

OBLICZENIA I WNIOSKOWANIE W SYSTEMIE COQ

Małgorzata Biernacka

Wykład 8
23.04.2013

DOBRY STYL DOWODZENIA

- ▶ skrypt dowodu powinien być możliwie zwięzły, czytelny i odporny na zmiany kontekstu
- ▶ należy unikać ciągów pojedynczych taktyk (skrypt jest długi, nieczytelny, trudno modyfikowalny)
- ▶ należy unikać używania nazw zmiennych generowanych automatycznie przez Coqa – nadajemy własne nazwy i/lub wykorzystujemy konstrukcję `match goal ...` języka taktyk
- ▶ dobrze jest automatyzować skrypty pisząc własne taktyki-makra wywołujące często spotykane kombinacje prostszych taktyk i heurystyki używane przy szukaniu dowodów

METODY AUTOMATYZACJI DOWODZENIA

- ▶ używanie wbudowanych taktyk automatycznych (atomowych)
- ▶ używanie operatorów budujących taktyki złożone (*tacticals*)
- ▶ używanie baz twierdzeń (*hint databases*)
- ▶ wykorzystanie języka Ltac do pisania zaawansowanych taktyk implementujących procedury decyzyjne i heurystyki, m. in. z wykorzystaniem nawrotów

SUKCES I PORAŻKA TAKTYKI

- ▶ każda taktyka kończy się sukcesem lub porażką
- ▶ sukces może być całkowity (udowodnienie celu) lub częściowy (wygenerowanie nowych podcelów lub przekształcenie celu)
- ▶ niektóre taktyki nie zmieniają celu (sukces)

TAKTYKI ATOMOWE (WBUDOWANE)

- ▶ bazowe – odpowiadają pojedynczym regułom wnioskowania (np. `intro`)
- ▶ często używane kombinacje innych taktyk (np. `intros`)
- ▶ często używane heurystyki/procedury decyzyjne (np. `auto`, `tauto`)

NIEKTÓRE TAKTYKI AUTOMATYCZNE

- ▶ `auto` – kombinacja `assumption`, `intros`, `apply` + odpowiedzi z baz (zawsze kończy się sukcesem)
- ▶ `eauto` – jak `auto`, tylko używa unifikacji zamiast dopasowania wzorca (używa `eapply` zamiast `apply`)
- ▶ `tauto` – procedura decyzyjna dla intuicjonistycznego rachunku zdań (korzysta z rachunku sekwentów Dyckhoffa)
- ▶ `intuition` – korzysta z drzewa poszukiwania dowodu konstruowanego przez `tauto` + bazy odpowiedzi dla `auto` (zawsze kończy się sukcesem)
- ▶ `firstorder` – częściowe rozszerzenie `tauto` do logiki pierwszego rzędu

NIEKTÓRE TAKTYKI AUTOMATYCZNE

- ▶ `congruence` – procedura decyzyjna dla równości z symbolami nieinterpretowanymi + konstruktorami (implementuje algorytm *congruence closure*)
- ▶ `omega` – procedura decyzyjna dla arytmetyki Presburgera (działa na `nat` i `Z`) (potrzebne `Require Import Omega.`)
- ▶ `ring`, `ring_simplify` – rozwiązuje (upraszcza) równania w pierścieniach

DOWODZENIE PRZY UŻYCIU BAZ TWIERDZEŃ

- ▶ taktyki `auto` i `eauto` korzystają z baz deklarowanych m.in. za pomocą komend:
 - `Hint Resolve t` – dodanie taktyki `apply t` (koszt – liczba podceli)
 - `Hint Immediate` – jak `Hint Resolve`, ale rozwiązywane za pomocą `apply + trivial` (koszt 1)
 - `Hint Constructors T` – dodanie taktyki `apply ci` dla każdego konstruktora typu indukcyjnego `T`
 - `Hint Unfold c` – dodanie taktyki `unfold c` (dla `autounfold`, koszt 4)
 - `Hint Extern` – dodanie dowolnej taktyki o dowolnym koszcie
 - `Hint Transparent/Opaque c.` – tylko dla baz opartych na DT
- ▶ niektóre standardowe bazy:
 - `arith` – zawiera twierdzenia arytmetyczne
 - `zarith` – zawiera twierdzenia arytmetyczne na typie liczb całkowitych w reprezentacji binarnej
 - `datatypes` – zawiera twierdzenia o typach danych, takich jak listy, ciągi
 - `sets` – zawiera twierdzenia o zbiorach i relacjach
- ▶ taktyka `auto` domyślnie używa bazy `core` zawierającej podstawowe twierdzenia o spójnikach logicznych

DOWODZENIE PRZY UŻYCIU BAZ TWIERDZEŃ

- ▶ taktyka autorewrite może korzystać z baz twierdzeń dla przepisywania
- ▶ `Hint Rewrite t : id.` – deklaruje identyfikator `t` w bazie `id` dla taktyki autorewrite
- ▶ `Hint Rewrite <- t : id.` – jw. ale z orientacją przeciwną
- ▶ uwaga na możliwość zapętlenia!

```
Hint Rewrite plus_assoc : my_rewrite_db.
```

```
Lemma assoc:
```

```
forall n m p, (n + m) + p = n + (m + p).
```

```
Proof.
```

```
intros.
```

```
autorewrite with my_rewrite_db.
```

```
trivial.
```

```
Qed.
```

