1 Motivation

1) Was "macht" ein Programm mit dem Arbeitsspeicher?
Ein Programm "rutscht" auf dem Arbeitspeicher rum und verändert die Inhalte mancher Speicherstellen.
Ein Programm entspricht also einer Abbildung, die die Werte bestimmter Speicherstellen ändert.

Zunächst wird die Syntax der einfachen Programmiersprache (IMP) definiert.
Die Eindeutigkeit der Syntax wird durch die Klammerung (z.B. [a+b]) erreicht.
Auf der Syntax aufbauend wird dann die operationale Semantik eingeführt.

Bei der Definition der Syntax und Semantik werden nur "induktiv definierte Mengen" verwendet. Diese bilden die Basis der Ausarbeitung.
"Ein weiterer Grund ist, ein Bewußtsein dafür zu schaffen, daß in der Informatik fast uberall nur mit Wasser gekocht wird, wobei das Wasser die Konzepte des induktiven Definierens und Beweisens sind." (Zitat Prof.Kindler).

In dieser Ausarbeitung werden rekursive Forneln entwickelt ("Semantiksatz der arithmetischen Ausdrücke", " Semantiksatz der boolschen Ausdrücke", "Semantiksatz der Anweisungen"), mit denen man den Wert von arithmetischen und boolschen Ausdrücken "berechnen" kann.
Der "Rekursionssatz" (zu Beginn der Ausarbeitung) wird benötigt, um den "Semantiksatz der arithmetischen Ausdrücke" und den " Semantiksatz der boolschen Ausdrücke" zu beweisen.

2) Beispiel:
Die Variable i sei mit dem Wert 2 belegt, d.h. h(i) = 2
Dann wird das Programm
[while $0 \leq i$ do i: = i-1]
gestartet.
Danach hat i den Wert -1

Formal:
N([while $0 \leq i$ do i: = i-1] ,h) = h[-1/i]

wobei h[-1/i] die Belegung bedeutet, die i den Wert -1 zuordnet.
Genauere Erklärungen dazu in der Ausarbeitung unten.

3) Bemerkung:
Diese Ausarbeitung bezieht sich auf das Skript "Formale Semantik" von Prof. Kindler.

Dieses Skript ist noch im Aufbau und sicherlich noch mit einigen Fehlern behaftet.

2 Vorbereitungen

2.1 Definition

X_A und X_B, I sind Grundmengen.
r : |N \{0} --> |N \{0}
Für alle i>0 sind $W_{i,1}$ und $W_{i,2}$ Teilmengen, konkret:
$W_{i,1} \subset \bigcup_{i \in N \setminus \{0\}} X\_A^i$ x X_A

$W_{i,2} \subset \bigcup_{i \in N \setminus \{0\}} X\_B^i$ x X_B

$\emptyset \neq$ R_2_Ax $\subset \{ (\emptyset,(a,b)) \mid a \in X\_A \wedge b \in X\_B\}$
$R_{i,1} := \{ (\{a_1, ..., a_{r(i)}\},a) \mid i \in I \wedge ((a_1, ..., a_{r(i)}),a) \in W_{i,1}$
$R_{i,2} := \{ (\{b_1, ..., b_{r(i)}\},b) \mid i \in I \wedge ((b_1, ..., b_{r(i)}),b) \in W_{i,2}$

Dann wird mit
$R\_2 := \{ (\{(a_1,b_1), ..., (a_{r(i)}, b_{r(i)})\},(a,b)) \mid i \in I \wedge (\{a_1, ..., a_r\},a) \in W_{i,1} \wedge (\{b_1, ..., b_r\},b) \in W_{i,2} \}$
$\cup$ R_2_Ax

eine Regelmenge über X_A x X_B bezeichnet.

$R1(R\_2) := \{ (\{a_1, ..., a_{r(i)}\},a) \mid i \in I \wedge ((a_1, ..., a_{r(i)}),a) \in W_{i,1}\} \cup \{(\emptyset,a) \mid \exists b\ (\emptyset,(a,b)) \in R\_2\}$

Bemerkung:
Um die Notation nicht zu komplex werden zu lassen, wird im Folgenden statt
r(i) meist nur r geschrieben.

2.2 Definition monotektonisch

R ist eine Regelmenge über X.
I(R) ist monotektonisch : <==>
$\forall x \in I(R)\ \exists$ genau ein $Y \subset I(R)\ (Y,x) \in R$, d.h:
$\forall x \in I(R)\ \forall Y \subset I(R)\ \forall Y' \subset I(R)$
$(Y,x) \in R$ und $(Y',x) \in R$ ==> Y = Y'

2.3 Definition regeleindeutig

I(R) ist regeleindeutig :<==>
$R := \bigcup_{i \in I} R_i$ und

$x \in I(R)$ und $(Y,x) \in I(R)$ ==> $\exists$ genau ein $i \in I$ mit $(Y,x) \in R_i$

2.4 Lemma L1

Beh:
$(a,b) \in I(R\_2)$ ==> $a \in I(R1(R\_2))$

Beweis: Induktion über I(R_2)
Definiere:
$B(a_1,b_1)$: <==> $(a,b) \in I(R)$ ==> $a \in I(R1(R\_2))$

Definiere als Abkürzung:
R1 := R1(R_2)

1) zeige:
B(a,b) für alle Atome (a,b) aus R_2
Es sei: $(\varnothing,(a,b)) \in R\_2$
also: $(\varnothing,a) \in R1$, also $a \in I(R1)$

2) Induktionsvoraussetzungen IV:
$(x_1,y_1) \in I(R\_2) \wedge ... \wedge (x_r,y_r) \in I(R\_2)$ und
$B(x_1,y_1) \wedge ... \wedge B(x_r, y_r)$ und
$(\{(x_1,y_1), ..., (x_r, y_r)\},(x,y)) \in R\_2$
Zeige: B(x,y)

Nach den IV $(a_1,b_1) \in I(R\_2)$ ==> $a_1 \in I(R1)$ und ... und $(a_r,b_r) \in I(R\_2)$ ==> $a_r \in I(R1)$ folgt:
$a_1 \in I(R1)$ und ... und $a_r \in I(R1)$
Da $(\{(a_1,b_1), ..., (a_r, b_r)\},(a,b)) \in R\_2$, existiert ein i mit $((a_1, ..., a_r),a)) \in W_{i,1}$
also $(\{a_1, ..., a_r\},a)) \in R_{i,1}$, also $(\{a_1, ..., a_r\},a)) \in R1$
Da $a_1 \in I(R1)$ und ... und $a_r \in I(R1)$ und $(\{a_1, ..., a_r\},a)) \in R1$, folgt $a \in I(R1)$

Bemerkung:
Im Allgemeinen gilt nicht:
$a \in I(R1(R\_2))$ ==> $(a,b) \in I(R\_2)$

2.5 Lemma L2

Voraussetzungen:
I(R1(R_2)) ist regeleindeutig und monotektonisch und D_1(R_2) ist rechtseindeutig und $W_{i,1}$ ist für alle i>0 linkseindeutig und $W_{i,2}$ ist für alle i>0 rechtseindeutig.

Behauptung:
I(R_2) ist rechtseindeutig

Beweis: Induktion über I(R_2)
Definiere:
B(a,b) : <==> (a,b)∈I(R_2) und (a,b')∈I(R_2) ==> b = b'

1) zeige:
(∅,(a,b))∈R_2 ==> B(a,b)
Es sei: (∅,(a,b))∈R_2
also: b ist eindeutig

2) zeige:
Aus den Induktionsvoraussetzungen (IV)
$(a_1,b_1)\in I(R\_2) \wedge ... \wedge (a_r,b_r)\in I(R\_2)$ und
$B(a_1,b_1) \wedge ... \wedge B(a_r, b_r)$ und
$(\{(a_1,b_1), ..., (a_r, b_r)\},(a,b))\in R\_2$
folgt: B(a,b)

Es sei: (a,b)∈I(R_2) und (a,b')∈I(R_2)
Da (a,b')∈I(R_2), existieren
$a'_1, ..., a'_r, b'_1, ..., b'_r, b' , i'$ mit:
$(\{(a'_1,b'_1), ..., (a'_r, b'_r)\},(a,b'))\in R\_2$ und
$(a'_1,b'_1)\in I(R\_2) \wedge ... \wedge (a'_r,b'_r)\in I(R\_2)$ und
$((a'_1, ..., a'_r),a) \in W_{i',1}$ und $((b'_1, ..., b'_r),b') \in W_{i',2}$
außerdem nach vorigem Lemma: $a'_1 \in I(R1(R\_2)) \wedge ... \wedge a'_r \in I(R1(R\_2))$, also:
$\{a'_1, ..., a'_r\} \subset I(R1(R\_2))$
außerdem folgt nach den IV $(a_1,b_1)\in I(R\_2) \wedge ... \wedge (a_r,b_r)\in I(R\_2)$ und dem vorigem Lemma:
$a_1 \in I(R1(R\_2)) \wedge ... \wedge a_r \in I(R1(R\_2))$, also:
$\{a_1, ..., a_r\} \subset I(R1(R\_2))$
Da $(\{(a_1,b_1), ..., (a_r, b_r)\},(a,b))\in R\_2$ existiert ein i mit:
$((a_1, ..., a_r),a) \in W_{i,1}$ und $((b_1, ..., b_r),b) \in W_{i,2}$
Damit gilt:
$(\{a'_1, ..., a'_r\},a) \in R_{i',1}$ und $(\{a_1, ..., a_r\},a) \in R_{i,1}$
Da I(R1(R_2)) monotektonisch ist, folgt: $\{a'_1, ..., a'_r\} = \{a_1, ..., a_r\}$
also:
$(\{a_1, ..., a_r\},a) \in R_{i',1}$ und $(\{a_1, ..., a_r\},a) \in R_{i,1}$
Da I(R1(R_2)) regeleindeutig ist, folgt: i = i'
also
$((a_1, ..., a_r),a) \in W_{i,1}$ und $((a'_1, ..., a'_r),a) \in W_{i,1}$
Da $W_{i,1}$ linkseindeutig, folgt
$a_1 = a'_1$ und ... und $a_r = a'_r$
Da laut IV gilt:

