Uniwersytet Mikołaja Kopernika
w Toruniu
Wydział Matematyki i Informatyki

Systemy dowodzenia twierdzeń i ich implementacje w pakiecie "Isabelle"

Krzysztof Grąbczewski

Praca magisterska napisana w Zakładzie
Matematycznych Podstaw Informatyki
pod kierunkiem
prof. dra hab. Grzegorza Jarzembskiego

Toruń 1994

Spis treści

Wstęp

Istnieje wiele różnych nurtów logiki matematycznej. Najczęściej prezentowane są dwa z nich. Pierwszy skupia się na pojęciach spełniania i prawdziwości formuł (ang. satisfaction-based view of logic), natomiast drugi polega na szukaniu syntaktycznych dowodów formuł (ang. proof-based view of logic). Mimo zasadniczej różnicy, są to jednak tylko dwa różne spojrzenia na ten sam problem. Łączą je podstawowe pojęcia i definicje takie jak *język*, *formuły*, itd.

Kiedy za podstawę rozważań przyjmiemy pojęcie *relacji konsekwencji* (ang. consequence-based view), to łatwo zauważymy, że jest to uogólnienie obydwu wspomnianych wyżej ujęć. Jeśli w danej interpretacji zakładamy prawdziwość pewnych formuł aby badać prawdziwość innej formuły, to tak naprawdę wyznaczamy pewną relację konsekwencji. Podobnie kiedy szukamy syntaktycznego dowodu formuły ze zbioru przesłanek. Sprowadzając obydwa zagadnienia na wspólny grunt możemy formalnie pokazać, że są one sobie równoważne.

Mając pewność o równoważności semantycznej prawdziwości i syntaktycznej wywodliwości dla danego języka logicznego, możemy wykorzystać środki syntaktyczne do sprawdzania prawdziwości formuł (i odwotnie). Jeśli ponadto potrafimy zaimplementować metody wyprowadzania na komputerze, to otrzymyjemy automatyczny system stwierdzania o prawdziwości formuł.

Automatyczne dowodzenie twierdzeń jest pojęciem, które obejmuje wiele różnych sposobów budowania dowodów. Najlepsze systemy dowodzenia powinny stwarzać możliwość budowania dowodów według każdego ze sposobów. Poza tym system dowodzenia powinien mieć takie cechy jak :

- łatwość budowania nowych języków logiki,
- zrozumiały meta-język,
- przejrzyste rozumowanie na poziomie meta,
- możliwość ingerowania użytkownika w kształt dowodu,
- zgodność z semantyczną prawdziwością.

Nie oczekujemy zupełności, bowiem wiąże się to przeważnie z nieskończonością obliczeń w przypadku nieistnienia szukanego dowodu.

W pierwszym rozdziale pracy przedstawione są podstawowe dla dalszych rozważań definicje, takie jak język, systemy waluacji itp.

Drugi rozdział przedstawia systemy semantyczne i syntaktyczne pod kątem wyznaczonych przez nie relacji konsekwencji. Trzy najczęściej spotykane w logice systemy dowodzenia pokazane są na przykładach logiki klasycznej pierwszego rzędu. Pokazana jest ich zgodność i zupełność w stosunku do semantycznej prawdziwości, oraz ich wzajemna równoważność.

Trzeci rozdział przedstawia system dowodzenia twierdzeń "Isabelle", którego autorami są Lawrence C. Paulson z Laboratorium Komputerowego Uniwersytetu w Cambridge i Tobias Nipkow z Uniwersytetu Technicznego w Monachium. Rozdział ten pokazuje dowodzenie twierdzeń z punktu widzenia użytkownika, a także prezentuje niektóre aspekty techniczne pakietu i budowanie nowych systemów logicznych.

Podstawowe pojęcia i definicje użyte w tej pracy zostały zaczerpnięte z literatury. Dowód twierdzenia o zupełności został oparty na metodzie Schütte'a i dostosowany do przyjętych w pracy definicji. Większość dowodów syntaktycznych i przykładów oraz znaczna część dowodów równoważności systemów syntaktycznych, zostały stworzone w trakcie pisania pracy.

Serdeczne podziękowania za pomoc w tworzeniu tej pracy, za uwagi i sugestie składam panu profesorowi Grzegorzowi Jarzembskiemu.

Rozdział 1

Wiadomości wstępne

Pojęciem podstawowym dla rozważań na temat dowodzenia twierdzeń jest *język*. Rozdział ten wprowadza definicje języka zdaniowego i predykatowego oraz przedstawia ich interpretacje. Pełną interpretację języka stonowić będzie system waluacji interpretujący operatory języka wraz z przypisaniem interpretującym zmienne, termy, formuły atomowe.

1.1 Języki zdaniowe i predykatowe

Przedstawimy teraz podstawowe definicje, na których bazować będziemy rozważając interpretacje oraz systemy wyprowadzania [1]. Zdefiniujemy pojęcia języka, termów, formuł (lub zdań).

1.1.1 Język zdaniowy

Definicja 1.1 Językiem zdaniowym *nazywamy trójkę* $L = (P, O, arg)$*, gdzie* P *i* O *są dowolnymi zbiorami, natomiast* arg *jest dowolną funkcją o wartościach naturalnych określoną na zbiorze* O*. Zbiór* P *nazywamy zbiorem zdań atomowych,* O *zbiorem operatorów, funkcję* arg *argumentowością (lub arnością) operatorów.*

Definicja 1.2 Zbiór zdań *języka zdaniowego* $L = (P, O, arg)$ *to najmniejszy zbiór zawierający* P *i domknięty ze względu na operatory z* O*.*

Uwaga : Od tej pory symbolu L będziemy używać także dla oznaczenia zbioru zdań języka L.

Przykład 1.1 *Niech* $P = \{x\}$*,* $O = \{\prime, \oplus\}$*, oraz* $arg(\prime) = 1$*,* $arg(\oplus) = 2$*. Dla tak zdefiniowanego języka zdaniami są na przykład wyrażenia postaci :* x*,*$x \oplus x\prime$*,* $(x \oplus x)\prime$*,* $(x \oplus x\prime) \oplus x\prime\prime$ *...*

Uwaga : Nawiasy stosujemy dla uniknięcia dwuznaczności w określaniu zdań.

Przykład 1.2 *W klasycznej logice zdaniowej mamy :*

$P = \{p, q, r \ldots\}$

$O = \{\vee, \wedge, \neg, \rightarrow, \perp\}$

$arg(\vee) = arg(\wedge) = arg(\rightarrow) = 2$*,* $arg(\neg) = 1$*, oraz* $arg(\perp) = 0$*.*

Zbiór zdań dla klasycznej logiki zdaniowej to zbiór zawierający wszystkie elementy zbioru P*, oraz wyrażenia postaci :* $p \vee q$*,* $(p \wedge q) \rightarrow (r \rightarrow \perp)$ *...*

1.1.2 Język predykatowy

Definicja 1.3 Język predykatowy *to dowolna szóstka* $L = (P, T, V, O, Q, arg)$*, taka że :*

1. P, T, O, Q *to dowolne parami rozłączne zbiory*
2. $V \subseteq T$
3. $arg : P \cup O \to N$

Kolejne składowe języka predykatowego noszą następujące nazwy :

1. P to zbiór symboli predykatowych
2. T to zbiór termów
3. V to zbiór zmiennych
4. O to zbiór operatorów
5. Q to zbiór kwantyfikatorów
6. arg to argumentowość operatorów i predykatów

Porównując tę definicję z definicją 1.1 języka zdaniowego zbiory P i T odpowiadają zbiorowi zdań atomowych, ponieważ służą tworzeniu formuł atomowych. Natomiast zbiory O, V i Q odpowiadają operatorom języka zdaniowego jako, że budują formuły złożone.

Nie interesuje nas tu sposób budowania termów (zbiór termów jest z góry zadany). Przyjmujemy tylko, że są one budowane z wykorzystaniem zbioru zmiennych V, że potrafimy wyznaczyć zbiór zmiennych dowolnego termu oraz wykonać wewnątrz termu zastąpienie zmiennej termem.

Oznaczenie : $var(t)$ – zbiór zmiennych termu t.

Niech $L = (P, T, V, O, Q, arg)$ będzie dowolnym, ustalonym językiem predykatowym.

Definicja 1.4 Formuły atomowe *języka L powstają przy użyciu symboli predykatowych i termów zgodnie z następującą zasadą :*

$$\textit{jeśli } p \in P, arg(p) = n, t_1, \ldots, t_n \in T \textit{ to } p(t_1, \ldots, t_n) \textit{ jest formułą atomową.}$$

Jeśli $arg(p) = 0$ to p jest **stałą formułą atomową**.

Definicja 1.5 (Formuły i ich główne symbole logiczne) Zbiorem formuł języka L *nazywamy najmniejszy zbiór spełniający następujące własności :*

1. *Jeśli A jest formułą atomową to A jest formułą. Mówimy, że A nie ma głównego symbolu logicznego.*
2. *Jeśli $A_1, \ldots, A_n$ są formułami i o jest operatorem arności n to napis $o(A_1, \ldots, A_n)$ jest formułą. Mówimy, że jego głównym symbolem logicznym jest symbol o.*
3. *Jeśli A jest formułą, $q \in Q$ oraz $x \in V$ to napis qxA jest formułą. Głównym symbolem logicznym formuły qxA jest symbol q.*

Uwaga : Od tej pory symbolu L będziemy używać także dla oznaczenia zbioru formuł języka L.

Definicja 1.6 Zbiór zmiennych wolnych *formuły A (oznaczenie $free(A)$) określają następujące reguły :*

Niech $p \in P$, $arg(p) = n$, $o \in O$, $arg(o) = m$, $t_1, \ldots, t_n \in T$, $q \in Q$ oraz niech $A_1, \ldots, A_m \in L$

1. $free(p(t_1, \ldots, t_n)) = var(t_1) \cup \ldots \cup var(t_n)$
2. $free(o(A_1, \ldots, A_m)) = free(A_1) \cup \ldots\ free(A_m)$
3. $free(qxA) = free(A) - \{x\}$*. W tym przypadku każde wystąpienie zmiennej x w formule A nazywa się związanym.*

Uwaga : Definicja ta określa języki pierwszego rzędu tzn. kwantyfikujemy tylko termy (a dokładniej zmienne), a nie zbiory termów czy ich własności.

Przykład 1.3 (Klasyczna logika pierwszego rzędu) *Język predykatowy reprezentujący logikę klasyczną pierwszego rzędu to szóstka* $L = (P, T, V, O, Q, arg)$, *gdzie* $O = \{\vee, \wedge, \rightarrow, \neg, \perp\}$, $V = \{x, y, z, x_1, x_2, \ldots\}$, $Q = \{\forall, \exists\}$, *oraz* $arg(\vee) = arg(\wedge) = arg(\rightarrow) = 2$, $arg(\neg) = 1$, $arg(\perp) = 0$.

W ten sposób zdefiniowaliśmy nie jeden konkretny język, ale całą rodzinę języków klasycznej logiki pierwszego rzędu, bowiem nie określiliśmy zbiorów predykatów i termów. Budowaniu termów służą zazwyczaj symbole funkcyjne, ale na naszym poziomie rozważań nie wnikamy w wewnętrzną strukturę termów. Przyjmijmy teraz, że t_1, t_2, t_3 *są termami, oraz* $\{p, r\} \subseteq P$, *gdzie* $arg(p) = 2, arg(r) = 0$. *Wówczas formułami atomowymi dla tego języka będą napisy :* $p(t_1, t_3)$, $p(t_2, t_2)$, r ... *Do zbioru formuł będą należały na przykład napisy :* $\neg p(t_2, t_3)$, $p(t_1, t_2) \rightarrow r$, $\forall x p(t_1, t_1)$, ... *W pierwszych dwóch z wymienionych formuł wszystkie występujące w nich zmienne są wolne. W ostatniej zmienna* x *występuje w termach* t_1 *i* t_2 *jako związana, natomiast zbiór zmiennych wolnych tej formuły* $free(\forall x p(t_1, t_1)) = var(t_1) \cup var(t_2) - \{x\}$.

1.2 Systemy waluacji

Niniejszy podrozdział mówi o tym jak interpretujemy wprowadzone w poprzednim podrozdziale syntaktyczne definicje języka, formuły itd. System waluacji definiuje zbiór wartości logicznych i przyporządkowuje operatorom języka pewne funkcje. Funkcje te są interpretacją zbioru operatorów. Poza tym wyróżniamy pewne wartości logiczne i definiujemy spełnianie formuły jako przyjmowanie w danej interpretacji jednej z tych wyróżnionych wartości. Formuły atomowe i termy interpretujemy funkcją zwaną przypisaniem. Połączenie systemu waluacji i przypisania daje nam pełną interpretację języka. Pozwoli nam to mówić w następnym rozdziale o logicznej konsekwencji.

1.2.1 Język zdaniowy

Niech $L = (P, O, arg)$ będzie językiem zdaniowym.

Definicja 1.7 Systemem waluacji *języka* L *nazywamy każdą trójkę* $\underline{M} = (M, D, F)$, *gdzie :*

1. *M jest zbiorem o mocy $\geqslant 2$ (nazywamy go zbiorem wartości logicznych).*
2. *D jest niepustym podzbiorem zbioru M (zbiór wyróżnionych wartości logicznych).*
3. *$F = \{f_{o_1}, f_{o_2}, \ldots, f_{o_n}\}$ jest zbiorem funkcji odpowiadających symbolom operacji ze zbioru $O = \{o_1, o_2, \ldots, o_n\}$ takim, że dla dowolnego $o \in O$ $f_o : M^{arg(o)} \rightarrow M$.*

Mówimy, że f_o interpretuje operator o a zbiór F jest interpretacją zbioru O.

Wychodząc od takiej definicji systemu waluacji skupiamy się na własnościach operatorów, nie wnikamy natomiast w strukturę zdań atomowych (podobnie jak w programach, gdzie nie interesuje nas np. wewnętrzna struktura pętli while, for, czy instrukcji if, ale budujemy z nich większe struktury, które są dość dokładnie analizowane).

Definicja 1.8 Przypisaniem *w systemie waluacji* $\underline{M} = (M, D, F)$ *dla języka* L *nazywamy każdą funkcję* $a : P \rightarrow M$.

Definicja 1.9 Interpretacją *języka* L *w systemie waluacji* $\underline{M} = (M, D, F)$ *dla przypisania* a *nazywamy funkcję* $v_a : L \rightarrow M$ *określoną następująco :*

1. *$v_a(p) = a(p)$ dla $p \in P$.*
2. *$v_a(o(A_1, \ldots, A_n)) = f_o(v_a(A_1), \ldots, v_a(A_n))$, gdzie $n = arg(o)$ (arność operatora o) oraz f_o interpretuje operator o w systemie $\underline{M}$.*

Uwaga : W dalszej części będziemy często pisać np. *dla każdej interpretacji* $v_a \ldots$, co będzie równoważne zwrotowi *dla każdego przypisania* $a \ldots$, ale zwiększy czytelność niektórych określeń.

Definicja 1.10 *Zdanie A jest* **spełnione** *w systemie waluacji $\underline{M} = (M, D, F)$ przy interpretacji v_a , o ile $v_a(A) \in D$.*

Definicja 1.11 *Zdanie A jest* **prawdziwe** *(jest* **tautologią***) w systemie waluacji $\underline{M} = (M, D, F)$, o ile jest spełnione w każdej interpretacji.*

Systemy waluacji nie określają wartości logicznych zdań atomowych języka, a tylko opisują operatory tego języka. Zatem system waluacji to klasa interpretacji języka, które w ten sam sposób interpretują operatory, natomiast mogą różnić się interpretacją zdań atomowych.

Interpretacja języka jest więc jednoznacznie wyznaczona przez system waluacji oraz przypisanie. Funkcja przypisania interpretuje zdania atomowe, a system waluacji operatory.

Zauważmy, że zbiór formuł języka zdaniowego L to algebra wolna typu O nad alfabetem P. System waluacji $\underline{M} = (M, D, F)$ możemy traktować jak algebrę (M, F). Zatem przypisanie jest zanurzeniem generatorów, a waluacja jego jednoznacznym rozszerzeniem do homomorfizmu algebr.

Przykład 1.4 *Weźmy język zdaniowy z przykładu 1.2, określony dla klasycznej logiki zdaniowej.*

Systemem waluacji dla tego języka jest system $\underline{M} = (M_2, D, F)$, gdzie $M_2 = \{t, f\}$, $D = \{t\}$, a $F = \{f_\vee, f_\wedge, f_\neg, f_\rightarrow, f_\perp\}$, przy czym funkcje wchodzące w skład F określone są tak jak dla algebry Boole'a, przy czym dodatkowo $F_\perp = f$.

Dla tego samego języka określmy inny system waluacji $\underline{M} = (M_3, D, F)$ ustalając trójelementowy zbiór wartości logicznych $M = \{t, f, u\}$. Element u interpretujemy jako wartość nieokreśloną. Wówczas elementy zbioru F są funkcjami, które można definiować na wiele różnych sposobów. Funkcje $f_\vee$ i $f_\wedge$ mogą obrazować porządek w trójelementowej kracie Boole'a. Dla każdego z przedstawionych poniżej porządków otrzymamy inny system waluacji.

$$\begin{array}{c} t \\ \uparrow \\ u \\ \uparrow \\ f \end{array} \qquad\qquad \begin{array}{ccc} t & & f \\ \nwarrow & & \nearrow \\ & u & \end{array}$$

porządek prawdziwości *porządek określoności*

Jeszcze inny system waluacji otrzymamy przyjmując, że wartości logiczne formuł złożonych obliczamy sekwencyjnie. Wartość logiczna u oznacza, że algorytm nie daje odpowiedzi (nie kończy się). Sekwencyjność obliczania powoduje, że jeśli zdanie p nie jest określone, to zdanie $p \vee q$ też nie jest określone, nawet jeśli q ma wartość t, (w przypadku obliczania równoległego otrzymalibyśmy dla tego samego zdania $p \vee q$ wynik t). Zbiór F w tym przypadku będzie wyglądał następująco :

$f_\vee$	t	u	f
t	t	t	t
u	u	u	u
f	t	u	f

$f_\wedge$	t	u	f
t	t	u	f
u	u	u	u
f	f	f	f

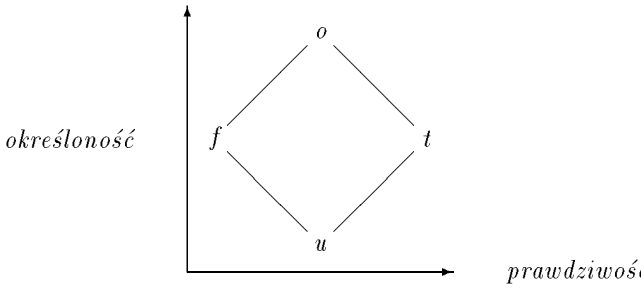
$f_\rightarrow$	t	u	f
t	t	u	f
u	u	u	u
f	t	t	t

$f_\neg$	
t	f
u	u
f	t

$f_\perp$ jest określone jak dla M_2.

Również zmieniając podzbiór D otrzymujemy inny system waluacji. Na przykład gdy interesuje nas fakt prawdziwości zdania, to przyjmiemy $D = \{t\}$, natomiast jeśli chcemy rozstrzygać czy dane zdanie ma określoną wartość logiczną, to weźmiemy $D = \{t, f\}$.

Niech teraz $M_4 = \{u, t, f, o\}$. Wartość o możemy interpretować jako prawdę i fałsz jednocześnie (tak jak u nie jest ani prawdą, ani fałszem). Podobnie jak dla M_3 mamy tutaj porządek prawdziwości i określoności.



Często eksponuje się porządek określoności zapisując zbiór M_4 w postaci $\{\emptyset, \{t\}, \{f\}, \{t, f\}\}$. Funkcje ze zbioru F definiujemy wówczas następująco ($f_\perp = f$) :

$f_\vee$	t	u	f	o
t	t	t	t	t
u	t	u	u	t
f	t	u	f	o
o	t	t	o	o

$f_\wedge$	t	u	f	o
t	t	u	f	o
u	u	u	f	f
f	f	f	f	f
o	o	f	f	o

$f_\rightarrow$	t	u	f	o
t	t	f	f	f
u	t	t	f	?
f	t	t	t	t
o	t	?	f	t

$f_\neg$	
t	f
u	u
f	t
o	o

Z wyjątkiem dwóch przypadków (oznaczonych znakiem ?), w których wartości należy ustalać wyłącznie według własnego uznania, wszystkie są zgodne z intuicją (funkcja $f_\rightarrow$ obrazuje relację $\leqslant$ dla porządku prawdziwości).

