1. Lab zu „Formale Methoden im Softwareentwurf“: PROMELA/Spin

TECHNISCHE UNIVERSITÄT DARMSTADT

Abgabe (Ausschlussfrist): 17.12.2018, 9:00 (strikt)
via Moodle unter
https://moodle.informatik.tu-darmstadt.de/mod/assign/view.php?id=17062

Hinweise:

- Die Lösung ist als ein einziges zip-Archiv mit dem Namen: `lab1_GruppeXY.zip` abzugeben, wobei Sie bitte `XY` mit Ihrer Labgruppennummer ersetzen.

  Das Archiv muss die Unterverzeichnisse `aufgabe1` und `aufgabe2` enthalten. In diesen Unterverzeichnissen legen Sie dann bitte jeweils die geforderten Dateien mit den Lösungen der Aufgaben ab. Die Dateien müssen alle Informationen für die Lösung beinhalten.

  Die abzugebenden PROMELA-Modelle dürfen keine Syntaxfehler beinhalten.

  Bis zur Ausschlussfrist kann die Lösung beliebig häufig hochgeladen werden. Bewertet wird die zuletzt hochgeladene Version. Es reicht, wenn eine Person Ihrer Labgruppe die Lösung hochlädt.

Der Fachbereich Informatik misst der Einhaltung der Grundregeln der wissenschaftlichen Ethik großen Wert bei. Zu diesen gehört auch die strikte Verfolgung von Plagiarismus. Weiter Informationen finden Sie unter: https://www.informatik.tu-darmstadt.de/de/sonstiges/plagiarismus/

Hinweise:

- LTL Formeln können im Modell wie folgt definiert und benannt werden: `ltl NAME {LTL_FORMEL}` mit `NAME` und `LTL_FORMEL` als Platzhalter für die entsprechenden Elemente

Aufgabe 1 Denksport — 20 Punkte

Dora, Bernd, Karin und Thomas stehen zusammen am Ende einer langen, engen Brücke in der tiefsten Nacht. Sie wollen die Brücke überqueren, haben jedoch nur eine Taschenlampe dabei. Die Brücke kann höchstens von zwei Personen gleichzeitig überquert werden, die die Taschenlampe mit dabei haben müssen.

Die Personen sind unterschiedlich schnell. Laufen sie gemeinsam mit jemand anderem, so laufen sie mit der Geschwindigkeit der langsameren Person. Dora ist die Schnellste und benötigt nur 1 Minute, um die Brücke zu überqueren, Bernd benötigt 2 Minuten, Karin 5 Minuten und Thomas 10 Minuten.

Hinweise:

- Ihr Modell darf keine Hinweise auf die Lösung enthalten.
- Sehen Sie sich die Ergebnisse von Spin im Detail an und akzeptieren Sie nicht blindlings die gegebene Antwort.

Aufgabe 1.1 — 15 Punkte

Schreiben Sie ein PROMELA Modell für das gegebene Szenario. Es soll nicht-deterministisch auswählen, wer die Brücke überquert und dann die Zeit entsprechend erhöhen.

Zur Modellierung der Zeit genügt für dieses Rätsel eine **`byte`** Variable. Unabhängig davon, beachten Sie das Verhalten ihres Modells im Falle eines Overflows und beschreiben Sie in einem Kommentar ihren Umgang mit Overflows (bzw. wieso bei Ihnen keine auftreten) und u.U. welche notwendigen Einschränkungen Sie vornehmen mussten. Für das fertige Modell sollte Spin keine ungültigen Endzustände melden (sie dürfen begründete `endXXX`-Label setzen).

Geben Sie die Lösung als Datei unter `aufgabe1/puzzle.pml` ab.

Aufgabe 1.2 — 5 Punkte

Überlegen Sie sich, wie lange die schnellste Lösung dauert, sagen wir hier x Minuten. Formulieren Sie zunächst in Spin, dass es keine Lösung gibt, die schneller als x Minuten ist. Formalisieren Sie dann im nächsten Schritt, dass es keine Lösung in x Minuten gibt. Spin sollte Ihnen nun ein Gegenbeispiel liefern. Geben Sie die Lösung aus dem Gegenbeispiel in natürlicher Sprache und allgemein verständlich an.

Ob Sie die Aufgabe mit LTL lösen oder über Zusicherungen ist Ihnen freigestellt.

