# Programowanie z typami zależnymi i dowodzenie twierdzeń

Taktyki

28.04.2015

# Taktyka *d*estruct as

W rozwiązaniach FunOpt.v pojawiła się następująca sytuacja:

W założeniach było

```
IHe2 : varsType va vt ->
Some tNat = Some tNat ->
exists vv : val, eval va e2 = Some vv ∧ type_val vv = tNat
```

Wtedy `destruct IHe2 as (vv2, ?);` wprowadzi nowe cele dowodowe:

```
varsType va vt
Some tNat = Some tNat
```

i rozszerzy założenia aktualnego celu dowodowego o `vv2:val` i

```
NowaNazwa: eval va e2 = Some vv ∧ type_val vv = tNat
```

# Taktyka *d*estruct as

W rozwiązaniach FunOpt.v pojawiła się następująca sytuacja:

W założeniach było

```
IHe2 : varsType va vt ->
Some tNat = Some tNat ->
exists vv : val, eval va e2 = Some vv ∧ type_val vv = tNat
```

Wtedy `destruct IHe2 as (vv2, ?);` wprowadzi nowe cele dowodowe:

```
varsType va vt
Some tNat = Some tNat
```

i rozszerzy założenia aktualnego celu dowodowego o `vv2:val` i
`NowaNazwa: eval va e2 = Some vv ∧ type_val vv = tNat`

# Taktyka *d*estruct as cd.

```
Goal (exists n, R n) -> R 0 ∨ (exists m, R (S m)).
```

```
intro.
destruct H as ([ | l], W);[left|right];eauto.
```

Nawiasy kwadratowe — "łapanie" wartości pod kolejnymi konstruktorami
Nawiasy okrągłe — "łapanie" kolejnych argumentów pod jednym konstruktorem

# Taktyka *d*estruct as cd.

```
Goal (exists n, R n) -> R 0 ∨ (exists m, R (S m)).
```

```
intro.
destruct H as ([ | l], W);[left|right];eauto.
```

Nawiasy kwadratowe — "łapanie" wartości pod kolejnymi konstruktorami
Nawiasy okrągłe — "łapanie" kolejnych argumentów pod jednym konstruktorem

# Taktyka *d*estruct eqn

`destruct t eqn:?`

- jak destruct ale pozostawia równość pomiędzy termem i jego możliwymi wartościami
- po dwukropku wzór nazwy na tę równość; ? zostawia Coqowi decyzję co do nazwy

# Auto

- **auto** — używa rekurencyjnie assumption, intro i apply do dołożonych założeń i lematów z bazy hintów core pasujących do symbolu głowowego aktualnego celu dowodowego,
- **auto num** — używa auto na głebokośc num. Domyślnie jest 5.
- **auto with ident1 ... identn** — używa auto z bazami hintów ident1, ...., identn,
- **trivial** — próbuje tylko bardzo prostych lematów do zakończenia dowodu.

# Auto

- **auto** — używa rekurencyjnie assumption, intro i apply do dołożonych założeń i lematów z bazy hintów core pasujących do symbolu głowowego aktualnego celu dowodowego,
- **auto num** — używa auto na głebokośc num. Domyślnie jest 5.
- **auto with ident1 ... identn** — używa auto z bazami hintów ident1, ...., identn,
- **trivial** — próbuje tylko bardzo prostych lematów do zakończenia dowodu.

# Auto

- **auto** — używa rekurencyjnie assumption, intro i apply do dołożonych założeń i lematów z bazy hintów core pasujących do symbolu głowowego aktualnego celu dowodowego,
- **auto num** — używa auto na głebokośc num. Domyślnie jest 5.
- **auto with ident1 ... identn** — używa auto z bazami hintów ident1, ...., identn,
- **trivial** — próbuje tylko bardzo prostych lematów do zakończenia dowodu.

# Auto

- **auto** — używa rekurencyjnie assumption, intro i apply do dołożonych założeń i lematów z bazy hintów core pasujących do symbolu głowowego aktualnego celu dowodowego,
- **auto num** — używa auto na głebokośc num. Domyślnie jest 5.
- **auto with ident1 ... identn** — używa auto z bazami hintów ident1, ...., identn,
- **trivial** — próbuje tylko bardzo prostych lematów do zakończenia dowodu.