Przykład 1.4 ilustruje jak można rozszerzać zbiór wartości logicznych. Idąc dalej można budować systemy waluacji, w których zbiór ten jest nieskończony, a nawet nieprzeliczalny [1].

1.2.2 Język predykatowy

Definicja 1.12 *Niech $L = (P, T, V, O, Q, arg)$ będzie językiem predykatowym (definicja 1.3).* **Systemem waluacji** *dla języka L nazywamy czwórkę $\underline{M} = (M, D, F, G)$ określoną następująco :*

1. *M jest zbiorem co najmniej dwuelementowym*
2. *D jest niepustym, właściwym podzbiorem zbioru M*
3. *$F = \{f_o : M^{arg(o)} \rightarrow M; o \in O\}$ (interpretacja operatorów)*
4. *$G = \{g_q : 2^M \rightarrow M; q \in Q\}$ (interpretacja kwantyfikatorów)*

Uwaga : Zbiory M, D, F mają takie samo znaczenie jak w systemie waluacji dla języka zdaniowego. Dodatkowy zbiór G jest zbiorem funkcji interpretujących kwantyfikatory.

Także przypisanie dla języka predykatowego musi wyglądać nieco inaczej niż dla języka zdaniowego, gdyż zbiór formuł atomowych w nowym języku nie jest zadany z góry, ale tworzony przy użyciu symboli predykatowych i termów. Zatem przypisanie musi teraz wartościować także termy.

Definicja 1.13 **Przypisaniem** *dla języka predykatowego $L = (P, T, V, O, Q)$ w systemie waluacji $\underline{M} = (M, D, F, G)$ nazywamy parę (a, I) taką, że I jest niepustym zbiorem (zwanym dziedziną indywiduową), natomiast $a : T \cup V \cup P \rightarrow I \cup \bigcup_{n \in N}(I^n \rightarrow M)$ jest funkcją spełniającą następujące warunki :*

1. *Jeśli $t \in T$ to $a(t) \in I$*
2. *Jeśli $x \in V$ to $a(x) \in I$*
3. *Jeśli $p \in P, arg(p) = n$ to $a(p) : I^n \rightarrow M$*

Uwaga : W dalszej części będziemy utożsamiać przypisanie (a, I) z funkcją a.

Proponowane poniżej oznaczenie jest pomocne w opisie interpretacji formuł.

Oznaczenie : Niech a i a' będą przypisaniami o tej samej dziedzinie indywiduowej, oraz niech $x \in V$. Piszemy $a \sim_x a'$ o ile spełnione są następujące warunki :

1. Jeśli $t \in T$ oraz $x \notin var(t)$ to $a(t) = a'(t)$
2. Jeśli $y \in V$ oraz $y \neq x$ to $a(y) = a'(y)$
3. Jeśli $p \in P$ to $a(p) = a'(p)$

Definicja 1.14 *Niech $\underline{M} = (M, D, F, G)$ będzie systemem waluacji dla języka predykatowego L, oraz niech a będzie przypisaniem w $\underline{M}$.* **Waluacją** *nazywamy funkcję $v_a : L \to M$ określoną następująco :*

1. *Jeśli $p \in P, arg(p) = n$ oraz $t_1, \ldots, t_n \in T$ to $v_a(p(t_1, \ldots, t_n)) = a(p)(a(t_1), \ldots, a(t_n))$*
2. *Jeśli $o \in O, arg(o) = m$ oraz $A_1, \ldots, A_m \in L$ to $v_a(o(A_1, \ldots, A_n)) = f_o(v_a(A_1), \ldots, v_a(A_n))$*
3. *Jeśli $q \in Q, x \in V, A \in L$ to $v_a(qxA) = g_q(\{v_{a'}(A); a' \sim_x a\})$*

Zauważmy, że wartość logiczna formuły qxA dla przypisania a nie zależy od wartości tego przypisania dla zmiennej x.

Definicja 1.15 *Formuła A jest* **spełniona** *w systemie waluacji $\underline{M} = (M, D, F, G)$ przy interpretacji v_a , o ile $v_a(A) \in D$.*

Definicja 1.16 *Formuła A jest* **prawdziwa w systemie waluacji** $\underline{M} = (M, D, F, G)$, *o ile jest spełniona w każdej interpretacji v_a.*

Przykład 1.5 (Logika klasyczna pierwszego rzędu) *Niech $L = (P, T, V, O, Q, arg)$ będzie językiem logiki klasycznej pierwszego rzędu (zdefiniowanym w przykładzie 1.3). System waluacji dla tego języka to czwórka $\underline{M} = (M, D, F, G)$, taka, że $M = \{t, f\}$, $D = \{t\}$, F jest zdefiniowany jak dla języka zdaniowego, natomiast $G = \{g_\forall, g_\exists\}$, gdzie :*

$$g_\forall(X) = \begin{cases} f & \text{o ile } f \in X \\ t & \text{w przeciwnym wypadku} \end{cases}$$
$$g_\exists(X) = \begin{cases} t & \text{o ile } t \in X \\ f & \text{w przeciwnym wypadku} \end{cases}$$

Przyjrzyjmy się interpretacji formuł z kwantyfikatorami dla konkretnego języka. Ustalmy $T = V = \{x, y\}$, $P = \{p, r\}$. Niech poza tym $arg(p) = 1$, oraz $arg(r) = 0$. Zauważmy, że w tak ustalonym języku mamy tylko trzy formuły atomowe : $p(x)$, $p(y)$, r.

Niech a będzie przypisaniem takim, że : $a(x) = t$, $a(y) = f$, $a(r) = t$, oraz $a(p) = id_{M_2}$.

Wówczas

$v_a(\forall x p(x)) = g_\forall(\{v_{a'}(p(x)); a' \sim_x a\}) = g_\forall(\{t, f\}) = f$

$v_a(\exists x p(x)) = g_\exists(\{t, f\}) = t$

$v_a(\exists y r) = g_\exists(\{v_{a'}(r); a' \sim_x a\}) = g_\exists(\{t\}) = t$

Rozdział 2

Relacje konsekwencji

Centralnym pojęciem tego rozdziału jest pojęcie relacji konsekwencji. W postaci takiej relacji przedstawimy zarówno *satisfaction–based view of logic* jak i *proof–based view*. Pozwoli to na traktowanie na tym samym poziomie obydwu ujęć logiki, a zatem na wykazywanie, że syntaktyczne metody dowodzenia rzeczywiście zachowują własności logicznej konsekwencji, a co za tym idzie pozwalają zautomatyzować dowodzenie.

2.1 Relacja konsekwencji

Niektóre pojęcia definiujemy tak samo dla języka zdaniowego i predykatowego. W takich przypadkah piszemy po prostu *język* mając na uwadze, że odnosi się to zarówno do jednego jak i do drugiego.

Definicja 2.1 *Niech L będzie językiem, natomiast Γ dowolnym zbiorem zdań (formuł) tego języka.*

Relacją konsekwencji *dla języka L nazywamy każdą relację $\vdash \subseteq 2^L \times L$ spełniającą następujące własności :*

inkluzja : *jeśli $A \in \Gamma$ to $\Gamma \vdash A$*

monotoniczność : $\dfrac{\Gamma \vdash A}{\Gamma, \Delta \vdash A}$

obcinanie : $\dfrac{\Gamma \vdash C \quad \Delta, C \vdash A}{\Delta, \Gamma \vdash A}$

Przecinek oddzielający zbiory po lewej stronie symbolu $\vdash$ traktujemy jako sumę mnogościową. Formułę stojącą z lewej strony symbolu $\vdash$ traktujemy jako zbiór jednoelementowy.

Kreskę poziomą czytamy następująco : *jeśli zachodzi relacja zapisana nad kreską to zachodzi również relacja zapisana pod kreską.*

Pojęcie relacji konsekwencji bywa definiowane na wiele różnych sposobów, ale zazwyczaj są to rozszerzenia powyższej listy warunków. Taką definicję przedstawiono w [1].

Przykład 2.1 *Dla dowolnie zdefiniowanego języka L relacja $\ni$ określona (w sposób standardowy) dla $2^L \times L$ jest relacją konsekwencji. Jest to najmniejsza relacja konsekwencji języka L.*

Przykład 2.2 *Niech X będzie dowolnym zbiorem, oraz $(\Omega X, (q : q \in \Omega))$ algebrą wolną nad alfabetem X taką, że $\Omega = \Omega_2 = \{=\}$. Język L w tym przypadku to zbiór wszystkich Ω-równości postaci $t_1 = t_2$. Ponieważ nie mamy w tym języku operatorów logicznych, to zbiór formuł jest zbiorem formuł atomowych. Definiujemy relację $\vdash \in 2^L \times L$ następująco : jeśli $D \subset L$, oraz $t \in L$, to :*

$$D \vdash t \overset{def}{\Leftrightarrow} t_1 \rho_D t_2,$$

gdzie ρ_D jest najmniejszą kongruencją inwariantną zawierającą D.

Tak określona relacja $\vdash$ jest relacją konsekwencji.

Dowód : Własność inkluzji jest oczywista. Monotoniczność wynika z faktu, że $\rho_\Gamma \in \rho_{\Gamma\cup\Delta}$ dla dowolnych $\Gamma, \Delta \in L$.

Dla pokazania własności obcinania załóżmy, że $\Gamma \vdash t = p$, oraz $\Delta, t = p \vdash r = s$. Inaczej $t = p \in \rho_\Gamma$ i $r = s \in \rho_{\Delta\cup\{t=p\}}$. Z monotoniczności i inkluzji mamy, że $t = p \in \rho_{\Gamma\cup\Delta}$, oraz $\Delta \in \rho_{\Gamma\cup\Delta}$. A zatem pisząc

$$r = s \in \rho_{\Delta\cup\{t=p\}} \in \rho_{\Gamma\cup\Delta}$$

otrzymujemy tezę.

Definicja 2.2 *Relacja konsekwencji $\vdash$ jest* **zwarta** *o ile spełnia następujący warunek :*
Jeśli $\Gamma \vdash A$, to istnieje skończony podzbiór $\Gamma_0 \subseteq \Gamma$ taki, że $\Gamma_0 \vdash A$.

Ograniczając się do rozpatrywania relacji zwartych możemy wyciągać konkluzje tylko ze skończonej liczby przesłanek. Jest to jednak ograniczenie bardzo "naturalne", a poza tym daje możliwość analizowania relacji konsekwencji przy pomocy komputera.

Definicja 2.3 *Niech $L = (P, O, arg)$ będzie językiem zdaniowym.* **Podstawieniem** *dla języka L nazywamy każdą funkcję $* : P \to L$*

Ponieważ zbiór formuł to algebra wolna typu O nad zbiorem P, więc każde podstawienie można jednoznacznie rozszerzyć do funkcji działającej ze zbioru wszystkich formuł języka L (tzn. do homomorfizmu).

W dalszej części będziemy używali dla tego rozszerzenia takiego samego oznaczenia jak dla rozszerzanego podstawienia (tzn. $*$).

Definicja 2.4 *Niech $L = (P, T, V, O, Q, arg)$ będzie językiem predykatowym.* **Podstawieniem** *dla języka L nazywamy każdą funkcję $* : V \to T$*

Uwaga : Każde podstawienie można jednoznacznie rozszerzyć do funkcji działającej ze zbioru T. Rozszerzenie to w sposób jednoznaczny wyznacza funkcję odwzorowującą P w L, a ta da się rozszerzyć do endomorfizmu algebry L.

W dalszej części będziemy używali dla wszystkich tych rozszerzeń takiego samego oznaczenia jak dla pierwotnego podstawienia (tzn. $*$).

Oznaczenie : Niech $A \in L$ oraz $\Gamma \subset L$. Dla uproszczenia dalszych rozważań przyjmujemy następujące skróty :

$$A^* \stackrel{ozn}{=} *(A)$$

$$\Gamma^* \stackrel{ozn}{=} \{B^*; B \in \Gamma\}$$

Definicja 2.5 *Relacja konsekwencji $\vdash$ ma* **własność podstawiania** *o ile dla dowolnego podstawienia $*$ relacja $\Gamma \vdash A$ pociąga za sobą relację $\Gamma^* \vdash A^*$.*

Definicja 2.6 *Relacja konsekwencji $\vdash$ jest* **standardowa** *o ile ma własności zwartości i podstawiania.*

Definicja 2.7 *Relacja konsekwencji $\vdash$ ma* **własność dedukcji** *o ile istnieje w języku L operator binarny "$\to$" taki, że dla dowolnych zdań A, B i zbioru zdań Γ zachodzi*

$$\Gamma \vdash A \to B \quad \Leftrightarrow \quad \Gamma, A \vdash B$$

Własność dedukcji jest bardzo istotna dla dowodzenia twierdzeń w logice, gdyż wiąże implikację z poziomu **meta–logiki** oraz implikację jako operator w danej logice.

Przykład 2.3 *Relacja $\ni$ z przykładu 2.1 ma własności zwartości i podstawiania.*

Własność dedukcji zależna jest od operacji $\to$ określonej dla konkretnego języka L i dlatego nie można w tak ogólnym przypadku powiedzieć czy relacja $\ni$ ma tę własność. Każdy konkretny język należy rozpatrywać osobno.

Przykład 2.4 *Relacja określona w przykładzie 2.2 ma własności zwartości i podstawiania.*

Dowód : Aby udowodnić zwartość wystarczy pokazać, że :

$$\rho_D = \bigcup_{\substack{D_0 \subset D \\ D_0 - sko\acute{n}czony}} \rho_{D_0}$$

Równość ta wynika z faktu, że $D \subset \cup \rho_{D_0}$ oraz $\cup \rho_{D_0}$ jest kongruencją.
Własność podstawiania jest wynikiem inwariantności kongruencji ρ_D.

2.2 Relacja logicznej konsekwencji

Korzystając z pojęcia systemów waluacji możemy wprowadzić pojęcie logicznej konsekwencji.

Definicja 2.8 *Niech L będzie dowolnym językiem, Γ pewnym zbiorem zdań (formuł) tego języka, oraz $\underline{M}$ systemem waluacji dla języka L o zbiorze wartości logicznych M i wyróżnionym podzbiorze D.*

Mówimy, że A jest logiczną konsekwencją zbioru Γ w systemie waluacji $\underline{M}$ (piszemy $\Gamma \models_{\underline{M}} A$) wtedy i tylko wtedy, gdy dla każdego przyporządkowania a

jeśli dla dowolnego $B \in \Gamma$, $v_a(B) \in D$ to $v_a(A) \in D$.

Twierdzenie 2.1 *Relacja logicznej konsekwencji ($\models_{\underline{M}}$) jest relacją konsekwencji.*

Dowód : Musimy pokazać, że relacja $\models_{\underline{M}}$ spełnia własności inkluzji, monotoniczności i obcinania. Weźmy dowolne przyporządkowanie a.

inkluzja : Niech $A \in \Gamma$. Jeśli $v_a(B) \in D$ dla dowolnego $B \in \Gamma$ to oczywiście $v_a(A) \in D$. A zatem $\Gamma \models_{\underline{M}} A$.

monotoniczność : Niech $\Gamma \models_{\underline{M}} A$. Musimy pokazać, że $\Gamma, \Delta \models_{\underline{M}} A$. Jeśli $v_a(B) \in D$ dla wszystkich $B \in \Gamma \cup \Delta$ to także $v_a(B') \in D$ dla wszystkich $B' \in \Gamma$. Z faktu, że $\Gamma \models_{\underline{M}} A$ i z powyższego mamy, że $v_a(A) \in D$, a zatem $\Gamma, \Delta \models_{\underline{M}} A$.

własność obcinania : Niech $\Gamma \models_{\underline{M}} C$ oraz $\Delta, C \models_{\underline{M}} A$. Należy pokazać, że $\Delta, \Gamma \models_{\underline{M}} A$. Zakładamy, że $v_a(B) \in D$ dla wszystkich $B \in \Delta \cup \Gamma$. Pokażemy, że $v_a(A) \in D$. Z założenia wynika, że także $v_a(B') \in D$ dla $B' \in \Gamma$. Ponieważ $\Gamma \models_{\underline{M}} C$ więc $v_a(C) \in D$. Stąd i z tego, że $v_a(B'') \in D$ dla $B'' \in \Delta$ mamy ostatecznie, że $v_a(A) \in D$.

Przy założeniach, że L jest dowolnym językiem a $\underline{M}$ dowolnym systemem waluacji języka L można udowodnić, że relacja $\models_{\underline{M}}$ ma własność podstawiania [1]. Natomiast nie da się przy tak ogólnych założeniach wykazać, że relacja $\models_{\underline{M}}$ spełnia własności zwartości czy redukcji. Takie tezy należy dowodzić osobno dla konkretnych systemów waluacji.

2.3 Relacja syntaktycznej konsekwencji

Przedmiotem zainteresowania *teorii dowodów* jest syntaktyczny opis relacji konsekwencji. Definiowanie relacji konsekwencji polega tu na podaniu reguł i środków budowania *dowodów*. Konkluzje budowane są z przesłanek metodami syntaktycznymi tworząc listę czy drzewo zwane dowodem. Sposoby tworzenia takich dowodów mogą być różne. Nazywamy je *reprezentacjami* (bądź też *prezentacjami*). Najczęściej w logice spotykamy trzy takie reprezentacje :

- reprezentacja Hilberta
- naturalna dedukcja
- reprezentacja sekwentowa

Każda z tych reprezentacji definiuje na swój sposób pojęcie dowodu, stosuje inne środki do budowania dowodów, ale wspólne dla nich wszystkich jest pojęcie *relacji konsekwencji syntaktycznej.*

Definicja 2.9 *Zdanie (formuła) A jest* **konsekwencją syntaktyczną** *zbioru zdań (formuł) Γ wtedy i tylko wtedy, gdy istnieje dowód zdania (formuły) A z przesłanek Γ.*

Wystarczy więc sprecyzować pojęcie dowodu, aby jednoznacznie określić relację konsekwencji syntaktycznej. Możemy zatem mówić, że każda reprezentacja jednoznacznie wyznacza pewną relację konsekwencji.

Uwaga : W dalszej części będziemy pisać po prostu *formuła* zamiast *zdanie (formuła)* pamiętając o tym, że wprowadzane definicje dotyczą zarówno języka zdaniowego jak i predykatowego.

Aby ująć w jedną całość formułę dowodzoną i przesłanki wprowadza się pojęcie sekwentu.

Definicja 2.10 Sekwentem *nazywamy wyrażenie postaci $\Gamma \vdash \Delta$, gdzie Γ, Δ są zbiorami formuł. Zbiory Γ i Δ nazywamy odpowiednio poprzednikiem i następnikiem sekwentu $\Gamma \vdash \Delta$.*

Uwaga : Od tej pory symbol $\vdash$ będzie elementem syntaktyki dowodów, a nie oznaczeniem relacji.

Przecinek użyty w zapisie sekwentu rozumiemy jako sumę mnogościową, a pojedyncze formuły jako jednoelementowe zbiory formuł. Zamiast $\emptyset \vdash \Delta$ często piszemy $\vdash \Delta$.

Zamiast mówić o dowodzie formuły A z przesłanek Γ możemy mówić o dowodzie sekwentu $\Gamma \vdash A$.

2.3.1 Reprezentacja Hilberta

Reprezentacja Hilberta (zwana również aksjomatyczną) opisuje relację konsekwencji przez zbiór aksjomatów, oraz skończony zbiór reguł dowodzenia. Zbiór aksjomatów to zbiór wyróżnionych formuł, natomiast reguły dowodzenia to napisy postaci :

$$\frac{A_1 \ \dots \ A_n}{A}$$

gdzie $A_1, \dots, A_n, A$ są formułami. Formuły $A_1, \dots, A_n$ nazywamy przesłankami, a formułę A konkluzją. Aksjomaty możemy traktować jako reguły bez przesłanek.