PRZEPISYWANIE TERMÓW – TAKTYKA pattern

- ▶ taktyka rewrite służy do przepisywania podtermów według zadanej równości
- ▶ czasem trzeba przepisać tylko pewne konkretne wystąpienia podtermu, a inne pozostawić bez zmian
- ▶ służy do tego taktyka pattern t at $n_1 \dots n_k$
- ▶ taktyka pattern t przeprowadza β -ekspansję ze względu na t , tj:
 - niech cel będzie formułą $A(t)$
 - pattern t przekształca cel do postaci $(\text{fun } x \Rightarrow A(x))\ t$
 - w przypadku podania numerów wystąpień, tylko te wystąpienia są zastępowane przez zmienną x
 - następne użycie taktyki rewrite do termu t spowoduje przepisanie tylko wybranych wystąpień
- ▶ ogólniejsze przepisywanie: w setoidzie (zbiór z relacją równoważności traktowaną jak równość przez taktyki przepisywania)

TAKTYKI OPARTE NA KONWERSJI

- ▶ `simpl` – wykonuje redukcje w termie
- ▶ `simpl t at  $n_1 \dots n_k$` – wykonuje redukcje podtermu t tylko w wybranych wystąpieniach tego podtermu
- ▶ `change t` – zamienia bieżący cel na term t pod warunkiem, że są one konwertowalne
- ▶ `lazy`, `cbv` – taktyki wykonujące redukcje według odpowiedniej strategii; ich argumentami mogą być `beta`, `zeta`, `delta`, `iota`
- ▶ `autorewrite with b` – wykonuje automatyczne przepisywanie z użyciem twierdzeń ze zbioru b (tworzone za pomocą `Hint Rewrite t`)
- ▶ `subst`, `subst x` – wykonuje podstawienia wszystkich zmiennych x , dla których istnieje równanie $x = t$ lub $t = x$ wśród przesłanek (usuając x całkowicie z kontekstu i celu)

LTAC – BUDOWANIE TAKTYK ZŁOŻONYCH

- ▶ `idtac` – nie zmienia celu, zawsze kończy się sukcesem
- ▶ `fail` – zawsze kończy się porażką
- ▶ `t1; t2` – złożenie (zastosuj `t2` do wszystkich podcelów generowanych przez `t1`)
- ▶ `t0; [t1 | t2 | ... | tn]` – uogólnione złożenie (zastosuj `t1` do podcelu nr `i` generowanego przez `t0`)
- ▶ `t1 || t2` – alternatywa (jeśli `t1` się nie powiedzie, to zastosuj `t2`)
- ▶ `try t` – równoważne `t || idtac`
- ▶ `repeat t` – powtarza `t` dopóki się da; zawsze kończy się sukcesem
- ▶ `first [t1 | ... | tn]` – próbuje kolejne taktyki, aż do napotkania pierwszej, która nie kończy się porażką
- ▶ `solve [t1 | ... | tn]` – j.w., ale wybiera pierwszą taktykę, która kończy się całkowitym sukcesem

PROGRAMOWANIE W Ltac

- ▶ Coq pozwala na tworzenie własnych, złożonych taktyk z taktyk podstawowych oraz na parametryzację takich makr
- ▶ Ltac jest typowany dynamicznie
- ▶ Ltac jest językiem funkcyjnym i imperatywnym
- ▶ `Ltac id arg1 ... argm := expr` definiuje nową taktykę *id* oczekującą *m* argumentów (argument to taktyka lub term)
- ▶ taktyki mogą być rekurencyjne i wzajemnie rekurencyjne
- ▶ istnieje możliwość używania taktyk zewnętrznych
- ▶ w czasie wykonania mogą występować błędy typu i nieterminacja

JĘZYK LTAC – DOPASOWANIE CELU

- ▶ taktyka `elim_absurd` wyszukuje wśród przesłanek pary formuł P i $\sim P$; jeśli takie są, eliminuje fałsz; jeśli nie, zostawia cel bez zmian:

```
Ltac elim_absurd :=  
match goal with  
| [ H :  $\sim ?X$  , H1:  $?X \mid - \_$  ] => elim H; assumption  
| _ => idtac  
end.
```

```
Lemma absurd :  
forall (P Q:Prop), P ->  $\sim P$  -> Q.  
Proof. intros. elim_absurd. Qed.
```

- ▶ zmienne egzystencjalne (zmienne unifikowane) $?X$
- ▶ semantyka konstrukcji `match` – przeszukiwanie z nawrotami

SEMANTYKA KONSTRUKCJI `match`

- ▶ pierwszy sukces kończy wykonanie taktyki
- ▶ w razie porażki taktyki w danej gałęzi, próbuje się inaczej dopasować wzorzec
- ▶ w razie porażki po przeszukaniu wszystkich możliwych dopasowań wzorca dla danej gałęzi, następuje przejście do następnej gałęzi
- ▶ użycie taktyki `fail n` w danej gałęzi powoduje natychmiastową porażkę dla danej gałęzi oraz odcięcie nawrotów w `n` otaczających konstrukcjach `match` (powrót do `n+1`. otaczającego poziomu)
- ▶ zmienne egzystencjalne mogą powtarzać się we wzorcu
- ▶ zmienne egzystencjalne nie mogą unifikować się z termami zawierającymi lokalne zmienne związane