Institut für Theoretische Informatik
Logik und Formale Methoden
Prof. Dr. B. Beckert, Prof. Dr. P. H. Schmitt, D. Bruns, D. Faragó, Dr. C. Gladisch, C. Scheben, M. Ulbrich

03.12.2012

Praktikum
Formale Entwicklung objektorientierter Software
Übungsblatt 4: KeY

Auf diesem Übungsblatt sollen Ihnen Grundlagen zur Verwendung von KeY vermittelt werden. Wir haben Ihnen dazu unter Ihrem Praktikumsaccount die aktuelle KeY-Version bereitgestellt. Um KeY von Ihrem Praktikumsaccount aus zu starten verwenden Sie den Befehl `./key-master/bin/runProver`. Falls Sie von Zuhause aus arbeiten, finden sie auf der Webseite zum Praktikum einen Link zu den *Nightly builds* des KeY-Tools. Nach der Installation der Bytecode-Version kann KeY durch Aufrufen von `bin/runProver` gestartet werden. Bitte beachten Sie, dass das Praktikum auf der momentanen Entwickler-Version aufbaut. Die Beispiele sind mit dem KeY-Snapshot (Version 1.7.3797) auf Ihrem Praktikumsaccount getestet worden.

Aufgabe 15 — Prädikatenlogik

Laden Sie sich das Archiv `keyex1.tgz` von der Praktikums-Webseite herunter. Entpacken Sie die Datei mit dem Befehl:

```
tar xzvf keyex1.tgz
```

Sie erhalten im Verzeichnis `keyex1/` sieben Dateien mit Namen `p1.key` bis `p7.key`. Jede dieser Dateien enthält eine Beweisaufgabe, die Sie mit dem KeY-Beweiser lösen sollen. Dabei ist die letzte Aufgabe (`p7.key`) optional.

Starten Sie den KeY-Beweiser. Problemdateien (Dateiendung `.key`) können jetzt mit dem Menüpunkt `File | Load` geladen werden. **Zur Abgabe** speichern Sie die Aufgaben mit `File | Save` als `p1.proof` bis `p7.proof`. Lassen Sie das KeY-tool die Probleme zuerst automatisch lösen und versuchen Sie die Regelanwendungen nachzuvollziehen. Anschließend sollen die Probleme *ohne* Anwendung von "Strategien" gelöst werden, d.h. Sie sollen alle notwendigen Regeln *von Hand* anwenden. Versuchen Sie sich bei jedem Schritt darüber klar zu werden, was Sie gerade machen! Sie sollen in der Lage sein, Ihre Lösung vorzuführen.

Stellen Sie zur Lösung dieser Aufgabe zunächst sicher, dass der One-Step-Simplifier unter `Options | One Step Simplification` abgewählt ist. Sobald Sie die Aufgabenteile erfolgreich gelöst haben, versuchen Sie die Probleme `p2.key` und `p4.key` erneut mit eingeschaltetem One-Step-Simplifier zu lösen. Der One-Step-Simplifier aggregiert eine Menge von Vereinfachungsregeln und wendet diese effizient an. Ist der One-Step-Simplifier eingeschaltet, stehen diese Vereinfachungsregeln allerdings nicht mehr zur manuellen Anwendung zur Verfügung. Stattdessen gibt es eine neue Regel mit dem Namen *One Step Simplification*, welche alle Regeln des Simplifers auf einmal auf eine Formel anwendet. *Die Regel kann allerdings immer nur auf den äußersten Top-Level-Operator angewendet werden!*

Es folgen einige Hinweise zu den einzelnen Problemen:

`p1.key` Zu beweisen ist die triviale Aussage `true`. Klicken Sie auf die Formel `true`. Es erscheint ein Menü mit möglichen Regelanwendungen. `closeTrue` schließt den Ast. Es erscheint ein Dialog mit der Meldung „Proved", d.h. der Beweis ist fertig. Das geschlossene Ziel ist in der linken Hälfte des Beweiserfensters grün dargestellt.

