



Einführung in die Programmierung

Prof. Dr. Bertrand Meyer

Vorlesung 5: Invarianten und Logik



In Verbindung mit einem Feature:

- Vorbedingungen
- Nachbedingungen

In Verbindung mit einer Klasse:

- Klasseninvariante

Zusicherungen

```
remove_all_segments
```

```
-- Alle Stationen ausser der südlichen Endstation entfernen.
```

```
ensure
```

```
nur_eine_bleibt: count = 1
```

```
beide_enden_gleich: south_end = north_end
```

Zusicherungen

```
extend(s: STATION)
```

```
-- s am Ende der Linie hinzufügen.
```

```
ensure
```

```
neue_station_da: i_th(count) = s
```

```
im_norden: north_end = s
```

```
eine_mehr: count = old count + 1
```

```
deposit (v: INTEGER)
    -- Addiere v zum Kontostand
require
    positiv:  $v > 0$ 
do
    ...
ensure
    addiert: balance = old balance + v
end
```

Zusicherung



Die Invariante drückt Konsistenzbedingungen aus, die zwischen Abfragen in der Klasse erfüllt sein müssen.

invariant

sued_ist_erste: *south_end = i_th(1)*

nord_ist_letzte: *north_end = i_th(count)*



1. Korrekte Software
2. Dokumentation der Software, im Speziellen Dokumentation der Programmierschnittstelle.
3. Testen & Fehlerbeseitigung

(Später noch mehr!)

Laufzeiteffekt: Einstellung im Compiler (siehe Projects -> Settings in EiffelStudio)

Verträge in anderen Sprachen



Java: Java Modeling Language (JML), iContract etc.

C#: Spec# (Microsoft Research Erweiterung)

UML: Object Constraint Language

Python

C++: Nana

etc.



Programmieren heisst logisch denken.

Logik ist die Wissenschaft des logischen Denkens.

Wir benutzen Logik tagtäglich.

*"Sokrates ist ein Mensch.
Alle Menschen sind sterblich.*

Daher muss Sokrates sterblich sein."



Logik ist die Grundlage von:

- Mathematik: Beweise sind nur gültig, falls sie den Regeln der Logik genügen.
- Softwareentwicklung:
 - Bedingungen in Verträgen:
" x darf nicht null sein, so dass wir $\frac{x+7}{x}$ berechnen können."
 - Bedingungen in Programmen:
"Falls i positiv ist, führe diese Instruktion aus."
(Mehr dazu in einer späteren Lektion)



Eine Bedingung wird durch einen **Boole'schen Ausdruck** ausgedrückt.

Ein solcher besteht aus

- **Boole'schen Variablen** (Bezeichner, die Boole'sche Werte bezeichnen)
- **Boole'schen Operatoren** (**not**, **or**, **and**, **=**, **implies**)

Und repräsentiert mögliche

- **Boole'sche Werte** (Wahrheitswerte, entweder **True** oder **False**)



Beispiele von Boole'schen Ausdrücken:
(mit *rain_today* und *cuckoo_sang_last_night* als
Boole'sche Variablen):

➤ *rain_today*

(eine Boole'sche Variable ist ein Boole'scher
Ausdruck)

➤ *not rain_today*

➤ *(not cuckoo_sang_last_night) implies rain_today*

(Durch Klammern bildet man Unterausdrücke.)

Die Negation (not)



a	not a
True	False
False	True

Für jeden Boole'schen Ausdruck e und alle Werte von Variablen gilt:

- Entweder e oder **not e** hat den Wahrheitswert **True**.
- Entweder e oder **not e** hat den Wahrheitswert **False**.
(Prinzip des ausgeschlossenen Dritten)
- e und **not e** können nicht beide den Wahrheitswert **True** haben.
(Satz des Widerspruchs)

Die Disjunktion (or)



<i>a</i>	<i>b</i>	<i>a or b</i>
True	True	True
True	False	True
False	True	True
False	False	False

Der **or** - Operator ist **nicht-exklusiv**

Der **or** - Operator ist **kommutativ**

Disjunktionsprinzip:

- Eine **or**-Disjunktion hat den Wahrheitswert **True**, ausser beide Operanden haben den Wert **False**.

