University Freibur, 2007.

[13] *Problog*. 30. Dez. 2020. URL: https://dtai.cs.kuleuven.be/problog/index.html#.

[14] Robertson, W. MIMIC-III, a freely accessible critical care database. In: 2016.

[15] Russell, S. und Norvig, P. Artificial intelligence: a modern approach. In: 2002.

[16] Securing Databases from Probabilistic Inference. In: URL: https://mguarnieri.github.io/publication/csf2017/ (besucht am 30. 12. 2020).

[17] Sellami, M., Hacid, M.-S., Gammoudi und Mohsen, M. A FCA framework for inference control in data integration systems. In: *Distributed and Parallel Databases* 37(4):543–586, 2019.

[18] *Synthea*. 2016. URL: https://github.com/synthetichealth/synthea.

[19] *Synthea Module Gallery*. 23. Jan. 2021. URL: https://github.com/synthetichealth/synthea/wiki/Module-Gallery.

[20] *Synthea Module Gallery Osteoporosis*. 15. Feb. 2021. URL: https://synthetichealth.github.io/synthea/graphviz/osteoporosis.png.

[21] *United States Census bureau*. URL: https://www.census.gov/.

[22] Walonoski, J., Kramer, M., Nichols, J., Quina, A., Moesel, C., Hall, D., Duffett, C., Dube, K., Gallagher, T. und McLachlan, S. Synthea: An approach, method, and software mechanism for generating synthetic patients and the synthetic electronic health care record. In: *Journal of the American Medical Informatics Association* 25(3):230–238, 2018.