# Technika a styl programování v Prologu

# Technika a styl programování v Prologu

- Styl programování v Prologu
  - některá pravidla správného stylu
  - správný vs. špatný styl
  - komentáře
- Ladění
- Efektivita

# Styl programování v Prologu I.

- Cílem stylistických konvencí je
  - redukce nebezpečí programovacích chyb
  - psaní čitelných a srozumitelných programů, které se dobře ladí a modifikují

# Styl programování v Prologu I.

- Cílem stylistických konvencí je
  - redukce nebezpečí programovacích chyb
  - psaní čitelných a srozumitelných programů, které se dobře ladí a modifikují
- Některá pravidla správného stylu
  - krátké klauzule
  - krátké procedury; dlouhé procedury pouze s uniformní strukturou (tabulka)

# Styl programování v Prologu I.

- Cílem stylistických konvencí je
  - redukce nebezpečí programovacích chyb
  - psaní čitelných a srozumitelných programů, které se dobře ladí a modifikují
- Některá pravidla správného stylu
  - krátké klauzule
  - krátké procedury; dlouhé procedury pouze s uniformní strukturou (tabulka)
  - klauzule se základními (hraničními) případy psát před rekurzivními klauzulemi
  - vhodná jmena procedur a proměnných
    - nepoužívat seznamy (`[...]`) nebo závorky (`{...}`, `(...)`) pro termy pevné arity
  - vstupní argumenty psát před výstupními

# Styl programování v Prologu I.

- Cílem stylistických konvencí je
  - redukce nebezpečí programovacích chyb
  - psaní čitelných a srozumitelných programů, které se dobře ladí a modifikují
- Některá pravidla správného stylu
  - krátké klauzule
  - krátké procedury; dlouhé procedury pouze s uniformní strukturou (tabulka)
  - klauzule se základními (hraničními) případy psát před rekurzivními klauzulemi
  - vhodná jmena procedur a proměnných
    - nepoužívat seznamy (`[...]`) nebo závorky (`{...}`, `(...)`) pro termy pevné arity
  - vstupní argumenty psát před výstupními
  - **struktura programu** – **jednotné konvence** v rámci celého programu, např.
    - mezery, prázdné řádky, odsazení
    - klauzule stejné procedury na jednom místě; prázdné řádky mezi klauzulemi; každý cíl na zvláštním řádku

# Správný styl programování

- konstrukce setříděného seznamu `Seznam3 ze setříděných seznamů Seznam1, Seznam2: merge( Seznam1, Seznam2, Seznam3 )`
- `merge( [2,4,7], [1,3,4,8], [1,2,3,4,4,7,8] )`

# Správný styl programování

- konstrukce setříděného seznamu `Seznam3` ze setříděných seznamů `Seznam1`, `Seznam2`: `merge( Seznam1, Seznam2, Seznam3 )`
- `merge( [2,4,7], [1,3,4,8], [1,2,3,4,4,7,8] )`
- `merge( [], Seznam, Seznam ) :-`

# Správný styl programování

- konstrukce setříděného seznamu `Seznam3 ze setříděných seznamů Seznam1, Seznam2: merge( Seznam1, Seznam2, Seznam3 )`
- `merge( [2,4,7], [1,3,4,8], [1,2,3,4,4,7,8] )`
- ```
  merge( [], Seznam, Seznam ) :-
         !.                              % prevence redundantních řešení
  ```
  ```

# Správný styl programování

- konstrukce setříděného seznamu `Seznam3` ze setříděných seznamů `Seznam1`, `Seznam2`: `merge( Seznam1, Seznam2, Seznam3 )`
- `merge( [2,4,7], [1,3,4,8], [1,2,3,4,4,7,8] )`
- ```
  merge( [], Seznam, Seznam ) :-
        !.                                  % prevence redundantních řešení
  merge( Seznam, [], Seznam ).
  ```
  ```

# Správný styl programování

- konstrukce setříděného seznamu `Seznam3` ze setříděných seznamů `Seznam1`, `Seznam2`: `merge( Seznam1, Seznam2, Seznam3 )`
- `merge( [2,4,7], [1,3,4,8], [1,2,3,4,4,7,8] )`
- 

```
merge( [], Seznam, Seznam ) :-
        !.                              % prevence redundantních řešení

merge( Seznam, [], Seznam ).

merge( [X|Telo1], [Y|Telo2], [X|Telo3] ) :-
```

# Správný styl programování

- konstrukce setříděného seznamu `Seznam3` ze setříděných seznamů `Seznam1`, `Seznam2`: `merge( Seznam1, Seznam2, Seznam3 )`
- `merge( [2,4,7], [1,3,4,8], [1,2,3,4,4,7,8] )`
- 
```
merge( [], Seznam, Seznam ) :-
        !.                         % prevence redundantních řešení

merge( Seznam, [], Seznam ).

merge( [X|Telo1], [Y|Telo2], [X|Telo3] ) :-
        X < Y, !,
```

# Správný styl programování

- konstrukce setříděného seznamu `Seznam3 ze setříděných seznamů Seznam1, Seznam2: merge( Seznam1, Seznam2, Seznam3 )`
- `merge( [2,4,7], [1,3,4,8], [1,2,3,4,4,7,8] )`
- 

```
merge( [], Seznam, Seznam ) :-
        !.                                % prevence redundantních řešení

merge( Seznam, [], Seznam ).

merge( [X|Telo1], [Y|Telo2], [X|Telo3] ) :-
        X < Y, !,
        merge( Telo1, [Y|Telo2], Telo3 ).
```

# Správný styl programování

- konstrukce setříděného seznamu `Seznam3` ze setříděných seznamů `Seznam1`, `Seznam2`: `merge( Seznam1, Seznam2, Seznam3 )`
- `merge( [2,4,7], [1,3,4,8], [1,2,3,4,4,7,8] )`
- 

```
merge( [], Seznam, Seznam ) :-
        !.                                  % prevence redundantních řešení

merge( Seznam, [], Seznam ).

merge( [X|Telo1], [Y|Telo2], [X|Telo3] ) :-
        X < Y, !,
        merge( Telo1, [Y|Telo2], Telo3 ).

merge( Seznam1, [Y|Telo2], [Y|Telo3] ) :-
```

# Správný styl programování

- konstrukce setříděného seznamu `Seznam3` ze `setříděných seznamů`
  `Seznam1, Seznam2: merge( Seznam1, Seznam2, Seznam3 )`
- `merge( [2,4,7], [1,3,4,8], [1,2,3,4,4,7,8] )`
- ```
  merge( [], Seznam, Seznam ) :-
         !.                                  % prevence redundantních řešení

  merge( Seznam, [], Seznam ).

  merge( [X|Telo1], [Y|Telo2], [X|Telo3] ) :-
         X < Y, !,
         merge( Telo1, [Y|Telo2], Telo3 ).

  merge( Seznam1, [Y|Telo2], [Y|Telo3] ) :-
         merge( Seznam1, Telo2, Telo3 ).
  ```
  ```

# Špatný styl programování