Ergänzen Sie die Datei `aufgabe1/puzzle.pml` mit den benötigten Zusicherungen bzw. LTL Formeln. Geben Sie außerdem den Trace des Gegenbeispiels als Datei `aufgabe1/puzzle.pml.trail` an und legen Sie Ihre Erklärung als Textdatei unter `aufgabe1/puzzle.txt` ab.

Aufgabe 2 Protokolle — 45 Punkte

Um eine sichere Kommunikation über ein unsicheres Netzwerk zu ermöglichen, wurde eine Vielzahl von Kommunikationsprotokollen entworfen. Die meisten dieser Protokolle hatten eines gemeinsam: Kritische Fehler in ihrer Originalversion. Nicht selten wurden auch bei der Beseitigung von Fehlern neue Fehler eingebaut.

In dieser Aufgabe sind die an der geheimen Kommunikation beteiligten Personen Beate und Ingo beteiligt. Beate und Ingo wollen untereinander geheime Nachrichten austauschen. Wir kodieren eine Nachricht im folgenden Format:

$$S \to R : (C)_{PK}$$

Dies bedeutet: Der Sender S schickt eine Nachricht an den Empfänger R mit Inhalt C, welcher mit dem öffentlichen Schlüssels PK verschüsselt ist.

Das Kommunikationsprotokoll dieser Aufgabe etabliert die geheime Kommunikation nach folgendem Ablauf (S Sender, E Empfänger) :

(`msgId1`) $S \to E : (S, NS)_{kE}$

(`msgId2`) $E \to S : (NS, NE)_{kS}$

(`msgId3`) $S \to E : (NE)_{kE}$

O.b.d.A. beginnt bei uns Beate und initiiert die verschlüsselte Session mit Ingo gemäß dem obigen Protokoll wie folgt:

Beate schickt zunächst eine Nachricht an Ingo mit dem Inhalt „*Hallo, ich bin Beate und möchte mit Dir reden. Hier hast Du meine Nonce (NB)*[1]“ und sie verschlüsselt die Nachricht mit Ingos öffentlichem Schlüssel *kI*. Nach Empfang der Nachricht antwortet Ingo mit einer unter Verwendung von Beates öffentlichem Schlüssel verschlüsselten Nachricht an Beate. Diese Nachricht enthält sowohl seine Nonce (*NI*) als auch Beates Nonce aus der vorherigen Nachricht. Abschließend antwortet Beate Ingo zur Bestätigung mit einer Nachricht, die Ingos Nonce enthält und mit seinem öffentlichen Schlüssel verschlüsselt ist.

Der Zweck dieses Protokolls ist, dass Beate und Ingo ein gemeinsames Geheimnis etablieren und sich nach Abschluss des Austausches sicher sein können, miteinander zu sprechen.

Wir wollen nun dieses Protokoll als PROMELA-Programm modellieren. Das Programm für Beate ist in der Datei `protocol.pml` enthalten. In einer echten Implementierung müssten die Noncen zufällig generiert werden. In unserem Modell können wir dies vereinfachen und nehmen an die Noncen `nonceB` und `nonceI` sind vorberechnet. Wichtig ist jedoch, dass in dem Modell berücksichtigt wird, dass am Anfang Beate und Ingo nur Ihre jeweiligen Noncen kennen und jeder andere Teilnehmer (insbesondere potentielle Angreifer) *keine* der Noncen von vornherein kennt.

Verschlüsselte Nachrichten realisieren wir in unserem Modell als ein Record `EncMsg` := (`key`, `content1`, `content2`), welches aus einem Schlüssel `key` und zwei Dateneinträgen `content1` und `content2` besteht. Das Verschlüsseln einer Nachricht besteht dann aus dem Erzeugen eines solchen Records und das Entschlüsseln aus dem Vergleich des in der Nachricht enthaltenen Schlüssels mit dem eigenen Schlüssel.

Um das Modell einfach zu halten, modellieren wir das Netzwerk als einen globalen Rendez-Vous Kanal (Bufferkapazität 0). Eine über das Netzwerk verschickte Nachricht ($msgId, to, enc$) besteht aus einer Nachrichtennummer *msgId* (also `msgId1`, `msgId2` oder `msgId3`), dem beabsichtigten Empfänger *to* und der verschlüsselten Nachricht *enc*. Zu Verifikationszwecken gibt es die vier globalen Variablen (ghost variables) `partyB`, `partyI`, `statusB`, `statusI`. Die Variable `partyB` enthält den beabsichtigten Kommunikationspartner von Beate (analog `partyI`). Zu Anfangs gilt `partyB ==agentI`.

