Grundlagen der Kognitiven Informatik

Resolutionskalkül und Prolog

Ute Schmid

Kognitive Systeme, Angewandte Informatik, Universität Bamberg

letzte Änderung: 14. Dezember 2010

Automatische Beweise

- Der Hauptgewinn einer formal definierten Logik ist, dass eindeutig festgelegt werden kann, wie logisch korrekte Schlussfolgerungen gezogen werden können.
- Darauf aufbauend können dann Algorithmen entwickelt werden, die es ermöglichen, dass logische Schlussfolgerungen automatisch, durch einen Computer, ausführbar werden.
- Der wesentliche Schritt zu diesem sogenannten automatischen Theorembeweisen war das von Robinson 1965 eingeführte **Resolutionsverfahren**.
- Automatische Theorembeweiser nutzen die Beziehung zwischen logischen Schlussfolgerungen und syntaktischen Ausdrücken aus.
- Wenn beispielsweise rein syntaktisch, mithilfe von Inferenzregeln, gezeigt werden kann, dass $(\wedge_{i=1}^{n} F_i) \wedge \neg G$ zu falsch auswertet, so ist gezeigt, dass G logisch aus der Formelmenge F folgt! Das heisst, G ist das bewiesene Theorem.

Resolutionskalkül

- Ist gegebenes Wissen formal repräsentiert, etwa aussagenlogisch oder prädikatenlogisch, so können Schlüsse rein mechanisch, durch ein Computerprogramm, ausgeführt werden.
- Das Resolutionskalkül ist eines der bekanntesten Verfahren zum automatischen Schlussfolgern.
- Es besteht nur aus einer Regel, nämlich:

$$(P_1 \vee \ldots \vee P_n \vee Q) \wedge (\neg Q \vee R_1 \vee \ldots \vee R_n) \rightarrow (P_1 \vee \ldots \vee P_n \vee R_1 \vee \ldots \vee R_n).$$

Resolutionskalkül

- Betrachtet man also zwei Klauseln, die beide als wahr angenommen werden (deshalb werden diese konjunktiv verknüpft), und taucht ein Prädikat in einer Klausel positiv und in der anderen negiert auf, so können die beiden Klauseln zu einer zusammengefügt werden, wobei die Klausel, die positiv und negativ auftauchte, gestrichen wird.
- Für zwei Klauseln $P \wedge \neg P$ resultiert diese Regel in der leeren Klausel.
- Offensichtlich kann ein Prädikat nie gleichzeitig mit seinem Gegenteil gelten – die leere Klausel repräsentiert also den Wahrheitswert "falsch" beziehungsweise einen Widerspruch!

Resolutionskalkül

- Die Beweisidee hinter dem Resolutionskalkül entspricht also der oben angegebenen Vorgehensweise, dass $(\wedge_{i=1}^{n} F_i) \wedge \neg G$ zu falsch auswertet.
- Resolution basiert also darauf, einen Widerspruchsbeweis zu führen: Um zu zeigen, dass eine Formel G aus einer Menge von Formeln F logisch folgt, wird G negiert zur Formelmenge hinzugefügt und gezeigt, dass dadurch generell "falsch" abgeleitet wird – was gleichbedeutend damit ist, dass ein Widerspruch in der Formelmenge steckt.

Resolutionskalkül

- Viele komplexere Deduktionssysteme sowie die Programmiersprache **Prolog** basieren auf Resolution.
- Damit das Verfahren angewendet werden kann, müssen Formeln in einheitlicher Form repräsentiert werden – nämlich in **Klauselform**.
- Eine Klausel ist eine Disjunktion von Prädikaten.
- Variablen müssen alle über Allquantoren gebunden sein.
- Beliebige Formeln können durch Äquivalenzumformungen in Klauselform umgewandelt werden.
- Alle notwendigen Schritte, um eine solche Umformung vorzunehmen werden in Logikbüchern und KI-Lehrbüchern dargestellt.