```
merge( S1, S2, S3 ) :-
 S1 = [], !, S3 = S2;            %  první seznam je prázdný
 S2 = [], !, S3 = S1;            %  druhý seznam je prázdný
 S1 = [X|T1],
 S2 = [Y|T2],
 ( X < Y, !,
   Z = X,                        %  Z je hlava seznamu  S3
   merge( T1, S2, T3 );
   Z = Y,
   merge( S1, T2, T3) ),
 S3 = [ Z | T3 ].
```

# Styl programování v Prologu II.

- **Středník** „;" může způsobit nesrozumitelnost klauzule
  - nedávat středník na konec řádku, používat závorky
  - v některých případech: rozdělení klauzle se středníkem do více klauzulí

# Styl programování v Prologu II.

- **Středník** „;“ může způsobit nesrozumitelnost klauzule
  - nedávat středník na konec řádku, používat závorky
  - v některých případech: rozdělení klauzle se středníkem do více klauzulí
- Opatrné používání **operátoru řezu**
  - preferovat použití zeleného řezu (nemění deklarativní sémantiku)
  - červený řez používat v jasně definovaných konstruktech

    negace: `P, !, fail; true` `\+ P`

    alternativy: `Podminka, !, Cil1 ; Cil2` `Podminka -> Cil1 ; Cil2`

# Styl programování v Prologu II.

- **Středník** „;” může způsobit nesrozumitelnost klauzule
  - nedávat středník na konec řádku, používat závorky
  - v některých případech: rozdělení klauzle se středníkem do více klauzulí
- Opatrné používání **operátoru řezu**
  - preferovat použití zeleného řezu (nemění deklarativní sémantiku)
  - červený řez používat v jasně definovaných konstruktech

    negace: `P, !, fail; true` `\+ P`

    alternativy: `Podminka, !, Cil1 ; Cil2` `Podminka -> Cil1 ; Cil2`
- Opatrné používání **negace** „\+”
  - negace jako neúspěch: negace není ekvivalentní negaci v matematické logice

# Styl programování v Prologu II.

- **Středník** „;" může způsobit nesrozumitelnost klauzule
  - nedávat středník na konec řádku, používat závorky
  - v některých případech: rozdělení klauzle se středníkem do více klauzulí
- Opatrné používání **operátoru řezu**
  - preferovat použití zeleného řezu (nemění deklarativní sémantiku)
  - červený řez používat v jasně definovaných konstruktech

    negace: `P, !, fail; true` `\+ P`

    alternativy: `Podminka, !, Cil1 ; Cil2` `Podminka -> Cil1 ; Cil2`
- Opatrné používání **negace** „\+"
  - negace jako neúspěch: negace není ekvivalentní negaci v matematické logice
- Pozor na **assert a retract**: snižuji transparentnost chování programu

# Dokumentace a komentáře

- co program dělá, jak ho používat (jaký cíl spustit a jaké jsou očekávané výsledky), příklad použití

# Dokumentace a komentáře

- co program dělá, jak ho používat (jaký cíl spustit a jaké jsou očekávané výsledky), příklad použití
- které predikáty jsou hlavní (*top-level*)

# Dokumentace a komentáře

- co program dělá, jak ho používat (jaký cíl spustit a jaké jsou očekávané výsledky), příklad použití
- které predikáty jsou hlavní (*top-level*)
- jak jsou hlavní koncepty (objekty) reprezentovány

# Dokumentace a komentáře

- co program dělá, jak ho používat (jaký cíl spustit a jaké jsou očekávané výsledky), příklad použití
- které predikáty jsou hlavní (*top-level*)
- jak jsou hlavní koncepty (objekty) reprezentovány
- doba výpočtu a paměťové nároky

# Dokumentace a komentáře

- co program dělá, jak ho používat (jaký cíl spustit a jaké jsou očekávané výsledky), příklad použití
- které predikáty jsou hlavní (*top-level*)
- jak jsou hlavní koncepty (objekty) reprezentovány
- doba výpočtu a paměťové nároky
- jaké jsou limitace programu

# Dokumentace a komentáře

- co program dělá, jak ho používat (jaký cíl spustit a jaké jsou očekávané výsledky), příklad použití
- které predikáty jsou hlavní (*top-level*)
- jak jsou hlavní koncepty (objekty) reprezentovány
- doba výpočtu a paměťové nároky
- jaké jsou limitace programu
- zda jsou použity nějaké speciální rysy závislé na systému

# Dokumentace a komentáře

- co program dělá, jak ho používat (jaký cíl spustit a jaké jsou očekávané výsledky), příklad použití
- které predikáty jsou hlavní (*top-level*)
- jak jsou hlavní koncepty (objekty) reprezentovány
- doba výpočtu a paměťové nároky
- jaké jsou limitace programu
- zda jsou použity nějaké speciální rysy závislé na systému
- jaký je význam predikátů v programu, jaké jsou jejich argumenty, které jsou vstupní a které výstupní (pokud víme)
  - vstupní argumenty „+", výstupní „−" `merge( +Seznam1, +Seznam2, -Seznam3 )`
  - `JmenoPredikatu/Arita` `merge/3`

# Dokumentace a komentáře

- co program dělá, jak ho používat (jaký cíl spustit a jaké jsou očekávané výsledky), příklad použití
- které predikáty jsou hlavní (*top-level*)
- jak jsou hlavní koncepty (objekty) reprezentovány
- doba výpočtu a paměťové nároky
- jaké jsou limitace programu
- zda jsou použity nějaké speciální rysy závislé na systému
- jaký je význam predikátů v programu, jaké jsou jejich argumenty, které jsou vstupní a které výstupní (pokud víme)
  - vstupní argumenty „+", výstupní „–" `merge( +Seznam1, +Seznam2, -Seznam3 )`
  - `JmenoPredikatu/Arita` `merge/3`
- algoritmické a implementační podrobnosti

# Ladění

- Přepínače na trasování: `trace/0`, `notrace/0`
- Trasování specifického predikátu: `spy/1`, `nospy/1`
  - `spy( merge/3 )`
- `debug/0`, `nodebug/0`: pro trasování pouze predikátů zadaných spy/1

# Ladění

- Přepínače na trasování: `trace/0`, `notrace/0`
- Trasování specifického predikátu: `spy/1`, `nospy/1`
  - `spy( merge/3 )`
- `debug/0`, `nodebug/0`: pro trasování pouze predikátů zadaných spy/1
- Libovolná část programu může být spuštěna

  zadáním vhodného dotazu: **trasování cíle**
  - vstupní informace: jméno predikátu, hodnoty argumentů při volání
  - výstupní informace
    - při úspěchu hodnoty argumentů splňující cíl
    - při neúspěchu indikace chyby
  - nové vyvolání přes ";": stejný cíl je volán při backtrackingu

# Krabičkový (4-branový) model

- Vizualizace řídícího toku (backtrackingu) na úrovni predikátu
  - `Call`: volání cíle
  - `Exit`: úspěšné ukončení volání cíle
  - `Fail`: volání cíle neuspělo
  - Redo: jeden z následujících cílů neuspěl a systém backtrackuje, aby nalezl alternativy k předchozímu řešení