`p2.key` Hier ist zu zeigen, dass $(p \to q) \leftrightarrow (\neg q \to \neg p)$ eine (aussagenlogische) Tautologie ist. Wenden Sie wieder Regeln an, indem Sie auf die Operatoren (`->`, `<->`, `!`) klicken. Die für Sie interessanten Regeln für die Aussagenlogik haben Namen, die sich aus dem Operator und der Seite der Sequenz zusammensetzen, also etwa `equiv_right` für eine Äquivalenz rechts vom Sequenzenpfeil und `impLeft` für eine Implikation links. Wenn Sie mit der Mauszeiger etwas länger über dem Regelnamen verweilen, wird ein "Tooltip" angezeigt, der sagt, was eine Regelanwendung machen wird. Einige der Regelanwendungen werden zu Fallunterscheidungen („Case 1/2" im linken Teil des Fensters) führen. Durch Auswahl der Ziele im linken Teil können Sie auswählen, an welchem Sie arbeiten wollen. Letztlich müssen alle geschlossen werden. Wenn eine Formel links *und* rechts vom Sequenzenpfeil vorkommt, kann ein Ast geschlossen werden. Dazu wendet man bei der rechten Formel `close` an. Insgesamt entstehen vier zu schließende Äste.

`p3.key` Es geht um die prädikatenlogische Tautologie $(\exists x.\forall y.p(x,y)) \to (\forall v.\exists u.p(u,v))$. Überlegen Sie sich zunächst, warum diese Aussage gilt, und wie man sie beweisen könnte. Zerlegen Sie wieder die Formel, indem Sie bei den äußeren Operatoren anfangen. Wenn Sie einen $\exists$-Quantor links bzw. einen $\forall$-Quantor rechts abarbeiten (mit der Regel `exLeft` bzw. `allRight`), wird eine *Skolemkonstante* eingeführt, deren Name vom Namen der quantifizierten Variablen abgeleitet ist (z.B. x_0 für eine Quantifiziertung über x). Um einen $\forall$-Quantor links bzw. einen $\exists$-Quantor rechts mit einem Term t zu instanziieren, bietet die GUI mehrere Möglichkeiten:

- Man kann auf die quantifizierte Formel die Regel `allLeft` bzw. `exRight` anwenden. In dem sich öffnenden Instanziierungsdialog muss man den gewünschten Term t eingeben. Falls der Term bereits in der Sequenz vorkommt, kann man ihn auch per Drag&Drop in die Eingabemaske ziehen.
- Wenn der Term t bereits vorkommt, ist eine weitere Möglichkeit die Regel `instAll` bzw. `instEx`. Dazu klickt man *nicht* auf die quantifizierte Formel, sondern auf den Term t.
- Wenn der Term t bereits vorkommt, kann man als dritte Möglichkeit den Term einfach per Drag&Drop auf die zu instanziierende Variable ziehen.

`p4.key` Hier ist zu zeigen, dass $a = b \ \wedge \ b = c \to a = c$ eine Tautologie ist, wobei a, b, c irgendwelche Konstanten sind. Zerlegen Sie die Formel wieder mit `impRight` und `andLeft`. Sie können nun Gleichungen auf Terme anwenden. Um etwa die Gleichung `a=b` irgendwo anzuwenden, gibt es mehrere Möglichkeiten:

- Wählen Sie die Regel `make_insert_eq` auf dieser Gleichung. Es existiert nun eine Regel `insert_eq`, die für jedes Vorkommen des Terms `a` angeboten wird, und die diesen in `b` überführt.
- Alternativ können Sie auf ein Vorkommnis des Terms `a` klicken, und dort die Regel `applyEq` anwenden. Eventuell öffnet sich ein Instanziierungsdialog, in dem Sie `a=b` als die anzuwendende Gleichung auswählen müssen.
- Als dritte Möglichkeit können Sie die Gleichung `a=b` per Drag&Drop auf ein Vorkommnis des Terms `a` ziehen, und in dem sich öffnenden Kontextmenü die Regel `applyEq` auswählen.

Probieren Sie es aus! Wenn Sie eine Gleichung in der anderen Richtung, also von rechts nach links, anwenden wollen, können Sie sie mit `eqSymm` umdrehen, und dann wieder nach einer der beiden beschriebenen Möglichkeiten vorgehen.

`p5.key` Das Problem lautet $((\forall x.f(g(x)) = g(x)) \wedge (g(a) = b)) \to (f(b) = g(a))$. Es kann mit den bisher vorgestellten Techniken gelöst werden. Überlegen Sie sich vorher, welche

Instanz der all-quantifizierten Formel sie benötigen, und besorgen Sie sich diese mit einer der oben erwähnten Möglichkeiten, einen Allquantor zu instanziieren.