[1] Nonce: number used once; für jede Kontaktaufnahme einmalig zufällig generierte Nummer.

Aufgabe 2.1 — 15 Punkte

Erweitern Sie das gegebene Modell (`protocol.pml`) um eine Modellierung von Ingo, so dass das oben beschriebene Modell realisiert wird. Lassen Sie Ingo die Variable `statusI` als letzte Anweisung auf den Wert `ok` setzen, wenn das Protokoll erfolgreich der Beschreibung entsprechend durchlaufen wurde (ansonsten darf Ingo z.B. blocken). Führen Sie einige Simulationsläufe durch. Es sollte alles gemäß dem Protokoll ablaufen und am Ende sollte bei beiden Prozessen immer die jeweilige Statusvariable `statusB` bzw. `statusI` auf den Wert `ok` gesetzt worden sein. Spezifizieren Sie in LTL, dass dies der Fall ist und verifizieren Sie es mit Spin.

Legen Sie Ihre Lösung als Datei `aufgabe2/protocol21.pml` bei, die PROMELA Datei muß auch die LTL Formel unter dem Namen `BEIDE_OK` enthalten.

Aufgabe 2.2 — 6 Punkt

Wir wollen jetzt unser Protokoll auf Sicherheit überprüfen. Dazu untersuchen wir, ob die Kommunikation zwischen Beate und Ingo durch einen dritten Teilnehmer genannt `Angreifer` gefährdet ist.

Das PROMELA Modell für den `Angreifer` ist in der Datei `attacker.pml` enthalten. Es fällt auf, dass der Prozess des `Angreifers` in seinen möglichen Aktionen und deren Abfolge hochgradig nichtdeterministisch realisiert ist. Die Fähigkeiten des Angreifers sind trotzdem recht begrenzt: er kann Nachrichten aus dem Netzwerk entgegennehmen, empfangene Nachrichten erneut wiederverschicken oder einen neuen Verbindungsaufbau gemäß des Protokolls starten.

Kopieren Sie den Prozess des `Angreifers` in eine Kopie ihres bestehenden Modells aus der vorherigen Teilaufgabe. Wenn Sie jetzt das Modell auf die gleiche Eigenschaft wie in der vorherigen Teilaufgabe (`BEIDE_OK`) verifizieren, sollten Sie ein Gegenbeispiel erhalten. Legen Sie ihr Modell und das Gegenbeispiel unter dem Namen `aufgabe2/protocol22.pml` bzw. `aufgabe2/protocol22.pml.trail` ab. Erklären Sie anhand des erzeugten Gegenbeispiels, warum die Verifikation fehlschlägt. Legen Sie ihre Erklärung als `aufgabe2/gegenbeispiel22.txt` ab.

Aufgabe 2.3 — 2 Punkte

Bis jetzt ist der `Angreifer` ein Außenstehender. Das Modell des Angreifers soll jetzt erweitert werden, so dass der Angreifer auch ein gleichberichtigter Teilnehmer an Kommunikationen sein kann. Dazu soll er ein möglicher Kommunikationspartner von Beate werden, d.h., Beate soll nicht nur eine Kommunikation mit Ingo anfangen, sondern auch mit dem Angreifer.

Ändern Sie dazu das Modell von Beate so ab, dass sie nicht-deterministisch entweder Ingo oder den Angreifer als Ihren Kommunikationspartner für einen Lauf des Protokolls wählt. Das Ergebnis der Wahl sollte sich in dem Wert der Variablen `partyB` widerspiegeln. Sobald die Wahl getroffen ist, werden die nachfolgenden Nachrichten mit dem Schlüssel des entsprechenden Partners verschlüsselt, d.h. wenn `partyB` den Wert `agentI` hat, dann soll der Schlüssel `keyI` genommen werden. Falls `partyB` jedoch `agentA` (der Angreifer) ist, dann mit dem entsprechenden Schlüssel `keyA`.

Geben Sie ihr geändertes Modell als Datei `aufgabe2/protocol23.pml` ab.