Beispiel

Im folgenden wird Resolution anhand eines Beispiels eingeführt. Wieder betrachten wir den einfachen Syllogismus von der Sterblichkeit, diesmal in prädikatenlogischer Form:

$$(1)\ \forall(x)\mathit{mensch}(x) \rightarrow \mathit{sterblich}(x)$$

$$(2)\ \mathit{mensch(Sokrates)}.$$

Da ein einzelnes Prädikat bereits in Klauselform (mit null Disjunktionen) ist, muss nur die erste Formel umgewandelt werden. Durch Auflösen der Implikation ergibt sich:

$$(1')\ \forall(x)(\neg\mathit{mensch}(x) \vee \mathit{sterblich}(x))$$

Die Formel enthält genau eine allquantifizierte Variable. Da Klauseln, wie oben definiert, nur allquantifizierte Variablen enthalten, kann der Quantor zur Vereinfachung weggelassen werden (die Formel ist dann implizit allquantifiziert):

$$(1'')\ \neg\mathit{mensch}(x) \vee \mathit{sterblich}(x)$$

Beispiel

Um nun zu prüfen, ob *sterblich(sokrates)* eine gültige Aussage ist (aus der Formelmenge folgt), wird diese "Vermutung" negiert zur Klauselmenge hinzugefügt:

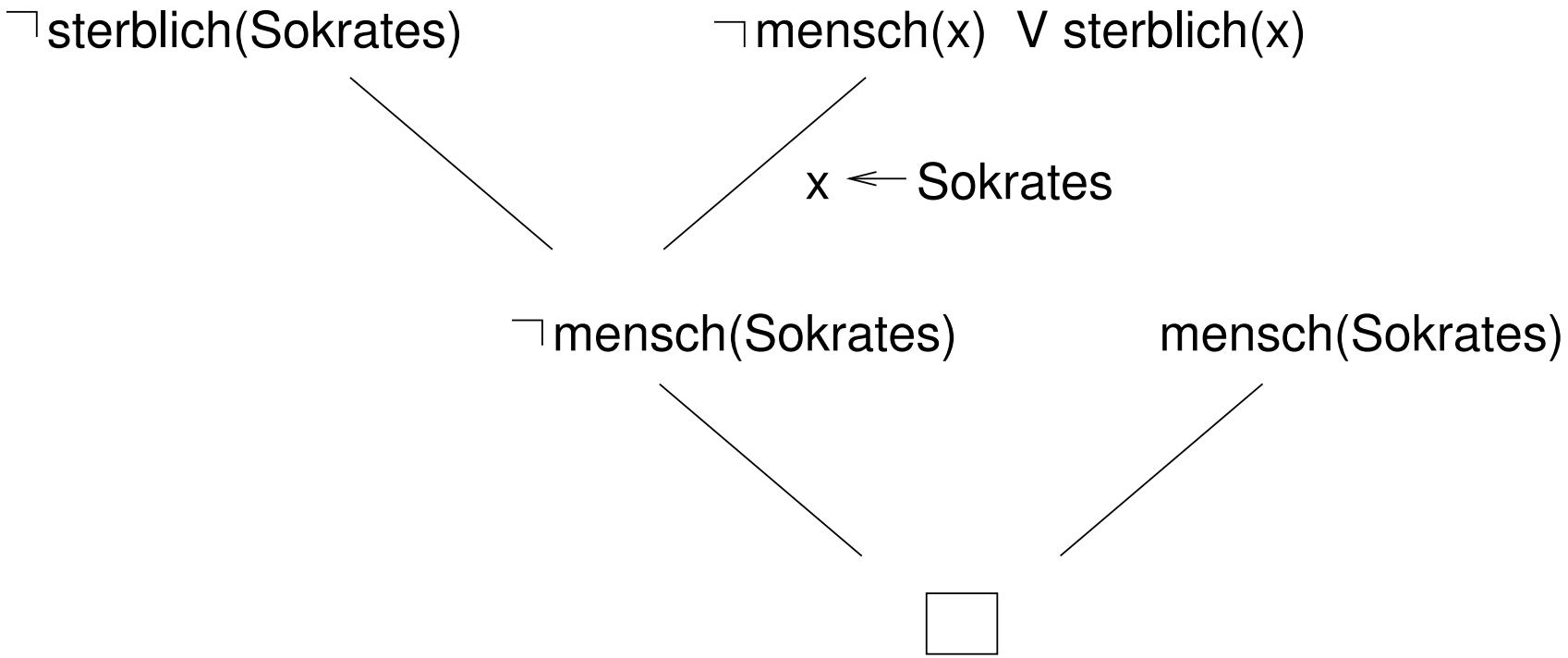
$$(3)\ \neg sterblich(sokrates).$$

- Der Resolutionsbeweis wird nun so geführt, dass schrittweise immer zwei Formeln betrachtet werden, auf die die Resolutionsregel angewendet wird – solange bis die leere Klausel abgeleitet wird.
- Ableiten einer leeren Klausel bedeutet, dass ein Widerspruch im Formelsystem steckt.
- Wenn mit einer negierten Formel G ein Widerspruch in der Formelmenge erzeugt wird, ist dies gleichbedeutend damit, dass die positive Formulierung von G aus der Formelmenge folgerbar ist.

Resolutionsbeweis

- Resolution kann in Form eines sogenannten **Refutationsbaums** dargestellt werden.
- Wenn man mit der negierten Formel *(3)* startet und Formel *(1")* hinzufügt (die Konjunktion wird hier nicht mitnotiert), so taucht das Prädikat *sterblich* in Formel *(3)* negiert und in Formel *(1")* positiv auf.
- Die Variable x darf durch die Konstante *Sokrates* ersetzt werden.
- Im Allgemeinen müssen für jeden Resolutionsschritt zwei Formeln unifiziert werden, was heißt, dass Variablen so durch Konstanten oder komplexere Terme ersetzt werden müssen, dass die in der Resolution herauszuschneidenden Prädikatausdrücke identisch werden.
- Nun kann der Resolutionsschritt durchgeführt werden.
- Die resultierende Formel kann nun zusammen mit Klausel *(2)* betrachtet werden, was in einem Widerspruch resultiert.
- Damit wurde nun durch Widerspruch bewiesen, dass die Formel *sterblich(Sokrates)* aus Formeln *(1)* und *(2)* logisch folgt.

Refutationsbaum



Resolutionsalgorithmus

Suchstrategie

- Ob ein Widerspruch (in endlicher Zeit) wirklich gefunden wird, wenn ein solcher existiert, hängt unter anderem von der Reihenfolge ab, in der die Klauseln in die Resolution einbezogen werden.
- Je größer die Menge an Formeln (Wissen) ist, aus der Schlüsse gezogen werden sollen, desto komplexer wird das Problem der Suche nach geeigneten Klauselpaaren.
- Entsprechend wird ein Resolutionsverfahren immer zusammen mit einer Suchstrategie (zum Beispiel Tiefensuche) realisiert.
- Um das Ziehen von Schlüssen effizienter zu machen, wird ein logisches System häufig eingeschränkt, in dem die sogenannte *closed-world assumption* eingeführt wird.
  Basierend auf dieser Annahme kann man dann davon ausgehen, dass etwas, was nicht als wahr ableitbar ist, nicht gilt.

