Interaktywne dowodzenie twierdzeń w logikach modalnych

(Interactive theorem proving in modal logics)

Mikołaj Korobczak

Praca inżynierska

Promotor: dr Małgorzata Biernacka

Uniwersytet Wrocławski
Wydział Matematyki i Informatyki
Instytut Informatyki

11 sierpnia 2022

Streszczenie

Od początku istnienia komputerów są one wykorzystywane w badaniu logiki. Dotychczas powstało wiele narzędzi do badania logik klasycznych, ale w przypadku logiki modalnej, dostępność takich narzędzi jest znacznie mniejsza, a narzędzi podobnych do Coqa, badających intuicjonistyczną logikę modalną, w zasadzie nie ma. W pracy przedstawię PAMLa – asystenta dowodzenia twierdzeń w intuicjonistycznej logice modalnej. Narzędzie to pozwala interaktywnie dowodzić twierdzenia w popularnych modelach logiki modalnej przy pomocy reguł dedukcji naturalnej, implementuje automatyzację i pozwala generować dowody w postaci plików latex. Opiszę teoretyczne podstawy działania programu, interfejs oraz nakreślę szczegóły implementacyjne.

Since the beginning of computers, they have been used in the study of logic. Many tools for the study of classical logics have been developed so far, but in the case of modal logic, the availability of such tools is much more limited, and Coq-like tools that explore intuitionistic modal logic are basically non-existent. In this paper, I will introduce PAML – an assistant for proving theorems in intuitionistic modal logic. This tool allows for interactive proof theorems in popular models of modal logic using rules of natural deduction, implements automation and allows to generate proofs in the form of latex files. I will describe the theoretical basis of the program's operation, the interface and outline implementation details.

Spis treści

1. Wstęp **7**

2. Wprowadzenie do logik modalnych **9**

2.1. Podstawy logiki modalnej 9

2.2. Formalna definicja konstruktywnej logiki modalnej 10

2.2.1. Formuły 10

2.2.2. Modele modalne 10

2.3. Reguły dedukcji naturalnej 12

2.3.1. Podejście top-down 13

2.3.2. Podejście bottom-up 15

2.3.3. Przykład konieczności reguły osłabiania 15

2.3.4. Równoważność systemów top-down i bottom-up 15

2.4. Przykłady 17

2.5. Zastosowania 19

3. Interaktywne dowodzenie **23**

3.1. Historia, istniejące asystenty dowodzenia twierdzeń 23

3.2. Interface 24

3.2.1. Wprowadzenie 24

3.2.2. Tryby działania PAMLa 24

3.2.3. Taktyki 26

3.2.4. Podpowiedzi, Auto i Fauto 27

3.2.5. Obsługa plików 29

3.3. Przykładowe użycie 29

4. Implementacja **31**

4.1. Główne moduły 31

4.2. Formuły, drzewa dowodu i reguły dowodzenia 32

4.3. Taktyki i drzewo celów 33

4.4. Rozszerzalność i możliwości dalszego rozwoju 34

5. Podsumowanie **35**

Bibliografia **37**

Rozdział 1.

Wstęp

Podstawy logiki znane są już od starożytności, a jej zasady są używane w wielu dziedzinach nauki. Jest ona stosowana w wielu miejscach w informatyce, od budowy procesorów, przez ważny w teorii złożoności obliczeniowej problem SAT, kończąc na formalnej budowie języków programowania. Ta ostatnia używa izomorfizmu Currego-Howarda, dzięki któremu formuły logiczne stają się typami termów. Aby to było możliwe, potrzebny jest mechanizm rozstrzygający, czy term jest poprawnie stypowany. Do tego celu używa się logikę intuicjonicjonistyczną, w której formuła jest prawdziwa, jeżeli istnieje drzewo dowodu zbudowane przy pomocy reguł dedukcji naturalnej. Z tego powodu badanie logiki intuicjonistycznej jest ważne dla współczesnej informatyki.

Gdy w połowie XX wieku powstawały pierwsze komputery, pojawił się pomysł, żeby je wykorzystać również w badaniach logiki. Dotychczas powstało wiele takich narzędzi, ale my skupimy się na takich, które pozwalają interaktywnie dowodzić twierdzenia w logikach intuicjonistycznych. Jednym z najbardziej znanych jest system dowodzenia twierdzeń Coq [1]. Składa się on z jądra, sprawdzającego poprawność dowodów, którego poprawność została formalnie udowodniona [12], oraz taktyk, dzięki którym użytkownik może wygodnie zbudować drzewo dowodu. Na wzór Coqa powstał PAML [8] (*Proof Assistant for Modal Logic*), który skupia się konkretnie na dowodzeniu twierdzeń w intuicjonistycznej logice modalnej.

Podstawowe idee logik modalnych istnieją w myśli filozoficznej już od dawna, ale ich formalna definicja i badanie zaczęły się dopiero w XX wieku. Tak jak w przypadku innych logik, powstało kilka narzędzi, pozwalających wykorzystać komputery do badań nad logiką modalną. Strona konferencji *Advances in Modal logic* zbiera takie narzędzia [2]. Oprócz różnych, które wizualizują logiki modalne, lub tłumaczą do innych systemów logicznych, powstało też wiele systemów wspomagających dowodzenie twierdzeń, które różnymi metodami pozwalają dowodzić twierdzenia w tych logikach. Większość opiera się jednak na metodach znanych z badania logik klasycznych (np. implementując algorytm tableau). Nie znalazłem jednak takiego, który pozwala dowodzić twierdzenia w intuicjonistycznej logice modalnej. PAML

jest więc jednym z pierwszych tego typu asystentów dowodzenia twierdzeń.

PAML jest programem tekstowym, który pozwala użytkownikowi interaktywnie dowodzić twierdzenia przy wykorzystaniu taktyk. Poprawność każdego dowodu jest weryfikowana przez moduł, który implementuje reguły dedukcji naturalnej. PAML pozwala w wygodny sposób definiować relacje widoczności lub pracować w wybranym modelu modalnym, używać wcześniej udowodnionych twierdzeń w budowaniu dowodu oraz dzięki taktykom *auto* i *fauto*, dowodzić automatycznie. Pozwala również zapisać pracę oraz wygenerować plik latex zawierający, zbudowane w trakcie pracy, drzewa dowodów.

Rozdział 2.

Wprowadzenie do logik modalnych

2.1. Podstawy logiki modalnej

Logika modalna rozszerza logikę klasyczną o dwa nowe, unarne spójniki $\Box$ oraz $\Diamond$. Intuicyjnie będziemy o nich myśleć jako, odpowiednio, o konieczności (*necessity*) oraz możliwości (*possibility*). Prawdziwość formuł $\Box A$ i $\Diamond A$ nie zależy od prawdziwości A, więc trzeba przyjąć jakąś semantykę do formalnego badania nowych spójników.

W 1959 roku Saul Kripke zaproponował semantykę logiki modalnej (zwaną semantyką Kripkego), w której, aby mówić coś o konieczności i możliwości, wprowadzimy model światów, w których zachodzą pewne formuły oraz binarną relację widoczności (*accessibility* albo *visibility*), która zawiera informację o tym, czy jakiś świat w „widzi" świat v. W takiej interpretacji definiujemy konieczność następująco: formuła „*Koniecznie A*" w świecie w jest prawdziwa wtedy i tylko wtedy gdy we wszystkich światach widzianych przez w zachodzi A. Analogicznie dla możliwości mamy, że „*Możliwe A*" jest prawdą w w wtedy i tylko wtedy, gdy istnieje jakiś świat w' widoczny z w, w którym A jest prawdziwe. W tej pracy skupimy się właśnie na semantyce Kirpkego. Powstało wiele modeli logiki modalnej, różniące się przyjętym zbiorem założeń, które skrótowo nazwiemy logikami modalnymi [16].

Klasyczna logika opiera badania formuł logicznych na wartościowaniu zmiennych zdaniowych, a następnie całych formuł, dwiema wartościami – prawdą i fałszem. Innym systemem logicznym jest logika intuicjonistyczna, która opiera się na dowodliwości formuły. W tym celu możemy skorzystać z zestawu reguł dostarczonych przez dedukcję naturalną. Reguły dowodzenia dzielimy na reguły wprowadzania i eliminacji dostępnych spójników logicznych. Przykładowo reguła wprowadzenia implikacji mówi, że jeżeli przy założeniu, że zachodzi formuła A, wiemy, że zachodzi formuła B, to możemy stwierdzić, że prawdziwa jest również formuła $A \supset B$, co zapiszemy następującym drzewem dowodu:

$$\frac{\Gamma, A \vdash B}{\Gamma \vdash A \supset B} \ (\supset I)$$

Zdania, z których wnioskujemy w regule, nazywamy przesłankami i zapisujemy nad kreską, a wniosek nazywamy konsekwencją i zapisujemy go pod kreską. W przyjętej przeze mnie dedukcji naturalnej, opartej na tej wprowadzonej w pracy A. K. Simpsona [11], będę jawnie wskazywał, które formuły zachodzą w którym świecie oraz które światy są ze sobą w relacji widoczności. Nie jest to konieczne i można zdefiniować zasady dedukcji naturalnej dla logiki modalnej, w której nie trzeba jawnie wprowadzać pojęcia światów, na przykład F. Pfenning w [10] zamiast wiedzy o światach ma dwa zbiory założeń.

2.2. Formalna definicja konstruktywnej logiki modalnej

2.2.1. Formuły

Formułę w języku rachunku zdań definiujemy indukcyjnie:

1. $\bot$ jest atomową, niedowodliwą formułą.
2. Każda zmienna zdaniowa jest formułą.
3. Jeśli A i B są formułami to $A \wedge B$ też.
4. Jeśli A i B są formułami to $A \vee B$ też.
5. Jeśli A i B są formułami to $A \supset B$ też.
6. Jeśli A jest formułą to $\Box A$ też.
7. Jeśli A jest formułą to $\Diamond A$ też.

Dodatkowo

1. jeśli A jest formułą to przez $\neg A$ będziemy oznaczać $A \supset \bot$,
2. przez $\top$ będziemy oznaczać $\neg\bot$,
3. jeśli A i B są formułami to przez $A \leftrightarrow B$ będziemy oznaczać $(A \supset B) \wedge (B \supset A)$.

