Interaktywne dowodzenie twierdzeń w logikach modalnych

(Interactive proving of theses in modal logics)

Mikołaj Korobczak

Praca inżynierska

Promotor: dr Małgorzata Biernacka

Uniwersytet Wrocławski
Wydział Matematyki i Informatyki
Instytut Informatyki

24 lipca 2022

Streszczenie

. . .

. . .

Spis treści

Rozdział 1.

Motywacja, zakres pracy

Podstawowe idee logik modalnych istnieją w myśli filozoficznej już od dawna, ale ich formalna definicja i badanie zaczęły się dopiero w XX wieku. Nie jest to jednak popularna dziedzina badań, a co za tym idzie, dotychczas powstało niewiele narzędzi pozwalających badać własności tych logik przy pomocy komputerów. Strona konferencji *Advances in Modal logic* zbiera takie narzędzia [1]. Oprócz różnych, które wizualizują logiki modalne, lub tłumaczą do innych systemów logicznych, powstało też wiele proverów, które różnymi metodami pozwalają dowodzić twierdzenia w tych logikach. Większość opiera się jednak na metodach znanych z badania logik klasycznych (np. implementując algorytm tableau). Inne podejście (implementowane np. przez system Coq) opiera się na intuicjonistycznym badaniu logiki przy użyciu reguł dedukcji naturalnej. W tej pracy opiszę provera, który używając takiego podejścia, implementuje interaktywne dowodzenie twierdzeń dla logik modalnych.

Rozdział 2.

Wprowadzenie do logik modalnych

2.1. Podstawy logiki modalnej

Logiki modalne rozszerzają logiki klasyczne o dwa nowe, unarne spójniki $\Box$ oraz $\diamond$. Intuicyjnie będziemy o nich myśleć jako o odpowiednio konieczności (*necessity*) oraz możliwości (*posibility*). 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 Kripkiego), 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 trzyma 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 Kirpkiego.

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 semantyce Kirpkiego dodajemy jeszcze reguły dostarczane przez własności relacji widoczności.

2.2. Formalna definicja konstruktywnej logiki modalnej

2.2.1. Formuły

Formułę w języku logiki modalnej 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

1. $w :: \alpha$ wtw $\alpha \in V(w)$,
2. nie jest możliwe $w :: \bot$,

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*)	$\forall xyz.xRy \wedge xRz \supset \exists w.yRw \wedge zRw$

Rysunek 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	$\diamond\Box A \supset \Box\diamond A$

Rysunek 2.2: Aksjomaty

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 relacje. Przedstawia to tabelka 2.1. Oznaczenia relacji w tej tabeli wiążą się z nazwami aksjomatów logik modalnych, które stają się prawdziwe dopiero, gdy relacja ma odpowiednią własność. Aksjomaty te widać w tabeli 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 (*serial*)
- **T** - relacja jest zwrotna
- **B** - relacja jest zwrotna i symetryczna
- **S4** - relacja jest zwrotna i przechodnia
- **S5** - relacja jest zwrotna i euklidesowa (*Euclidean*)

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

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

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

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

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

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

$$\frac{\Gamma_1 \vdash_{TD} x :: A \qquad \Gamma_2 \vdash_{TD} xRy}{\Gamma_1; \Gamma_2 \vdash_{TD} x :: \Diamond A}\ (\Diamond I) \qquad \frac{\Gamma_1 \vdash_{TD} x :: \Diamond A \qquad \Gamma_2; y :: A; xRy \vdash_{TD} z :: B}{\Gamma_1; \Gamma_2 \vdash_{TD} z :: B}\ (\Diamond E)^{**}$$

* Dodatkowe założenia dla $(\Box I)$: y to świeża zmienna to znaczy y różny od x i y nie występuje w żadnych założeniach w Γ.

** Dodatkowe założenia dla $(\Diamond E)$: y to świeża zmienna, to znaczy y różny od x i z i y nie występuje w żadnych założeniach w Γ.

Rysunek 2.3: Reguły dedukcji naturalnej dla spójników w podejściu top-down

2.3. Reguły dedukcji naturalnej

Istnieją dwa równoważne podejścia definiowania reguł dedukcji naturalnej: top-down i bottom-up. Zasadniczo różnią się definiowaniem reguły (Ass), gdzie metoda bottom-up wymaga jedynie, aby zbiór założeń zawierał potrzebne założenie, natomiast top-down wymaga dodatkowo, aby nie posiadał on innych założeń.

