Úvod do Prologu, backtracking, unifikace, SLD stromy

IB015 Neimperativní programování

Kolektiv cvičících IB015

Fakulta informatiky, Masarykova univerzita

podzim 2018

Názvosloví

```
1 human(petr).
2 dog(nero).
3 friends(X, Y) :- human(X), dog(Y).
```

```
1 ?- friends(petr, nero).
```

term = základní datová struktura

- **konstanta** (1, 'petr'), **proměnná** (X, Var1)
- **složený term** = funktor aplikovaný na argumenty

predikát = seznam faktů a pravidel se stejným názvem a aritou

- **pravidlo** = hlava, jeden nebo více cílů v těle
- **fakt** = jenom hlava, tělo je prázdné

program = množina predikátů

- **dotaz** = prázdná hlava, jeden nebo více cílů v těle

Interpret jazyka Prolog

Interpret/kompilátor: *SWI-Prolog* (příkaz `swipl`)
Základní příkazy/predikáty (musí končit tečkou):

- `help/0` Zobrazí základní nápovědu o použití nápovědy.
- `help/1` Zobrazí nápovědu k predikátu v argumentu.
- `apropos/1` Zobrazí predikáty, které mají v názvu nebo popisu zadaný výraz.
- `consult/1` Načte (zkompiluje) zdrojový kód ze zadaného souboru. Nebo také `['file.pl']` a `[file]`.
- `make/0` Znovu načte zdrojový kód ze změněných souborů.
- `halt/0` Ukončí interpret *SWI-Prolog*.

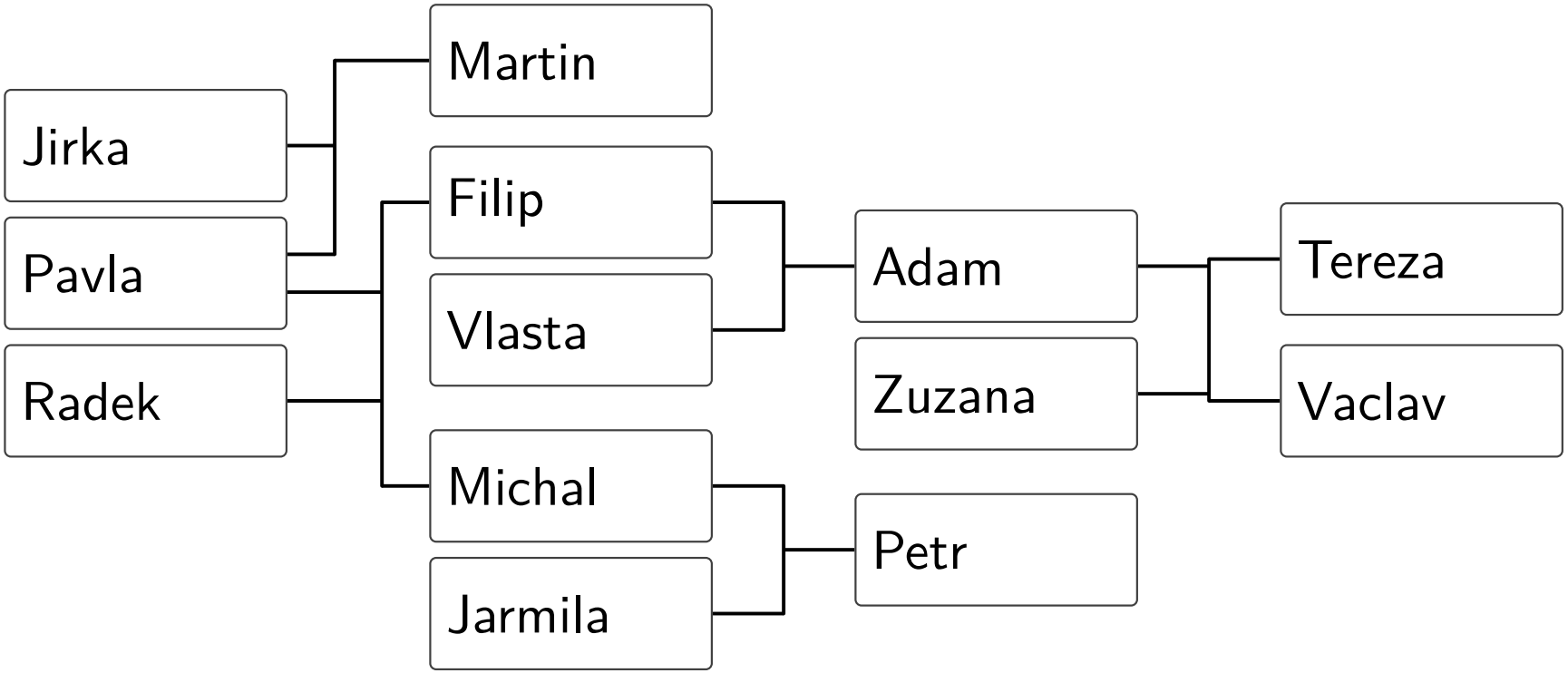
Další řešení si vyžádáme středníkem (`;`),
výpočet ukončíme tečkou (`.`).

Rodokmen

Příklad 9.1.2: Načtěte rodokmen ze souboru `09_pedigree.pl` do prostředí Prologu. Soubor naleznete v ISu a v příloze sbírky. Formulujte vhodné dotazy a pomocí interpretru zjistěte odpovědi na následující otázky:

a) Je Pavla rodičem Filipa?
b) Je Pavla rodičem Vlasty?
c) Jaké děti má Pavla?
d) Má Adam dceru?
e) Kdo je otcem Petra?
f) Které dvojice otec-syn známe?

Rodokmen – obrázek



Komplikovanější ukázka

Interpretace: `:-` je implikace zpět ($\Leftarrow$), čárka je konjunkce ($\wedge$).

```
1 friends(X, X).
2 friends(X, vlasta).
3 friends(X, Y) :- Y == martin.
4 friends(X, Y) :- human(X), dog(Y).
```

Komplikovanější ukázka

Interpretace: `:-` je implikace zpět ($\Leftarrow$), čárka je konjunkce ($\wedge$).

```
1 friends(X, X).
2 friends(X, vlasta).
3 friends(X, Y) :- Y == martin.
4 friends(X, Y) :- human(X), dog(Y).
```

Jak se budou vyhodnocovat následující dotazy?

```
1 ? - friends(vlasta, nero).
2 ? - friends(petr, vlasta).
3 ? - friends(petr, Z).
```

Postup hledání pravidla:

1. Hledám predikát se stejným jménem a aritou.
2. Pro každý řádek (alternativu):
   - Jestli jde unifikovat s hlavou pravidla, přepisuju za tělo.
   - Jestli nejde, zkouším další řádek.

Unifikace

Dva termy jsou unifikovatelné, pokud jsou identické nebo je možné dosadit za proměnné termy tak, že se původní termy stanou identickými.

- `parent(X) = parent(petr)`

Unifikace

Dva termy jsou unifikovatelné, pokud jsou identické nebo je možné dosadit za proměnné termy tak, že se původní termy stanou identickými.

- `parent(X) = parent(petr)`
  unifikovatelné, unifikace `X = petr`
- `parent(X,Y) = parent(petr)`

Unifikace

Dva termy jsou unifikovatelné, pokud jsou identické nebo je možné dosadit za proměnné termy tak, že se původní termy stanou identickými.

- `parent(X) = parent(petr)`
  unifikovatelné, unifikace `X = petr`
- `parent(X,Y) = parent(petr)`
  neunifikovatelné, různá arita funktorů
- `parent(X,Y) = father(petr, Z)`

Unifikace

Dva termy jsou unifikovatelné, pokud jsou identické nebo je možné dosadit za proměnné termy tak, že se původní termy stanou identickými.

- `parent(X) = parent(petr)`
  unifikovatelné, unifikace `X = petr`
- `parent(X,Y) = parent(petr)`
  neunifikovatelné, různá arita funktorů
- `parent(X,Y) = father(petr, Z)`
  neunifikovatelné, různá jména funktorů
- `parent(X,X) = parent(petr, Z)`

Unifikace

Dva termy jsou unifikovatelné, pokud jsou identické nebo je možné dosadit za proměnné termy tak, že se původní termy stanou identickými.

- `parent(X) = parent(petr)`
  unifikovatelné, unifikace `X = petr`
- `parent(X,Y) = parent(petr)`
  neunifikovatelné, různá arita funktorů
- `parent(X,Y) = father(petr, Z)`
  neunifikovatelné, různá jména funktorů
- `parent(X,X) = parent(petr, Z)`
  unifikovatelné, unifikace `X = Z, Z = petr`
- `parent(X,eva) = parent(petr, X)`

Unifikace

Dva termy jsou unifikovatelné, pokud jsou identické nebo je možné dosadit za proměnné termy tak, že se původní termy stanou identickými.

- `parent(X) = parent(petr)`
  unifikovatelné, unifikace `X = petr`
- `parent(X,Y) = parent(petr)`
  neunifikovatelné, různá arita funktorů
- `parent(X,Y) = father(petr, Z)`
  neunifikovatelné, různá jména funktorů
- `parent(X,X) = parent(petr, Z)`
  unifikovatelné, unifikace `X = Z, Z = petr`
- `parent(X,eva) = parent(petr, X)`
  neunifikovatelné

Unifikace v SWI-Prologu nedělá test na sebevýskyt (*occurs check*)!

Unifikace – příklad

Příklad 9.2.1: Které unifikace uspějí, které ne a proč? Jaký je výsledek provedených unifikací?

a) `man(X) = woman(X)`
b) `X = blue(Y)`
c) `green(X) = green(X,X)`
d) `X = man(X)`
e) `jmeno(X,X) = jmeno(Petr,novak)`
f) `weekend(day(saturday),day(sunday)) = weekend(X,Y)`
g) `weekend(day(saturday),X) = weekend(X,day(sunday))`

Krabicový model

Krabicový model je jednoduchá vizualizace průběhu výpočtu:

- Call: voláme predikát, abychom zjistili, jestli uspěje
- Exit: predikát uspěl (pokračujeme voláním dalšího predikátu)
- Fail: predikát neuspěl (backtrackujeme zpátky)
- Redo: voláme predikát v rámci backtrackingu (snažíme se uspět dle jiného pravidla)

Krokování výpočtu

Krokování umožňuje vidět výpočetní strom, a tak pomáhá debugování.

- Zapnutí/vypnutí krokování pomocí predikátů `trace/0`, `notrace/0`.
- Používá se terminologie krabicového modelu.

Vyzkoušejte si:

```
1 ?- trace.
2 ?- father(X, petr).
```

Pokročilejší rodinné vztahy I.

