Semantik und Syntax von Logik erster Stufe

Harrison Kapitel 3 Abschnitte 3 and 4

Amelie Flatt

AG Formale Mathematik, 2016

Übersicht

Semantik von Logik erster Stufe

- Variablenmengen
- Gültigkeit und Erfüllbarkeit

Syntaxoperationen

- Generalisierung
- Substitution in Terme
- Substitution in Formeln

Propositionale Logik vs Logik erster Ordnung

Semantische Unterschiede

Propositionale Logik	Logik erster Ordnung
Komponenten:	
Junktoren $\neg \wedge \vee \Rightarrow \Leftrightarrow$	Junktoren $\neg \wedge \vee \Rightarrow \Leftrightarrow$
	Quantoren $\exists \forall$
Variablen x,y,z,..	Variablen x,y,z,..
	Funktionssymbole f,g,h,..
	Relationssymbole P,Q,R,..
Interpretation:	
Auswertung von Variablen	Auswertung von Variablen
	Interpretation von Funktionssymbolen
	Interpretation von Relationssymbolen

Interpretation von Funktions- und Prädikatsymbolen

Eine Interpretation (Struktur) M besteht aus drei Teilen:

- Einer nichtleere Menge D als Grundmenge der Interpretation,
- für jedes n-stellige Funktionssymbol f einer Interpretation $f_M : D^n \to D$.
- für jedes n-stellige Relationssymbol P einer boolschen Interpretation $P_M : D^n \to \{false, true\}$.

In OCaml:

```
(domain,func,pred as m)
```

termval

$$\texttt{termval}\ M\ v\ x = v(x),$$

$$\texttt{termval M v (f(t}_1, \ldots t_n\texttt{))} = f_M(\texttt{termval}\ M\ v\ t_1, \ldots, \texttt{termval}\ M\ v\ t_n).$$

In Ocaml:

```
let rec termval (domain,func,pred as m) v tm =
match tm with
    Var(x) -> apply v x
  | Fn(f,args) -> func f (map (termval m v) args);;
```

holds

```
let rec holds (domain,func,pred as m) v fm =
match fm with
    False -> false
  | True -> true
  | Atom(R(r,args)) -> pred r (map (termval m v) args)
  | Not(p) -> not(holds m v p)
  | And(p,q) -> (holds m v p) && (holds m v q)
  | Or(p,q) -> (holds m v p) || (holds m v q)
  | Imp(p,q) -> not(holds m v p) || (holds m v q)
  | Iff(p,q) -> (holds m v p = holds m v q)
  | Forall(x,p) ->  forall (fun a -> holds m ((x |-> a) v) p) domain
  | Exists(x,p) -> exists (fun a -> holds m ((x |-> a) v) p) domain;;
```

Beispiel: Boolean algebra

```
let bool_interp =
    let func f args  =
    match (f,args) with
        ("0",[]) -> false
      | ("1",[]) -> true
      | ("+",[x;y]) -> not(x = y)
      | ("*",[x;y]) -> x && y
      | _ -> failwith "uninterpreted function"
    and pred p args =
    math (p,args) with
        ("=",[x;y]) -> x = y
      | _ -> failwith "uninterpreted predicate" in
    ([false;true],func,pred);;
```

Variablen in einem Term

FVT(t) beschreibt die Menge der Variablen in einem Term t, e.g. $\mathrm{FVT}(f(x+z, y*z)) = \{x, y, z\}$.

```
let rec fvt tm =
match tm with
    Var x -> [x]
  | Fn(f,args) -> unions (map fvt args);;
```

Theorem

Stimmen zwei Auswertungen v und v' für alle Variablen eines Terms t überein, d.h. $\forall x \in FVT(t)$ *gilt* $v(x) = v'(x)$*, dann gilt auch* `termval` $M\ v\ t =$ `termval` $M\ v'\ t$.

Variablen in einer Formel

Alle Variablen:

```
let rec var fm =
match fm with
    False | True -> []
  | Atom(R(p,args)) -> unions (maps fvt args)
  | Not(p) -> var p
  | And(p,q) | Or(p,q) | Imp(p,q) | Iff(p,q) -> union (var p) (var q)
  | Forall (x,p) | Exists(x,p) -> insert x (var p);;
```

Freie Variablen:

```
let rec fv fm =
match fm with
    False | True -> []
  | Atom(R(p,args)) -> unions (maps fvt args)
  | Not(p) -> fv p
  | And(p,q) | Or(p,q) | Imp(p,q) | Iff(p,q) -> union (fv p) (fv q)
  | Forall (x,p) | Exists(x,p) -> subtract (fv p) [x];;
```

Satz

Stimmen zwei Auswertungen v und v' für alle freien Variablen eine Formel p berein, d.h. $\forall x \in FV(p)$ *gilt* $v(x) = v'(x)$,
dann gilt auch `holds` M v $p =$ `holds` M v' p.

