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

12 czerwca 2022

Streszczenie

. . .

. . .

Spis treści

Rozdział 1.

Wprowadzenie do logik modalnych

1.1. Podstawy logiki modalnej

Logiki modalne rozszerzają logiki klasyczne o dwa nowe, unarne spójniki $\Box$ oraz . 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 Kirpke zaproponował semantykę logiki modalnej (zwaną semantyką Kirpkiego), 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, że formuła „*Koniecznie* A" w świecie w jest prawdziwa, trzeba pokazać, że 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.

Można wyróżnić rózne 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 przechodnia
- **B** - relacja jest przechodnia i symetryczna
- **S4** - relacja jest zwrotna i przechodnia
- **S5** - relacja jest zwrotna i Euklidesowa (*Euclidean*)

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 skupia się na pełnym sformalizowaniu budowania dowodu. W tym celu korzysta 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, a wniosek nazywamy konsekwencją. W semantyce Kirpkiego dochodzą jeszcze reguły dostarczane przez własności relacji widoczności.

1.2. Formalna definicja konstruktywnej logiki modalnej

1.2.1. 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 $(A \supset B) \wedge (B \supset A)$.

Model modalny to trójka $\mathcal{M} = \langle W, R, V \rangle$, gdzie W to nieepusty 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 zachodzenia (*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 $w : \bot$,
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 $\forall v \in W, wRv \supset v : A$,
7. $w : \diamond A$ wtw $\exists v \in W, wRv \supset v : A$.

Relacja widoczności może mieć następujące własności:

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

1.2.2. Reguły dedukcji naturalnej

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

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 formuł logiki modalnej są następujące:

$$\frac{}{x : A \vdash x : A}\ (Ass) \qquad \frac{\Gamma \vdash x : A}{\Gamma; y : B \vdash x : A}\ (W) \qquad \frac{\Gamma \vdash x : \bot}{\Gamma \vdash y : A}\ (\bot E)$$

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

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

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

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

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

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

* Dodatkowe założenia dla $(\Box I)$: y musi być różny od x i nie może występować w żadnych założeniach w Γ.

** Dodatkowe założenia dla $(\Diamond E)$: y musi być różny od x i z i nie może występować w żadnych założeniach w Γ.

Do zestawu reguł można dodać dodatkowe zależnie od własności relacji widoczności:

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

$$\frac{\Gamma_1 \vdash xRy \qquad \Gamma_2; yRx \vdash z : A}{\Gamma_1; \Gamma_2 \vdash z : A}\ (R_B) \qquad \frac{\Gamma_1 \vdash xRy \qquad \Gamma_2 \vdash yRz \qquad \Gamma_3; xRz \vdash w : A}{\Gamma_1; \Gamma_2; \Gamma_3 \vdash w : A}\ (R_4)$$

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

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

* Dodatkowe założenia dla (R_D): y musi być różny od x i z i nie może występować w żadnych założeniach w Γ.

** Dodatkowe założenia dla (R_2): w musi być różny od x, y, z, v i nie może występować w żadnych założeniach w Γ.

W metodzie 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 dodwać kolejne założenia do wyższych poziomów drzewa. Zasada osłabiania *(W)* nie jest potrzebna, a zasada *(Ass)* wygląda wtedy następpująco:

$$\frac{}{\Gamma; x : A \vdash x : A}\ (Ass)$$

Zasady z więcej niż jedną przesłanką nie rozróżniają osobnych zbiorów Γ, więc przykładowo zasada $(\wedge I)$ wygląda wtedy następująco:

$$\frac{\Gamma \vdash x : A \qquad \Gamma \vdash x : B}{\Gamma \vdash x : A \wedge B}\ (\wedge I)$$

W analogiczny sposób można przepisać pozostałe reguły w tym systemie, a powstałe drzewa dowodu w obu podejściach będą równoważne co udowodnie w dalszych sekcjach.

1.2.3. Przykład konieczności reguły osłabiania

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

$$\cfrac{\cfrac{\cfrac{}{x : A; x : B \vdash x : A}\ (Ass)}{x : A \vdash x : B \supset A}\ (\supset I)}{\vdash x : A \supset B \supset A}\ (\supset I)$$

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

$$\cfrac{\cfrac{\cfrac{\cfrac{}{x : A \vdash x : A}\ (Ass)}{x : A; x : B \vdash x : A}\ (W)}{x : A \vdash x : B \supset A}\ (\supset I)}{\vdash x : A \supset B \supset A}\ (\supset I)$$

Przykład ten dobrze obrazuje różnice między oboma systemami.