Příklad 9.1.3: Pro rodokmen ze souboru `09_pedigree.pl` naprogramujte následující predikáty:

a) `child/2`, který uspěje, jestliže první argument je dítětem druhého.
b) `grandmother/2`, který uspěje, jestliže první argument je babičkou druhého.
c) `brother/2`, který uspěje, jestliže první argument je bratrem druhého argumentu (mají tedy alespoň jednoho společného rodiče).

Pokročilejší rodinné vztahy II.

Příklad 9.1.5: Pro rodokmen ze souboru `09_pedigree.pl` napište predikát `descendant/2`, který uspěje, když je první argument potomkem druhého (ne nutně přímý). Bez použití interpretru určete, v jakém pořadí budou nalezeni potomci Pavly, když použijeme dotaz `?- descendant(X,pavla).`
Jaký vliv má pořadí klauzulí a cílů v predikátu `descendant` na jeho funkci?

SLD stromy – teorie

SLD stromy = způsob vizualizace výpočtu v Prologu

- v kořenu je dotaz
- jednotlivé podcíle se vyhodnocují zleva doprava
- při více možnostech se strom větví
- hrany jsou anotované provedenými unifikacemi
- prázdné listy značí úspěch
- neúspěšné větve označeny výrazem **fail**

SLD stromy – jednoduchá ukázka

Příklad 9.3.2: Uvažme následující program představující podmínky k dosažení bakalářského titulu (pravidla) a věci, které máte úspěšně za sebou (fakta). Nakreslete odpovídající výpočetní SLD strom pro dotaz `?- bcDegree.`

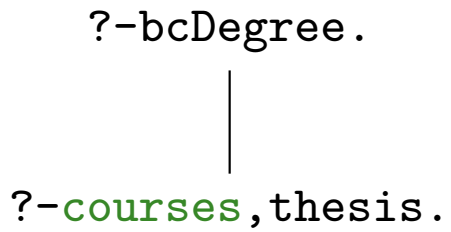
```
1 bcDegree :- courses, thesis.
2 courses :- programming, maths.
3 courses :- programming, exception.
4 exception :- deanAgrees.
5 exception :- rectorAgrees.
```

```
1 programming.
2 thesis.
3 deanAgrees.
```

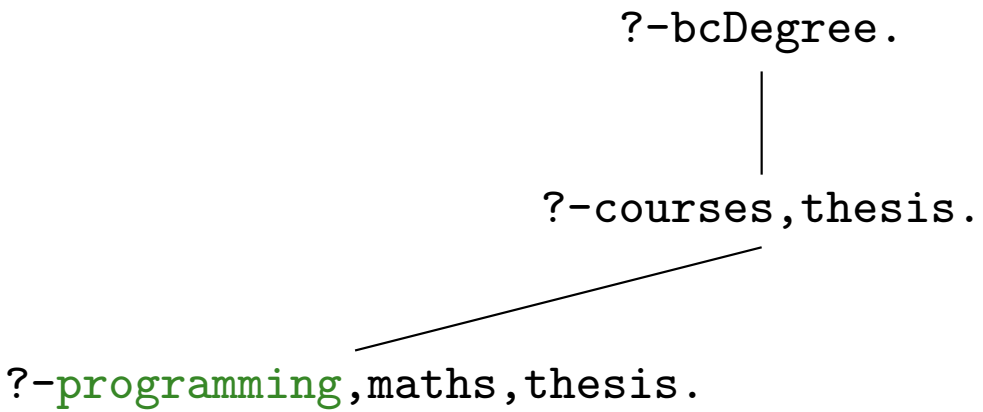
SLD stromy – jednoduchá ukázka (řešení)

```
?-bcDegree.
```

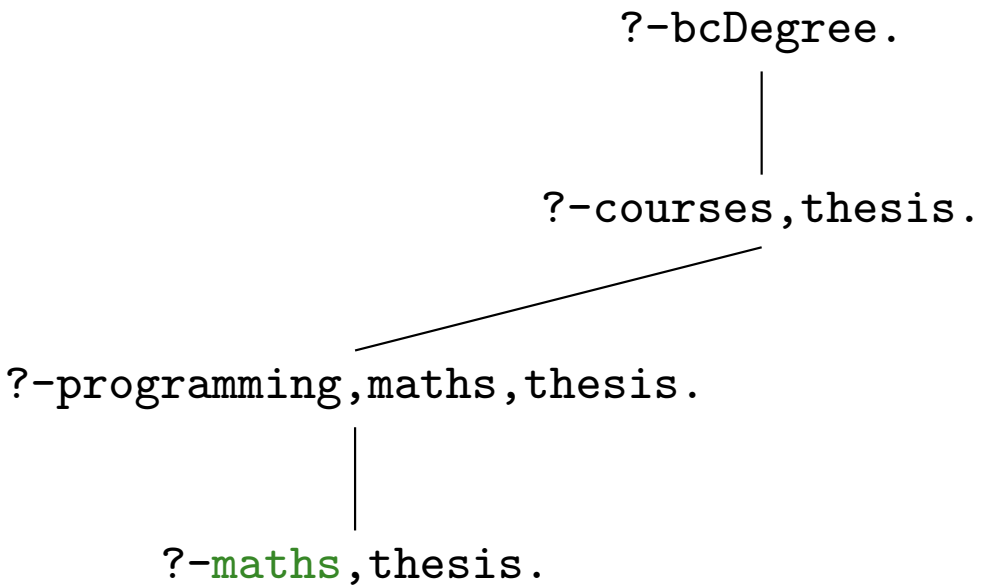
SLD stromy – jednoduchá ukázka (řešení)



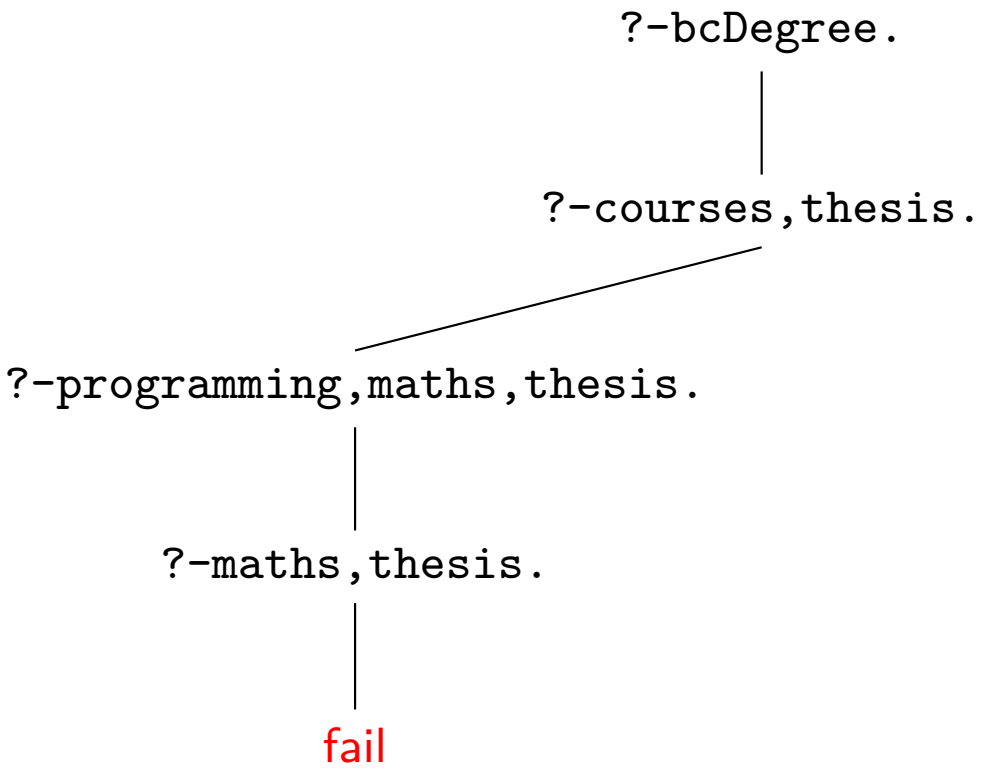
SLD stromy – jednoduchá ukázka (řešení)



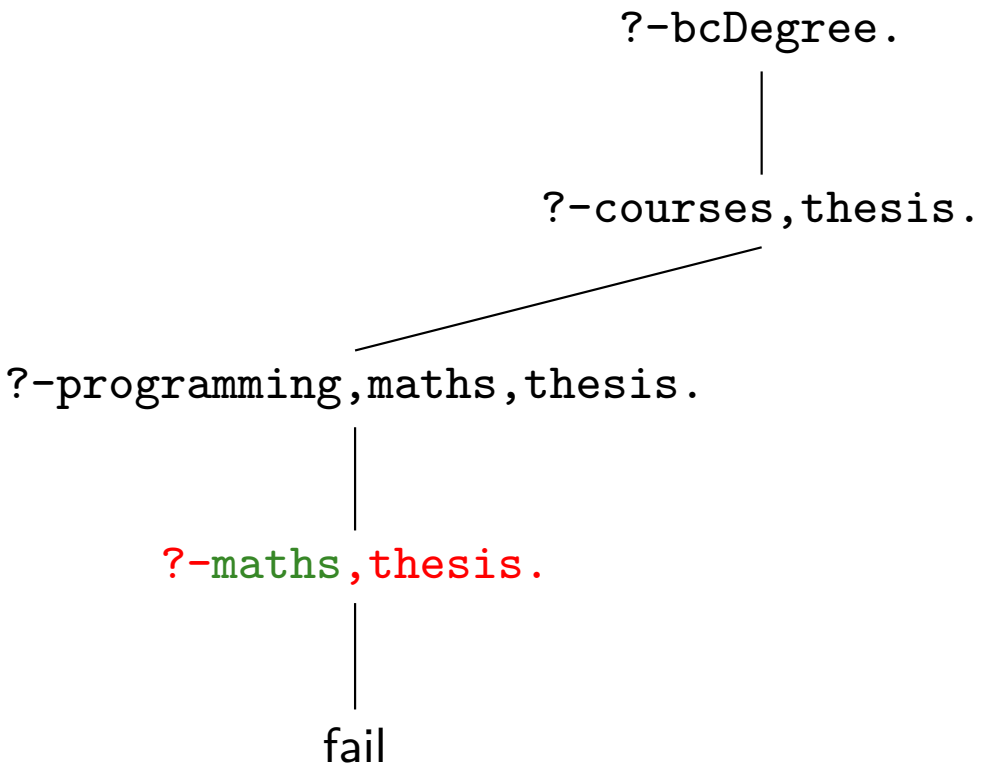
SLD stromy – jednoduchá ukázka (řešení)



