Prof. Dr. A. Poetzsch-Heffter
Peter Zeller, M. Sc.

TU Kaiserslautern
Fachbereich Informatik
AG Softwaretechnik

Übungsblatt 6: Logik (SS 2017)

Abgabe: Freitag, 26. Mai, 15:30
Abgabekästen neben Raum 34-401.7 (bei AG Softwaretechnik)
Bitte geben Sie zu dritt ab.

Bitte beachten Sie die Hinweise zu den **Ersatzübungen** für die ausfallenden Übungen am 25.05. (Christi Himmelfahrt) auf der Vorlesungs-Homepage!

Aufgabe 1 Gentzen-Sequentzkalkül

Zeigen Sie die folgenden Aussagen im Gentzen-Sequentzkalkül. Notieren Sie die Beweise wie in der Vorlesung bottom-up und baumartig. Notieren Sie in jedem Schritt, welche Regel angewandt wurde.

a) $\vdash_G (p \to q) \to \Big((p \to \neg q) \to \neg p\Big)$

b) $p \to q, r \vee \neg q \vdash_G q \vee \neg p$

c) $\vdash_G \Big(p \vee (q \wedge r)\Big) \to \Big((p \vee q) \wedge (p \vee r)\Big)$

Aufgabe 2 Vollständigkeit des Gentzen-Sequentzkalküls

Wir betrachten hier der Einfachheit halber nur Formeln mit den Operatoren $\neg$ und $\vee$.

Seien $A_1, \ldots, A_n, B_1, \ldots, B_m \in F_{\{\neg,\vee\}}$ und es gelte $\models \neg A_1 \vee \ldots \neg A_n \vee B_1 \vee \cdots \vee B_m$. Zeigen Sie, dass dann die Sequenz $A_1, \ldots, A_n \vdash_G B_1, \ldots B_m$ im Sequentzkalkül herleitbar ist.

Verwenden Sie dazu Induktion über die Gesamtzahl der Operatoren ($\neg$ und $\vee$) in $A_1, \ldots, A_n, B_1, \ldots, B_m$.

Aufgabe 3 Tableaux (Anwendung)

a) Prüfen Sie mit der Tableaux-Methode, ob folgende Formel eine Tautologie ist:
$\Big((p \to q) \wedge (p \to \neg q)\Big) \vee p$

b) Prüfen Sie mit der Tableaux-Methode, ob folgende Formel erfüllbar ist:
$\Big(\Big(r \to \neg(q \vee \neg p)\Big) \wedge p\Big) \wedge \Big((\neg q \vee r) \wedge (q \vee \neg p)\Big)$

c) Prüfen Sie mit der Tableaux-Methode, ob folgende Formel erfüllbar ist:
$(\neg b \to f) \wedge \Big(\Big((b \wedge f) \to \neg e\Big) \wedge \Big((e \wedge \neg b) \to \neg f\Big)\Big)$

Aufgabe 4 Implementierung der Tableaux-Methode

Geben Sie diese Aufgabe bis Freitag, 2. Juni 15:30 ab. Schicken Sie dazu Ihr gesamtes Projekt als Zip-Archiv per Email an Ihren Tutor. Diese Aufgabe wird mit der doppelten Punktzahl (8 mögliche Punkte) bewertet.

Laden Sie sich für diese Aufgabe die Datei `tableaux_java.zip` herunter, welche eine Vorlage für diese Aufgabe enthält. Die Vorlage ist ein Projekt, welches das Build-System Gradle verwendet. Den Java-Code finden Sie unter `src/main/java/logic`. Der Code für die JUnit Tests liegt unter `src/test/java/logic`.

Einrichtung Melden Sie sich bei Problemen mit der Einrichtung bei Peter Zeller (Gebäude 34, Raum 407 oder `p_zeller@cs.uni-kl.de`).

1. Installieren Sie das Java JDK Version 1.8 (nicht das JRE). Wenn Sie `javac -version` auf der Konsole ausführen, sollte die Version angezeigt werden.
2. Öffnen Sie eine Konsole und wechseln Sie in das Projekt-Verzeichnis. In diesem Ordner liegen die Dateien `gradlew` und `gradlew.bat`. Mit diesen Dateien wird Gradle ausgeführt. Führen Sie `./gradlew compileJava` auf der Konsole aus, um den `compileJava` Task auszuführen (unter Windows statt dessen `.\gradlew compileJava`). Danach sollten Sie `BUILD SUCCESSFUL` auf der Konsole sehen.

   Wenn Sie den Fehler "`Could not find tools.jar`" erhalten, dann verwendet Gradle die JRE statt das JDK. Überprüfen Sie dann die Einstellung der Umgebungsvariable `PATH`. Wenn Sie den Fehler "`Could not find method compileOnly()`" erhalten, verwenden Sie wahrscheinlich eine alte Version von Gradle. Verwenden Sie statt dessen den Gradle-Wrapper `gradlew`, welcher sich bereits im Projekt befindet.
3. Andere Gradle Tasks, die Sie zum Entwickeln benötigen sind:
   - `./gradlew test`
     führt die JUnit Tests aus. Das Ergebnis wird in der Datei `./build/reports/tests/index.html` gespeichert.
   - `./gradlew test --tests *and`
     führt nur den Test mit Namen `and` aus.
   - `./gradlew run -q`
     Führt die `Main`-Prozedur des Programms aus.

