Aussagenlogik

Dorothee Henke,
Leonie Mädje,
Christian Schneider

2.1 Syntax der Aussagenlogik

- Aussagen werden durch Formeln repräsentiert → wahr oder falsch
- Bestandteile der Formeln
  - Konstanten „true“ und „false“
  - atomare Aussagen (Atome)
    - entsprechen Variablen in Algebra
    - keine Analyse der internen Struktur
  - logische Verknüpfungen (not, and, or, …)

Darstellung in OCaml

Formeln in OCaml → Datentyp „formula“

- „true“ und „false“
- atomare Formeln
- Operatoren „not“, „and“, „or“, „imp“, „iff“
- Menge der Atome
  - unendlich groß
  - Typ der Atome ist Parameter der Definition des Typs formula

Darstellung in OCaml

```
type ('a)formula = False
                 | True
                 | Atom of 'a
                 | Not of ('a)formula
                 | And of ('a)formula * ('a)formula
                 | Or of ('a)formula * ('a)formula
                 | Imp of ('a)formula * ('a)formula
                 | Iff of ('a)formula * ('a)formula
                 | Forall of string * ('a)formula
                 | Exists of string * ('a)formula;;
```

Konkrete Syntax

- Bool: algebraische Symbole (+ und *)
- Vorteil: $p(q+r) = pq + pr$
- Nachteil: $p + qr = (p+q)(p+r)$
- heute: Standardsymbole für Verknüpfungen

Konkrete Syntax

English	Symbolic	ASCII	OCaml
false	$\bot$	`false`	`False`
true	$\top$	`true`	`True`
not p	$\neg p$	`~p`	`Not p`
p and q	$p \wedge q$	`p /\ q`	`And(p,q)`
p or q	$p \vee q$	`p \/ q`	`Or(p,q)`
p implies q	$p \Rightarrow q$	`p ==> q`	`Imp(p,q)`
p iff q	$p \Leftrightarrow q$	`p <=> q`	`Iff(p,q)`

Prioritätsregeln:

- $p \Rightarrow q \wedge \neg r \vee s$
- $p \wedge q \wedge r, \quad p \Rightarrow q \Rightarrow r$

Konkrete Syntax

- p, q und r: Formeln
- x, y und z: Atome
- Funktionen parse_formula und print_qformula

Primitive Aussagen

- fixiere Typ für primitive Aussagen → Aussagenformeln, indiziert durch Namen

```
type prop = P of string;;
```

```
let pname(P s) = s;;
```

- Parser für atomare Aussagen:

```
let parse_propvar vs inp =
  match inp with
    p::oinp when p <> "(" -> Atom(P(p)),oinp
  | _ -> failwith "parse_propvar";;
```

Primitive Aussagen

```
# <<p \/ q ==> r>>;;
- : prop formula = <<p \/ q ==> r>>
# let fm = <<p ==> q <=> r /\ s \/ (t <=> ~ ~u /\ v)>>;;
val fm : prop formula = <<p ==> q <=> r /\ s \/ (t <=> ~(~u) /\ v)>>
```

Syntax Operationen

- Formel → OCaml Funktion:

```
let mk_and p q = And(p,q) and mk_or p q = Or(p,q)
and mk_imp p q = Imp(p,q) and mk_iff p q = Iff(p,q)
and mk_forall x p = Forall(x,p) and mk_exists x p = Exists(x,p);;
```

- Aufteilen einer Formel in Bestandteile:

```
let dest_or fm =
  match fm with Or(p,q) -> (p,q) | _ -> failwith "dest_or";;

let rec disjuncts fm =
  match fm with Or(p,q) -> disjuncts p @ disjuncts q | _ -> [fm];;
```

Syntax Operationen

- $p \Rightarrow q$
- p: Antecedent (erstes Glied, Bedingungsteil)
- q: Consequent (folglich)

```
let antecedent fm = fst(dest_imp fm);;
let consequent fm = snd(dest_imp fm);;
```

Funktion „onatoms“

- wendet eine Funktion f auf alle Atome einer Formel fm an

```
let rec onatoms f fm =
  match fm with
    Atom a -> f a
  | Not(p) -> Not(onatoms f p)
  | And(p,q) -> And(onatoms f p,onatoms f q)
  | Or(p,q) -> Or(onatoms f p,onatoms f q)
  | Imp(p,q) -> Imp(onatoms f p,onatoms f q)
  | Iff(p,q) -> Iff(onatoms f p,onatoms f q)
  | Forall(x,p) -> Forall(x,onatoms f p)
  | Exists(x,p) -> Exists(x,onatoms f p)
  | _ -> fm;;
```

2.2 Semantik der Aussagenlogik

- Formel: Behauptung → wahr oder falsch
- Auswertung eines algebraischer Ausdruck:
  x+y+1 → Werte für x und y
- Auswertung einer Formel in Aussagenlogik:
  Wahrheitswerte der Atome
- Auswertungsfunktion (valuation):
  Menge der Atome → {true, false}

2.2 Semantik der Aussagenlogik

```
let rec eval fm v =
  match fm with
    False -> false
  | True -> true
  | Atom(x) -> v(x)
  | Not(p) -> not(eval p v)
  | And(p,q) -> (eval p v) & (eval q v)
  | Or(p,q) -> (eval p v) or (eval q v)
  | Imp(p,q) -> not(eval p v) or (eval q v)
  | Iff(p,q) -> (eval p v) = (eval q v);;
```