# Eauto

Taktyka **eauto** to uogólnienie auto (używa eapply, zamiast apply)

```
Coq< Hint Resolve ex_intro.

Coq < Goal forall P:nat -> Prop, P 0 -> exists n, P n.

1 subgoal


============================
forall P0 : nat -> Prop, P0 0 -> exists n : nat, P0 n


Coq < eauto.
No more subgoals.
```

# Eauto

Taktyka **eauto** to uogólnienie auto (używa eapply, zamiast apply)

```
Coq< Hint Resolve ex_intro.

Coq < Goal forall P:nat -> Prop, P 0 -> exists n, P n.

1 subgoal


============================
forall P0 : nat -> Prop, P0 0 -> exists n : nat, P0 n


Coq < eauto.
No more subgoals.
```

# Hint

- lematy używane przez auto i eauto są w bazach hintów,
- bazy ideksowane głowowym symbolem lematu; z tym symbolem związana jest lista hintów,
- każdy hint ma swój koszt; hinty tańsze są próbowane wcześniej,
- auto próbuje hintów dla symbolu głowowego pasującego do głowy aktualnego celu dowodowego lub hintów ogólnych.

```
Create HintDb ident
```

# Hint

- lematy używane przez auto i eauto są w bazach hintów,
- bazy ideksowane głowowym symbolem lematu; z tym symbolem związana jest lista hintów,
- każdy hint ma swój koszt; hinty tańsze są próbowane wcześniej,
- auto próbuje hintów dla symbolu głowowego pasującego do głowy aktualnego celu dowodowego lub hintów ogólnych.

```
Create HintDb ident
```

# Hint

- lematy używane przez auto i eauto są w bazach hintów,
- bazy ideksowane głowowym symbolem lematu; z tym symbolem związana jest lista hintów,
- każdy hint ma swój koszt; hinty tańsze są próbowane wcześniej,
- auto próbuje hintów dla symbolu głowowego pasującego do głowy aktualnego celu dowodowego lub hintów ogólnych.

```
Create HintDb ident
```

# Hint

- lematy używane przez auto i eauto są w bazach hintów,
- bazy ideksowane głowowym symbolem lematu; z tym symbolem związana jest lista hintów,
- każdy hint ma swój koszt; hinty tańsze są próbowane wcześniej,
- auto próbuje hintów dla symbolu głowowego pasującego do głowy aktualnego celu dowodowego lub hintów ogólnych.

```
Create HintDb ident
```

# Hint

- lematy używane przez auto i eauto są w bazach hintów,
- bazy ideksowane głowowym symbolem lematu; z tym symbolem związana jest lista hintów,
- każdy hint ma swój koszt; hinty tańsze są próbowane wcześniej,
- auto próbuje hintów dla symbolu głowowego pasującego do głowy aktualnego celu dowodowego lub hintów ogólnych.

```
Create HintDb ident
```

# Hint

- lematy używane przez auto i eauto są w bazach hintów,
- bazy ideksowane głowowym symbolem lematu; z tym symbolem związana jest lista hintów,
- każdy hint ma swój koszt; hinty tańsze są próbowane wcześniej,
- auto próbuje hintów dla symbolu głowowego pasującego do głowy aktualnego celu dowodowego lub hintów ogólnych.

```
Create HintDb ident
```

# Dodawanie hintów do baz ident1 ... identn

```
Hint hint_def : ident1 ... identn
```

gdzie `hint_def` to

- **Resolve** *t*erm — dodaje apply *t*erm do listy hintów o odpowiednim symbolu głowowym; koszt hintu to liczba wygenerowanych "subgoals". Jeśli "subgoals" mają zmienne wolne to ten hint będzie używany tylko przez eauto.
- **Immediate** *t*erm — dodaje apply *t*erm; trivial do listy hintów o odpowiednim symbolu głowowym. Koszt to 1. Przydatny do używania własności typu symetria równości.
- **Constructor** *i*dent — dodaje wszystkie konstruktory typu indukcyjnego *i*dent jako hinty typu Resolve.
- **Unfold** *i*dent — rozwija *i*dent, jeśli jest w głowie celu dowodowego.
- **Extern** *n*um [pattern] => tactic — ogólna postać

# Dodawanie hintów do baz ident1 ... identn

```
Hint hint_def : ident1 ... identn
```

gdzie `hint_def` to

- **Resolve** *t*erm — dodaje apply *t*erm do listy hintów o odpowiednim symbolu głowowym; koszt hintu to liczba wygenerowanych "subgoals". Jeśli "subgoals" mają zmienne wolne to ten hint będzie używany tylko przez eauto.
- **Immediate** *t*erm — dodaje apply *t*erm; trivial do listy hintów o odpowiednim symbolu głowowym. Koszt to 1. Przydatny do używania własności typu symetria równości.
- **Constructor** *i*dent — dodaje wszystkie konstruktory typu indukcyjnego *i*dent jako hinty typu Resolve.
- **Unfold** *i*dent — rozwija *i*dent, jeśli jest w głowie celu dowodowego.
- **Extern** *n*um [pattern] => tactic — ogólna postać

