Deduktive Systeme in Isabelle

1 Installation

Auf den Tux-Rechnern des SCI ist Isabelle bereits für Sie installiert. Sie können dort Isabelle einfach mit dem Befehl `isabelle jedit` vom Terminal aus starten. Für die Installation auf eigenen Rechnern finden Sie Downloads für die gängigen Betriebssysteme unter `https://isabelle.in.tum.de/`.

Windows Der Download ist eine `exe`-Datei. Führen Sie die Datei aus, um Isabelle an einem beliebigen Ort zu entpacken. Nach dem Entpacken sollte Isabelle automatisch starten. Ansonsten können Sie auch die ausführbare Datei `Isabelle2016-1.exe` im entpackten Ordner ausführen.

Linux Der Download für Linux ist eine Zip-Datei, welche einfach entpackt werden kann. Im Ordner `bin` befindet sich die ausführbare Datei `isabelle`, welche mit dem Befehl `./isabelle jedit` gestartet werden kann.

2 Bedienung

Nach dem Starten öffnet sich der Editor JEdit. Beim ersten Starten prüft Isabelle die Beweise in der Standardbibliothek, was einige Minuten dauern kann. Anschließend sollte sich der Editor wie in Abbildung 1 öffnen. Oben befindet sich der Code-Editor, unten befindet sich das *Output*-Fenster, in dem Fehlermeldungen und der aktuelle Zustand von Beweisen angezeigt werden. Zum Anzeigen des Beweis-Zustands muss die Option *Proof state* aktiviert werden. Der Inhalt des *Output*-Fensters bezieht sich immer auf die aktuelle Position des Cursors im Code oben.

Eingabe von Sonderzeichen Isabelle unterstützt eine Reihe von Symbolen, welche nicht direkt über eine normale Tastatur eingegeben werden können. Die unterstützten Symbole lassen sich über das *Symbols*-Fenster (siehe Abbildung 2 auswählen. Zu jedem Symbol gibt es außerdem eine oder mehrere Abkürzungen, welche angezeigt werden, wenn man mit der Maus über das entsprechende Symbol fährt. Zum Beispiel lässt sich der Implikationspfeil ($\longrightarrow$) als `-->` und die Negation ($\neg$) als `~` schreiben.

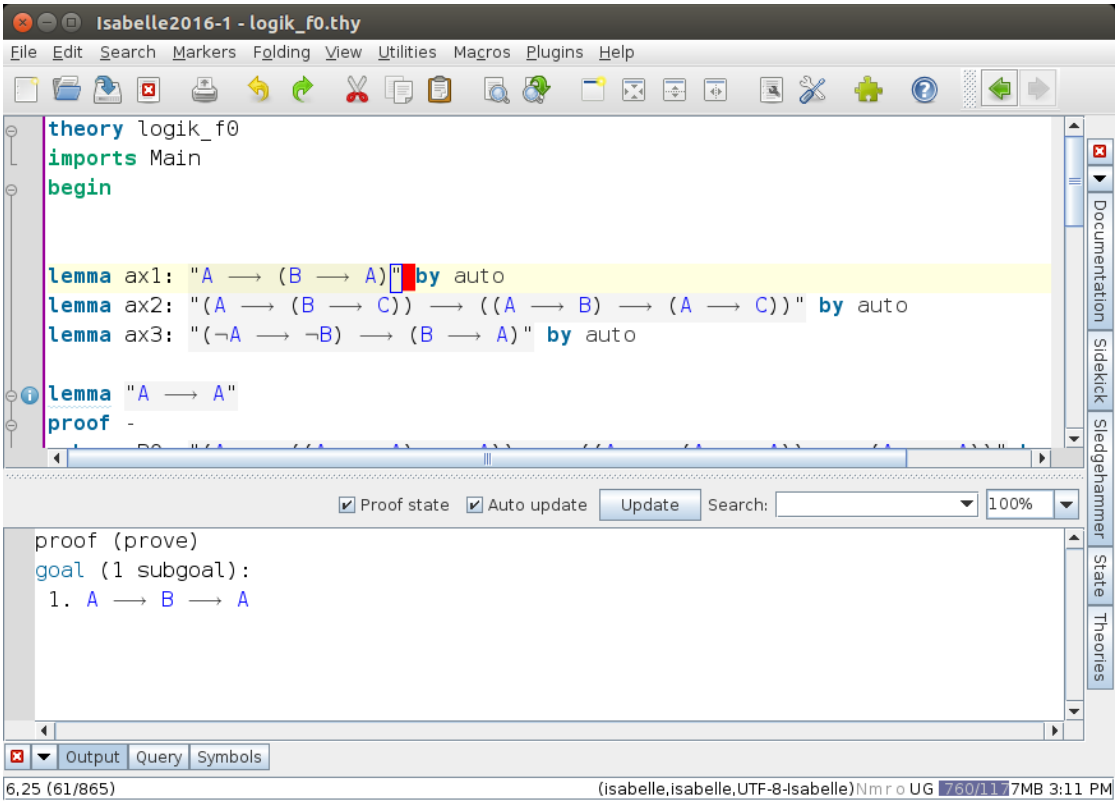

Abbildung 1: Der Isabelle JEdit Editor.