$B(a_1,b_1) \wedge ... \wedge B(a_r, b_r)$
folgt:
$(a_1,b_1) \in I(R\_2)$ und $(a'_1,b'_1) \in I(R\_2) \Longrightarrow b_1 = b'_1$ (weil $a_1 = a'_1$)
...
$(a_r,b_r) \in I(R\_2)$ und $(a'_r,b'_r) \in I(R\_2) \Longrightarrow b_r = b'_r$ (weil $a_r = a'_r$)
Da
$((b_1, ..., b_r),b) \in W_{i,2}$ und $((b'_1, ..., b'_r),b') \in W_{i,2}$ und
$b_1 = b'_1$ und ... und $b_r = b'_r$ und $W_{i,2}$ rechtseindeutig ist, folgt:
$b = b'$

Bemerkung:
Ohne die Linkseindeutigkeit von $W_{i,1}$ gilt der Satz nicht.
Gegenbeispiel:
$W_{1,1} = \{((a,a+1),2a+1) \mid a \text{ ist eine ganze Zahl } \}$
$W_{1,2} = \{((b1,b2),b1-b2) \mid b \text{ ist eine ganze Zahl } \}$

$R\_2 = \{ (\{(a_1,b_1), (a_2,b_2)\},(a,b)) \mid (\{a_1,a_2\},a) \in W_{1,1} \wedge (\{b_1,b_2\},b) \in W_{1,2} \}$
$\cup \{(n,n) \mid n \text{ ist eine ganze Zahl}\}$
Damit: $(8,8) \in I(R\_2)$ und $(9,9) \in I(R\_2)$
also: $(8,8) \in I(R\_2)$ und $(9,9) \in I(R\_2) \Longrightarrow (8+9,8-9) \in I(R\_2)$
$(9,9) \in I(R\_2)$ und $(8,8) \in I(R\_2) \Longrightarrow (9+8,9-8) \in I(R\_2)$
also $(17,-1) \in I(R\_2)$ und $(17,1) \in I(R\_2)$

2.6 Definition

Wenn I(R) rechtseindeutig
Dann wird definiert:
$N(a) := b \Longleftrightarrow (a,b) \in I(R)$

2.7 Rekursionssatz

Voraussetzungen:
$I(R1(R\_2))$ ist monotektonisch und regeleindeutig und $D_1(R\_2)$ ist rechtseindeutig und $W_{i,1}$ ist für alle $i>0$ linkseindeutig und $W_{i,2}$ ist für alle $i>0$ rechtseindeutig.

Behauptung:
Wenn $(a,b) \in I(R\_2)$, dann gilt:
$N(a) = W_{i,2}(N(a_1), ..., N(a_r))$, falls $\exists\, i>0\ ((a_1, ..., a_r),a) \in W_{i,1}$
$N(a) = b$, falls $(\varnothing ,(a,b)) \in R\_2$

Beweis: Induktion über I(R)
Definiere:
$B(a,b) :\Longleftrightarrow (a,b) \in I(R\_2)$ und $(a,b') \in I(R\_2) \Longrightarrow b = b'$

1) zeige:
$(\varnothing,(a,b)) \in R\_2 \Longrightarrow B(a,b)$
Es sei: $(\varnothing,(a,b)) \in R\_2$
Nach Lemma L2 ist $I(R\_2)$ rechtseindeutig.

Also gilt per Definition N(a) = b
Da nach Voraussetzung $(\varnothing,(a,b)) \in R\_2$, folgt Behauptung

2) zeige:
Aus den Induktionsvoraussetzungen (IV)
$(a_1,b_1) \in I(R\_2) \ \wedge ... \wedge \ (a_r,b_r) \in I(R\_2)$ und
$B(a_1,b_1) \wedge ... \wedge B(a_r, b_r)$ und
$(\{(a_1,b_1), ..., (a_r, b_r)\},(a,b)) \in R\_2$
folgt: B(a,b)

Es sei: $(a,b) \in I(R\_2)$ und $(a,b') \in I(R\_2)$
Nach Lemma L2 ist I(R_2) rechtseindeutig.
Also gilt per Definition N(a) = b
Da $(a_1,b_1) \in I(R\_2) \ \wedge ... \wedge \ (a_r,b_r) \in I(R\_2)$ und I(R_2) rechtseindeutig, folgt:
$N(a_1) = b_1$ und ... und $N(a_r) = b_r$
Da $(\{(a_1,b_1), ..., (a_r, b_r)\},(a,b)) \in R\_2$ existiert ein i mit:
$((a_1, ..., a_r),a) \in W_{i,1}$ und $((b_1, ..., b_r),b) \in W_{i,2}$
Da $W_{i,2}$ rechtseindeutig, gilt $b = W_{i,2}((b_1, ..., b_r))$
Also :
$N(a) = b = W_{i,2}((b_1, ..., b_r)) = W_{i,2}((N(a_1), ..., N(a_r))$

2.8 (allgemeine) Terme

K sei eine Menge von Konstanten, wie z.B. Zahlen.
RZ = {#1 ; ... ; #m } ist eine Menge von Rechenzeichen.
V = {v1 ; ... ; vn } ist eine Menge aus Variablen.
M = V ∪ RZ ∪ K ∪ { (;) }
X_Term = M*
R_Term =
{({x}, #x) | x∈ X_Term ∧ #∈RZ} ∪
{({x1, x2}, [x1 # x2]) | x1,x2∈X_Term ∧ #∈RZ} ∪ { (∅, a) | a∈K }

Regelnkurznotation: (x1,x2∈ X_Term , #∈RZ)

∅	{x1 ; x2}	{x}
--- wobei a∈K	----------	----
a	[x1 # x2]	#x

Die induktive definierte Menge Aterm := I(R_Term) ist die Menge aller allgemeinen Terme.

2.8.1 Hilfslemma HLT1

In einem Term ist die Anzahl schliessender Klammern gleich der Anzahl öffnender Klammern.

Beweis: (Induktion über Termaufbau)
B(T) :<==> Anzahl öffnender Klammern in T = Anzahl schliessender Klammern in T
1) a∈K
Da eine Konstante keine Klammern besitzt, gilt die Behauptung.

2) Induktionsvoraussetzung B(T1)∈ I(R_Term) und B(T2) ∈ I(R_Term) und #∈RZ
also folgt sofort B([T1#T2]) und B(#T1)

2.8.2 Hilfslemma HLT2

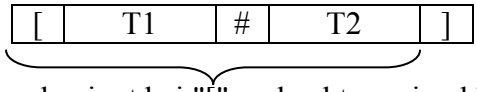
Die Zeichenkette s ist echtes Anfangsstück eines Terms T, wobei T nicht von der Form #A sein darf, kurz s < T ==>
Die Anzahl der öffnenden Klammern in s ist größer als die Anzahl der schliessenden Klammern in s.

Beweis: (Induktion über Termaufbau)
B(T) :<==> obige Aussage
1) T ist ein Atom
a) Fall1: T = a∈K
trivial

b) Fall2: T = #A
darf nach Voraussetzung nicht auftreten

2) Induktionsvoraussetzung B(T1)∈ I(R_T) und B(T2) ∈ I(R_Term) #∈RZ
also folgt sofort B([T1#T2]).
2) Es gelte Behauptung für einen Term T1 und für einen Term T2.
a) Zeige Behauptung für [T1 # T2]

Betrachte:



s beginnt bei "[" und geht maximal bis ein Zeichen vor "]"

Fall1: s = [
trivial

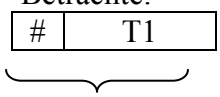
Fall2: Das Ende von s reicht in T1 rein
Da nach Voraussetzung die Anzahl der öffnenden Klammern größer der Anzahl der schliessenden Klammern ist, folgt die Behauptung.

Fall3: Das Ende von s ist "#"
trivial

Fall4: Das Ende von s reicht in T2 rein
Da nach Voraussetzung die Anzahl der öffnenden Klammern größer der Anzahl der schliessenden Klammern ist, folgt die Behauptung.

b) Zeige Behauptung für #T2

Betrachte:



s beginnt bei "#" und geht maximal bis ein Zeichen vor "#"

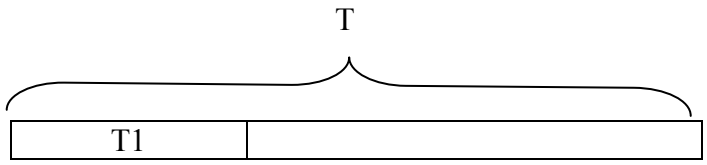
Fall1: s = [
trivial

Fall2: Das Ende von s reicht in T1 rein
Da nach Voraussetzung die Anzahl der öffnenden Klammern größer der Anzahl der schliessenden Klammern ist, folgt die Behauptung.

2.8.3 Hilfslemma HLT3

Ein Term kann nicht echtes Anfangsstück eines anderen arithmetischen Zahlenterms sein.