2.2.2. Modele modalne

Model modalny to trójka $\mathcal{M} = \langle W, R, V \rangle$, gdzie W to niepusty zbiór światów, $R \subseteq W \times W$, a V to funkcja z W w zbiór formuł, które zachodzą dla danego świata. Przez :: będziemy rozumieli relację spełniania (*satisfaction*), czyli zapis $w :: A$ będzie oznaczać, że w świecie w zachodzi formuła A. Relację tę definiujemy indukcyjnie:
Dla dowolnego świata w

Oznaczenie	Nazwa	Własność
R_D	szeregowalność (*seriality*)	$\forall x.\exists y.xRy$
R_T	zwrotność (*reflexivity*)	$\forall x.xRx$
R_B	symetryczność (*symmetry*)	$\forall xy.xRy \supset yRx$
R_4	przechodniość (*transitivity*)	$\forall xyz.xRy \wedge yRz \supset xRz$
R_5	euklidesowość (*Euclideanness*)	$\forall xyz.xRy \wedge xRz \supset yRz$
R_2	kierunkowość (*directedness*)	$\forall xyz.xRy \wedge xRz \supset \exists w.yRw \wedge zRw$

Tabela 2.1: Własności relacji widoczności

Nazwa	Formuła
D	$\Diamond\top$
T	$\Box A \supset A$
B	$\Diamond\Box A \supset A$
4	$\Box A \supset \Box\Box A$
5	$\Diamond\Box A \supset \Box A$
2	$\Diamond\Box A \supset \Box\Diamond A$

Tabela 2.2: Aksjomaty

1. $w :: \alpha$ wtw $\alpha \in V(w)$,
2. nie jest możliwe $w :: \bot$,
3. $w :: A \wedge B$ wtw $w :: A$ i $w :: B$,
4. $w :: A \vee B$ wtw $w :: A$ lub $w :: B$,
5. $w :: A \supset B$ wtw $w :: A$ implikuje $w :: B$,
6. $w :: \Box A$ wtw dla każdego świata v jeśli wRv to $v :: A$,
7. $w :: \Diamond A$ wtw istnieje świat v taki, że wRv implikuje $v :: A$,

Relacja widoczności może mieć różne własności. Przedstawia to Tabela 2.1. Własności relacji można równoważnie zdefiniować przy pomocy aksjomatów, które są prawdziwe, gdy relacja spełnia odpowiednią własność. Aksjomaty te przedstawia Tabela 2.2.

Można wyróżnić różne systemy modalne zależnie od tego, jaką wiedzę mamy o relacji widoczności. Najczęściej badane modele to:

- **K** - bez założeń
- **D** - relacja jest szeregowa
- **T** - relacja jest zwrotna
- **B** - relacja jest zwrotna i symetryczna

$$\frac{}{w :: A \vdash_{TD} w :: A}\ (Ass) \qquad \frac{\Gamma \vdash_{TD} w :: \bot}{\Gamma \vdash_{TD} v :: A}\ (\bot E)$$

$$\frac{\Gamma_1 \vdash_{TD} w :: A \qquad \Gamma_1 \vdash_{TD} w :: B}{\Gamma_1; \Gamma_2 \vdash_{TD} w :: A \wedge B}\ (\wedge I) \qquad \frac{\Gamma \vdash_{TD} w :: A \wedge B}{\Gamma \vdash_{TD} w :: A}\ (\wedge E_1) \qquad \frac{\Gamma \vdash_{TD} w :: A \wedge B}{\Gamma \vdash_{TD} w :: B}\ (\wedge E_2)$$

$$\frac{\Gamma \vdash_{TD} w :: A}{\Gamma \vdash_{TD} w :: A \vee B}\ (\vee I_1) \qquad \frac{\Gamma \vdash_{TD} w :: B}{\Gamma \vdash_{TD} w :: A \vee B}\ (\vee I_2)$$

$$\frac{\Gamma_1 \vdash_{TD} w :: A \vee B \qquad \Gamma_2; w :: A \vdash_{TD} v :: C \qquad \Gamma_3; w :: B \vdash_{TD} v :: C}{\Gamma_1; \Gamma_2; \Gamma_3 \vdash_{TD} v :: C}\ (\vee E)$$

$$\frac{\Gamma; w :: A \vdash_{TD} w :: B}{\Gamma \vdash_{TD} w :: A \supset B}\ (\supset I) \qquad \frac{\Gamma_1 \vdash_{TD} w :: A \supset B \qquad \Gamma_2 \vdash_{TD} w :: A}{\Gamma_2; \Gamma_1 \vdash_{TD} w :: B}\ (\supset E)$$

$$\frac{\Gamma; xRy \vdash_{TD} v :: A}{\Gamma \vdash_{TD} w :: \Box A}\ (\Box I)^* \qquad \frac{\Gamma_1 \vdash_{TD} w :: \Box A \qquad \Gamma_2 \vdash_{TD} wRv}{\Gamma_1; \Gamma_2 \vdash_{TD} v :: A}\ (\Box E)$$

$$\frac{\Gamma_1 \vdash_{TD} y :: A \qquad \Gamma_2 \vdash_{TD} wRv}{\Gamma_1; \Gamma_2 \vdash_{TD} w :: \Diamond A}\ (\diamond I) \qquad \frac{\Gamma_1 \vdash_{TD} w :: \Diamond A \qquad \Gamma_2; v :: A; wRv \vdash_{TD} u :: B}{\Gamma_1; \Gamma_2 \vdash_{TD} u :: B}\ (\Diamond E)^{**}$$

* Dodatkowe założenia dla $(\Box I)$: v to świeża zmienna to znaczy $v \neq w$ i v nie występuje w żadnych założeniach w Γ.

** Dodatkowe założenia dla $(\Diamond E)$: v to świeża zmienna.

Ilustracja 2.1: Reguły dedukcji naturalnej dla spójników w podejściu top-down

- **S4** - relacja jest zwrotna i przechodnia

- **S5** - relacja jest zwrotna i euklidesowa

2.3. Reguły dedukcji naturalnej

Istnieją dwa równoważne podejścia definiowania reguł dedukcji naturalnej: top-down i bottom-up. Podejście top-down mocno ogranicza liczbę założeń w środowisku, co daje bardziej zwięzły zapis oraz większą czytelność dowodu. Natomiast podejście bottom-up pozwala na dużą dowolność tego, co znajduje się w założeniach w każdym miejscu drzewa, dzięki czemu tworzenie dowodu w tym podejściu jest znacznie wygodniejsze. Aby skorzystać z zalet obu, PAML przy budowaniu dowodu używa bottop-up, natomiast gotowe dowody generowane są w top-down.

$$\frac{\Gamma; wRv \vdash_{TD} u :: A}{\Gamma \vdash_{TD} u :: A}\ (R_D)^* \qquad \frac{\Gamma; wRw \vdash_{TD} v :: A}{\Gamma \vdash_{TD} v :: A}\ (R_T)$$

$$\frac{\Gamma_1 \vdash_{TD} wRv \qquad \Gamma_2; vRw \vdash_{TD} u :: A}{\Gamma_1; \Gamma_2 \vdash_{TD} u :: A}\ (R_B)$$

$$\frac{\Gamma_1 \vdash_{TD} wRv \qquad \Gamma_2 \vdash_{TD} vRu \qquad \Gamma_3; wRu \vdash_{TD} z :: A}{\Gamma_1; \Gamma_2; \Gamma_3 \vdash_{TD} z :: A}\ (R_4)$$

$$\frac{\Gamma_1 \vdash_{TD} wRv \qquad \Gamma_2 \vdash_{TD} wRu \qquad \Gamma_3; vRu \vdash_{TD} z :: A}{\Gamma_1; \Gamma_2; \Gamma_3 \vdash_{TD} z :: A}\ (R_5)$$

$$\frac{\Gamma_1 \vdash_{TD} wRv \qquad \Gamma_2 \vdash_{TD} wRu \qquad \Gamma_3; vRz; uRz \vdash_{TD} q :: A}{\Gamma_1; \Gamma_2; \Gamma_3 \vdash_{TD} q :: A}\ (R_2)^{**}$$

* Dodatkowe założenia dla (R_D): y jest świeżą zmienną.

** Dodatkowe założenia dla (R_2): w jest świeżą zmienną.

Ilustracja 2.2: Reguły dedukcji naturalnej dla własności relacji widoczności w podejściu top-down

2.3.1. Podejście top-down

Niech Γ oznacza pewien zbiór założeń, a notacja $\Gamma_1; \Gamma_2; x : A$ oznacza $\Gamma_1 \cup \Gamma_2 \cup \{x : A\}$. Reguły top-down wnioskowania dla dostępnych spójników oraz własnoći relacji widoczności przedstawiają Ilustracja 2.1 i Ilustracja 2.2.

W przedstawionym ujęciu Γ jest zbiorem, czyli nie zawiera powtórzeń ani nie rozróżnia kolejności, więc nie potrzebujemy reguły kontrakcji (usuwania powtórzonych założeń), ani zamiany (*exchange*) do zmiany kolejności założeń w Γ. Potrzebujemy jednak reguły osłabiania, która pozwala dołożyć do założeń dowolne nowe. Intuicyjnie, jeżeli twierdzenie zachodziło przy założeniach Γ, to zachodzi dalej przy tych samych założeniach Γ i pewnym dodatkowym. Reguła osłabiania wygląda następująco:

$$\frac{\Gamma \vdash_{TD} x :: A}{\Gamma; y :: B \vdash_{TD} x :: A}\ (W)$$

Przy rozważaniu modeli modalnych zawsze bierzemy wszystkie reguły spójników, regułę osłabiania oraz wszystkie te reguły własności relacji, które wynikają z założeń o relacji dla modelu oraz nic ponad to. Czyli przykładowo model **K** będzie zawierał wszystkie reguły spójników i regułę osłabiania, natomiast model **S4** poza tym co model **K**, będzie używać również reguł (R_T) i (R_4).