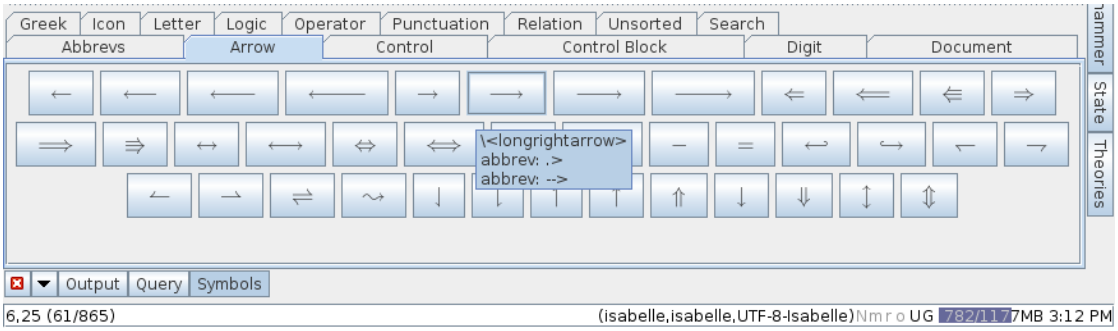

Abbildung 2: Symbole in Isabelle.

Anpassen der Schriftgröße Die Standard-Schriftgröße ist für niedrige Auflösungen relativ groß. Sie kann unter `Utilities` → `Global Options` unter `Text Area` → `Text font` angepasst werden.

3 Grundlagen

Erstellen Sie eine neue Datei und speichern Sie diese unter `logik_f0.thy` ab. Geben Sie dann den folgenden Text in die Datei ein:

```
1 theory logik_f0
2 imports Main
3 begin
4
5
6 end
```

Dies ist der grundlegende Aufbau einer Isabelle Datei. In Zeile 1 steht der Name der Theorie, welche genau mit dem Dateinamen (ohne `.thy`) übereinstimmen muss. In Zeile 2 wird die Theorie `Main` importiert, welche wir verwenden werden. Der Code unserer Theorie wird dann zwischen `begin` und `end` stehen.

3.1 Ein erster Beweis

Wir wollen nun einen ersten Beweis aus dem deduktiven System $\mathcal{F}_0$ mit Isabelle beweisen. Da die Axiome von $\mathcal{F}_0$ nicht in Isabelle eingebaut sind, müssen wir diese erst einführen:

```
lemma ax1: "A ⟶ (B ⟶ A)" by auto
lemma ax2: "(A ⟶ (B ⟶ C)) ⟶ ((A ⟶ B) ⟶ (A ⟶ C))" by auto
lemma ax3: "(¬A ⟶ ¬B) ⟶ (B ⟶ A)" by auto
```

Wir haben die 3 Axiome hier als Lemmata bewiesen. Ein Lemma startet mit dem Schlüsselwort `lemma`. Danach kann ein Name für das Lemma folgen und anschließend folgt die Formel, die bewiesen werden soll. Formeln stehen in Isabelle immer in Anführungszeichen. Nach der Formel folgt der Beweis, wobei wir für die Axiome einfach einen automatischen Beweis mit der Anweisung `by auto` gewählt haben.

Beweis von `A ⟶ A`: Wir wollen nun den ersten $\mathcal{F}_0$-Beweis aus der Vorlesung nach Isabelle übertragen. Wir beginnen mit dem Aufschreiben der Aussage als `lemma` unter den bereits definierten Axiomen und starten dann den Beweis mit der Anweisung `proof -`.

```
lemma beispiel1: "A ⟶ A"
proof -
```

Im *Output*-Fenster wird nun das aktuelle *Goal* `A ⟶ A` angezeigt. Der Strich nach `proof` ist hier wichtig und bedeutet, dass keine Regel verwendet wird um den Beweis zu starten. Das *Goal* ist also identisch mit der Aussage des Lemmas.