Die Konjunktion (and)



<i>a</i>	<i>b</i>	<i>a and b</i>
True	True	True
True	False	False
False	True	False
False	False	False

Der **and**-Operator ist **kommutativ**.

Dualität von **and** und **or**:

- $(a \text{ and } b) = \text{not}(\text{not } a \text{ or } \text{not } b)$
- $(a \text{ or } b) = \text{not}(\text{not } a \text{ and } \text{not } b)$

Konjunktionsprinzip:

- Eine **and**-Konjunktion hat den Wahrheitswert **False**, ausser beide Operanden haben den Wert **True**.



Auch komplexere Boole'sche Ausdrücke sind möglich.

Beispiele:

$a \text{ and } (b \text{ and } (\text{not } c))$

$\text{not } (\text{not } (\text{not } (\text{not } (\text{not } a))))$



Belegungen und Wahrheitstabellen

Eine **Belegung** für eine Menge von Variablen: eine bestimmte Wahl von Wahrheitswerten (**True** oder **False**) für jede Variable.

Eine Belegung erfüllt einen Ausdruck, falls der Wahrheitswert des Ausdrucks **True** ist.

Eine Wahrheitstabelle für einen Ausdruck mit n Variablen hat

- $n + 1$ Spalten
- 2^n Zeilen

Wahrheitstabelle für die Grundoperationen



<i>a</i>	<i>b</i>	not <i>a</i>	<i>a</i> or <i>b</i>	<i>a</i> and <i>b</i>
True	True	False	True	True
True	False		True	False
False	True	True	True	False
False	False		False	False

Tautologie: Ein Boole'scher Ausdruck, der für jede mögliche Belegung den Wahrheitswert **True** hat.

Beispiele:

- $a \text{ or } (\text{not } a)$
- $\text{not } (a \text{ and } (\text{not } a))$
- $(a \text{ and } b) \text{ or } ((\text{not } a) \text{ or } (\text{not } b))$



Widerspruch: Ein Boole'scher Ausdruck, der für alle möglichen Belegungen den Wahrheitswert **False** hat.

Beispiele:

- $a \text{ and } (\text{not } a)$

Erfüllbarer Ausdruck: Ein Ausdruck ist erfüllbar, sofern er für mindestens eine Belegung den Wahrheitswert **True** hat.

- Jede Tautologie ist erfüllbar.
- Jeder Widerspruch ist unerfüllbar.

Äquivalenz (=)



a	b	$a = b$
True	True	True
True	False	False
False	True	False
False	False	True

Der $=$ Operator ist kommutativ.

($a = b$ hat denselben Wert wie $b = a$)

Der $=$ Operator ist reflexiv.

($a = a$ ist eine Tautologie für jedes a)

Substitution:

- Für alle Ausdrücke u , v und e gilt: Falls $u = v$ eine Tautologie ist und e' der Ausdruck ist, den man erhält, wenn man in e jedes Vorkommen von u durch v ersetzt, dann ist $e = e'$ eine Tautologie.



De Morgan'sche Gesetze: Tautologien

- $(\text{not } (a \text{ or } b)) = ((\text{not } a) \text{ and } (\text{not } b))$
- $(\text{not } (a \text{ and } b)) = ((\text{not } a) \text{ or } (\text{not } b))$

Weitere Tautologien (Distributivität):

- $(a \text{ and } (b \text{ or } c)) = ((a \text{ and } b) \text{ or } (a \text{ and } c))$
- $(a \text{ or } (b \text{ and } c)) = ((a \text{ or } b) \text{ and } (a \text{ or } c))$



Vorrangregeln (höchster Vorrang zuerst): **not**, **and**, **or**, **implies** (wird später vorgestellt), =

and und **or** sind **assoziativ**:

- $a \text{ and } (b \text{ and } c) = (a \text{ and } b) \text{ and } c$
- $a \text{ or } (b \text{ or } c) = (a \text{ or } b) \text{ or } c$

Stilregeln:

Wenn Sie einen Boole'schen Ausdruck schreiben, können Sie folgende Klammern weglassen:

- Die Klammern auf beiden Seiten des "=", falls der gesamte Ausdruck eine Äquivalenz ist.
- Die Klammern um aufeinanderfolgende elementare Terme, falls sie durch den gleichen assoziativen Operator getrennt sind.