```
                  *-------------------------------------------*
   Call           |                                           |     Exit
-----------------> +   predek( X, Z ) :- rodic( X, Z ).       + --------->
                  |                                           |
                  |   predek( X, Z ) :- rodic( X, Y ),        |
<----------------- +                     predek( Y, Z ).       + <---------
   Fail           |                                           |      Redo
                  *-------------------------------------------*
```

# Příklad: trasování

```
a(X) :- nonvar(X).
a(X) :- c(X).
a(X) :- d(X).
c(1).
d(2).
```

```
         *-------------------*
 Call   |                   |  Exit
------> + a(X) :- nonvar(X).| ------>
        | a(X) :- c(X).     |
<------ + a(X) :- d(X).     + <------
  Fail  |                   |   Redo
        *-------------------*
```

# Příklad: trasování

```
a(X) :- nonvar(X).
a(X) :- c(X).
a(X) :- d(X).
c(1).
d(2).
```

```
         *-------------------*
 Call    |                   |  Exit
------> + a(X) :- nonvar(X).| ------>
        | a(X) :- c(X).      |
<------ + a(X) :- d(X).      + <------
  Fail  |                    |   Redo
        *--------------------*
```

```
| ?-  a(X).
        1      1 Call: a(_463) ?
        2      2 Call: nonvar(_463) ?
        2      2 Fail: nonvar(_463) ?
```

# Příklad: trasování

```
a(X) :- nonvar(X).
a(X) :- c(X).
a(X) :- d(X).
c(1).
d(2).
```

```
         *-------------------*
 Call    |                   |  Exit
------> + a(X) :- nonvar(X).| ------>
        | a(X) :- c(X).      |
<------ + a(X) :- d(X).      + <------
  Fail  |                    |   Redo
        *--------------------*
```

```
| ?-  a(X).
        1      1 Call: a(_463) ?
        2      2 Call: nonvar(_463) ?
        2      2 Fail: nonvar(_463) ?
        3      2 Call: c(_463) ?
        3      2 Exit: c(1) ?
?       1      1 Exit: a(1) ?
X = 1 ?
```

# Příklad: trasování

```
a(X) :- nonvar(X).
a(X) :- c(X).
a(X) :- d(X).
c(1).
d(2).
```

```
          *-------------------*
 Call     |                   |  Exit
------> + a(X) :- nonvar(X).| ------>
        | a(X) :- c(X).       |
<------ + a(X) :- d(X).     + <------
  Fail  |                   |   Redo
        *-------------------*
```

```
| ?-  a(X).
        1      1 Call: a(_463) ?
        2      2 Call: nonvar(_463) ?
        2      2 Fail: nonvar(_463) ?
        3      2 Call: c(_463) ?
        3      2 Exit: c(1) ?
?       1      1 Exit: a(1) ?
X = 1 ? ;
        1      1 Redo: a(1) ?
        4      2 Call: d(_463) ?
```

# Příklad: trasování

```
a(X) :- nonvar(X).
a(X) :- c(X).
a(X) :- d(X).
c(1).
d(2).
```

```
          *-------------------*
 Call    |                    |   Exit
------> + a(X) :- nonvar(X).| ------>
         | a(X) :- c(X).      |
<------ + a(X) :- d(X).       + <------
  Fail   |                    |    Redo
         *--------------------*
```