Der Beweis lässt sich dann wie folgt aufschreiben:

```
lemma beispiel1: "A ⟶ A"
proof -
  have B0: "(A ⟶ ((A ⟶ A) ⟶ A)) ⟶ ((A ⟶ (A ⟶ A)) ⟶ (A ⟶ A))" by (rule ax2)
  have B1: "A ⟶ ((A ⟶ A) ⟶ A)" by (rule ax1)
  have B2: "(A ⟶ (A ⟶ A)) ⟶ (A ⟶ A)" using B0 B1 by (rule mp)
  have B3: "A ⟶ (A ⟶ A)" by (rule ax1)
  show B4: "A ⟶ A" using B2 B3 by (rule mp)
qed
```

Hierbei startet jeder Zwischenschritt mit `have` und der letzte Schritt, welcher dem *Goal* entspricht mit `show`. Ein einzelner Schritt ist wie folgt aufgebaut:

- Zuerst kommt das Schlüsselwort `have` (bzw. `show` für den letzten Schritt).
- Dann folgt ein Name für den Schritt gefolgt von einem Doppelpunkt, zum Beispiel `B0:`.
- Anschließend folgt die Formel in Anführungszeichen, welche in diesem Schritt gezeigt wird.
- Nach der Formel folgt der Beweis für die Formel. Wir betrachten hier nur zwei Arten von Beweisen:
  - Wenn die Formel einem Axiom entspricht, kann Sie mit `by (rule ax1)` (bzw. `by (rule ax2)` oder `by (rule ax3)`) bewiesen werden.
  - Wenn die Formel sich mit Modus-Ponens ergibt kann sie mit `using A B by (rule mp)` bewiesen werden. Hierbei sind `A` und `B` Namen von Formeln, die in vorherigen Schritten bewiesen worden. Die Reihenfolge ist hier wichtig: Die erste Formel muss von der Form `B ⟶ C` sein, wobei `B` die zweite Formel und `C` die Formel im aktuellen Schritt ist.

Am Ende wird der Beweis mit `qed` abgeschlossen.

3.2 Verwenden von Annahmen

Es ist auch möglich einen Beweis mit zusätzlichen Annahmen zu führen, wie zum Beispiel $\neg B, B \vdash_{\mathcal{F}_0} A$. Dazu werden die zusätzlichen Annahmen beim Start des Beweises mit `assumes` angegeben und die zu zeigende Formel mit `shows`. Die Annahmen könne wie andere Formeln auch mit Namen versehen werden, wie das folgende Beispiel zeigt:

```
lemma widerspruch:
assumes A1: "¬B"
assumes A2: "B"
shows "A"
proof -
  have B1: "¬B ⟶ (¬A ⟶ ¬B)" by (rule ax1)
  have B2: "¬A ⟶ ¬B" using B1 A1 by (rule mp)
  have B3: "(¬A ⟶ ¬B) ⟶ (B ⟶ A)" by (rule ax3)
  have B4: "B ⟶ A" using B3 B2 by (rule mp)
  show B5: "A" using B4 A2 by (rule mp)
qed
```

3.3 Abgekürzte Beweise

Ist ein Lemma bereits bewiesen, kann es wie ein Axiom oder eine Regel in anderen Beweisen verwendet werden. Dies wird dann abgekürzter Beweis genannt. Zum Beispiel verwendet der folgende Beweis für `B2` das oben bewiesene Lemma mit dem Namen `beispiel1`.

```
lemma abgek: "((A ⟶ B) ⟶ A) ⟶ ((A ⟶ B) ⟶ B)"
proof -
  have B1: "((A ⟶ B) ⟶ (A ⟶ B))
              ⟶ (((A ⟶ B) ⟶ A) ⟶ ((A ⟶ B) ⟶ B))" by (rule ax2)
  have B2: "(A ⟶ B) ⟶ (A ⟶ B)" by (rule beispiel1)
  show B3: "((A ⟶ B) ⟶ A) ⟶ ((A ⟶ B) ⟶ B)" using B1 B2 by (rule mp)
qed
```

Wenn das verwendete Lemma Annahmen hat, wie zum Beispiel das Lemma `widerspruch` oben, dann müssen diese wie bei der Modus-Ponens Regel mit **`using`** angegeben werden. Die Reihenfolge ist auch hier wichtig und muss der Reihenfolge in der Formulierung des Lemmas entsprechen, zum Beispiel:

```
lemma abgek2: "A ⟶ (A ⟶ ¬A) ⟶ B"
proof (rule impI)+
  assume B1: "A"
  assume B2: "A ⟶ ¬A"
  have B3: "¬A" using B2 B1 by (rule mp)
  show B4: "B" using B3 B1 by (rule widerspruch)
qed
```