`p6.key` Überlegen sie sich, weshalb folgendes eine Tautologie ist

$$\forall x.\forall y.\forall z.(((P(x,y) \land P(y,z)) \rightarrow P(x,z)) \land \neg P(x,x)) \rightarrow \forall x.\forall y.(P(x,y) \rightarrow \neg P(y,x))$$

und beweisen Sie es danach im KeY-Beweiser.

`p7.key` **(Optional)**
Lesen Sie sich folgendes Rätsel aus dem Lufthansa-Magazin 11/2002 durch:

> Genau zwei der Personen A, B, C, D und E lügen. Welche?
> A: "B lügt dann und nur dann, wenn D die Wahrheit sagt!"
> B: "Sollte C ein wahrheitsliebender Mensch sein, so ist entweder A oder D ein Lügner."
> C: "E kann man auf keinen Fall trauen, und auch A oder B hält bzw. halten es mit der Wahrheit nicht so genau!"
> D: "Würde B die Wahrheit sagen, dann könnte man A oder C vertrauen!"
> E: "Unter den Personen A, C und D befindet sich mindestens ein Lügner!"

Überlegen Sie sich die Lösung des Problems und formalisieren Sie die Aufgabenstellung und Ihre Lösung als logische Formel. Beweisen Sie diese dann mit Hilfe von KeY. Sie können hierzu die Strategie *Java DL* verwenden!

Hinweis: Die zu beweisende Formel geben Sie im `\problem` Abschnitt von `p7.key` an. Die anderen Abschnitte in `p7.key` dienen der Deklaration der benötigten Signatur.

Aufgabe 16 — Dynamische Logik

Laden Sie sich das Archiv `keyex2.tgz` von der Praktikums-Webseite herunter, und entpacken Sie es mit dem Befehl:

```
tar xzvf keyex2.tgz
```

Wie schon in Aufgabe 16 erhalten Sie sieben Dateien mit Namen `p1.key` bis `p7.key`, von denen jede eine Beweisaufgabe enthält, die Sie mit dem KeY-Beweiser lösen sollen. Hierbei ist in dieser Aufgabe der Beweis zu `p5.key` optional. Verfahren Sie zum Laden, Beweisen und Abgeben der Probleme wie in Aufgabe 16 beschrieben. Benutzen Sie insbesondere die automatische Beweissuche (Strategien) nur dann, wenn es in den untenstehenden Erläuterungen explizit angegeben ist.

Stellen Sie vor dem Beweisen sicher, dass unter Options — Default Taclet Options — intRules die Einstellung intRules:arithmeticSemanticsIgnoringOF ausgewählt ist. Mit dieser Einstellung ignoriert der KeY-Beweiser die Möglichkeit von Integer-Überläufen, was uns hier im Praktikum das Arbeiten erleichtert.

Falls Sie bei einem Beweis nicht weiter kommen sollten und den One-Step-Simplifier aktiviert haben, prüfen Sie ob dieser nicht vielleicht irgendwo anwendbar ist!

`p1.key` Hier soll gezeigt werden, dass der Wert einer Integer-Variablen `i` nach Ausführung der Anweisung `i++` um eins höher ist als vorher.

Dazu arbeitet man schrittweise die Programmformeln ab ("symbolische Ausführung"). Befindet sich der Mauszeiger über einer Programmformel, wird die jeweils nächste zu behandelnde Anweisung grau unterlegt. Die Regel zur symbolischen Ausführung dieser Anweisung ist normalerweise die oberste aus der Liste der anwendbaren Regeln, etwa `postincrement`, `cast`, `variableDeclaration` etc.

Nach Abarbeitung des Programms bleibt unter anderem ein Term mit dem Funktionssymbol `javaAddInt` übrig. Dieses Funktionssymbol kann (aufgrund der ausgewählten Behandlung von Integerüberläufen) mit Hilfe der Regel `translateJavaAdd` in eine mathematische Addition auf $\mathbb{Z}$ umgewandelt werden.

`p2.key` Hier sollen Sie zeigen, dass nach Ausführung des Programmstücks mit den lokalen Variablen `i` und `k`:

```
if (k==0) {
  i=k;
}
else {
  i=0;
}
```

`i` den Wert 0 hat. Führen Sie wieder das Programm symbolisch aus. Sie werden Fallunterscheidungen machen, die Sie beim "intuitiven Überprüfen" des Programms auch machen würden (z.B. ob `k` den Wert 0 hat oder nicht). Denken Sie daran, dass eine Sequenz und eine Implikation auch dann wahr werden, wenn Sie deren linke Seite zu *false* vereinfachen können.