$$\frac{}{\Gamma \vdash_{BU} w :: A}\ (Ass) \qquad \frac{\Gamma \vdash_{BU} w :: \bot}{\Gamma \vdash_{BU} v :: A}\ (\bot E)$$

$$\frac{\Gamma \vdash_{BU} w :: A \qquad \Gamma \vdash_{BU} w :: B}{\Gamma \vdash_{BU} w :: A \wedge B}\ (\wedge I) \qquad \frac{\Gamma \vdash_{BU} w :: A \wedge B}{\Gamma \vdash_{BU} w :: A}\ (\wedge E_1) \qquad \frac{\Gamma \vdash_{BU} w :: A \wedge B}{\Gamma \vdash_{BU} w :: B}\ (\wedge E_2)$$

$$\frac{\Gamma \vdash_{BU} w :: A}{\Gamma \vdash_{BU} w :: A \vee B}\ (\vee I_1) \qquad \frac{\Gamma \vdash_{BU} w :: B}{\Gamma \vdash_{BU} w :: A \vee B}\ (\vee I_2)$$

$$\frac{\Gamma \vdash_{BU} w :: A \vee B \qquad \Gamma; w :: A \vdash_{BU} v :: C \qquad \Gamma; w :: B \vdash_{BU} v :: C}{\Gamma \vdash_{BU} v :: C}\ (\vee E)$$

$$\frac{\Gamma; w :: A \vdash_{BU} w :: B}{\Gamma \vdash_{BU} w :: A \supset B}\ (\supset I) \qquad \frac{\Gamma \vdash_{BU} w :: A \supset B \qquad \Gamma \vdash_{BU} w :: A}{\Gamma \vdash_{BU} w :: B}\ (\supset E)$$

$$\frac{\Gamma; wRv \vdash_{BU} v :: A}{\Gamma \vdash_{BU} w :: \Box A}\ (\Box I)^{*} \qquad \frac{\Gamma \vdash_{BU} w :: \Box A \qquad \Gamma \vdash_{BU} wRv}{\Gamma \vdash_{BU} v :: A}\ (\Box E)$$

$$\frac{\Gamma \vdash_{BU} w :: A \qquad \Gamma \vdash_{BU} wRv}{\Gamma \vdash_{BU} w :: \Diamond A}\ (\diamond I) \qquad \frac{\Gamma \vdash_{BU} w :: \Diamond A \qquad \Gamma; v :: A; wRv \vdash_{BU} u :: B}{\Gamma \vdash_{BU} u :: B}\ (\Diamond E)^{**}$$

$$\frac{\Gamma; wRv \vdash_{BU} u :: A}{\Gamma \vdash_{BU} u :: A}\ (R_D)^{***} \qquad \frac{\Gamma; wRw \vdash_{BU} v :: A}{\Gamma \vdash_{BU} v :: A}\ (R_T)$$

$$\frac{\Gamma \vdash_{BU} wRv \qquad \Gamma; vRw \vdash_{BU} u :: A}{\Gamma \vdash_{BU} u :: A}\ (R_B)$$

$$\frac{\Gamma \vdash_{BU} wRv \qquad \Gamma \vdash_{BU} vRu \qquad \Gamma; wRu \vdash_{BU} z :: A}{\Gamma \vdash_{BU} z :: A}\ (R_4)$$

$$\frac{\Gamma \vdash_{BU} wRv \qquad \Gamma \vdash_{BU} wRu \qquad \Gamma; vRu \vdash_{BU} z :: A}{\Gamma \vdash_{BU} z :: A}\ (R_5)$$

$$\frac{\Gamma \vdash_{BU} wRv \qquad \Gamma \vdash_{BU} wRu \qquad \Gamma; vRz; uRz \vdash_{BU} q :: A}{\Gamma \vdash_{BU} q :: A}\ (R_2)^{****}$$

* Dodatkowe założenia dla $(\Box I)$: y to świeża zmienna.

** Dodatkowe założenia dla $(\Diamond E)$: y to świeża zmienna.

*** Dodatkowe założenia dla (R_D): y jest świeżą zmienną.

**** Dodatkowe założenia dla (R_2): w jest świeżą zmienną.

Ilustracja 2.3: Reguły dedukcji naturalnej w systemie bottom-up

2.3.2. Podejście bottom-up

W systemie top-down drzewo dowodu czytamy od liści do korzenia, a wszystkie założenia są tworzone przez regułę *(Ass)* i zamykane na niższych poziomach drzewa. W podejściu bottom-up, drzewo będziemy czytać od korzenia do liści, a kolejne reguły będą dodawać kolejne założenia do wyższych poziomów drzewa. Reguła osłabiania *(W)* nie jest potrzebna. Wszystkie zasady przedstawia Ilustracja 2.3.

2.3.3. Przykład konieczności reguły osłabiania

Powiedzmy, że chcemy udowodnić twierdzenie $x :: A \supset B \supset A$. Twierdzenie to oczywiście zachodzi i w obu systemach istnieją drzewa, które to pokazują. Prostsze jest drzewo bottom-up, które wygląda następująco:

$$\dfrac{\dfrac{\dfrac{}{x :: A; x :: B \vdash_{BU} x :: A}\,(Ass)}{x :: A \vdash_{BU} x :: B \supset A}\,(\supset I)}{\vdash_{BU} x :: A \supset B \supset A}\,(\supset I)$$

W metodzie top-down jednak sytuacja się komplikuje, ponieważ założenie $x :: B$ nie występuje w żadnym liściu drzewa. Trzeba więc je dodać przy pomocy reguły osłabiania:

$$\dfrac{\dfrac{\dfrac{\dfrac{}{x :: A \vdash_{TD} x :: A}\,(Ass)}{x :: A; x :: B \vdash_{TD} x :: A}\,(W)}{x :: A \vdash_{TD} x :: B \supset A}\,(\supset I)}{\vdash_{TD} x :: A \supset B \supset A}\,(\supset I)$$

Przykład ten dobrze obrazuje różnice między oboma systemami.

2.3.4. Równoważność systemów top-down i bottom-up

Zanim przejdziemy do zdefiniowania twierdzenia o równoważności obu systemów i dowodu, najpierw udowodnijmy lemat o systemie bottom-up

Lemat 1 *Dla każdego zbioru założeń* Γ *i* Γ'*, dla każdego świata* x *i formuły* A *jeśli istnieje drzewo dowodu* $\Gamma \vdash_{BU} x :: A$*, to istnieje również drzewo dowodu* $\Gamma; \Gamma' \vdash_{BU} x :: A$*.*

Dowód.
Załóżmy, że istnieje drzewo dowodu $\Gamma \vdash_{BU} x :: A$. Przeprowadzimy indukcję względem struktury tego drzewa. Rozważmy korzeń drzewa:

- $\dfrac{}{\Gamma \vdash_{BU} x :: A}\,(Ass)$. Oznacza to, że $x :: A \in \Gamma$, czyli $x :: A \in \Gamma \cup \Gamma'$ dla dowolnego Γ'. Czyli $\dfrac{}{\Gamma; \Gamma' \vdash_{BU} x :: A}\,(Ass)$ jest poprawnym drzewem dowodu.

- $\dfrac{\Gamma \vdash_{BU} x :: \bot}{\Gamma \vdash_{BU} y :: A}\,(\bot E)$. Weźmy dowolne Γ'. Z założenia indukcyjnego istnieje drzewo wyprowadzenia $\Gamma; \Gamma' \vdash_{BU} x :: \bot$, więc można zbudować drzewo

$$\dfrac{\Gamma; \Gamma' \vdash_{BU} x :: \bot}{\Gamma; \Gamma' \vdash_{BU} y :: A}\,(\bot E)$$

- $\dfrac{\Gamma \vdash_{BU} x :: A \qquad \Gamma \vdash_{BU} x :: B}{\Gamma \vdash_{BU} x :: A \wedge B}\,(\wedge I)$. Weźmy dowolne Γ'. Z założenia indukcyjnego istnieją drzewa dowodu dla przesłanek z dodaniem Γ' to zbioru założeń. Więc można poprawnie zbudować drzewo

$$\dfrac{\Gamma; \Gamma' \vdash_{BU} x :: A \qquad \Gamma; \Gamma' \vdash_{BU} x :: B}{\Gamma; \Gamma' \vdash_{BU} x :: A \wedge B}\,(\wedge I)$$

- Pozostałe przypadki są analogiczne.

□

Twierdzenie 1 *Dla każdego twierdzenia* $\Gamma \vdash_{TD} x :: A$ *istnieje jego drzewo dowodu wtedy i tylko wtedy, gdy dla pewnego* $\Gamma' \supseteq \Gamma$ *istnieje drzewo dowodu twierdzenia* $\Gamma' \vdash_{BU} x :: A$.

Dowód.

⇒ Indukcja względem struktury drzewa wyprowadzenia w regułach top-down.

Załóżmy, że istnieje drzewo wyprowadzenia twierdzenia $\Gamma \vdash_{TD} x :: A$. Rozważmy korzeń drzewa:

(i) $\dfrac{}{\Gamma \vdash_{TD} x :: A}\,(Ass)$. Weźmy dowolne $\Gamma' \supseteq \Gamma$. Ponieważ drzewo $\dfrac{}{\Gamma \vdash_{BU} x :: A}\,(Ass)$ jest poprawne, to korzystając z lematu drzewo $\dfrac{}{\Gamma' \vdash_{BU} x :: A}\,(Ass)$ też.

(ii) $\dfrac{\Gamma \vdash_{TD} x :: A}{\Gamma; y :: B \vdash_{TD} x :: A}\,(W)$. Z założenia indukcyjnego istnieje drzewo $\Gamma' \vdash_{BU} x :: A$ dla pewnego $\Gamma' \supseteq \Gamma$, więc z lematu dla $\Gamma'' = \Gamma' \cup \{y :: B\} \supseteq \Gamma \cup \{y :: B\}$ istnieje drzewo $\Gamma'' \vdash_{BU} x :: A$.

(iii) $\dfrac{\Gamma_1 \vdash_{TD} x :: A \qquad \Gamma_2 \vdash_{TD} x :: B}{\Gamma_1; \Gamma_2 \vdash_{TD} x :: A \wedge B}\,(\wedge I)$. Z założenia indukcyjnego istnieją drzewa dowodu $\Gamma_1' \vdash_{BU} x :: A$ i $\Gamma_2' \vdash_{BU} x :: B$ dla pewnych $\Gamma_1' \supseteq \Gamma_1$ i $\Gamma_2' \supseteq \Gamma_2$. Niech $\Gamma' = \Gamma_1' \cup \Gamma_2'$. Wtedy z lematu istnieją drzewa $\Gamma' \vdash_{BU} x :: A$ i $\Gamma' \vdash_{BU} x :: B$, więc istnieje też drzewo

$$\frac{\Gamma' \vdash_{TD} x :: A \qquad \Gamma' \vdash_{TD} x :: B}{\Gamma' \vdash_{TD} x :: A \wedge B}\ (\wedge I)$$

(iv) Pozostałe przypadki są analogiczne.

