Einführung in Induktionsbeweise

Institut für Informatik Potsdam

Seminar Inferenzmethoden

Marc Hunger
marc.hunger@lycos.de

Inhaltsverzeichnis

Einleitung

Dieses Dokument wurde im Rahmen des Seminars Inferenzmethoden an der Universität Potsdam angefertigt. Es beinhaltet eine Einführung in Induktionsbeweise sowie Techniken zur Automatisierung, Induktionsbeweise werden für Aussagen über rekursiv definierten Objekten oder Prozeduren benötigt. Dazu zählen rekursive Datentypen, wie natürliche Zahlen oder Listen, Programme, welche Rekursion oder Iteration beinhalten, oder elektrische Schaltkreise mit Rückkopplung.
Eine Aufgabe der Induktion ist es, den Beweis einer Aussage für unendlich viele Objekte eines Datentyps (oder unendlich viele Programmverläufe eines Programm) auf endlich viele Teilbeweise zu reduzieren. Weiterhin gibt es für eine Menge von Gleichungen viele Modelle. Ein Beweis in dem, in der Aussage, nur Gleiches durch Gleiches ersetzt wird, bis eine Tautologie der Form: x=x entsteht, zeigt die Gültigkeit der Aussage in allen Modellen. Die zweite Aufgabe der Induktion ist es somit die Gültigkeit einer Aussage nur für einen bestimmten Datentyp zu zeigen. Die beiden Aufgaben der Induktion können mit zwei verschiedenen Ansätzen durchgeführt werden.

1. explizite Induktion

Hierbei wird aus Funktionsdefinitionen und einer Vermutung ein Induktionsaxiom gewonnen und mit verschiedenen Heuristiken bewiesen. Diese Axiome bestehen aus einer Menge von Basisfällen, in denen die Aussage für alle minimalen Elemente bewiesen werden muss. Und aus einer Menge von Induktionsschritten, in denen gezeigt wird, dass die Aussage für ein Element wahr wird, wenn sie für alle „kleineren“ Elemente wahr ist. Die bekannteste Form ist die Peano Induktion (oder vollständige) für natürliche Zahlen.

$$\frac{P \in N(0) \wedge \forall n \in N\ (P(n) \Rightarrow P(succ(n)))}{\forall n \in N\ (P(n))}$$

Zur Durchführung solcher Beweise sind eine Menge von Heuristiken Notwendig. Dazu zählen Verallgemeinerrungen der Vermutung, Rippling (Umformen des Induktionsschritt bis die Induktionsannahme als Teilaussage im Induktionsschritt vorkommt), Fertilization (Abschluss des Rippling, in dem die Induktionsannahme als Ersetzungsregel verwendet wird) oder das Spekulieren und Beweisen von Lemmata.

2. implizite Induktion

Eine Menge von Gleichungen E, definiert auf der Menge der Grundterme (Terme in denen keine Variablen vorkommen) eine Kongruenzrelation bezüglich der Funktionssymbole. Wenn durch Hinzufügen der Vermutung P (einer Gleichungen), diese Relation nicht verändert wird, bedeutet dies, dass alle Grundterme welche, durch E∪P kongruent sind, auch durch E kongruent sind. Das heißt das Hinzufügen von P setzt nicht mehr gleich, als bereits gleich war und damit ist P wahr. Verändert P diese Relation, setzt also Terme gleich, welche durch E ungleich waren, dann ist dies ein Widerspruch zu E und die Aussage ist falsch. In der impliziten berechnet man aus E und E∪P vollständige Termersetzungssysteme und führt einen Konsistenztest durch. In solchen Systemen kann jeder Term zu einer eindeutigen Normalform (ein irreduzibler Term) abgeleitet werden. Die Kongruenzrelation, ist durch die Menge der Normalformen definiert. Denn eine Gleichung s=t gilt in E, bzw. dessen vollständiges Ersetzungssystem, genau dann, wenn die Normalformen von s und t identisch sind. Der Konsistenztest besteht darin, für beide Ersetzungssysteme eine endliche Beschreibung der Normalformen zu finden und diese zu vergleichen. Diese Form der Induktion wird in diesem Dokument nicht beschrieben.

Im Folgenden werden verschiedene Formen von Induktionsregeln, Möglichkeiten der Definition von rekursiven Datentypen und Funktionen sowie Techniken zur Beweisführung beschrieben. Dazu zählen verschiedene Ersetzungsregeln, wie Fertlization sowie einige Entscheidungsprozeduren. Heuristiken und die Grenzen der Induktion werden an anderer Stelle des Seminars beschrieben.

Iduktionsregeln

Fundierte Relationen

Jedes Induktionsaxiom basiert auf einer binären fundierten (terminierend oder nœthersch) Relation < über einer Menge M. Dabei enthält jede nichtleere Teilmenge S von M, wenigstens ein minimales Element s (s ist minimal in S, wenn es kein x in S gibt, sodass x<s). Eine äquivalente Definition ist: Es gibt keine unendliche absteigende Kette der Form: $\ldots x_n<\ldots<x_1<x_0$. Aus diesen Definitionen ergibt sich die Irreflexivität und Antisymmetrie von fundierten Relationen. Diese Relationen sind nicht notwendigerweise transitiv (und damit keine Ordnungsrelation).
Beispiele für fundierte Relationen sind:

- n<m gdw. m=succ(n) (nicht transitive, Relation der vollständigen Induktion)
- die kleiner-Relation der natürlichen Zahlen
- lexikographische Ordnungen: $(t_1,\ldots,t_n)<_{lex}(s_1,\ldots,s_m)$
  - $t_1<s_1$ oder
  - $s_1=t_1$ und $(t_2,\ldots,t_n)<_{lex}(s_2,\ldots,s_m)$
  - n=0 und m>0 oder
- Abbildungen in natürliche Zahlen wie: Tiefe und Breite von Termen oder die Länge von Listen

Nœthersche Induktion

Für verschiedene Datentypen und Funktionsdefinitionen existieren in der expliziten Induktion unterschiedliche Induktionsschemen. Sie basieren alle auf einer beliebigen fundierten Relation < und sind Spezialfälle der Nœtherschen Induktion (benannt nach Emily Noether).

$$\frac{\forall x\in M[\forall y\in M\,(y<x\Rightarrow P(y))\Rightarrow P(x)]}{\forall x\in M\;P(x)}$$

In diesem Axiom ist die Prämisse $\forall y\,(y<x\Rightarrow P(y))$ Trivialerweise wahr für alle minimalen Elemente (da es keine kleineren Elemente gibt). Dadurch ist für diese Fälle $\top\Rightarrow P(x)$ zu beweisen. Für jedes nicht minimale Element muss P(x) nur unter der Annahme der Prämisse bewiesen werden. Der Erfolg des Beweises für eine Aussage ist abhängig von der Wahl der Induktionsvariablen und der fundierten Relation.