```
| ?-  a(X).
        1      1 Call: a(_463) ?
        2      2 Call: nonvar(_463) ?
        2      2 Fail: nonvar(_463) ?
        3      2 Call: c(_463) ?
        3      2 Exit: c(1) ?
?       1      1 Exit: a(1) ?
X = 1 ? ;
        1      1 Redo: a(1) ?
        4      2 Call: d(_463) ?
        4      2 Exit: d(2) ?
        1      1 Exit: a(2) ?
X = 2 ? ;
no
% trace
| ?-
```

# Efektivita

- Čas výpočtu, paměťové nároky, a také časové nároky na vývoj programu
  - u Prologu můžeme častěji narazit na problémy s časem výpočtu a pamětí
  - Prologovské aplikace redukují čas na vývoj
  - vhodnost pro symbolické, nenumerické výpočty se strukturovanými objekty a relacemi mezi nimi

# Efektivita

- Čas výpočtu, paměťové nároky, a také časové nároky na vývoj programu
  - u Prologu můžeme častěji narazit na problémy s časem výpočtu a pamětí
  - Prologovské aplikace redukují čas na vývoj
  - vhodnost pro symbolické, nenumerické výpočty se strukturovanými objekty a relacemi mezi nimi
- Pro zvýšení efektivity je nutno se zabývat **procedurálními aspekty**
  - **zlepšení efektivity při prohledávání**
    - odstranění zbytečného backtrackingu
    - zrušení provádění zbytečných alternativ co nejdříve
  - návrh **vhodnějších datových struktur**, které umožní efektivnější operace s objekty

# Zlepšení efektivity: základní techniky

- **Optimalizace posledního volání (LCO) a akumulátory**
- **Rozdílové seznamy** při spojování seznamů
- **Caching**: uložení vypočítaných výsledků do programové databáze

# Zlepšení efektivity: základní techniky

- **Optimalizace posledního volání (LCO) a akumulátory**
- **Rozdílové seznamy** při spojování seznamů
- **Caching**: uložení vypočítaných výsledků do programové databáze
- **Indexace** podle prvního argumentu
  - např. v SICStus Prologu
  - při volání predikátu s prvním nainstaniovaným argumentem se používá hašovací tabulka zpřístupňující pouze odpovídající klauzule
  - `zamestnanec( Prijmeni, KrestniJmeno, Oddeleni, ...)`
  - `seznamy( [], ...) :- ... .`

    `seznamy( [H|T], ...) :- ... .`

# Zlepšení efektivity: základní techniky

- **Optimalizace posledního volání (LCO) a akumulátory**
- **Rozdílové seznamy** při spojování seznamů
- **Caching**: uložení vypočítaných výsledků do programové databáze
- **Indexace** podle prvního argumentu
  - např. v SICStus Prologu
  - při volání predikátu s prvním nainstaniovaným argumentem se používá hašovací tabulka zpřístupňující pouze odpovídající klauzule
  - `zamestnanec( Prijmeni, KrestniJmeno, Oddeleni, ...)`
  - `seznamy( [], ...) :- ... .`

    `seznamy( [H|T], ...) :- ... .`
- **Determinismus**:
  - rozhodnout, které klauzule mají uspět vícekrát, ověřit požadovaný determinismus

# Predikátová logika 1.řádu

# Teorie logického programování

- PROLOG: PROgramming in LOGic, část predikátové logiky 1.řádu
  - fakta: `rodic(petr,petrik)`, $\forall X\, a(X)$
  - klauzule: $\forall X\, \forall Y$ `rodic(X,Y)` $\Rightarrow$ `predek(X,Y)`

# Teorie logického programování

- PROLOG: PROgramming in LOGic, část predikátové logiky 1.řádu
  - fakta: `rodic(petr,petrik)`, $\forall X\, a(X)$
  - klauzule: $\forall X\, \forall Y$ `rodic(X,Y)` $\Rightarrow$ `predek(X,Y)`
- Predikátová logika I. řádu (PL1)
  - soubory objektů: lidé, čísla, body prostoru, . . .
  - syntaxe PL1, sémantika PL1, pravdivost a dokazatelnost

# Teorie logického programování

- PROLOG: PROgramming in LOGic, část predikátové logiky 1.řádu
  - fakta: `rodic(petr,petrik)`, $\forall X\, a(X)$
  - klauzule: $\forall X\, \forall Y$ `rodic(X,Y)` $\Rightarrow$ `predek(X,Y)`
- Predikátová logika I. řádu (PL1)
  - soubory objektů: lidé, čísla, body prostoru, . . .
  - syntaxe PL1, sémantika PL1, pravdivost a dokazatelnost
- Rezoluce ve výrokové logice, v PL1
  - dokazovací metoda
- Rezoluce v logickém programování
- Backtracking, řez, negace vs. rezoluce

# Predikátová logika I. řádu (PL1)

Abeceda $\mathcal{A}$ **jazyka** $\mathcal{L}$ PL1 se skládá ze symbolů:

- **proměnné** X, Y, . . . označují libovolný objekt z daného oboru

# Predikátová logika I. řádu (PL1)

Abeceda $\mathcal{A}$ **jazyka** $\mathcal{L}$ PL1 se skládá ze symbolů:

- **proměnné** X, Y, . . . označují libovolný objekt z daného oboru
- **funkční symboly** f, g, . . . označují operace (příklad: +, $\times$ )
  - **arita** = počet argumentů, $n$-**ární** symbol, značíme f/n
  - nulární funkční symboly – **konstanty**: označují význačné objekty (příklad: 0, 1, . . . )

# Predikátová logika I. řádu (PL1)

Abeceda $\mathcal{A}$ **jazyka** $\mathcal{L}$ PL1 se skládá ze symbolů:

- **proměnné** X, Y, ... označují libovolný objekt z daného oboru
- **funkční symboly** f, g, ... označují operace (příklad: +, $\times$ )
  - **arita** = počet argumentů, $n$-**ární** symbol, značíme f/n
  - nulární funkční symboly – **konstanty**: označují význačné objekty (příklad: 0, 1, ...)
- **predikátové symboly** p,q, ... pro vyjádření vlastností a vztahů mezi objekty
  - **arita** = počet argumentů, $n$-**ární** symbol, značíme p/n     příklad: $<$, $\in$

# Predikátová logika I. řádu (PL1)

Abeceda $\mathcal{A}$ **jazyka** $\mathcal{L}$ PL1 se skládá ze symbolů:

- **proměnné** X, Y, ... označují libovolný objekt z daného oboru
- **funkční symboly** f, g, ... označují operace (příklad: +, $\times$ )
  - **arita** = počet argumentů, $n$-**ární** symbol, značíme f/n
  - nulární funkční symboly – **konstanty**: označují význačné objekty (příklad: 0, 1, ...)
- **predikátové symboly** p,q, ... pro vyjádření vlastností a vztahů mezi objekty
  - **arita** = počet argumentů, $n$-**ární** symbol, značíme p/n příklad: $<$, $\in$