$\Leftarrow$ Indukcja względem struktury drzewa wyprowadzenia w regułach bottom-up. Załóżmy, że istnieje drzewo wyprowadzenia twierdzenia $\Gamma \vdash x :: A$. Rozważmy korzeń drzewa:

(i) $\frac{}{\Gamma'; x :: A \vdash_{BU} x :: A}\ (Ass)$, gdzie Γ' jest postaci $\{y :: B, \ldots, z :: C\}$. Niech $\Gamma = \{x :: A\}$. Wtedy drzewo $\frac{}{\Gamma \vdash_{TD} x :: A}\ (Ass)$ jest równoważnym drzewem w regułach top-down.

(ii) $\frac{\Gamma \vdash_{BU} x :: A \qquad \Gamma \vdash_{BU} x :: B}{\Gamma \vdash_{BU} x :: A \wedge B}\ (\wedge I)$. Z założenia indukcyjnego istnieją drzewa $\Gamma_1 \vdash_{TD} x :: A$ i $\Gamma_2 \vdash_{TD} x :: B$, gdzie $\Gamma_1, \Gamma_2 \subseteq \Gamma$. Intuicyjnie Γ_1 i Γ_2 są najmniejszymi zbiorami założeń potrzebnych do zamknięcia dowodu w odpowiednich poddrzewach. Wtedy drzewo

$$\frac{\Gamma_1 \vdash_{TD} x :: A \qquad \Gamma_2 \vdash_{TD} x :: B}{\Gamma_1; \Gamma_2 \vdash_{TD} x :: A \wedge B}\ (\wedge I)$$

jest równoważne względem drzewa w bottom-up.

(iii) Pozostałe przypadki są analogiczne.

□

2.4. Przykłady

Rozważmy następujące przykłady w systemie top-down. Na początku rozważmy system **K**, najbardziej podstawowy, który nie zakłada niczego o relacji R. Przykładowa tautologia w tym modelu, mówiąca o rozdzielności konieczności względem implikacji to $\Box(A \supset B) \supset (\Box A \supset \Box B)$, a dowód tej tautologii przedstawia drzewo na Ilustracji 2.4. Czytając dowód od korzenia, dwa pierwsze przejścia są wprowadzeniem implikacji. Następnie używamy wprowadzenia konieczności, co wymaga stworzenia świeżego świata y. Następnie używamy eliminacji implikacji $y :: A \supset B$. Na koniec, w obu powstałych podrzewach korzystamy z wiedzy, że w x zachodzi $\Box(A \supset B)$ i $\Box A$ oraz z tego, że xRy, co pozwala zamknąć dowód eliminacją konieczności.

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{x :: \Box(A \supset B); \vdash x :: \Box(A \supset B)}\,(Ass) \quad \dfrac{}{xRy; \vdash xRy}\,(Ass)}{x :: \Box(A \supset B); xRy; \vdash y :: A \supset B}\,(\Box E) \quad \dfrac{\dfrac{}{x :: \Box A; \vdash x :: \Box A}\,(Ass) \quad \dfrac{}{xRy; \vdash xRy}\,(Ass)}{x :: \Box A; xRy; \vdash y :: A}\,(\Box E)}{x :: \Box(A \supset B); xRy; x :: \Box A; \vdash y :: B}\,(\supset E)}{x :: \Box(A \supset B); x :: \Box A; \vdash x :: \Box B}\,(\Box I)}{x :: \Box(A \supset B); \vdash x :: \Box A \supset \Box B}\,(\supset I)}{\vdash x :: \Box(A \supset B) \supset \Box A \supset \Box B}\,(\supset I)$$

Ilustracja 2.4: Drzewo dowodu $\Box(A \supset B) \supset (\Box A \supset \Box B)$

$$\dfrac{\dfrac{}{x :: \Diamond(A \vee B); \vdash x :: \Diamond(A \vee B)}\,(Ass) \quad \dfrac{\dfrac{}{y :: A \vee B; \vdash y :: A \vee B}\,(Ass) \quad \dfrac{\dfrac{\dfrac{}{y :: A; \vdash y :: A}\,(Ass) \quad \dfrac{}{xRy; \vdash xRy}\,(Ass)}{y :: A; xRy; \vdash x :: \Diamond A}\,(\Diamond I)}{y :: A; xRy; \vdash x :: \Diamond A \vee \Diamond B}\,(\vee I1) \quad \dfrac{\dfrac{\dfrac{}{y :: B; \vdash y :: B}\,(Ass) \quad \dfrac{}{xRy; \vdash xRy}\,(Ass)}{y :: B; xRy; \vdash x :: \Diamond B}\,(\Diamond I)}{y :: B; xRy; \vdash x :: \Diamond A \vee \Diamond B}\,(\vee I2)}{y :: A \vee B; xRy; \vdash x :: \Diamond A \vee \Diamond B}\,(\vee E)}{x :: \Diamond(A \vee B); \vdash x :: \Diamond A \vee \Diamond B}\,(\Diamond E)}{\vdash x :: \Diamond(A \vee B) \supset \Diamond A \vee \Diamond B}\,(\supset I)$$

Ilustracja 2.5: Drzewo dowodu $\Diamond(A \vee B) \supset (\Diamond A \vee \Diamond B)$

$$\dfrac{\dfrac{\dfrac{}{x :: \Diamond\bot; \vdash x :: \Diamond\bot}\,(Ass) \quad \dfrac{\dfrac{\dfrac{}{y :: \bot; \vdash y :: \bot}\,(Ass)}{y :: \bot; \vdash x :: \bot}\,(\bot E)}{xRy; y :: \bot; \vdash x :: \bot}\,(W)}{x :: \Diamond\bot; \vdash x :: \bot}\,(\Diamond E)}{\vdash x :: \Diamond\bot \supset \bot}\,(\supset I)$$

Ilustracja 2.6: Drzewo dowodu $\neg\Diamond\bot$

Analogicznie w tym systemie zachodzi również rozdzielność możliwości względem implikacji – $\Diamond(A \vee B) \supset (\Diamond A \supset \Diamond B)$, czy zdanie, że niemożliwy jest fałsz – $\neg\Diamond\bot$, a dowody tych twierdzeń przedstawiają odpowiednio rysunki 2.5 i 2.6.

Model **K** przez to, że nie zakłada niczego o relacji, nie pozwala udowodnić wielu twierdzeń, które są podstawą innych modeli. Weźmy na przykład twierdzenie $\Box A \supset A$. Próba dowiedzenia takiego twierdzenia (od korzenia) wygląda następująco:

$$\dfrac{x :: \Box A \vdash x :: A}{\vdash x :: \Box A \supset A}\,(\supset I)$$

W tym miejscu, aby skorzystać z założenia $x :: \Box A$ trzeba również mieć jakiś świat, który jest widoczny z x. Aby dokończyć ten dowód potrzebujemy, aby relacja widoczności R była zwrotna, co pozwala skorzystać z założenia, że xRx, a pełne drzewo wygląda następująco:

$$\dfrac{\dfrac{\dfrac{\dfrac{}{x :: \Box A \supset x :: \Box A}\,(Ass) \quad \dfrac{}{xRx \vdash xRx}\,(Ass)}{x :: \Box A; xRx \vdash x :: A}\,(\Box E)}{x :: \Box A \vdash x :: A}\,(R_T)}{\vdash x :: \Box A \supset A}\,(\supset I)$$

Najczęściej rozważanymi modelami modalnymi są modele **S4** i **S5**. Model **S4** jest definiowany przez powyższą tautologię (**D**, która jest równoważna ze zwrotnością relacji) oraz przez $\Box A \supset \Box\Box A$ (jeśli coś jest konieczne, to jest konieczne, żeby to było konieczne) co wymaga założenia o przechodniości relacji. Drzewo dowodu tego twierdzenia obrazuje rysunek 2.7. **S5** jest definiowany albo przez te aksjomaty co **S4** oraz dodatkowo przez $A \supset \Box\Diamond A$ (jeśli coś zachodzi, to konieczne jest, że jest to możliwe), które wymaga symetryczności relacji, lub przez **D** (zwrotność) i $\Diamond A \supset \Box\Diamond A$ (jeśli coś jest możliwe, to konieczne jest, że jest to możliwe), które wymaga euklidesowości. Dowody obu twierdzeń obrazują odpowiednio rysunki 2.8 i 2.9.

2.5. Zastosowania

Dzięki izomorfizmowi Currego-Howarda zapis drzewa dowodu w logice może być opisem termu w językach programowania, a odpowiadające im formuły stają się typu termów. Przykładowo implikacja staje się typem lambda abstrakcji w typowanym

$$\dfrac{\dfrac{\dfrac{\dfrac{}{xR4y;\vdash xR4y}\,(Ass) \quad \dfrac{}{yR4z;\vdash yR4z}\,(Ass)}{xR4y;yR4z;x :: \Box A;\vdash z :: A}\,(\Box I) \quad \dfrac{\dfrac{}{x :: \Box A;\vdash x :: \Box A}\,(Ass) \quad \dfrac{}{xR4z;\vdash xR4z}\,(Ass)}{x :: \Box A;xR4z;\vdash z :: A}\,(R_4)}{xR4y;x :: \Box A;\vdash y :: \Box A}\,(\Box E)}{x :: \Box A;\vdash x :: \Box\Box A}\,(\Box I)}{\vdash x :: \Box A \supset \Box\Box A}\,(\supset I)$$

Ilustracja 2.7: Drzewo dowodu $\Box A \supset \Box\Box A$

$$\dfrac{\dfrac{\dfrac{}{xRBy;\vdash xRBy}\,(Ass) \quad \dfrac{\dfrac{}{x :: A;\vdash x :: A}\,(Ass)}{x :: A;yRBx;\vdash y :: \Diamond A}\,(R_B)}{xRBy;x :: A;\vdash y :: \Diamond A}\,(\Box I)}{x :: A;\vdash x :: \Box\Diamond A}\,(\supset I)}{\vdash x :: A \supset \Box\Diamond A}\,(\supset I) \qquad \dfrac{\dfrac{}{yRBx;\vdash yRBx}\,(Ass)}{yRBx;\vdash yRBx}\,(\Diamond I)$$