# Dodawanie hintów do baz ident1 ... identn

```
Hint hint_def : ident1 ... identn
```

gdzie `hint_def` to

- **Resolve** *t*erm — dodaje apply *t*erm do listy hintów o odpowiednim symbolu głowowym; koszt hintu to liczba wygenerowanych "subgoals". Jeśli "subgoals" mają zmienne wolne to ten hint będzie używany tylko przez eauto.
- **Immediate** *t*erm — dodaje apply *t*erm; trivial do listy hintów o odpowiednim symbolu głowowym. Koszt to 1. Przydatny do używania własności typu symetria równości.
- **Constructor** *i*dent — dodaje wszystkie konstruktory typu indukcyjnego *i*dent jako hinty typu Resolve.
- **Unfold** *i*dent — rozwija *i*dent, jeśli jest w głowie celu dowodowego.
- **Extern** *n*um [pattern] => tactic — ogólna postać

# Dodawanie hintów do baz ident1 ... identn

```
Hint hint_def : ident1 ... identn
```

gdzie `hint_def` to

- **Resolve** *t*erm — dodaje apply *t*erm do listy hintów o odpowiednim symbolu głowowym; koszt hintu to liczba wygenerowanych "subgoals". Jeśli "subgoals" mają zmienne wolne to ten hint będzie używany tylko przez eauto.
- **Immediate** *t*erm — dodaje apply *t*erm; trivial do listy hintów o odpowiednim symbolu głowowym. Koszt to 1. Przydatny do używania własności typu symetria równości.
- **Constructor** *i*dent — dodaje wszystkie konstruktory typu indukcyjnego *i*dent jako hinty typu Resolve.
- **Unfold** *i*dent — rozwija *i*dent, jeśli jest w głowie celu dowodowego.
- **Extern** *n*um [pattern] => tactic — ogólna postać

# Dodawanie hintów do baz ident1 ... identn

```
Hint hint_def : ident1 ... identn
```

gdzie `hint_def` to

- **Resolve** *t*erm — dodaje apply *t*erm do listy hintów o odpowiednim symbolu głowowym; koszt hintu to liczba wygenerowanych "subgoals". Jeśli "subgoals" mają zmienne wolne to ten hint będzie używany tylko przez eauto.
- **Immediate** *t*erm — dodaje apply *t*erm; trivial do listy hintów o odpowiednim symbolu głowowym. Koszt to 1. Przydatny do używania własności typu symetria równości.
- **Constructor** *i*dent — dodaje wszystkie konstruktory typu indukcyjnego *i*dent jako hinty typu Resolve.
- **Unfold** *i*dent — rozwija *i*dent, jeśli jest w głowie celu dowodowego.
- **Extern** *n*um [pattern] => tactic — ogólna postać

# Dodawanie hintów do baz ident1 ... identn

```
Hint hint_def : ident1 ... identn
```

gdzie `hint_def` to

- **Resolve** *t*erm — dodaje apply *t*erm do listy hintów o odpowiednim symbolu głowowym; koszt hintu to liczba wygenerowanych "subgoals". Jeśli "subgoals" mają zmienne wolne to ten hint będzie używany tylko przez eauto.
- **Immediate** *t*erm — dodaje apply *t*erm; trivial do listy hintów o odpowiednim symbolu głowowym. Koszt to 1. Przydatny do używania własności typu symetria równości.
- **Constructor** *i*dent — dodaje wszystkie konstruktory typu indukcyjnego *i*dent jako hinty typu Resolve.
- **Unfold** *i*dent — rozwija *i*dent, jeśli jest w głowie celu dowodowego.
- **Extern** *n*um [pattern] => tactic — ogólna postać

# Przykład na Hint Extern

```
Hint Extern 4 (_<> _) => congruence.

Theorem bool_neq : true <> false.
auto.
Qed.
```

auto spróbuje tego hintu, jeśli głowa celu dowodowego jest nierównością i używanie hintów o koszcie mniejszym niż $4$ nie zakończy dowodu.

# Przykład na Hint Extern

```
Hint Extern 4 (_<> _) => congruence.
```

```
Theorem bool_neq : true <> false.
auto.
Qed.
```

auto spróbuje tego hintu, jeśli głowa celu dowodowego jest nierównością i używanie hintów o koszcie mniejszym niż $4$ nie zakończy dowodu.