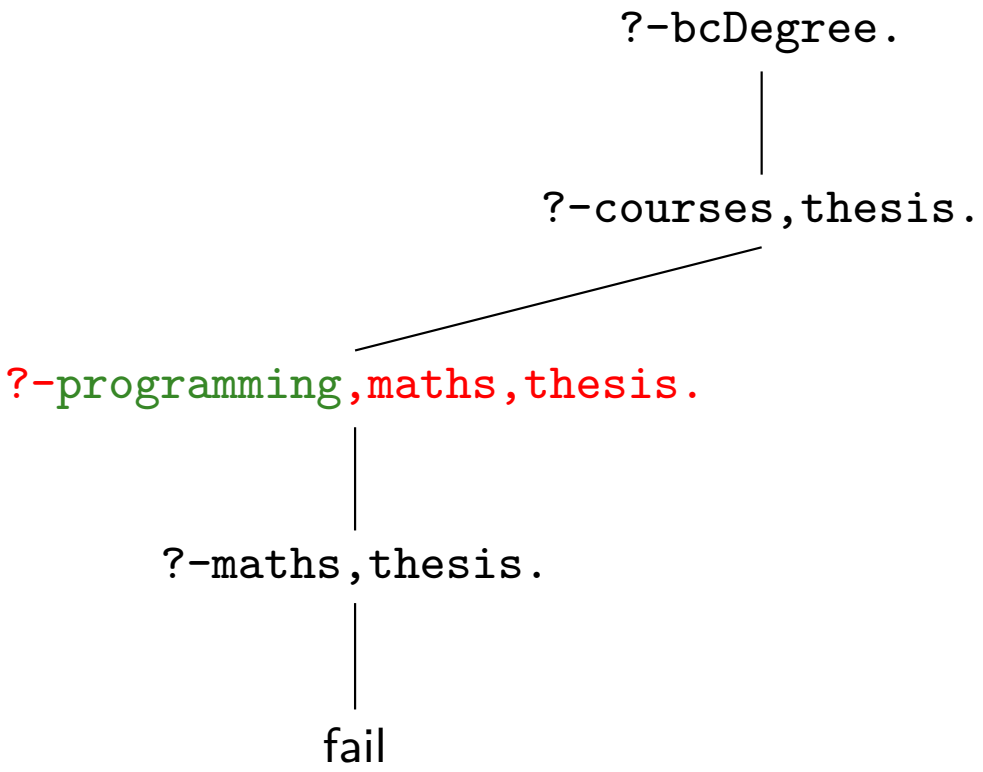
SLD stromy – jednoduchá ukázka (řešení)



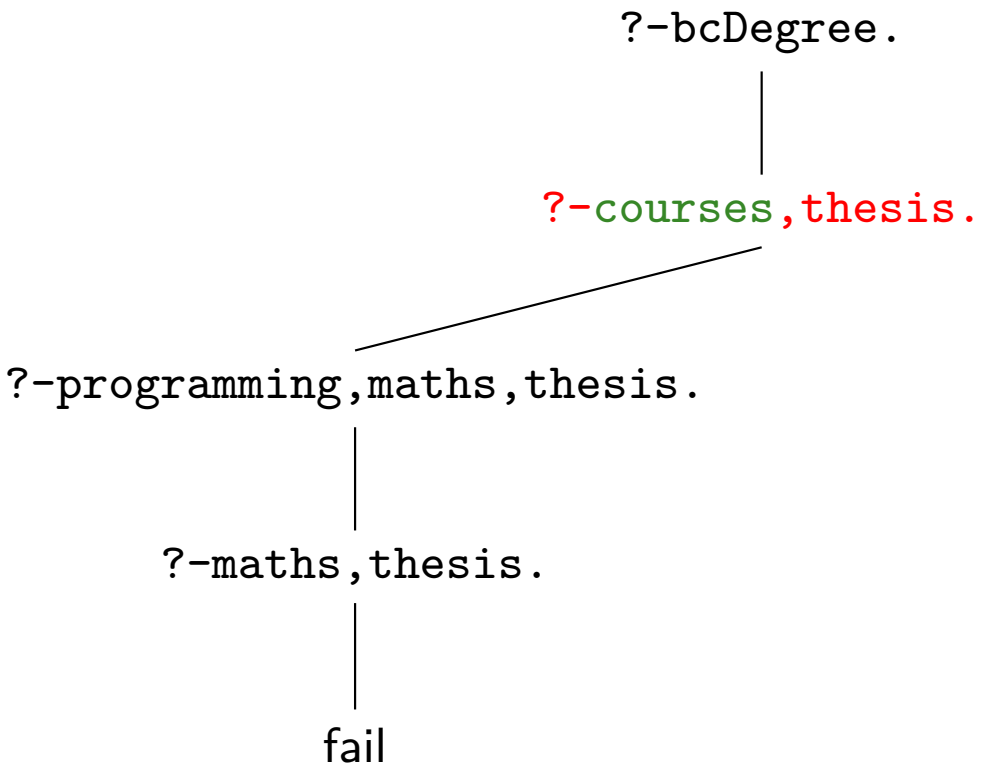
SLD stromy – jednoduchá ukázka (řešení)



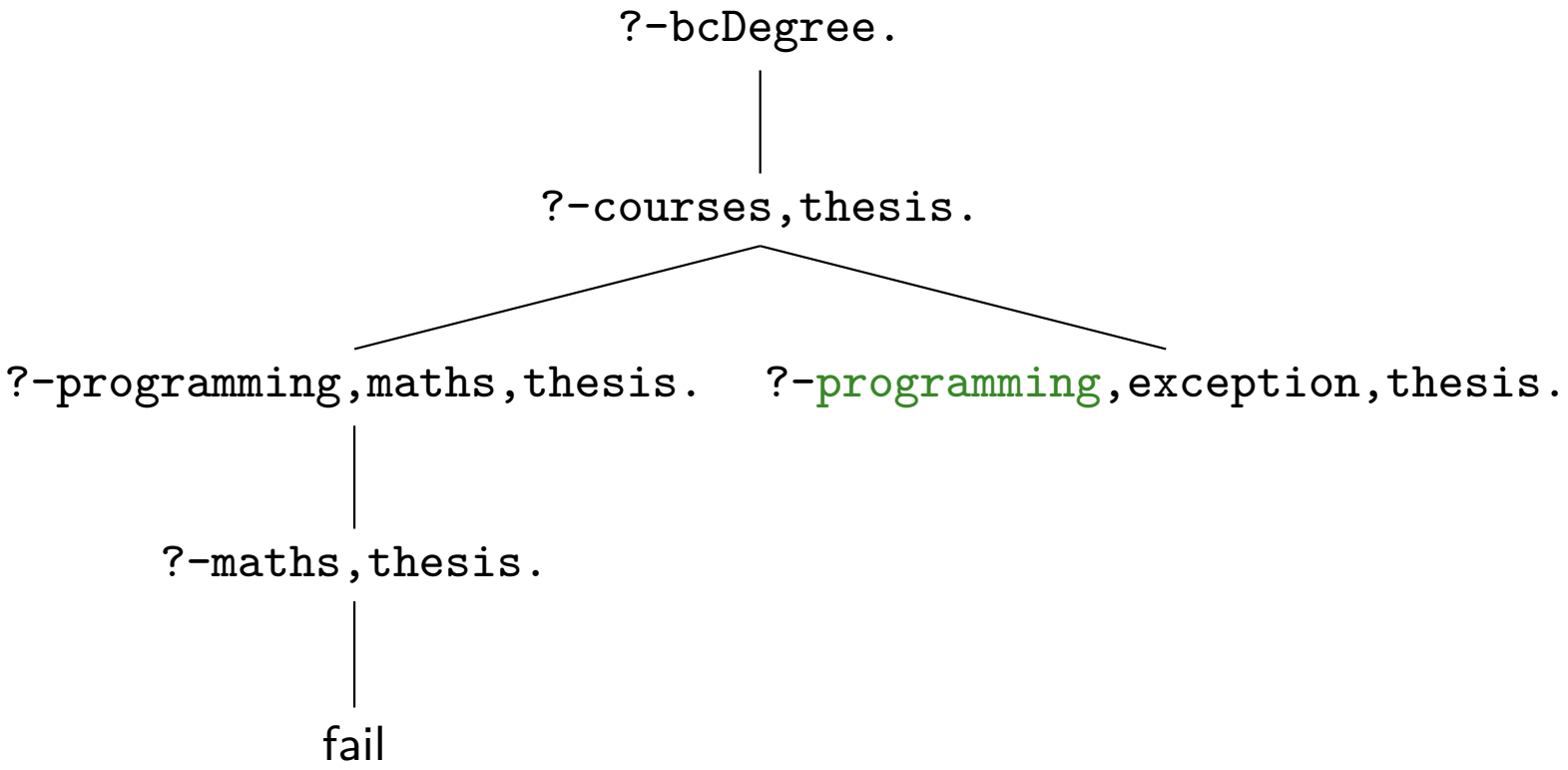
SLD stromy – jednoduchá ukázka (řešení)



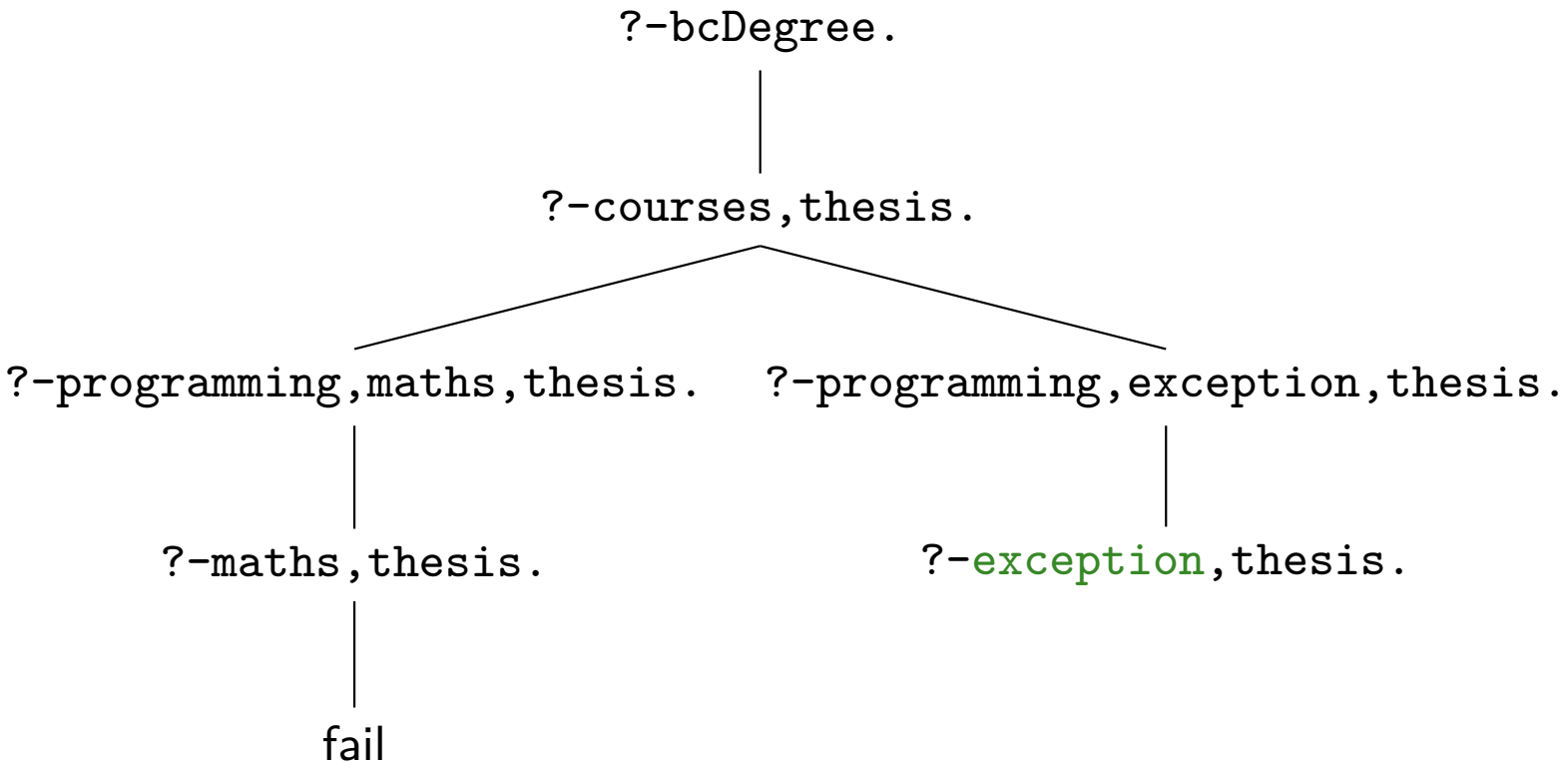
SLD stromy – jednoduchá ukázka (řešení)