`p3.key` Bevor Sie die Problemdatei in den KeY-Beweiser laden, öffnen Sie die Datei in einen Editor Ihrer Wahl und betrachten Sie die DL-Formel, die innerhalb von `problem{ ... }` steht. Sie enthält folgendes Java-Programm in einem Diamond:

```
int i=0;
try {
   throw e;
   i=i+1;
} catch (RuntimeException e1) {
   i=i+4;
} finally {
   i=i+8;
}
```

Hierbei ist `e` ein Objekt (nicht mit `null` belegt) vom Typ `IllegalArgumentException`. `IllegalArgumentException` ist eine Unterklasse von `RuntimeException`. Ersetzen Sie in der hinter dem Diamond stehenden Teilformel `i = XXX` das `XXX` durch eine solche Zahl, dass die DL-Formel allgemeingültig ist. Falls Ihnen die Java-Semantik bezüglich der Behandlung von Exceptions nicht mehr geläufig ist, schlagen Sie sie in der Java Language Specification[1] nach.

Laden Sie dann die so veränderte Problemdatei in den KeY-Beweiser und weisen Sie die Allgemeingültigkeit nach.

(a) Machen Sie zu Beginn jede Regelanwendung ohne Zuhilfename der Strategien, bis einschließlich der Anwendung von `tryCatchFinallyThrow`. Beschreiben Sie den Effekt dieser Regel.

(b) Führen Sie den Rest des Beweises, indem Sie die Strategie `Java DL` einschalten und auf den Knopf ▷ klicken. Der Beweis läuft dann automatisch ab. Im Proof-Tab können Sie nachvollziehen, welche Regeln angewandt wurden.

[1] `http://java.sun.com/docs/books/jls/second_edition/html/jTOC.doc.html`

`p4.key` Zu dieser Aufgabe gehört die Java-Klasse `MyClass.java`, die Sie im Unterverzeichnis `classes` finden. Laden Sie erneut die .key-Datei in einen Editor und überlegen Sie sich, durch was Sie `XXX` ersetzen müssen, damit die Eingabeformel allgemeingültig ist. Natürlich sollte nicht schon die Teilformel hinter dem Diamond allein eine Tautologie sein. Beweisen Sie die Allgemeingültigkeit der veränderten Formel mit KeY.

`p5.key` **(Optional)**
Nun sollen Sie sich um das Ergebnis des folgenden Programmstücks kümmern:

```
i=i0;
if (((j=i+=i++)+ ++i)== i=++i)
  i=i++;
```

Überlegen Sie sich, wie die Ausdrücke ausgewertet werden. Ersetzen Sie danach die beiden `XXX` durch die richtigen Ausdrücke, die nur aus `+`, `i0` und Zahlenliteralen bestehen dürfen. Beweisen Sie die veränderte Eingabedatei. Sie können dazu die `Java DL`-Strategie verwenden.

`p6.key` Die folgende Regel (`loopUnwind`)

$$\frac{\Gamma \Longrightarrow \langle \texttt{if (b) \{l1 : \{l2 : \{}\alpha'\texttt{\} while (b)\{}\alpha\texttt{\}\}\}}\rangle\Phi,\ \Delta}{\Gamma \Longrightarrow \langle \texttt{while (b)\{}\alpha\texttt{\}}\rangle\Phi,\ \Delta}$$

führt genau einen Schleifendurchlauf aus. Dabei stimmt α' mit α überein, außer dass jedes in α vorkommende `break;` durch ein `break l1;` und jedes `continue;` durch ein `break l2;` ersetzt ist.

(a) Laden Sie das Problem in Ihren Editor. Im Abschnitt `problem` finden Sie ein Java-Programm. Überlegen Sie sich wieder den Wert der Variablen `i` nach Ausführung des Programms und tragen Sie ihn anstelle von `XXX` ein.

(b) Laden Sie Ihre modifizierte Beweisereingabe `p6.key` in den Beweiser. Stellen Sie die Anzahl der maximalen automatischen Regelanwendungen im Proof Search Strategy-Tab auf mindestens 100 und wählen Sie wie üblich die Strategie `Java DL` aus. Stellen Sie unter Java DL Options — Loop treatment den Wert None ein. Mit dieser Einstellung stoppt die Strategie, wenn die symbolische Ausführung eine Schleife erreicht hat. Lassen Sie die Strategie laufen.