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 rysunki 2.3 i 2.4.

Ponieważ zbiór założeń Γ jest zbiorem, czyli nie zawiera powtórzeń ani nie rozróżnia kolejności, to nie potrzebujemy reguły kontrakcji (usuwania powtórzonych założeń), ani zamiany (*exchange*) do zmiany kolejności założeń w Γ. Potrzebujemy

$$\frac{\Gamma; xRy \vdash_{TD} z :: A}{\Gamma \vdash_{TD} z :: A}\,(R_D)^* \qquad \frac{\Gamma; xRx \vdash_{TD} y :: A}{\Gamma \vdash_{TD} y :: A}\,(R_T)$$

$$\frac{\Gamma_1 \vdash_{TD} xRy \qquad \Gamma_2; yRx \vdash_{TD} z :: A}{\Gamma_1; \Gamma_2 \vdash_{TD} z :: A}\,(R_B)$$

$$\frac{\Gamma_1 \vdash_{TD} xRy \qquad \Gamma_2 \vdash_{TD} yRz \qquad \Gamma_3; xRz \vdash_{TD} w :: A}{\Gamma_1; \Gamma_2; \Gamma_3 \vdash_{TD} w :: A}\,(R_4)$$

$$\frac{\Gamma_1 \vdash_{TD} xRy \qquad \Gamma_2 \vdash_{TD} xRz \qquad \Gamma_3; yRz \vdash_{TD} w :: A}{\Gamma_1; \Gamma_2; \Gamma_3 \vdash_{TD} w :: A}\,(R_5)$$

$$\frac{\Gamma_1 \vdash_{TD} xRy \qquad \Gamma_2 \vdash_{TD} xRz \qquad \Gamma_3; yRw; zRw \vdash_{TD} v :: A}{\Gamma_1; \Gamma_2; \Gamma_3 \vdash_{TD} v :: A}\,(R_2)^{**}$$

* Dodatkowe założenia dla (R_D): y jest świeżą zmienną, to znaczy y różny od x i z i y nie występuje w żadnych założeniach w Γ.

** Dodatkowe założenia dla (R_2): w jest świeżą zmienną, to znaczy w różny od x, y, z, v i w nie występuje w żadnych założeniach w Γ.

Rysunek 2.4: Reguły dedukcji naturalnej dla własności relacji widoczności w podejściu top-down

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).

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 rysunek 2.5.

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

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

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

$$\frac{\Gamma_1 \vdash_{BU} x :: A \vee B \qquad \Gamma_2; x :: A \vdash_{BU} y :: C \qquad \Gamma_3; x :: B \vdash_{BU} y :: C}{\Gamma_1; \Gamma_2; \Gamma_3 \vdash_{BU} y :: C}\ (\vee E)$$

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

$$\frac{\Gamma; xRy \vdash_{BU} y :: A}{\Gamma \vdash_{BU} x :: \Box A}\ (\Box I)^{*} \qquad \frac{\Gamma_1 \vdash_{BU} x :: \Box A \qquad \Gamma_2 \vdash_{BU} xRy}{\Gamma_1; \Gamma_2 \vdash_{BU} y :: A}\ (\Box E)$$

$$\frac{\Gamma_1 \vdash_{BU} x :: A \qquad \Gamma_2 \vdash_{BU} xRy}{\Gamma_1; \Gamma_2 \vdash_{BU} x :: \Diamond A}\ (\Diamond I) \qquad \frac{\Gamma_1 \vdash_{BU} x :: \Diamond A \qquad \Gamma_2; y :: A; xRy \vdash_{BU} z :: B}{\Gamma_1; \Gamma_2 \vdash_{BU} z :: B}\ (\Diamond E)^{**}$$

$$\frac{\Gamma; xRy \vdash_{BU} z :: A}{\Gamma \vdash_{BU} z :: A}\ (R_D)^{***} \qquad \frac{\Gamma; xRx \vdash_{BU} y :: A}{\Gamma \vdash_{BU} y :: A}\ (R_T)$$

$$\frac{\Gamma_1 \vdash_{BU} xRy \qquad \Gamma_2; yRx \vdash_{BU} z :: A}{\Gamma_1; \Gamma_2 \vdash_{BU} z :: A}\ (R_B)$$

