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

TU Kaiserslautern
Fachbereich Informatik
AG Softwaretechnik

Sheet 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.

Aufgabe 1 Gentzen Sequent Calculus

Prove the following statements using the Gentzen sequent calculus. Construct the proofs bottom up and denote them in a tree-like structure. Give the rule or axiom used for each step.

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 Completeness of the Gentzen Sequent Calculus

For simplicity we only consider formulas with operators $\neg$ and $\vee$.

Let $A_1, \ldots, A_n, B_1, \ldots, B_m \in F_{\{\neg,\vee\}}$ with $\models \neg A_1 \vee \ldots \neg A_n \vee B_1 \vee \cdots \vee B_m$. Show that this implies that the sequent $A_1, \ldots, A_n \vdash_G B_1, \ldots B_m$ is derivable in the Gentzen sequent calculus.

Use induction over the overall number of operators ($\neg$ und $\vee$) in $A_1, \ldots, A_n, B_1, \ldots, B_m$.

Note: As a special case of the statement above we get: if $\models B_1$ then $\vdash_G B_1$ is derivable (for $B_1 \in F_{\{\neg,\vee\}}$)

Aufgabe 3 Tableaux (Anwendung)

a) Use the tableau method to check whether the following formula is a tautology:
$\Big((p \to q) \wedge (p \to \neg q)\Big) \vee p$

b) Use the tableau method to check, whether the following formula is satisfiable:
$\Big(\Big(r \to \neg(q \vee \neg p)\Big) \wedge p\Big) \wedge \Big((\neg q \vee r) \wedge (q \vee \neg p)\Big)$

c) Use the tableau method to check, whether the following formula is satisfiable:
$(\neg b \to f) \wedge \Big(\Big((b \wedge f) \to \neg e\Big) \wedge \Big((e \wedge \neg b) \to \neg f\Big)\Big)$

Aufgabe 4 Implementierung der Tableaux-Methode

Submit this task until Friday, June 2nd, 15:30. To do this, send your complete project as a zip file to your tutor (or `p_zeller@cs.uni-kl.de`*). This task is worth double the normal points: up to 8 points are possible for this task.*

Download the template `tableaux_java.zip` for this task. The template is a project using the build system Gradle. The Java-Code is located under `src/main/java/logic` and the JUnit tests are in `src/test/java/logic`.

Setup If you have problems with the setup contact Peter Zeller (room 34-407 or `p_zeller@cs.uni-kl.de`).

1. Make sure that you have the Java JDK version 1.8 (not the JRE) installed. Running `javac -version` on the command line gives you the version number.
2. On the command line, change to the directory with the extracted files. In this folder you should see a file `gradlew` and a file named `gradlew.bat`. These scripts will download the build tool Gradle and can be used to build the project.

   First run `./gradlew compileJava` from the command line to run the `compileJava` task of the project (on Windows Systems execute `.\gradlew compileJava` instead). At the end you should see `BUILD SUCCESSFUL` in the output. If you are seeing an error like "Could not find tools.jar" it is most likely means that you have the JRE installed instead of the JDK.
3. Besides `compileJava` there are some other useful tasks:

   - `./gradlew test`
     runs the JUnit tests. The test reports are stored in `./build/reports/tests/index.html`.
   - `./gradlew test --tests *and`
     only execute tasks with name "`and`".
   - `./gradlew run -q`
     Runs the `main` procedure of the program.

Template

- The template contains classes representing formulas. These classes are located in package `ast`. A `Formula` is either a `Variable`, a `Negation`, or a `BinaryFormula`. A `BinaryFormula` has a left and right subformula and is of Type `And`, `Or`, or `Implication`.
- The template contains a parser for formulas, so that a String like `"(p | q) & (q -> !(p & q))"` can be transformed to the respective class instances. The operators used are: `|` for $\vee$, `&` for $\wedge$, `->` for $\rightarrow$ and `!` for $\neg$. The parser is defined in the files `formula.cup` and `formula.flex`. When running Gradle, Java code is generated from these files and stored folder `src-generated`.
- The class `TableauxMethod` is the template where you have to add your implementation.
- The class `Main` contains a `main`-procedure, which reads formulas from standard input and passes them to your implementation of the tableau method.

Your Task

Implement the procedure `public static Map<Variable, Boolean> findSat(Formula f)` in class `TableauxMethod`. This method shall use the tableau method to check whether the given formula is satisfiable. If the formula is unsatisfiable the method shall return `null`. Otherwise it shall return a satisfying variable assignment for the formula. We represent variable assignments as a `Map<Variable, Boolean>`.

You can choose how you want to implement the tableau method. The hints below yield one of many possible solutions, but you do not have to follow them.

The hints are not translated, contact Peter Zeller if you need the translated hints to solve this task.

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.