Definicja 2.11 Dowodem formuły A z przesłanek Γ *nazywamy skończony ciąg formuł zakończony formułą A taki, że każda formuła tego ciągu spełnia jeden z poniższych warunków :*

1. *jest jednym z aksjomatów,*
2. *jest elementem zbioru Γ,*
3. *można ją otrzymać z poprzedzających ją formuł stosując jedną z reguł dowodzenia.*

Definicja 2.12 *Formuła A jest* **twierdzeniem** *gdy istnieje dowód formuły A z pustego zbioru przesłanek (tzn. $\emptyset \vdash A$).*

Twierdzenie 2.2 *Relacja wyznaczona przez reprezentację Hilberta jest relacją konsekwencji.*

Dowód : Własności inkluzji i monotoniczności są oczywiste.

Aby pokazać własność obcinania wystarczy zauważyć, że jeśli mamy dowód formuły A z przesłanek $\Delta \cup \{C\}$ (ciąg $\Delta, \dots, A$), oraz dowód formuły C z przesłanek Γ (ciąg $\Gamma, \dots, C$) to dowodem formuły A z przesłanek $\Delta \cup \Gamma$ jest ciąg $\Delta, \Gamma, \dots, C, \dots, A$.

Twierdzenie 2.3 *Relacja konsekwencji opisana w reprezentacji Hilberta jest zwarta.*

W konkretnych językach można także dowieść własności podstawiania i dedukcji.

Przykład 2.5 (Klasyczny język zdaniowy) *Relację konsekwencji syntaktycznej dla klasycznej logiki zdaniowej (przykład 1.2) opisują następujące trzy schematy aksjomatów [1] :*

$A1: \quad A \to (B \to A)$

$A2: \quad (A \to (B \to C)) \to ((A \to B) \to (A \to C))$

$A3: \quad (\neg A \to \neg B) \to (B \to A)$

oraz jedną regułę wnioskowania zwaną modus ponens *:*

$$\frac{A \quad A \to B}{B}$$

Użyliśmy pojęcia "schematy aksjomatów", bowiem każdy z nich opisuje całą rodzinę aksjomatów powstałą przez zastępowanie A, B, C przez konkretne zdania. Stąd wynika, że opisana relacja konsekwencji ma własność podstawiania. Jeśli zbiór zdań rozważanego języka jest nieskończony, to zbiór aksjomatów także (każdy schemat opisuje nieskończoną ilość aksjomatów).

W aksjomatach i regule nie używaliśmy symboli $\vee, \wedge$ oraz $\perp$, bowiem definiujemy je za pomocą $\to$ i $\neg$ następująco : $A \vee B \equiv \neg(A \to B)$, $A \wedge B \equiv \neg(A \to \neg B)$, $\perp \equiv \neg(A \to A)$.

Poniżej zamieszczony jest przykład dowodu – dowód twierdzenia $\neg B \to (B \to A)$

1.	$\neg B \to (\neg A \to \neg B)$	$A1$
2.	$(\neg A \to \neg B) \to (B \to A)$	$A3$
3.	$((\neg A \to \neg B) \to (B \to A)) \to$ $(\neg B \to ((\neg A \to \neg B) \to (B \to A)))$	$A1$
4.	$\neg B \to ((\neg A \to \neg B) \to (B \to A))$	$M.P.2,3$
5.	$(\neg B \to ((\neg A \to \neg B) \to (B \to A))) \to$ $((\neg B \to (\neg A \to \neg B)) \to (\neg B \to (B \to A)))$	$A2$
6.	$(\neg B \to (\neg A \to \neg B)) \to (\neg B \to (B \to A))$	$M.P.4,5$
7.	$\neg B \to (B \to A)$	$M.P.1,6$

Twierdzenie 2.4 *Relacja konsekwencji wyznaczona przez system Hilberta klasycznego języka zdaniowego spełnia własność dedukcji.*

Dowód : *Jeśli istnieje dowód sekwentu $\Gamma \vdash A \to B$ to stosując raz regułę* modus ponens *otrzymamy dowód sekwentu $\Gamma, A \vdash B$.*

Załóżmy, że istnieje dowód sekwentu $\Gamma, A \vdash B$. Musimy udowodnić, że istnieje dowód sekwentu $\Gamma \vdash A \to B$. Pokażemy to indukcją ze względu na długość dowodu sekwentu $\Gamma, A \vdash B$.

1. *Jeśli długość dowodu jest równa 1, to sekwent dowodzony ma postać $B \vdash B$. Wystarczy więc pokazać, że istnieje dowód sekwentu $\vdash B \to B$. Wygląda on następująco :*

1.	$B \to ((A \to B) \to B)$	$A1$
2.	$(B \to ((A \to B) \to B)) \to ((B \to (A \to B)) \to (B \to B))$	$A2$
3.	$(B \to (A \to B)) \to (B \to B)$	*M.P. 1,2*
4.	$B \to (A \to B)$	$A1$
5.	$B \to B$	*M.P. 3,4*

2. *Załóżmy, że własność zachodzi dla dowodów długości $\leqslant n$. Niech* **D** *będzie dowodem sekwentu $\Gamma, A \vdash B$ długości $n+1$. Może zaistnieć jeden z dwóch przypadków :*

   (a) *Ostatnia formuła dowodu jest aksjomatem lub elementem zbioru przesłanek. Jeśli $B = A$ to szukany dowód łatwo otrzymać z dowodu przedstawionego w pierwszym punkcie indukcyjnym. W przeciwnym przypadku $B \in \Gamma$, a wtedy istnienie dowodu sekwentu $\Gamma \vdash A \to B$ jest oczywiste.*

   (b) *Ostatnią formułę otrzymano stosując regułę* modus ponens, *a zatem wśród formuł dowodu muszą występować dwie postaci C oraz $C \to B$, gdzie C jest pewną formułą. Zatem dowód* **D** *można skrócić do dowodów sekwentów $\Gamma, A \vdash C$ oraz $\Gamma, A \vdash C \to B$. Obydwa te dowody mają długości $\leqslant n$, stąd z założenia indukcyjnego istnieją dowody* *sekwentów* $\Gamma \vdash A \to C$ *oraz* $\Gamma \vdash A \to (C \to B)$. *Wystarczy więc pokazać, że z przesłanek* $A \to C$ *i* $A \to (C \to B)$ *można wyprowadzić formułę* $A \to B$. *Czyni to następujący dowód :*

1.	$A \to C$	
2.	$A \to (C \to B)$	
3.	$A \to (C \to B) \to ((A \to C) \to (A \to B))$	*A2*
4.	$(A \to C) \to (A \to B)$	*M.P. 2,3*
5.	$(A \to B)$	*M.P. 1,4*

Przykład 2.6 (Klasyczny język predykatowy) *Rozważmy język predykatowy logiki klasycznej pierwszego rzędu (zdefiniowany w przykładzie 1.3).*

Definicja 2.13 *Niech* A *będzie formułą,* x *zmienną. Term* t *jest wolny dla* x *w* A *wtedy i tylko wtedy, gdy* x *nie występuje w sposób wolny w* A *w zasięgu kwantyfikatora wiążącego zmienną ze zbioru* $var(t)$.

Innymi słowy term t *jest wolny dla zmiennej* x *w formule* A *o ile podstawiając w miejsce każdego wystąpienia zmiennej* x *w* A *term* t*, "nie zmieniamy sposobu działania kwantyfikatorów". Na przykład dla formuły* $\forall xp(y)$ *term* z *(zmienna) jest wolny dla* y*, bo po podstawieniu otrzymamy formułę* $\forall xp(z)$*, ale term* x *nie jest wolny dla* y *bo podstawiając otrzymujemy* $\forall xp(x)$*, co zasadniczo zmieniło sposób oddziaływania kwantyfikatora.*

Reprezentacja Hilberta dla klasycznego języka predykatowego jest rozszerzeniem reprezentacji dla języka zdaniowego. Do zbioru aksjomatów (A1-A3) dodajemy następujące trzy [1] :

$A4: \ (\forall xA) \to A$

$A5: \ (\forall xA(x)) \to A(t)$, *o ile* t *jest wolny dla* x *w* $A(x)$

$A6: \ \forall x(A \to B) \to (A \to \forall xB)$, *o ile* x *nie występuje w* A *w sposób wolny*

Do zbioru reguł dodajemy następującą (zwaną regułą uogólnienia*) :*

$$\frac{A}{\forall xA}$$

.

Systemy Hilberta mają jednak kilka wad, przez które nie są one często wykorzystywane. Budowanie dowodów w tej reprezentacji jest skomplikowane. Dowody (nawet dla bardzo prostych formuł) są mało przejrzyste co pokazał dowód przedstawiony powyżej (dla logiki zdaniowej).

2.3.2 Naturalna dedukcja

W reprezentacji naturalnej dedukcji nie istnieje pojęcie aksjomatu są tylko reguły wnioskowania. Brak aksjomatów zastępuje się pewną taktyką stosowaną w dowodzeniu; w każdym miejscu dowodu można wprowadzić dowolne formuły jako hipotezy. Z takiej hipotezy można korzystać stosując reguły dowodzenia, ale tylko do jej wyeliminowania. Eliminowanie hipotezy odbywa się przez zastosowanie stosownej reguły (reguły eliminującej założenia). Jednym z częściej używanych sposobów przedstawiania graficznego wprowadzania i eliminowania hipotez jest "otwieranie i zamykanie prostokątów". Prostokąt jest fragmentem dowodu, w którym można używać otwierającej go hipotezy. Regułami eliminującymi hipotezy są reguły, które w swoim opisie zawierają prostokąty.

Formalny opis dowodu przedstawia następująca definicja.

Definicja 2.14 Dowodem *formuły* A *z przesłanek* Γ *(w reprezentacji naturalnej dedukcji) nazywamy skończony ciąg formuł taki, że*

1. *Ostatnią formułą ciągu jest formuła* A

2. *Każda z formuł ciągu spełnia jeden z poniższych warunków :*

- *jest elementem zbioru* Γ
- *powstaje z formuł poprzedzających ją przez zastosowanie jednej z reguł wnioskowania bądź przez przepisanie*
- *jest dowolną inną formułą i wtedy mówimy, że jest hipotezą*

3. Wszystkie hipotezy są wyeliminowane.

Eliminacji hipotezy dokonuje się przez zastosowanie reguły eliminującej założenia. Budując dowody przestrzegamy zasady, że eliminować można tylko tą hipotezę, która jest ostatnią hipotezą w ciągu (dowodzie).

Zazwyczaj zbiór reguł dowodzenia zawiera po dwie reguły dla każdego operatora rozpatrywanego języka (jedna reguła wprowadzająca ten operator i druga eliminująca go).

Przykład 2.7 (Klasyczna logika pierwszego rzędu) *Naturalna dedukcja dla logiki klasycznej pierwszego rzędu jest wyznaczona zbiorem reguł przedstawionym w tabeli 2.1.*

Stosowanie reguł, które nie zawierają prostokątów jest bardzo proste. Omówimy stosowanie niektórych reguł zawierających prostokąty. Na przykład regułę $\rightarrow$ *i interpretujemy następująco : jeśli z formuły* A *(hipotezy) potrafimy wywnioskować* B*, to mamy zarazem dowód formuły* $A \rightarrow B$*. Zatem jeśli założyliśmy wcześniej jako hipotezę formułę* A *i następnie stosując reguły wnioskowania wyprowadziliśmy formułę* B*, to możemy przez zastosowanie reguły* $\rightarrow$ *i pozbyć się założenia* A *zamykając prostokąt i dopisując do dowodu formułę* $A \rightarrow B$*.*

Reguła $\vee$*e mówi, że jeśli mamy dowód formuły* $A \vee B$*, oraz wychodząc zarówno z założenia* A *jak i* B *potrafimy wywnioskować* C*, to tym samym mamy dowód formuły* C*.*

Reguła $\forall$*i mówi, że aby dowieść formułę* $\forall x A(x)$ *wystarczy dla zmiennej* y*, która nie wystąpiła dotychczas w dowodzie wywnioskować* $A(y)$*. Natomiast reguła* $\exists$*e mówi, że jeśli mamy dowód formuły* $\exists x A(x)$ *i zakładając dla pewnego* y *formułę* $A(y)$ *potrafimy wywnioskować* B *to udowodniliśmy formułę* B*.*

Powyższe nieformalne interpretacje reguł uzasadniają nazwę "naturalna dedukcja". Przedstawiony sposób wnioskowania jest bardzo bliski rozumowaniu często spotykanemu w matematyce. Na przykład często schemat dowodzenia twierdzeń mówiących o pewnych własnościach wszystkich elementów pewnego zbioru Z *wygląda następująco : "Weźmy dowolne* z *ze zbioru* Z*. Wówczas ... ". Zatem wprowadzając pewną zmienną* z *pokazujemy prawdziwość tezy* $A(z)$ *i jest to dowód tezy* $\forall x \in Z A(x)$*. Ten "schemat postępowania" jest sformalizowany przez regułę* $\forall$*i.*

Dla zilustrowania definicji dowodu i przedstawienia sposobu stosowania reguł przedstawmy dowód formuły $\neg B \rightarrow (B \rightarrow A)$*.*

1	$\neg B$	*założenie*
2	B	*założenie*
3	$\perp$	$\neg$*e 1,2*
4	A	$\perp$ *e 3*
5	$B \rightarrow A$	$\rightarrow$ *i 2–4*
6	$\neg B \rightarrow (B \rightarrow A)$	$\rightarrow$ *i 1–5*

Spójrzmy na powyższy dowód od końca, tzn. od formuły dowodzonej (6). Głównym symbolem logicznym tej formuły jest operator $\rightarrow$*, stąd można się domyślić, że ostatnią użytą regułą będzie* $\rightarrow$ *i. Zatem przyjmując za hipotezę* $\neg B$ *(1) próbujemy dowieść* $B \rightarrow A$ *(5). Znowu dowodzimy formułę, której głównym symbolem logicznym jest* $\rightarrow$*, a więc możemy jeszcze raz wykonać analogiczny krok. Teraz z założenia* B *(2) musimy dowieść* A *(4). Ale z (1) i (2) według reguły* $\neg$*e otrzymujemy* $\perp$ *(3), co z kolei stosownie do reguły* $\perp$ *e daje nam (4).*

$$\frac{A \quad B}{A \wedge B} \wedge i \qquad\qquad \frac{A \wedge B}{A} \wedge e_1 \quad \frac{A \wedge B}{B} \wedge e_2$$

$$\frac{A}{A \vee B} \vee i_1 \quad \frac{B}{A \vee B} \vee i_2 \qquad\qquad \frac{A \vee B \quad \boxed{\begin{array}{c} A \\ \vdots \\ C \end{array}} \quad \boxed{\begin{array}{c} B \\ \vdots \\ C \end{array}}}{C} \vee e$$

$$\frac{\boxed{\begin{array}{c} A \\ \vdots \\ B \end{array}}}{A \rightarrow B} \rightarrow i \qquad\qquad \frac{A \quad A \rightarrow B}{B} \rightarrow e$$

$$\frac{\boxed{\begin{array}{c} A \\ \vdots \\ \bot \end{array}}}{\neg A} \neg i \qquad\qquad \frac{A \quad \neg A}{\bot} \neg e$$

$$\frac{\bot}{A} \bot e$$

$$\frac{\boxed{\begin{array}{ll} y & \\ & \vdots \\ & A(y) \end{array}}}{\forall x A(x)} \forall i \qquad\qquad \frac{\forall x A(x)}{A(t)} \forall e$$

$$\frac{A(t)}{\exists x A(x)} \exists i \qquad\qquad \frac{\exists x A(x) \quad \boxed{\begin{array}{ll} y & A(y) \\ & \vdots \\ & B \end{array}}}{B} \exists e$$

$$\frac{\neg\neg A}{A} \neg\neg$$

W regułach $\forall i$ i $\exists e$ zmienna y może występować tylko wewnątrz prostokąta.

Tabela 2.1: Reguły naturalnej dedukcji dla logiki klasycznej pierwszego rzędu

2.3.3 Reprezentacja sekwentowa

Rachunek sekwentowy jest bardzo zbliżony do rachunku naturalnej dedukcji. Podstawową różnicą między tymi reprezentacjami jest to, że w rachunku sekwentowym nie operujemy formułami, ale całymi sekwentami przez co przesłanki i założenia są wyraźnie zaznaczone w każdym punkcie dowodu. W ten sposób omijamy skomplikowaną procedurę eliminowania hipotez w naturalnej dedukcji.

Spośród wszystkich sekwentów wyróżniamy *sekwenty bazowe* zwane też *aksjomatami* jako te postaci $A \vdash A$. Często pojęcie aksjomatu traktuje się nieco szerzej jako każdy sekwent postaci $\Gamma \vdash \Delta$ gdzie $\Gamma \cap \Delta \neq \emptyset$.

Definicja 2.15 Regułą wyprowadzania (wnioskowania) *nazywamy każde wyrażenie postaci*

$$\frac{S_1 \ldots S_n}{S},$$

gdzie $S_1, \ldots, S_n, S$ są sekwentami. Sekwent S nazywamy sekwentem dolnym tej reguły, natomiast pozostałe sekwentami górnymi.

Przez drzewo sekwentów rozumiemy dowolne drzewo, którego wierzchołki są etykietowane sekwentami. Korzeń drzewa jest uznawany za jego "dół", a liście za "górę". W tym kontekscie rozumiemy pojęcia występowania sekwentu "nad" lub "pod" innym sekwentem.

Definicja 2.16 Dowodem *w reprezentacji sekwentowej nazywamy drzewo sekwentów spełniające następujące warunki :*

- *liście drzewa to aksjomaty,*
- *każdy sekwent drzewa jest górnym sekwentem reguły wyprowadzania, której dolny sekwent występuje w drzewie bezpośrednio pod tym sekwentem.*

Sekwent, który jest korzeniem drzewa nazywamy **sekwentem dowodzonym**, *a samo drzewo nazywane jest* **dowodem tego sekwentu**. **Dowodem formuły** A *nazywamy dowód sekwentu* $\vdash A$ *i mówimy wówczas, że A jest* **twierdzeniem**.

Przykład 2.8 (Klasyczna logika pierwszego rzędu) *Naturalna dedukcja reprezentowana w sposób sekwentowy dla logiki klasycznej pierwszego rzędu jest wyznaczona przez zbiór reguł wyprowadzania przedstawiony w tabeli 2.2*

Dowód formuły $\neg B \to (B \to A)$, który przedstawiliśmy jako przykład dowodu w reprezentacji naturalnej dedukcji przybierze w reprezentacji sekwentowej następującą postać :

$$\dfrac{\dfrac{\dfrac{\dfrac{\neg B \vdash \neg B \quad B \vdash B}{\neg B, B \vdash \bot}\,\neg e}{\neg B, B \vdash A}\,\bot\, e}{\neg B \vdash B \to A}\,\to i}{\vdash \neg B \to (B \to A)}\,\to i$$

W bardziej skomplikowanych dowodach wyraźniej widać strukturę drzewiastą. Tutaj mamy do czynienia z dowodem liniowym.

2.3.4 Gentzenowska reprezentacja sekwentowa

Najczęściej stosowaną metodą w automatycznych systemach dowodzenia jest metoda redukcyjna (ang. backword proof). Dowodząc dany sekwent sprowadzamy go przy pomocy stosownych reguł do pewnych łatwiejszych do udowodnienia sekwentów. Sekwent, który chcemy udowodnić nazywamy **celem**, a sekwenty powstałe w wyniku kolejnych redukcji **podcelami**.