$$\frac{\Gamma_1 \vdash_{BU} xRy \qquad \Gamma_2 \vdash_{BU} yRz \qquad \Gamma_3; xRz \vdash_{BU} w :: A}{\Gamma_1; \Gamma_2; \Gamma_3 \vdash_{BU} w :: A}\ (R_4)$$

$$\frac{\Gamma_1 \vdash_{BU} xRy \qquad \Gamma_2 \vdash_{BU} xRz \qquad \Gamma_3; yRz \vdash_{BU} w :: A}{\Gamma_1; \Gamma_2; \Gamma_3 \vdash_{BU} w :: A}\ (R_5)$$

$$\frac{\Gamma_1 \vdash_{BU} xRy \qquad \Gamma_2 \vdash_{BU} xRz \qquad \Gamma_3; yRw; zRw \vdash_{BU} v :: A}{\Gamma_1; \Gamma_2; \Gamma_3 \vdash_{BU} v :: A}\ (R_2)^{****}$$

* Dodatkowe założenia dla $(\Box I)$: y to świeża zmienna to znaczy y różny od x i y nie występuje w żadnych założeniach w Γ.

** Dodatkowe założenia dla $(\Diamond E)$: y to świeża zmienna, to znaczy y różny od x i z i y nie występuje w żadnych założeniach w Γ.

*** Dodatkowe założenia dla (R_D): y jest świeżą zmienną, to znaczy y różny od x i z i y nie występuje w żadnych założeniach w Γ.

**** Dodatkowe założenia dla (R_2): w jest świeżą zmienną, to znaczy w różny od x, y, z, v i w nie występuje w żadnych założeniach w Γ.

Rysunek 2.5: Reguły dedukcji naturalnej w systemie bottom-up

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

(iv) Pozostałe przypadki są analogiczne.

⇐ Indukcja względem struktury drzewa wyprowadzenia w regułach bottom-up.

Załóżmy, że istnieje drzewo wyprowadzenia twierdzenia $\Gamma \vdash x : A$. Rozważmy korzeń drzewa:

(i) $\dfrac{}{\Gamma; x :: A \vdash_{BU} x :: A}\,(Ass)$, gdzie Γ jest postaci $\{y :: B, \ldots, z :: C\}$. Wtedy można stworzyć nastpujące drzewo w regułach top-down:

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

Rysunek 2.6: Drzewo dowodu $\Box(A \supset B) \supset (\Box A \supset \Box B)$

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

Rysunek 2.7: Drzewo dowodu $\diamond(A \vee B) \supset (\diamond A \supset \diamond B)$

$$\dfrac{\dfrac{\dfrac{}{x :: A \vdash_{TD} x :: A}\,(Ass)}{x :: A; y :: B \vdash_T Dx :: A}\,(W)}{\dfrac{\vdots}{\Gamma \vdash_{TD} x :: A}\,(W)}$$

(ii) $\dfrac{\Gamma \vdash_{BU} x :: A \qquad \Gamma \vdash_{BU} x :: B}{\Gamma \vdash_{BU} x :: A \wedge B}\,(\wedge I)$. Z założenia indukcyjnego istnieją drzewa $\Gamma_1 \vdash_{TD} x :: A$ i $\Gamma_2 \vdash_{TD} x :: B$, gdzie $\Gamma_1, \Gamma_2 \subseteq \Gamma$. Wtedy istnieje drzewo $\dfrac{\Gamma_1 \vdash_{TD} x :: A \qquad \Gamma_2 \vdash_{TD} x :: B}{\Gamma_1; \Gamma_2 \vdash_{TD} x :: A \wedge B}\,(\wedge I)$.

(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 rysunku 2.6.

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 2.7 i 2.8.

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

Rysunek 2.8: Drzewo dowodu $\neg \diamond \bot$

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

Rysunek 2.9: Drzewo dowodu $\Box A \supset \Box\Box A$

Model **K** przez to, że nie zakłada niczego o relacji, nie pozwala udowodnić wielu twierdzeń, które są podstawą innych modeli. Weźmy na przykład twierdzenie, że jeśli A jest konieczne, to zachodzi ($\Box A \supset A$). 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 skożystać 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:

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

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 2.9. **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 2.10 i 2.11.