Prolog

- Prolog ist wie Lisp eine deklarative Programmiersprache.
- Lisp ist eine funktionale Sprache, Prolog eine (*die*) logische Sprache.
- Prolog basiert auf dem Resolutionskalkül.
- Als Suchstrategie wird die SLD-Resolution verwendet:
  - D steht für definite Klauseln.
  - S steht für *select*.
  - L steht für lineare Strategie.

```
is_a(fisch,tier).
is_a(steinbutt,fisch).
is_a(herz,organ).
has_prop(tier,herz).

simpleTrans(X, Y) :- is_a(X, Z), is_a(Z, Y).
```

Syntax von Prolog

Fakten und Regeln

- Prädikatsymbol mit Argumenten in runden Klammern.
- Abschluss einer Klausel immer mit Punkt.
- Konstanten beginnen mit Kleinbuchstaben.
- Variablen beginnen mit Großbuchstaben.
- Auswertung von Klauseln zu wahr oder falsch.
- **Fakten** werten immer zu wahr aus.
- `is_a(fisch,tier).` ist Abkürzung für `is_a(fisch, tier) :- true.`
- **Regeln** haben einen Kopf und einen Körper. `:-` entspricht einem umgekehrten Implikationspfeil.
- Komma ist eine konjunktive Verknüpfung. Semikolon eine disjunktive Verknüpfung.

Syntax von Prolog

```
simpleTrans(X, Y) :- is_a(X, Z), is_a(Z, Y).
```

entspricht der definiten Hornklausel

$$\neg \text{ is\_a}(X, Z) \vee \neg \text{ is\_a}(Z, Y) \vee \text{simpleTrans}(X, Y)$$

Herleitung

- *Modus barbara* (Transitiver Schluss)
  $\forall X, Y, Z \quad \text{is\_a}(X, Z) \wedge \text{is\_a}(Z, Y) \rightarrow \text{simpleTrans}(X, Y)$
- Umwandlung in Klauselform
  $\neg$ (is_a$(X, Z) \wedge$ is_a(Z, Y)) $\vee$ simpleTrans(X, Y)
  $\equiv \neg$ is_a$(X, Z) \vee \neg$ is_a$(Z, Y) \vee$ simpleTrans(X, Y)
- Definite Hornklausel: Maximal ein positives Literal (entspricht Regelkopf)

Auswertung mit Pattern-Matching

- In Lisp werden Funktionen mit einem eval-apply Interpreter ausgewertet.
- Das Prädikat `atom(cdr(L))` wird ausgewertet, indem von der an Variable `L` gebundenen Liste diese Liste ohne das erste Element geliefert wird. Ist diese Restliste leer entspricht dies dem Atom `nil`. `atom(nil)` werte zu `T`, also wahr, aus.
- In Prolog werden Ausdrücke dagegen nicht ausgewertet, sondern über Pattern-Matching belegt und geprüft, ob der Ausdruck wahr ist.

Append in Lisp:

```
(defun append (L1, L2)
  (cond ((null L1) L2)
        (T (cons (car L1) (append (cdr L1) L2)))
) )
```

Append in Prolog:

```
append([], L, L).
append([X|Xs], L2, [X|L]) :- append(Xs, L2, L).
```

Funktionen versus Relationen

- In Lisp wird append als Funktion definiert, die zwei Listen als Argument erhält und die verknüpfte Liste zurück liefert.
- In Prolog ist append dagegen eine Relation.
- Diese Relation ist wahr, wenn das dritte Argument genau der Verknüpfung der beiden ersten Argumente entspricht.
- Entsprechend kann append auch dazu verwendet werden, das erste oder das zweite Argument zu berechnen.

```
?- append([a,b],[c], L).
L = [a, b, c].

?- append(L, [c], [a,b,c]).
L = [a, b]
```

```
?- append(X,Y,[a,b,c]).
X = [],
Y = [a, b, c] ;
X = [a],
Y = [b, c] ;
X = [a, b],
Y = [c] ;
X = [a, b, c],
Y = [] ;
fail.
```