Zatem spośród reguł wyprowadzania stosowanych w naturalnej dedukcji musimy wybrać (i ewentualnie uzupełnić) zbiór reguł, które będą produkować prostsze podcele. Nasuwa się pytanie co to znaczy, że jeden sekwent jest łatwiejszy do udowodnienia niż inny. Dobrym kryterium oceny czy dana reguła produkuje z danego celu prostsze podcele jest własność podformuły.

$$\frac{\Gamma \vdash A \quad \Delta \vdash B}{\Gamma, \Delta \vdash A \wedge B} \wedge i \qquad \frac{\Gamma \vdash A \wedge B}{\Gamma \vdash A} \wedge e_1 \quad \frac{\Gamma \vdash A \wedge B}{\Gamma \vdash B} \wedge e_2$$

$$\frac{\Gamma \vdash A}{\Gamma \vdash A \vee B} \vee i_1 \quad \frac{\Gamma \vdash B}{\Gamma \vdash A \vee B} \vee i_2 \qquad \frac{\Gamma \vdash A \vee B \quad \Delta, A \vdash C \quad \Theta, B \vdash C}{\Gamma, \Delta, \Theta \vdash C} \vee e$$

$$\frac{\Gamma, A \vdash B}{\Gamma \vdash A \rightarrow B} \rightarrow i \qquad \frac{\Gamma \vdash A \quad \Delta \vdash A \rightarrow B}{\Gamma, \Delta \vdash B} \rightarrow e$$

$$\frac{\Gamma, A \vdash \perp}{\Gamma \vdash \neg A} \neg i \qquad \frac{\Gamma \vdash A \quad \Delta \vdash A \rightarrow B}{\Gamma, \Delta \vdash \perp} \neg e$$

$$\frac{\Gamma \vdash \perp}{\Gamma \vdash A} \perp e$$

$$\frac{\Gamma \vdash A(y)}{\Gamma \vdash \forall x A(x)} \forall i \qquad \frac{\Gamma \vdash \forall x A(x)}{\Gamma \vdash A(t)} \forall e$$

$$\frac{\Gamma \vdash A(t)}{\Gamma \vdash \exists x A(x)} \exists i \qquad \frac{\Gamma \vdash \exists x A(x) \quad \Delta, A(y) \vdash B}{\Gamma, \Delta \vdash B} \exists e$$

$$\frac{\Gamma \vdash \neg\neg A}{\Gamma \vdash A} \neg\neg$$

W regułach $\forall i$ oraz $\exists e$ zmienna y nie występuje
w sposób wolny w Γ, $A(x)$ oraz B.

Tabela 2.2: Reguły rachunku sekwentowego dla logiki klasycznej pierwszego rzędu

Definicja 2.17 *Relację podformuły na zbiorze wszystkich formuł języka predykatowego opisują następujące warunki :*

1. *Formuła $\perp$ jest podformułą każdej formuły*
2. *Formuła A jest podformułą formuł A i $\neg A$*
3. *Formuły A i B są podformułami formuł $A \vee B$, $A \wedge B$ oraz $A \rightarrow B$*
4. *Dla dowolnego termu t formuła $A(t)$ jest podformułą formuł $\forall x A(x)$ i $\exists x A(x)$.*

Definicja 2.18 *Reguła wyprowadzania* $\frac{\Gamma \vdash A}{\Gamma' \vdash A'}$ *spełnia* **własność podformuły** *wtedy i tylko wtedy, gdy każda formuła ze zbioru $\Gamma \cup \{A\}$ jest podformułą pewnej formuły ze zbioru $\Gamma' \cup \{A'\}$.*

Przykład 2.9 (Logika klasyczna) *Łatwo zauważyć, że w systemie sekwentowym naturalnej dedukcji dla klasycznej logiki pierwszego rzędu żadna z reguł eliminacji nie spełnia własności podformuły, natomiast wszystkie reguły wprowadzające operatory posiadają tę własność. Reguła $\neg\neg$ również nie ma własności podformuły.*

Zbiór reguł rachunku Gentzena dla logiki klasycznej pierwszego rzędu, oraz regułę obcinania przedstawia tabela 2.3.

Z punktu widzenia automatyzacji dowodzenia system Gentzena jest zdecydowanie najlepszym spośród przedstawionych systemów. Dowody w systemie Hilberta są żmudne i mało przejrzyste. Naturalna dedukcja jest bardziej czytelna, ale budowanie dowodów utrudnia brak własności podformuły w wielu regułach (reguły eliminacji). W systemie Gentzena każda reguła ma własność podformuły, co jest bardzo istotne w automatycznym dowodzeniu.

Reguły strukturalne

$$\frac{}{\Gamma, A \vdash A, \Delta}(inkluzja) \qquad \frac{\Gamma \vdash \Theta, C \quad C, \Delta \vdash \Lambda}{\Gamma, \Delta \vdash \Theta, \Lambda}\text{CUT}$$

$$\frac{\Gamma \vdash \Delta}{\Gamma, A \vdash \Delta}\text{MONO}_l \qquad \frac{\Gamma \vdash \Delta}{\Gamma \vdash A, \Delta}\text{MONO}_r$$

Reguły logiczne

$$\frac{\Gamma, A \vdash \Delta}{\Gamma, A \wedge B \vdash \Delta} \wedge l_1 \frac{\Gamma, B \vdash \Delta}{\Gamma, A \wedge B \vdash \Delta} \wedge l_2 \qquad \frac{\Gamma \vdash A, \Delta \quad \Gamma \vdash B, \Delta}{\Gamma \vdash A \wedge B, \Delta} \wedge r$$

$$\frac{\Gamma, A \vdash \Delta \quad \Gamma, B \vdash \Delta}{\Gamma, A \vee B \vdash \Delta} \vee l \qquad \frac{\Gamma \vdash A, \Delta}{\Gamma \vdash A \vee B, \Delta} \vee r_1 \frac{\Gamma \vdash B, \Delta}{\Gamma \vdash A \vee B, \Delta} \vee r_2$$

$$\frac{\Gamma \vdash A, \Delta \quad \Gamma, B \vdash \Delta}{\Gamma, A \rightarrow B \vdash \Delta} \rightarrow l \qquad \frac{\Gamma, A \vdash B, \Delta}{\Gamma \vdash A \rightarrow B, \Delta} \rightarrow r$$

$$\frac{\Gamma \vdash A, \Delta}{\Gamma, \neg A \vdash \Delta}\neg l \qquad \frac{\Gamma, A \vdash \Delta}{\Gamma \vdash \neg A, \Delta}\neg r$$

$$\frac{\Gamma, A(t) \vdash \Delta}{\Gamma, \forall x A(x) \vdash \Delta}\forall l \qquad \frac{\Gamma \vdash A(y), \Delta}{\Gamma \vdash \forall x A(x), \Delta}\forall r$$

$$\frac{\Gamma, A(y) \vdash \Delta}{\Gamma, \exists x A(x) \vdash \Delta}\exists l \qquad \frac{\Gamma \vdash A(t), \Delta}{\Gamma \vdash \exists x A(x), \Delta}\exists r$$

W regułach $\forall r$ oraz $\exists l$ zmienna y nie występuje
w sposób wolny w Γ, Δ oraz $A(x)$.

Tabela 2.3: Reguły rachunku Gentzena dla logiki klasycznej pierwszego rzędu

2.4 Zgodność i zupełność

Pokazaliśmy, że systemy syntaktycznego wyprowadzania można budować na wiele różnych sposobów. Można tworzyć systemy formalne, które nie mają nic wspólnego ze znanymi dotychczas gałęziami logiki i matematyki. Jednakże najczęściej chcemy budować takie systemy, które ściśle odzwierciedlają znane z logiki zagadnienia. Chcemy więc, aby każda formuła, dla której potrafimy zbudować dowód syntaktyczny była istotnie prawdziwa w języku dla którego dany system syntaktyczny budowaliśmy. Z drugiej strony najlepszym systemem syntaktycznym dla danego języka będzie ten, w którym potrafimy dowieść każdą formułę, która jest prawdziwa. Stąd poniższa definicja.

Definicja 2.19 *Niech L będzie dowolnym językiem. Niech $\vdash$ będzie relacją syntaktycznej konsekwencji, oraz $\models$ relacją konsekwencji logicznej (w języku L). Relacja $\vdash$ jest* **zgodna z** $\models$ *wtedy i tylko wtedy, gdy dla dowolnego zbioru formuł Γ, oraz formuły A*

$$\Gamma \vdash A \textit{ pociąga za sobą } \Gamma \models A$$

Relacja $\vdash$ jest **zupełna względem** $\models$ *wtedy i tylko wtedy, gdy*

$$\Gamma \models A \textit{ pociąga za sobą } \Gamma \vdash A$$

Od każdej relacji syntaktycznej konsekwencji żądamy, aby była zgodna z relacją konsekwencji logicznej. Nie zawsze natomiast udaje się znaleźć system dedukcji, który jest zupełny w stosunku do rozważanego języka.

Dowodzenie zgodności jest zazwyczaj dużo łatwiejsze niż dowodzenie zupełności. Aby pokazać, że dany system syntaktyczny jest zgodny z relacją logicznej konsekwencji wystarczy pokazać, że wszystkie aksjomaty tego systemu są prawdziwe oraz, że każda reguła wyprowadzania zachowuje logiczną konsekwencję tzn. jeśli jej przesłanki są prawdziwe, to prawdziwa jest również konkluzja. W przypadku systemów Hilberta dowód jest indukcyjny, ze względu na długość dowodu. Dowodzenie zupełności jest zdecydowanie trudniejsze. Często takie dowody przeprowadza się przez negację tzn. pokazujemy, że jeśli nie istnieje dowód danej formuły, to potrafimy znaleźć dla niej "przeciwmodel" (taką interpretację, w której formuła ta nie jest spełniona).

Każdy aksjomat i każda reguła systemu redukcji ma istotny wpływ na jego zgodność i zupełność, a zatem własności te trzeba dowodzić osobno dla każdego konkretnego systemu. Aby zilustrować dowód tych własności przedstawimy go dla logiki klasycznej pierwszego rzędu, a dokładniej dla jej reprezentacji Gentzenowskiej najczęściej stosowanej w automatycznym dowodzeniu.

Twierdzenie 2.5 *Niech A będzie dowolną formułą logiki klasycznej pierwszego rzędu. Formuła A jest prawdziwa wtedy i tylko wtedy gdy posiada dowód w Gentzenowskiej reprezentacji sekwentowej.*

Dowód :

$\Leftarrow$ Dowód zgodności jest prostym dowodem indukcyjnym ze względu na długość dowodu.

$\Rightarrow$ Zupełność reprezentacji Gentzena jest bezpośrednim wnioskiem z lematu 2.6.

Lemat 2.6 mówi o zupełności w nieco ogólniejszym ujęciu, bowiem mówi o prawdziwości całych sekwentów, a nie tylko formuł. W związku z tym musimy najpierw powiedzieć co to znaczy, że sekwent jest prawdziwy, czy też spełniony w danej interpretacji.

Definicja 2.20 *Niech $\underline{M} = (M, D, F, G)$ będzie systemem waluacji dla logiki klasycznej pierwszego rzędu.*

Sekwent $\Gamma \vdash \Delta$ jest **spełniony w interpretacji** v_a *(w systemie waluacji $\underline{M}$) wtedy i tylko wtedy, gdy istnieje formuła w Γ, która nie jest spełniona w interpretacji v_a, lub istnieje formuła w Δ, która jest spełniona w v_a.*

Sekwent jest **prawdziwy** *(w systemie waluacji $\underline{M}$) gdy jest spełniony w dowolnej interpretacji v_a dla systemu $\underline{M}$.*

Lemat 2.6 *Niech S będzie sekwentem w Gentzenowskiej reprezentacji logiki klasycznej pierwszego rzędu. Wówczas albo istnieje dowód sekwentu S, w którym nie zastosowano reguły obcinania, albo istnieje interpretacja, w której S nie jest spełniony (a stąd S nie jest prawdziwy).*

Dowód : Dla danego sekwentu S zbudujemy drzewo (możliwe, że nieskończone) zwane drzewem redukcji sekwentu S, na podstawie którego można zbudować dowód sekwentu S, bądź wskazać interpretację, w której sekwent ten nie będzie spełniony. Metoda ta nosi nazwę metody Schütte'a (od nazwiska matematyka, który ją przedstawił jako pierwszy).

Będziemy posługiwać się pojęciem drzewa i innymi związanymi z nim pojęciami takimi jak : liście, wierzchołki, gałęzie itp. Niektóre używane określenia odnoszą się do reprezentacji graficznej drzewa. Na przykład przez "najwyższe sekwenty drzewa" rozumiemy liście drzewa itd.

Drzewo redukcji sekwentu S to drzewo, którego wierzchołkami są sekwenty i które budujemy zgodnie z poniższym opisem indukcyjnym. Poszczególne kroki dotyczą kolejnych reguł wywodu. Będziemy wtedy mówili, że do pewnych formuł stosujemy regułę (lub redukcję). Będziemy też stawiać założenia typu "niech A będzie formułą, do której nie stosowano w poprzednich krokach żadnej redukcji".

Krok 0: Piszemy S na samym dole drzewa (jako korzeń).

Krok k ($k > 0$): Jeśli wszystkie najwyższe sekwenty drzewa posiadają formułę wspólną dla poprzednika i następnika, to kończymy budowanie drzewa. W przeciwnym przypadku wykonyjemy czynności zależne od $k \ mod\ 13$. Dla każdego symbolu logicznego mamy po dwa punkty dotyczące go, a pozostały jeden punkt odpowiedzialny jest za kończenie gałęzi, jeśli najwyższy sekwent jest aksjomatem.

Wszystkie zmienne wolne, które pojawiły się w kroku k lub poprzednich nazywamy "dostępnymi w kroku k". Jeśli nie ma takiej to bierzemy dowolną zmienną, nie występującą dotychczas w żadnym sekwencie i mówimy o niej, że jest dostępna. Poszczególne kroki definiujemy jak następuje :

1. $k \equiv 1$ (mod 13). Niech $\Pi \vdash \Lambda$ będzie dowolnym najwyższym sekwentem drzewa zdefiniowanego w kroku $k-1$. Niech $\neg A_1, \ldots, \neg A_n$ będą wszystkimi formułami w Π, których głównym symbolem logicznym jest $\neg$, i do których nie stosowano w poprzednich krokach żadnej redukcji. Piszemy
$$\Pi \vdash \Lambda, A_1, \ldots, A_n$$
nad sekwentem $\Pi \vdash \Lambda$. Mówimy, że lewa reguła negacji została zastosowana do formuł $\neg A_1, \ldots, \neg A_n$.

2. $k \equiv 2$. Niech $\neg A_1, \ldots, \neg A_n$ będą wszystkimi formułami w Λ których głównym symbolem logicznym jest $\neg$, i do których nie stosowano w poprzednich krokach żadnej redukcji. Piszemy
$$A_1, \ldots, A_n, \Pi \vdash \Lambda$$
nad sekwentem $\Pi \vdash \Lambda$. Mówimy, że prawa reguła negacji została zastosowana do formuł $\neg A_1, \ldots, \neg A_n$.

3. $k \equiv 3$. Niech $A_1 \wedge B_1, \ldots, A_n \wedge B_n$ będą wszystkimi formułami w Π, których głównym symbolem logicznym jest $\wedge$, i do których nie stosowano w poprzednich krokach żadnej redukcji. Piszemy
$$A_1, B_1, \ldots, A_n, B_n, \Pi \vdash \Lambda$$
nad sekwentem $\Pi \vdash \Lambda$. Mówimy, że lewa reguła koniunkcji została zastosowana do formuł $A_1 \wedge B_1, \ldots, A_n \wedge B_n$.

4. $k \equiv 4$. Niech $A_1 \wedge B_1, \ldots, A_n \wedge B_n$ będą wszystkimi formułami w Λ których głównym symbolem logicznym jest $\wedge$, i do których nie stosowano w poprzednich krokach żadnej redukcji. Piszemy
$$\Pi \vdash \Lambda, C_1, \ldots, C_n$$
gdzie $C_i \in \{A_i, B_i\}$ nad sekwentem $\Pi \vdash \Lambda$. Takich możliwości jest 2^n, więc nad $\Pi \vdash \Lambda$ piszemy 2^n sekwentów. Mówimy, że prawa reguła koniunkcji została zastosowana do formuł $A_1 \wedge B_1, \ldots, A_n \wedge B_n$.

5. $k \equiv 5$. Lewa reguła alternatywy stosowana jest analogicznie do prawej reguły koniunkcji.

6. $k \equiv 6$. Prawa reguła alternatywy stosowana jest analogicznie do lewej reguły koniunkcji.

7. $k \equiv 7$. Niech $A_1 \to B_1, \ldots, A_n \to B_n$ będą wszystkimi formułami w Π, których głównym symbolem logicznym jest $\to$, i do których nie stosowano w poprzednich krokach żadnej redukcji. Piszemy

$$B_1, \ldots, B_n, \Pi \vdash \Lambda$$

oraz $\Pi \vdash \Lambda, A_i$ dla $1 \leqslant i \leqslant n$. nad sekwentem $\Pi \vdash \Lambda$. Mówimy, że lewa reguła implikacji została zastosowana do formuł $A_1 \to B_1, \ldots, A_n \to B_n$.

8. $k \equiv 8$. Niech $A_1 \to B_1, \ldots, A_n \to B_n$ będą wszystkimi formułami w Λ których głównym symbolem logicznym jest $\to$, i do których nie stosowano w poprzednich krokach żadnej redukcji. Piszemy

$$A_1, \ldots, A_n, \Pi \vdash \Lambda, B_1, \ldots, B_n$$

nad sekwentem $\Pi \vdash \Lambda$. Mówimy, że prawa reguła implikacji została zastosowana do formuł $A_1 \to B_1, \ldots, A_n \to B_n$.

9. $k \equiv 9$. Niech $\forall x_1 A_1(x_1), \ldots, \forall x_n A_n(x_n)$ będą wszystkimi formułami w Π, których głównym symbolem logicznym jest $\forall$. Dla $1 \leqslant i \leqslant n$ niech a_i będzie zmienną dostępną w tym kroku, która nie była użyta do redukcji formuły $\forall x_i A(x_i)$. Piszemy

$$A_1(a_1), \ldots, A_n(a_n), \Pi \vdash \Lambda$$

nad sekwentem $\Pi \vdash \Lambda$. Mówimy, że lewa reguła kwantyfikatora $\forall$ została zastosowana do formuł $\forall x_1 A_1(x_1), \ldots, \forall x_n A_n(x_n)$.

10. $k \equiv 10$. Niech $\forall x_1 A_1(x_1), \ldots, \forall x_n A_n(x_n)$ będą wszystkimi formułami w Λ których głównym symbolem logicznym jest $\forall$, i do których nie stosowano w poprzednich krokach żadnej redukcji. Niech $a_1, \ldots, a_n$ będą zmiennymi, które nie są dostępne w tym kroku. Piszemy

$$\Pi \vdash \Lambda, A_1(a_1), \ldots, A_n(a_n)$$

nad sekwentem $\Pi \vdash \Lambda$. Mówimy, że prawa reguła kwantyfikatora $\forall$ została zastosowana do formuł $\forall x_1 A_1(x_1), \ldots, \forall x_n A_n(x_n)$. Zmienne $a_1, \ldots, a_n$ stają się nowymi dostępnymi zmiennymi.

11. $k \equiv 11$.

    Lewą redukcję kwantyfikatora $\exists$ definiujemy analogicznie do prawej kwantyfikatora $\forall$.

12. $k \equiv 12$.

    Prawą redukcję kwantyfikatora $\exists$ definiujemy analogicznie do lewej kwantyfikatora $\forall$.

13. $k \equiv 0$. Jeśli Π i Λ mają wspólną formułę, to nie piszemy nic nad sekwentem $\Pi \vdash \Lambda$ (pozostaje on sekwentem najwyższym czyli liściem drzewa). W przeciwnym wypadku przepisujemy ten sam sekwent.

Powstały w powyższym procesie zbiór sekwentów wraz z częściowym porządkiem wyznaczonym przez pisanie "nad ..." jest drzewem redukcji sekwentu S i oznaczamy go przez $T(S)$.

Pokażemy teraz, że dla dowolnego sekwentu S albo istnieje dowód tego sekwentu nie stosujący reguły obcinania, albo istnieje interpretacja, w której S nie jest spełniony.

Jeśli drzewo $T(S)$ jest skończone, to łatwo można z niego wyprodukować dowód sekwentu S nie zawierający reguły obcinania. W przeciwnym przypadku istnieje nieskończona gałąź drzewa $T(S)$. Niech będzie to gałąź $S_0 = S, S_1, \ldots, S_n, \ldots$. Niech ponadto $S_i = \Gamma_i \vdash \Delta_i$ dla $i = 1, 2, \ldots$.