Beweis:
Fall1: T1 ist nicht von der Form #A
Wäre T1 echtes Anfangsstück von T, dann wäre nach vorigem Lemma:
Die Anzahl der öffnenden Klammern in T1 ist größer als die Anzahl der schliessenden Klammern in T1. Dann wäre nach einem obigen Lemma T1 kein arithmetischer Zahlenterm.



Fall2: T1 = #A
T kann dann nur von folgender Form sein:
a) Konstante
kann nicht sein, da #A nicht Teil der Konstanten sein kann
b) T = [S1#S2]
Dann beginnt aber #A mit der öffnenden Klammer [, was nicht sein kann, da #A mit # beginnt.

2.8.4 Satz

I(R_Term) ist monotektonisch

Beweis:
1) Sei $T \in I(R\_Term)$
zeige: $\exists$ genau ein M mit
$M = \{T1,T2\}$ und $T = [T1\#T2]$ oder
$M = \{T1\}$ und $T = \#T1$ oder
$(\varnothing,T) \in R\_Term$

2) Eindeutigkeit:
a) Annahme: $T = [T1\#_1T2] = [T'1\#_2T'2]$ wobei $T1 \neq T'1$ und $T2 \neq T'2$ und T'1,T2 sind Terme.

[	T1	$\#_1$	T2	]

oder
Wäre T nicht eindeutig, könnte T wie folgt aussehen.

[	T'1	$\#_2$	T'2	]

Damit wäre T'1 echtes Anfangsstück von T1. Widerspruch.

b) Annahme: $T = \#T1 = \#T'1$ wobei $T1 \neq T'1 \Longrightarrow T1 = T'1$ Widerspruch

c) Annahme: $(\varnothing,T) \in R\_Term \Longrightarrow M = \varnothing$

3 Die einfach implementierte Programmiersprache IMP

Jedes in IMP geschriebene Progrmm besteht aus einfachen Anweisungen, if-else-Verzweigungen und while-Schleifen (die verschachtelt sein können).
Dies wird im Folgenden formalisiert.

3.1 Die Syntax von IMP

3.1.1 Definition der arithmetischen Ausdrücke (Aexp)

|Z : = Menge aller ganzen Zahlen
|V : = endliche, geordnete Menge von Programmvariablen, wie z.B: $\{v_1 ; v_2 ; \ldots ; v_n\}$
X_Aexp : = $\{+ ; - ; * ;) ; (\} \cup |V \cup |Z)^*$ ist eine Menge von Worten

R_Aexp =
$\{(\{a_1, a_2\}, [a_1 + a_2]) \mid a_1 \in X\_Aexp \wedge a_2 \in X\_Aexp \} \cup$
$\{ (\{a_1, a_2\}, [a_1 - a_2]) \mid a_1 \in X\_Aexp \wedge a_2 \in X\_Aexp \} \cup$
$\{ (\{a_1, a_2\}, [a_1 * a_2]) \mid a_1 \in X\_Aexp \wedge a_2 \in X\_Aexp \} \cup$
$\{(\varnothing ; v) \mid v \in |V \} \cup$
$\{(\varnothing ; z) \mid z \in |Z \}$

Regelnkurznotation: ($a_1, a_2 \in X\_Aexp$, $z \in |Z$, $v \in |V$)

$\varnothing$	$\varnothing$	$\{a_1 ; a_2\}$	$\{ a_1 ; a_2\}$	$\{ a_1 ; a_2\}$
z	v	$(a_1 + a_2)$	$(a_1 - a_2)$	$(a_1 * a_2)$

Die induktive definierte Menge Aexp := I(R_Aexp) ist die Menge aller arithmetischen Terme.

Einige arithmetischen Terme:
3
v, falls $v \in V$
$[[a_1+a_1]*a_2]$

3.1.2 Monotektonischer Aufbau der Terme

3.1.2.1 Satz

Jeder Term ist monotektonisch aufgebaut, d.h:
R_Aexp ist eine Regelmenge über X_Aexp.
I(R_Aexp) ist monotektonisch : <==>
$\forall a \in I(R\_Aexp)\ \exists$ genau ein $M \subset I(R\_Aexp)$ $(M,b) \in R\_Aexp$

Beweis:
siehe oben

3.1.3 Definition der boolschen Ausdrücke (Bexp)

X_Bexp : = {true ; false ; $\wedge$; $\vee$; $\neg$;) ; (} $\cup$ Aexp)* ist eine Menge von Worten

R_Bexp =
{ ({b_1, b_2}, [$b_1 \wedge b_2$]) | $b_1 \in$ X_Bexp $\wedge$ $b_2 \in$ X_Bexp } $\cup$
{ ({b_1, b_2}, [$b_1 \vee b_2$]) | $b_1 \in$ X_Bexp $\wedge$ $b_2 \in$ X_Bexp } $\cup$
{ ({b}, $\neg$b) | b $\in$ X_Bexp } $\cup$
{($\varnothing$; b) | b $\in$ {true ; false} } $\cup$
{($\varnothing$, [$a_1 = a_2$]) | $a_1 \in$ Aexp $\wedge$ $a_2 \in$ Aexp } $\cup$
{ ($\varnothing$, [$a_1 \leq a_2$]) | $a_1 \in$ Aexp $\wedge$ $a_2 \in$ Aexp }

Regelnkurznotation: ($a_1 \in$ Aexp , $a_2 \in$ Aexp , $b_1 \in$ X_Bexp , $b_2 \in$ X_Bexp)

$\varnothing$	$\varnothing$	$\varnothing$	$\varnothing$	{b_1 ; b_2}	{b_1 ; b_2}	{b}
true	false	[$a_1 = a_2$]	[$a_1 \leq a_2$]	[$b_1 \wedge b_2$]	[$b_1 \vee b_2$]	$\neg$ b

Die induktive definierte Menge Bexp := I(R_Bexp) ist die Menge aller boolschen Terme.

Einige boolsche Terme:
true
[[$b_1 \wedge b_2$] $\wedge$false]

3.1.4 Monotektonischer Aufbau der boolschen Ausdrücke

3.1.4.1 Satz

Jeder boolscher Ausdruck ist monotektonisch aufgebaut, d.h:
R_Bexp ist eine Regelmenge über X_Bexp.
I(R_Bexp) ist monotektonisch : <==>
$\forall b \in$ I(R_Bexp) $\exists$ genau ein M$\subset$ I(R_Bexp) (M,b)$\in$ R_Bexp

Beweis:
siehe oben

3.1.5 Definition der Anweisungen (Com)

X_Com = { skip ;) ; (; := ; if ; then ; else ; while ; do} ∪ Aexp ∪ Bexp)*
ist eine Menge von Worten

R_Com =
{(∅ ; skip) } ∪
{(∅ ; v := a) | a∈Aexp ∧ v∈ |V } ∪
{({c_1 ; c_2} ; [c1;c2]) | c_1∈X_Com ∧ c_2∈X_Com } ∪
{({c_1 ; c_2} ; [if b then c_1 else c_2]) | c_1∈X_Com ∧ c_2∈X_Com ∧ b∈Bexp} ∪
{({c} ; [while b do c]) | c∈X_Com ∧ b∈Bexp} ∪

Bem:
skip bedeutet die leere Anweisung

Regelnkurznotation: (c1∈ X_Com , c2∈ X_Com, a∈Aexp, b∈X_Bexp
∅
---
skip

∅
---
v:=a

{c_1 ; c_2}

[c_1;c_2]

{c_1 ; c_2}

[if b then c_1 else c_2]

{c}

[while b do c]

Die induktive definierte Menge I(R_Com) ist die Menge aller Anweisungen.

Bemerkung
skip bedeutet die leere Anweisung

3.1.6 Monotektonischer Aufbau der Anweisungen

3.1.7 Hilfslemma HLC1

In einer Anweisung ist die Anzahl schliessender Klammern gleich der Anzahl öffnender Klammern.

Beweis: (Induktion über Aufbau der Anweisungen)
B(c) :<==> Anzahl öffnender Klammern in c = Anzahl schliessender Klammern in c
a) c = skip
trivial, da skip keine Klammern besutzt.
b) c = v := a
Da die Anzahl der schliessenden und öffnenden Klammern in einem Term (siehe HLT1) gleich groß ist und v keine Klammern besitzt, gilt die Behauptung.

2) Induktionsvoraussetzung B(c1)$\in$ I(R_Com) und B(c2) $\in$ I(R_Com) und
B(c) $\in$ I(R_Com) und b$\in$Bexp.
Es gilt also:
Die Anzahl der schliessenden und öffnenden Klammern in c1 ist gleich groß.
Die Anzahl der schliessenden und öffnenden Klammern in c2 ist gleich groß.
Die Anzahl der schliessenden und öffnenden Klammern in c ist gleich groß.
Die Anzahl der schliessenden und öffnenden Klammern in b ist gleich groß.
Also gilt die Behauptung auch für:
[c1;c2] und
[if b then c_1 else c_2] und
[while b do c]

3.1.8 Hilfslemma HLC2

Die Zeichenkette s ist echtes Anfangsstück einer Anweisung c, wobei c nicht von der Form v := a sein darf, kurz s < c ==>
Die Anzahl der öffnenden Klammern in s ist größer als die Anzahl der schliessenden Klammern in s.

Beweis: (Induktion über Aufbau der Anweisungen)
B(c) :<==> obige Aussage
1)
a) c = skip
trivial, da skip keine Klammern besitzt.

2) Induktionsvoraussetzung B(c1)∈ I(R_Com) und B(c2) ∈ I(R_Com) und
B(c) ∈ I(R_Com) und b∈Bexp.
Fall1:
Betrachte:

[	c1	;	c2	]

s beginnt bei "[" und geht maximal bis ein Zeichen vor "]"

Unterfall1: s = [
trivial

Unterfall 2: Das Ende von s reicht in c1 rein
Da nach Voraussetzung die Anzahl der öffnenden Klammern größer der Anzahl der schliessenden Klammern ist, folgt die Behauptung.