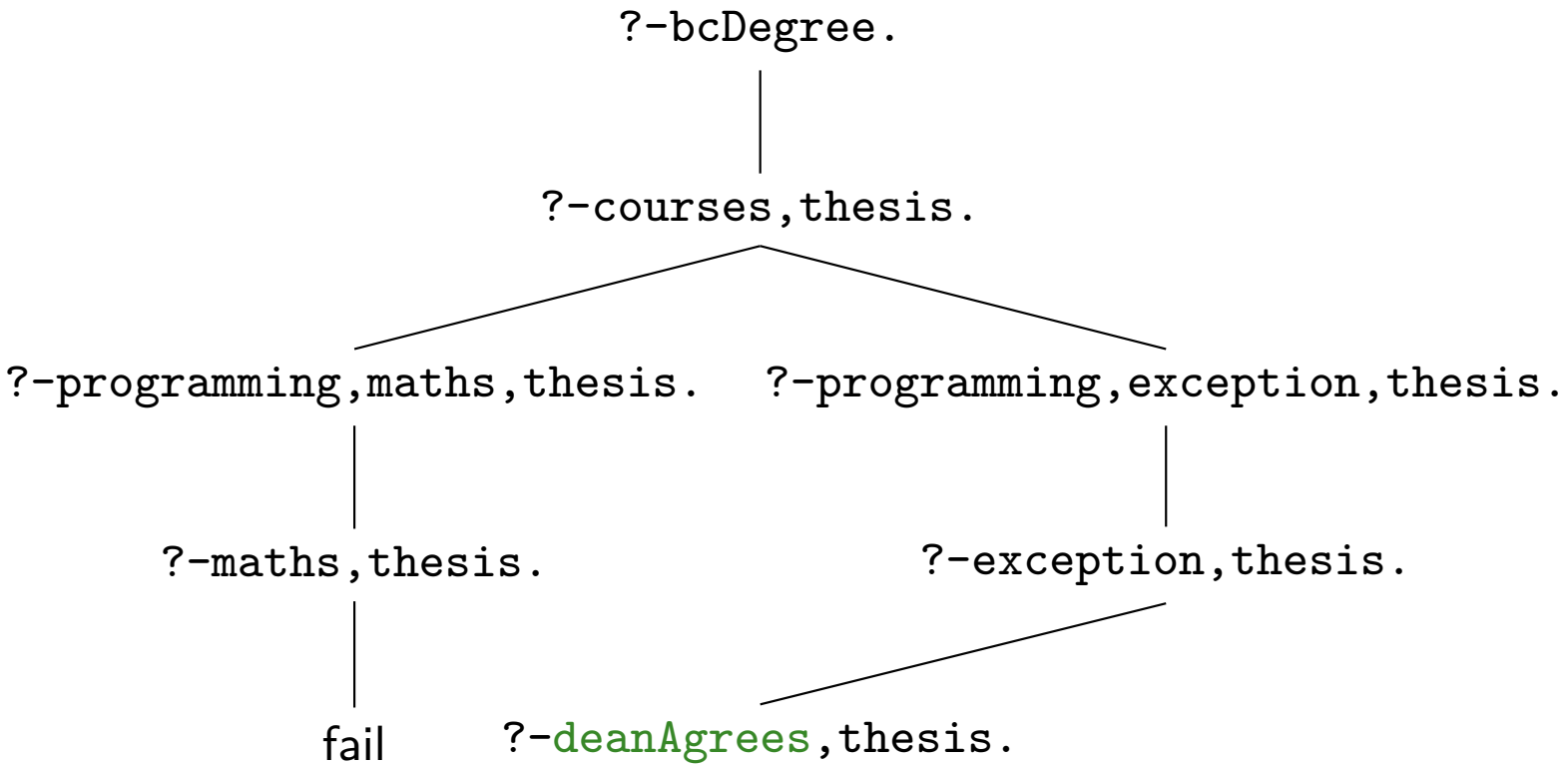
SLD stromy – jednoduchá ukázka (řešení)



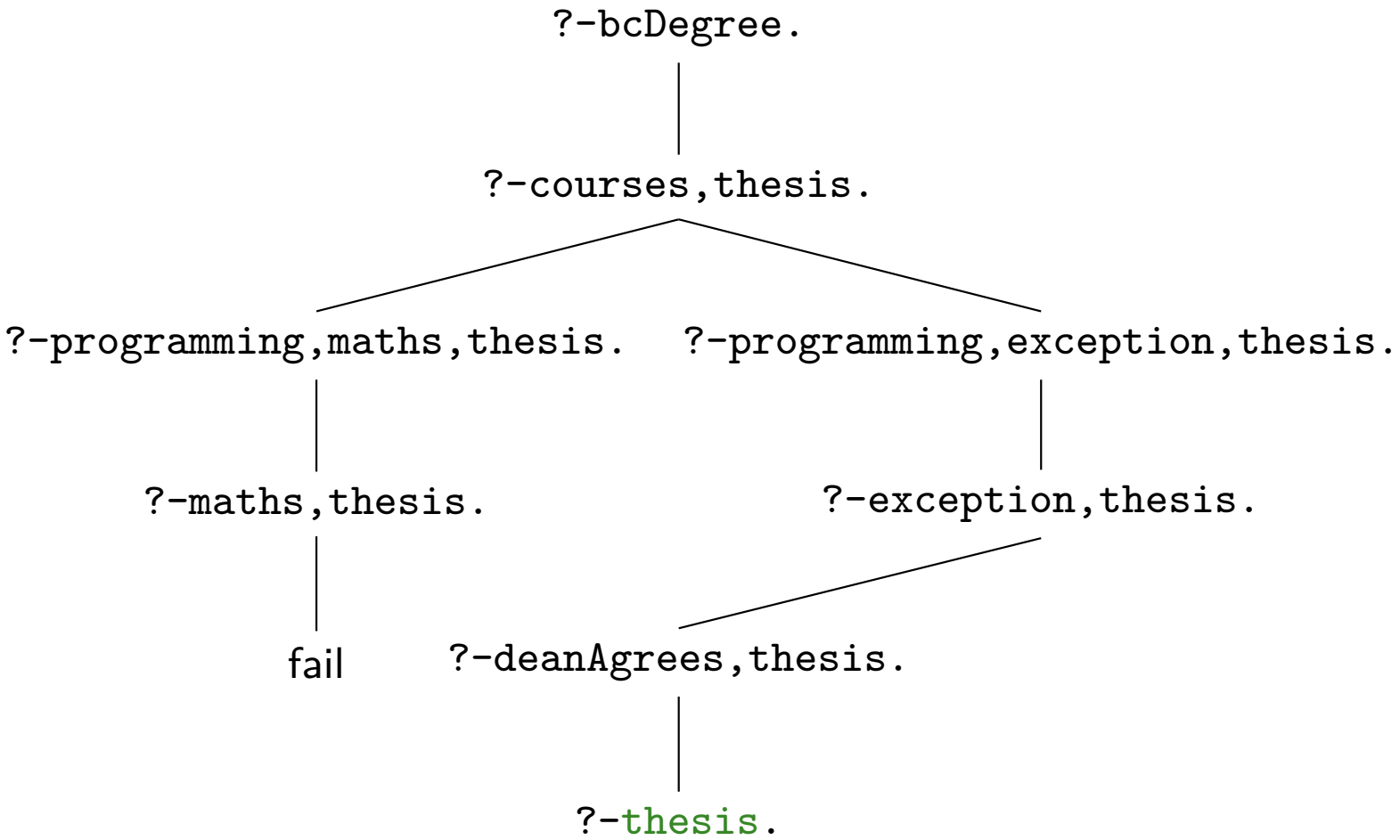
SLD stromy – jednoduchá ukázka (řešení)