Man kann die Korrektheit dieses Prinzips mit folgendem Widerspruchsbeweis zeigen:

Sei $Q=\{x \mid \neg P(x)\}$, das Nœthersche Induktionsaxiom sei wahr für die fundierte Relation < über M.
Dann ist das Prinzip ist korrekt wenn $Q=\emptyset$ ist.
Wenn $Q\neq\emptyset$, dann gibt es ein minimales Element m in Q.
Damit sind alle Elemente m_i, für die gilt: $m_i<m$, nicht Element von Q
(sonst wäre m nicht minimal in Q).
Daraus folgt $\forall m_i\,(m_i<m\Rightarrow P(m_i))$ und somit gilt P(m) und $m\notin Q$. □

Konstruktor- und Destruktorform

Die meisten Beweissysteme verwenden statt der allgemeinen fundierten Induktion speziellere Schemen, welche sich aus der Definition der Funktionen ergeben. Dazu zählt die Peano Induktion für natürliche Zahlen und ähnliche Formen für Listen etc. Anstatt, das nœthersche Induktionsaxiom mit einer konkreten fundierten Relation und einer Menge zu instanziieren, wird eine Konjunktion aus Basisfällen und Induktionsschritten gebildet. Für eine Aussage $\forall x\in M\;P(x)$, wird für jedes minimales Element m_i ein Basisfall gebildet, indem in x durch m_i ersetzt wird. Bei der Bildung der Induktionsschritte kann in Konstruktor- und Destruktorform unterschieden werden. In Konstruktorform wird eine Implikation der Form:

$$P(x_1)\wedge\ldots\wedge P(x_n)\Rightarrow P(c(x_1,\ldots,x_n)) \qquad \text{mit } x_i < c(x_{1,\ldots,}x_n)$$

für jeden Konstruktor c gebildet. Peano Induktion ist ein Beispiel für derartige Induktionsaxiome.
In Destruktorform werden Induktionsschritte der Form:

$$P(d_1(x))\wedge\ldots\wedge P(d_n(x))\Rightarrow P(x) \qquad \text{mit } d_i(x) < x$$

gebildet.

Die fundierte Relation auf denen solche Axiome beruhen ist durch die Konstruktoren bzw. den Destruktoren definiert. Dabei ist $x<y$ gdw. $y<c_i(z_1,\ldots,x,\ldots,z_n)$ für einen beliebigen Konstruktor (bzw. $x=d_i(y)$ für einen beliebigen Destruktor).

mehrere universelle Variablen

Wenn eine Aussage über mehrere Variable bewiesen wird, dann besteht die Möglichkeit eine als Induktionsvariable zu wählen. Dabei entstehen gewisse Freiheiten in der Verwendung der Induktionsannahme. Die Gültigkeit des Quantors der Induktionsvariablen ist der komplette Induktionsschritt. Die Quantoren der übrigen Variablen beschränken sich auf die Prämisse bzw. den Schluss.

$$\forall n\in S\ [(\forall m\in S\ Q(n,m)) \Rightarrow (\forall m\in S\ (Q(s(n),m)))]$$

Dadurch kann m in der Prämisse umbenannt werden.

$$\forall n\in S\ [(\forall M\in S\ Q(n,M)) \Rightarrow (\forall m\in S\ (Q(s(n),m)))]$$

Die Auflösung der Implikation ergibt:

$$\forall n\in S\ [\neg(\forall M\in S\ Q(n,M)) \vee (\forall m\in S\ Q(s(n),m))]$$
$$\forall n\in S\ [\ (\exists M\in S\ \neg Q(n,M)) \vee (\forall m\in S\ Q(s(n),m))]$$

Bildet man hierfür die Skolemnormalform, dann wird die Variable M, in der Hypothese zu einer Skolemkonstanten.

$$\forall n\in S\ \forall m\in S\ \exists M\in S\ [\ \neg Q(n,M) \vee Q(s(n),m)]$$
$$\forall n\in S\ \forall m\in S\ [\ \neg Q(n,\ skolem(n,m)) \vee Q(s(n),m)]$$
$$\forall n\in S\ \forall m\in S\ [Q(n,\ skolem(n,m)) \Rightarrow Q(s(n),m)]$$

Sollte die Hypothese zum Beweis des Schlusses verwendet werden entsteht eine Freiheit, in der M nicht mit m unifiziert werden muss. Stattdessen kann M durch eine beliebigen Term ersetzt werden. Alle Allquantifizierten Variablen sind beliebig aber fest. D.h. sie dürfen nur mit sich selbst unifiziert werden. Standardmäßig werden die allquantifizierte Variablen klein und Skolemkonstanten groß geschrieben. Der Umgang mit diesen beiden Typen von Variablen wird durch ersetzen der Implikationen durch $\vdash$ verdeutlicht.

$$Q(n,\ M) \vdash Q(s(n),m)$$

Diese Freiheit ist zum Beispiel für den Beweis der Kommutativität ($\forall x\forall y\ x+y=y+x$) der Addition in den natürlichen Zahlen hilfreich.

Bsp: (1) $x+0=x$ (2) $0+x=x$ (3) $x+succ(y)=succ(x)+y$

Für den Induktionsschritt ist: $x+Y=Y+x \vdash succ(x)+y=y+succ(x)$ zu beweisen.