Vorlage

- Die Vorlage enthält Klassen, welche Formeln repräsentieren. Diese Klassen befinden sich im Ordner `ast`. Eine `Formula` ist entweder eine `Variable`, eine `Negation` oder eine `BinaryFormula`. Eine `BinaryFormula` hat eine linke und rechte Formel und ist entweder vom Typ `And`, `Or`, oder `Implication`.
- Die Vorlage enthält einen Parser für Formeln, so dass ein String wie `"(p | q) & (q -> !(p & q))"` in die entsprechende Formel umgewandelt werden kann. Die verwendeten Operatoren sind `|` für $\vee$, `&` für $\wedge$, `->` für $\rightarrow$ und `!` für $\neg$. Der Parser ist in den Dateien `formula.cup` und `formula.flex` definiert. Aus diesen Dateien wird beim Ausführen von Gradle Java-Code im Ordner `src-generated` generiert.
- Die Klasse TableauxMethod ist eine Vorlage für Ihre Implementierung.
- Die Klasse `Main` enthält eine `main`-Prozedur, welche Formeln von der Standardeingabe liest und auf diese Ihre Tableaux-Methode anwendet.

Aufgabe

Implementieren Sie die Prozedur **`public static`** `Map<Variable, Boolean> findSat(Formula f)` in der Klasse `TableauxMethod`. Diese Methode nimmt eine Formel und soll mit der Tableaux-Methode prüfen, ob die Formel erfüllbar ist. Falls die Formel unerfüllbar ist, soll die Methode **`null`** zurückgeben. Wenn die Formel erfüllbar ist soll die Methode eine Variablenbelegung zurückgeben, welche die Formel erfüllt. Eine Variablenbelegung wird hier durch eine `Map<Variable, Boolean>` dargestellt.

Sie können die Tableaux-Methode auf beliebige Art implementieren. Die folgenden Lösungshinweise führen zu einer von vielen möglichen Implementierungen und müssen nicht befolgt werden.

Lösungshinweise

a) Definieren Sie ein `enum` mit den verschiedenen Arten von Formeln:

```
enum FormulaKind {
    alpha, beta, doubleNegation, literal
}
```

Schreiben Sie dann eine Methode `FormulaKind kind(Formula f)`, welche zu einer Formel die Art der Formel berechnet. Wenn die Formel `f` eine α-Formel ist, dann soll zum Beispiel `FormulaKind.alpha` zurückgegeben werden.

Sie können dazu zum Beispiel **`instanceof`** und Casts verwenden oder Methoden in den verschiedenen Ast-Klassen implementieren.

b) Als nächsten Schritt implementieren Sie eine Methode `SplitResult split(Formula f)`, welche eine Formel aufteilt (Eine α-Formel in α_1 und α_2, eine β-Formel in β_1 und β_2).

Um die zwei Ergebnis-Formeln zurückgeben zu können, definieren Sie eine Klasse `SplitResult`, in der zwei Formeln gespeichert werden können.

c) Im Tableaux-Algorithmus wollen wir einen Ast durch eine Liste von Formeln darstellen. Dazu können wir folgende einfach verkettete Liste definieren:

```
class FormulaList {
    final FormulaList rest;
    final Formula formula;

    public FormulaList(FormulaList rest, Formula formula) {
        this.rest = rest;
        this.formula = formula;
    }
}
```

Außerdem können die folgenden Methoden definiert werden, um Listen zu erstellen:

```
    // Creates a list with only one formula
    static FormulaList list(Formula f) {
        return new FormulaList(null, f);
    }

    // Creates a new list with formula f and l as the rest of the list
    static FormulaList plus(FormulaList l, Formula f) {
        return new FormulaList(l, f);
    }
```

Schreiben Sie eine Methode **`boolean`** `containsNegation(FormulaList list, Formula f)`, welche prüft ob die gegebene Liste die Negation der Formel `f` enthält, oder ob `f` die Negation einer Formel in der Liste ist.

d) Schreiben Sie eine Methode `Map<Variable, Boolean> getVariableAssignment(FormulaList branch)`, welche aus einem offenen Ast eine Variablenbelegung berechnet, indem sie alle Literale im Ast betrachtet.

e) Zur Implementierung der Methode `findSat` definieren wir eine Hilfsmethode:
`Map<Variable, Boolean> findSatH(FormulaList branch, FormulaList worklist)`
Diese nimmt als ersten Parameter den aktuellen Ast und als zweiten Parameter eine Liste der Formeln, die im aktuellen Ast noch abgearbeitet werden müssen. In der Original-Methode wird dann einfach die Hilfsmethode mit der Startformel aufgerufen:

```
public static Map<Variable, Boolean> findSat(Formula f) {
  return findSatH(list(f), list(f));
}
```

Die Methode `findSatH` soll dann wie folgt rekursiv arbeiten:

1. Wenn die `worklist` leer ist, ist der Ast abgearbeitet und offen. Dann wird die entsprechende Bewertung für den Ast zurückgegeben.
2. Wenn die `worklist` nicht leer ist, nehme eine Formel `f` aus der `worklist` heraus.
   - Wenn `f` eine α-Formel ist, prüfe ob durch Hinzufügen von α_1 und α_2 der Ast abgeschlossen wird. Falls ja, gebe **`null`** zurück. Falls nein, rufe `findSatH` auf mit α_1 und α_2 im Ast und in der Worklist.
   - Wenn `f` eine Doppelnegation ist, kann ähnlich wie im Fall der α-Formel vorgegangen werden.
   - Wenn `f` eine β-Formel ist, behandle β_1 und β_2 getrennt:

     Wenn der Ast durch β_1 nicht abgeschlossen wird, rufe `findSatH` rekursiv auf mit β_1 im Ast und in der Worklist. Wenn das Ergebnis des rekursiven Aufrufs eine erfüllende Variablenbelegung liefert, gebe diese zurück. Ansonsten fahre analog zu β_1 mit β_2 fort. Wenn β_1 und β_2 beide keine Variablenbelegung liefern, gib **`null`** zurück.
   - Wenn `f` ein Literal ist, fahre mit der nächsten Formel aus der Worklist fort.