Unterfall 3: Das Ende von s ist ";"
trivial

Unterfall 4: Das Ende von s reicht in c2 rein
Da nach Voraussetzung die Anzahl der öffnenden Klammern größer der Anzahl der schliessenden Klammern ist, folgt die Behauptung.

Fall2:
Betrachte:

[	if	b	then	c1	else	c2	]

s beginnt bei "[" und geht maximal bis ein Zeichen vor "]"

Unterfall 1: s = [
trivial

Unterfall 2: s = [if
trivial

Unterfall 3: Das Ende von s reicht in b rein
Da nach Voraussetzung in b die Anzahl der öffnenden Klammern maximal der Anzahl der schliessenden Klammern ist, folgt die Behauptung.

Unterfall 4: s = [if b then
Da nach Voraussetzung in b die Anzahl der öffnenden Klammern gleich der Anzahl der schliessenden Klammern ist, folgt die Behauptung.

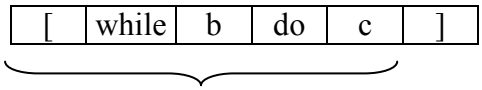
Unterfall 5: Das Ende von s reicht in c1 rein
Da nach Voraussetzung in c1 die Anzahl der öffnenden Klammern maximal der Anzahl der schliessenden Klammern ist, folgt die Behauptung.

Unterfall 6: s = [if b then c1 else
Da nach Voraussetzung in b bzw. c1 die Anzahl der öffnenden Klammern gleich der Anzahl der schliessenden Klammern ist, folgt die Behauptung.

Unterfall 7: Das Ende von s reicht in c2 rein

Da nach Voraussetzung in c2 die Anzahl der öffnenden Klammern maximal der Anzahl der schliessenden Klammern ist, folgt die Behauptung.

Fall 3:
Betrachte:



s beginnt bei "[" und geht maximal bis ein Zeichen vor "]"

Unterfall 1: s = [
trivial

Unterfall 2: s = [while
trivial

Unterfall 3: Das Ende von s reicht in b rein
Da nach Voraussetzung in b die Anzahl der öffnenden Klammern maximal der Anzahl der schliessenden Klammern ist, folgt die Behauptung.

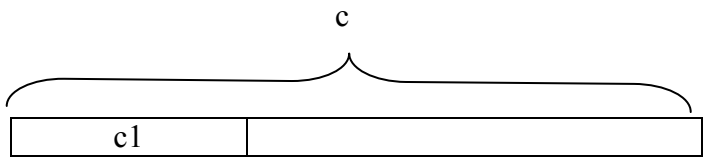
Unterfall 4: s = [while b do
Da nach Voraussetzung in b die Anzahl der öffnenden Klammern gleich der Anzahl der schliessenden Klammern ist, folgt die Behauptung.

Unterfall 5: Das Ende von s reicht in c rein
Da nach Voraussetzung in c die Anzahl der öffnenden Klammern maximal der Anzahl der schliessenden Klammern ist, folgt die Behauptung.

3.1.9 Hilfslemma HLC3

Eine Anweisung kann nicht echtes Anfangsstück einer anderen Anweisung sein.

Beweis:
Fall1: c ist nicht von der Form v:=a
Wäre c1 echtes Anfangsstück von c, dann wäre nach vorigem Lemma:
Die Anzahl der öffnenden Klammern in c1 ist größer als die Anzahl der schliessenden Klammern in c1. Dann wäre nach einem obigen Lemma c1 keine Anweisung.



Fall2: c von der Form v:=a
c kann dann nur von folgender Form sein:
a) skip
kann nicht sein, da v:=a nicht Teil von skip sein kann
b) c muß eine der folgenden Formen annehmen:
c = [com1;com2] oder
c = [if b then c1 else c2] oder
c=[while b do c]
Dann beginnt aber v:=a mit der öffnenden Klammer [, was nicht sein kann, da v:=a mit v beginnt.

Analog dazu gilt:

3.1.10 Hilfslemma HLC4

Eine Anweisung kann nicht echtes Endstück einer anderen Anweisung sein.

Beweis:
analog

3.1.11 Hilfslemma HLC5

Für alle b,b'∈Bexp, c1,c2,c'1,c'2,c∈Com gilt:
1) [c1;c2] = [c'1;c'2] ==> c1=c'1 und c2 = c'2
2) [if b then c1 else c2] = [if b' then c'1 else c'2] ==> b=b' und c1=c'1 und c2 = c'2
3) [while b do c] = [while b' do c'] ==> b = b' und c=c'
4) [c1;c2] ≠ [if b' then c'1 else c'2]
5) [c1;c2] ≠ [while b' do c']
6) [if b then c1 else c2] ≠ [while b' do c']

Beweis:
1) [c1;c2] = [c'1;c'2]

[	c1	#	c2	]

oder
Wäre c nicht eindeutig, könnte c wie folgt aussehen.

[	c'1	#	c'2	]

Damit wäre c'1 echtes Anfangsstück von c1 (oder umgekehrt). Widerspruch, also c1 = c'1

2)
[if b then c1 else c2] = [if b' then c'1 else c'2]
Fall1:

[	if	b	then	c1	else	c2	]

=

[	if	b'	then	c'1	else	c'2	]

Fall2:
Damit wäre b' echtes Anfangsstück von b (oder umgekehrt). Widerspruch, also b1 = b'1

Fall3:

[	if	b	then	c'1	else	c'2	]

=

[	if	b	then	c1	else	c2	]

Damit wäre c1 echtes Anfangsstück von c'1 (oder umgekehrt). Widerspruch, also c1 = c'1

Fall4:

[	if	b	then	c1	else	c2	]

=

[	if	b	then	c1	else	c'2	]

Damit wäre c2 echtes Anfangsstück von c'2. Widerspruch, also c2 = c'2

3)
[while b do c] = [while b' do c']

Fall1:

[	while	b	do	c	]

=

[	while	b'	do	c'	]

Damit wäre b echtes Anfangsstück von b' (oder umgekehrt). Widerspruch, also b = b'

Fall2:

[	while	b	do	c	]

=

[	while	b'	do	c'	]

Damit wäre c' echtes Anfangsstück von c (oder umgekehrt). Widerspruch, also c = c'

4)
Annahme: [if b then c1 else c2] = [while b' do c']
Widerspruch: weil das Token "if" dentisch mit dem Token "while" wäre.

5)
[c1;c2] = [if b then c'1 else c'2]

Fall1:

[	c1	;	c2	]

=

[	if	b'	then	c'1	else	c'2	]

Damit wäre c'2 echtes Endstück von c2 (oder umgekehrt). Widerspruch , also c2 = c'2

Fall2:

[	c1	;	c2	]

=

[	if	b'	then	c'1	else	c'2	]

Damit wäre das Token "else" identisch mit dem Token ";"
Widerspruch.

6) zeige:
[c1;c2] = [while b' do c']

Fall1:

[	c1	;	c2	]

=

[	while	b'	do	c'	]

Damit wäre c' echtes Endstück von c2 (oder umgekehrt). Widerspruch, also c2 = c'2

Fall2:

[	c1	;	c2	]

=

[	while	b'	do	c'	]

Damit wäre das Token ";" identisch mit dem Token "do"
Widerspruch.

3.1.12 Monotektoniesatz über Com

Die induktive definierte Menge I(R_Com), d.h. die Menge aller Anweisungen.ist monotektonisch aufgebaut.

Beweis:
Zeige:
$\forall x \in I(R\_Com)\ \exists$ genau ein $M \subset I(R\_Com)\ (M,x) \in R$

1) x = skip
==> Existiert genau ein M = ∅ mit (M,x) ∈Com

2) x = [c1;c2]∈I(R_Com) und ({c'1;c'2};x)∈R_Com
Fall1: x = [c'1;c'2]
==> [c1;c2] = [c'1;c'2] ==> c1=c'1 und c2 = c'2

Fall2: x = [if b' then c'1 else c'2]
==> [c1;c2] = [if b' then c'1 else c'2] ==> Widerspruch nach vorigem Lemma

Fall3: x = [while b' do c']
==> [c1;c2] = [while b' do c'] ==> Widerspruch nach vorigem Lemma
also: existiert genau ein M = {c1;c2} mit ({c1;c2};x)∈R_Com

3) x = [if b then c1 else c2]∈I(R_Com) und ({c'1;c'2};x)∈R_Com
Fall1: x = [c'1;c'2]
==> [if b then c1 else c2] = [c'1;c'2] ==> Widerspruch nach vorigem Lemma

Fall2: x = [if b' then c'1 else c'2]
==> [if b then c1 else c2] = [if b' then c'1 else c'2] ==> b = b' und c1=c'1 und c2 = c'2

Fall3: x = [while b' do c']
==> [if b then c1 else c2] = [while b' do c'] ==> Widerspruch nach vorigem Lemma
also: existiert genau ein M = {c1;c2} mit ({c1;c2};x)∈R_Com

4) x = [while b do c]∈I(R_Com) und ({c'1;c'2};x)∈R_Com
Fall1: x = [c'1;c'2]
==> [while b do c] = [c'1;c'2] ==> Widerspruch nach vorigem Lemma

Fall2: x = [if b' then c'1 else c'2]
==> [while b do c] = [if b' then c'1 else c'2] ==> Widerspruch nach vorigem Lemma