Anfragen und Auswertung

- Ein Prolog Programm wird in den Interpreter geladen (`consult(name).`)
- Danach können Anfragen gestellt werden.
  z.B. `is_a(X,tier).`
- Prolog antwortet mit der Variablenbelegung oder mit `fail`.
- Durch Eingabe von Semikolon (*oder*) kann gefragt werden, ob es noch weitere Belegungen gibt, mit denen die Klauseln zu wahr auswerten.
- Die Variablenbelegung wird über Resolution berechnet.
- Die Anfrage wird negiert und der Klauselmenge (dem Programm) hinzugefügt.
- Danach werden die Klauseln von oben nach unten (*select* ist top-down, left-to-right)) betrachtet – solange bis eine Resolvente mit der Anfrage berechnet werden kann (Matching und Unifikation).
- Die lineare Strategie legt fest, dass die Resolvente festgehalten wird und für diese eine weitere Klausel gesucht wird, mit der die Resolution fortgeführt werden kann.

Ein semantisches Netz in Prolog

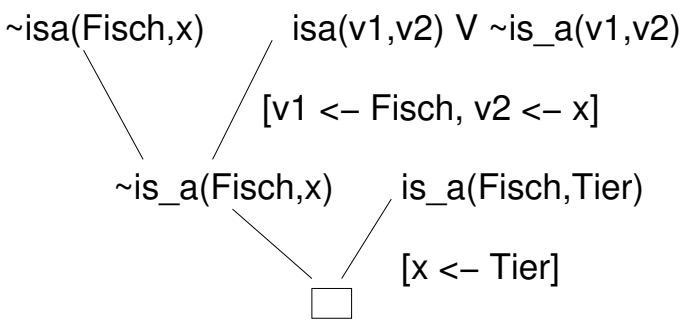
```
/* explizite Kanten im Netz  */
is_a(tier,lebewesen).
is_a(fisch,tier).
is_a(steinbutt,fisch).
is_a(herz,organ).
has_prop(tier,herz).
has_prop(organ,gewebe).
has_prop(gewebe,zellen).

/* ------------------------------------------------- */
/* Ableitungsregeln im Netz                          */
/* A,B,C sind Konzepte; X,Y,Z sind Eigenschaften     */

isa(A,B) :- is_a(A,B).              /* R1: direkter Fall isa       */
isa(A,C) :- is_a(A,B), isa(B,C).    /* R2: Transitivität von isa   */

has(A,X) :- has_prop(A,X).          /* R3: direkter Fall has       */
has(X,Z) :- has_prop(X,Y), has(Y,Z). /* R4: Transitivität von has  */

has(A,X) :- has_prop(A,Y), isa(Y,X). /* R5: Verallg. von has bzgl. isa */
has(A,X) :- is_a(A,B), has(B,X).    /* R6: Vererb. von has bzgl. isa */
```