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

Rysunek 2.10: Drzewo dowodu $A \supset \Box \diamond A$

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

Rysunek 2.11: Drzewo dowodu $\diamond A \supset \Box \diamond A$

2.5. Zastosowania

Dzięki izomorfizmowi Currego-Howarda logiki mogą być opisem termów w językach programowania, a odpowiadające im formuły stają się typem takiego termu. Przykładowo implikacja staje się typem lambda abstrakcji w typowanym 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 [2] 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 [9]. 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 ożyjemy $\Box$, a takie obliczenia nazywamy mobilnymi.

Rozdział 3.

Interaktywne dowodzenie

3.1. Historia, istniejące provery

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, która dąży do stworzenia formalnego dowodu. Jednym z pierwszych większych projektów skupionych na tym zagadnieniu był SAM (Semi-Automated Mathematics), w ramach którego powstało kilka programów, które pozwalały 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. Jednak najważniejszym z punktu widzenia tej pracy proverem był stworzony w 1989 roku *Coq*. Został napisany pierwotnie w języku CAML, dzisiaj w OCamlu. Użytkownik definiuje problemy w języku podobnym do OCamla czy Haskella, a następnie przy pomocy komend zwanych taktykami, dowodzi twierdzenia. Taktyki mogą być proste, służące do jednego celu (jak **split** do rozbijania celów na mniejsze części) po skomplikowane z dużym stopniem automatyzacji (jak **lia**, która potrafi automatycznie dowieźć twierdzenia w arytmetyce liniowej liczb całkowitych (*linear integer arithmetic*)). Pozwala on również definiować własne taktyki przy pomocy komendy **Ltac** w specjalnym do tego języku, a w nowszych wersjach również w OCamlu. W końcu dzięki możliwości ekstraktowania dowodów do programów (dzięki izomorfizmowi Currego-Howarda) pozwala tworzyć formalnie udowodnione biblioteki do OCamla czy Haskella.

prover opisywany w tej pracy ma znacznie mniej możliwości niż Coq, ale jego interface był mocno na nim wzorowany, co widać w układzie graficznym czy nazwach i zastosowaniu taktyk.

3.2. Interface

3.2.1. Wprowadzenie

Celem programu jest pomóc użytkownikowi zbudować poprawne drzewo dowodu, przy użyciu czytelnego interfacu oraz kontrolując na bieżące 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 celi 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). Użytkonik 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 eliminację możliwości korzystając z dostępnego założenia. Takie drzewo wygląda następująco:

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

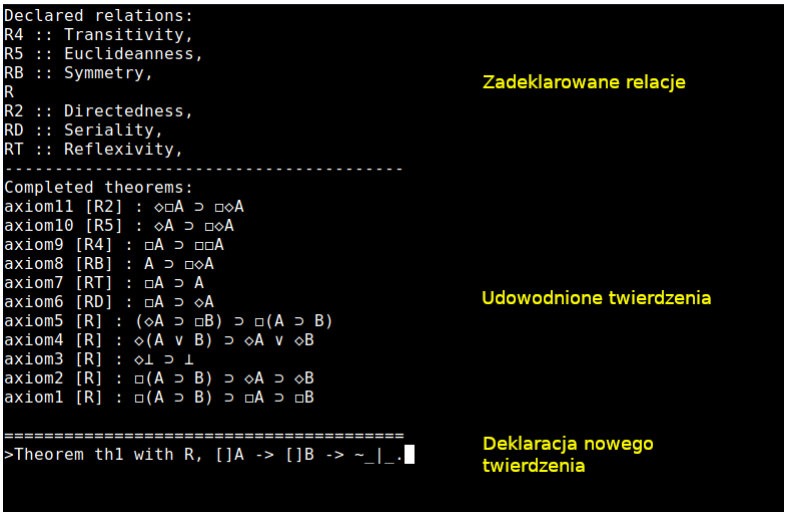
Następnie użytkownik używa eliminacji alternatywy $y :: A \vee B$ z założenia, co powoduje utworzenie dwóch nowych celów:

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

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

3.2.2. Tryby działania provera

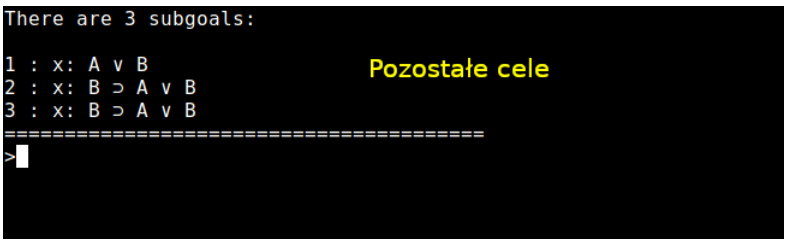
Program ma trzy podstawowe stany, w których może się znajdować. Po uruchomieniu użytkownik znajdzie się w panelu zadeklarowanych relacji i udowodnionych



Rysunek 3.1: Przykładowy widok panelu deklarowania

twierdzeń. Z tego panelu użytkownik może stworzyć nową relacje 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.