Fall3: x = [while b' do c']
==> [while b do c] = [while b' do c'] ==> b = b' und c=c'
also: existiert genau ein M = {c1;c2} mit ({c1;c2};x)∈R_Com

3.2 Die Semantik von IMP

3.2.1 Definition

Die Abbildung
h : |V ----> |Z nennt man Zustand.
Die Menge dieser Abbildungen bezeichnet man mit Σ
Durch diese Abbildung werden allen Variablen eine ganze Zahl (ein Wert) zugeordnet.
Sei n ∈ |Z , dann wird definiert:
h[n/v] (z) : = h(z), falls z ≠ v
: = n, falls z = v

h[n1/v1][n2/v2] : = (h[n1/v1]) [n2/v2]

3.2.2 Semantik der arithmetischen Ausdrücke

3.2.2.1 Definitionen

<Σ --> |Z> : = Menge aller Abbildungen von Σ --> |Z
Es sei n∈|Z, v∈|V dann wird definiert:
1)
$F_{\_k}(h)$: = k für alle h
$F_{\_v}(h)$ = h(v) für alle h

Es sei h∈Σ, $F_1, F_2 \in$ <Σ -->|Z> dann wird definiert:
2)
$(F_1 + F_2)(h) : = F_1(h) + F_2(h)$:= add($F_1(h)$, $F_2(h)$)
$(F_1 - F_2)(h) : = F_1(h) - F_2(h)$:= sub($F_1(h)$, $F_2(h)$)
$(F_1 * F_2)(h) : = F_1(h) * F_2(h)$:= mul($F_1(h)$, $F_2(h)$)

3)
A = X_Aexp
B = <Σ --> |Z>
r(1) = r(2) = r(3) = 2

4)
Für alle $a_1, a_2 \in$ X_Aexp gilt:
$W_{1,1}((a_1, a_2)) = [a_1 + a_2]$
$W_{2,1}((a_1, a_2)) = [a_1 - a_2]$
$W_{3,1}((a_1, a_2)) = [a_1 * a_2]$

Für alle $F_1, F_2 \in$ <Σ --> |Z> gilt:
$W_{1,2}(F_1, F_2) = F_1 + F_2$
$W_{2,2}(F_1, F_2) = F_1 - F_2$
$W_{3,2}(F_1, F_2) = F_1 * F_2$

5)
R_2_Ax : = { (∅, (n, $F_{\_n}$)) | n ∈ |Z } ∪ { (∅, (v, $F_{\_v}$)) | v ∈ |V }

6)
Dadurch werden die Regeln R_SAexp definiert.

3.2.2.2 Regelnkurznotation

$F_1 \in <\Sigma \dashrightarrow |Z>$ und $F_2 \in <\Sigma \dashrightarrow |Z>$ und $F \in <\Sigma \dashrightarrow |Z>$ und
$a_1 \in A$ und $a_2 \in A$ und $n \in |Z$ und $v \in |V$

$$\frac{\varnothing}{(n, F\_n)}$$

$$\frac{\varnothing}{(v, F\_v)}$$

$$\frac{(a_1, F_1)\ ,\ (a_2, F_2)}{([a_1 + a_2]\ ,\ F_1 + F_2}$$

$$\frac{(a_1, F_1)\ ,\ (a_2, F_2)}{([a_1 - a_2]\ ,\ F_1 - F_2\ )}$$

$$\frac{(a_1, F_1)\ ,\ (a_2, F_2)}{([a_1 * a_2]\ ,\ F_1 * F_2\ )}$$

3.2.3 Lemma

R1(R_Aexp) = R_Aexp

Beweis:
$R1(R\_Aexp) = \{ (\{a_1, a_2\}, a) \mid i \in \{1;2;3\} \wedge ((a_1, a_2), a) \in W_{i,1}\} \cup$
$\{(\varnothing, a) \mid \exists b\ (\varnothing, (a,b)) \in R\_2\} =$
$\{(\{a_1, a_2\}, [a_1 + a_2]) \mid a_1 \in X\_Aexp \wedge a_2 \in X\_Aexp\} \cup$
$\{(\{a_1, a_2\}, [a_1 - a_2]) \mid a_1 \in X\_Aexp \wedge a_2 \in X\_Aexp\} \cup$
$\{(\{a_1, a_2\}, [a_1 * a_2]) \mid a_1 \in X\_Aexp \wedge a_2 \in X\_Aexp\} \cup$
$\{(\varnothing\ ;\ v) \mid v \in |V\} \cup$
$\{(\varnothing\ ;\ z) \mid z \in |Z\}$

3.2.4 Lemma

$a \in Aexp \implies \exists b \in <\Sigma \dashrightarrow |Z>\ (a,b) \in I(R\_Aexp)$

Beweis:
Beweis: Induktion über Aexp := I(R_Aexp)
Definiere:
$B(a) : <\!\!\implies a \in Aexp \implies \exists b \in <\Sigma \dashrightarrow |Z>\ (a,b) \in I(R\_Aexp)$

1) zeige:
$(\varnothing,a)\in R\_Aexp \Longrightarrow B(a,b)$
Es sei: $(\varnothing,a)\in R\_Aexp$
Fall1: $a\in|V$
$\Longrightarrow (\varnothing, (a, F_{\_a}))\in R\_Aexp \Longrightarrow (a, F_{\_a})\in I(R\_SAexp)$

Fall2: $a\in|Z$
$\Longrightarrow (\varnothing, (a, F_{\_a}))\in R\_SAexp \Longrightarrow (a, F_{\_a})\in I(R\_SAexp)$

2) zeige:
Aus den Induktionsvoraussetzungen (IV)
$a_1\in I(R\_Aexp) \wedge a_2\in I(R\_Aexp)$ und
$B(a_1) \wedge B(a_2)$ und
$(\{a_1,a_2\},[a_1 + a_2])\in R\_Aexp$
folgt: $B([a_1 + a_2])$ (analoges gilt für - und *)

Es sei $[a_1 + a_2]\in Aexp$
Da $a_1\in I(R\_Aexp) \wedge a_2\in I(R\_Aexp)$, gibt es $F_1,F_2\in<\Sigma \dashrightarrow |Z>$ mit
$(a_1,F_1)\in I(R\_SAexp)$ und $(a_2,F_2)\in I(R\_SAexp)$
$W_{1,1}((a_1, a_2)) = [a_1 + a_2]$
$W_{1,2}(F_1, F_2) = F_1 + F_2$
also:
$(\{(a_1,F_1), (a_2,F_2)\}, ([a_1 + a_2], F_1 + F_2)) \in R\_SAexp$
also:
$([a_1 + a_2], F_1 + F_2)) \in I(R\_SAexp)$

3.2.5 Semantiksatz der arithmetischen Ausdrücke

Wenn $a\in Aexp$, dann gilt:
$N(a) = N(a_1) + N(a_2)$, falls $a = [a_1 + a_2]$ und $a_1\in Aexp \wedge a_2\in Aexp$
$N(a) = N(a_1) - N(a_2)$, falls $a = [a_1 - a_2]$ und $a_1\in Aexp \wedge a_2\in Aexp$
$N(a) = N(a_1) * N(a_2)$, falls $a = [a_1 * a_2]$ und $a_1\in Aexp \wedge a_2\in Aexp$
$N(z) = F_{\_z}$ falls $z\in|Z$
$N(v) = F_{\_v}$ falls $v\in|V$

Bemerkungen:
$N(v)(h) = h(v)$ für alle h
$N(z)(h) = z$

Beweis:
1)
$a \in Aexp \implies \exists b \in <\Sigma \dashrightarrow |Z> (a,b) \in I(R\_SAexp)$
Damit ist eine Voraussetzung des Rekursionssatzes erfüllt

2)
Da I(R_Aexp) monotektonisch und regeleindeutig, ist auch I(R1(R_SAexp)) monotektonisch und regeleindeutig.
(da R1(R_SAexp) = R_Aexp)
Damit ist eine Voraussetzung des Rekursionssatzes erfüllt

3)
$D_1(R\_2) = \{(n, F\_n) \mid n \in |Z \} \cup \{ (v, F\_v) \mid v \in |V \}$ ist rechtseindeutig
Damit ist eine Voraussetzung des Rekursionssatzes erfüllt

4)
$W_{i,1}$ ist für alle i>0 linkseindeutig, da Terme monotektonisch sind.
Damit ist eine Voraussetzung des Rekursionssatzes erfüllt

5)
$W_{i,2}$ ist für alle i>0 rechtseindeutig
Damit ist eine Voraussetzung des Rekursionssatzes erfüllt

6)
Damit sind alle Voraussetzung des Satzes erfüllt, also gilt:
$N(a) = W_{i,2}(N(a_1), ..., N(a_r))$, falls $\exists\, i>0\ ((a_1, ..., a_r),a) \in W_{i,1}$
$N(a) = b$, falls $(\varnothing ,(a,b)) \in R\_2$

konkret:
$N(a) = N(a_1) + N(a_2)$, falls $W_{1,1}((a_1, a_2)) = [a_1 + a_2]$ und $a_1 \in Aexp \wedge a_2 \in Aexp$
$N(a) = N(a_1) - N(a_2)$, falls $W_{2,1}((a_1, a_2)) = [a_1 - a_2]$ und $a_1 \in Aexp \wedge a_2 \in Aexp$
$N(a) = N(a_1) * N(a_2)$, falls $W_{3,1}((a_1, a_2)) = [a_1 * a_2]$ und $a_1 \in Aexp \wedge a_2 \in Aexp$
$N(z) = F\_z$ falls $z \in |Z$
$N(v) = F\_v$ falls $v \in |V$

Bemerkung:
Wenn N doppeldeutig auch in anderen Zusammenhängen vorkommen sollte, dann wird es -je nach Bedeutung - mit N_A bzw. N_B bzw. N_C bezeichnet.