$x+Y=Y+x$	$\vdash$	$succ(x)+y=y+s(x)$	
	$\vdash$	$succ(x)+y=$**$succ(y)+x$**	(3)
	$\vdash$	$succ(x)+y=$**$x+ succ(y)$**	ersetzen von **Y durch succ(y)** und Anwenden der Induktionsannahme
	$\vdash$	$succ(x)+y=$**$succ(x)+y$**	(3)

Rekursive Datentypen und Funktionen

rekursive Datentypen

Rekursive Datentypen werden durch eine Menge von Funktionssymbolen, den Konstrukroren, gebildet. Der Datentyp, der durch diese Menge definiert wird, wird als die Menge der Terme über den Funktionssymbolen interpretiert.

Anhand der Gleichheitsrelation zwischen diesen Termen, können Datentypen in freie und nichtfreie Datentypen unterschieden werden. Bedeutet Gleichheit syntaktische Gleichheit der Terme, dann spricht man von freien Datentypen, sonst von nicht freien. Zu freien Datentypen zählen natürliche Zahlen, welche durch: {0:→nat, succ:nat→nat} definiert werden. können. Weitere wichtige Datentypen sind Listen von einem bestimmten Elementtyp, wie natürliche Zahlen oder Listen. Diese können mit Hilfe von: {nil:→list, :: :nat×list→list} definiert werden können.

Für nicht freie Datentypen ist neben den Funktionssymbole eine Menge von Gleichungen notwendig, welche die Gleichheit definieren. Ein Beispiel ist die Menge der ganzen Zahlen, welche durch {0:→int,

s:int→int, p:int→int} sowie den Gleichungen: s(p(X))=X und p(s(X))=X, beschrieben werden kann. Ein anderer nicht freier Datentyp ist set: {empty:→set, insert:nat,set→set} mit den Gleichungen: insert (X,insert(X,set))=insert(x,set) und insert(X,insert(Y,set))= insert(y,insert(x,set)).

Rekursive Definitionen

Eine Funktion f wird durch eine Menge von Gleichungen:$cond_i \Rightarrow l_i=r_i$, definiert. Hierbei ist l_i ein f-Term, d.h. $l_i=f(t_1,\ldots,t_n)$) und die Menge der Variablen in r_i ist eine Teilmenge der Variablen in l_i. Die Funktion f ist rekursiv definiert wenn, in beiden Seiten einer Gleichung ein f-Term vorkommt. $cond_i$ ist eine Konjunktion von Gleichungen und Ungleichungen über den Argumenten von f.

Strukturelle Rekursion

In der einfachsten Form rekursiver Funktionen f, gibt es in der Definition für f ein rekursives Argument, welches ein Konstruktorterm der Form: $c_i(x_1,..,x_n)$, ist. Alle übrigen Argumente sind Variablen. In den rechten Seiten, werden die Funktionen mit Konstruktoren definiert, wobei deren Argumente rekursive Aufrufe von f enthalten können. Für diese Aufrufe werden syntaktisch einfachere Argumente verwendet. Die Addition für ganze Zahlen und Konkatenation von Listen sind Beispiel hiervon.

(1) X+0=x (2) X+s(Y)=s(X+Y) (3) X+p(Y)=p(X+Y)

(1) nil<>X=nil (2) (T::X)<>Y=T::(X<>Y)

Solche Definitionen sind vollständig und eindeutig, da es für jedes Paar von Konstruktoren genau eine Gleichung gibt, welche angewendet werden kann. Weiterhin sind die rekursiven Aufrufe syntaktisch einfacher als die linke Seite der Gleichung.
Um Aussagen über solche Funktionen zu beweisen, kann für jede Regel, welche nicht rekursiv ist ein Basisfall und für jede rekursive Definition ein Induktionsschritt verwendet werden.

$$P(0) \wedge P(x) \vdash P(s(x)),\ P(x) \vdash P(p(x))$$

Bei struktureller Rekursion über nichtfreie können Inkonsistenzen entstehen. Es ist möglich, dass Gleichungen über Konstruktorterme entstehen die, aus den Konstruktorgleichungen nicht folgen.
Ein Beispiel wären die Ganzen Zahlen mit einer Funktionsdefinition für +, welche inkonsistent ist.

(1) X+0=X (2) X+s(Y)=s(X+Y) (5)X+p(Y)=**0**

Nach den Konstruktorregeln ist zu prüfen ob x+s(p(y))=x+p(s(x), da s(p(y)) und p(s(y)) gleich sind.
Nach Definition gilt: x+p(s(0))=0 und x+ s(p(0))=s(x+p(0))=s(0) aber: 0≠s(0). Damit besteht bei nicht eindeutig definierten Funktionen über nicht freien Konstruktoren die Gefahr, dass falsche Aussagen abgeleitet werden.

Nicht Strukturelle Rekursion

Neben der Definition von Funktionen über Konstruktoren, existieren weitere Formen von Rekursion. Zum Beispiel können Funktionen auch über Destruktoren definiert werden. Eine Definition der Addition in den natürlichen Zahlen ist:
(1) X=0⇒X+Y=Y (2) X≠0 ⇒ X+Y= succ(pred(X)+Y) (3) pred(0)=0 (4) pred(succ(X))=Y
Die rekursiven Aufrufe müssen auch nicht Argumente der Konstruktoren sein.
(1) quicksort(nil)=nil (2)quicksort(H::T)=quicksort(lesseq(H,T))<>quicksort(H ::greater(H,T))
Hierbei liefert lesseq alle Elemente in T, welche kleiner oder gleich als H sind und greater liefert alle Größeren.
Ein besondere Klasse nicht struktureller Rekursion sind rekursiv gegenseitig definierte Funktionen (mutual rekursion). Ein Beispiel hiervon ist:
(1) even(succ(X))=odd(Y) (2) odd(succ(X))=even(Y) (3)even(0)=⊥ (4) odd(0)=⊤
Wollte man hierbei die Aussage P: $\forall x \in nat \forall y \in nat$ (even(x)=⊤ and even(y)=⊤) ⇒ even(x+y)=⊤ beweisen so würde Peano Induktion (die Definition über s suggeriert den Erfolg der vollständigen Induktion) nicht zum Erfolg führen. Den Basisfall kann man leicht zeigen indem eine Variable 0 gesetzt wird. Der Induktionsschritt kann aber nicht bewiesen werden:
(even(x)=⊤ and even(Y)=⊤ ⇒even(x+Y)=⊤)