1.2.4. Równoważmność metod top-down i bottom-up

Twierdzenie 1 *Dla każdego twierdzenia $\Gamma \vdash x : A$ istnieje jego drzewo dowodu w regułach top-down wtedy i tylko wtedy, gdy dla $\Gamma' \supseteq \Gamma$ istnieje drzewo dowodu twierdzenia $\Gamma' \vdash x : A$ w regułach bottom-up.*

Dowód.

$\Rightarrow$ Indukcja względem struktury drzewa wyprowadznenia w regułach top-down. Załóżmy, że istnieje drzewo wyprowadzenia twierdzenia $\Gamma \vdash x : A$. Rozważmy korzeń drzewa:

(i) $\dfrac{}{x : A \vdash x : A}\,(Ass)$. Takie wyprowadzenie jest już bottom-up.

(ii) $\dfrac{\Gamma \vdash x : A}{\Gamma; y : B \vdash x : A}\,(W)$. Niech $\Gamma' = \Gamma \cup \{y : B\}$. Z założenia indukcyjnego istnieje wyprowadzenie dla $\Gamma' \vdash x : A$ w bottom-up.

(iii) $\dfrac{\Gamma_1 \vdash x : A \qquad \Gamma_2 \vdash x : B}{\Gamma_1; \Gamma_2 \vdash x : A \wedge B}\,(\wedge I)$. Niech $\Gamma' = \Gamma_1 \cup \Gamma_2$. Założenia indukcyjnego istnieją drzewa wyprowadzenia dla $\Gamma' \vdash x : A$ i $\Gamma' \vdash x : B$ w bottom-up, więc istnieje drzewo o korzeniu $\dfrac{\Gamma' \vdash x : A \qquad \Gamma' \vdash x : B}{\Gamma' \vdash x : A \wedge B}\,(\wedge I)$.

(iv) Pozostałe przypadki są analogiczne do powyższyego.

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

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

$$\dfrac{\dfrac{\dfrac{}{x : A \vdash x : A}\,(Ass)}{x : A; y : B \vdash x : A}\,(W)}{\dfrac{\vdots}{\Gamma \vdash x : A}\,(W)}$$

(ii) W pozostałych przypadkach tworzymy potrzebne zbiory założeń $\Gamma_1 = \Gamma_2 = \Gamma_3 = \Gamma$ i korzystając z założenia indukcyjnego oraz odpowiedniej reguły potrafimy zbudować drzewo top-down.

□

1.3. Przykłady

Przykładowe tautologie udowodnione w powyższym systemie:

1. $\Box(A \supset B) \supset (\Box A \supset \Box B)$

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

2. $x : (\Diamond A \supset \Box B) \supset \Box(A \supset B)$

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

oraz przykładowe tautologie w modelu **S5**

3. $\Box A \supset A$

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

4. $\Diamond A \supset \Box \Diamond A$

$$\dfrac{\dfrac{\dfrac{\dfrac{}{x : \Diamond A; \vdash x : \Diamond A}\,(Hyp) \quad \dfrac{\dfrac{}{xRy; \vdash xRy}\,(Hyp) \quad \dfrac{}{xRz; \vdash xRz}\,(Hyp) \quad \dfrac{\dfrac{}{z : A; \vdash z : A}\,(Hyp) \quad \dfrac{}{yRz; \vdash yRz}\,(Hyp)}{z : A; yRz; \vdash y : \Diamond A}\,(\Diamond I)}{xRy; xRz; z : A; \vdash y : \Diamond A}\,(R_5)}{x : \Diamond A; xRy; \vdash y : \Diamond A}\,(\Diamond E)}{x : \Diamond A; \vdash x : \Box \Diamond A}\,(\Box I)}{\vdash x : \Diamond A \supset \Box \Diamond A}\,(\supset I)$$

Rozdział 2.

Interaktywne dowodzenie

2.1. Historia, istniejące provery

...