3.2.6 Semantik der booleschen Ausdrücke

3.2.6.1 Definitionen

T : = {wahr, falsch}
<Σ --> T> : = Menge aller Abbildungen von Σ --> T
Es wird definiert:
1)
$F\_{w}(h)$:= wahr für alle $h \in \Sigma$
$F\_{f}(h)$:= falsch für alle $h \in \Sigma$

Es sei $h \in \Sigma$, $F_1, F_2 \in <\Sigma \text{ --> } T>$ dann wird definiert:
2)
$F\_{[a1 = a2]}(h)$: = wahr, falls $N\_A(a_1)(h) = N\_A(a_2)(h)$
$F\_{[a1 = a2]}(h)$: = falsch, falls $N\_A(a_1)(h) \neq N\_A(a_2)(h)$

3)
$F\_{[a1 \leq a2]}(h)$: = wahr, falls $N\_A(a_1)(h) \leq N\_A(a_2)(h)$
$F\_{[a1 \leq a2]}(h)$: = falsch, falls $N\_A(a_1)(h) > N\_A(a_2)(h)$

4)
(non (F))(h) : = nicht (F(h))
$(F_1 \text{ et } F_2)(h)$: = $F_1(h)$ and $F_2(h)$
$(F_1 \text{ vel } F_2)(h)$: = $F_1(h)$ or $F_2(h)$

5)
A := X_Bexp
B := <Σ --> T>
r(1) = 1 und r(2) = r(3) = 2

6)
Für alle $b, b_1, b_2 \in$ Bexp und alle $F_1, F_2 \in <\Sigma \text{ --> } T>$ wird definiert:
$W_{1,1}(b) := \neg b$
$W_{2,1}((b_1, b_2)) := [b_1 \wedge b_2]$ mit
$W_{3,1}((b_1, b_2)) := [b_1 \vee b_2]$

$W_{1,2}(\neg F) :=$ non (F)
$W_{2,2}(F_1, F_2) := F_1 \text{ et } F_2$
$W_{3,2}(F_1, F_2) := F_1 \text{ vel } F_2$

7)
R2_Ax : =
$\{ (\varnothing, (\text{true}, F\_{w})) \mid F\_{w} \in <\Sigma \text{ --> } T> \} \cup$
$\{ (\varnothing, (\text{false}, F\_{f})) \mid F\_{f} \in <\Sigma \text{ --> } T> \} \cup$
$\{ (\varnothing, ([a_1 = a_2], F\_{[a1 = a2]})) \mid a_1 \in \text{Aexp und } a_2 \in \text{Aexp und } F\_{[a1 = a2]} \in <\Sigma \text{ --> } T> \cup$
$\{ (\varnothing, ([a_1 \leq a_2], F\_{[a1 \leq a2]})) \mid a_1 \in \text{Aexp und } a_2 \in \text{Aexp und } F\_{[a1 \leq a2]} \in <\Sigma \text{ --> } T>$

8)
Dadurch werden die Regeln R_SBexp definiert.

3.2.6.2 Regelnkurznotation

$(a1 \in Aexp , a2 \in Aexp , b1 \in X\_Bexp , b2 \in X\_Bexp)$

$$\frac{\varnothing}{(true, F\_{w})}$$

$$\frac{\varnothing}{(false, F\_{f})}$$

$$\frac{\varnothing}{([a_1 = a_2], F\_{[a1 = a2]})}$$

$$\frac{\varnothing}{([a_1 \leq a_2], F\_{[a1 \leq a2]})}$$

$$\frac{(b,F)}{(\neg b , non(F))}$$

$$\frac{(b_1,F_1) \ , \ (b_2,F_2)}{([b_1 \wedge b_2], \ F_1 \text{ et } F_2)}$$

$$\frac{(b_1,F_1) \ , \ (b_2,F_2)}{([b_1 \vee b_2], \ F_1 \text{ vel } F_2)}$$

3.2.7 Semantiksatz der boolschen Ausdrücke

Wenn $b \in Bexp$, dann gilt:

$N(b) = non\ N(b_1)$, falls $b = [\neg b_1]$ und $b_1 \in Bexp$

$N(b) = N(b_1)\ et\ N(b_2)$, falls $b = [b_1 \wedge b_2]$ und $b_1, b_2 \in Bexp$

$N(b) = N(b_1)\ vel\ N(b_2)$, falls $b = [b_1 \vee b_2]$ und $b_1, b_2 \in Bexp$

$N(true) = F\_w$

$N(false) = F\_f$

$N([a_1 = a_2]) = F\_[a1 = a2]$ falls $a_1, a_2 \in Aexp$

$N([a_1 \leq a_2]) = F\_[a1 \leq a2]$ falls $a_1, a_2 \in Aexp$

Bemerkungen:

1)

$N([a_1 = a_2])(h) = wahr$, falls $N\_A(a_1)(h) = N\_A(a_2)(h)$

$N([a_1 \leq a_2])(h) = falsch$, falls $N\_A(a_1)(h) \neq N\_A(a_2)(h)$

$N(true)(h) = wahr$

$N(false)(h) = falsch$

Beweis:

fehlt noch

3.2.8 Semantik der Anweisungen

3.2.8.1 Definitionen

1)
R_SCom =
$\{ (\varnothing, ((skip,h),h) \mid h \in \Sigma \} \cup$

$\{ (\varnothing, ((v:=a,h),h[N\_A(v)(h)/v]) \mid h \in \Sigma \wedge a \in Aexp \wedge v \in |V \ \} \cup$

$\{ (\varnothing,(([while\ b\ do\ c],h),h) \mid c \in Com \wedge h \in \Sigma \ \wedge N\_B(b)(h)=falsch \} \cup$

$\{ (\{ ((c_1,h),h') , ((c_2,h'),h'')\}, (([c_1;c_2],h),h'') \mid c_1,c_2 \in Com \wedge h,h',h'' \in \Sigma \} \cup$

$\{ (\{ ((c_1,h),h'') \}, (([if\ b\ then\ c_1\ else\ c_2],h),h') \mid c_1,c_2 \in Com \wedge h,h',h'' \in \Sigma \ \wedge N\_B(b)(h)=wahr\} \cup$

$\{ (\{ ((c_2,h),h') \}, (([if\ b\ then\ c_1\ else\ c_2],h),h') \mid c_1,c_2 \in Com \wedge h,h',h'' \in \Sigma \ \wedge N\_B(b)(h)=wahr\} \cup$

$\{ (\{((c,h),h')) ; ((([while\ b\ do\ c],h'),h'')\}; (([while\ b\ do\ c],h),h'')) \mid c \in Com \wedge \ h,h',h'' \in \Sigma \wedge$
$N\_B(b)(h)=wahr \}$

abgekürzt:

$R_1 = \{ (\varnothing, ((skip,h),h) \mid h \in \Sigma \}$

$R_2 = \{ (\varnothing, ((v:=a,h),h[N\_A(v)(h)/v]) \mid h \in \Sigma \wedge a \in Aexp \wedge v \in |V \ \}$

$R_3 = \{ (\{ ((c_1,h),h') , ((c_2,h'),h'')\}, (([c_1;c_2],h),h'') \mid c_1,c_2 \in Com \wedge h,h',h'' \in \Sigma \}$

$R_4 :=$
$\{(\{ ((c_1,h),h'') \}, (([if\ b\ then\ c_1\ else\ c_2],h),h') \mid c_1,c_2 \in Com \wedge h,h',h'' \in \Sigma \ \wedge N\_B(b)(h)=wahr\}$

$R_5 : =$
$\{ (\{ ((c_2,h),h') \}, (([if\ b\ then\ c_1\ else\ c_2],h),h') \mid c_1,c_2 \in Com \wedge h,h',h'' \in \Sigma \ \wedge N\_B(b)(h)=wahr\}$

$R_6 : =$
$\{ (\{((c,h),h')) ; ((([while\ b\ do\ c],h'),h'')\}; ((while\ b\ do\ c,h),h'')) \mid c \in Com \wedge \ h,h',h'' \in \Sigma \wedge$
$N\_B(b)(h)=wahr \}$

Regelnkurznotation:

$$\frac{\varnothing}{((\text{skip}, h), h)}$$

$$\frac{\varnothing}{((v:=a, h), h[N\_A(v)(h)/v])} \quad a \in Aexp \text{ und } v \in |V$$

$$\frac{\varnothing}{(([\text{while } b \text{ do } c], h), h)} \quad N\_B(b)(h) = \text{falsch} \text{ und } b \in Bexp$$

$$\frac{((c_1, h), h') , ((c_2, h'), h'')}{(([c_1;c_2], h), h'')}$$

$$\frac{((c_1, h), h')}{(([\text{if } b \text{ then } c1 \text{ else } c2], h), h')} \quad N\_B(b)(h) = \text{wahr} \text{ und } b \in Bexp$$

$$\frac{((c_2, h), h')}{(([\text{if } b \text{ then } c1 \text{ else } c2], h), h')} \quad N\_B(b)(h) = \text{falsch} \text{ und } b \in Bexp$$

$$\frac{(c, h)h'), ([\text{while } b \text{ do } c], h'), h'')}{([\text{while } b \text{ do } c], h), h'')} \quad N\_B(b)(h) = \text{wahr} \text{ und } b \in Bexp$$

Bemerkungen:
Warum definiert man nicht (wie bei Aexp und Bexp) wie folgt?
$W_{1,1}((c_1, h) , (c_2, h')) := ([c_1;c_2], h)$
Antwort:
Da dann $W_{1,1}$ nicht mehr linkseindeutig ist und man den Rekursionssatz nicht mehr anwenden kann.