⊢ (even(succ(x))=⊤ ∧ even(y)=⊤ ⇒ even(succ(x)+y)=⊤)
⊢ (even(succ(x))=⊤ ∧ even(y)=⊤ ⇒ even(**succ(x+y)**)=⊤) (Regel der Addition)
⊢ (even(succ(x))=⊤ ∧ even(y)=⊤ ⇒**odd**(x+y) =⊤) (1)
⊢ (**odd**(x)=⊤ ∧ even(y)=⊤ ⇒ odd(x+y) =⊤) (1)

Weiteres Ersetzen ist nicht möglich und die Induktionsannahme war an keiner Stelle anwendbar. Die Aussage kann aber mit einem anderen Induktionsschema bewiesen werden:

$$P(0,y) \wedge P(succ(0),Y) \wedge P(x,y) \vdash P(succ(succ(x)),y)$$

Im Abschnitt Beweistechniken wird gezeigt, wie die einfachste Form (wie in even und odd) von gegenseitiger Rekursion umgeformt werden kann, um anschließend bessere Axiome aus den Gleichungen zu gewinnen.

Dualität zwischen Rekursion und Induktion

Auf die gleiche Weise, in der das Verhalten einer Funktion mit Hilfe der Rekursion beschrieben werden kann, ist es auch möglich eigenschaften dieser Funktion durch Induktion zu zeigen. Definiert man even in etwas abgewandelter Form:

$$(1)\ even(succ(succ(X)))=even(X) \quad (2)\ even(0)=\top \quad (3)\ even(succ(0))=\bot$$

dann werden für eine Aussage P über even, die Regeln (2) und (3) zur Konstruktion der Basisfälle dienen. Die erste Gleichung beschreibt einen Induktionsschritt indem aus P(x) P(s(s(x)) geschlossen wird. Das entsprechende Induktionsschema für diese Definition ist: $P(0) \wedge P(succ(0)) \wedge P(x,Y) \vdash P(succ(succ(x)))$.
In [3], [4], [5] und [7] wird mit Hilfe der Cover Set Methode gezeigt, wie direkt aus terminierenden Ersetzungsregeln Induktionsaxiome gewonnen werden können.

Notwendigkeit der Induktion

Rekursion erlaubt es das Verhalten einer Funktion für einen bestimmten Datentyp zu spezifizieren. Induktion ermöglicht es, Aussagen über einer Funktion nur für einen bestimmten Datentyp zu beweisen und damit Aussagen zu beweisen, welche nicht in allen Modellen der Gleichungen gelten.
Zum Beispiel ist die Aussage:

$$\forall x \in nat.\ 0+x=x$$

wahr für den Datentyp nat mit folgender Definition der Addition:

$$(1)\ X+0=X \quad (2)\ X+succ(Y)=succ(X+Y)$$

Dies kann mit der Peano Induktion $0+0=0 \wedge 0+x=x \vdash 0+succ(x)=succ(x)$ bewiesen werden. Der Basisfall ist eine Instanz der Regel (1). Der Induktionsschritt kann wie folgt bewiesen werden:

$0+x=x$ $\vdash 0+succ(x)=succ(x)$
$\vdash$ **succ**$(0+x)=succ(x)$ (2)
$\vdash succ(\mathbf{x})=succ(x)$ nach Annahme

Neben der Interpretation von nat, als die Menge der Terme über 0 und succ, und + als Funktion über diesen Termen gibt es weitere Interpretationen, welche die Gleichungen (1) und (2) erfüllen. Ein derartiges Model ist zum Beispiel der Datentyp nat', welcher gleich der Menge der Terme über succ, 0 und einer weiteren Konstanten 0', weiterhin wird + als Funktion über nat' interpretiert, in der genau die Gleichungen gelten, die aus (1) und (2) folgen. In diesem Modell ist die Gleichung 0+0'=0' falsch, da mit (1) und (2) keine Äquivalenz über Termen definiert wird, welche 0' als Teilterm besitzen. Damit ist die Aussage:
$\forall x \in nat'.\ 0+x=x$ falsch in diesem Modell.

Würde man aber ein vollständiges Termersetzungssystem bilden, welches die Gleichung 0+x=x beweist indem, geprüft wird ob die Normalformen von 0+x und x identisch sind, dann würden sowohl $\forall x{:}nat\ 0+x=x$ als auch $\forall x{:}nat'\ 0+x=x$ gelten. Dies ist aber falsch, sodass kein vollständiges Termersetzungssystem für die Gleichungen (1) und (2) die Gleichung 0+x=x beweisen kann. Neben einfachem Ersetzen von Gleichem durch Gleiches sind also stärkere Beweisverfahren, wie Induktion nötig.

Beweistechniken

Ersetzungsregeln

In der Regel reicht es in Induktionsbeweisen nicht aus ein Induktionsaxiom aufzustellen zu können um anschließend die Induktionsannahme im Induktionsschluss wahr zu setzen. Wie in dem Beispiel, zur Kommutativität der Addition in den natürlichen Zahlen zu sehen war, sind zur Verwendung der Induktionsannahme zunächst eine Menge von Ersetzungsschritten notwendig. In der einfachsten Form werden die Funktionsdefinitionen hierfür verwendet.

$$\frac{l \rightarrow r \wedge E[l\varphi]}{E[l\varphi \leftarrow r\ \varphi]} \qquad \frac{a+s(b) \rightarrow s(b)+a,\ s(x)+Y=Y+s(x)}{s(x)+Y=s(Y)+x}$$

Lemmas als Rewrite Rules

Eine wichtige Heuristik in Induktionsbeweisen ist es, Lemmata zu spekulieren und zu Beweisen. Diese können dann ebenfalls als Ersetzungsregeln verwendet werden. Definiert man z.B. die Addition durch:

(1) $X+0 \rightarrow X$ (2) $X+succ(Y) \rightarrow succ(X+Y)$

dann wäre ein Hilfreiches Lemma: $succ(X)+Y=succ(X+Y)$. Dieses kann anschließend zu einer terminierenden Ersetzungsregel gerichtet werden. Problematisch sind aber Lemmata, wie die Kommutativität: $x+y=y+x$, welche nicht zu einer terminierenden Regel geformt werden können.