Niech $\bigcup \Gamma_i$ będzie zbiorem wszystkich formuł pojawiających się w Γ_i dla pewnego i, a $\bigcup \Delta_j$ zbiorem formuł pojawiających się w Δ_j dla pewnego j. Wskażemy interpretację, w której będą spełnione wszystkie formuły ze zbioru $\bigcup \Gamma_i$, oraz żadna ze zbioru $\bigcup \Delta_j$.

Pamiętamy, że rozważamy język $L = (P, T, V, O, Q, arg)$, gdzie $O = \{\vee, \wedge, \rightarrow, \neg, \perp\}$, $V = \{x, y, z, x_1, x_2, \ldots\}$, $Q = \{\forall, \exists\}$. Interpretację w tym języku wyznacza para (a,I) (przypisanie) spełniająca warunki definicji 1.13. Niech więc $I = V$, oraz niech a będzie funkcją określoną następująco :

- $a(x) = x$ dla dowolnego $x \in V$,
- $a(t)$ niech będzie dowolne ale ustalone dla każdego $t \in T$,
- dla dowolnego $p \in P$ jeśli $arg(p) = n$ to $a(p)$ jest funkcją taką, że dla dowolnych $a_1, \ldots, a_n \in V (= I)$
  - jeśli $p(a_1, \ldots, a_n) \in \bigcup \Gamma_i$, to $a(a_1, \ldots, a_n) = t$
  - jeśli $p(a_1, \ldots, a_n) \in \bigcup \Delta_j$, to $a(a_1, \ldots, a_n) = f$

Taka definicja funkcji a jest uzasadniona, ponieważ zbiory $\bigcup \Gamma_i$, $\bigcup \Delta_j$ nie mają wspólnej formuły atomowej. Pokażemy, że ta interpretacja spełnia wszystkie formuły ze zbioru $\bigcup \Gamma_i$ i nie spełnia żadnej ze zbioru $\bigcup \Delta_j$.

Dowód tego faktu jest indukcyjny ze względu na ilość symboli logicznych w formule. Z definicji funkcji a wynika, że spełnia ona żądany warunek. Przypuśćmy, że spełnia też ten warunek dla formuł o liczbie symboli logicznych mniejszej bądź równej n. Niech formuła A ma $n + 1$ symboli logicznych. Musimy rozważyć dwa przypadki :

1. $A \in \bigcup \Gamma_i$

   Jeśli A jest postaci $\neg B$, to występuje ona w pewnym sekwencie S_i (wybranej, nieskończonej gałęzi S_0) postaci
   $$\Gamma_i^1, \neg B, \Gamma_i^2 \vdash \Delta_i$$
   takim, że sekwent S_{i+1} ma postać
   $$\Gamma_{i+1} \vdash \Delta_{i+1}^1, B, \Delta_{i+1}^2$$
   Zatem $B \in \bigcup \Delta_j$, oraz B ma n symboli logicznych. Z założenia indukcyjnego a nie spełnia formuły B, a zatem spełnia formułę A.

   Jeśli $A \equiv \forall x B(x)$, to występuje ona w pewnym sekwencie S_i postaci
   $$\Gamma_i^1, \forall x B(x), \Gamma_i^2 \vdash \Delta_i$$
   Aby udowodnić, że formuła A jast spełniona w interpretacja a musimy wykazać, że formuła $B(x)$ jest spełniona w każdej interpretacji różniącej się od a co najwyżej w punkcie x. Ale ponieważ $a(x) = x$ dla każdego $x \in V$, więc wystarczy pokazać, że dla dowolnej zmiennej y formuła $B(y)$ jest spełniona przez przypisanie a. Ale z konstrukcji drzewa T(S) wynika, że wszystkie takie $B(y) \in \bigcup \Gamma_i$. Ponieważ wszystkie one mają n symboli logicznych, więc z założenia indukcyjnego są spełnione przez interpretację (a,I).

   Dla pozostałych operatorów dowód wygląda analogicznie.

2. $A \in \bigcup \Delta_j$

   Jeśli $A \equiv \neg B$, to występuje ona w pewnym sekwencie S_i postaci
   $$\Gamma_i \neg B \vdash \Delta_i^1, \neg B, \Delta_i^2$$
   takim, że sekwent S_{i+1} ma postać
   $$\Gamma_{i+1}^1, B, \Gamma_{i+1}^2 \vdash \Delta_{i+1}$$
   Zatem $B \in \bigcup \Gamma_i$, oraz B ma n symboli logicznych. Z założenia indukcyjnego a spełnia formułę B, a zatem nie spełnia formuły A.

Jeśli $A \equiv \forall x B(x)$, to występuje ona w pewnym sekwencie S_i postaci

$$\Gamma_i \vdash \Delta_i^1, \forall x B(x), \Delta_i^2$$

takim, że sekwent $i+1$ ma postać

$$\Gamma_{i+1} \vdash \Delta_{i+1}^1, B(a_0), \Delta_{i+1}^2$$

Stąd $B(a_0) \in \bigcup \Delta_j$, a ponieważ ma n symboli logicznych, więc nie jest spełniona przez (a, I). Stąd nie jest też spełniona formuła A.

Dla pozostałych operatorów dowód wygląda analogicznie.

Zatem teza została udowodniona.

Zauważmy, że przedstawiony dowód nie podaje algorytmu, który dla dowolnego sekwentu dałby nam odpowiedź, czy istnieje dowód tego sekwentu czy też nie istnieje. Jeśli dany sekwent jest prawdziwy, to algorytm ukazany w dowodzie w skończonej ilości kroków znajdzie jego dowód, natomiast w przeciwnym przypadku budowane jest drzewo redukcji, które nie jest skończone, a zatem algorytm nie daje odpowiedzi. Innymi słowy problem posiadania dowodu w logice klasycznej pierwszego rzędu jest rekurencyjny, ale nie jest rekurencyjnie przeliczalny.

Z powyższego lematu wynika wprost twierdzenie o eliminacji cięć (jedno z istotniejszych dla logiki klasycznej pierwszego rzędu).

Twierdzenie 2.7 (Gentzen) *W logice klasycznej pierwszego rzędu jeśli sekwent posiada dowód, to posiada dowód, w którym nie stosuje się reguły obcinania.*

Wnioskowanie tego faktu z twierdzenia o zupełności nie wymaga żadnego nakładu pracy, ale nie jest konstruktywne. Twierdzenie to posiada jednak dowód konstruktywny (przedstawiono go dosyć szczegółowo w [2]). Pokazuje się tam wprost, że dowolny dowód można przekształcić w dowód jemu równoważny i nie zawierający reguły obcinania.

2.5 Równoważność opisów konsekwencji syntaktycznej

Przedstawiliśmy różne systemy syntaktyczne dla języka logiki klasycznej pierwszego rzędu, a zatem dla każdego z nich powinniśmy wykazać zgodność i zupełność. Zamiast tego pokażemy równoważność wszystkich tych systemów co wobec przedstawionego wcześniej dowodu zgodności i zupełności systemu Gentzena daje nam podobny dowód dla pozostałych.

2.5.1 Równoważność systemów Hilberta i naturalnej dedukcji

Twierdzenie 2.8 *Sekwent $\Gamma \vdash A$ posiada dowód w systemie Hilberta dla logiki klasycznej pierwszego rzędu wtedy i tylko wtedy, gdy posiada dowód w systemie naturalnej dedukcji dla tej logiki.*

Dowód :
$\Rightarrow$ Aby pokazać, że każdy dowód w systemie Hilberta posiada odpowiadający mu dowód w naturalnej dedukcji wystarczy pokazać, że wszystkie aksjomaty dadzą się udowodnić z pustego zbioru przesłanek oraz że reguły wyprowadzania systemu Hilberta można uzyskać przez stosowne złożenie reguł naturalnej dedukcji. Poza tym należy wykazać, że definicje operatorów $\vee$, $\wedge$, $\perp$ oraz kwantyfikatora $\exists$ są równoważnościami w naturalnej dedukcji.

Dowód aksjomatu $A1$ może wyglądać następująco :

1	A	założenie
2	B	założenie
3	$A \wedge B$	$\wedge i$ 1,2
4	A	$\wedge e_1$ 3
5	$B \rightarrow A$	$\rightarrow i$ 2–4
6	$A \rightarrow (B \rightarrow A)$	$\rightarrow i$ 1–5

Dowód aksjomatu $A2$ łatwo można uzyskać stosując jedynie reguły $\rightarrow i$ oraz $\rightarrow e$. Aksjomat $A3$ można udowodnić następująco :

1	$\neg A \rightarrow \neg B$	założenie
2	B	założenie
3	$\neg A$	założenie
4	$\neg B$	$\rightarrow e$ 1,3
5	$\bot$	$\neg e$ 2,4
6	$\neg\neg A$	$\neg i$ 3-5
7	A	$\neg\neg$ 6
8	$B \rightarrow A$	$\rightarrow i$ 2-7
9	$(\neg A \rightarrow \neg B) \rightarrow (B \rightarrow A)$	$\rightarrow i$ 1-8

Dowód aksjomatu $A5$ może być następujący :

1	$\forall x A(x)$	założenie
2	$A(t)$	$\forall e$ 1
3	$\forall x A(x) \rightarrow A(t)$	$\rightarrow i$ 1-2

Jeżeli przyjmiemy, że zamiast $\forall x A$ będziemy pisać $\forall x A(x)$ (co nie zmienia kontekstu wyrażenia), to aksjomat $A4$ jest przypadkiem szczególnym aksjomatu $A5$ ($t := x$, zmienna też jest termem), a zatem mamy już jego dowód.
Zgodnie z powyższym ustaleniem aksjomat $A6$ możemy przepisać w postaci :

$$\forall x(A(x) \rightarrow B(x)) \rightarrow (A(x) \rightarrow \forall x B(x))$$

i wtedy dowód tego aksjomatu ma postać :

1		$\forall x(A(x) \rightarrow B(x))$	założenie
2		$A(x)$	założenie
3	y	$A(y) \rightarrow B(y)$	$\forall e$ 1
4		$A(y)$	$x := y$ 2
5		$B(y)$	$\rightarrow e$ 3,4
6		$\forall x B(x)$	$\forall i$ 3-5
7		$A(x) \rightarrow \forall x B(x)$	$\rightarrow i$ 2-6
8		$\forall x(A(x) \rightarrow B(x)) \rightarrow (A(x) \rightarrow \forall x B(x))$	$\rightarrow i$ 2-7

Zatem wszystkie aksjomaty Hilberta są dowodliwe w naturalnej dedukcji. Jeśli chodzi o reguły, to wystarczy zauważyć, że *modus ponens* to dokładnie reguła $\rightarrow e$, natomiast reguła uogólnienia jest po prostu inną formą zapisu reguły $\forall i$.

Pozostaje jeszcze wykazać poprawność definicji pozostałych operatorów i kwantyfikatora. Przypomnijmy, że w sysemie Hilberta były one definiowane następująco :

$$\begin{aligned} A \wedge B &\equiv \neg(A \rightarrow \neg B) \\ A \vee B &\equiv \neg A \rightarrow B \\ \bot &\equiv \neg(A \rightarrow A) \\ \exists x A(x) &\equiv \neg\forall x \neg A(x) \end{aligned}$$

Musimy zatem dla każdej z powyższych równoważności pokazać, że przyjmując za przesłankę formułę stojącą z lewej strony można przy pomocy środków dowodzenia stosowanych w naturalnej dedukcji osiągnąć jako konkluzję formułę stojącą z prawej strony i odwrotnie. Na przykład poprawność pierwszej równoważności uzasadniają poniższe dwa dowody :

1	$A \wedge B$	
2	$A \to \neg B$	założenie
3	A	$\wedge e_1$ 1
4	B	$\wedge e_2$ 1
5	$\neg B$	$\to e$ 2,3
6	$\perp$	$\neg e$ 4,5
7	$\neg(A \to \neg B)$	$\neg i$ 2-6

1	$\neg(A \to \neg B)$	
2	$\neg(A \wedge B)$	założenie
3	A	założenie
4	B	założenie
5	$A \wedge B$	$\wedge i$ 3,4
6	$\perp$	$\neg e$ 2,5
7	$\neg B$	$\neg i$ 4–6
8	$A \to \neg B$	$\to i$ 3–7
9	$\perp$	$\neg e$ 1,8
10	$\neg\neg(A \wedge B)$	$\neg i$ 2–9
11	$A \wedge B$	$\neg\neg$ 10

Podobnie uzasadniamy definicję $\vee$:

1	$A \wedge B$			
2	A	założenie	B	założenie
3	$\neg A$	założenie	$\neg A$	założenie
4	$\perp$	$\neg e$ 2,3	B	2
5	B	$\perp e$ 4	$\neg A \to B$	$\to i$ 3–4
6	$\neg A \to B$	$\to i$ 3–5		
7	$\neg A \to B$	$\vee e$ 1,2–6		

1	$\neg A \to B$	
2	$\neg(A \vee B)$	założenie
3	$\neg A$	założenie
4	B	$\to e$ 1,3
5	$A \vee B$	$\vee i_2$ 4
6	$\perp$	$\neg e$ 2,5
7	$\neg\neg A$	$\neg i$ 3–6
8	A	$\neg\neg$ 7
9	$A \vee B$	$\vee i_1$ 8
10	$\perp$	$\neg e$ 1,9
11	$\neg\neg(A \vee B)$	$\neg i$ 2–10
12	$A \vee B$	$\neg\neg$ 11

Dowody uzasadniające poprawność definicji $\perp$ są trywialne.

Pozostaje zatem wykazać poprawność definicji kwantyfikatora $\exists$. Zadanie to spełniają poniższe dwa dowody.

1		$\exists x F(x)$	
2		$\forall x(\neg F(x))$	założenie
3	y	$F(y)$	założenie
4		$\neg F(y)$	$\forall e$ 2
5		$\perp$	$\neg e$ 3,4
6		$\perp$	$\exists e$ 1,3–5
7		$\neg\forall x(\neg F(x))$	$\neg i$ 2–6

1		$\neg\forall x(\neg F(x))$	
2		$\neg\exists x F(x)$	założenie
3	y	$F(y)$	założenie
4		$\exists x F(x)$	$\exists i$ 3
5		$\perp$	$\neg e$ 2,4
6		$\neg F(y)$	$\neg i$ 3–5
7		$\forall x(\neg F(x))$	$\forall i$ 3–6
8		$\perp$	$\neg e$ 1,7
9		$\neg\neg\exists x F(x)$	$\neg i$ 2–8
10		$\exists x F(x)$	$\neg\neg$ 9

$\Leftarrow$ Ponieważ w systemie naturalnej dedukcji nie ma aksjomatów, więc dowód tej części twierdzenia polega na wskazaniu dla każdej reguły odpowiadającego jej fragmentu dowodu w reprezentacji Hilberta. Każdą regułę należy przekształcić zgodnie z definicjami symboli $\vee$, $\wedge$, $\perp$, $\exists$, tak aby zawierały tylko operatory $\rightarrow$, $\neg$ i $\forall$. Budowanie dowodów tych własności jest bardzo żmudne a zbudowane dowody są bardzo nieczytelne. Wyjątkami są reguły dla operatora $\rightarrow$, bo $\rightarrow e$ to *modus ponens*, a $\rightarrow i$ odpowiada własności dedukcji, spełnionej w systemie Hilberta. Jeden z prostszych dowodów to dowód dla reguły $\neg\neg$, ale i on pokazuje jedynie jak trudne jest budowanie dowodów w systemie Hilberta. Wygląda bowiem następująco :

1.	$\neg\neg A$	założenie
2.	$\neg\neg A \rightarrow (\neg\neg\neg\neg A \rightarrow \neg\neg A)$	A1
3.	$\neg\neg\neg\neg A \rightarrow \neg\neg A$	M.P. 1,2
4.	$(\neg\neg\neg\neg A \rightarrow \neg\neg A) \rightarrow (\neg A \rightarrow \neg\neg\neg A)$	A3
5.	$\neg A \rightarrow \neg\neg\neg A$	M.P. 3,4
6.	$(\neg A \rightarrow \neg\neg\neg A) \rightarrow (\neg\neg A \rightarrow A)$	A3
7.	$\neg\neg A \rightarrow A$	M.P. 5,6
8.	A	M.P. 1,7

Naturalna dedukcja i jej wersja sekwentowa

Przedstawiliśmy dwa systemy naturalnej dedukcji – jeden operujący na formułach i drugi sekwentowy. Obydwa systemy noszą nazwę naturalnej dedukcji, ponieważ są jedynie różnymi sposobami zapisu tych samych dowodów. W wersji sekwentowej założenia pamiętane są w każdym punkcie dowodu (są elementami zbioru stojącego po lewej stronie sekwentu). Wprowadzenie założenia A jako hipotezy, czy też jako przesłanki dowodzonego sekwentu w naturalnej dedukcji jest w wersji sekwentowej wprowadzeniem sekwentu bazowego $A \vdash A$. Równoważność tych reprezentacji jest więc oczywista.

2.5.2 Naturalna dedukcja a system Gentzena

Lemat 2.9 *Sekwent $\Gamma \vdash A, \Delta$ posiada dowód w reprezentacji Gentzena logiki klasycznej pierwszego rzędu wtedy i tylko wtedy, gdy sekwent $\Gamma \vdash A$ posiada dowód lub $\Gamma \vdash \Delta$ posiada dowód.*

Dowód : Lemat ten jest prostym wnioskiem z twierdzenia o zgodności i zupełności reprezentacji Gentzena logiki klasycznej pierwszego rzędu.

Wniosek : Jeśli w systemie Gentzena dla logiki klasycznej pierwszego rzędu (tabela 2.3 na stronie 20) zastąpimy reguły $\rightarrow l$ i $\neg l$ regułami :

$$\frac{\Gamma \vdash A \quad \Gamma, B \vdash \Delta}{\Gamma, A \rightarrow B \vdash \Delta} \rightarrow l' \qquad \frac{\Gamma \vdash A}{\Gamma, \neg A \vdash \Delta} \neg l'$$

to otrzymamy system równoważny wyjściowemu.

Twierdzenie 2.10 *Sekwent $\Gamma \vdash A$ posiada dowód w reprezentacji sekwentowej naturalnej dedukcji dla logiki klasycznej pierwszego rzędu wtedy i tylko wtedy, gdy posiada dowód w systemie Gentzena dla tej logiki.*

Dowód : Dowód w obie strony jest indukcyjny ze względu na *głębokość* dowodu tzn. długość najdłuższej gałęzi.
$\Rightarrow$ Ponieważ sekwenty bazowe w obydwóch systemach definiowane są tak samo, więc aby uzyskać pełny dowód indukcyjny wystarczy pokazać, że każda reguła reprezentacji sekwentowej naturalnej dedukcji może być stosowana w systemie Gentzena tzn. że odpowiada jej pewien fragment dowodu (lub wręcz jedna reguła) w tym systemie. Oczywiście każdej regule wprowadzającej operator w naturalnej dedukcji odpowiada prawa reguła Gentzena dla tego operatora. Każda reguła eliminacji

operatora posiada stosowny dowód wykorzystujący lewą regułę Gentzena dla tego operatora (oraz regułę CUT). Na przykład regułę $\to e$ można dowieść następująco :

$$\dfrac{\dfrac{\Gamma \vdash A \quad \Gamma, B \vdash B}{\Gamma, A \to B \vdash B} \to l \qquad \Gamma \vdash A \to B}{\Gamma \vdash B} \text{ CUT}$$

Dla pozostałych operatorów dowody wyglądają bardzo podobnie.

Musimy jeszcze wskazać fragment dowodu realizujący regułę $\neg\neg$. Wygląda on następująco :

$$\dfrac{\Gamma \vdash \neg\neg A \qquad \dfrac{\dfrac{A \vdash A}{\vdash A, \neg A} \neg r}{\neg\neg A \vdash A} \neg l}{\Gamma \vdash A} \text{ CUT}$$

$\Leftarrow$ Rozpatrujemy system Gentzena, w którym zastąpiono reguły $\to l$ oraz $\neg l$ przez $\to l'$ i $\neg l'$.