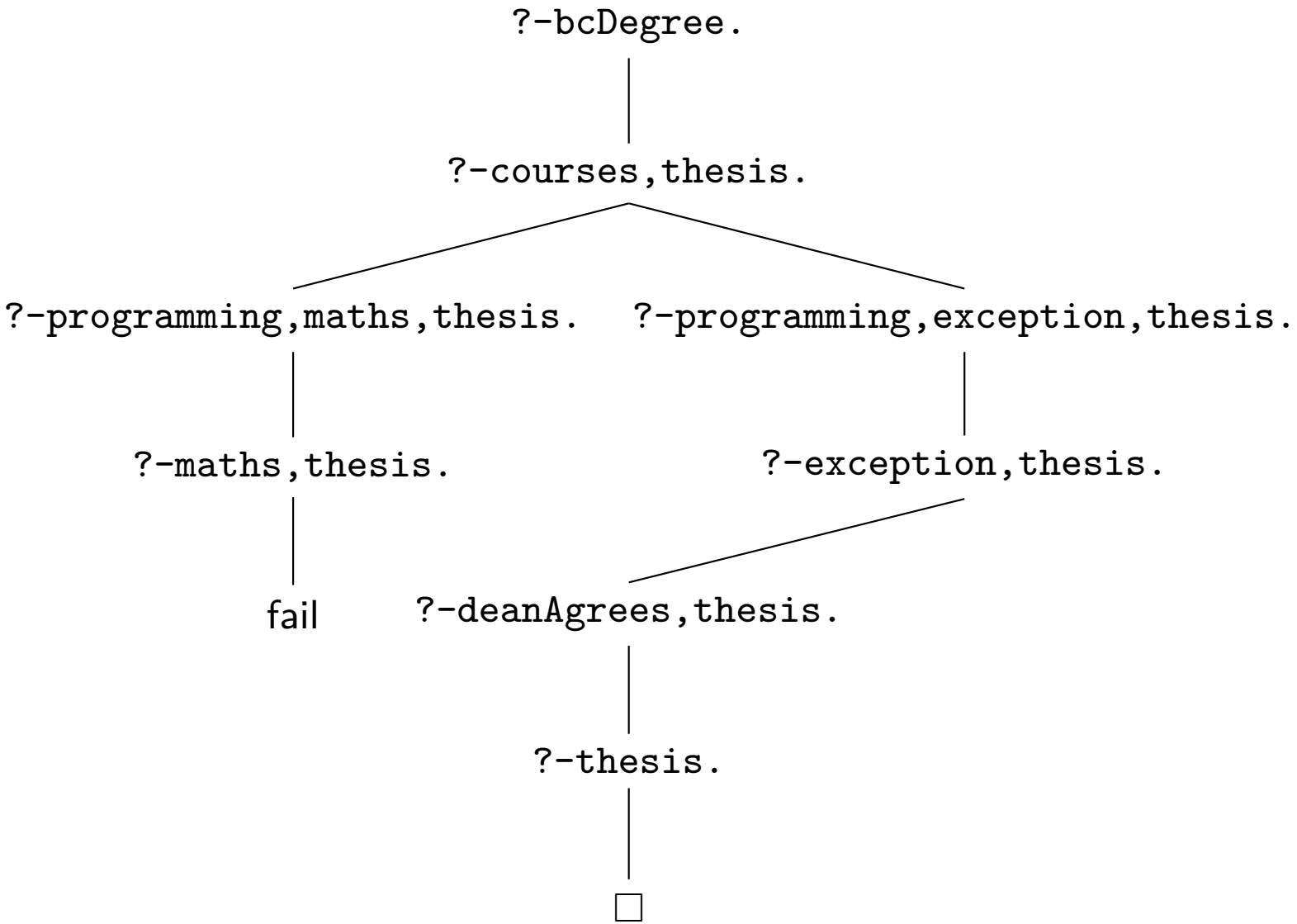
SLD stromy – jednoduchá ukázka (řešení)



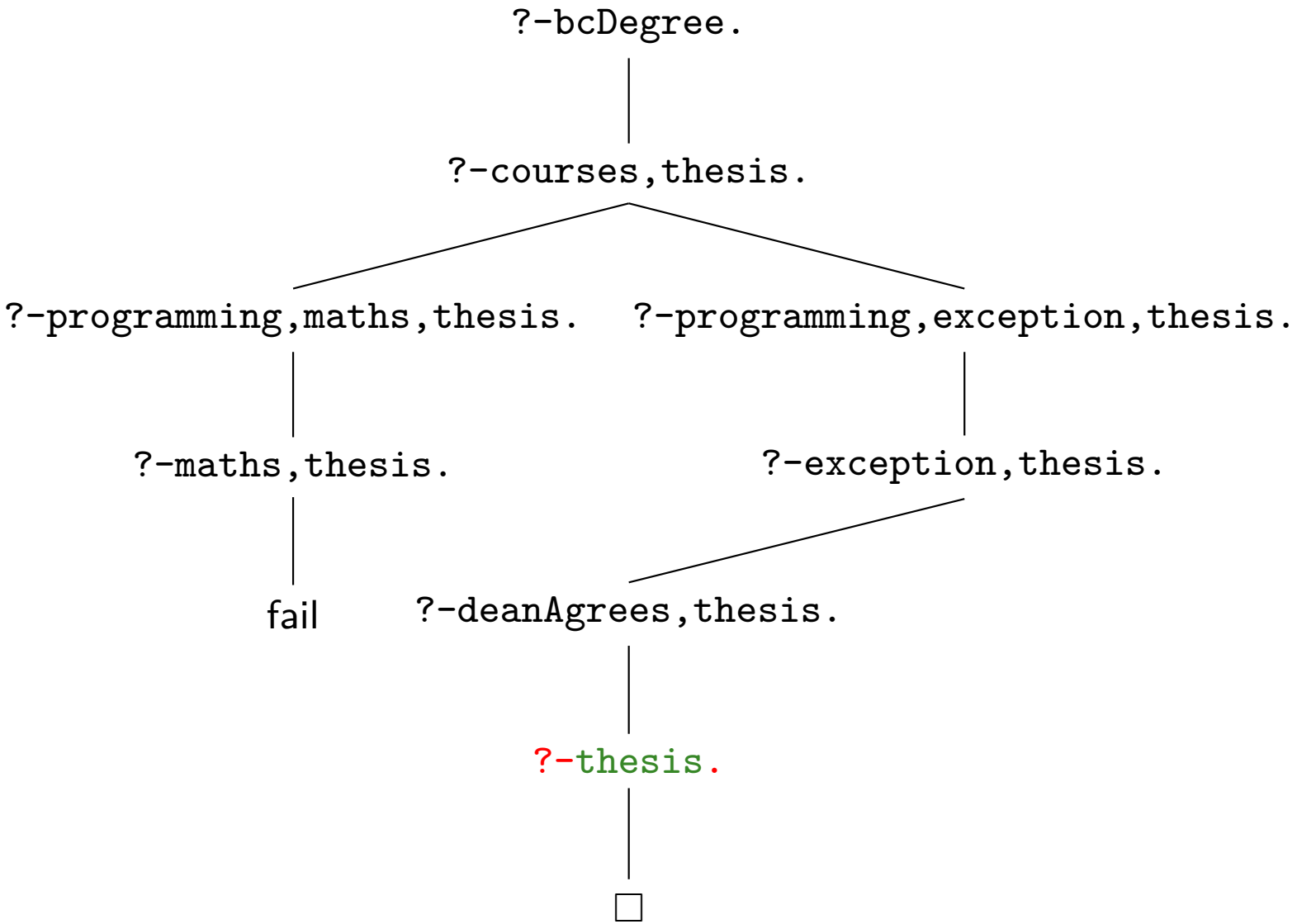
SLD stromy – jednoduchá ukázka (řešení)



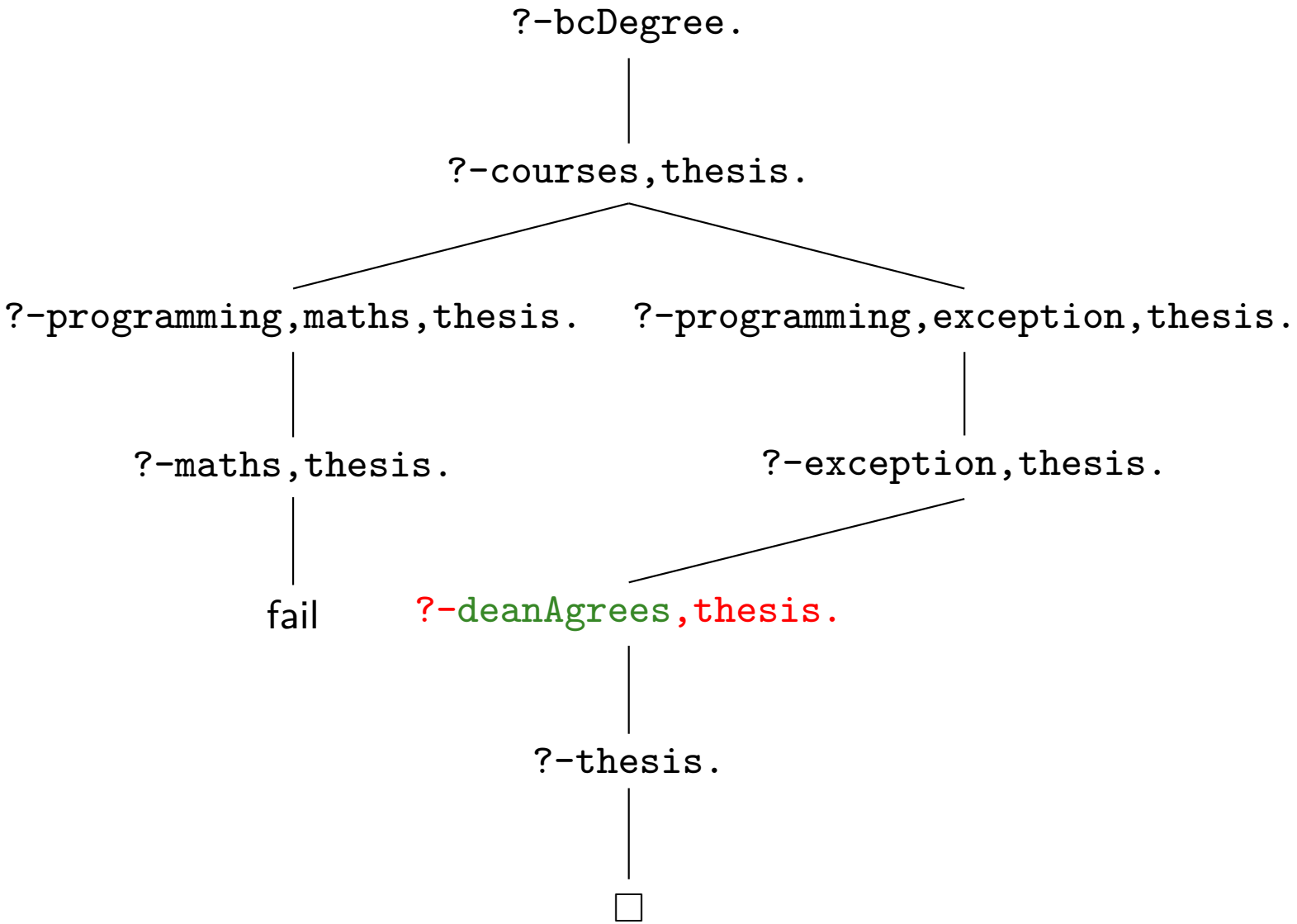
SLD stromy – jednoduchá ukázka (řešení)