- **logické spojky** $\wedge$, $\vee$, $\neg$, $\Rightarrow$, $\equiv$

# Predikátová logika I. řádu (PL1)

Abeceda $\mathcal{A}$ **jazyka** $\mathcal{L}$ PL1 se skládá ze symbolů:

- **proměnné** X, Y, . . . označují libovolný objekt z daného oboru
- **funkční symboly** f, g, . . . označují operace (příklad: +, $\times$ )
  - **arita** = počet argumentů, $n$-**ární** symbol, značíme f/n
  - nulární funkční symboly – **konstanty**: označují význačné objekty (příklad: 0, 1, . . . )
- **predikátové symboly** p,q, . . . pro vyjádření vlastností a vztahů mezi objekty
  - **arita** = počet argumentů, $n$-**ární** symbol, značíme p/n      příklad: $<$, $\in$
- **logické spojky** $\wedge$, $\vee$, $\neg$, $\Rightarrow$, $\equiv$
- **kvantifikátory** $\forall$, $\exists$
  - logika I. řádu používá **kvantifikaci pouze pro individua** (odlišnost od logik vyššího řádu)
  - v logice 1. řádu nelze: v $\mathbb{R} : \forall A \subseteq \mathbb{R}, \forall f : \mathbb{R} \rightarrow \mathbb{R}$

# Predikátová logika I. řádu (PL1)

Abeceda $\mathcal{A}$ **jazyka** $\mathcal{L}$ PL1 se skládá ze symbolů:

- **proměnné** X, Y, ... označují libovolný objekt z daného oboru
- **funkční symboly** f, g, ... označují operace (příklad: +, $\times$ )
  - **arita** = počet argumentů, $n$-**ární** symbol, značíme f/n
  - nulární funkční symboly – **konstanty**: označují význačné objekty (příklad: 0, 1, ...)
- **predikátové symboly** p,q, ... pro vyjádření vlastností a vztahů mezi objekty
  - **arita** = počet argumentů, $n$-**ární** symbol, značíme p/n     příklad: $<$, $\in$
- **logické spojky** $\wedge$, $\vee$, $\neg$, $\Rightarrow$, $\equiv$
- **kvantifikátory** $\forall$, $\exists$
  - logika I. řádu používá **kvantifikaci pouze pro individua** (odlišnost od logik vyššího řádu)
  - v logice 1. řádu nelze: v $\mathbb{R} : \forall A \subseteq \mathbb{R}, \forall f : \mathbb{R} \to \mathbb{R}$
- **závorky**: ),(

# Jazyky PL1

- Specifikace jazyka $\mathcal{L}$ je definována funkčními a predikátovými symboly
  symboly tedy určují oblast, kterou jazyk popisuje
- **Jazyky s rovností**: obsahují predikátový symbol pro rovnost „=”

# Jazyky PL1

- Specifikace jazyka $\mathcal{L}$ je definována funkčními a predikátovými symboly
  symboly tedy určují oblast, kterou jazyk popisuje
- **Jazyky s rovností**: obsahují predikátový symbol pro rovnost „=”

## Příklady

- jazyk teorie uspořádání
  - jazyk s =, binární prediátový symbol $<$

# Jazyky PL1

- Specifikace jazyka $\mathcal{L}$ je definována funkčními a predikátovými symboly
  symboly tedy určují oblast, kterou jazyk popisuje
- **Jazyky s rovností**: obsahují predikátový symbol pro rovnost „=”

**Příklady**

- jazyk teorie uspořádání
  - jazyk s =, binární prediátový symbol $<$
- jazyk teorie množin
  - jazyk s =, binární predikátový symbol $\in$

# Jazyky PL1

- Specifikace jazyka $\mathcal{L}$ je definována funkčními a predikátovými symboly
  symboly tedy určují oblast, kterou jazyk popisuje
- **Jazyky s rovností**: obsahují predikátový symbol pro rovnost „=”

## Příklady

- jazyk teorie uspořádání
  - jazyk s =, binární prediátový symbol $<$
- jazyk teorie množin
  - jazyk s =, binární predikátový symbol $\in$
- jazyk elementární aritmetiky
  - jazyk s =, nulární funkční symbol 0 pro nulu,
    unární funkční symbol s pro operaci následníka,
    binární funkční symboly pro sčítání + a násobení $\times$

# Term, atomická formule, formule

- **Term** nad abecedou $\mathcal{A}$
  - každá proměnná z $\mathcal{A}$ je term
  - je-li $f/n$ z $\mathcal{A}$ a $t_1, \ldots, t_n$ jsou termy, pak $f(t_1, \ldots, t_n)$ je také term
  - každý term vznikne konečným počtem užití přechozích kroků f( X, g(X,0) )

# Term, atomická formule, formule

- **Term** nad abecedou $\mathcal{A}$
  - každá proměnná z $\mathcal{A}$ je term
  - je-li $f/n$ z $\mathcal{A}$ a $t_1, \ldots, t_n$ jsou termy, pak $f(t_1, \ldots, t_n)$ je také term
  - každý term vznikne konečným počtem užití přechozích kroků f( X, g(X,0) )
- **Atomická formule (atom)** nad abecedou $\mathcal{A}$
  - je-li $p/n$ z $\mathcal{A}$ a $t_1, \ldots, t_n$ jsou termy, pak $p(t_1, \ldots, t_n)$ je atomická formule f(X) < g(X,0)

# Term, atomická formule, formule

- **Term** nad abecedou $\mathcal{A}$
  - každá proměnná z $\mathcal{A}$ je term
  - je-li $f/n$ z $\mathcal{A}$ a $t_1, \ldots, t_n$ jsou termy, pak $f(t_1, \ldots, t_n)$ je také term
  - každý term vznikne konečným počtem užití přechozích kroků f( X, g(X,0) )
- **Atomická formule (atom)** nad abecedou $\mathcal{A}$
  - je-li $p/n$ z $\mathcal{A}$ a $t_1, \ldots, t_n$ jsou termy, pak $p(t_1, \ldots, t_n)$ je atomická formule f(X) < g(X,0)
- **Formule** nad abecedou $\mathcal{A}$
  - každá atomická formule je formule
  - jsou-li $F$ a $G$ formule, pak také $(\neg F), (F \wedge G), (F \vee G), (F \Rightarrow G), (F \equiv G)$ jsou formule
  - je-li X proměnná a $F$ formule, pak také $(\forall \mathrm{X}\ F)$ a $(\exists \mathrm{X}\ F)$ jsou formule
  - každá formule vznikne konečným počtem užití přechozích kroků $(\exists \mathrm{X}\ ((f(\mathrm{X}) = 0) \wedge p(0)))$

# Interpretace

- **Interpretace** $\mathcal{I}$ jazyka $\mathcal{L}$ nad abecedou $\mathcal{A}$ je dána
  - neprázdnou množinou $\mathcal{D}$ (také značíme $|\mathcal{I}|$, nazývá se **univerzum**) a
  - zobrazením, které
    - každé konstantě $c \in \mathcal{A}$ přiřadí nějaký **prvek** $\mathcal{D}$
    - každému funkčnímu symbolu $f/n \in \mathcal{A}$ přiřadí $n$-ární **operaci** nad $\mathcal{D}$