(c) In der resultierenden Situation kann die Regel `loopUnwind` ausgeführt werden. Schauen Sie sich die Sequenz vor Ausführung der Regel und diejenige danach an. Erklären Sie die Funktionsweise der `loopUnwind` Regel.

(d) Führen Sie den Beweis mithilfe der eingestellten Strategie zu Ende.

`p7.key` Betrachten Sie erneut die Klasse `MyClass.java` und ihre Unterklassen und überlegen Sie sich, was die Methode `p` zurückliefert. Ersetzen Sie `XXX` durch den Rückgabewert einer `MyClass2`-Instanz beim Aufruf ihrer `p`-Methode. Beweisen Sie wie zuvor Ihre Behauptung unter Zuhilfenahme der Strategien.

Aufgabe 17 — Spezifizieren und Verifizieren: Amount

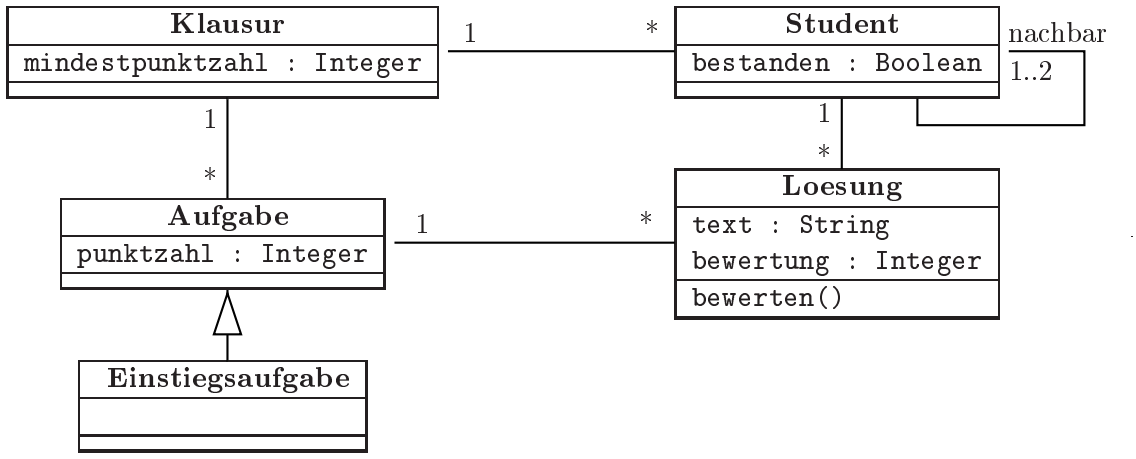
Ergänzen Sie Ihre Spezifikationen für die Datei `Amount.java` von Aufgabenblatt 3 zu einer vollständigen funktionalen JML-Spezifikation. Unter vollständig funktional ist dabei zu verstehen, dass die Verträge der Methoden das Verhalten der Implementierung vollkommen beschreibt, also dass die Implementierung keine Wahlmöglichkeiten mehr besitzt.

Geben Sie für jede Methode und den Konstruktor je eine oder mehrere `requires`-, `ensures`- und `assignable`-Klauseln an. Beweisen Sie mit KeY, dass die Implementierung Ihrer Verträge korrekt ist.

Hinweis: Der JML-Dialekt von KeY weicht ein wenig vom Standard ab. Insbesondere muss man explizit angeben, wenn Methodenargumente oder -ergebnisse ihre Invariante erfüllen. Verwenden Sie das Prädikat `\invariant_for(p)`, um zu spezifizieren, dass die Objektinvariante von `p` gilt.

