ETH Zürich

Chair of Software Engineering

Einführung in die Programmierung

Prof. Dr. Bertrand Meyer

Vorlesung 5: Invarianten und Logik

Erinnerung: Verträge

In Verbindung mit einem Feature:

- Vorbedingungen
- Nachbedingungen

In Verbindung mit einer Klasse:

- Klasseninvariante

Verträge

```
remove_all_segments
        -- Alle Stationen ausser der ersten entfernen.
    ensure
        nur_eine_bleibt: count = 1
        beide_enden_gleich: first = last
```

Zusicherungen

```
append (s: STATION)
        -- s am Ende der Linie hinzufügen.
    ensure
        neue_station_ist_letzte: last = s
        eine_mehr: count = old count + 1
```

Zusicherungen

Verträge

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

Zusicherung

Klasseninvarianten

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

```
invariant
    anzahl_positiv: count > 0
    definition_von_first: first = i_th (1)
    definition_von_last: last = i_th (count)
```

Anwendungen von Verträgen

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# (Erweiterung durch Microsoft Research)

UML: Object Constraint Language

Python

C++: Nana

etc.

Logik

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."

Logisches Denken und Programmieren

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)

Boole'sche Ausdrücke

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

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*

  (Mittels 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)

a	*b*	*a* or *b*
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)

a	*b*	*a* **and** *b*
True	**True**	**True**
True	**False**	**False**
False	**True**	**False**
False	**False**	**False**

Der **and**-Operator ist kommutativ.

Dualität von **and** und **or**:

- (*a* **and** *b*) = **not**((**not** *a*) **or** (**not** *b*))
- (*a* **or** *b*) = **not**((**not** *a*) **and** (**not** *b*))

Konjunktionsprinzip:

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

Komplexere Ausdrücke

Auch komplexere Boole'sche Ausdrücke sind möglich.

Beispiele:

> *a* **and** (*b* **and** (**not** *c*))
>
> **not** (**not** (**not** (**not** (**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

a	*b*	not *a*	*a* or *b*	*a* and *b*
True	True	False	True	True
True	False		True	False
False	True	True	True	False
False	False		False	False

Tautologien

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

Beispiele:

- *a* **or** (**not** *a*)
- **not** (*a* **and** (**not** *a*))
- (*a* **and** *b*) **or** ((**not** *a*) **or** (**not** *b*))

Widersprüche und Erfüllbarkeit

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

Beispiele:

- *a* **and** (**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

De Morgan'sche Gesetze: Tautologien

- (**not** (*a* **or** *b*)) = ((**not** *a*) **and** (**not** *b*))
- (**not** (*a* **and** *b*)) = ((**not** *a*) **or** (**not** *b*))

Weitere Tautologien (Distributivität):

- (*a* **and** (*b* **or** *c*)) = ((*a* **and** *b*) **or** (*a* **and** *c*))
- (*a* **or** (*b* **and** *c*)) = ((*a* **or** *b*) **and** (*a* **or** *c*))

Syntaxkonvention und Vorrangregeln

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

and und **or** sind assoziativ:

- *a* **and** (*b* **and** *c*) = (*a* **and** *b*) **and** *c*
- *a* **or** (*b* **or** *c*) = (*a* **or** *b*) **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* **= (not** *a***) 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.

Implikationen in natürlichen Sprachen

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* **implies** *b* = (**not** *a*) **implies** (**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 **implies** *rich* [1]

(**not** *live_near_lake*) **implies** (**not** *rich*) [2]

Die Umkehrung der Implikation (2)

Korrekt:

a **implies** *b* = (**not** *b*) **implies** (**not** *a*)

Beispiel:

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

live_near_lake **implies** *rich* =
(**not** *rich*) **implies** (**not** *live_near_lake*)

Implikation

Semi-strikte Boole'sche Operatoren (1)

Ein Beispielausdruck (x ist eine ganze Zahl):

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

False für $x < 0$

Undefiniert für $x = 0$

Semi-strikte Boole'sche Operatoren (2)

ABER:

- Division durch Null: x darf nicht 0 sein.

$$(x \mathrel{/=} 0) \textbf{ and } (((x + 7) / x) > 1)$$

False für $x < 0$

False für $x = 0$

Semi-strikte Boole'sche Operatoren (3)

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.

$(x$ /= $0)$ **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*

Semi-strikte Implikation

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*

Eiffel-Notation für Boole’sche Operatoren

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

Aussagen- und Prädikatenkalkül

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?

> *Haldenbach.is_exchange* **or**
> *ETH_Universitaetsspital.is_exchange* **or**
> *Haldenegg.is_exchange* **or**
> … (alle Stationen der Linie 10)

Der Existenzquantor: *exists* oder ∃

> ∃ *s* : *Line10.stations* | *s.is_exchange*

> "Es gibt eine Haltestelle *s* in *Line10.stations* 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?

Haldenbach.is_exchange **and**
ETH_Universitatetsspital.is_exchange **and**
Haldenegg.is_exchange **and** …

(alle Stationen der Linie 10)

Der Allquantor: *for_all* oder ∀

∀ *s: Line10.stations* | *s.is_exchange*

"Für alle *s* in *Line10.stations* gilt *s.is_exchange*."

Ausdrücke mit dem Existenzquantor

Ein Boole'scher Ausdruck:

$$\exists\, s : EINE\_MENGE \mid s.eine\_eigenschaft$$

- *True* genau dann, wenn mindestens ein Element von *EINE_MENGE* die Eigenschaft *eine_eigenschaft* erfüllt.

Beweise:

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

Ausdrücke mit dem Allquantor

Ein Boole'scher Ausdruck:

$\forall$ *s: EINE_MENGE* | *s.eine_eigenschaft*

- *True* genau dann, wenn jedes Element von *EINE_MENGE* *eine_eigenschaft* erfüllt.

Beweise:

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

Dualität

Die Verallgemeinerung des De Morgan'schen Gesetzes:

not $(\exists\ s : EINE\_MENGE \mid P) = \forall\ s : EINE\_MENGE \mid$ **not** P

not $(\forall\ s : EINE\_MENGE \mid P) = \exists\ s : EINE\_MENGE \mid$ **not** P

Leere Mengen

$\exists s : EINE\_MENGE \mid eine\_eigenschaft$

Falls *EINE_MENGE* leer ist: immer False

$\forall s : EINE\_MENGE \mid eine\_eigenschaft$

Falls *EINE_MENGE* leer ist: immer True

Was wir heute gesehen haben:

- 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

Lesen Sie auf nächste Woche...

Kapitel 1 bis 6

Lesen Sie im Speziellen das Kapitel 5 (Logik), da wir nur kurz auf den in "*Diskrete Mathematik*" behandelten Teil eigegangen sind und uns auf die Anwendungen in der Programmierung konzentriert haben.