3.4 Verwendung des Deduktionstheorems

Das Deduktionstheorem besagt, dass $\Sigma \vdash_{\mathcal{F}_0} A \longrightarrow B$ genau dann gilt, wenn $\Sigma, A \vdash_{\mathcal{F}_0} B$ gilt. Als Anwendungsbeispiel betrachten wir den Beweis von $\neg\neg A \longrightarrow A$ aus der Vorlesung. Das Deduktionstheorem kann beim Start des Beweises angewendet werden. Dazu starten wir den Beweis mit **`proof (rule impI)`** statt mit **`proof -`**. Der Unterschied zeigt sich im *Output*-Fenster: Statt dem *Goal* $\neg\neg A \longrightarrow A$ wird dort nun $\neg\neg A \Longrightarrow A$ angezeigt. Der Pfeil $\Longrightarrow$ entspricht in Isabelle dem Symbol $\vdash$ aus der Vorlesung.

Für den Beweis in Isabelle bedeutet das, dass die Formeln links von $\Longrightarrow$ angenommen werden können (mit **`assume`**) und die Formel rechts mit **`show`** gezeigt werden muss. Die **`assume`**-Anweisung ist wie **`have`** aufgebaut, allerdings ohne Beweis. Wird bei **`assume`** eine falsche Formel angegeben, so schlägt der **`show`** Befehl später mit der Fehlermeldungen "Failed to refine any pending goal" fehl.

Für das Beispiel aus der Vorlesung lässt sich das Deduktionstheorem wie folgt verwenden:

```
lemma beispiel2: "¬¬A ⟶ A"
proof (rule impI)
  assume B1: "¬¬A"
  have B2: "¬¬A ⟶ (¬¬¬¬A ⟶ ¬¬A)" by (rule ax1)
  have B3: "¬¬¬¬A ⟶ ¬¬A" using B2 B1 by (rule mp)
```

```
  have B4: "(¬¬¬¬A ⟶ ¬¬A) ⟶ (¬A ⟶ ¬¬¬A)" by (rule ax3)
  have B5: "¬A ⟶ ¬¬¬A" using B4 B3 by (rule mp)
  have B6: "(¬A ⟶ ¬¬¬A) ⟶ (¬¬A ⟶ A)" by (rule ax3)
  have B7: "¬¬A ⟶ A" using B6 B5 by (rule mp)
  show B8: "A" using B7 B1 by (rule mp)
qed
```

Es ist auch möglich, das Deduktionstheorem mehrmals zu verwenden. Dazu können einfach mehrere Anwendungen zu Beginn des Beweises durch Kommata getrennt angegeben werden, wie folgendes Beispiel zeigt:

```
lemma "A ⟶ (A ⟶ B) ⟶ B"
proof (rule impI, rule impI)
  assume B1: "A"
  assume B2: "A ⟶ B"
  show B3: "B" using B2 B1 by (rule mp)
qed
```

Alternativ kann hier auch die Notation `proof (rule impI)+` verwendet werden, welche das Deduktionstheorem so oft wie möglich anwendet. In beiden Fällen ist das *Goal* im Beispiel `A ⟹ A ⟶ B ⟹ B` [1].

Das Deduktionstheorem kann auch in die umgekehrte Richtung angewendet werden. Dazu lässt sich die Regel `rev_mp` verwenden, bei der man mit `[OF A1]` angeben kann, dass Formel `A1` von den Annahmen verwendet werden soll. Zum Beispiel:

```
lemma widerspruch2:
assumes A1: "¬B"
assumes A2: "B"
shows "A"
proof (rule rev_mp[OF A1], rule rev_mp[OF A2])
  ...
  show "B ⟶ (¬ B ⟶ A)"  ...
qed
```

4 Weiterführendes Material

Zum Einstieg eignet sich das Buch "Concrete Semantics" von Tobias Nipkow (Download unter `http://www.concrete-semantics.org/`). Der erste Teil des Buches gibt eine Einführung in Isabelle/Hol, der zweite Teil zeigt wie Semantik von Programmiersprachen in Isabelle modelliert werden können.

Die offizielle Dokumentation zu Isabelle befindet sich unter `https://isabelle.in.tum.de`. "The Isabelle/Isar Reference Manual" bietet eine umfassende Referenz für alle Befehle und die Syntax von Isabelle.

[1] Mehrere Annahmen auf der linken Seite werden auch wieder durch `⟹` getrennt. Unter `Plugins` → `Plugin Options` → `Isabelle` → `General` kann als `Print Mode` der Modus `brackets` eingetragen werden, welcher näher an der Notation der Vorlesung ist und das *Goal* als `⟦A; A ⟶ B⟧ ⟹ B` anzeigt. Die Option erfordert einen Neustart von Isabelle.