Die Implikation (implies)



a	b	a implies b
True	True	True
True	False	False
False	True	True
False	False	True

Für jedes a, b gilt: a implies $b = (\text{not } a) \text{ or } b$

In a implies b ist a der **Vordersatz**, b der **Nachsatz**.

Implikationsprinzip:

- Eine Implikation hat den Wahrheitswert **True**, ausser der Vordersatz hat den Wert **True** und der Nachsatz hat den Wert **False**.
- Zudem: Immer **True** falls der Vordersatz **False** ist.



implies hat in natürlichen Sprachen oft die Bedeutung von Kausalität (Wenn... dann...).

- *"Wenn das Wetter schön ist, gehen wir baden."*
- *"Wenn du dieses Getränk ins Handgepäck nimmst, lassen sie dich nicht ins Flugzeug."*

Ein häufiges Missverständnis über Implikationen

Immer wenn a **False** ist, ergibt a **implies** b **True**, unabhängig von b :

- "Falls heute Mittwoch ist, ist $2+2=5$."
- "Falls $2+2=5$, ist heute Mittwoch."

Beide der obigen Implikationen ergeben **True**.

Die Fälle, in denen der Vordersatz **False** ist, sagen nichts über den Wahrheitswert des Nachsatzes aus.

Die Umkehrung der Implikation (1)



Im Allgemeinen gilt folgendes **nicht**:

~~$a \text{ implies } b = (\text{not } a) \text{ implies } (\text{not } b)$~~

Ein (falsches!) Beispiel:

- *"Alle Zürcher, die am See wohnen, sind reich. Ich wohne nicht am See, also bin ich nicht reich."*

$live_near_lake \text{ implies } rich$ [1]

$(\text{not } live_near_lake) \text{ implies } (\text{not } rich)$ [2]

Die Umkehrung der Implikation (2)



Korrekt:

$$a \text{ implies } b = (\text{not } b) \text{ implies } (\text{not } a)$$

Beispiel:

- "Alle Leute, die am See wohnen, sind reich. Alice ist nicht reich, also kann sie nicht in Küsnacht wohnen."

$$\textit{live_near_lake} \text{ implies } \textit{rich} = \\ (\text{not } \textit{rich}) \text{ implies } (\text{not } \textit{live_near_lake})$$

Semi-strikte Boole'sche Operatoren (1)



Ein Beispielausdruck (x ist eine ganze Zahl):

$$\frac{x + 7}{x} > 1$$

False für $x < 0$

Undefiniert für $x = 0$



ABER:

➤ Division durch null: x darf nicht 0 sein.

$$(x \neq 0) \text{ and } (((x + 7) / x) > 1)$$

False für $x < 0$

False für $x = 0$



ABER:

- Unser Programm würde während der Auswertung der Division abstürzen.

Wir brauchen eine nicht-kommutative Version von and (und or):

Semi-strikte Boole'sche Operatoren

Semi-strikte Operatoren (and then, or else)



a **and then** b ergibt dasselbe wie a **and** b falls a und b definiert sind, und ergibt immer **False** wenn a den Wert **False** hat.

a **or else** b ergibt dasselbe wie a **or** b falls a und b definiert sind, und ergibt immer **True** wenn a den Wert **True** hat.

$(x \neq 0)$ **and then** $((x + 7) / x) > 1$

Semi-strikte Operatoren ermöglichen es uns, eine Auswertungsreihenfolge zu definieren (von links nach rechts)

Wichtig für Programmierer, da undefinierte Objekte zu Programmabstürzen führen können!

Normale vs. Semi-strikte Boole'sche Operatoren

Benutzen Sie...

- normale boole'sche Operatoren (**and** und **or**), falls Sie garantieren können, dass beide Operanden definiert sind.
- **and then**, falls eine Bedingung nur dann Sinn ergibt, wenn eine andere wahr ist.
- **or else**, falls eine Bedingung nur dann Sinn ergibt, wenn eine andere falsch ist.