    - každému predikátovému symbolu $p/n \in \mathcal{A}$ přiřadí $n$-ární **relaci** nad $\mathcal{D}$

# Interpretace

- **Interpretace** $\mathcal{I}$ jazyka $\mathcal{L}$ nad abecedou $\mathcal{A}$ je dána
  - neprázdnou množinou $\mathcal{D}$ (také značíme $|\mathcal{I}|$, nazývá se **univerzum**) a
  - zobrazením, které
    - každé konstantě $c \in \mathcal{A}$ přiřadí nějaký **prvek** $\mathcal{D}$
    - každému funkčnímu symbolu $f/n \in \mathcal{A}$ přiřadí $n$-ární **operaci** nad $\mathcal{D}$
    - každému predikátovému symbolu $p/n \in \mathcal{A}$ přiřadí $n$-ární **relaci** nad $\mathcal{D}$
- Příklad: uspořádání na $\mathbb{R}$
  - jazyk: predikátový symbol $mensi/2$
  - interpretace: univerzum $\mathbb{R}$; zobrazení: $mensi(x, y) := x < y$

# Interpretace

- **Interpretace** $\mathcal{I}$ jazyka $\mathcal{L}$ nad abecedou $\mathcal{A}$ je dána
  - neprázdnou množinou $\mathcal{D}$ (také značíme $|\mathcal{I}|$, nazývá se **univerzum**) a
  - zobrazením, které
    - každé konstantě $c \in \mathcal{A}$ přiřadí nějaký **prvek** $\mathcal{D}$
    - každému funkčnímu symbolu $f/n \in \mathcal{A}$ přiřadí $n$-ární **operaci** nad $\mathcal{D}$
    - každému predikátovému symbolu $p/n \in \mathcal{A}$ přiřadí $n$-ární **relaci** nad $\mathcal{D}$
- Příklad: uspořádání na $\mathbb{R}$
  - jazyk: predikátový symbol $mensi/2$
  - interpretace: univerzum $\mathbb{R}$; zobrazení: $mensi(x,y) := x < y$
- Příklad: elementární aritmetika nad množinou $\mathbb{N}$ (včetně 0)
  - jazyk: konstanta $zero$, funční symboly $s/1$, $plus/2$
  - interpretace:
    - univerzum $\mathbb{N}$; zobrazení: $zero := 0, \quad s(x) := 1 + x, \quad plus(x,y) := x + y$

# Sémantika formulí

- **Ohodnocení proměnné** $\varphi(X)$: každé proměnné $X$ je přiřazen prvek $|\mathcal{I}|$
- **Hodnota termu** $\varphi(t)$: každému termu je přiřazen prvek univerza
  - příklad: nechť $\varphi(X) := 0$

    $\varphi(plus(s(zero), X)) =$

# Sémantika formulí

- **Ohodnocení proměnné** $\varphi(X)$: každé proměnné $X$ je přiřazen prvek $|\mathcal{I}|$
- **Hodnota termu** $\varphi(t)$: každému termu je přiřazen prvek univerza
  - příklad: nechť $\varphi(X) := 0$

    $\varphi(plus(s(zero), X)) = \varphi(s(zero)) + \varphi(X) =$

# Sémantika formulí

- **Ohodnocení proměnné** $\varphi(X)$: každé proměnné $X$ je přiřazen prvek $|\mathcal{I}|$
- **Hodnota termu** $\varphi(t)$: každému termu je přiřazen prvek univerza
  - příklad: nechť $\varphi(X) := 0$

    $\varphi(plus(s(zero), X)) = \varphi(s(zero)) + \varphi(X) = (1 + \varphi(zero)) + 0 =$

# Sémantika formulí

- **Ohodnocení proměnné** $\varphi(X)$: každé proměnné $X$ je přiřazen prvek $|\mathcal{I}|$
- **Hodnota termu** $\varphi(t)$: každému termu je přiřazen prvek univerza
  - příklad: nechť $\varphi(X) := 0$

    $\varphi(plus(s(zero), X)) = \varphi(s(zero)) + \varphi(X) = (1 + \varphi(zero)) + 0 = (1 + 0) + 0 = 1$

# Sémantika formulí

- **Ohodnocení proměnné** $\varphi(X)$: každé proměnné $X$ je přiřazen prvek $|\mathcal{I}|$
- **Hodnota termu** $\varphi(t)$: každému termu je přiřazen prvek univerza
  - příklad: nechť $\varphi(X) := 0$

    $\varphi(plus(s(zero), X)) = \varphi(s(zero)) + \varphi(X) = (1 + \varphi(zero)) + 0 = (1 + 0) + 0 = 1$
- Každá **dobře utvořená formule** označuje **pravdivostní hodnotu** **(PRAVDA, NEPRAVDA)** v závislosti na své struktuře a interpretaci

  **Pravdivá formule** $\mathcal{I} \models_\varphi Q$: formule $Q$ označena PRAVDA

  **Nepravdivá formule** $\mathcal{I} \not\models_\varphi Q$: formule $Q$ označena NEPRAVDA

# Sémantika formulí

- **Ohodnocení proměnné** $\varphi(X)$: každé proměnné $X$ je přiřazen prvek $|\mathcal{I}|$
- **Hodnota termu** $\varphi(t)$: každému termu je přiřazen prvek univerza
  - příklad: nechť $\varphi(X) := 0$

    $\varphi(plus(s(zero), X)) = \varphi(s(zero)) + \varphi(X) = (1 + \varphi(zero)) + 0 = (1 + 0) + 0 = 1$
- Každá **dobře utvořená formule** označuje **pravdivostní hodnotu (PRAVDA, NEPRAVDA)** v závislosti na své struktuře a interpretaci

  **Pravdivá formule** $\mathcal{I} \models_\varphi Q$: formule $Q$ označena PRAVDA

  **Nepravdivá formule** $\mathcal{I} \not\models_\varphi Q$: formule $Q$ označena NEPRAVDA
  - příklad: $p/1$ predikátový symbol, tj. $p \subseteq |\mathcal{I}|$ $\quad p := \{\langle 1\rangle, \langle 3\rangle, \langle 5\rangle, \ldots\}$

    $\mathcal{I} \models p(zero) \wedge p(s(zero))$

# Sémantika formulí

- **Ohodnocení proměnné** $\varphi(X)$: každé proměnné $X$ je přiřazen prvek $|\mathcal{I}|$
- **Hodnota termu** $\varphi(t)$: každému termu je přiřazen prvek univerza
  - příklad: nechť $\varphi(X) := 0$

    $\varphi(plus(s(zero), X)) = \varphi(s(zero)) + \varphi(X) = (1 + \varphi(zero)) + 0 = (1 + 0) + 0 = 1$
- Každá **dobře utvořená formule** označuje **pravdivostní hodnotu (PRAVDA, NEPRAVDA)** v závislosti na své struktuře a interpretaci

  **Pravdivá formule** $\mathcal{I} \models_\varphi Q$: formule $Q$ označena PRAVDA

  **Nepravdivá formule** $\mathcal{I} \not\models_\varphi Q$: formule $Q$ označena NEPRAVDA
  - příklad: $p/1$ predikátový symbol, tj. $p \subseteq |\mathcal{I}|$ $\quad p := \{\langle 1\rangle, \langle 3\rangle, \langle 5\rangle, \ldots\}$

    $\mathcal{I} \models p(zero) \wedge p(s(zero))$ iff $\mathcal{I} \models p(zero)$ a $\mathcal{I} \models p(s(zero))$

# Sémantika formulí

- **Ohodnocení proměnné** $\varphi(X)$: každé proměnné $X$ je přiřazen prvek $|\mathcal{I}|$
- **Hodnota termu** $\varphi(t)$: každému termu je přiřazen prvek univerza
  - příklad: nechť $\varphi(X) := 0$

    $\varphi(plus(s(zero), X)) = \varphi(s(zero)) + \varphi(X) = (1 + \varphi(zero)) + 0 = (1 + 0) + 0 = 1$