Implicational Rewrites

Bei dieser Art von Ersetzungsregeln wird die Kongruenz ausgenutzt. Für den Fall, dass $X_1=Y_1 \wedge \ldots \wedge X_n=Y_n$ gilt, dann folgt daraus $f(X_1,\ldots,Y_n)=f(X_1,\ldots,Y_n)$ für alle Funktionssymbole.
Im Falle von freien Konstruktoren gilt auch die Umkehrung. Diese Regeln können im Beweis verwendet werden um den Schluss an die Hypothese anzunähern. Dies ist zum Beispiel in dem Beweis der Assoziativität der Konkatenation hilfreich.

(1) $nil \diamond Y \rightarrow Y$ (2)$(T::X) \diamond Y \rightarrow T::(X \diamond Y)$

Hierbei ist:: ein freier Konstruktor ist und somit kann die Regel: (3)$T::X=T::Y \Rightarrow X=Y$ verwendet werden kann. Die Assoziativität kann mit:

$nil \diamond (Y \diamond Z)= (nil \diamond Y) \diamond Z$

$t \diamond (Y \diamond Z)= (t \diamond Y) \diamond Z \vdash h::t \diamond (y \diamond z)= (h::t \diamond y) \diamond z$

bewiesen werden. Der Basisfall lässt sich leicht mit der Regel (1) in eine Tautologie umformen.
Der Induktionsschritt kann mit der Regel (2) und der Regel für freie Konstruktoren bewiesen werden.

$(t \diamond Y) \diamond Z=t \diamond (Y \diamond Z)$	$\vdash((h::t) \diamond y) \diamond z=(h::t) \diamond (y \diamond z)$	
	$\vdash((h::t) \diamond y) \diamond z=\mathbf{h::}(t \diamond (y \diamond z))$	(2)
	$\vdash \mathbf{h::}((t \diamond y) \diamond z)=h::(t \diamond (y \diamond z))$	2*(2)
	$\vdash \mathbf{h=h} \wedge (t \diamond y) \diamond z=t \diamond (y \diamond z)$	(3)

Die linke Seite der Konjunktion ist eine Instanz der Induktionsannahme und kann, wie die Gleichung t=t, wahr gesetzt werden. Für nicht freie Konstruktoren ist diese Regel nicht zulässig. Zum Beispiel gilt für Mengen:

insert(t, inset(t, empty))=inset(t, empty) aber insert(t, empty)≠empty

Fertilization

Wenn der Ripplingprozess erfolgreich terminiert ist die Hypothese (mit Substitution) im Schluss enthalten und kann entfernt werden (wahr gesetzt werden).

$$\frac{IH \Rightarrow IC[IH\varphi]}{IH \Rightarrow IC[IH\ \varphi \leftarrow \top]}$$

Dies wird strong fertilization genannt. Diese Form kann im oberen Beispiel angewendet werden.

$t \diamond (Y \diamond Z)=(t \diamond Y) \diamond Z$	$\vdash h=h \wedge\ t \diamond (y \diamond z)=(t \diamond y) \diamond z$	
	$\vdash h=h \wedge \top$	

Mitunter kann diese Situation mit Rewriting nicht erreicht werden. Sollte der Schluss aus einer Gleichung bestehen in der auf jeder Seite die Annahme als Teilterm vorkommt, dann kann die Annahme als Ersetzungsregel verwendet werden.

$\underline{IH1=IH2 \Rightarrow IC1[IH1\varphi]=IC2}$ $\underline{IH1=IH2 \Rightarrow IC1=IC2[IH2\varphi]}$

$IH1=IH2 \Rightarrow IC1[IH1\varphi \leftarrow \top]=IC2$ $IH1=IH2 \Rightarrow IC1=IC2[IH2\varphi \leftarrow \top]$

Diese Form wurde bereits im Abschnitt über mehrere universelle Variablen verwendet:

x+Y=Y+x	⊢succ(x)+y=y+succ(x)	
	⊢succ(x)+y=**succ(y)+x**	(2)
	⊢succ(x)+y=**x+ succ(y)**	ersetzen von Y durch s(y) und fertilization
	⊢succ(x)+y=succ(x)+y	(2)

Diese Form der Fertilization ist aufwändiger, kann aber in Situationen eingesetzt werden in denen die starke Form nicht anwendbar ist.

Terminierung

Eine Möglichkeit eine Vermutung zu Beweisen, ist das Anwenden von Ersetzungsregeln bis keine Regel mehr anwendbar ist. Die Frage ob solch ein System terminiert ist äquivalent zum Halteproblem und damit unentscheidbar.
Vom besonderen Interesse, sind Ersetzungsregeln, welche als Funktionsdefinitionen verwendet werden. Um festzustellen, ob solch System terminierend ist, recht es nicht aus zu prüfen ob, die rechten Regelseiten bezüglich einer fundierten Relation kleiner sind als die linken. Zum Beispiel ist in: x+(y+z) →x+x, die rechte Regelseite bzgl. der Anzahl der Additionssymbole oder der Länge kleiner als die Linke. Beides sind fundierte Relationen. Wendet man aber auf den Term: ((a+a)+a)+(a+a) die Regel an, so ergibt sich ((a+a)+a)+(a+a)+ ((a+a)+a)+(a+a), also ein größerer Term. Um aus den Regeln eines Termersetzungssystems, Terminierung festzustellen, ist zusätzlich die Monotonie notwendig. Termordnungen sind fundierte Relationen, welche monoton ($t<s \Rightarrow f(\ldots,t,\ldots)< f(\ldots,s,\ldots)$)sind.
Ein Termersetzungssystem ist genau dann terminierend, wenn es eine Termordnung < gibt, sodass und für jede Substitution φ gilt: $\varphi(l)>\varphi(r)$ für alle Regeln l→r. Diese Eigenschaft ist aber schwer nachzuweisen, da jede Substitution untersucht werden muss.
In Simplifikationsordnungen gilt zusätzlich die Teiltermeigenschaft, d.h. $t_i<f(t_1,\ldots,t_n)$. Gilt für diese Ordnungen in allen Regeln l>r, dann ist das System terminierend.