Pierwszy krok indukcyjny jest oczywisty. Załóżmy, że każdy dowód w systemie Gentzena długości $\leqslant n$ posiada odpowiadający mu dowód w naturalnej dedukcji. Weźmy dowód sekwentu $\Gamma \vdash A$ o głębokości $n+1$. Po obcięciu tego dowodu o ostatnie wyprowadzenie otrzymamy jeden lub dwa dowody o głębokości $\leqslant n$, które na mocy założenia indukcyjnego posiadają stosowne odpowiedniki w naturalnej dedukcji. Wystarczy więc pokazać, że ostatnią użytą regułę można "przetłumaczyć" na system naturalnej dedukcji.

- Jeśli jest to reguła strukturalna, to jest to oczywiste, bowiem naturalna dedukcja spełnia własności inkluzji, monotoniczności i obcinania, a reguła MONO$_r$ (jeśli użyta jako ostatnia to $\Delta = \emptyset$) jest po prostu regułą $\perp e$.
- Jeśli jest to prawa reguła operatora to (jak poprzednio $\Delta = \emptyset$) jest ona regułą wprowadzającą ten operator w naturalnej dedukcji.
- Jeśli jest to reguła lewa operatora to można wskazać odpowiadający tej regule fragment dowodu w naturalnej dedukcji (wykorzystujący regułę eliminacji tego operatora i własności strukturalne, które oczywiście są spełnione w naturalnej dedukcji). Zmiana reguł $\to l$ oraz $\neg l$ była niezbędna, aby w wyniku redukcji sekwentu $\Gamma \vdash A$ nie otrzymać sekwentu, który ma więcej niż jedną formułę w następniku. Wszystkie te dowody wykonuje się analogicznie. Oto dowody dla $\vee l$ oraz $\to l'$:

$$\dfrac{\dfrac{A \vee B \vdash A \vee B \quad \Gamma, A \vdash C \quad \Delta, B \vdash C}{A \vee B, \Gamma, \Delta \vdash C} \vee e}{\Gamma, \Delta, A \vee B \vdash C} *$$

$$\dfrac{\dfrac{\dfrac{\Gamma \vdash A}{\Gamma, A \to B \vdash A} \text{MONO}_l \quad \dfrac{A \to B \vdash A \to B}{\Gamma, A \to B \vdash A \to B} \text{MONO}_l}{\Gamma, A \to B \vdash B} \to e \qquad \dfrac{\Gamma, B \vdash C}{\Gamma, A \to B, B \vdash C} \text{MONO}_l}{\Gamma, A \to B \vdash C} \text{CUT}$$

W miejscu oznaczonym znakiem "$*$" wykorzystaliśmy fakt, że poprzednik sekwentu jest zbiorem.

Rozdział 3

Isabelle

Rozdział ten opisuje sposób dowodzenia twierdzeń, którym posługuje się pakiet Isabelle [3, 4, 5]. W pierwszych podrozdziałach przedstawione są podstawowe informacje na temat struktury pakietu, definicje niezbędne do konstruowania drzew wyprowadzeń oraz przykłady dowodów w logice klasycznej (w reprezentacji naturalnej dedukcji i Gentzena). W następnych znalazł miejsce dokładniejszy opis techniczny pakietu służący budowaniu własnych języków logiki. Opis ten jest zilustrowany przykładem systemu wnioskowania dla λ-cube. W dalszej części dla określenia systemów logicznych, w których chcemy dowodzić twierdzenia będziemy używać nazwy "logika przedmiotowa" (dla odróżnienia od meta-logiki).

W poprzednim rozdziale przedstawione zostały trzy systemy dowodzenia. Były to opisy formalne dowodów nie pokazujące algorytmów ich budowania. Istnieją dwa sposoby tworzenia dowodów : dedukcyjny (ang. forward proof) i redukcyjny (ang. backword proof).

Sposób dedukcyjny polega na budowaniu drzewa dowodu od liści do korzenia. Wychodząc od aksjomatów stosujemy reguły wyprowadzania tak, aby na koniec uzyskać udowodnione twierdzenie.

Metodą redukcyjną tworzymy dowód "od końca". Wychodząc od formuły, którą chcemy udowodnić stosujemy reguły wyprowadzania redukując tę formułę do formuły (lub kilku formuł) "łatwiejszej" do udowodnienia. Analogiczną redukcję stosuje się do każdej nowej formuły tak, aby otrzymać na koniec same aksjomaty.

Isabelle jest systemem interakcyjnym, pozwalającym na budowanie dowodów zarówno metodą dedukcji jak i redukcji [3]. Oczywiście dowód dedukcyjny musi być tworzony całkowicie przez użytkownika. Jedynie metoda redukcyjna może być automatyzowana. Dlatego też w dalszej części zajmować się będziemy tylko metodą redukcji.

Isabelle została napisana w ML-u. W związku z tym będziemy często pisać o ML-owskich typach obiektów wchodzących w jej skład i cytować fragmenty kodu w tym właśnie języku. Sposób komunikacji pakietu Isabelle z użytkownikiem jest dokładnie taki sam jak w ML-u.

3.1 Pure Isabelle

Isabelle została zaprojektowana jako drzewo systemów logicznych. Oznacza to, że definiując nową logikę można oprzeć się na innej już zaimplementowanej dziedzicząc wszystkie jej własności (stałe, reguły itp.). Korzeniem drzewa jest *Pure Isabelle*, gdzie określony jest meta-język i wprowadzone elementy wspólne dla wszystkich logik przedmiotowych.

Każda logika przedmiotowa jest implementowana jako *teoria*. Teorie mają w ML-u typ `theory`.

Do opisu pojęcia dowodu (redukcyjnego) w Isabelle używamy następujących pojęć :

- cel, podcel,
- krok dowodu, taktyka,
- stan dowodu, poziom dowodu.

Cel dowodu to formuła w logice przedmiotowej, którą chcemy dowodzić.
Dowód wykonujemy w krokach. Każdy krok (np. zastosowanie jednej reguły) wykonujemy stosując *taktykę*. Taktyki mają ML-owski typ `tactic`.
Formuły powstałe w kolejnych krokach nazywamy *podcelami.*
Poziom dowodu to ilość wykonanych kroków (zastosowanych taktyk). Na początku dowodu mamy więc poziom 0. Każdemu poziomowi dowodu odpowiada pewien *stan dowodu.* Jest to lista aktualnych podcelów. Jeśli zastosowana taktyka może produkować wiele różnych stanów dowodu, to na nowym poziomie dowodu jeden z nich staje się stanem bieżącym, a pozostałe są pamiętane jako alternatywne. Tak więc dowód w Isabelle na poziomie n to lista par :

$$[(\psi_n, \Psi_n),\ (\psi_{n-1}, \Psi_{n-1}),\ \ldots,\ (\psi_0, \Psi_0)],$$

gdzie ψ_n jest bieżącym stanem dowodu, ψ_{n-1} poprzednim, itd. aż do ψ_0, który jest stanem początkowym. Ψ_i to lista alternatywnych stanów dla poziomu i. Oczywiście Ψ_0 jest zawsze listą pustą. Jeśli okaże się, że stan ψ_n nie prowadzi do zakończenia dowodu sukcesem, to nie musimy budować całego dowodu od początku, ale możemy wrócić do poziomu n, wybrać jeden z alternatywnych stanów dowodu jako bieżący i kontynuować dowód od tego poziomu.

Każdy dowód redukcyjny w Isabelle rozpoczyna się deklaracją celu. Cel ten zostaje przetłumaczony na język meta-logiki. Wszystkie operacje wykonywane na podcelach są przetwarzaniem meta-formuł. Każda zmiana stanu dowodu dokonuje się za poprzez zastosowanie taktyki. Jeśli taktyka się powiedzie, to otrzymujemy nowy stan dowodu. W przypadku, gdy taktyka nie potrafi stworzyć nowego stanu mówimy, że taktyka zawodzi. Isabelle sygnalizyje taką sytuację wypisując na ekranie komunikat "`tactic failed`". Dowód kończy się sukcesem gdy po kolejnej redukcji nie mamy żadnego podcelu do udowodnienia. Isabelle informuje nas o tym pisząc "`No subgoals!`" w czasie drukowania bieżącego stanu dowodu.

Aby zilustrować powyższe pojęcia konkretnym dowodem musimy najpierw poznać meta-język, którym posługuje się Isabelle, oraz notację komputerową jego składowych.

Formuły w logikach przedmiotowych są pamiętane jako obiekty typu `term`. Termy w Isabelle są termami λ-rachunku z typami. Ich odpowiedniki w kodach ASCII wyglądają następująco [3] :

t `::` σ	$t :: \sigma$	związanie typu
`%`x`.`t	$\lambda x.t$	meta-abstrakcja
`%`$x_1 \ldots x_n$`.`t	$\lambda x_1 \ldots x_n.t$	
$t(u)$	$t(u)$	meta-aplikacja

Operator meta-abstrakcji nie jest zazwyczaj widoczny dla użytkownika. Wykorzystywany jest do definiowania nowych operatorów wiążących zmienne (np. kwantyfikatorów).

Każdemu termowi jest przyporządkowany jakiś typ. Typy termów meta-logiki mają ML-owski typ `typ`. Typy funkcyjne budujemy przy pomocy konstruktora `=>`.

Reguły dowodzenia logik przedmiotowych są twierdzeniami meta-logiki. Mają w ML-u typ `thm`. Konstruktory i ich odpowiedniki w kodach ASCII przedstawiają się następująco (litery greckie oznaczają meta-formuły, a,b - termy, x,y,z - zmienne) :

a `==` b	$a \equiv b$	meta-równość
ϕ `==>` ψ	$\phi \Longrightarrow \psi$	meta-implikacja
`[|` ϕ_1`;` $\ldots$`;` ϕ_n `|] ==>` ψ	$\phi_1 \Longrightarrow (\cdots \phi_n \Longrightarrow \psi \cdots)$	implikacja zagnieżdżona
`!!`xyz`.`ϕ	$\bigwedge xyz.\phi$	meta-kwantyfikowanie

Także symbole poszczególnych logik przedmiotowych muszą mieć swój zapis znakami ASCII. Te stosowane w Isabelle dla zdefiniowanych już logik wyglądają następująco :

P `&` Q	$P \,\&\, Q$	koniunkcja
P `|` Q	$P \vee Q$	alternatywa
`~` P	$\neg P$	negacja
P `-->` Q	$P \rightarrow Q$	implikacja
P `<->` Q	$P \leftrightarrow Q$	równoważność
`ALL` $xyz.P$	$\forall xyz.P$	dla każdego
`EX` $xyz.P$	$\exists xyz.P$	istnieje

W meta-logice wyróżnia się pewne zmienne poprzedzając je znakiem "?" (nazywa się je *zmiennymi schematycznymi*). Zmienne te są wykorzystywane w procesie *unifikacji formuł*. Możemy je traktować jako miejsca, w które można wstawić dowolne formuły.

Unifikacja (uzgodnienie) dwóch meta-formuł polega na podstawieniu za zmienne schematyczne takich termów, aby otrzymać formuły syntaktycznie równe.

Każda reguła, która jest w rzeczywistości schematem reguł powinna zamiast zwykłych zmiennych stosować zmienne schematyczne. Isabelle traktuje każdą definiowaną regułę jako schemat reguł. Zwalnia użytkownika z obowiązku pisania znaków "?" wykonując automatycznie odpowiednią konwersję zmiennych.

Dla zilustrowania przedstawionej notacji przyjrzyjmy się regule

$$\frac{P \quad Q}{P \wedge Q}$$

Jako meta-aksjomat będzie ona zapisana w postaci

$$\bigwedge PQ.\, P \Longrightarrow (Q \Longrightarrow P \,\&\, Q)$$

W przekładzie na znaki osiągalne z klawiatury

```
!! P Q. P ==> (Q ==> P&Q)
```

Możemy zrezygnować z pisania początkowych kwantyfikatorów na rzecz zmiennych schematycznych otrzymując z powyższego :

```
?P ==> (?Q ==> ?P & ?Q)
```

Pomijając zbędne nawiasy (bo `==>` wiąże od prawej) i stosując notację dla zagnieżdżonej implikacji otrzymujemy :

```
[| ?P; ?Q |] ==> ?P & ?Q
```

Jeśli ta formuła reprezentuje regułę dowodzenia to z powodu wspomnianej konwersji można ją pisać następująco :

```
[| P; Q |] ==> P & Q
```

Mówimy, że dany podcel jest *dowodliwy z założenia*, jeśli ma on kształt meta-implikacji, w której wśród przesłanek istnieje formuła uzgadnialna z konkluzją. Podcel jest *dowodliwy przez sprzeczność*, gdy ma kształt meta-implikacji, w której wśród przesłanek istnieją dwie formuły uzgadnialne do postaci P i $\neg P$.

3.1.1 Podstawowe taktyki

Pure Isabelle implementuje operacje, przy pomocy których możemy określać cel dowodu, wykonywać poszczególne kroki dowodu, anulować ostatni krok dowodu itp.

W poniższych opisach fragmenty dowodów są drukowane czcionką maszynową. Linie poprzedzone znakiem `>` to linie które wpisuje użytkownik w czasie sesji Isabelle. Pozostałe linie to odpowiedzi Isabelle na polecenia użytkownika. W pierwszym przykładzie dowodu obok podcelów drukowanych przez Isabelle znakami ASCII przedstawione są ich notacje klasyczne.

Aby rozpocząć nowy dowód wystarczy napisać :

```
goal theory formula;
```

gdzie `theory` jest nazwą teorii (logiki przedmiotowej), w której chcemy dowodzić nasz cel, natomiast `formula` jest formułą zapisaną jako ML-owski string.

Jeśli na przykład chcemy rozpocząć dowód formuły

$$\bigwedge PQR.\, P \,\&\, Q \rightarrow (R \rightarrow P \,\&\, R)$$

w systemie naturalnej dedukcji klasycznej logiki intuicjonistycznej pierwszego rzędu, to musimy podać komendę

```
> goal IFOL.thy "P&Q  --> (R-->P&R)";
Level 0
P & Q --> R --> P & R
 1. P & Q --> R --> P
```
$P \,\&\, Q \rightarrow (R \rightarrow P \,\&\, R)$

Zbiór reguł dowodzenia dla tej logiki to zbiór przedstawiony w tabeli 2.1 oprócz reguły $\neg\neg$. Nazwy reguł stosowane w Isabelle pozwalają na łatwe odnalezienie tej reguły w tej tabeli.

Aby wykonać krok dowodu (tzn. zastosować taktykę do bieżącego stanu dowodu) należy napisać :

```
by tactic;
```

Najprostsze taktyki stosują reguły dowodzenia do podcelu tworząc nowe podcele. Taktyki złożone stosują reguły wielokrotnie do istniejących podcelów, potrafią stosować nawroty (w sytuacji gdy bieżącego stanu dowodu nie można już zredukować) i wyszukiwać stany alternatywne. Stosowanie tak złożonych taktyk zmniejsza znacznie ilość kroków niezbędnych do wykonania w dowodzie (czasami nawet do jednego kroku).

Wybór taktyki, stosowanej do bieżącego stanu dowodu należy do użytkownika. Decyzja powinna być uzależniona od wyglądu aktualnych podcelów. W niektórych sytuacjach można łatwo zauważyć, że dana taktyka zawiedzie. Na przykład każda taktyka, która wymaga podania numeru podcelu zawiedzie zawsze, gdy podamy jej numer większy niż liczba aktualnych podcelów.

W rozpoczętym powyżej dowodzie mamy tylko jeden podcel, którego głównym symbolem logicznym jest $\rightarrow$. Aby zastosować regułę $\rightarrow$i do tego podcelu podajemy polecenie :

```
> by (resolve_tac [impI] 1);
Level 1
P & Q --> R --> P & R
 1. P & Q ==> R --> P & R
```
$P \,\&\, Q \Longrightarrow R \rightarrow P \,\&\, R$

Funkcja `resolve_tac`, jest jedną z najczęściej używanych w budowaniu dowodów. Wywołuje się ją następująco :

```
resolve_tac thms i
```

Argument `thms` jest listą reguł, które po kolei próbuje się zastosować do podcelu o numerze `i`. Wartością tak wywołanej funkcji jest taktyka, którą należy przekazać jako argument dla polecenia `by`. Jeśli kilka reguł z podanej listy da się zastosować do tego podcelu, to zapamiętane zostaną również stany alternatywne. Taktyka zawodzi jeśli żadnej z reguł listy `thms` nie uda się zastosować do podcelu `i`.

Zastosowana w naszym przykładzie reguła jast regułą eliminującą założenie, a zatem użyta jako reguła redukcji produkuje podcel, który posiada założenie ($P \,\&\, Q$). Nowy podcel jast formułą meta-języka. Jednak z prawej strony mamy nadal implikację. Stosujemy ponownie $\rightarrow$i

```
> by (resolve_tac [impI] 1);
Level 2
P & Q --> R --> P & R
 1. [| P & Q; R |] ==> P & R
```
$P \,\&\, Q \Longrightarrow (R \Longrightarrow P \,\&\, R)$

Mamy teraz dwa założenia ($P \,\&\, Q$ oraz R). Stosujemy regułę &i :

```
> by (resolve_tac [conjI] 1);
Level 3
P & Q --> R --> P & R
 1. [| P & Q; R |] ==> P
 2. [| P & Q; R |] ==> R
```
$P \,\&\, Q \Longrightarrow (R \Longrightarrow P)$

$P \,\&\, Q \Longrightarrow (R \Longrightarrow R)$

Użyta taktyka wyprodukowała dwa podcele. Podcel 2 jest prawdziwy z założenia.

Funkcja assume_tac produkuje taktykę dowodzącą podcele z założenia. Wywołuje się ją następująco :

```
assume_tac i
```

Argument i to numer podcelu, który ma być udowodniony z założenia. Wartością jest taktyka, którą przekazuje się jako argument dla polecenia by. Taktyka ta zawodzi jeśli podcel i nie jest dowodliwy z założenia.

W celu udowodnienia podcelu 2 piszemy więc :

```
> by (assume_tac 2);
Level 4
P & Q --> R --> P & R
 1. [| P & Q; R |] ==> P
```

$$P \,\&\, Q \Longrightarrow (R \Longrightarrow P)$$

Ponieważ jednym z założeń pozostałego podcelu jest $P \,\&\, Q$, więc stosujemy regułę &E :

```
> by (resolve_tac [conjunct1] 1);
Level 5
P & Q --> R --> P & R
 1. [| P & Q; R |] ==> P & ?Q3
```

$$P \,\&\, Q \Longrightarrow (R \Longrightarrow P\&?Q3)$$

Powstały podcel zawiera zmienną schematyczną ?Q3, za którą może być podstawiona dowolna formuła. A zatem podcel 1 daje się dowieść z założenia.

```
> by (assume_tac 1);
Level 6
P & Q --> R --> P & R
No subgoals!
```

Nie mamy już żadnych podcelów, co oznacza, że dowód został zakończony.

Inną funkcją generującą taktyki jest funkcja eresolve_tac. Ostatnie dwa kroki zakończonego właśnie dowodu można zredukować do jednego przez zastosowanie tej funkcji. Wywołuje się ją następująco :

```
eresolve_tac thms i
```

Argumenty mają taką samą interpretację jak argumenty funkcji resolve_tac. Wynikiem funkcji jest taktyka, która działa podobnie jak resolve_tac thms i z następującą po niej assume_tac i tzn. stosuje odpowiednią regułę do podcelu i, a następnie próbuje dowieść z założenia pierwszy z nowo powstałych podcelów. Dodatkowo usuwa to uzgodnione założenie z pozostałych podcelów. Mimo, że nie jest to taktyka niezbędnie potrzebna (taktyki resolve_tac i assume_tac są wystarczające do zbudowania każdego dowodu) znacznie upraszcza dowody używające reguł eliminacji i zapobiega wielokrotnemu użyciu tej samej przesłanki.

Dla zilustrowania działania tej funkcji stawiamy sobie za cel formułę $(P \vee Q) \rightarrow (Q \vee P)$ i stosujemy regułę $\rightarrow$i :

```
> goal IFOL.thy "P|Q --> Q|P";
Level 0
P | Q --> Q | P
 1. P | Q --> Q | P
> by (resolve_tac [impI] 1);
Level 1
P | Q --> Q | P
 1. P | Q ==> Q | P
```

Jedyną przesłanką, z której możemy skorzystać jest $P \vee Q$. Używamy więc reguły $\vee$E, ale wykorzystując `eresolve_tac` :

```
> by (eresolve_tac [disjE] 1);
Level 2
P | Q --> Q | P
 1. P ==> Q | P
 2. Q ==> Q | P
```

Funkcja `eresolve_tac` upraszcza także dowód powstałych dwóch podcelów :

```
> by (eresolve_tac [disjI2] 1);
Level 3
P | Q --> Q | P
 1. Q ==> Q | P
> by (eresolve_tac [disjI1] 1);
Level 4
P | Q --> Q | P
No subgoals!
```

Dowód tej samej formuły przy pomocy `resolve_tac` oraz `assume_tac` byłby o trzy kroki dłuższy i mniej czytelny.