Ilustracja 2.8: Drzewo dowodu $A \supset \Box\Diamond A$

$$\dfrac{\dfrac{\dfrac{\dfrac{}{x :: \Diamond A;\vdash x :: \Diamond A}\,(Ass)}{x :: \Diamond A;xR5y;\vdash y :: \Diamond A}\,(\Box I)}{x :: \Diamond A;\vdash x :: \Box\Diamond A}\,(\supset I)}{\vdash x :: \Diamond A \supset \Box\Diamond A}\,(\supset I) \qquad \dfrac{\dfrac{\dfrac{}{xR5y;\vdash xR5y}\,(Ass) \quad \dfrac{}{xR5z;\vdash xR5z}\,(Ass)}{xR5y;xR5z;z :: A;\vdash y :: \Diamond A}\,(\Diamond E) \quad \dfrac{\dfrac{}{z :: A;\vdash z :: A}\,(Ass)}{z :: A;yR5z;\vdash y :: \Diamond A}\,(R_5) \quad \dfrac{\dfrac{}{yR5z;\vdash yR5z}\,(Ass)}{yR5z;\vdash yR5z}\,(\Diamond I)$$

Ilustracja 2.9: Drzewo dowodu $\Diamond A \supset \Box\Diamond A$

rachunku lambda, co zapisujemy $\lambda x : A.x : A \to A$, natomiast koniunkcja opisuje typ pary. W ten sposób skonstruowanie poprawnego drzewa typu dla termu w takim języku staje się dowodem na poprawne typowanie termu. Z takiego mechanizmu korzystają języki funkcyjne takie jak OCaml czy Haskell. Pfenning w [6] opisał język zbudowany w taki sposób w logice modalnej w systemie S4. W tym systemie dzieli on termy ze względu na kolejność ich ewaluacji. Termy ewaluowane w pierwszej kolejności (statyczne), są reprezentowane przez zwykłe typy, natomiast termy ewaluowane w drugiej fazie (dynamiczne), obudowane są w konstruktor **box** i oznaczane typem $\Box$. W ten sposób odpowiednio zarządzając fazami ewaluacji, można zyskać na efektywności obliczeń.

Inną interpretacją typów reprezentowanych przez logiki modalne, tym razem z systemu S5 jest problem języka dla obliczeń rozproszonych [15]. Wyobraźmy sobie pewną liczbę hostów w sieci. Każdy ma dostęp wyłącznie do swoich zasobów, ale jest połączony z pewną liczbą innych hostów. Wtedy światy stają się interpretacją poszczególnych hostów a zapis $w :: A$ oznacza, że host w ma zasób A. Typ $\Diamond$ oznacza, że posiadamy adres do zasobu, czyli hosta, który zasób posiada, ale możemy go odpakować wyłącznie na wskazanym hoście. W takiej rozproszonej sieci chcemy również oznaczać obliczenia, które mogą być wykonane przez dowolnego hosta w sieci. W tym celu użyjemy $\Box$, a takie obliczenia nazywamy mobilnymi.

Rozdział 3.

Interaktywne dowodzenie

3.1. Historia, istniejące asystenty dowodzenia twierdzeń

W [7] dowiadujemy się, że pierwsze pomysły, żeby wykorzystać powstające w tym czasie komputery do dowodzenia twierdzeń, sięgają lat 50. i 60. XX wieku. Interaktywne dowodzenie twierdzeń to współpraca użytkownika i komputera, dążąca do stworzenia formalnego dowodu. Jednym z pierwszych większych projektów skupionych na tym zagadnieniu był SAM (Semi-Automated Mathematics), w ramach którego powstało kilka programów, pozwalających na interaktywne dowodzenie twierdzeń. W latach 80. powstały dwa skrajne podejścia do interaktywnego dowodzenia twierdzeń. Pierwsze skupiało się na tym, że większość pracy ma być po stronie komputera, a użytkownik zaledwie opisuje problem do rozwiązania. Drugie z kolei ograniczało rolę komputera do zwykłego sprawdzania poprawności kolejnych kroków dowodu, które wykonuje użytkownik. W mojej pracy skupiłem się na tym drugim podejściu.

Pierwszym programem, który korzystał z izomorfizmu Currego-Howorda w reprezentacji twierdzeń był *Automath* z 1967 roku [3]. Jednak najważniejszym z punktu widzenia tej pracy asystentem dowodzenia twierdzeń był stworzony w 1989 roku *Coq* [1]. Został on napisany pierwotnie w języku CAML, współcześnie w OCamlu. Użytkownik definiuje problemy w języku Galina podobnym do OCamla czy Haskella, a następnie przy pomocy komend zwanych taktykami, dowodzi twierdzenia. Taktyki mogą posiadać różny stopień skomplikowania: od prostych (jak **split** do rozbijania celów na mniejsze części), po skomplikowane z dużym stopniem automatyzacji (jak **lia**, która potrafi automatycznie dowieść twierdzenia w arytmetyce liniowej liczb całkowitych (*linear integer arithmetic*)). Coq pozwala również definiować własne taktyki przy pomocy komendy **Ltac** w specjalnym do tego języku, a w nowszych wersjach również w OCamlu. Ponadto dzięki możliwości ekstrahowania dowodów do programów (dzięki izomorfizmowi Currego-Howarda) pozwala tworzyć formalnie udowodnione biblioteki do OCamla czy Haskella.

3.2. Interface

3.2.1. Wprowadzenie

Celem PAMLa jest pomóc użytkownikowi zbudować poprawne drzewo dowodu, przy użyciu czytelnego interfejsu oraz kontrolując na bieżąco poprawność dowodu. Proces budowania drzewa odbywa się iteracyjnie w systemie bottom-up. Użytkownik zaczyna od korzenia, czyli pełnego twierdzenia do udowodnienia. Miejsce w nieukończonym drzewie dowodu, które wymaga ukończenia dowodu nazywamy **celem**. W trakcie budowy drzewa może powstać wiele celów w różnych miejscach drzewa. Użytkownik może dowolnie przełączać się między nimi. Aby lepiej zobrazować sposób pracy i budowanie drzewa rozważmy następujący przykład: celem jest udowodnić twierdzenie $\Diamond(A \vee B) \supset \Diamond A \vee \Diamond B$ (twierdzenie nie wymaga żadnych dodatkowych założeń o relacji). Użytkownik zaczyna od drzewa z jednym celem w korzeniu:

$$\begin{array}{c} \text{Cel} \\ \vdash x :: \Diamond(A \vee B) \supset \Diamond A \vee \Diamond B \end{array}$$

Następnie używa wprowadzenia implikacji i eliminacji możliwości, korzystając z dostępnego założenia. Takie drzewo wygląda następująco:

$$\dfrac{\dfrac{\dfrac{}{x :: \Diamond(A \vee B) \vdash x :: \Diamond(A \vee B)}\,(Ass) \qquad \begin{array}{c}\text{Cel}\\ y :: A \vee B; xRy \vdash x :: \Diamond A \vee \Diamond B\end{array}}{x :: \Diamond(A \vee B) \vdash x :: \Diamond A \vee \Diamond B}\,(\supset E)}{\vdash x :: \Diamond(A \vee B) \supset \Diamond A \vee \Diamond B}\,(\supset I)$$

Następnie użytkownik używa eliminacji alternatywy $y :: A \vee B$ z założenia, co powoduje utworzenie dwóch nowych celów:

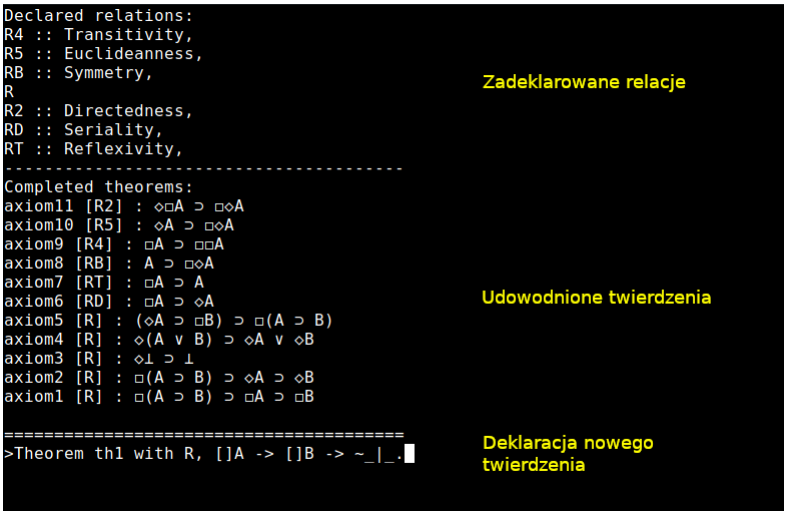
$$\dfrac{\dfrac{\dfrac{}{x :: \Diamond(A \vee B) \vdash x :: \Diamond(A \vee B)}\,(Ass) \qquad \dfrac{\dfrac{}{y :: A \vee B \vdash y :: A \vee B}\,(Ass) \qquad \begin{array}{c}\text{Nowy cel}\\ y :: A; xRy \vdash x :: \Diamond A \vee \Diamond B\end{array} \qquad \begin{array}{c}\text{Nowy cel}\\ y :: B; xRy \vdash x :: \Diamond A \vee \Diamond B\end{array}}{y :: A \vee B; xRy \vdash x :: \Diamond A \vee \Diamond B}\,(\vee E)}{x :: \Diamond(A \vee B) \vdash x :: \Diamond A \vee \Diamond B}\,(\supset E)}{\vdash x :: \Diamond(A \vee B) \supset \Diamond A \vee \Diamond B}\,(\supset I)$$

Gdy użytkownik zamknie wszystkie cele w drzewie, drzewo takie staje się poprawnym dowodem twierdzenia. Program weryfikuje je, przepisuje do systemu top-down i dodaje do środowiska. Skończonego drzewa można następnie użyć w innych dowodach lub poprosić program o przepisanie go do latexa.