Beispiel:

- "Falls Sie nicht ledig sind, muss Ihr Ehepartner den Vertrag unterschreiben."

is_single or else spouse_must_sign



Beispiel:

- "Falls Sie nicht ledig sind, muss Ihr Ehepartner den Vertrag unterschreiben."

(not is_single) implies spouse_must_sign

Definition von **implies**: in unserem Fall **immer semi-strikt!**

- *a implies b = (not a) or else b*



Schlüsselwort in Eiffel	Mathematisches Symbol
not	\sim oder \neg
or	\vee
and	\wedge
=	\Leftrightarrow
implies	\Rightarrow



Aussagenkalkül:

Eigenschaft p gilt für ein einziges Objekt

Prädikatenkalkül:

Eigenschaft p gilt für mehrere Objekte

Ein allgemeineres or



G : eine Gruppe von Objekten, p : eine Eigenschaft

or: Ist p für mindestens ein Objekt in G erfüllt?

Kann man an mindestens einer Haltestelle der Linie 8 auf eine andere Linie umsteigen?

Station_Balard.is_exchange or

Station_Lourmel.is_exchange or

Station_Boucicaut.is_exchange or

... (alle Stationen der Linie 8)

Der Existenzquantor: *exists* oder \exists

$\exists s: Stations_8 \mid s.is_exchange$

"Es gibt eine Haltestelle s in *Stations_8* so dass *s.is_exchange* wahr ist."

Ein allgemeineres and



and: Ist p für jedes Objekt in G erfüllt?

Sind alle Haltestellen der Linie 8 Haltestellen, an denen man umsteigen kann?

*Station_Balard.is_exchange and
Station_Lourmel.is_exchange and
Station_Boucicaut.is_exchange and ...*
(alle Stationen der Linie 8)

Der Allquantor: *for_all* oder \forall

$\forall s: Stations_8 \mid s.is_exchange$

"Für alle s in $Stations_8$ gilt $s.is_exchange$."



Ein Boole'scher Ausdruck:

$\exists s: \text{SOME_SET} \mid s.\text{some_property}$

- *True* genau dann, wenn mindestens ein Element von *SOME_SET* die Eigenschaft *some_property* erfüllt.

Beweise:

- *True*: Finden Sie ein Element in *SOME_SET*, welches die Eigenschaft erfüllt.
- *False*: Beweisen Sie, dass kein Element von *SOME_SET* die Eigenschaft erfüllt. (Sie müssen also alle Elemente überprüfen.)



Ein Boole'scher Ausdruck:

$\forall s: \text{SOME_SET} \mid s.\text{some_property}$

- *True* genau dann, wenn jedes Element von *SOME_SET* *some_property* erfüllt.

Beweise:

- *True*: Beweisen Sie, dass jedes Element von *SOME_SET* die Eigenschaft erfüllt. (Sie müssen also alle Elemente überprüfen.)
- *False*: Finden Sie ein Element von *SOME_SET*, welches die Eigenschaft nicht erfüllt.



Die Verallgemeinerung des De Morgan'schen Gesetzes:

$$\text{not } (\exists s : \text{SOME_SET} \mid P) = \forall s : \text{SOME_SET} \mid \text{not } P$$

$$\text{not } (\forall s : \text{SOME_SET} \mid P) = \exists s : \text{SOME_SET} \mid \text{not } P$$



$\exists s: \text{SOME_SET} \mid \text{some_property}$

Falls *SOME_SET* leer ist: immer **False**

$\forall s: \text{SOME_SET} \mid \text{some_property}$

Falls *SOME_SET* leer ist: immer **True**

Lesen Sie auf nächste Woche...



... das Kapitel 6 (object creation)

Beginnen Sie mit dem Lesen des Kapitels 7 (control structures)



- Die Logik als Werkzeug des logischen Denkens
- Boole'sche Operationen und ihre Wahrheitstabellen
- Eigenschaften von Boole'schen Operatoren: Benutzen Sie keine Wahrheitstabellen!
- Das Prädikatenkalkül: Logische Aussagen über Mengen
- Semi-strikte Boole'sche Operatoren