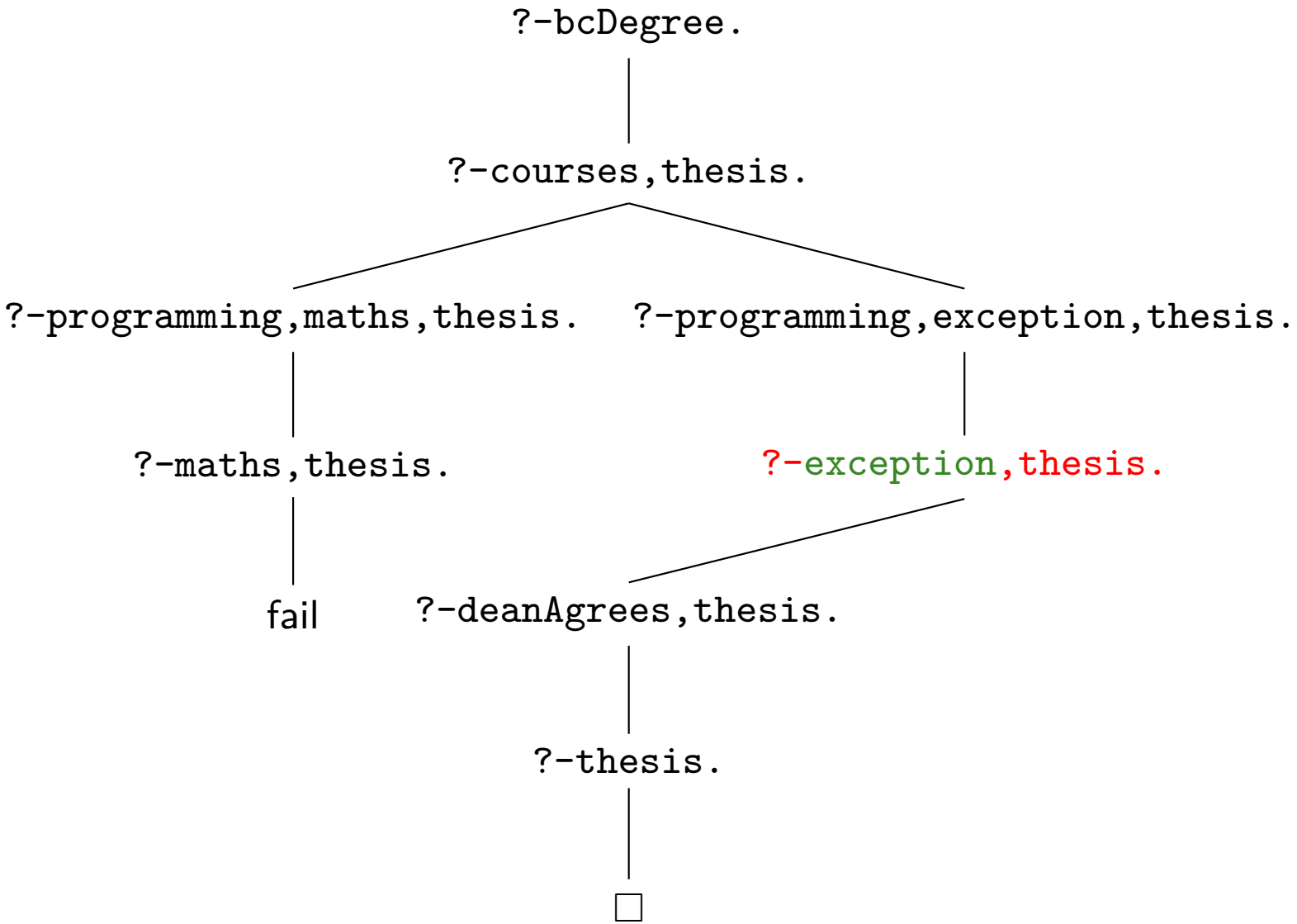
SLD stromy – jednoduchá ukázka (řešení)



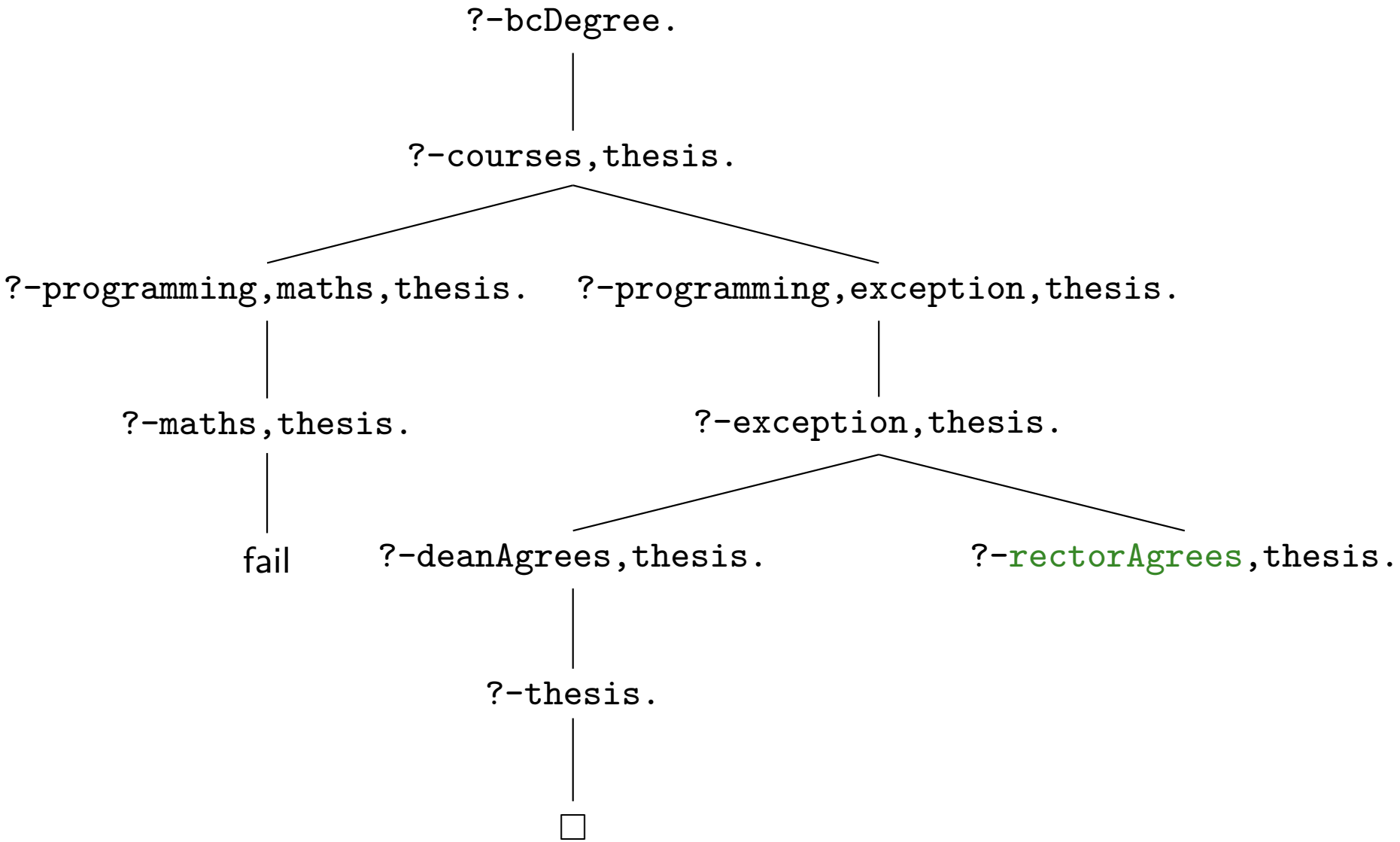
SLD stromy – jednoduchá ukázka (řešení)



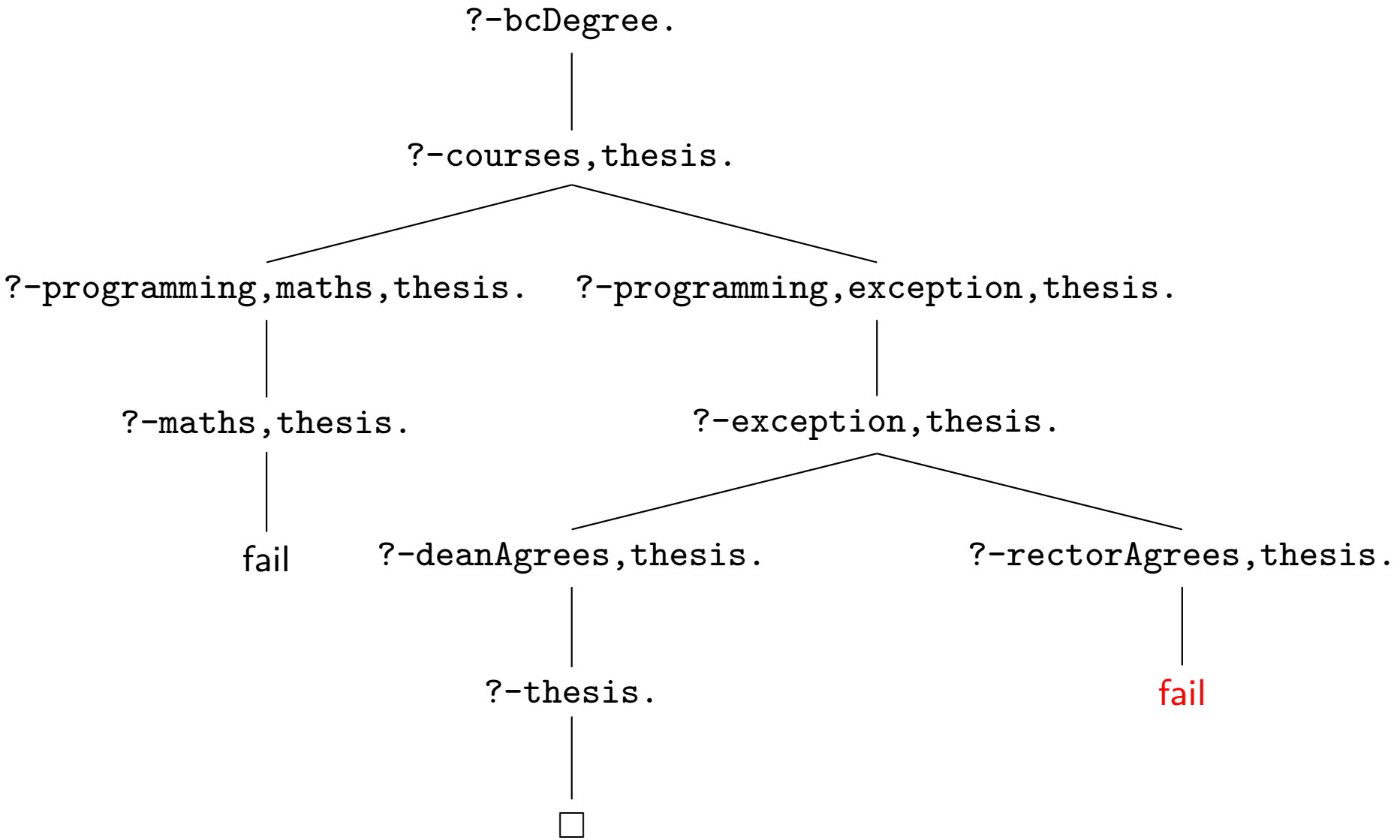
SLD stromy – jednoduchá ukázka (řešení)