3.2.2. Tryby działania PAMLa

Program ma trzy podstawowe stany, w których może się znajdować. Po uruchomieniu użytkownik znajdzie się w panelu zadeklarowanych relacji i udowodnionych

twierdzeń. Z tego panelu użytkownik może stworzyć nową relację, podając jej własności ręcznie lub stworzyć relację dla konkretnego modelu modalnego z rozdziału 2.2.2. Gdy istnieje już jakaś relacja widoczności, można przejść do dowodzenia twierdzeń.

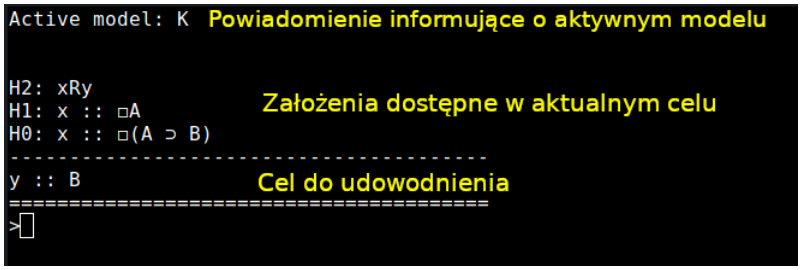


Ilustracja 3.1: Przykładowy widok panelu deklarowania

Po wpisaniu nowego twierdzenia do udowodnienia, program wchodzi w tryb dowodzenia i rozpoczyna budowanie drzewa dowodu. W trakcie dowodzenia mamy dostępne dwa stany: skupiony (*focused*) na konkretnym celu oraz nieskupiony (*unfocused*). Cel to miejsce w drzewie, które wciąż wymaga udowodnienia. W stanie nieskupionym użytkownik może zobaczyć wszystkie pozostałe cele i przejść do jednego z nich.

Ilustracja 3.2: Przykładowy tryb nieskupiony z trzema celami

W stanie skupionym użytkownik widzi wszystkie dostępne w tym punkcie drzewa założenia i ma możliwość używania taktyk, które implementują reguły bottom-up z sekcji 2.2. Gdy użytkownik zakończy budowanie dowodu używa, polecenia **Qed**. Następnie program przejdzie przez tymczasowe drzewo zbudowane przy pomocy taktyk i zbuduje pełny dowód przy pomocy zasad top-down z rozdziału 2.2.



Ilustracja 3.3: Przykładowy tryb skupiony

3.2.3. Taktyki

Taktyki są podstawowym narzędziem do dowodzenia twierdzeń. Operują one regułami bottom-up i pozwalają budować drzewo dowodu. Wszystkie taktyki będą próbowały automatycznie zamknąć przesłanki o widoczności światów (np. xRy), ponieważ takie cele można udowodnić jedynie zasadą (Ass). Jeżeli nie będzie to możliwe, taktyka zwróci błąd wykoniania. Jeżeli taktyka używa reguły, która wprowadza nowe założenie do środowiska, użytkownik może podać nazwę tego założenia (lub założeń) lub pozwolić programowi na wygenerowanie nazw automatycznie. Podobnie, jeżeli reguła wymaga stworzenia świeżego świata. Dostępne są następujące taktyki:

- **Intro** – taktyka pozwalająca użyć reguł wprowadzania $(\supset I), (\Box I)$ i $(\Diamond I)$. Wprowadzenie możliwości wymaga zmiany świata na jakiś już istniejący i w tym przypadku użytkownik musi go podać samodzielnie.

- **Apply** – taktyka ta ma kilka zastosowań. Pierwszą jest użycie zasad eliminacji $(\supset E), (\wedge E), (\vee E), (\Box E)$ i $(\Diamond E)$. Apply potrafi nieco rozszerzyć zastosowanie eliminacji implikacji tak, że dopasowywany będzie sufiks eliminowanej implikacji do celu. Przykładowo, jeżeli w celu mamy $x :: C \supset D$ i użyjemy eliminacji $x :: A \supset B \supset C \supset D$, to powstanie następujące drzewo:

$$\dfrac{\dfrac{\text{Nowy cel}}{x :: A \supset B \supset C \supset D} \qquad \dfrac{\text{Nowy cel}}{x :: A}\,(\supset E)}{x :: B \supset C \supset D} \qquad \dfrac{\text{Nowy cel}}{x :: B}\,(\supset E)}{x :: C \supset D}$$

  Taktyka ta pozwala również używać wspomnianych zasad na założeniach, to znaczy, gdyby w powyższym przykładzie $x :: A \supset B \supset C \supset D$ było założeniem dostępnym w aktualnym kontekście, na którym użyty został apply, efekt byłby analogiczny, przy czym cel (udowodnienie pełnej eliminowanej implikacji) zostałby zamknięty zasadą *(Ass)*. Oznacza to, że jeżeli mamy udowodnić coś, co jest już w założeniach, użycie **Apply** na tym założeniu zamknie dowód. Ostatnim zastosowaniem **Apply** jest użycie wcześniej udowodnionych twierdzeń analogicznie jak założeń. Jeżeli twierdzenie do aplikowania zostało udowodnione dla odpowiedniej relacji (tj. relacji, której zbiór własności zawiera

się w zbiorze własności relacji dla tego twierdzenia), to stanie się to samo, co przy aplikowaniu założenia, przy czym w miejsce reguły *(Ass)* zostanie wstawione drzewo dowodu aplikowanego twierdzenia. Bardzo często przy aplikowaniu twierdzenia wymagana jest zmiana nazw zmiennych w aplikowanym dowodzie. Użytkownik może samodzielnie podać nowe nazwy dla wszystkich lub części zmiennych. Wszystkie zmienne, które będą wymagały nazwy, a nie zostaną zmienione przez użytkownika, program będzie próbował dopasować do nazw w aktualnym kontekście. PAML podmieni też nazwy w aplikowanym drzewie tak, żeby nie doszło do kolizji nazw.
Przykładowo wcześniej udowodniliśmy, że $A \supset \Box\Diamond A$ (posiadamy w środowisku drzewo takie jak w rozdziale 2.4.). Chcielibyśmy go użyć do udowodnienia twierdzenia $y :: B \supset \Box\Diamond B$. W tym celu program musi podmienić początkowy świat x na y, świat pośredni y na jakiś nowy (żeby zapobiec kolizji nazw), oraz dopasować zmienną A do zmiennej B.

- **Contra** – pozwala użyć reguły $(\bot E)$.
- **Split** – pozwala użyć reguły $(\wedge I)$
- **Left** i **Right** – odpowiednie reguły $(\vee E)$.
- *Nazwa własności relacji* – Pozwala użyć reguły związanej z daną własnością relacji widoczności (o ile relacja ma tę własność).
- **Assumption** – przeszukuje lokalne środowisko w poszukiwaniu założenia identycznego z celem i aplikuje je. Jeśli nie ma takiego założenia, zwrócony zostanie błąd.
- **Try** – bierze jako argument inną taktykę i wykonuje ją. Jeżeli taktyka miałaby zwrócić błąd, zamiast tego nic się nie dzieje. Ta taktyka jest przydatna przy kolejkowaniu taktyk.
- **;** – ta taktyka służy do kolejkowania taktyk. Zapis *taktyka1* **;** *taktyka2* wykona taktykę pierwszą, a następnie do wszystkich powstałych celów zastosuje taktykę drugą. Jeśli wykonanie taktyki pierwszej lub któregokolwiek użycia taktyki drugiej wywoła błąd, całe polecenie zwróci błąd i cel pozostanie niezmieniony (nawet jeśli wykonanie taktyki pierwszej się powiodło).

3.2.4. Podpowiedzi, Auto i Fauto

Aby ułatwić użytkowanie PAMLa, można włączyć funkcję podpowiedzi komendą **hints on**. Sprawi to, że w trybie skupionym pod listą założeń pojawią się możliwe do użycia komendy proponowane przez program. Są one generowane na podstawie obecnego celu, listy założeń oraz własności relacji. Podpowiedzi są jednak tylko propozycjami i nie ma gwarancji, że doprowadzą do zamknięcia dowodu. Mimo

to, ponieważ często jest tak, że dla prostszych celów podpowiedzi są w stanie doprowadzić do rozwiązania, można użyć taktyki **Auto**. Auto przyjmuje liczbę możliwych zagłębień n (domyślnie 5) i działa według następującego algorytmu:

1. Jeśli n wynosi 0, zwróć niezmieniony cel.
2. Jeśli dla danego celu można zastosować jakąś regułę wprowadzenia, zastosuj ją i rekurencyjnie wywołaj się na powstałych podcelach z niezmienionym n.
3. Jeśli da się użyć **Apply** z jakimś założeniem, to spróbuj go użyć i sprawdź, czy **Auto** z $n-1$ na powstałych podcelach skończy dowód. Jeśli nie, spróbuj kolejną możliwą aplikację z listy podpowiedzi.
4. Jeśli nie ma już możliwych poleceń do wykonania, zwróć niezmieniony cel.

Obserwacja 1. Auto nigdy nie będzie probowało używać reguł eliminacji, jeżeli da się użyć jakiejś reguły wprowadzania.
Obserwacja 2. Auto nigdy nie będzie próbowało używać reguł generowanych przez własności relacji.

Auto jest zaprojektowane tak, że nawet przy stosunkowo wysokim dozwolonym zagłębieniu zakończy pracę stosunkowo szybko. Może się jednak zdarzyć, że podpowiedzi doprowadzą zamknięcia dowodu, a **Auto** nie znajdzie rozwiązania (niezależnie od zagłębienia). Może się tak zdarzyć mianowicie wtedy, gdy do zamknięcia będzie trzeba użyć reguły eliminacji przed regułą wprowadzenia lub własności relacji. W takim przypadku można użyć taktyki **Fauto** (nazwa pochodzi od *full auto*). Użycie taktyki jest takie samo, jak **Auto**, czyli przyjmuje możliwe zagłębienie (domyślnie 5) i próbuje zakończyć dowód. Jeżeli się to nie uda, zwróci niezmieniony cel. Taktyka **Fauto** w odróżnieniu od **Auto** będzie próbowała użyć wszystkich dostępnych podpowiedzi, przez co liczba obliczeń znacząco rośnie przy większych zagłębieniach względem **Auto**.