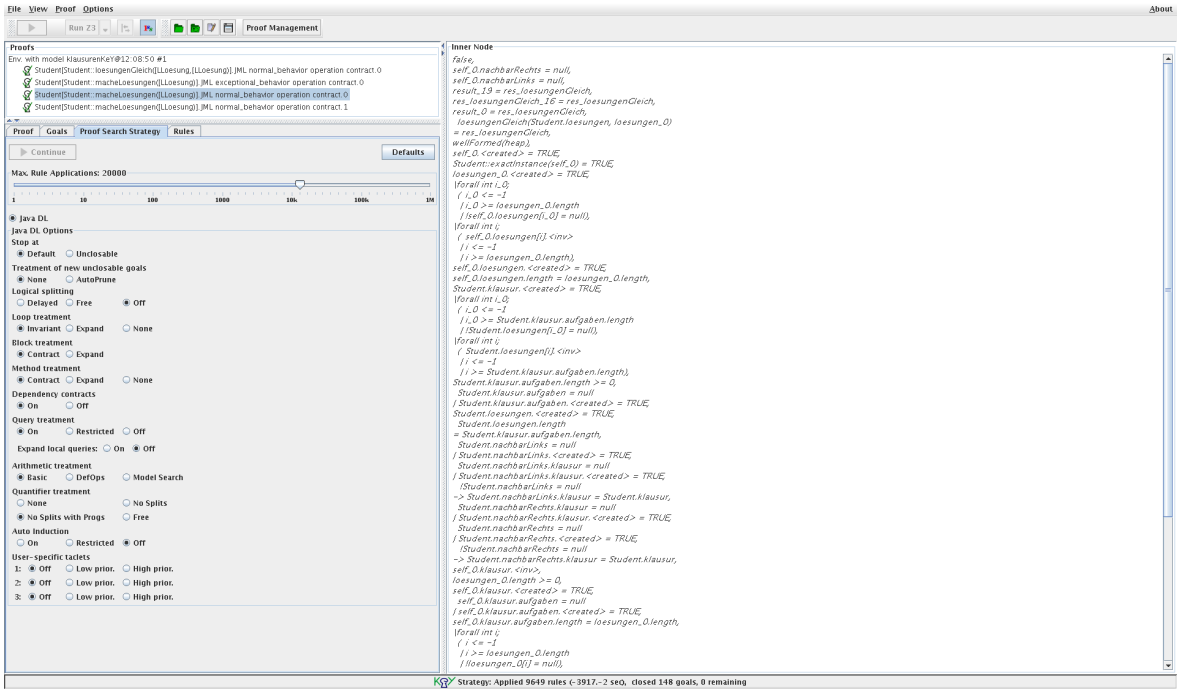
Aufgabe 18 — Spezifizieren und Verifizieren: Klausuren

Gegeben sei folgendes bereits in Aufgabe 3 auf Übungsblatt 1 vorgestellte Programm:



Eine bereits mit JML-Spezifikationen versehene Version dieses Programms ist in der Datei `Klausuren2.java` auf der Praktikumswebseite verfügbar.

(a) Beweisen Sie die Korrektheit der Methodenspezifikation von `Student.macheLoesungen` (*JML normal_behavior operation contract [id: 2 / Student::macheLoesungen]*). Hierzu müssen Sie die Invariante, Variante und Assignable-Klausel der in `macheLoesungen` enthaltenen Schleife spezifizieren. Stellen Sie sicher, dass folgende Strategieoptionen eingestellt sind:



Falls Sie eine korrekte und hinreichend starke Schleifeninvariante spezifiziert haben, sollte der Beweis mit den angegeben Strategieoptionen automatisch durchlaufen.

(b) Spezifizieren Sie außerdem

- einen `exceptional_behavior` Vertrag, der beschreibt, wann eine `IllegalArgumentException` geworfen wird und das es für diesen Fall nur zu einer Exception dieses Typs kommen kann.
- einen `normal_behavior` Vertrag, der den Fall abdeckt, daß die Lösungen mit denen eines Nachbarn übereinstimmen.

Beweisen Sie auch diese beiden Verträge mit den in (a) angegeben Strategieoptionen.

Abgabe bis Dienstag 18.12

Abgabe (als Java-, ASCII- oder PDF-Dateien, oder tar-Archiv solcher) per E-Mail an Christoph Gladisch und Christoph Scheben. Es braucht pro Gruppe und Aufgabe nur *eine* Lösung abgegeben werden. Bitte dokumentieren Sie Ihre Lösungen ausreichend und seien Sie darauf vorbereitet, sie auf Nachfrage zu erklären.

Praktikums-Webseite: `http://formal.iti.kit.edu/teaching/keypraktWS1213/`

Daniel Bruns:	R. 223,	Tel. 608-45268,	E-Mail: `bruns@kit.edu`
David Faragó:	R. 308,	Tel. 608-47322,	E-Mail: `farago@ira.uka.de`
Christoph Gladisch:	R. 223,	Tel. 608-45268,	E-Mail: `gladisch@ira.uka.de`
Christoph Scheben:	R. 106,	Tel. 608-44338,	E-Mail: `scheben@ira.uka.de`
Mattias Ulbrich:	R. 106,	Tel. 608-44338,	E-Mail: `mulbrich@ira.uka.de`