# Autorewrite

- **autorewrite with ident1 . . . identn** — wykonuje przepisywania według reguł po kolei z baz ident1, ..., identn.
- Może się zapętlić, jeśli reguły z jednej bazy tworzą nieterminujący system przepisywania.
- Może "pójść w złą stronę" i nie znaleźć odpowiedniego ciągu przepisań nawet jeśli on istnieje.
- **Hint Rewrite** służy do rozszerzania baz dla autorewrite.

# Autorewrite

- **autorewrite with ident1 . . . identn** — wykonuje przepisywania według reguł po kolei z baz ident1, ..., identn.
- Może się zapętlić, jeśli reguły z jednej bazy tworzą nieterminujący system przepisywania.
- Może "pójść w złą stronę" i nie znaleźć odpowiedniego ciągu przepisań nawet jeśli on istnieje.
- **Hint Rewrite** służy do rozszerzania baz dla autorewrite.

# Autorewrite

- **autorewrite with ident1 . . . identn** — wykonuje przepisywania według reguł po kolei z baz ident1, ..., identn.
- Może się zapętlić, jeśli reguły z jednej bazy tworzą nieterminujący system przepisywania.
- Może "pójść w złą stronę" i nie znaleźć odpowiedniego ciągu przepisań nawet jeśli on istnieje.
- **Hint Rewrite** służy do rozszerzania baz dla autorewrite.

# Autorewrite

- **autorewrite with ident1 . . . identn** — wykonuje przepisywania według reguł po kolei z baz ident1, ..., identn.
- Może się zapętlić, jeśli reguły z jednej bazy tworzą nieterminujący system przepisywania.
- Może "pójść w złą stronę" i nie znaleźć odpowiedniego ciągu przepisań nawet jeśli on istnieje.
- **Hint Rewrite** służy do rozszerzania baz dla autorewrite.

# Procedury decyzyjne

- **contradiction** — kończy dowód, jesli `False` albo `A` i `¬A` są w założeniach,
- **tauto** — udowadnia instancje zdaniowych tautologii intuinicjonistycznych; rozumie spójniki logiczne, nie rozumie lematów,
- **intuition** lub **intuition tac** — działa jak tauto, ale jak już nie ma co robić, próbuje tac lub auto with *. Intuition upraszcza cel dowodowy.
- **firstorder** — eksperymentalne rozszerzenie tauto na logikę pierwszego rzędu,
- **congruence** — rozwiązuje równości zbudowane z konstruktorów i symboli funkcyjnych,
- **omega** — rozwiązuje formuły bez kwantyfikatorów w liniowej arytmetyce z równościami, nierównościami.

# Procedury decyzyjne

- **contradiction** — kończy dowód, jesli `False` albo `A` i `¬A` są w założeniach,
- **tauto** — udowadnia instancje zdaniowych tautologii intuinicjonistycznych; rozumie spójniki logiczne, nie rozumie lematów,
- **intuition** lub **intuition tac** — działa jak tauto, ale jak już nie ma co robić, próbuje tac lub auto with *. Intuition upraszcza cel dowodowy.
- **firstorder** — eksperymentalne rozszerzenie tauto na logikę pierwszego rzędu,
- **congruence** — rozwiązuje równości zbudowane z konstruktorów i symboli funkcyjnych,
- **omega** — rozwiązuje formuły bez kwantyfikatorów w liniowej arytmetyce z równościami, nierównościami.

# Procedury decyzyjne

- **contradiction** — kończy dowód, jesli `False` albo `A` i `¬A` są w założeniach,
- **tauto** — udowadnia instancje zdaniowych tautologii intuinicjonistycznych; rozumie spójniki logiczne, nie rozumie lematów,
- **intuition** lub **intuition tac** — działa jak tauto, ale jak już nie ma co robić, próbuje tac lub auto with *. Intuition upraszcza cel dowodowy.
- **firstorder** — eksperymentalne rozszerzenie tauto na logikę pierwszego rzędu,
- **congruence** — rozwiązuje równości zbudowane z konstruktorów i symboli funkcyjnych,
- **omega** — rozwiązuje formuły bez kwantyfikatorów w liniowej arytmetyce z równościami, nierównościami.

# Procedury decyzyjne

- **contradiction** — kończy dowód, jesli `False` albo `A` i `¬A` są w założeniach,
- **tauto** — udowadnia instancje zdaniowych tautologii intuinicjonistycznych; rozumie spójniki logiczne, nie rozumie lematów,
- **intuition** lub **intuition tac** — działa jak tauto, ale jak już nie ma co robić, próbuje tac lub auto with *. Intuition upraszcza cel dowodowy.
- **firstorder** — eksperymentalne rozszerzenie tauto na logikę pierwszego rzędu,
- **congruence** — rozwiązuje równości zbudowane z konstruktorów i symboli funkcyjnych,
- **omega** — rozwiązuje formuły bez kwantyfikatorów w liniowej arytmetyce z równościami, nierównościami.