Corollary

Ist p ein Satz, d.h. $FV(p) = \emptyset$, *dann gilt für jede Struktur M und alle Auswertungen v und v':* `holds` M v $p =$ `holds` M v' p.

Gültigkeit und Erfüllbarkeit

Gültigkeit:

- *logisch gültige* Formel erster Ordnung:
  wird von allen Interpretationen und allen Auswertungen erfüllt
- *logisch äquivalente* Formeln erster Ordnung p und q:
  $p \Leftrightarrow q$ logisch gültig

Erfüllbarkeit:

- Eine Interpretation M *erfüllt* eine Formel erster Ordnung p (p holds in M): für alle Auswertungen v gilt `holds` $M\ v\ p = \text{true}$.
- Eine Interpretation M erfüllt eine Menge von Formeln S (S holds in M): M erfüllt simultan jede Formel der Menge S.

Generalisierung

Idee:
Quantifiziere universell über alle freien Variablen einer Formel erster Ordnung. ($\forall$)

Dann gilt für alle Auswertungen v und alle $a \in D$ gilt `holds` M $((x \mapsto a)v)p =$ `holds` M v p.

Implementation:

```
let generalize fm = itlist mk_forall (fv fm) fm;;
```

Substitution in Terme

```
let rec tsubst sfn tm =
    match tm with
        Var x -> tryapplyd sfn x tm
      | Fn(f,args) -> Fn(f,map (tsubst sfn) args);;
```

Lemma

Für jeden Term t und jede Instanz i sind die freien Variablen im durch Substitution erhaltenen Term genau die freien Variablen der in freie Variablen von t substituierten Terme, d.h.

$$FVT(\texttt{tsubst}\ i\ t) = \bigcup_{y \in FVT(t)} FVT(i(y)).$$

Lemma

Für jeden Term t, jede Instanz i, jede Struktur M und jede Auswertung v hat der durch Substitution erhaltene Term den gleichen Wert wie der ursprngliche Term mit der modifizierten Wertzuweisung $\texttt{termval}\ M\ v \circ i$, *d.h.*
$\texttt{termval}\ M\ v\ (\texttt{tsubst}\ i\ t) = \texttt{termval}\ M\ (\texttt{termval}\ M\ v \circ i)t.$

Substitution in Formeln

```
let rec variant x vars =
    if mem x vars then variant (x^"'") vars else x;;
```

```
let rec subst subfn fm =
match fm with
    False -> False;
  | True -> True;
  | Atom(R(p,args)) -> Atom(R(p,map (tsubst subfn) ars))
  | Not(p) -> Not(subst subfn p)
  | And(p,q) -> And(subst subfn p,subst subfn q)
  | Or(p,q) -> Or(subst subfn p,subst subfn q)
  | Imp(p,q) -> Imp(subst subfn p,subst subfn q)
  | Iff(p,q) -> Iff(subst subfn p,subst subfn q)
  | Forall(x,p) -> substq subfn mk_forall x p
  | Exists(x,p) -> subst subfn mk_exists x p
```

```
and substq subfn quant x p =
    let x' = if exists (fun y -> mem x (fvt(tryapplyd subfn y (Var y))))
                 (subtract (fv p) [x])
             then variant x (fv(subst (undefine x subfn) p)) else x in
    quant x' (subst ((x |-> Var x') subfn) p);;
```

Lemma

Für jede Formel p und jede Instanz i sind die freien Variablen in der durch Substitution erhaltenen Formel genau die freien Variablen der in freie Variablen von p substituierten Terme, d.h.

$$FV(\texttt{subst}\ i\ p) = \bigcup_{y \in FV(p)} FVT(i(y)).$$

Theorem

Für jede Formel p, jede Instanz i, jede Struktur M und jede Auswertung v gilt $\texttt{holds}\ M\ v\ (\texttt{subst}\ i\ p) = \texttt{holds}\ M\ (\texttt{termval}\ M\ v \circ i)p$.

Corollary

Für eine gültige Formel ist auch jede Substitutionsinstanz gültig.