Wichtige Beispiele für Simplifikationsordnungen sind die rekursive und die lexikographische Pfadordnung. Für beide Formen ist eine Ordnung der Funktionssymbole (Präzedenz) nötig.

- Rekursive Pfadordnung:
  Ein Term wird kleiner, wenn ein Unterterm u durch u' ersetzt wird, dessen äußeres Funktionssymbol kleiner ist als das von u und dessen Unterterme kleiner als u sind.
  $f(t_1,\ldots,t_n)<g(s_1,\ldots,s_m)$ gdw:
  (1) $s_i \geq f(t_1,\ldots,t_n)$ für ein j oder
  (2) $f<g$ und $g(s_1,\ldots,s_m)< t_i$ für alle i oder
  (3) f=g und für die Multimenge $\{s_1,\ldots,s_m\}$, $\{t_1,\ldots,t_n\}$ gilt:
  $\{s_1,\ldots,s_m\} \neq \{t_1,\ldots,t_n\} \wedge \forall x \in \{t_1,\ldots,t_n\} \setminus \{s_1,\ldots,s_m\} \exists y \in \{s_1,\ldots,s_m\} \setminus \{t_1,\ldots,t_n\}.\ x<y$
  Dies bedeutet Argumente dürfen nicht größer werden.
- Lexikographische Pfadordnung
  Für die dritte Regel spielt die Reihenfolge der Argumente eine Rolle.
  (3) f=g und $(t_1,\ldots,t_n)<_{lex}(s_1,\ldots,s_m)$

Das Beispiel: (1) X+0→X (2)X+succ(Y)→succ(X+Y), ist bezüglich der Präzedenz: 0<succ<+ als lexikographische und rekursive Pfadordnung terminierend. Fügt man aber das Lemma: (X+Y)+Z→X+ (Y+Z) hinzu, dann gilt die rekursive Pfadordnung nicht, denn (Y+Z) ist weder kleiner als Z noch kleiner als (X+Y) ist. Fügt man das Lemma: 0+succ(X)→succ(X)+0 hinzu, dann gilt die lexikographische Pfadordnung nicht mehr, denn 0 ist kleiner als succ(x) (mit der Präzedenz: succ<0<+ bleibt sie erhalten). Der Vorteil dieser Ordnungen ist, dass Terminierung direkt an den Regeln, und nicht an allen Substitutionen, festgestellt werden kann. Weiterhin muss nur eine Ordnung der Funktionssymbole gefunden werden.

Destruktoreliminierung

Fertilization basiert darauf, dass die Induktionsannahme als Teilausdruck des Induktionsschluss ist. Diese Situation kann erreicht werden indem auf den Schluss Ersetzungsregeln angewendet wurden. Dies setzt eine Indutionsregel in Konstruktorform vorraus. Im Falle der Destruktorform könnte man annehmen, dass der duale Prozess (rewriting und fertilization auf Annahme) verwendet werden könnte. Aber die Regel:

$$\frac{IH[IC\varphi] \Rightarrow IC}{IH[IC\varphi \leftarrow \top] \Rightarrow IC}$$

ist logisch nicht korrekt. Ein Beweissystem, welches Induktionsaxiome automatisch aus Regeln wie: $x!=0 \Rightarrow x+pred(y) \rightarrow pred(x+y)$ gewinnt (wobei pred ein Destruktor ist), könnte versuchen den Induktionsschritt in eine äquivalente Aussage zu überführen in der keine Destruktoren in der Annahme vorkommen. Heirzu sind lediglich zwei Schritte notwendig.

- Anwenden der Regeln $x \rightarrow c(d_1(x),...,d_n(x))$ auf jedes vorkommen von x im Induktionsschluss. Die Regel muss nicht auf Vorkommen von $d_i(x)$ angewendet werden da: $d_i(c(d_1(x),...,d_n(x)))=d_i(x)$
- Ersetzen aller $d_i(x)$ durch y_i

Beispiel für die Peano Induktion:

$x \neq 0 \wedge P(pred(x)) \vdash P(x)$
$\Leftrightarrow$**succ(pred(x))**$\neq 0 \wedge P(pred($**succ(pred(x))**$)) \vdash P($**succ(pred(x))**$)$
$\Leftrightarrow succ(pred(x)) \neq 0 \wedge P($**pred(x)**$) \vdash P(succ(pred(x)))$
$\Leftrightarrow s($**y**$) \neq 0 \wedge P($**y**$) \vdash P(s($**y**$))$
$\Leftrightarrow P(y) \vdash P(s(y))$

Auf dieses Axiom kann nun Rewriting und Rippling auf den Induktionsschluss angewendet werden.

Eliminieren von einfacher Mutual Rekursion

Sollten f und g gegenseitig definiert sein, dann ist es schwierig hilfreiche Induktionsaxiome aus den Regeln zu bilden, da es keine Selbstaufrufe gibt. Ein Beispiel hierfür ist even.