Aufgabe 2.4 — 6 Punkte

Der Angreifer soll nun auch Nachrichten, die mit seinem eigenen Schlüssel verschlüsselt sind empfangen, entschlüsseln und auf den Inhalt zugreifen können. Erweitern Sie Ihr Modell entsprechend. Fügen Sie dazu Ihrem Modell zwei weitere globale Variablen (ghost variables) hinzu, die das Wissen des Angreifers bzgl. der Noncen von Beate und Ingo widerspiegeln:

```
/* Globale Ghostvariablen */
...
bool learned_nonceB, learned_nonceI;
```

Initialisieren Sie diese Variablen mit dem Wert **`false`**. In Ihrem geänderten Modell setzen Sie die Werte auf **`true`** genau dann wenn der Angreifer eine Nachricht erfolgreich entschlüsseln konnte und der Inhalt der entsprechenden Nachricht den Wert `nonceB` oder `nonceI` enthält.

Um das Modell so zu erweitern, dass der Angreifer eine an ihn geschickte Nachricht erfolgreich entschlüsseln kann, fügen Sie einen weiteren **if** Block hinzu, der den Schlüssel der empfangenen Nachricht mit dem eigenen Schlüssel des Angreifers vergleicht und falls diese übereinstimmen den Inhalt überprüft und gegebenenfalls die neuen Variablen `learned_nonceB`/`learned_nonceI` entsprechend setzt. Stimmen die Schlüssel nicht überein wird die Überprüfung übersprungen (**skip**).

Des Weiteren soll der Angreifer jetzt beim Erstellen neuer Nachrichten die Werte `nonceB` und `nonceI` verwenden dürfen, falls die entsprechende Variable `learned_nonceB` bzw. `learned_nonceI` den Wert **true** hat. Um außerdem den vorhandenen Nichtdeterminismus zu verkleinern, lassen Sie den Angreifer, wenn er den Inhalt für die dritte Nachricht des Protokolls (`msgId3`) zusammensetzt, den Wert von `content2` auf 0 setzen, da dieser für Nachrichten des Typs 3 nicht benötigt wird.

Geben Sie Ihr erweitertes Modell als Datei `aufgabe2/protocol24.pml` ab.

Aufgabe 2.5 — 16 Punkte

Jetzt wollen wir unser erweitertes Modell auf Sicherheitseigenschaften analysieren. Das Protokoll soll die folgenden drei Sicherheitsgarantien zusichern:

a) Wenn Beate das Ende des Protokolllaufs erreicht (`statusB` hat den Wert `ok`) and glaubt mit Ingo zu reden (`partyB` hat den Wert `agentI`), dann kennt der Angreifer nicht Beates Nonce (`!learned_nonceB`).

b) Analog für Ingos Nonce. Wenn Ingo das Ende des Protokolllaufs erreicht hat and glaubt mit Beate zu reden, dann kennt der Angreifer nicht Ingos Nonce.

c) Wenn Beate und Ingo am Ende Ihres Laufs angekommen sind (d.h., `statusB` wie auch `statusI` sind auf wahr gesetzt), dann muss Beates Kommunikationspartner Ingo gewesen sein und der Kommunikationspartner von Ingo dementsprechend Beate.

Formalisieren Sie die obigen Eigenschaft in LTL und versuchen Sie diese mit Spin zu verifizieren. Da das präsentierte Protokoll einen Fehler hat, sollte Ihnen Spin ein Gegenbeispiel liefern. Welche der drei Eigenschaften gelten oder gelten nicht? Erklären Sie die Situation anhand *eines* der/des Gegenbeispiele(s).

Geben Sie ihr, um die drei LTL Formeln ergänztes Modell als Datei `aufgabe2/protocol25.pml` ab, sowie das/die erzeugte(n) Gegenbeispiel(e) mit geeigneter Benennung. Weiter geben Sie bitte die geforderte Erklärung *eines* der Gegenbeispiele als Textdatei `aufgabe2/erklaerung25.txt` ab. Geben Sie in der Erklärung außerdem explizit an, welche der Eigenschaften gelten oder nicht gelten.

Hinweis:

- Achten Sie darauf, dass ein Gegenbeispiel auch wirklich eines für die Verletzung einer der LTL Formeln ist und nicht etwa „nur“ für einen ungültigen Endzustand wie einem Deadlock.