Dowody z kwantyfikatorami. Za jeden z ważniejszych aspektów Isabelle autorzy [5] uważają sposób traktowania kwantyfikatorów. Ilustracją tej własności Isabelle może być porównanie dowodu formuły $\forall x.\exists y.x = y$ z próbą udowodnienia fałszywej formuły $\exists y.\forall x.x = y$, która zawodzi z powodu zakresu wiązania kwantyfikatora.

Twierdzenie $\forall x.\exists y.x = y$ jest jednym z najprostszych zawierających obydwa kwantyfikatory. Jego dowód ilustruje stosowanie reguł $\forall$i oraz $\exists$i.

Zaczynamy dowód deklarując cel :

```
> goal IFOL.thy "ALL x. EX y. x=y";
Level 0
ALL x. EX y. x = y
 1. ALL x. EX y. x = y
```

Jedyną regułą, która może być użyta jest $\forall$i :

```
> by (resolve_tac [allI] 1);
Level 1
ALL x. EX y. x = y
 1. !! x. EX y. x = y
```

Podwójny wykrzyknik oznacza kwantyfikowanie na poziomie meta. W tej sytuacji musimy udowodnić formułę (z logiki przedmiotowej) `EX y. x = y` dla każdej wartości zmiennej `x`.

Stosujemy regułę $\exists$i.

```
> by (resolve_tac [exI] 1);
Level 2
ALL x. EX y. x = y
 1. !! x. x = ?y1(x)
```

Zmienna `y` została zamieniona na `?y1(x)`, gdzie `?y1` jest zmienną schematyczną. Jednocześnie `?y1` jest funkcją zmiennej `x`. Oznacza to, że za zmienną `?y1` można podstawić term zawierający `x`. W naszym przypadku za `?y1` zostanie podstawiona funkcja identycznościowa, co zmieni `?y1(x)` w `x`. Isabelle wykona odpowiednie podstawienie jeśli zażądamy zastosowania reguły zwrotności relacji równości :

```
> by (resolve_tac [refl] 1);
Level 3
ALL x. EX y. x = y
No subgoals!
```

Dowód jest zakończony. Zmienna `?y1` nie występowała w żadnym innym podcelu, więc niestety nie widzimy efektu podstawienia.

Formuła $\exists y.\forall x.x = y$ nie jest twierdzeniem. Zatem próba dowodu nie może się powieść :

```
> goal IFOL.thy "EX y. ALL x. x=y";
Level 0
EX y. ALL x. x = y
 1. EX y. ALL x. x = y
```

Jedyną regułą którą możemy zastosować jest $\exists$i :

```
> by (resolve_tac [exI] 1);
Level 1
EX y. ALL x. x = y
 1. ALL x. x = ?y
```

Zmienna `?y` może być zastąpiona dowolnym termem. Jednakże nie przyniesie to żadnego rezultatu dla podcelu takiej postaci. Żaden term nie jest równy każdej zmiennej x. Musimy zastosować $\forall$i :

```
> by (resolve_tac [allI] 1);
Level 2
EX y. ALL x. x = y
 1. !! x. x = ?y
```

Porównajmy bieżący stan dowodu ze stanem poprzedniego na poziomie 2. W obydwu przypadkach mamy zmienne schematyczne (`?y1` lub `?y`), i możemy za nie wstawić term, który jest wolny w całym stanie dowodu (tzn. nie zawiera zmiennych związanych występujących w tym stanie). Takie podstawienie może zmienić `?y1(x)` w term zależny od `x`, natomiast nic nie da się zrobić w aktualnej sytuacji. Próba zastosowania aksjomatu zwrotności równości tym razem zawodzi :

```
> by (resolve_tac [refl] 1);
by: tactic failed

uncaught exception ERROR
```

3.2 Logika klasyczna pierwszego rzędu

Isabelle jest systemem, który pozwala implementować dowolne systemy logiczne. Kilka powszechnie znanych znalazło miejsce w pakiecie obok Pure Isabelle. Między innymi znajdują się systemy dla klasycznej logiki pierwszego rzędu : zarówno w wersji naturalnej dedukcji jak i w wersji sekwentowej Gentzena.

Do każdego systemu dodane są również pomocnicze taktyki, dzięki którym dowody mogą być krótsze i bardziej przejrzyste.

Reguły wyprowadzania dzielimy na *bezpieczne* i *niebezpieczne*.

Definicja 3.1 *Regułę nazywamy* **bezpieczną** *o ile zastosowana do podcelu, który posiada dowód produkuje podcele, posiadające dowody.*

Regułę, która nie jest bezpieczna nazywamy **niebezpieczną**.

Na przykład reguła $\vee$i nie jest bezpieczna, ponieważ $P \vee Q$ może być formułą prawdziwą, podczas gdy P nie jest prawdziwa. Podobnie jest z regułą $\exists x.P(x)$.

W automatyzacji dowodzenia preferowane są oczywiście reguły bezpieczne, oraz te, które produkują najmniejszą liczbę nowych podcelów. Reguły niebezpieczne wymagają wykonywania nawrotów i przeszukiwań. Nawroty wykonywane są wtedy, gdy w wyniku zastosowania taktyk otrzymujemy stan dowodu, którego nie potrafimy przekształcić. Wówczas powracamy do stanu dowodu, w którym mieliśmy możliwość wyboru nowego stanu, wybieramy inny i dalej prowadzimy dowód.

3.2.1 Logika pierwszego rzędu w systemie naturalnej dedukcji

Isabelle implementuje w systemie naturalnej dedukcji zarówno logikę klasyczną (FOL) jak intuicjonistyczną (IFOL). Najpierw definiowana jest logika intuicjonistyczna, a z niej otrzymujemy klasyczną przez dodanie reguły podwójnej negacji.

Syntaktykę języka FOL przedstawia ryc. 3.1.

Teoria intuicjonistyczna nosi w ML-u nazwę `IFOL.thy`. Aksjomaty i reguły dla tej teorii oraz ich ML-owskie nazwy przedstawia tabela 3.2.

Najtrudniejszą do stosowania jest reguła eliminacji implikacji. Spośród przesłanek podcelu trzeba odnaleźć wszystkie pary formuł postaci $P \rightarrow Q$ i P. Wiedząc o prawdziwości $P \rightarrow Q$ aby udowodnić formułę Q wystarczy udowodnić P. Jeśli dowód formuły P utknie w martwym punkcie, to musimy stosować powroty. Poza tym dowodzenie w logice intuicjonistycznej często wymaga kilkakrotnego użycia przesłanki $P \rightarrow Q$ jak w poniższym przykładzie :

$$\dfrac{P \rightarrow Q \qquad \dfrac{(P \rightarrow Q) \rightarrow P \qquad P \rightarrow Q}{P}}{Q}$$

Dlatego też stosowanie reguły *modus ponens* wymaga szczególnej ostrożności.

Taktyki stworzone dla rachunku intuicjonistycznego znalazły miejsce w strukturze `Int`. Najważniejsze z nich są opisane poniżej. Opisy te dotyczą w rzeczywistości funkcji generujących taktyki (tzn. funkcje te wywołane z odpowiednimi argumentami dają taktyki). Mimo to w dalszej części będziemy dla tych funkcji używać także terminu *taktyki*.

Taktyka `mp_tac` stosuje regułę *modus ponens*. Użyta w postaci `mp_tac i` wyszukuje wśród przesłanek i-tego podcelu formuł postaci $P \rightarrow Q$ i P. Zastępuje ten podcel nowym, zamieniając $P \rightarrow Q$ na Q. Przy wyszukiwaniu odpowiedniej pary taktyka wykonuje stosowne unifikacje, dzięki czemu znaleziona zostanie każda implikacja, której poprzednik jest unifikowalny z jakąś inną przesłanką. Na przykład zastosowana do podcelu postaci :

```
[| P --> Q; Q --> R; ?S|] ==> R
```

znajdzie jako wspomnianą parę formuły `P --> Q`, `?S`, podstawi formułę `P` w miejsce `?S` i zastąpi przesłankę `P --> Q` formułą `Q` otrzymując następujący podcel :

```
[| Q; Q --> R; P|] ==> R
```

W przypadku gdy pasujących par jest więcej niż jedna, zapamiętane zostaną rozwiązania alternatywne. W naszym przykładzie alternatywny stan wyznacza para `Q --> R`, `?S`.

Taktyka `safestep_tac` wykonuje jeden krok dowodu stosując tylko reguły bezpieczne. Wywołana w postaci `safestep_tac i` próbuje dowieść podcel i z założenia lub przez sprzeczność, następnie próbuje zastosować `mp_tac`, a potem inne reguły. Nazwa tej taktyki może mylić, ponieważ taktyka ta nie jest bezpieczna z powodu wykonywanych unifikacji. Jeśli podstawia za zmienne pewne konkretne termy to może się okazać, że wyprodukowany podcel nie jest już dowodliwy.

Wywołanie `safe_tac i` powoduje próbę udowodnienia i-tego podcelu z wykorzystaniem nawrotów oraz taktyki `safestep_tac i` jako pojedynczy krok dowodu. Taktyka ta wykonuje "łatwe części" dowodu pozostawiając "trudniejsze" nieudowodnionymi.

Taktyka `step_tac` jest podstawą głównych strategii – stanowi ich pojedyncze kroki. Najpierw wykonuje `safestep_tac`, potem próbuje stosować reguły niebezpieczne.

symbol	*meta-typ*	*opis*
`=`	$[term, term] \to form$	równość ($=$)
`&`	$[form, form] \to form$	koniunkcja ($\&$)
`|`	$[form, form] \to form$	alternatywa ($\vee$)
`-->`	$[form, form] \to form$	implikacja ($\to$)
`<->`	$[form, form] \to form$	wtedy i tylko wtedy ($\leftrightarrow$)

OPERATORY

---

symbol	*meta-typ*	*opis*
`Trueprop`	$form \to prop$	meta-predykat prawdziwości
`~`	$form \to form$	negacja ($\neg$)
`Forall`	$(term \to form) \to form$	kwantyfikator ogólny ($\forall$)
`Exists`	$(term \to form) \to form$	kwantyfikator szczegółowy ($\exists$)
`True`	$form$	tautologia ($\top$)
`False`	$form$	formuła sprzeczna ($\bot$)

STAŁE

---

wewnętrznie	*zewnętrznie*	*notacja standardowa*
`ALL` $x.\ P$	`Forall`$(\lambda x.P)$	$\forall x.P$
`EX` $x.\ P$	`Exists`$(\lambda x.P)$	$\exists x.P$

TRANSLACJE

$$
\begin{array}{rcl}
form & = & \texttt{ALL}\ identifier\ \{\ identifier\ \}\ .\ form \\
 & | & \texttt{EX}\ \ identifier\ \{\ identifier\ \}\ .\ form \\
 & | & \texttt{\~{}}\ form \\
 & | & itd\ldots
\end{array}
$$

GRAMATYKA

Rysunek 3.1: Syntaktyka FOL

```
refl        a=a
sym         a=b ==> b=a
trans       [| a=b;  b=c |] ==> a=c
```

Aksjomaty równości

```
conjI       [| P;  Q |] ==> P&Q
conjunct1   P&Q ==> P
conjunct2   P&Q ==> Q

disjI1      P ==> P|Q
disjI2      Q ==> P|Q
disjE       [| P|Q;  P ==> R;  Q ==> R |] ==> R

impI        (P ==> Q) ==> P-->Q
mp          [| P-->Q;  P |] ==> Q

FalseE      False ==> P

True_def    True == False-->False
not_def     ~P == P-->False
iff_def     P<->Q == (P-->Q) & (Q-->P)
```

Reguły zdaniowe

```
allI        "(!!x. P(x)) ==> (ALL x.P(x))
spec        (ALL x.P(x)) ==> P(x)

exI         P(x) ==> EX x.P(x)
exE         "[| EX x.P(x);  !!x. P(x) ==> R |] ==> R"
```

Reguły dla kwantyfikatorów

```
allE        [| ALL x.P(x); P(x) ==> R |] ==> R
all_dupE    [| ALL x.P(x);  [| P(x); ALL x.P(x) |] ==> R |] ==> R

exCI        "(ALL x. ~P(x) ==> P(a)) ==> EX x.P(x)

swap        "~P ==> (~Q ==> P) ==> Q"
```

Reguły dziedziczone

Rysunek 3.2: Meta-aksjomaty dla IFOL

Taktykami "automatycznymi" opartymi na `step_tac` są taktyki `fast_tac` oraz `best_tac`. Są to główne taktyki struktury `Int`. Obie jako argument otrzymują numer podcelu, który mają zredukować. Różnią się między sobą sposobem wybierania dowodzonych podcelów. Taktyka `fast_tac` redukuje podcele "wgłąb", tzn. redukuje zadany (jako argument) podcel, następnie powstałe w wyniku tej redukcji nowe podcele itd. Taktyka `best_tac` stosuje przeszukiwanie "od najlepszego" (w każdym kolejnym etapie podcel, który ma być redukowany jest wybierany w zależności od ilości przesłanek oraz *rozmiaru termu* tzn. łącznej ilości atomów i abstrakcji w termie).

Teoria klasyczna jest rozszerzeniem intuicjonistycznej o następującą regułę :

$$\frac{\boxed{\begin{matrix}\neg P \\ \vdots \\ P\end{matrix}}}{P}$$

Oczywiście rozszerzenie IFOL o tę regułę jest równoważne rozszerzeniu o regułę podwójnej negacji.

Taktyki dla tego systemu zostały złożone w strukturę o nazwie `Cla`. Główne funkcje tej struktury mają takie same nazwy jak dla IFOL. Wywołuje się je następująco :

`safestep_tac` *cs i*
`safe_tac` *cs*
`step_tac` *cs i*
`fast_tac` *cs i*
`best_tac` *cs i*

gdzie *cs* jest stosownym zbiorem reguł pogrupowanych tak, aby taktyka mogła rozróżnić reguły bezpieczne od niebezpiecznych oraz reguły eliminujące operatory od wprowadzających je. Nie musimy tych zbiorów definiować, bowiem Isabelle dostarcza nam następujących :

`prop_cs` : zbiór reguł dla logiki zdaniowej,

`FOL_cs` : rozszerzenie `prop_cs` o reguły bezpieczne `allI`, `exE`, oraz niebezpieczne `allE`, `exI`,

`FOL_dup_cs` : rozszerzenie `prop_cs` o reguły bezpieczne `allI`, `exE`, oraz niebezpieczne `all_dupE`, `exCI`.

Korzystanie z `FOL_cs` daje gwarancję, że formuła z kwantyfikatorem nie będzie użyta więcej niż raz. `FOL_dup_cs` daje możliwość wielokrotnego korzystania z tej samej formuły z kwantyfikatorem, jednak taktyki używające tego zbioru mogą nie skończyć pracy. Dlatego też ten drugi zbiór najlepiej używać tylko z taktyką `best_tac`

Poza przedstawioną różnicą wymienione taktyki działają dokładnie tak samo jak ich odpowiedniki w `Int`.

3.2.2 Przykłady dowodów w systemie naturalnej dedukcji

Dobrą ilustracją dowodzenia w FOL jest dowód formuły $\exists y.\forall x.P(y) \to P(x)$. W logice klasycznej jest ona prawdziwa, bowiem jeśli istnieje y takie, że $\neg P(y)$, to wystarczy wziąć to właśnie y. W przeciwnym przypadku $\forall x.P(x)$ jest prawdziwe, a stąd i cała formuła.

Formalny dowód nie ma jednak nic wspólnego z powyższym wywodem logicznym. Punktem kluczowym jest zastosowanie na samym początku reguły `exCI` :

```
> goal FOL.thy "EX y. ALL x. P(y)-->P(x)";
Level 0
EX y. ALL x. P(y) --> P(x)
 1. EX y. ALL x. P(y) --> P(x)
> by (resolve_tac [exCI] 1);
Level 1
EX y. ALL x. P(y) --> P(x)
 1. ALL y. ~(ALL x. P(y) --> P(x)) ==> ALL x. P(?a) --> P(x)
```

Następne kroki są oczywistym upraszczaniem przesłanek i konkluzji :

```
> by (resolve_tac [allI] 1);
Level 2
EX y. ALL x. P(y) --> P(x)
 1. !! x. ALL y. ~ (ALL x. P(y) --> P(x)) ==>  P(?a) --> P(x)
> by (resolve_tac [impI] 1);
Level 3
EX y. ALL x. P(y) --> P(x)
 1. !! x. [| ALL y. ~ (ALL x. P(y) --> P(x)); P(?a) |] ==> P(x)
> by (eresolve_tac [allE] 1);
Level 4
EX y. ALL x. P(y) --> P(x)
 1. !! x. [| P(?a); ~ (ALL xa. P(?y3(x)) --> P(xa)) |] ==> P(x)
```

W logice klasycznej zanegowana przesłanka jest równoważna konkluzji. Konkluzja `P(x)` może więc stać się po zanegowaniu założeniem. Otrzymujemy ten efekt stosując taktykę :

```
> by (eresolve_tac [swap] 1);
Level 5
EX y. ALL x. P(y) --> P(x)
 1. !! x [| P(?a); ~ P(x) |] ==> ALL xa. P(?y3(x)) --> P(xa)
```

Dalej analizujemy nową konkluzję :

```
> by (resolve_tac [allI] 1);
Level 6
EX y. ALL x. P(y) --> P(x)
 1. !! x xa. [| P(?a); ~ P(x) |] ==> P(?y3(x)) --> P(xa)
> by (resolve_tac [impI] 1);
Level 7
EX y. ALL x. P(y) --> P(x)
 1. !! x xa. [| P(?a); ~ P(x); P(?y3(x)) |] ==> P(xa)
```

Otrzymaliśmy jeden podcel z trzema przesłankami. Możemy wykorzystać sprzeczność pomiędzy przesłankami `~P(x)` oraz `P(?y3(x))`.

```
> by (eresolve_tac [notE] 1);
Level 8
EX y. ALL x. P(y) --> P(x)
 1. !! x xa. [| P(?a); P(?y3(x)) |] ==> P(x)
> by (assume_tac 1);
Level 9
EX y. ALL x. P(y) --> P(x)
No subgoals!
```

To samo twierdzenie można udowodnić stosując taktykę `best_tac` :

```
> goal FOL.thy "EX y. ALL x. P(y)-->P(x)";
Level 0
EX y. ALL x. P(y) --> P(x)
 1. EX y. ALL x. P(y) --> P(x)
> by (best_tac FOL_dup_cs 1);
Level 1
EX y. ALL x. P(y) --> P(x)
No subgoals!
```

3.3 Definiowanie nowego języka logiki

Isabelle dostarcza pewnych środków, które czynią definiowanie nowych logik przedmiotowych (teorii) bardzo przejrzystym, oraz pozwalają użytkownikowi nie wnikać głęboko w aspekty techniczne pakietu. Niezbędna jest znajomość tylko niewielkiej części rozwiązań wewnętrznych systemu.

Każda logika przedmiotowa jest reprezentowana wewnętrznie jako obiekt ML-owskiego typu `theory`. Zaimplementowano szereg funkcji (w ML-u), które pozwalają rozbudowywać teorie. Można rozszerzać wcześniej zdefiniowane o nowe symbole logiki, reguły, itp. Można także łączyć dwie teorie w jedną, itd.

Aby ułatwić użytkownikowi pracę i oszczędzić mu pisania skomplikowanego kodu w ML-u autorzy Isabelle stworzyli własny język służący definiowaniu teorii.