(1) $even(s(X)) \rightarrow odd(X)$ (2) $odd(s(X)) \rightarrow even(X)$ (3)$even(0) \rightarrow \top$ (4)$odd(0) \rightarrow \bot$

In einfachen Fällen gegenseitiger definierter Funktionen f und g, in denen in keiner rechten Regelseite f und g vorkommen, kann diese Art der Rekursion umgeformt werden. Für das even Beispiel muss nur aus der Regel (1) der Term odd(X) mit den linken Regelseiten für odd unifiziert werden. In diesem Fall ist dies mit der zweiten Regel: $odd(X)=odd(s(X1) \Rightarrow x=s(X1)$ möglich. Anschließend kann auf $even(s(X)) =odd(X)$ diese Substitution angewendet werden: $even(s(s(X_1)) \rightarrow odd(s(X_1))$. Nun lässt sich auf: odd(s(X)) die 2. Regel anwenden sodass die Regel: (5) $even(s(s(X_2)) \rightarrow even(X_2)$ entsteht. Auf die gleiche Weise entsteh die Regel (6) $even(s(0)) \rightarrow \bot$, wenn odd(X) mit der linken Regelseite der Regel (4) unifiiziert wird. Um nun eine Aussage wie $(even(x)=\top$ and $even(y)=\top) \Rightarrow even(x+y)=\top$ zu beweisen, können aus den Regeln: (3), (5), (6) ein Axiom gewonnen werden, indem mit den Argumenten der Regeln (3) und (6) die Basisfälle P(0,y) und P(s(0),y) erzeugt werden. Die rekursive Regel (5) definiert den Induktionsschritt, indem das Argument der Funktion für den Induktionsschluss verwendet wird und das Argument im rekursiven Aufruf für die Annahme: $P(x,Y) \vdash P(s(s(x)),y)$.

A: $(even(0)=\top \wedge even(y)=\top \Rightarrow even(0+y)=\top) \wedge$
$(even(s(0))=\top \wedge even(y)=\top \Rightarrow even(s(0)+y)=\top) \wedge$
$(even(x)=\top \wedge even(Y)=\top) \Rightarrow even(x+Y)=\top) \vdash$
$(even(s(s(x)))=\top \wedge even(y)=\top \Rightarrow even(s(s(x))+y)=\top)$

Der erste Basisfall lässt sich umschreiben in:

$(even(y)=\top \Rightarrow even($**y**$)=\top)$

Der zweite Basisfall kann in:

$(\bot=\top \wedge even(y)=\top \Rightarrow even(s(0)+y)=\top)$
$(\bot \Rightarrow even(s(0)+y)=\top)$

umgewandelt werden. Beide Aussagen sind Tautologien. Der Induktionsschritt kann ebenfalls zu einer Tautologie umgeformt werden.

$(even(x)=\top$ and $even(Y)=\top \Rightarrow even(x+Y)=\top)$
$\vdash (even(s(s(x)))=\top \wedge even(y)=\top \Rightarrow even($**s(s(x+y))**$)=\top)$
$\vdash (even($**x**$)=\top \wedge even(y)=\top \Rightarrow even($**x+y**$)=\top)$ 2*(5)

$\vdash \top$ (Fertilization)

Diese einfache Umwandlung ist aber in Fällen in denen f zusammen mit g in rechten Regelseiten vorkommt, nicht möglich. Für solche Fälle werden in [4] unrolled cover sets eingeführt.

Entscheidungsprozeduren

Viele Probleme, welche durch Induktionsbeweise gelöst werden sollen, sind Teil einer entscheidbaren Klasse in der Gödels Unvollständigkeitssatz nicht gilt. Da Rippling und Fertilization nur Heuristiken sind, ist es möglich, dass eine Aussage mit Induktion nicht bewiesen werden kann, obwohl sie entscheidbar ist. Deshalb sind Entscheidungsprozeduren ein wichtiger Teil von Induktionsbeweissystemen.

Tautology Check

Viele Teilprobleme können in propositionelle Logik verallgemeinert und durch Anwenden von Regeln der Logik vereinfacht werden. Dazu müssen eventuell Teilaussagen durch Variable ersetzt werden. Sind die Teilprobleme Tautologien, dann können diese wahr gesetzt werden. Für den Test der Tautologie können z.B. Ordererd Binary Decision Diagramms verwendet werden, welche Normalformen aus einer Aussage bilden. In [6] ist eine Form von OBDD's beschrieben, welche Aussagenlogische Formeln in if-then-else Sätze (if(A,B,C)) um. Dabei werden zunächst Regeln zur Bildung von if-then-else Sätzen aus verwendet und anschließend Regeln zu Bildung der Normalform

Regeln zur Transformation:	Regeln zur Bildung der Normalform:
$A \wedge B \rightarrow if(A,B,\bot)$	$if(\top,A,B)\rightarrow A$
$A \vee B \rightarrow if(A,\top,B)$	$if(\bot,A,B)\rightarrow B$
$\neg A \rightarrow if(A,\bot,\top)$	$if(A,B,B)\rightarrow B$
$A \Rightarrow B \rightarrow if(A,B,\top)$	$if(if(A,B,C),D,E)\rightarrow if(A,if(B,D,E),if(C,D,E))$
	$if(a,A,B)\rightarrow if(a,A[a\leftarrow\top], B[a\leftarrow\bot])$ für Variablen
	$a\rightarrow if(a,\top,\bot)$ für Blätter

Im Beispiel: $(y=0 \wedge x+y=x) \vee (y=s(z) \wedge x+y=s(x+z))$

ist der Induktionsanfang mit y=0:

$(0=0 \wedge x+0=x) \vee (0=s(z) \wedge x+0=s(x+z))$

Die Umformung in propositionelle Logik ergibt:

$(\top \wedge \top) \vee (\mathbf{a} \wedge \mathbf{b})$ mit den Variablen a und b

dies würde umgewandelt werden in:

$if(\top \wedge \top, \top, (a \wedge b))$

$if(\mathbf{if(\top,\top,\bot)},\top, \mathbf{if(a,b,\bot)})$

die Umwandlung in die Normalform ergibt:

$if(\mathbf{\top},\top, if(a,b,\bot))$

$\mathbf{\top}$

Wodurch der Induktionsanfang bewiesen ist.