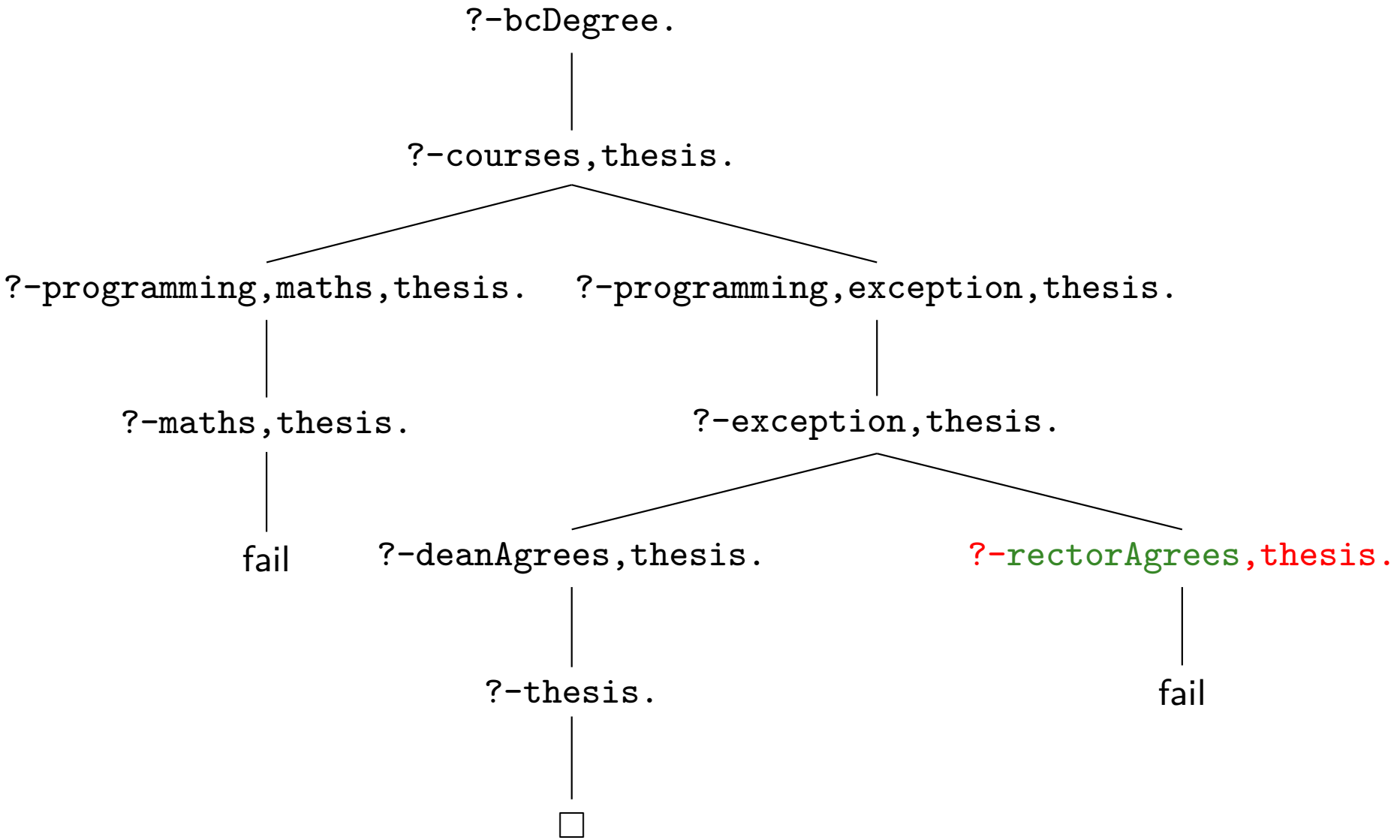
SLD stromy – jednoduchá ukázka (řešení)



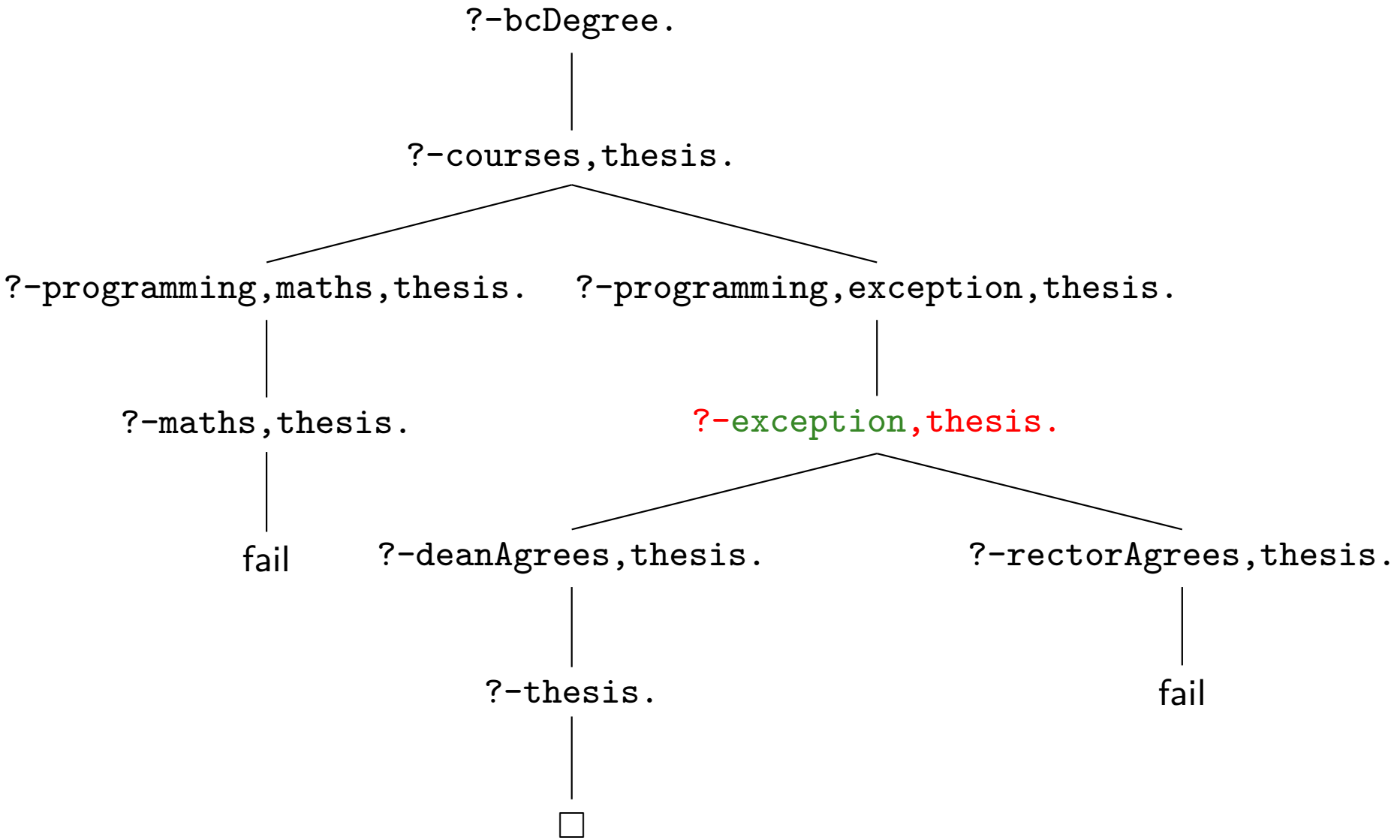
SLD stromy – jednoduchá ukázka (řešení)



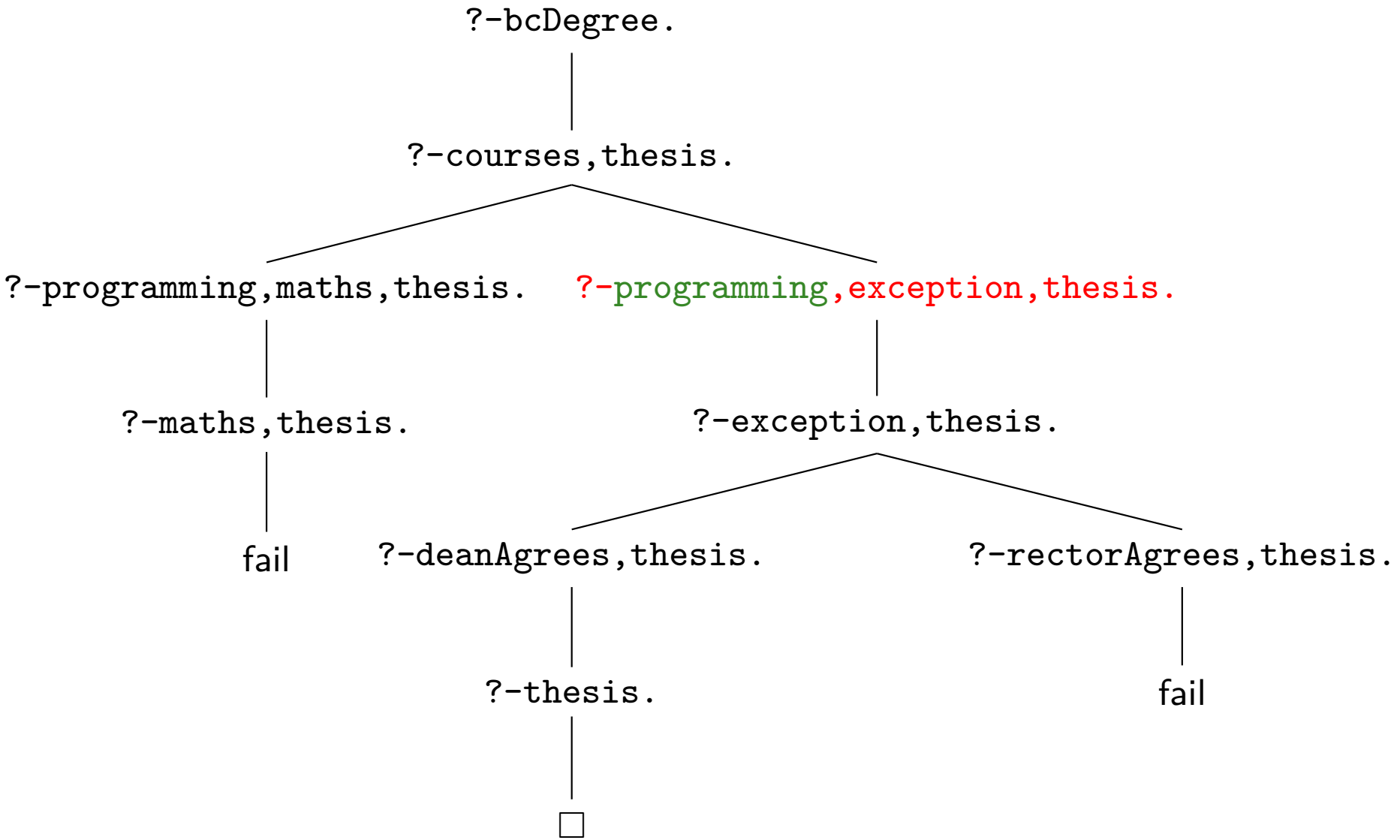
SLD stromy – jednoduchá ukázka (řešení)