3.2.8.2 Lemma über Regelndisjunktion

$R\_SCom = \bigcup_{i \in \{1,\ldots,6\}} R_i$ ist eine disjunkte Vereinigung.

Beweis:
Mit Hilfslemma HLC5

3.2.8.3 Lemma

1)
$(Y,x) \in R\_SCom$ und $x = (([c_1;c_2],h),h'') \Longrightarrow (Y,x) \in R_3$

2)
$(Y,x) \in R\_SCom$ und $x = (([\text{if b then c1 else c2}], h),h')$ und $N\_B(b)(h) = \text{wahr}$
$\Longrightarrow (Y,x) \in R_4$

3)
$(Y,x) \in R\_SCom$ und $x = (([\text{if b then c1 else c2}], h),h')$ und $N\_B(b)(h) = \text{wahr}$
$\Longrightarrow (Y,x) \in R_5$

4)
$(Y,x) \in R\_SCom$ und $x = (([\text{while b do c}], h),h'')$ und $N\_B(b)(h) = \text{wahr}$
$\Longrightarrow (Y,x) \in R_6$

Beweis: induktiv über den Aufbau von I(R_SCom)
1)
Es sei
$(Y,x) \in R\_SCom$ und $x = (([c_1;c_2],h),h'')$
Da
$\frac{((c_1, h),h') \ , \ ((c_2, h'),h'')}{(([c_1;c_2],h),h'')} \in R\_SCom$
und $R\_SCom = \bigcup_{i \in \{1,\ldots,6\}} R_i$ ist eine disjunkte Vereinigung, folgt die Behauptung.

2)
Es sei
$(Y,x) \in R\_SCom$ und $x = (([\text{if b then } c_1 \text{ else } c_2], h),h')$ und $N\_B(b)(h) = \text{wahr}$
Da
$\frac{((c_1, h),h')}{(([\text{if b then } c_1 \text{ else } c_2], h),h')} \in R\_SCom$
und $R\_SCom = \bigcup_{i \in \{1,\ldots,6\}} R_i$ ist eine disjunkte Vereinigung, folgt die Behauptung.

3) restliche Fälle analog

3.2.8.4 Lemma

R_SCom ist rechtseindeutig

Beweis: induktiv über den Aufbau von I(R_SCom)
B((c,h),h') :<==>
$\forall$((c,h),h'')$\in$I(R_SCom) ((c,h),h') = ((c,h),h'') ==> h' = h''

Es sei
((c, h),h')$\in$I(R_SCom) und ((c,h),h'')$\in$I(R_SCom)

Fall1: c = [c_1;c_2]
Dann gilt nach obigem Lemma für beliebige g_1,g_2 $\in\Sigma$
((c_1, h),g_1) , ((c_2, g_1),h')
---------- $\in$ R_SCom
(([c_1;c_2],h),h')

((c_1, h),g_2) , ((c_2, g_2),h'')
---------- $\in$ R_SCom
(([c_1;c_2],h),h'')

Nach IV gilt B((c_1, h),g_1) und B((c_2, g_1),h'), also:
((c_1, h),g_1) = ((c_1, h),g_2) ==> g_1 = g_2 also:
((c_2, g_1),h') = ((c_2, g_2),h'') ==> h' = h'', da g_1 = g_2

Fall2: c = [if b then c_1 else c_2] und N_B(b)(h) = wahr
Dann gilt nach obigem Lemma für beliebige g_1,g_2 $\in\Sigma$
((c_1, h),g_1)
---------- $\in$ R_SCom
(([if b then c_1 else c_2],h),h')

((c_1, h),g_2)
---------- $\in$ R_SCom
(([if b then c_1 else c_2],h),h'')

Nach IV gilt B((c_1, h),g_1) und B((c_2, g_1),h'), also:
((c_1, h),g_1) = ((c_1, h),g_2) ==> g_1 = g_2 also
((c_2, g_1),h') = ((c_2, g_2),h'') ==> h' = h'', da g_1 = g_2

3) restliche Fälle analog

3.2.9 Semantiksatz der Anweisungen

Voraussetzungen:
$((com,h),h') \in I(R\_SCom)$ und
$com = [c_1, c_2]$ oder $com = [\text{if } b \text{ then } c_1 \text{ else } c_2]$ oder $com = [\text{while } b \text{ do } c]$ oder $com = v := a$
und $a \in Aexp$, $v \in |V$

Behauptung:

$N([c_1, c_2],h) = N(c_2, N(c_1,h))$

$N([\text{if } b \text{ then } c_1 \text{ else } c_2],h) = N(c_1,h)$ falls $N\_B(b)(h)=wahr$

$N([\text{if } b \text{ then } c_1 \text{ else } c_2],h) = N(c_2,h)$ falls $N\_B(b)(h)=falsch$

$N([\text{while } b \text{ do } c],h) = N([\text{while } b \text{ do } c]), N(c,h))$ falls $N\_B(b)(h)=wahr$

$N(skip,h) = h$

$N(v := a,h) = h[N\_A(a)(h)/v])$

Beweis: induktiv über den Aufbau von $I(R\_SCom)$
$B((com,h),h') :\Leftrightarrow$ obige Behauptung

Fall1: $com = [c_1;c_2]$

$$\frac{((c_1, h),g) , ((c_2, g),h')}{([c_1;c_2],h),h')} \in R\_SCom$$

Da $((com,h),h') \in I(R\_SCom)$, folgt $h' = N(com,h)$, also
$h' = N([c_1, c_2],h)$
Nach IV gilt:
$B((c_1, h),g)$ und $B((c_2, g),h')$
Da $((c_1, h),g) , ((c_2, g),h'') \in I(R\_SCom)$, folgt:
$g = N(c_1, h)$ und $h' = N(c_2, g)$
also:
$N([c_1, c_2],h) = h' = N(c_2, g) = N(c_2, N(c_1, h))$, also
$N([c_1, c_2],h) = N(c_2, N(c_1,h))$

restliche Fälle analog

3.2.10 Beispiel

Allgemeine Voraussetzung: h(i) = K
also: h[K/i] = h

3.2.10.1 Behauptung1

N_A(i-1)(h[K-n/i]) = K-n-1

Beweis:
// Schreibweise korrekt oder N(i=i-1,h)
N_A(i-1)(h) = N_A(i)(h) - N_A(1)(h) = h(i)-1 = K-1
N_A(i)(h[K-n/i]) = F_{-i}(h[K-n/i]) = (h[K-n/i])(i) = K-n
N_A(i-1)(h[K-n/i]) = N_A(i)(h[K-n/i]) - N_A(1)(h[K-n/i]) = K-n-1

3.2.10.2 Behauptung2

N(i:=i-1,h)[K-n/i]) = h[K-n-1/i]

Beweis:
N(i:=i-1,h) = h[N_A(i-1)(h)/i] = h[K-1/i]
Da aus h(i) = K folgt h[K/i] = h, gilt:
N(i:=i-1,h[K/i]) = h[N_A(i-1)(h)/i] = h[K-1/i]
weiter:
N(i:=i-1,h[K-1/i]) = h[N_A(i-1)(h[K-1/i]) /i] = h[K-2 /i]
N(i:=i-1,h[K-2/i]) = h[N_A(i-1)(h[K-2/i])/i] = h[K-3/i]
...

3.2.10.3 Behauptung3

N_B([0≤ i])(h[K-n/i]) = wahr, falls 0 ≤ K-n
N_B([0≤ i])(h[K-n/i]) = falsch, falls 0 > K-n

Beweis:
N_B([0≤ i])(h[K-n/i]) = wahr, falls N_A(0)(h[K-n/i]) ≤ N_A(i)(h[K-n/i])
N_B([0≤ i])(h[K-n/i]) = falsch, falls N_A(0)(h[K-n/i]) > N_A(i)(h[K-n/i])

also
N_B([0≤ i])(h[K-n/i]) = wahr, falls 0 ≤ K-n
N_B([0≤ i])(h[K-n/i]) = falsch, falls 0 > K-n

3.2.10.4 Behauptung4

Voraussetzungen:
com ≡ [while 0≤i do i: = i-1]
h(i) : = K := 2
Dann gilt:
N([while 0≤i do i: = i-1] ,h) = h[-1/i]

Beweis:
Also folgt nach obiger Behauptung:
N_B([0≤ i])(h[2-n/i]) = wahr, falls 0 ≤ 2-n
N_B([0≤ i])(h[2-n/i]) = falsch, falls 0 > 2-n, also

N_B([0≤ i])(h[2-n/i]) = wahr, falls n ≤ 2
N_B([0≤ i])(h[2-n/i]) = falsch, falls n > 2, also

// Schreibweisen korrekt ?
N([while 0≤i do i: = i-1] ,h) = N([while 0≤i do i: = i-1] , h[K/i]) =
N([while 0≤i do i: = i-1], N(i:=i-1, h[K/i])) = N([while 0≤i do i: = i-1], h[K-1/i]) =
N([while 0≤i do i: = i-1], N(i:=i-1, h[K-1/i])) = N([while 0≤i do i: = i-1], h[K-2/i])) =
N([while 0≤i do i: = i-1], N(i:=i-1, h[K-2/i])) = N([while 0≤i do i: = i-1], h[K-3/i])) =
h[K-3/i]) = h[-1/i])

4 Die Mächtigkeit von IMP

Wenn man die while-Anweisungen aus IMP entfernt, ergibt dies die neue Programmiersprache IMPE.
Intuitiv erwartet man, daß diese weniger mächtig ist. als IMP.