W celu zdefiniowania nowej teorii wystarczy stworzyć plik dyskowy (o nazwie zakończonej przez ".thy"), którego zawartość ma następującą postać [3] :

$T = S_1 + \ldots + S_n +$

`classes`	*deklaracje klas*
`default`	*sort*
`types`	*deklaracje typów*
`arities`	*opisy arności*
`consts`	*deklaracje stałych*
`rules`	*reguły dowodzenia*
`end`	
`ML`	*kod w ML-u*

Aby korzystać z tej teorii należy po jej zdefiniowaniu "załadować" ją z poziomu Isabelle wydając polecenie

```
> use_thy nazwa
```

gdzie `nazwa` jest nazwą pliku dyskowego (obciętą o końcówkę ".thy") zawierającego definicję teorii.

Isabelle połączy teorie $S_1, \ldots, S_n$ w jedną całość, rozszerzy wynik o dodatkowe deklaracje i tak utworzonej teorii nada nazwę T.

Poszczególne deklaracje są opisane poniżej. Każda z nich może zostać pominięta. Zatem najprostsza definicja teorii będzie opisywała połączenie teorii $S_1, \ldots, S_n$.

Nowa teoria musi dziedziczyć własności przynajmniej jednej teorii. W Pure Isabelle zdefiniowana została teoria *Pure*, która zawiera meta-logikę. Każda teoria musi dziedziczyć (bezpośrednio lub pośrednio) własności tej teorii.

Część deklarująca typy ma następującą postać :

`types` $id_1 \; k_1$

$\vdots$

$id_n \; k_n$

gdzie $id_1, \ldots, id_n$ są identyfikatorami, a $k_1, \ldots k_n$ liczbami naturalnymi. *Identyfikatorem* jest każdy ciąg znaków złożony z liter, cyfr, apostrofów i znaków podkreślenia rozpoczynający się literą.

Powyższa deklaracja mówi, że każdy id_i jest konstruktorem typu o argumentowości k_i.

Polimorfizmy w Isabelle są realizowane poprzez *klasy*. Klasa jest oznaczeniem zbioru typów. Jeśli definiując konstruktor typu bądź operator języka nie będziemy precyzować typu argumentu a wskażemy klasę jego typu, to ten konstruktor (lub operator) będziemy mogli stosować dla argumentów każdego typu należącego do wskazanej klasy.

Zbiór klas jest częściowo uporządkowany przez relację "*jest podklasą*". Jest to klasyczna relacja "*jest podzbiorem*" określona na zbiorze zbiorów typów. W Pure Isabelle została zdefiniowana klasa *logic*, której intencją jest zawieranie wszystkich typów, które będziemy rozważać w logikach przedmiotowych. W związku z tym każda nowa klasa powinna być deklarowana jako podklasa klasy *logic*. Ponieważ porządek na zbiorze klas jest częściowy, więc relacja podklasy jest przechodnia. Isabelle pilnuje wewnętrznie przechodniości, dzięki czemu możemy na przykład zadeklarować klasę *term* (wszystkich typów termów) jako poklasę klasy *logic*, a następnie klasę *arith* (wszystkich typów numerycznych) jako podklasę klasy *term*. Isabelle "rozumie", że *arith* jest też podklasą *logic*.

Deklaracja klas ma następującą postać :

classes id_1 < c_1
$\vdots$
id_n < c_n

gdzie $id_1, \ldots, id_n$ są identyfikatorami, a $c_1, \ldots, c_n$ istniejącymi klasami. Jedna klasa może być zadeklarowana jako podklasa kilku klas poprzez wypisanie ich wszystkich (oddzielonych przecinkami) po znaku "<".

Powyższy przykład deklaracji klas *term* i *arith* może być sformalizowany w Isabelle następująco :

```
classes  term < logic
         arith < term
```

Dla każdego konstruktora typu musimy sprecyzować typy (w przypadku konstruktorów polimorficznych - klasy) jego argumentów oraz tworzonego typu. Odpowiedzialna za to jest następująca deklaracja :

arities $tycon_1$ :: $arity_1$
$\vdots$
$tycon_n$ :: $arity_n$

gdzie $tycon_1, \ldots, tycon_n$ są identyfikatorami, natomiast $arity_1, \ldots, arity_n$ - *opisami arności*. Opis arności dla konstruktora 0-argumentowego to po prostu nazwa klasy tworzonego typu. Dla konstruktora k-argumentowego opis taki ma postać $(s_1, \ldots, s_k)c$, gdzie $s_1, \ldots, s_k$ są *sortami*, a c klasą. Każdy sort precyzuje typ odpowiedniego argumentu i ma postać $\{c_1, \ldots, c_m\}$, gdzie $c_1, \ldots, c_m$ są klasami. Sorty rozumiemy jako przekroje klas. Zatem do sortu $\{c_1, \ldots, c_m\}$ należą wszystkie typy, którę należy do każdej z klas $c_1, \ldots, c_m$. Zauważmy, że do sortu $\{\}$ należą wszystkie typy. W przypadku sortów jednoelementowych klamry można pominąć.

Jeśli zdefiniowaliśmy typy :

```
types  bool 0
       list 1
```

to możemy opisać ich arności następująco :

```
arities  bool :: term
         list :: (term)term
```

Stałymi danego języka przedmiotowego są wszystkie elemen tytego języka, którym przypisujemy typy (także funkcyjne). Mogą to być na przykład stałe boolowskie (*true*, *false*), a także operatory logiczne ($\neg$, $\rightarrow$, ...) oraz kwantyfikatory. Stosowna deklaracja ma następującą postać :

consts c_1 :: τ_1
$\vdots$
c_n :: τ_n

gdzie $c_1, \ldots, c_n$ są identyfikatorami, natomiast $\tau_1, \ldots, \tau_n$ – typami.
Jeśli chcemy nadać identyfikatorowi nazwę, która nie jest identyfikatorem (np. `-->`), to ujmujemy ją w cudzysłów. Kilka stałych tego samego typu można definiować w jednej linii oddzielając ich identyfikatory przecinkami. Dodatkowo obok deklaracji stałej możemy umieścić informacje o sposobie drukowania jej na ekranie, oraz jeśli jest to stała z argumentami o sile i sposobie wiązania (tzw. priorytetach). Są to jednak informacje nieistotne z logicznego punktu widzenia.

Jeśli zdefiniowaliśmy typ "*o*" jako typ wszystkich formuł języka przedmiotowego, to deklaracja stałych może wyglądać następująco :

```
consts  "&" :: "[o, o] => o"
        true, false :: "bool"
```

Deklaracja reguł wyprowadzania wygląda następująco :

rules id_1 "$rule_1$"

$\vdots$

id_n "$rule_n$"

gdzie $id_1, \ldots, id_n$ są ML-owskimi identyfikatorami, a $rule_1, \ldots, rule_n$ są wyrażeniami typu *prop*. Typ *prop* jest zdefiniowany w Pure Isabelle jako typ wszystkich meta-formuł.

Deklaracja

`default` s

gdzie s jest sortem mówi, że domyślnym sortem zmiennych typowych będzie s. Zatem każda zmienna typowa z którą nie ma jawnie dowiązanego (operatorem `::`) sortu będzie przez Isabelle uznawana za zmienną typową sortu s.

Słowo kluczowe `end` pojawia się zawsze na końcu listy deklaracji.

Słowo `ML` rozpoczyna część traktowaną jak źródło programu w ML-u. Możliwość dołączenia takiego kodu jest wykorzystywana w celu zdefiniowania pewnych dodatkowych funkcji związanych z definiowaną teorią.

Przykład 3.1 *Stwórzmy w Isabelle teorię dla reprezentacji Hilberta klasycznej logiki zdaniowej. W tym celu zakładamy plik dyskowy o następującej zawartości :*

```
Hilbert = Pure +
types
  form 0
arities
  form :: logic
consts
  Trueprop :: "form => prop"          ("_" 5),
  not      :: "form => form"          ("~ _" 20)
  "-->"    :: "[form, form] => form"  (infixr 10)
rules
  A1    "P --> Q --> P"
  A2    "(P --> Q --> R) --> (P --> Q) --> P --> R"
  A3    "(~ P --> ~ Q) --> (Q --> P)"
  MP    "[| P --> Q; P |] ==> Q"
end
```

Zdefiniowaliśmy w ten sposób nową teorię o nazwie `Hilbert.thy`*. W tym systemie istnieje 0-argumentowy konstruktor typu o nazwie* `form`*. Należy on do klasy* `logic`*.*

Wprowadziliśmy dwa nowe operatory logiczne (`~`*,*`-->`*). Wskazaliśmy typy ich argumentów oraz zwracanej wartości.*

Ponieważ chcemy operować formułami z meta-języka musimy wskazać Isabelle, że każda formuła stworzonej logiki przedmiotowej jest formułą w meta-języku. Sugestię tę wyraża operator `Trueprop`*.*

W nawiasach okrągłych podane są dodatkowe informacje mówiące o tym jak dane operatory będą wypisywane na ekranie, oraz jaka jest "siła ich wiązania". Napis `infixr` *mówi, że operator* `-->` *będzie pisany między argumentami.*

Po wczytaniu deklaracji z pliku poleceniem `use_thy` *mamy utworzoną teorię* `Hilbert.thy`*. Wiedząc, że* `getaxiom thy name` *to reguła teorii* `thy` *o nazwie* `name` *możemy przystąpić do budowania dowodów w nowej teorii :*

```
> goal Hilbert.thy "P --> P";
Level 0
P --> P
 1. P --> P
> by(resolve_tac [get_axiom Hilbert.thy "MP"] 1);
Level 1
```

```
P --> P
 1. ?P --> P --> P
 2. ?P
> by(resolve_tac [get_axiom Hilbert.thy "MP"] 1);
Level 2
P --> P
 1. ?P1 --> ?P --> P --> P
 2. ?P1
 3. ?P
> by(resolve_tac [get_axiom Hilbert.thy "A2"] 1);
Level 3
P --> P
 1. P --> ?Q2 --> P
 2. P --> ?Q2
> by(resolve_tac [get_axiom Hilbert.thy "A1"] 1);
Level 4
P --> P
 1. P --> ?Q2
> by(resolve_tac [get_axiom Hilbert.thy "A1"] 1);
Level 5
P --> P
No subgoals!
```

Dowód ten jest kolejnym potwierdzeniem faktu, że dowodzenie w systemie Hilberta jest bardzo żmudne i mało czytelne. Dlatego też najczęściej stosuje się systemy sekwentowe Gentzena lub systemy naturalnej dedukcji.

3.4 Lambda rachunek z typami (λ-kostka)

Podrozdział ten przedstawia syntaktyczny system wnioskowania dla λ-kostki w ujęciu Barendregta [7]. Nie rozróżniamy tutaj typów i termów, ale zajmujemy się *pseudo-wyrażeniami*, wśród których odnajdujemy zarówno jedne jak i drugie. Pseudo-wyrażenia zdefiniowane są następującą gramatyką bezkontekstową :

$$T = V \,|\, C \,|\, TT \,|\, \lambda V : T.T \,|\, \Pi V : T.T \,|\, T \to T,$$

gdzie V jest zbiorem zmiennych (typowych i termowych), a $C = \{*, \}$ jest zbiorem *sortów*.

Deklaracje to napisy postaci $M : \sigma$, gdzie M i σ są pseudo-wyrażeniami. Zbiór deklaracji jest zarazem zbiorem wszystkich formuł języka. Każdy skończony ciąg deklaracji jest nazywany *kontekstem*.

Opis syntaktyczny teorii dla tego systemu wygląda następująco :

```
Cube = Pure +

types term, context, typing 0

arities term :: logic

consts
  Abs,Prod   :: "[term, term=>term] => term"
  Trueprop   :: "[context,typing] => prop"     ("(_ |- _)")
  Trueprop1  :: "typing => prop"               ("(_)")
  MT_context :: "context"                      ("")
  ""         :: "id  => context"               ("_ ")
  ""         :: "var => context"               ("_ ")
```

```
  Context    :: "[typing,context] => context"  ("_ _")
  star       :: "term"                         ("*")
  box        :: "term"                         ("[]")
  "^"        :: "[term,term] => term"          (infixl 20)
  Lam        :: "[id,term,term] => term"       ("(Lam _:_. _)" [0,0,0] 10)
  Pi         :: "[id,term,term] => term"       ("(Pi _:_. _)" [0,0] 10)
  "->"       :: "[term,term] => term"          (infixr 10)
  Has_type   :: "[term,term] => typing"        ("(_: _)" [0,0] 5)

rules
  s_b        "*: []"

  strip_s    "[| A:*;  a:A ==> G |- x:X |] ==> a:A G |- x:X"
  strip_b    "[| A:[]; a:A ==> G |- x:X |] ==> a:A G |- x:X"

  app        "[| F:Prod(A,B); C:A |] ==> F^C: B(C)"

  pi_ss      "[| A:*; !!x. x:A ==> B(x):* |] ==> Prod(A,B):*"

  lam_ss     "[| A:*; !!x. x:A ==> f(x):B(x); !!x. x:A ==> B(x):* |]
              ==> Abs(A,f) : Prod(A,B)"

  beta       "Abs(A,f)^a == f(a)"
end
```

Pierwsza linia deklaracji mówi, że zostanie zbudowany system o nazwie *Cube*, który będzie rozszerzeniem systemu Pure o występujące dalej deklaracje.

W nowej teorii istnieją trzy nowe 0-argumentowe konstruktory typów o nazwach `term`, `context`, `typing`. Typ `term` należy do klasy `logic` i reprezentuje pseudo-wyrażenia języka λ-kostki.

Typ `typing` to typ wszystkich deklaracji. Posiada on konstruktor ":" o dwóch argumentach typu `term`.

Teoria *Cube* zawiera stałą typu `context` (o nazwie `MT_context`), która oznacza kontekst pusty, oraz dwie stałe typu `term` (`star`=$*$, `box`=). Nietrywialne konteksty budujemy składając deklaracje operatorem `Context`. Mamy też wszystkie konstruktory λ-termów i konstruktor typu funkcyjnego.

Formuły, które chcemy dowodzić w tej teorii muszą mieć postać `context |- typing` (wewnętrznie takie sekwenty budowane są przy pomocy stałej `Trueprop`. `Trueprop1` jest skróconym opisem sekwentu o pustym poprzedniku.

Dla ułatwienia wykonywanych β-redukcji konstruktory `Lam` i `Pi` są wewnętrznie tłumaczone zgodnie z poniższym schematem :

`Lam x:A. B` $\longrightarrow$ `Abs(A, %x. B)`
`Pi x:A. B` $\longrightarrow$ `Prod(A, %x. B)`

Konwersje te są zdefiniowane po słowie kluczowym `ML`.

Wszystkie przedstawione wyżej deklaracje określają podstawową teorię *lambda*-kostki, która nosi w Isabelle nazwę `Cube`. Definicje te są umieszczone w pliku `cube.thy`. Isabelle wczytując nową teorię (po otrzymaniu polecenia `use_thy` *nazwa*) czyta najpierw definicje z pliku *nazwa*`.thy`, a następnie szuka pliku o nazwie *nazwa*`.ML` w celu wczytania dodatkowych instrukcji. Plik `cube.ML` definiuje pozostałe systemy kostki poprzez dodanie do teorii `Cube` stosownych reguł wnioskowania. Rozszerzenia te noszą odpowiednio nazwy `L2_thy`, `Lomega_thy`, itp. Poza tym dla każdego systemu λ-kostki zdefiniowana jest lista jego reguł. Dla teorii `Cube` nazywa się ona `simple`, dla pozostałych odpowiednio `L2`, `Lomega`, itp.

Definiując dwie dodatkowe funkcje :

```
val UNTIL1 = DEPTH_SOLVE_1;
val UNTIL_THM = DEPTH_FIRST (has_fewer_prems 1);
```

możemy znacznie uprościć dowodzenie wielu twierdzeń λ-kostki. Pierwsza z nich wywołana w postaci `UNTIL1` *taktyka*, daje nową taktykę, która polega na stosowaniu taktyki *taktyka* do wszystkich nowych stanów dowodu, aż do zakończenia dowodu lub uzyskania stanu, w którym *taktyka* zawiedzie. Druga różni się od pierwszej tylko tym, że próbuje redukować najpierw te podcele, które mają najmniej przesłanek.

Wywołanie `ares_tac` *lista_reguł* n tworzy taktykę, która próbuje dowieść podcel o numerze n z założenia, a jeśli to się nie powiedzie, to stosuje taktykę `resolve_tac` *lista_reguł* n.

Używając powyższych narzędzi możemy budować wiele dowodów wykonując tylko jeden krok. Oto przykład :

```
> goal Cube.thy "A:* |- A->A : ?T";
Level 0
A: * |- A -> A: ?T
 1. A: * |- A -> A: ?T
val it = [] : thm list
> by(UNTIL_THM (ares_tac simple 1));
Level 1
A: * |- A -> A: *
 No subgoals!
val it = () : unit
```

Zauważmy, że jako cel podaliśmy formułę meta-języka zawierającą zmienną schematyczną. W ten sposób zadaliśmy pytanie ”*jaki typ ma term* `A->A`”. W wyniku zastosowania taktyki za zmienną `?T` został podstawiony term `*`, co doprowadziło do zakończenia dowodu. Podstawienie to jest odpowiedzią na postawione pytanie.

Ta sama taktyka daje odpowiedzi także dla bardziej skomplikowanych termów :

```
> goal L2_thy "|- Lam A:*. Lam a:A.a : ?T";
Level 0
Lam A:*. Lam a:A. a: ?T
 1. Lam A:*. Lam a:A. a: ?T
val it = [] : thm list
> by(UNTIL_THM (ares_tac L2 1));
Level 1
Lam A:*. Lam a:A. a: Pi x:*. x -> x
No subgoals!
val it = () : unit
```

W podobny sposób można stawiać pytania czy dany typ jest ”zamieszkały” :

```
> goal Cube.thy "A:* |- ?t : A->A";
Level 0
A: * |- ?t: A -> A
 1. A: * |- ?t: A -> A
val it = [] : thm list
> by (resolve_tac [strip_b] 1);
Level 1
A: * |- ?t: A -> A
 1. *: []
 2. A: * ==> ?t: A -> A
val it = () : unit
> by (resolve_tac [s_b] 1);
Level 2
A: * |- ?t: A -> A
 1. A: * ==> ?t: A -> A
```

```
val it = () : unit
> by (resolve_tac [lam_ss] 1);
Level 3
A: * |- Abs(A,?f2): A -> A
 1. A: * ==> A: *
 2. !!x. [| A: *; x: A |] ==> ?f2(x): A
 3. !!x. [| A: *; x: A |] ==> A: *
val it = () : unit
> by (assume_tac 1);
Level 4
A: * |- Abs(A,?f2): A -> A
 1. !!x. [| A: *; x: A |] ==> ?f2(x): A
 2. !!x. [| A: *; x: A |] ==> A: *
val it = () : unit
> by (assume_tac 2);
Level 5
A: * |- Abs(A,?f2): A -> A
 1. !!x. [| A: *; x: A |] ==> ?f2(x): A
val it = () : unit
> by (assume_tac 1);
Level 6
A: * |- Lam x:A. x: A -> A
No subgoals!
val it = () : unit
```

Zauważmy, że podstawienia za zmienne schematyczne, powodują zmianę dowodzonego celu. Twierdzeniem udowodnionym nie jest formuła, która była argumentem funkcji `goal`, ale ta, która została wypisana w ostatnim poziomie dowodu.

Autorzy Isabelle dostarczają razem z pakietem wielu interesujących przykładów dowodów dla każdej z zaimplementowanych logik przedmiotowych (także dla λ-kostki).

Spis Literatury

[1] M. Ryan and M. Sadler. *Valuation Systems and Consequence relations.* Handbook of Logic in Compueter Science, volume 1, Clarendon Press, Oxford 1992.

[2] G. Takeuti. *Proof Theory*, North-Holland Publishing Company, 1975.

[3] L. C. Paulson. *Introduction to Isabelle*, 1993.

[4] L. C. Paulson. *The Isabelle Reference Manual*, 1993.

[5] L. C. Paulson and Tobias Nipkow. *Isabelle Tuturial and Users Manual*, 1991.

[6] L. C. Paulson. *Designing a Theorem Prover*, University of Cambridge, 1990.

[7] H. P. Barendregt. *Lambda Calculi with Types.* Handbook of Logic in Compueter Science, volume 2, Clarendon Press, Oxford 1992.