Kongruenzabschluss

Eine andere wichtige Prozedur ist die Berechnung von Gleichheiten, denn wenn bekannt ist, dass zwei Terme gleich sind, kann dies zur Vereinfachung von Beweisen nützlich sein. Nelson und Oppen haben in [8] für die quantorenfreie und uninterpretierte Theorie der Gleichheit eine Entscheidungsprozedur entwickelt die zu einer gegebenen Konjunktion, von Gleichungen und Ungleichungen, entscheidet ob diese lösbar ist. Dazu wird der Kongruenzabschluss der Gleichheit mit Hilfe eines Graphen gebildet. Anschließend können mit Hilfe der Kongruenzen Gleichungen vereinfacht werden oder Ungleichungen widerlegt werden.

Bsp: $succ(a)=a \wedge succ(succ(a))+a \neq a+a$

Egal wie die Symbole succ, +, und a interpretiert werden, die Konjunktion kann keine Lösung haben.

Denn: $succ(a)=a \Rightarrow succ(succ(a)=succ(a) \Rightarrow a=succ(succ(a)) \Rightarrow$

$succ(succ((a))+a=a+a,$

womit die Konjunktion widerlegt ist. Andererseits ist dies eine Beweisprozedur für:

$succ(a)=a \wedge \mathbf{succ(succ(a))+a = a+a}$

in der uninterpretierten Theorie. Für den Fall, dass es für succ, + und a eine Gleichheitsrelation durch ein Termersetzungssystem festgelegt ist, würde sich die Aussage auf: succ(a)=a verkürzen.
Kongruenzabschlüsse werden auch für Entscheidungsprozeduren anderer Theorien, wie der Presburger Arithmetik verwendet.

Presburger Arithmetik (Lineare Arithmetik)

Diese Arithmetik ist eine entscheidbarer quantorenfreie Theorie der Addition von ganzen Zahlen.Sie besteht aus Gleichungen und Ungleichungen zwischen Termen, welche aus Addition aufgebaut werden (der rationale Teil ist ebenfalls entscheidbar). Eine Entscheidungsprozedur liefert für eine Menge von Gleichung entweder eine Lösung in Form von einer Variablensubstitution oder zeigt, dass die Gleichungen nicht erfüllbar sind. Nimmt man aus dem Beispiel für Tautology Checks die erste Teilformel weg:

$$\{y=succ(z), x+y=succ(x+z)\}$$

dann kann Induktionsanfang mit y=0 nicht bewiesen werden da:

$$\{0=succ(z), x+0=succ(x+z)\}$$

0=succ(z) keine Lösung hat.

Der Aufwand zur Lösung von solchen Gleichungen ist aber 3-fach exponentiell. Für praktische Beispiele ergibt sich aber eine deutlich schnellere Durchschnittsgeschwindigkeit. In [3] ist gezeigt, wie mit Hilfe einer Entscheidungsprozedur, Induktionsaxiome für Funktionen gewonnen werden können, welche durch der Presburger Arithmetik definiert sind (wie even). Weiterhin kann mit solch einer Prozedur, geprüft werden ob Funktionen vollständig definiert sind.

Kombinieren von Entscheidungsprozeduren

Wenn das Gleichungssystem nicht Teil einer entscheidbaren Theorie ist, dafür aber aus zwei disjunkten entscheidbare Theorien T_1 und T_2 besteht, dann können deren Entscheidungsprozeduren kombiniert werden.
Für eine Aussage P über T_1 und T_2 mit den Signaturen $S_1=(\{1,2\})$ und $S_2=(\{f\})$ wird P in eine separate Form: P1 and P2, überführt. Dabei ist P_i eine Aussage über T_i. Ist zum Beispiel P:

$$1<=x \wedge x<=2 \wedge f(x)=f(1) \wedge f(x)\neq f(2),$$

dann werden zwei Aussagen P1 und P2 berechnet, wobei: $P1 \wedge P2 \Leftrightarrow P$. In diesem Fall entstehen:

$$P1= 1<=x \wedge x<=2 \wedge z=1 \text{ and } y=2,$$
$$P2= f(x)=f(z) \wedge f(x)\neq f(y)$$

Nun könnte man auf P_1 und P_2 die Entscheidungsprozeduren SAT_1 und SAT_2 anwenden. Wenn P erfüllbar ist dann sind auch P_1 und P_2 erfüllbar. Für die Umkehrung müssen aber die Bedingungen der geteilten Variablen beachtet werden, in diesem Fall: x,y und z. Wenn es eine Äquivalenzrelation E über den geteilten Variablen gibt sodass $P1 \cup E$ und $P2 \cup E$ erfüllbar sind dann ist auch P erfüllbar. Gibt es keine solche Relation dann ist P unerfüllbar. In diesem Fall, gibt es Lösungen für E={x=z} Eine Beschreibung des Algorithmus ist z.B. in [2] angegeben.

Literaturliste

[1] Alan Bundy: Automation of Proof by Mathematical Induction
http://dream.inf.ed.ac.uk/publications/96-98/induction.ps

[2] Folien für die Nelson Oppen Kombination
http://www.cs.nyu.edu/courses/fall03/G22.3033-002/lec/lec9_h4.pdf

[3] Deepak Kapur and M. Subramaniam: New Uses of Linear Arithmetic in Automated Theorem Proving by Induction http://www.cs.unm.edu/~kapur/mypapers/spec.jar.95.ps.gz

[4] Deepak Kapur and M. Subramaniam: Automating Induction over Mutually Recursive Functions

[5] Vorlesung Theorembeweisen in der Induktiven Theorie
http://agent.informatik.uni-kl.de/ag-avenhaus/chronologisch/ws2003-2004/ind/

[6] Vorlesung Maschinelles Beweisen
http://www.informatik.uni-ulm.de/ki/Edu/Vorlesungen/Maschinelles.Beweisen/WS0203/

[7] Deepak Kapur and Hantao Zhang: Automating Induction: Explicite vs. Inductionless

[8] Greg Nelson and Derk C. Oppen: Fast Decision Procedures Based on Congruence Closure
http://raw.cs.berkeley.edu/Papers/NelsonOppenCong.pdf