2.2 Semantik der Aussagenlogik

p	q	$p \wedge q$	$p \vee q$	$p \Rightarrow q$	$p \Leftrightarrow q$
false	false	false	false	true	true
false	true	false	true	true	false
true	false	false	true	false	false
true	true	true	true	true	true

p	$\neg p$
false	true
true	false

2.2 Semantik der Aussagenlogik

```
# eval <<p /\ q ==> q /\ r>>
       (function P"p" -> true | P"q" -> false | P"r" -> true);;
...
- : bool = true
```

```
eval <<p /\ q ==> q /\ r>>
     (function P"p" -> true | P"q" -> true | P"r" -> false);;
```

Wahrheitswertetabellen

- Auswertung von Formel soll unabhängig von Werten der Atome sein, die nicht in Formel vorkommen
- Ziele
  - Funktion, die Atome aus einer Formel extrahiert
  - Wahrheitswertetabelle
- Definition der Atome:

Wahrheitswertetabelle

$$
\begin{aligned}
\texttt{atoms}(\bot) &= \emptyset \\
\texttt{atoms}(\top) &= \emptyset \\
\texttt{atoms}(x) &= \{x\} \\
\texttt{atoms}(\neg p) &= \texttt{atoms}(p) \\
\texttt{atoms}(p \wedge q) &= \texttt{atoms}(p) \cup \texttt{atoms}(q) \\
\texttt{atoms}(p \vee q) &= \texttt{atoms}(p) \cup \texttt{atoms}(q) \\
\texttt{atoms}(p \Rightarrow q) &= \texttt{atoms}(p) \cup \texttt{atoms}(q) \\
\texttt{atoms}(p \Leftrightarrow q) &= \texttt{atoms}(p) \cup \texttt{atoms}(q)
\end{aligned}
$$

Theorem 2.1

Für jede Formel p ist atoms(p) eine endliche Menge

Beweis per „struktureller Induktion“

- Beweis von Eigenschaften für Mengen, deren Elemente aus Grundelementen durch endliche Anzahl von Konstruktionsschritten entstehen
- Ind.- Anfang: Gültigkeit für Grundelemente
- Ind.- Schritt: Gültigkeit für rekursiv gebildete Elemente unter Voraussetzung, dass die Aussage für die in der Konstruktion verwendeten Elemente gilt

Theorem 2.2

Sei p eine Formel. Wenn die Auswertung v und v‘ auf der Menge atoms(p) übereinstimmen, folgt eval p v = eval p v‘.

Atome in OCaml

```
let atoms fm = atom_union (fun a -> [a]) fm;;
```

```
let rec overatoms f fm b =
  match fm with
    Atom(a) -> f a b
  | Not(p) -> overatoms f p b
  | And(p,q) | Or(p,q) | Imp(p,q) | Iff(p,q) ->
        overatoms f p (overatoms f q b)
  | Forall(x,p) | Exists(x,p) -> overatoms f p b
  | _ -> b;;
```

```
let atom_union f fm = setify (overatoms (fun h t -> f(h)@t) fm []);;
```

Atome in OCaml

For example:

```
# atoms <<p /\ q \/ s ==> ~p \/ (r <=> s)>>;;
- : prop list = [P "p"; P "q"; P "r"; P "s"]
```

Sei p Formel mit n Atomen → Auswertung von p durch 2^n mögliche Werte der Atome eindeutig festgelegt

→ Ziel: Wahrheitswertetabelle

Funktion „onallvaluations“

- rekursive Funktion
- testet ob die Funktion subfn „wahr“ zurückgibt für alle möglichen Werte der Atome ats, gegeben eine Auswertung v für alle anderen Atome

```
let rec onallvaluations subfn v ats =
  match ats with
    [] -> subfn v
  | p::ps -> let v' t q = if q = p then t else v(q) in
             onallvaluations subfn (v' false) ps &
             onallvaluations subfn (v' true) ps;;
```

Wahrheitswerttabelle

```
# print_truthtable <<p /\ q ==> q /\ r>>;;
p     q     r     | formula
-----------------------------
false false false |
false false true  |
false true  false |
false true  true  |
true  false false |
true  false true  |
true  true  false |
true  true  true  |
-----------------------------
- : unit = ()
```

Formale und natürliche Sprache

- Aussagenlogik: formale Art Aussagen, die in natürlichen Sprachen aufgestellt werden, aufzuschreiben
- nicht immer Wort- für- Wort- Beziehung
- Gruppierung von Aussagen (Klammerung) in natürlichen Sprachen schwierig
- Beispiel: $`p \wedge (q \vee r)`$ and $`(p \wedge q) \vee r`$

Formale und natürliche Sprache

- „und“, „oder“, „nicht“ kann direkt übersetzt werden
- in natürlicher Sprache hat „und“ oft kausale oder temporäre Bedeutung
- Problematisch: $p \Rightarrow q$
- sowohl im Sprachgebrauch als auch in Aussagenlogik: $p \Leftrightarrow q$ ist gleichbedeutend mit $p \Rightarrow q$ und $p \Leftarrow q$