- Každá dobře utvořená formule označuje **pravdivostní hodnotu** **(PRAVDA, NEPRAVDA)** v závislosti na své struktuře a interpretaci

  **Pravdivá formule** $\mathcal{I} \models_{\varphi} Q$: formule $Q$ označena PRAVDA

  **Nepravdivá formule** $\mathcal{I} \not\models_{\varphi} Q$: formule $Q$ označena NEPRAVDA
  - příklad: $p/1$ predikátový symbol, tj. $p \subseteq |\mathcal{I}|$ $\quad p := \{\langle 1 \rangle, \langle 3 \rangle, \langle 5 \rangle, \ldots\}$

    $\mathcal{I} \models p(zero) \wedge p(s(zero))$ iff $\mathcal{I} \models p(zero)$ a $\mathcal{I} \models p(s(zero))$

    iff $\langle \varphi(zero) \rangle \in p$ a $\langle \varphi(s(zero)) \rangle \in p$

# Sémantika formulí

- **Ohodnocení proměnné** $\varphi(X)$: každé proměnné $X$ je přiřazen prvek $|\mathcal{I}|$
- **Hodnota termu** $\varphi(t)$: každému termu je přiřazen prvek univerza
  - příklad: nechť $\varphi(X) := 0$

    $\varphi(plus(s(zero), X)) = \varphi(s(zero)) + \varphi(X) = (1 + \varphi(zero)) + 0 = (1 + 0) + 0 = 1$
- Každá **dobře utvořená formule** označuje **pravdivostní hodnotu (PRAVDA, NEPRAVDA)** v závislosti na své struktuře a interpretaci

  **Pravdivá formule** $\mathcal{I} \models_\varphi Q$: formule $Q$ označena PRAVDA

  **Nepravdivá formule** $\mathcal{I} \not\models_\varphi Q$: formule $Q$ označena NEPRAVDA
  - příklad: $p/1$ predikátový symbol, tj. $p \subseteq |\mathcal{I}|$ $\quad p := \{\langle 1 \rangle, \langle 3 \rangle, \langle 5 \rangle, \ldots\}$

    $\mathcal{I} \models p(zero) \wedge p(s(zero))$ iff $\mathcal{I} \models p(zero)$ a $\mathcal{I} \models p(s(zero))$

    iff $\langle \varphi(zero) \rangle \in p$ a $\langle \varphi(s(zero)) \rangle \in p$

    iff $\langle \varphi(zero) \rangle \in p$ a $\langle (1 + \varphi(zero) \rangle \in p$

# Sémantika formulí

- **Ohodnocení proměnné** $\varphi(X)$: každé proměnné $X$ je přiřazen prvek $|\mathcal{I}|$
- **Hodnota termu** $\varphi(t)$: každému termu je přiřazen prvek univerza
  - příklad: nechť $\varphi(X) := 0$

    $\varphi(plus(s(zero), X)) = \varphi(s(zero)) + \varphi(X) = (1 + \varphi(zero)) + 0 = (1 + 0) + 0 = 1$
- Každá dobře utvořená formule označuje **pravdivostní hodnotu** (PRAVDA, NEPRAVDA) v závislosti na své struktuře a interpretaci

  **Pravdivá formule** $\mathcal{I} \models_\varphi Q$: formule $Q$ označena PRAVDA

  **Nepravdivá formule** $\mathcal{I} \not\models_\varphi Q$: formule $Q$ označena NEPRAVDA
  - příklad: $p/1$ predikátový symbol, tj. $p \subseteq |\mathcal{I}|$ $\quad p := \{\langle 1\rangle, \langle 3\rangle, \langle 5\rangle, \dots\}$

    $\mathcal{I} \models p(zero) \wedge p(s(zero))$ iff $\mathcal{I} \models p(zero)$ a $\mathcal{I} \models p(s(zero))$
    iff $\langle \varphi(zero)\rangle \in p$ a $\langle \varphi(s(zero))\rangle \in p$
    iff $\langle \varphi(zero)\rangle \in p$ a $\langle (1 + \varphi(zero)\rangle \in p$
    iff $\langle 0\rangle \in p$ a $\langle 1\rangle \in p$

    $\langle 1\rangle \in p$ ale $\langle 0\rangle \notin p$, tedy formule je nepravdivá v $\mathcal{I}$

# Model

- Interpretace se nazývá **modelem** formule, je-li v ní tato formule pravdivá
  - interpretace množiny $\mathbb{N}$ s obvyklými operacemi je modelem formule ( $1 + s(0) = s(s(0))$ )
  - interpretace, která se liší přiřazením s/1: s(x):=x není modelem této formule

# Model

- Interpretace se nazývá **modelem** formule, je-li v ní tato formule pravdivá
  - interpretace množiny $\mathbb{N}$ s obvyklými operacemi je modelem formule ( 1 + s(0) = s(s(0)) )
  - interpretace, která se liší přiřazením s/1: s(x):=x není modelem této formule
- **Teorie** $\mathcal{T}$ jazyka $\mathcal{L}$ je množina formulí jazyka $\mathcal{L}$, tzv. **axiomů**
  - $\neg$ s(X) = 0 je jeden z axiomů teorie elementární aritmetiky

# Model

- Interpretace se nazývá **modelem** formule, je-li v ní tato formule pravdivá
  - interpretace množiny $\mathbb{N}$ s obvyklými operacemi je modelem formule ( 1 + s(0) = s(s(0)) )
  - interpretace, která se liší přiřazením s/1: s(x):=x není modelem této formule
- **Teorie** $\mathcal{T}$ jazyka $\mathcal{L}$ je množina formulí jazyka $\mathcal{L}$, tzv. **axiomů**
  - $\neg$ s(X) = 0 je jeden z axiomů teorie elementární aritmetiky
- **Model teorie**: libovolná interpretace, která je modelem všech jejích axiomů
  - všechny axiomy teorie musí být v této interpretaci pravdivé

# Model

- Interpretace se nazývá **modelem** formule, je-li v ní tato formule pravdivá
  - interpretace množiny $\mathbb{N}$ s obvyklými operacemi je modelem formule ( 1 + s(0) = s(s(0)) )
  - interpretace, která se liší přiřazením s/1: s(x):=x není modelem této formule
- **Teorie** $\mathcal{T}$ jazyka $\mathcal{L}$ je množina formulí jazyka $\mathcal{L}$, tzv. **axiomů**
  - $\neg$ s(X) = 0 je jeden z axiomů teorie elementární aritmetiky
- **Model teorie**: libovolná interpretace, která je modelem všech jejích axiomů
  - všechny axiomy teorie musí být v této interpretaci pravdivé
- **Pravdivá formule v teorii** $\mathcal{T} \models F$: pravdivá v každém z modelů teorie $\mathcal{T}$
  - říkáme také formule **platí v teorii** nebo je **splněna v teorii**
  - formule 1 + s(0) = s(s(0)) je pravdivá v teorii elementárních čísel

# Model

- Interpretace se nazývá **modelem** formule, je-li v ní tato formule pravdivá
  - interpretace množiny $\mathbb{N}$ s obvyklými operacemi je modelem formule ( 1 + s(0) = s(s(0)) )
  - interpretace, která se liší přiřazením s/1: s(x):=x není modelem této formule
- **Teorie** $\mathcal{T}$ jazyka $\mathcal{L}$ je množina formulí jazyka $\mathcal{L}$, tzv. **axiomů**