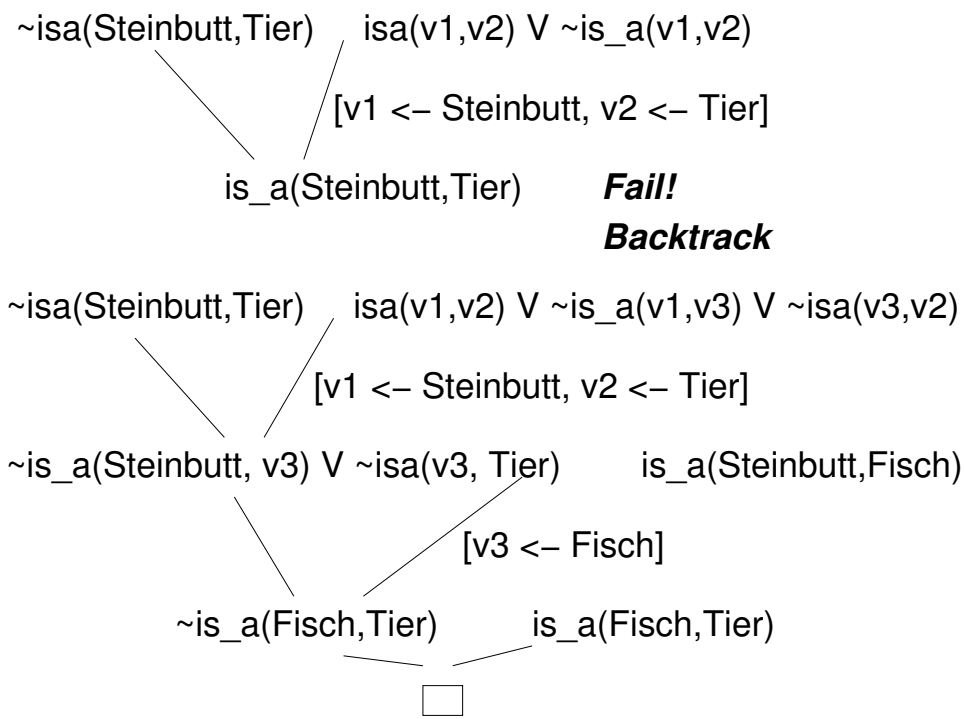
Prolog

	PROLOG	Logic	
Fact	`is_a(fisch,tier).`	$\text{is\_a}(\text{Fisch},\text{Tier})$	Positives Literal
	`is_a(steinbutt,fisch).`	$\text{is\_a}(\text{Steinbutt},\text{Fisch})$	
Rule	`isa(X,Y) :-` `is_a(X,Y).`	$\text{isa}(x,y) \vee \neg\text{is\_a}(x,y)$	Definite Klausel
	`isa(X,Z) :-` `is_a(X,Y), isa(Y,Z).`	$\text{isa}(x,z) \vee \neg\text{is\_a}(x,y)$ $\vee\neg\text{isa}(y,z)$	
Anfrage	`isa(steinbutt,tier).`	$\neg\text{isa}(\text{Steinbutt},\text{Tier})$	Behauptung
	`isa(Fisch,X)`	$\neg\text{isa}(\text{Fisch},x)$	

Auswertung mit Resolution

- Anfrage: isa(fisch,X)
  ($\exists x$ isa(Fisch, x))
- Negation der Anfrage: $\neg\exists x$: isa(Fisch, x) $\equiv \forall x$: $\neg$isa(Fisch, x)
- SLD-Resolution:*(Auszug)*





Anmerkungen

- Aufgrund der *select* Strategie ist Prolog unvollständig. Das heisst, es findet möglicherweise keinen Beweis für eine Anfrage, obwohl einer existiert.
- Effizienz: Fakten über Regeln.
- Termination: nicht-rekursive Regeln über rekursive.

```
% Program
is_a(steinbutt,fisch).
is_a(fisch,tier).

isa(X,Z) :- isa(X,Y), is_a(Y,Z).
isa(X,Y) :- is_a(X,Y).
```

```
% Query
? isa(steinbutt,tier).


isa(steinbutt,Y), is_a(Y,tier)
isa(steinbutt,Y'),is_a(Y',tier),
                  is_a(Y,tier)
...
```

Anwendungen des Resolutionskalküls

- Prolog als Sprache für Wissensrepräsentation und logisches Schließen
- Grundlegende Methode für automatischen Theorembeweis
- Beziehung zu Beschreibungslogiken/Ontologien (*semantic web*)
- CYC-Projekt (großes Projekt zur Modellierung von Ontologien)
- Expertensysteme, Frage-Antwort-Systeme

- Ja/Nein Fragen: Zusicherung/Anfrage `sterblich(s)`
- Was-Fragen: `isa(steinbutt,X)` *Was ist ein Steinbutt?*
  Variable X wird während der Resolution belegt. Antwort: *ein Fisch*, *ein Tier*
- `buys(peter,john,X)`: *Was kauft John von Peter?*
- `buys(peter,X,car)`: *Wer kauft ein Auto von Peter?*