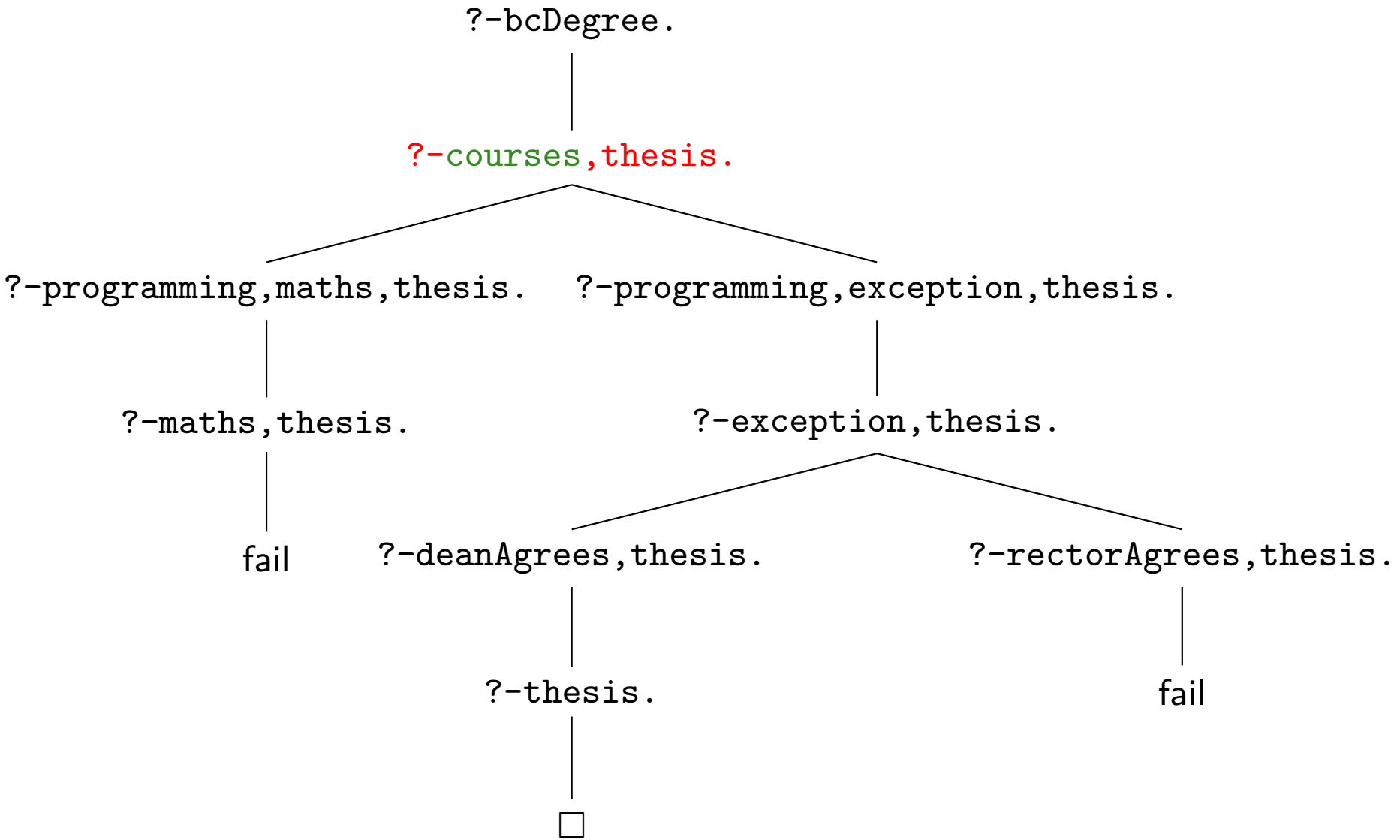
SLD stromy – jednoduchá ukázka (řešení)



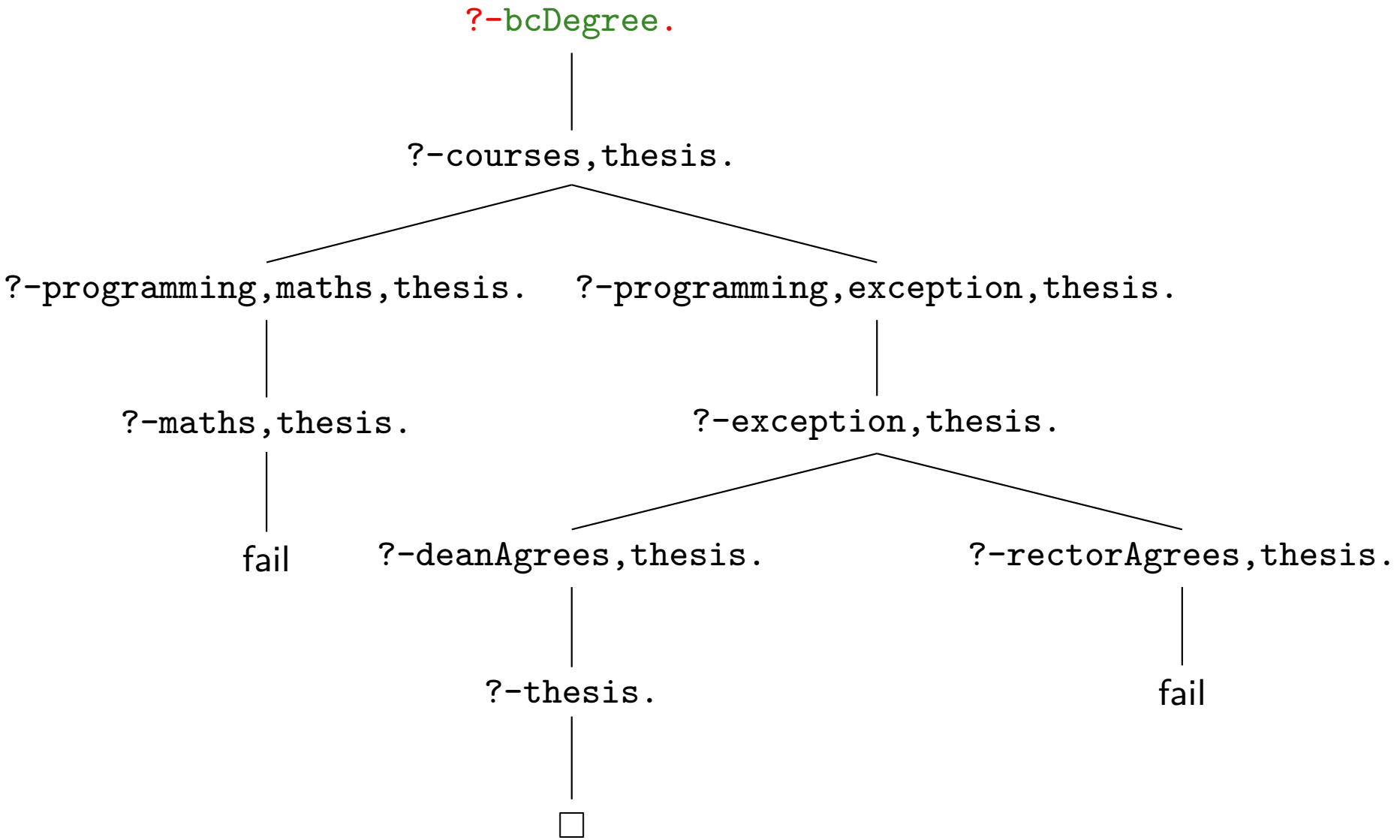
SLD stromy – jednoduchá ukázka (řešení)



SLD stromy – jednoduchá ukázka (řešení)



SLD stromy – jednoduchá ukázka (řešení)



SLD stromy – složitější případy I.

Uvažme následující pravidla a fakta o studentech, kteří se navzájem doučují a z nichž někteří rozumí Prologu.

```
1 teaches(martin, michal).
2 teaches(martin, tereza).
3 teaches(michal, jakub).
4
5 understandsProlog(martin).
6 understandsProlog(X) :- understandsProlog(Y),
      teaches(Y, X).
```

Zjistěte, kdo podle zadaných pravidel rozumí Prologu, tj. vyhodnoťte dotaz `?- understandsProlog(X).` a nakreslete odpovídající SLD strom.

SLD stromy – složitější případy II.

Jak se výsledný strom změní drobnou změnou:
Predikát `understandsProlog/1` má prohozená pravidla?

```
1 teaches(martin, michal).
2 teaches(martin, tereza).
3 teaches(michal, jakub).
4
5 understandsProlog(X) :-
6     understandsProlog(Y), teaches(Y, X).
7 understandsProlog(martin).
```

SLD stromy – složitější případy III.

Jak se výsledný strom změní drobnou změnou:
Predikát `understandsProlog/1` má prohozené podcíle v pravidle?

```
1 teaches(martin, michal).
2 teaches(martin, tereza).
3 teaches(michal, jakub).
4
5 understandsProlog(martin).
6 understandsProlog(X) :-
7     teaches(Y, X), understandsProlog(Y).
```

SLD stromy – složitější případy IV.

Jak se výsledný strom změní drobnou změnou:
Predikát `understandsProlog/1` má prohozená pravidla a podcíle?

```
1 teaches(martin, michal).
2 teaches(martin, tereza).
3 teaches(michal, jakub).
4
5 understandsProlog(X) :-
6     teaches(Y, X), understandsProlog(Y).
7 understandsProlog(martin).
```

Tipy pro psaní rekurzivních predikátů

- Fakta pište dříve než pravidla.
- Jednoduché podmínky pište co nejdřív.
- Nerekurzivní volání napište dříve než rekurzivní.
- Netvořte levorekurzivní pravidla.
- Pokuste se využít optimalizaci posledního volání (tail rekurzi).