# Procedury decyzyjne

- **contradiction** — kończy dowód, jesli `False` albo `A` i `¬A` są w założeniach,
- **tauto** — udowadnia instancje zdaniowych tautologii intuinicjonistycznych; rozumie spójniki logiczne, nie rozumie lematów,
- **intuition** lub **intuition tac** — działa jak tauto, ale jak już nie ma co robić, próbuje tac lub auto with *. Intuition upraszcza cel dowodowy.
- **firstorder** — eksperymentalne rozszerzenie tauto na logikę pierwszego rzędu,
- **congruence** — rozwiązuje równości zbudowane z konstruktorów i symboli funkcyjnych,
- **omega** — rozwiązuje formuły bez kwantyfikatorów w liniowej arytmetyce z równościami, nierównościami.

# Procedury decyzyjne

- **contradiction** — kończy dowód, jesli `False` albo `A` i `¬A` są w założeniach,
- **tauto** — udowadnia instancje zdaniowych tautologii intuinicjonistycznych; rozumie spójniki logiczne, nie rozumie lematów,
- **intuition** lub **intuition tac** — działa jak tauto, ale jak już nie ma co robić, próbuje tac lub auto with *. Intuition upraszcza cel dowodowy.
- **firstorder** — eksperymentalne rozszerzenie tauto na logikę pierwszego rzędu,
- **congruence** — rozwiązuje równości zbudowane z konstruktorów i symboli funkcyjnych,
- **omega** — rozwiązuje formuły bez kwantyfikatorów w liniowej arytmetyce z równościami, nierównościami.

# Tacticals

- **t1;t2** — aplikuje taktykę t1, a potem t2 na powstałych "subgoals"
- **t1 || t2** — aplikuje taktykę t1; jeśli to zawiedzie to aplikuje t2
- **try t** — aplikuje taktykę t; nic nie robi jeśli t zawiedzie
- **repeat t** — aplikuje taktykę t dopóki może być zaaplikowana i dopóki to coś zmienia w celu dowodowym
- **first [t1 | ... | tn ]** — aplikuje pierwszą aplikowalną taktykę z listy; zawodzi gdy nie ma takiej taktyki
- **solve [ expr1 | ... | exprn ]** — podobnie jak first ale zaaplikowana taktyka musi skończyć dowód
- **idtac** — taktyka identycznościowa (nic nie zmienia; często w skryptach)
- **fail** — taktyka która zawsze zawodzi
- **match** — ....

# Tacticals

- **t1;t2** — aplikuje taktykę t1, a potem t2 na powstałych "subgoals"
- **t1 || t2** — aplikuje taktykę t1; jeśli to zawiedzie to aplikuje t2
- **try t** — aplikuje taktykę t; nic nie robi jeśli t zawiedzie
- **repeat t** — aplikuje taktykę t dopóki może być zaaplikowana i dopóki to coś zmienia w celu dowodowym
- **first [t1 | ... | tn ]** — aplikuje pierwszą aplikowalną taktykę z listy; zawodzi gdy nie ma takiej taktyki
- **solve [ expr1 | ... | exprn ]** — podobnie jak first ale zaaplikowana taktyka musi skończyć dowód
- **idtac** — taktyka identycznościowa (nic nie zmienia; często w skryptach)
- **fail** — taktyka która zawsze zawodzi
- **match** — ....

# Tacticals

- **t1;t2** — aplikuje taktykę t1, a potem t2 na powstałych "subgoals"
- **t1 || t2** — aplikuje taktykę t1; jeśli to zawiedzie to aplikuje t2
- **try t** — aplikuje taktykę t; nic nie robi jeśli t zawiedzie
- **repeat t** — aplikuje taktykę t dopóki może być zaaplikowana i dopóki to coś zmienia w celu dowodowym
- **first [t1 | ... | tn ]** — aplikuje pierwszą aplikowalną taktykę z listy; zawodzi gdy nie ma takiej taktyki
- **solve [ expr1 | ... | exprn ]** — podobnie jak first ale zaaplikowana taktyka musi skończyć dowód
- **idtac** — taktyka identycznościowa (nic nie zmienia; często w skryptach)
- **fail** — taktyka która zawsze zawodzi
- **match** — ....