  - $\neg$ s(X) = 0 je jeden z axiomů teorie elementární aritmetiky
- **Model teorie**: libovolná interpretace, která je modelem všech jejích axiomů
  - všechny axiomy teorie musí být v této interpretaci pravdivé
- **Pravdivá formule v teorii** $\mathcal{T} \models F$: pravdivá v každém z modelů teorie $\mathcal{T}$
  - říkáme také formule **platí v teorii** nebo je **splněna v teorii**
  - formule 1 + s(0) = s(s(0)) je pravdivá v teorii elementárních čísel
- **Logicky pravdivá formule** $\models F$: libovolná interpretace je jejím modelem
  - nebo-li $F$ je pravdivá v každém modelu libovolné teorie
  - formule G $\vee$ $\neg$ G je logicky pravdivá, formule 1 + s(0) = s(s(0)) není logicky pravdivá

# Zkoumání pravdivosti formulí

- Zjištění pravdivosti provádíme důkazem

  **Důkaz:** libovolná posloupnost $F_1, \ldots, F_n$ formulí jazyka $\mathcal{L}$, v níž každé $F_i$ je buď axiom teorie jazyka $\mathcal{L}$ nebo lze $F_i$ odvodit z předchozích $F_j$ $(j < i)$ použitím určitých **odvozovacích pravidel**

# Zkoumání pravdivosti formulí

- Zjištění pravdivosti provádíme důkazem

  **Důkaz:** libovolná posloupnost $F_1, \ldots, F_n$ formulí jazyka $\mathcal{L}$, v níž každé $F_i$ je buď axiom teorie jazyka $\mathcal{L}$ nebo lze $F_i$ odvodit z předchozích $F_j$ $(j < i)$ použitím určitých **odvozovacích pravidel**

- Odvozovací pravidla – příklady
  - **pravidlo modus ponens**: z formulí $F$ a $F \Rightarrow G$ lze odvodit $G$

# Zkoumání pravdivosti formulí

- Zjištění pravdivosti provádíme důkazem

  **Důkaz:** libovolná posloupnost $F_1, \ldots, F_n$ formulí jazyka $\mathcal{L}$, v níž každé $F_i$ je buď axiom teorie jazyka $\mathcal{L}$ nebo lze $F_i$ odvodit z předchozích $F_j$ $(j < i)$ použitím určitých **odvozovacích pravidel**

- Odvozovací pravidla – příklady
  - **pravidlo modus ponens**: z formulí $F$ a $F \Rightarrow G$ lze odvodit $G$
  - **rezoluční princip**: z formulí $F \vee A$, $G \vee \neg A$ odvodit $F \vee G$

# Zkoumání pravdivosti formulí

- Zjištění pravdivosti provádíme důkazem

  **Důkaz:** libovolná posloupnost $F_1, \ldots, F_n$ formulí jazyka $\mathcal{L}$, v níž každé $F_i$ je buď axiom teorie jazyka $\mathcal{L}$ nebo lze $F_i$ odvodit z předchozích $F_j$ $(j < i)$ použitím určitých **odvozovacích pravidel**

- Odvozovací pravidla – příklady
  - **pravidlo modus ponens**: z formulí $F$ a $F \Rightarrow G$ lze odvodit $G$
  - **rezoluční princip**: z formulí $F \vee A$, $G \vee \neg A$ odvodit $F \vee G$

- $F$ je **dokazatelná z formulí** $A_1, \cdots, A_n$ $\qquad A_1, \cdots, A_n \vdash F$

  existuje-li důkaz $F$ z $A_1, \cdots, A_n$

# Zkoumání pravdivosti formulí

- Zjištění pravdivosti provádíme důkazem

  **Důkaz:** libovolná posloupnost $F_1, \ldots, F_n$ formulí jazyka $\mathcal{L}$, v níž každé $F_i$ je buď axiom teorie jazyka $\mathcal{L}$ nebo lze $F_i$ odvodit z předchozích $F_j$ $(j < i)$ použitím určitých **odvozovacích pravidel**

- Odvozovací pravidla – příklady
  - **pravidlo modus ponens**: z formulí $F$ a $F \Rightarrow G$ lze odvodit $G$
  - **rezoluční princip**: z formulí $F \vee A$, $G \vee \neg A$ odvodit $F \vee G$

- $F$ je **dokazatelná z formulí** $A_1, \cdots, A_n$ $\qquad A_1, \cdots, A_n \vdash F$

  existuje-li důkaz $F$ z $A_1, \cdots, A_n$

- Dokazatelné formule v teorii $\mathcal{T}$ nazýváme **teorémy** teorie $\mathcal{T}$

# Korektnost a úplnost

- **Uzavřená formule**: neobsahuje volnou proměnnou (bez kvantifikace)
  - $\forall Y\ (\ (0 < Y) \wedge (\ \exists X\ (X < Y)\ )\ )$ je uzavřená formule
  - $(\ \exists X\ (X < Y)\ )$ není uzavřená formule

# Korektnost a úplnost

- **Uzavřená formule**: neobsahuje volnou proměnnou (bez kvantifikace)
  - $\forall Y\ ( (0 < Y) \wedge ( \exists X\ (X < Y) ) )$ je uzavřená formule
  - $( \exists X\ (X < Y) )$ není uzavřená formule
- Množina odvozovacích pravidel se nazývá **korektní**, jestliže pro každou množinu uzavřených formulí $\mathcal{P}$ a každou uzavřenou formuli $F$ platí:

  jestliže $\mathcal{P} \vdash F$ pak $\mathcal{P} \models F$ (jestliže je něco dokazatelné, pak to platí)

# Korektnost a úplnost

- **Uzavřená formule**: neobsahuje volnou proměnnou (bez kvantifikace)
  - $\forall$Y ( (0 $<$ Y) $\wedge$ ( $\exists$X (X $<$ Y) ) ) je uzavřená formule
  - ( $\exists$X (X $<$ Y) ) není uzavřená formule
- Množina odvozovacích pravidel se nazývá **korektní**, jestliže pro každou množinu uzavřených formulí $\mathcal{P}$ a každou uzavřenou formuli $F$ platí:

  jestliže $\mathcal{P} \vdash F$ pak $\mathcal{P} \models F$ (jestliže je něco dokazatelné, pak to platí)

  Odvozovací pravidla jsou **úplná**, jestliže

  jestliže $\mathcal{P} \models F$ pak $\mathcal{P} \vdash F$ (jestliže něco platí, pak je to dokazatelné)

# Korektnost a úplnost

- **Uzavřená formule**: neobsahuje volnou proměnnou (bez kvantifikace)
  - $\forall$Y ( (0 < Y) $\wedge$ ( $\exists$X (X < Y) ) ) je uzavřená formule
  - ( $\exists$X (X < Y) ) není uzavřená formule
- Množina odvozovacích pravidel se nazývá **korektní**, jestliže pro každou množinu uzavřených formulí $\mathcal{P}$ a každou uzavřenou formuli $F$ platí:

  jestliže $\mathcal{P} \vdash F$ pak $\mathcal{P} \models F$ (jestliže je něco dokazatelné, pak to platí)

  Odvozovací pravidla jsou **úplná**, jestliže

  jestliže $\mathcal{P} \models F$ pak $\mathcal{P} \vdash F$ (jestliže něco platí, pak je to dokazatelné)
- PL1: úplná a korektní dokazatelnost, tj.
  pro teorii $\mathcal{T}$ s množinou axiomů $\mathcal{A}$ platí: $\mathcal{T} \models F$ **právě když** $\mathcal{A} \vdash F$