Z powodu swojego skomplikowania taktyki **Auto** i **Fauto** mogą czasem długo działać. Jeżeli użytkownik nie będzie chciał czekać na wynik ich działania, może przerwać ich wykonanie sygnałem *SIGINT* (wywoływany skrótem klawiszowym CTRL-C).

Aby zobrazować różnice między oboma taktykami, rozważmy przykład $\Box A \supset \Box\Box A$ z ilustracji 2.7. Aby udowodnić to twierdzenie, należy użyć trzy razy reguł wprowadzania, raz eliminacji formuły z założenia i raz posłużyć się regułą dotyczącą przechodniości relacji widoczności. Wszystkie te taktyki zostaną podpowiedziane przez system podpowiedzi, jednak ponieważ **Auto** nie używa taktyk związanych z własnościami relacji, nie będzie umiało samodzielnie udowodnić tego twierdzenia. **Fauto** z kolei bez problemu odnajdzie dowód tego twierdzenia. Rozważmy jeszcze przykład $\Diamond(A \vee B) \supset (\Diamond A \vee \Diamond B)$ z ilustracji 2.5. W tym przypadku po użyciu wprowadzenia implikacji możliwe jest użycie jednej z zasad wprowadzenia alternatywy, ale

ta droga nie doprowadzi do udowodnienia twierdzenia. Należy w tym miejscu użyć eliminacji na formule z założenia. W tym przypadku **Auto** nie będzie w stanie znaleźć rozwiązania, ponieważ nigdy nie będzie próbowało użyć zasady eliminacji, jeżeli dostępne jest jakieś wprowadzenie. Jednak ponieważ da się udowodnić to twierdzenie taktykami z podpowiedzi, **Fauto** będzie w stanie udowodnić to twierdzenie, choć dowód ten w późniejszych krokach może się nieco różnić od tego przedstawionego na ilustracji.

3.2.5. Obsługa plików

PAML ma możliwość robienia zapisów i wczytywania zapisanego stanu. W dowolnym momencie pracy można utworzyć zapis do pliku komendą **save**. Jeśli użytkownik nie jest w trakcie prowadzenia dowodu można również załadować zapis z pliku komendą **load**. Jeżeli nazwy relacji lub twierdzeń w zapisie w pliku będą kolidowały z tymi znajdującymi się w środowisku PAMLa, użytkownik zostanie zapytany, którą wersję chce zachować.

Jest też możliwość zapisania wszystkich udowodnionych twierdzeń w pliku latexowym. Przykłady z rozdziału 2.4. zostały wygenerowane w ten sposób po uprzednim udowodnieniu w PAMLu i są dostępne w katalogu *examples*.

3.3. Przykładowe użycie

Udowodnijmy przykładowe twierdzenie $\Diamond(A \vee B) \supset \Diamond A \vee \Diamond B$ z rozdziału 3.2.1. Twierdzenie to jest prawdziwe w modelu **K**, więc można wybrać relację o dowolnych własnościach. Zanim przejdziemy do dowodzenia trzeba stworzyć relację. Można to zrobić na dwa sposoby:

```
Relation R.
```

lub

```
Model K.
```

Wynikiem obu zapytań będzie powstanie relacji R bez założeń. W przypadku modelu program będzie ponadto powiadamiał, że użytkownik pracuje w modelu **K** i nie pozwoli na stworzenie nowych relacji bez wcześniejszego wyjścia z modelu.

Następnie deklarujemy nowe twierdzenie komendą

```
Theorem twierdzenie1 with R, x :: <>(A \/ B) -> <>A \/ <>B.
```

Alternatywnie można pominąć jawne wiązanie relacji – w przypadku modelu będzie użyta relacja modelu, a gdy relacja była deklarowana ręcznie, wybrana będzie ostatnio zadeklarowana. Można również pominąć deklarowanie nazwy świata. W takim

przypadku domyślnie pierwszy świat będzie miał nazwę x. Tę samą deklarację można zapisać

```
Theorem twierdzenie1, <>(A \/ B) -> <>A \/ <>B.
```

W tym momencie rozpoczęliśmy pisanie dowodu i znajdujemy się w stanie nieskupionym z jednym celem. Aby skupić się na pierwszym celu, piszemy

```
Proof.
```

lub

```
Focus.
```

Następnie dowodzimy przy użyciu taktyk:

```
intro. (* cel to x :: <>A \/ <>B *)
apply H0 with y. (* Aplikacja założenia x :: <>(A \/ B) *)
apply H2. (* Aplikacja założenia y :: A \/ B *)
```

W tym momencie po eliminacji alternatywy program wrócił do trybu nieskupionego z dwoma celami. Oba cele rozwiążemy pisząc

```
Focus.
left. (* right w drugim przypadku *)
intro with y. (* przejście z x :: <>A do y :: A *)
assumption. (* Mamy założenie y :: A z eliminacji alternatywy *)
```

Alternatywnie można skorzystać z **Auto** i kolejkowania taktyk i napisać w miejscu eliminacji alternatywy

```
apply H2; auto.
```

Warto zaznaczyć, że nie ma pewności, że **Auto** udowodni to tak samo jak my to zrobiliśmy wcześniej. W tym momencie nie ma już otwartych celów, więc kończymy dowód pisząc

```
Qed.
```

W tym momencie program wygenerował drzewo dowodu i wrócił do panelu deklarowania.

Rozdział 4.

Implementacja

4.1. Główne moduły

PAML został zaimplementowany w języku OCaml oraz przy użyciu menhira i ocamllexa, czyli narzędzi służących do implementacji leksera i parsera. Implementacja projektu podzielona jest na kilka modułów, z czego najważniejsze to:

- **Lexer** i **Parser** do przetwarzania komend użytkownika oraz zapisu do pliku.
- **Syntax** zawiera podstawowe definicje budowy formuł logicznych oraz drzew dowodu.
- **Core** – Moduł implementujący każdą regułę top-down jako funkcję. PAML uważa twierdzenie za poprawne tylko jeżeli uda się zbudować drzewo przy użyciu tych funkcji.
- **Proof_syntax** i **Proof** zawierają definicję niezakończonego drzewa dowodu (czyli takiego, w którym są jakieś cele) oraz wszystkie taktyki.
- **Relation** obsługuje relacje. Moduł ten jest osobny od **Syntax**, ponieważ relacje funkcjonują niezależnie od twierdzeń.
- **Commands** to moduł odpowiedzialny za interpretowanie. większości poleceń użytkownika. Odpowiada on za wysyłanie zapytań do **Proof**, zapisuje udowodnione twierdzenia oraz te w trakcie dowodzenia.
- **File_handler** odpowiedzialny jest za tworzenie zapisów, wczytywanie ich oraz tworzenie plików w latexu.

Podczas projektowania programu zapadła decyzja, że w trakcie budowania dowodu używane będą reguły bottom-up. Są one naturalne do tego celu, ponieważ zaczynamy od tego, co jest do udowodnienia, czyli od korzenia i nie wiadomo w kolejnych gałęziach drzewa, które założenia nam się przydadzą, a które są zbędne. Z kolei

drzewo zakończonego dowodu używa reguł top-down ze względu na ich zwięzłość. Zbiór założeń szybko rośnie w trakcie dowodu i wypisywanie wszystkich w każdym węźle drzewa jest niepotrzebne i zajmuje dużo miejsca w wygenerowanych plikach latex.

4.2. Formuły, drzewa dowodu i reguły dowodzenia

Podstawową jednostką logiczną w drzewie jest osąd (*judgement*), który może być albo stwierdzeniem o relacji widoczności między dwoma światami, albo relacją zachodzenia formuły w danym świecie.

```
type judgement = J of world * prop | R of world * world
```

Formuły logiczne przy użyciu izomorfizmu Currego-Howarda przedstawione są jako typ zdefiniowany następująco:

```
type prop =
  | F
  | Var of string
  | Con of prop * prop
  | Alt of prop * prop
  | Imp of prop * prop
  | Box of prop
  | Dia of prop
```

Warto zaznaczyć, że program nie rozpoznaje negacji, równoważności czy $\top$, a przy parsowaniu zmienia formuły z tymi spójnikami na równoważne przy użyciu powyższych spójników. Udowodnione twierdzenia są w postaci drzewa, które również jest zdefiniowany przez typ. Liście drzewa to reguły *(Ass)*, natomiast węzły mogą mieć jedno, dwa lub trzy poddrzewa. Zawierają one również informację o relacji, dla której twierdzenie jest dowodzone, założenia oraz osąd do udowodnienia.

```
type theorem_context = relation_name * assumptions * judgement

type theorem =
  | Assumption of theorem_rule * theorem_context
  | Single of theorem_rule * theorem * theorem_context
  | Double of theorem_rule * theorem * theorem * theorem_context
  | Triple of theorem_rule * theorem * theorem * theorem * theorem_context
```

Warto także zwrócić uwagę, że w każdym węźle drzewo pamięta, jaka reguła została użyta do zbudowania tego węzła (typ *theorem_rule*). Dzięki temu zapisanie rozwiązania do pliku latex jest wygodne i pozwala generować opisy tego, jaka reguła jest

używana w każdym miejscu. Ułatwia to również sprawdzenie poprawności drzewa przy wczytywaniu zapisu.

W pliku nagłówkowym modułu **Core** widać, że funkcje w tym module mają nazwy od kolejnych reguł dedukcji naturalnej. Brakuje jedynie reguły osłabiania *(W)*, która używana jest automatycznie, gdy zachodzi taka potrzeba, a więc kiedy trzeba, aby w przesłankach znalazło się jakieś założenie, którego tam nie ma, ponieważ nie zostało nigdy użyte. Każda funkcja zwraca typ *theorem*, czyli nowe drzewo z odpowiednią zasadą jako korzeniem. Istnieje też funkcja, która pozwala ocenić, czy istniejące drzewo dowodu poprawnie używa reguł z tego modułu. Daje to możliwość sprawdzenia, czy dowody z zapisu nie zostały naruszone i nie pozwala na zaakceptowanie niepoprawnego twierdzenia.