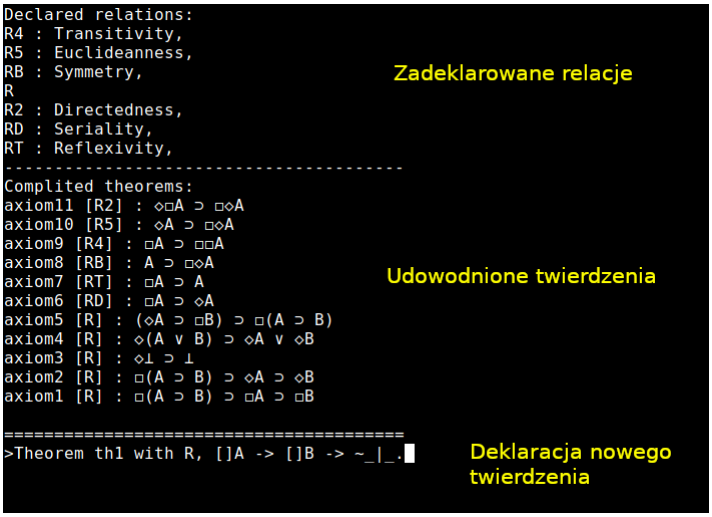
2.2. Interface

2.2.1. Tryby działania provera

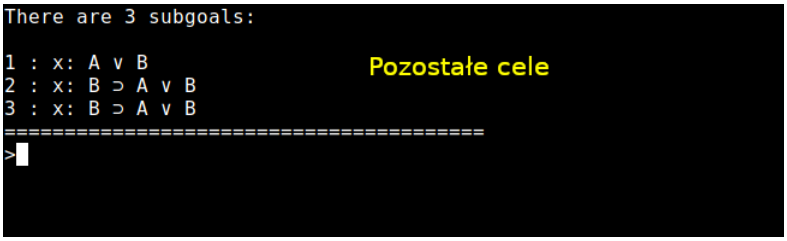
Program ma trzy podstawowe stany, w których może się znajdować. Po uruchomieniu użytkownik znajdzie się w panelu zadeklarowanych relacji i udowodnionych twierdzeń. Z tego panelu użytkownik może zadeklarować nową relację wraz z jej własnościami lub poprosić program o wygenerowanie odpowiedniej dla jednego ze standardowych modeli. 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. 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.

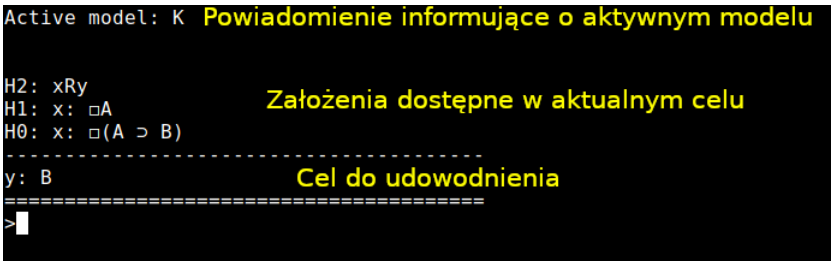
W stanie skupionym użytkownik widzi wszystkie dostępne w tym punkcie drzewa założenia i ma możliwość używania taktyk, które implementują reguły bottom-up z sekcji 1.2.. Gdy użytkownik zakończy budowanie dowodu używa polecenia **Qed**. Następnie program przejdzie przez tymczasowe drzewo zbudowane przy pomocy taktyk i zbuduje pełny dowód przy pomocy zasad top-down z rozdziału 1.2..



Rysunek 2.1: Przykładowy widok panelu deklarowania



Rysunek 2.2: Przykładowy tryb nieskupiony z trzema celami



Rysunek 2.3: Przykładowy tryb skupiony

2.2.2. Taktyki

Taktyki są podstawowym narzędziem do dowodzenia twierdzeń w proverze. 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 zasady, która wprowadza nowe założenie do środowiska, użytkownik może podać nazwę tego założenia (lub założeń) lub pozwolić programowi na wygenerowanie nazw automatycznie. Podobnie jeżeli zasada 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:

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

  Taktyka ta pozwala również używanie wspomnianych zasad na założeniach to znaczy w powyższym przykladzie gdyby $x : A \supset B \supset C \supset D$ było założeniem dostępnym w aktualnym kontekście, na którym wywlołany 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 **Apply** tego założenia 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 zasady *(Ass)* zostanie wstawione drzewo dowodu aplikowanego twierdzenia. Program domyślnie nie będzie próbował dopasowywać nazw zmiennych z aplikowanego twierdzenia do lokalnych nazw, więc użytkownik powinien podać przypisania zmiennych ręcznie (*W planie zmiana tego zachowania*).