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.

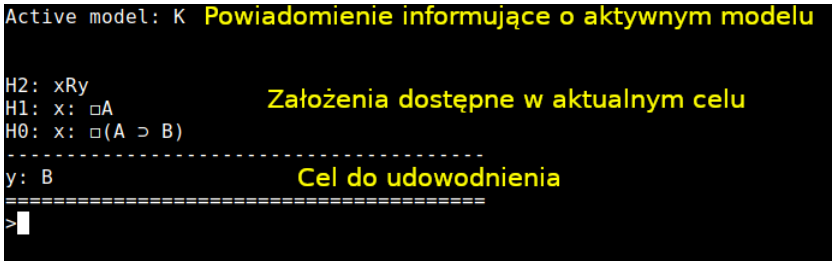


Rysunek 3.2: Przykładowy tryb nieskupiony z trzema celami

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.

3.2.3. Taktyki

Taktyki są podstawowym narzędziem do dowodzenia twierdzeń. Operują one regułami bottom-up i pozwalają budować drzewo dowodu. Wszystkie taktyki będą



Rysunek 3.3: Przykładowy tryb skupiony

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 stworzenie świeżego świata. Dostępne są następujące taktyki:

- **Intro** – taktyka pozwalająca użyć zasad 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 rozszerzyć nieco zastosowanie eliminacji implikacji tak, że dopasowywany będzie sufix 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:

$$\dfrac{\dfrac{\begin{array}{cc}\text{Nowy cel} & \text{Nowy cel}\\ x : A \supset B \supset C \supset D & x : A\end{array}}{x : B \supset C \supset D}\,(\supset E) \qquad \begin{array}{c}\text{Nowy cel}\\ x : B\end{array}}{x : C \supset D}\,(\supset E)$$

  Taktyka ta pozwala również używanie 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, aby udowodnić pełną eliminowaną implikację, 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. Program też podmieni 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 celi 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 powiodła).

3.2.4. Podpowiedzi, Auto i Fauto

Aby ułatwić użytkowanie provera, 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. Ponieważ jednak 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.

Obesrwacja 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, jeżeli do zamknięcia będzie trzeba użyć aplikacji przed wprowadzeniem 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).

3.2.5. Obsługa plików

prover 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 provera, 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 proverze 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

Prover 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 z pliku,
- **Syntax** zawiera podstawowe definicje budowy formuł logicznych oraz drzew dowodu,
- **Core** – Moduł implementujący każdą regułę top-down jako funkcję. Prover uważa twierdzenie za poprawne tylko jeżeli uda się zbudować drzewo przy użyciu tych funkcji,
- **Proof_syntax** i **Proof** zawierają definicję drzewa 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 provera 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 tym 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 tymu spójnikami na równoważne przy użyciu powyższych spójników. Udowodnione twierdzenia trzymane są w postaci drzewa, który również jest zdefiniowany przez typ. Liście drzewa to reguły *(Ass)*, natomiast węzły mogą mieć jedno, dwa lub trzy poddrzewa. Trzymają 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 też sprawdzenie poprawności drzewa przy wczytywaniu zapisu.

W pliku nagłówkowym modułu **Core** widać, że funkcje w tym module nazywają się od kolejnych reguł dedukcji naturalnej. Brakuje jedynie reguły osłabiania *(W)*, która używana automatycznie gdy jest taka potrzeba, a więc kiedy trzeba aby w przesłankach znalazło się jakieś założenie, którego tam nie ma, bo 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. Pozwala to sprawdzać, 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 tego 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 zapamiętuję 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
```

który 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 dodawać nowe taktyki i reguły w odpowiednich modułów z dużą łatwością. 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

...

Bibliografia

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

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

[3] Mikołaj Korobczak. Modal prover, 2022. URL: `https://github.com/mikikora/modalProover`.

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

[5] 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.

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

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

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

[9] 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.

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