# Tacticals

- **t1;t2** — aplikuje taktykę t1, a potem t2 na powstałych "subgoals"
- **t1 || t2** — aplikuje taktykę t1; jeśli to zawiedzie to aplikuje t2
- **try t** — aplikuje taktykę t; nic nie robi jeśli t zawiedzie
- **repeat t** — aplikuje taktykę t dopóki może być zaaplikowana i dopóki to coś zmienia w celu dowodowym
- **first [t1 | ... | tn ]** — aplikuje pierwszą aplikowalną taktykę z listy; zawodzi gdy nie ma takiej taktyki
- **solve [ expr1 | ... | exprn ]** — podobnie jak first ale zaaplikowana taktyka musi skończyć dowód
- **idtac** — taktyka identycznościowa (nic nie zmienia; często w skryptach)
- **fail** — taktyka która zawsze zawodzi
- **match** — ....

# Tacticals

- **t1;t2** — aplikuje taktykę t1, a potem t2 na powstałych "subgoals"
- **t1 || t2** — aplikuje taktykę t1; jeśli to zawiedzie to aplikuje t2
- **try t** — aplikuje taktykę t; nic nie robi jeśli t zawiedzie
- **repeat t** — aplikuje taktykę t dopóki może być zaaplikowana i dopóki to coś zmienia w celu dowodowym
- **first [t1 | ... | tn ]** — aplikuje pierwszą aplikowalną taktykę z listy; zawodzi gdy nie ma takiej taktyki
- **solve [ expr1 | ... | exprn ]** — podobnie jak first ale zaaplikowana taktyka musi skończyć dowód
- **idtac** — taktyka identycznościowa (nic nie zmienia; często w skryptach)
- **fail** — taktyka która zawsze zawodzi
- **match** — ....

# Tacticals

- **t1;t2** — aplikuje taktykę t1, a potem t2 na powstałych "subgoals"
- **t1 || t2** — aplikuje taktykę t1; jeśli to zawiedzie to aplikuje t2
- **try t** — aplikuje taktykę t; nic nie robi jeśli t zawiedzie
- **repeat t** — aplikuje taktykę t dopóki może być zaaplikowana i dopóki to coś zmienia w celu dowodowym
- **first [t1 | ... | tn ]** — aplikuje pierwszą aplikowalną taktykę z listy; zawodzi gdy nie ma takiej taktyki
- **solve [ expr1 | ... | exprn ]** — podobnie jak first ale zaaplikowana taktyka musi skończyć dowód
- **idtac** — taktyka identycznościowa (nic nie zmienia; często w skryptach)
- **fail** — taktyka która zawsze zawodzi
- **match** — ....

# Tacticals

- **t1;t2** — aplikuje taktykę t1, a potem t2 na powstałych "subgoals"
- **t1 || t2** — aplikuje taktykę t1; jeśli to zawiedzie to aplikuje t2
- **try t** — aplikuje taktykę t; nic nie robi jeśli t zawiedzie
- **repeat t** — aplikuje taktykę t dopóki może być zaaplikowana i dopóki to coś zmienia w celu dowodowym
- **first [t1 | ... | tn ]** — aplikuje pierwszą aplikowalną taktykę z listy; zawodzi gdy nie ma takiej taktyki
- **solve [ expr1 | ... | exprn ]** — podobnie jak first ale zaaplikowana taktyka musi skończyć dowód
- **idtac** — taktyka identycznościowa (nic nie zmienia; często w skryptach)
- **fail** — taktyka która zawsze zawodzi
- **match** — ....

# Tacticals

- **t1;t2** — aplikuje taktykę t1, a potem t2 na powstałych "subgoals"
- **t1 || t2** — aplikuje taktykę t1; jeśli to zawiedzie to aplikuje t2
- **try t** — aplikuje taktykę t; nic nie robi jeśli t zawiedzie
- **repeat t** — aplikuje taktykę t dopóki może być zaaplikowana i dopóki to coś zmienia w celu dowodowym
- **first [t1 | ... | tn ]** — aplikuje pierwszą aplikowalną taktykę z listy; zawodzi gdy nie ma takiej taktyki
- **solve [ expr1 | ... | exprn ]** — podobnie jak first ale zaaplikowana taktyka musi skończyć dowód
- **idtac** — taktyka identycznościowa (nic nie zmienia; często w skryptach)
- **fail** — taktyka która zawsze zawodzi
- **match** — ....