4.3. Taktyki i drzewo celów

W trakcie dowodzenia stopniowo budowane jest drzewo dowodu przy pomocy taktyk. Nie jest to jednak drzewo znane z modułu **Syntax** — to zostanie zbudowane po zakończeniu dowodzenia na podstawie drzewa z modułu **Proof_syntax**. Drzewo to jest podobne do drzewa *theorem*, ale zamiast trzymać *theorem_context* trzyma funkcję, która powinna zostać uruchomiona po skończeniu w tym miejscu drzewa, aby stworzyć poprawne drzewo dowodu. Typ tego drzewa wygląda następująco:

```
type proof =
  | Empty of goal_desc
  | Node1 of proof * (theorem -> theorem)
  | Node2 of proof * proof * (theorem -> theorem -> theorem)
  | Node3 of proof * proof * proof * (theorem -> theorem -> theorem -> theorem)
  | Leaf of theorem
```

gdzie *goal_desc* zawiera relację, listę założeń wraz z ich nazwami i osąd do udowodnienia. Drzewo stworzone przez konstruktor **Empty** jest „dziurą” w drzewie, którą wciąż trzeba udowodnić. Drzewo jest gotowe, gdy nie zawiera żadnych dziur. W trybie skupionym program „wchodzi” w drzewo do jednej z dziur i zapamiętuje drogę, którą przebył w postaci typu *path*:

```
type path =
  | Root
  | Left of path * proof * (theorem -> theorem -> theorem)
  | Right of path * proof * (theorem -> theorem -> theorem)
  | Mid of path * (theorem -> theorem)
  | Left3 of path * proof * proof * (theorem -> theorem -> theorem -> theorem)
  | Mid3 of path * proof * proof * (theorem -> theorem -> theorem -> theorem)
  | Right3 of path * proof * proof * (theorem -> theorem -> theorem -> theorem)
```

W trakcie pracy nad celem program posługuje się więc typem *goal*:

```
type goal = proof * path
```

Pozwala przebywać w dowolnym miejscu drzewa, a dzięki funkcji *unfocus*, odbudować drzewo i wrócić do korzenia. Analogicznie funkcja *focus* pozwala „wejść" do wybranego pustego miejsca drzewa. Tak zdefiniowane drzewo wraz z tymi narzędziami pozwala wygodnie przemieszczać się między celami i budować dowód. Funkcja *qed* przechodzi rekurencyjnie po skończonym drzewie i wywołuje zapisane w węzłach funkcje z modułu **Core** co skutkuje powstaniem drzewa dowodu *theorem*.

Taktyki to funkcje, które biorą potrzebne argumenty i przetwarzają *goal* w nowy *goal*, w większości przypadków utworzony zostanie nowy cel, ale są takie, które zwrócą drzewo bez pustych miejsc, czyli zamkną aktywną część dowodu. W takim przypadku moduł **Commands** ocenia, czy należy użyć funkcji *unfocus*. Taktyki przekazują wszystkie dostępne założenia w górę drzewa do liści implementując reguły bottom-down. Takie podejście jest naturalniejsze dla budowania dowodu od „dołu". Opis każdej z taktyk znajduje się w sekcji dotyczącej UI.

4.4. Rozszerzalność i możliwości dalszego rozwoju

Budowa projektu pozwala z dużą łatwością dodawać nowe taktyki i reguły w odpowiednich modułach. Poruszanie się po drzewie *proof* też nie jest skomplikowane dzięki kilku pomocniczym funkcjom stworzonym do tego celu.

W dalszym rozwoju naturalnym kierunkiem byłoby rozszerzenie narzędzia do logiki pierwszego rzędu. Pozwoliłoby to wprowadzić nowe typy indukcyjne, takie jak liczby naturalne co rozszerzyłoby możliwe zastosowania programu. Kolejnym wzbogaceniem może być dodanie możliwości konwersji logiki modalnej do logiki klasycznej pierwszego rzędu, dzięki czemu można by stosować różne rozwinięte metody dowodzenia logik klasycznych dla logik modalnych, których automatyczne dowodzenie nie jest jeszcze dobrze rozwinięte.

Rozdział 5.

Podsumowanie

PAML został stworzony jako odpowiedź na brak narzędzi do interaktywnego dowodzenia twierdzeń w intuicjonistycznej logice modalnej. Może być użyty zarówno w najbardziej popularnych modelach logiki modalnej, jak i pozwala dobrać relację do swoich potrzeb. Taką wszechstronność osiąga dzięki zaimplementowaniu reguł dedukcji naturalnej, w których jawnie przedstawione jest zachodzenie formuł w świecie oraz dodaniu nowych reguł zależnie od własności relacji widoczności. Wykorzystuje również dwa podejścia do przedstawiania reguł dedukcji naturalnej, dzięki czemu budowanie drzewa jest wygodne, a zarazem zakończony dowód jest zwięzły i czytelny. Do budowania dowodu użytkownik dostał do dyspozycji zestaw taktyk, od tych najprostszych, które odpowiadają regułom dedukcji naturalnej, po bardziej zaawansowane jak **Auto** i **Fauto**, które potrafią wykonać wiele kroków dowodu automatycznie. Dowód jest też zawsze weryfikowany przy użyciu reguł dedukcji naturalnej, dzięki czemu, nawet jeżeli taktyki doprowadzą do błędnie zbudowanego drzewa lub zapis zostanie uszkodzony, nie zostanie ono przyjęte przez program. Innymi atutami PAMLa są możliwe do włączenia podpowiedzi, które podpowiadają jakich taktyk można użyć oraz system zapisu, dzięki któremu możliwe jest odtworzenie wcześniej zbudowanych twierdzeń.

Pierwotnie PAML miał być jedynie dla modelu **S4**, ale wraz z rozwojem doszliśmy do wniosku, że rozszerzenie go na dowolny model nieznacznie tylko komplikuje implementację, a jest znacznie atrakcyjniejsze z punktu widzenia użytkownika. Interfesj użytkownika od początku miał opierać się na tym z Coqa, choć postawiliśmy na formę komend w stylu basha. Dużą pomocą w implementacji całego projektu były przykładowe programy z podręcznika *Types and Programming Languages* [5] oraz implementacja języka *et* [4], głównie dzięki swojej zwięzłości i czytelności. Najtrudniejszą częścią całego programu było stworzenie taktyk **Auto** i **Apply**. Pierwsza implementuje uproszczonego DFSa, a trudność stanowiło głównie poprawne odbudowanie drzewa w wywołaniach rekurencyjnych, gdzie łatwo było popełnić błąd, jego wykrycie natomiast często wymagało wielu testów. Apply z drugiej strony jest taktyką, która robi różne rzeczy zależnie od tego, co jest w celu i co jest argumen-

tem taktyki. Najbardziej zaawansowaną z funkcji było użycie na wcześniej udowodnionych twierdzeń. Wymaga to odpowiedniego dostosowania nazw zmiennych do obecnego celu, tak aby nie powstała kolizja nazw i tak, żeby odpowiednie zmienne przyjęły odpowiednie lokalne nazwy. Choć nie ma pewności, że taktyki te działają bezbłędnie, to jednak dotychczasowe testy nie wykazały błędów i wierzymy, że weryfikacja dowodu odrzuci niepoprawny dowód.

PAML jest programem skończonym według swojego pierwotnego założenia, co nie oznacza, że nie ma pola do rozwoju. Dwie najbardziej naturalne drogi rozwoju to rozszerzenie go do logiki pierwszego rzędu albo dodanie tłumaczenia do logiki klasycznej. Można również rozwinąć isteniejące taktyki, na przykład dodać możliwość używania **Apply**, czy **Split** na założeniach, nie tylko na celu. Można by również, podobnie jak robi to Coq, dać użytkownikowi możliwość tworzenia własnych taktyk, tak żeby nie musiał ingerować w kod źródłowy programu. Z kolei przeniesienie interfesju PAMLa z formy tekstowej do osobnego okienka mogłoby poprawić czytelność i ułatwić użytkowanie programu.

Bibliografia

[1] About coq. URL: https://coq.inria.fr/about-coq.

[2] Advances in modal logic. URL: http://www.cs.man.ac.uk/~schmidt/tools/.

[3] Automath archive. URL: https://www.win.tue.nl/automath/.

[4] et-lang. URL: https://github.com/mietek/et-lang.

[5] Resourses for types and programming languages:, url=.

[6] Rowan Davies and Frank Pfenning. A modal analysis of staged computation. *Journal of the ACM*, 48(3):555–604, May 2001.

[7] John Harrison, Josef Urban, and Freek Wiedijk. History of interactive theorem proving. In J{órg Siekmann, editor, *Handbook of the History of Logic vol. 9 (Computational Logic)*, pages 135–214. Elsevier, 2014.

[8] Mikołaj Korobczak. Paml - proof assistant for modal logic, 2022. URL: https://github.com/mikikora/modalProover.

[9] Agata Murawska. Intuitionistic modal logic is5. formalizations, interpretation, analysis. Master's thesis, University of Wroclaw, Institute of Computer Science, 2013.

[10] Frank Pfenning and Rowan Davies. A judgmental reconstruction of modal logic. *Mathematical Structures in Computer Science*, 11:511–540, 2001. Notes to an invited talk at the *Workshop on Intuitionistic Modal Logics and Applications* (IMLA'99), Trento, Italy, July 1999.

[11] Alex K. Simpson. *The Proof Theory and Semantics of Intuitionistic Modal Logic*. PhD thesis, University of Edinburgh, 1994.

[12] Matthieu Sozeau, Simon Boulier, Yannick Forster, Nicolas Tabareau, and Théo Winterhalter. Coq Coq Correct! Verification of Type Checking and Erasure for Coq, in Coq. *Proceedings of the ACM on Programming Languages*, pages 1–28, January 2020. URL: https://hal.archives-ouvertes.fr/hal-02380196, doi:10.1145/3371076.

[13] The Coq Development Team. *The Coq Proof Assistant Reference Manual, Version 8.4pl4*. INRIA, 2014. URL: `http://coq.inria.fr/distrib/current/refman/`.

[14] The Coq Development Team. The Coq proof assistant, v. 8.7, 2018. URL: `https://github.com/coq/coq`.

[15] Tom Murphy VII, Karl Crary, Robert Harper, and Frank Pfenning. A symmetric modal lambda calculus for distributed computing. In H. Ganzinger, editor, *Proceedings of the 19th Annual Symposium on Logic in Computer Science (LICS'04)*, pages 286–295, Turku, Finland, July 2004. IEEE Computer Society Press. Extended version available as Technical Report CMU-CS-04-105.

[16] Richard Zach. Boxes and diamonds an open introduction to modal logic, 2019.