4.1 Die Programmiersprache IMPE

4.1.1 Definition

IMPE (E wie einfach) erhält man durch Weglassen der while-Anweisung in IMP.

4.1.2 Definition

Sei P eine Anweisung in IMP, formal:$P \in I(R\_Com)$
V(P) := Menge aller in P vorkommenden Programmvariablen
Sei a ein arithemtischer Ausdruck in IMP, formal: $a \in Aexp$
V(a) := Menge aller in a vorkommenden Programmvariablen

4.1.3 Behauptung

$V(P) := \{v_1, ..., v_n\} \Longrightarrow h = h[h(v_1) / v_1] ... [h(v_n) / v_n]$

4.1.4 Definition

Es sei $x_1, ..., x_n \in |Z$
$p(x_1, ..., x_n)$: = Polynom in den Variablen $x_1, x_2, ..., x_n$

4.1.5 Behauptung

$a \in Aexp \Longrightarrow$
$N\_A(a)(h) \in p(h(v_1) , ..., h(v_n))$, falls $V(a) = \{v_1, ..., v_n\}$

Beweis: (Induktion über Aexp)
1) $a \in |Z$
$N\_A(a)(h) = F\_a(h) = a \in p(v_1, ..., v_n)$

2) $a \in |V$
$N\_A(a)(h) = F\_a(h) = h(a) \in |Z$, also $h(a) \in p(v_1, ..., v_n)$
Aus 1) folgt, daß ein i existiert mit $a = v_i$, also:
$N\_A(a)(h) = F\_a(h) = h(a) = h(v_i) = v'_i \in p(v_1, ..., v_n)$

3)
B(a) : <==>
$V(a) = \{v_1, ..., v_n\} \Longrightarrow N\_A(a)(h) \in p(h(v_1) , ..., h(v_n))$

Zeige: $B(a_1)$ und $B(a_2) \Longrightarrow B(a_1 + a_2)$

Es sei:
$V(a_1) = \{u_1, ..., u_p\}$
$V(a_2) = \{w_1, ..., w_q\}$

$V(a_1 + a_2) = \{v_1, ..., v_n\} = V(a_1) \cup V(a_2) = \{u_1, ..., u_p, w_1, ..., w_q\}$
$N\_A(a_1 + a_2)(h) = (N\_A(a_1) + N\_A(a_2))(h) = N\_A(a_1)(h) + N\_A(a_2)(h)$
Da
$N\_A(a_1)(h) \in p(u_1', ..., u_p')$ und $N\_A(a_2)(h) \in p(w_1', ..., w_q')$ und
$\{u'_1, ..., u'_p\} \cup \{w'_1, ..., w'_q\} = \{v'_1, ..., v'_n\}$, folgt
$N\_A(a_1)(h) + N\_A(a_2)(h) \in p(v_1', ..., v_n')$

4)
analog für * und -

4.1.6 Behauptung

Für alle Anweisungen P aus IMPE gilt:
$V(P) = \{v_1, ..., v_n\}$ ==>
$N(P,h)(w) \in p(h(v_1), ..., h(v_n))$, falls $w \in V(P)$
$N(P,h)(w) = h(w)$, sonst

Beweis: (Induktion über IMPE)
I)
1) $P \equiv v := a$
$N(v := a,h) = h[N\_A(a)(h)/v]$, also
$N(v := a,h)(w) = h[N\_A(a)(h)/v])(w)$
Da $w \notin V(P)$ und $v \in V(P)$, folgt $w \neq v$, also
$N(v := a,h)(w) = h[N\_A(a)(h)/v])(w) = h(w)$

2)
B(P) : <==>
$V(P) = \{v_1, ..., v_n\}$ und $w \notin V(P)$ ==> $N(P,h)(w) = h(w)$, sonst

Zeige: $B(P_1)$ und $B(P_2)$ ==> $B([P_1 ; P_2])$

Es sei:
$w \notin V([P_1 ; P_2])$, also $w \notin V(P_1)$ und $w \notin V(P_2)$, also
$N([P_1 ; P_2],h)(w) = N(P_1 , N(P_2,h))(w) = N(P_2,h)(w) = h(w)$

3)
Zeige: $B(P_1)$ und $B(P_2)$ ==> $B([\text{if } b \text{ then } P_1 \text{ else } P_2])$
Es sei:
$w \notin V([\text{if } b \text{ then } P_1 \text{ else } P_2])$, also $w \notin V(P_1)$ und $w \notin V(P_2)$, also
Fall1: falls N_B(b)(h)=wahr
$N([\text{if } b \text{ then } P_1 \text{ else } P_2],h)(w) = N(P_1,h)(w) = h(w)$

Fall2: falls N_B(b)(h) = falsch
$N([\text{if } b \text{ then } P_1 \text{ else } P_2],h)(w) = N(P_2,h)(w) = h(w)$

II)
1) $P \equiv v := a$
a)
Da a in P vorkommt, gilt $V(a) := \{u_1, ..., u_p\} \subset V(P) := \{v_1, ..., v_n\}$
b)
$N(v := a,h) = h[N\_A(a)(h)/v])$, also
$N(v := a,h)(w) = h[N\_A(a)(h)/v])(w)$
Fall1: $w = v$
$h[N\_A(a)(h)/v])(w) = N\_A(a)(h) \in p(u'_1, ..., u'_p)$, also
$h[N\_A(a)(h)/v])(w) = N\_A(a)(h) \in p(v'_1, ..., v'_n)$

Fall2: $w \neq v$
Da $w \in V(P)$, existiert ein i mit $w = v_i$
$h[N\_A(a)(h)/v])(v_i) = h(v_i) = v'_i \in p(v'_1, ..., v'_n)$

2)
B(P) : <==>
$V(P) = \{v_1, ..., v_n\}$ und $w \in V(P)$ ==> $N(P,h)(w) \in p(h(v_1) , ..., h(v_n))$

Zeige: $B(P_1)$ und $B(P_2)$ ==> $B([P_1 ; P_2])$

Es sei:
$V(P_1) = \{u_1, ..., u_p\}$
$V(P_2) = \{w_1, ..., w_q\}$
$V([P_1 ; P_2]) = \{v_1, ..., v_n\} = V(P_1) \cup V(P_2) = \{u_1, ..., u_p, w_1, ..., w_q\}$

Es sei $w \in V([P_1 ; P_2]) = V(P_1) \cup V(P_2)$
$N([P_1 ; P_2],h)(w) = N(P_1 , N(P_2,h))(w)$
Fall1: $w \in V(P_1)$
$N(P_1 , N(P_2,h))(w) \in p(N(P_2,h)(u_1) , ..., N(P_2,h)(u_p))$
Unterfall 1.1: $u_i \in V(P_2)$
also $N(P_2,h)(u_i) \in p(h(w_1) , ..., h(w_q))$, also:
$N(P_2,h)(u_i) \in p(h(v_1) , ..., h(v_n))$
Unterfall 1.2: $u_i \notin V(P_2)$
also $N(P_2,h)(u_i) = h(u_i) \in p(h(v_1) , ..., h(v_n))$, also zusammengefasst:
Für alle $i \in \{1, ..., n\}$ gilt $N(P_2,h)(u_i) \in p(h(v_1) , ..., h(v_n))$, also:
$N(P_1 , N(P_2,h))(w) \in p(h(v_1) , ..., h(v_n))$

Fall2: $w \notin V(P_1)$
$N(P_1, N(P_2,h))(w) = N(P_2,h)(w) \in p(h(w_1), ..., h(w_q))$, also:
$N(P_1, N(P_2,h))(w) = N(P_2,h)(w) \in p(h(v_1), ..., h(v_n))$

3)
Zeige: $B(P_1)$ und $B(P_2)$ ==> $B([\text{if b then } P_1 \text{ else } P_2])$

Es sei:
$V(P_1) = \{u_1, ..., u_p\}$
$V(P_2) = \{w_1, ..., w_q\}$
$V([\text{if b then } P_1 \text{ else } P_2]) = \{v_1, ..., v_n\} = V(P_1) \cup V(P_2) = \{u_1, ..., u_p, w_1, ..., w_q\}$

Es sei $w \in V([\text{if b then } P_1 \text{ else } P_2]) = V(P_1) \cup V(P_2)$
Fall1: falls N_B(b)(h) = wahr
$N([\text{if b then } P_1 \text{ else } P_2],h)(w) = N(P_1,h)(w)$
Unterfall1.1: $w \in V(P_1)$
$N(P_1,h)(w) \in p(h(u_1), ..., h(u_p))$, also
$N(P_1,h)(w) \in p(h(v_1), ..., h(v_n))$
Unterfall1.2: $w \notin V(P_1)$ und $w \in V(P_2)$
$N(P_1,h)(w) = h(w) \in p(h(v_1), ..., h(v_q))$, also
$h(w) \in p(h(v_1), ..., h(v_q))$,

Fall1: falls N_B(b)(h) = falsch
$N([\text{if b then } P_1 \text{ else } P_2],h)(w) = N(P_2,h)(w)$
Unterfall1.1: $w \in V(P_2)$
$N(P_2,h)(w) \in p(h(v_1), ..., h(v_q))$, also
$N(P_2,h)(w) \in p(h(v_1), ..., h(v_n))$
Unterfall1.2: $w \notin V(P_2)$ und $w \in V(P_1)$
$N(P_2,h)(w) = h(w) \in p(h(u_1), ..., h(u_p))$, also
$h(w) \in p(h(v_1), ..., h(v_q))$,