# Tacticals

- **t1;t2** — aplikuje taktykę t1, a potem t2 na powstałych "subgoals"
- **t1 || t2** — aplikuje taktykę t1; jeśli to zawiedzie to aplikuje t2
- **try t** — aplikuje taktykę t; nic nie robi jeśli t zawiedzie
- **repeat t** — aplikuje taktykę t dopóki może być zaaplikowana i dopóki to coś zmienia w celu dowodowym
- **first [t1 | ... | tn ]** — aplikuje pierwszą aplikowalną taktykę z listy; zawodzi gdy nie ma takiej taktyki
- **solve [ expr1 | ... | exprn ]** — podobnie jak first ale zaaplikowana taktyka musi skończyć dowód
- **idtac** — taktyka identycznościowa (nic nie zmienia; często w skryptach)
- **fail** — taktyka która zawsze zawodzi
- **match** — ....

# Match w konkluzji

```
Ltac find_if :=
  match goal with
    | [ ⊢ if ?X then _ else _ ] ⇒ destruct X
  end.

Theorem hmm : ∀ (a b c : bool),
  if a
    then if b
      then True
      else True
    else if c
      then True
      else True.
  intros; repeat find_if; constructor.
Qed.
```

# Match w konkluzji

```
Ltac find_if :=
  match goal with
    | [ ⊢ if ?X then _ else _ ] ⇒ destruct X
  end.

Theorem hmm : ∀ (a b c : bool),
  if a
    then if b
      then True
      else True
    else if c
      then True
      else True.
  intros; repeat find_if; constructor.
Qed.
```

# Context pattern

```
Ltac find_if_inside :=
  match goal with
    | [ ⊢ context[if ?X then _ else _] ] ⇒ destruct X
  end.
```

```
Theorem hmm2 : ∀ (a b : bool),
  (if a then 42 else 42) = (if b then 42 else 42).
  intros; repeat find_if_inside; reflexivity.
Qed.
```

# Context pattern

```
Ltac find_if_inside :=
  match goal with
    | [ ⊢ context[if ?X then _ else _] ] ⇒ destruct X
  end.
```

```
Theorem hmm2 : ∀ (a b : bool),
  (if a then 42 else 42) = (if b then 42 else 42).
  intros; repeat find_if_inside; reflexivity.
Qed.
```

# taktyka my_tauto

```
Ltac my_tauto :=
  repeat match goal with
           | [ H : ?P ⊢ ?P ] ⇒ exact H

           | [ ⊢ True ] ⇒ constructor
           | [ ⊢ _ ∧ _ ] ⇒ constructor
           | [ ⊢ _ → _ ] ⇒ intro

           | [ H : False ⊢ _ ] ⇒ destruct H
           | [ H : _ ∧ _ ⊢ _ ] ⇒ destruct H
           | [ H : _ ∨ _ ⊢ _ ] ⇒ destruct H

           | [ H1 : ?P → ?Q, H2 : ?P ⊢ _ ] ⇒ specialize (H1 H2)
         end.
```

# Semantyka match

Wewnątrz jednej gałęzi kontekst jest dopasowywany na wszystkie możliwe sposoby dopóki taktyka po => się nie powiedzie:

```
Theorem m2 : ∀ P Q R : Prop, P → Q → R → Q.
  intros; match goal with
            | [ H : _ ⊢ _ ] ⇒ idtac H
          end.
```

Coq wypisuje "*H1*", dowód się nie udał...

# Semantyka match

Wewnątrz jednej gałęzi kontekst jest dopasowywany na wszystkie możliwe sposoby dopóki taktyka po => się nie powiedzie:

```
Theorem m2 : ∀ P Q R : Prop, P → Q → R → Q.
  intros; match goal with
            | [ H : _ ⊢ _ ] ⇒ idtac H
          end.
```

Coq wypisuje "*H1*", dowód się nie udał...

# Semantyka match cd.

Poniższy skrypt zadziała:

```
  match goal with
    | [ H : _ ⊢ _ ] ⇒ exact H
  end.
Qed.
```

Jeśli match ma wiele gałęzi to najpierw sprawdzane są wszystkie dopasowania pierwszej, potem wszystkie dopasowania drugiej, itd.

# Semantyka match cd.

Poniższy skrypt zadziała:

```
  match goal with
    | [ H : _ ⊢ _ ] ⇒ exact H
  end.
Qed.
```

Jeśli match ma wiele gałęzi to najpierw sprawdzane są wszystkie dopasowania pierwszej, potem wszystkie dopasowania drugiej, itd.

# Semantyka match cd.

Taktyka *n*otHyp P sprawdza, że P nie jest wśród założeń.

```
Ltac notHyp P :=
  match goal with
    | [ _ : P ⊢ _ ] ⇒ fail 1
    | _ ⇒
      match P with
        | ?P1 ∧ ?P2 ⇒ first [ notHyp P1 | notHyp P2 | fail 2 ]
        | _ ⇒ idtac
      end
  end.
```

Wywołanie **fail n** wychodzi z aktualnego bloku match (albo try) i zmniejsza licznik przy fail.
Dochodzimy do idtac jeśli P nie jest koniunkcją.