- **Contra** – pozwala użyć reguły $(\perp E)$.

- **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.

2.2.3. Podpowiedzi i Auto

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. Są one 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, powstała taktyka **Auto**. Auto przyjmuje liczbę możliwych zagłębień n (domyślnie 5) i działa następującym algorytmem:

1. Jeśli n wynosi 0 zwróć nienaruszony cel.
2. Jeśli dla danego celu można zastosować jakąś regułę wprowadzenia, zastosuj ją i odpal 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 ze środowiska.
4. Jeśli się nie udało zwróć nienaruszony 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.
Uwaga. Auto nie jest traktowana przez program jako pełnoprawna taktyka tj. nie można jej dać jako argument to **Try** lub **;** (*Możliwe, że to się zmini*).

TODO: Rozszerzona wersja Auto

2.2.4. 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 sekcji 1.3. zostały wygenerowane w ten sposób po uprzednim udowodnieniu w proverze. I są dostępne w katalogu *examples*.

Rozdział 3.

Implementacja

3.1. Implementacja

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

- **Lexer** i **Parser** 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 nieskończonego 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 latex.

W trakcie projektowania provera podjąłem decyzję, 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 ktore są zbędne. Natomiast

po zakończeniu dowodu trzworzone drzewo dowodu jest regułach top-down. Wybrałem tę metodę przez jej 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.

3.1.1. Reguły dowodzenia i drzewo dowodu

Udowodnione twierdzenia trzymane są w postaci drzewa, którego liście 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, który może być albo zachodzeniem relacji między światami (xRy), albo relacją zachodzenia ($x : \varphi$).

```
type judgement = J of world * prop | R of world * world
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 zasada jest używana w każdym miejscu. Ułatwia też sprawdzenie poprawności drzewa przy wczytywaniu zapisu.

Zaglądając do pliku nagłówkowego modułu **Core** od razu widać, że funckje 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 założenia, 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 jest poprawne z zaimplementowanymi regułami. Pozwala to sprawdzać czy dowody z zapisu nie zostały naruszone i nie pozwala na zaakceptowanie niepoprawnego twierdzenia.

3.1.2. Taktyki i drzewo celi

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 nie 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 funkcją *unfocus*, odbudować drzewo i wrócić do korzenia. Analogicznie funkcja *focus* pozwala „wejść" do wybranego pustego miejsca drzewa. Tak zdefiniowane drzewo wraz z tyymi narzędziami pozwala wygodnie przemieszczać się miedzy 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 funckje, 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 regułu bottom-down. Takie podejście jest naturalniejsze dla budowania dowodu od „dołu". Opis każdej z taktyk znajduje się w sekcji dotyczącej UI.

3.2. Rozszerzalność

Budowa projektu sprawia, że jeżeli zaszłaby potrzeba dodać jakieś nowe reguły lub taktyki można je łatwo dodać do odpowiednich modułów. Poruszanie się po drzewie *proof* też nie jest skomplikowane dzięki kilku pomocnicznym funkcjom stworzonym do tego celu.

...

3.3. Curry-Howard, proof termy

...

Rozdział 4.

Rozszerzenia

Prover w tym momencie jest dość prostym, ale wygodnym narzędziem do dowodzenia prostych twierdzeń logik modalnych. Naturalnym rozszerzeniem takiego provera byłby dodanie wiązania zmiennych czyli rozszerzenie do logiki pierwszego rzędu. Takie rozszerzenie pozwoliłoby też możliwie dodać do provera nowe typy indukcuyjne takie jak liczby natualne, aby rozszerzyć możliwe twierdzenia, do wnioskowania. Kolejnym możliwym wzbogaceniem mogłaby być konwersja logiki modalnej do logiki klasycznej pierwszego rzędu, dzięki czemu możnaby stosować różne rozwinięte metody dowodzenia logik klasycznych dla logik modalnych, których automatyczne dowodzenie nie jest jeszcze dobrze rozwinięte.

...

Bibliografia

[1] ...