# Taktyki extend i completer

```
Ltac extend pf :=
  let t := type of pf in
    notHyp t; generalize pf; intro.
```

```
Ltac completer :=
  repeat match goal with
           | [ ⊢ _ ∧ _ ] ⇒ constructor
           | [ H : _ ∧ _ ⊢ _ ] ⇒ destruct H
           | [ H : ?P → ?Q, H' : ?P ⊢ _ ] ⇒ specialize (H H')
           | [ ⊢ ∀ x, _ ] ⇒ intro

           | [ H : ∀ x, ?P x → _, H' : ?P ?X ⊢ _ ] ⇒ extend (H X H')
         end.
```

# Taktyki extend i completer

```
Ltac extend pf :=
  let t := type of pf in
    notHyp t; generalize pf; intro.
```

```
Ltac completer :=
  repeat match goal with
           | [ ⊢ _ ∧ _ ] ⇒ constructor
           | [ H : _ ∧ _ ⊢ _ ] ⇒ destruct H
           | [ H : ?P → ?Q, H' : ?P ⊢ _ ] ⇒ specialize (H H')
           | [ ⊢ ∀ x, _ ] ⇒ intro

           | [ H : ∀ x, ?P x → _, H' : ?P ?X ⊢ _ ] ⇒ extend (H X H')
         end.
```

# Completer' — błędna wersja

```
Ltac completer' :=
  repeat match goal with
           | [ ⊢ _ ∧ _ ] ⇒ constructor
           | [ H : ?P ∧ ?Q ⊢ _ ] ⇒ destruct H;
             repeat match goal with
                      | [ H' : P ∧ Q ⊢ _ ] ⇒ clear H'
                    end
           | [ H : ?P → _, H' : ?P ⊢ _ ] ⇒ specialize (H H')
           | [ ⊢ ∀ x, _ ] ⇒ intro

           | [ H : ∀ x, ?P x → _, H' : ?P ?X ⊢ _ ] ⇒ extend (H X H')
         end.
```

Zmienna `?Q` została zastąpiona przez `_` w trzeciej gałęzi match; kwantyfikator uniwersalny zaczął tam pasować.

```
Section firstorder'.
  Variable A : Set.
  Variables P Q R S : A → Prop.

  Hypothesis H1 : ∀ x, P x → Q x ∧ R x.
  Hypothesis H2 : ∀ x, R x → S x.

  Theorem fo' : ∀ (y x : A), P x → S x.
    completer'.


  y : A
  H1 : P y → Q y ∧ R y
  H2 : R y → S y
  x : A
  H : P x
  ============================
   S x
```

# Użycie fresh

W rozwiązaniach FunOpt.v pojawiła się:

```
Ltac inv_te typexp :=
 let t := fresh "t"in
 destruct typexp as [t| ] ; [destruct t | idtac]; try congruence
```

fresh "t"generuje nową nazwę zaczynającą się na t

# Częściowa aplikacja specialize, semantyka ?w i match

W rozwiązaniach FunOpt.v pojawiła się:

```
Ltac sSome H :=
 match type of H with
  | forall t, _-> Some ?w=Some t -> _=> specialize H with w
  | forall t, _-> Some t=Some ?w -> _=> specialize H with w
 end.
```

```
Ltac autosS := match goal with [ H :_|- _] => sSome H end.
```

W dowodzie użyte `repeat autosS.`

# Częściowa aplikacja specialize, semantyka ?w i match

W rozwiązaniach FunOpt.v pojawiła się:

```
Ltac sSome H :=
 match type of H with
  | forall t, _-> Some ?w=Some t -> _=> specialize H with w
  | forall t, _-> Some t=Some ?w -> _=> specialize H with w
 end.
```

```
Ltac autosS := match goal with [ H :_|- _] => sSome H end.
```

W dowodzie użyte `repeat autosS.`

# Częściowa aplikacja specialize, semantyka ?w i match

W rozwiązaniach FunOpt.v pojawiła się:

```
Ltac sSome H :=
 match type of H with
  | forall t, _-> Some ?w=Some t -> _=> specialize H with w
  | forall t, _-> Some t=Some ?w -> _=> specialize H with w
 end.
```

```
Ltac autosS := match goal with [ H :_|- _] => sSome H end.
```

W dowodzie użyte `repeat autosS.`