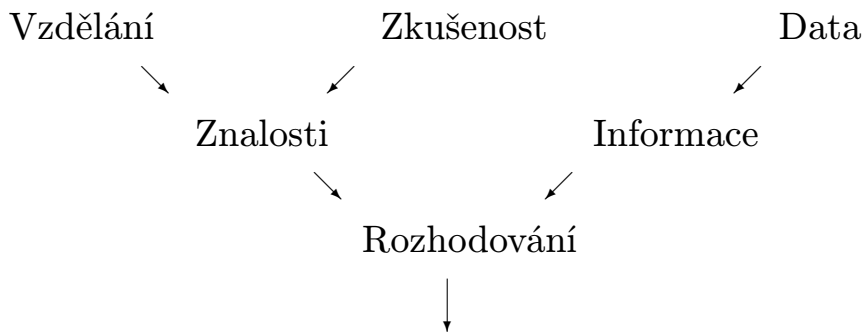
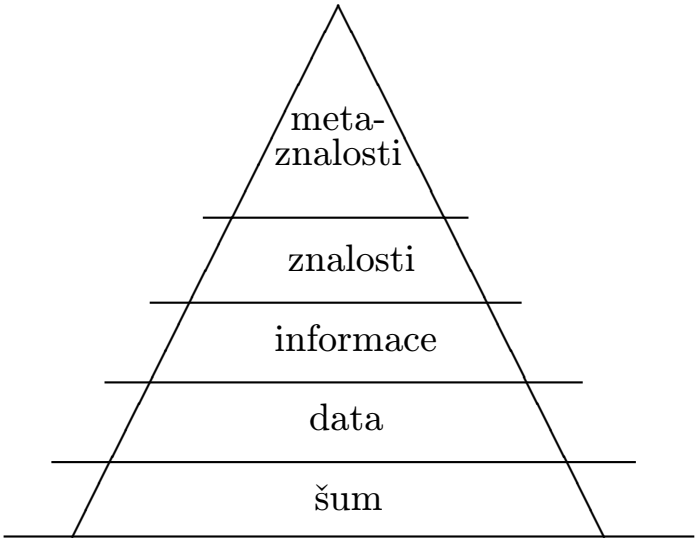
Data a znalosti



- Data
  - lze je získat automaticky nebo od úředníka;
  - správnost dat vzhledem k reálnému světu může být objektivně verifikována
- Znalosti
  - získáváme je od experta;
  - obsahují abstrakce a generalizace objemného materiálu;
  - typicky méně přesné, nemohou být objektivně verifikovány

Petr Berka, 2004

Hierarchie znalostí



Petr Berka, 2004

Základní prostředky reprezentace znalostí

- predikátová logika
- sémantické sítě
- rámce
- produkční pravidla
- případy

Predikátová logika

Nejlépe prozkoumaný prostředek pro reprezentaci znalostí.

- konstanty a proměnné
- predikátové symboly (relace)
- funkční symboly (operace)
- logické spojky $\neg, \wedge, \vee, \Rightarrow, \Leftrightarrow$
- kvantifikátory $\forall, \exists$

- termy (z proměnných, konstant a funkčních symbolů):
  - každá konstanta a proměnná je term
  - jsou-li t_1,..., t_n termy a f funkční symbol, je $f(t_1, ..., t_n)$ term
- formule (z predikátů, termů, logických spojek a kvantifikátorů):
  - atomická formule - $P(t_1, ..., t_n)$, kde P je predikátový symbol
  - $\neg\phi,\ \phi \wedge \psi,\ \phi \vee \psi,\ \phi \Rightarrow \psi,\ \phi \Leftrightarrow \psi$
  - $\forall x\, \phi(x),\ \exists x\, \phi(x)$

přednosti:

- přirozenost
- modularita
- uniformita
- je možná procedurální i deklarativní sémantika

$$A_1 \,\&\, A_2 \,\&\, ... \,\&\, A_n \ \Rightarrow\ B$$

(chceš-li dokázat B, dokaž A_1 ... A_n) (platí-li A_1 ... A_n, platí i B)

Klauzule

- Obecná formule

$$(\forall \varepsilon > 0)(\exists \delta > 0)(\forall x)\ |x - a| < \delta \Rightarrow |f(x) - f(a)| < \varepsilon$$

- Klauzule

formule tvořené disjunkcemi literálů, kde literál je atomická formule nebo její negace.

$$P(x, y) \vee Q(z) \vee \neg R(w)$$

- Hornovy klauzule

Klauzule, které obsahují nejvýše jeden pozitivní literál

$$P(x, y) \vee \neg Q(z) \vee \neg R(w)$$

- tuto klauzuli lze zapsat ve tvaru

$$Q(z) \wedge R(w) \Rightarrow P(x, y).$$

Prolog

Prolog (PROgramming in LOGic) - 1972 Marseille (A. Colmerauer a P. Roussel). 1977 standardní implementace v Edinburgu.

Deklarativní způsob programování - specifikuje se *co* se má spočítat, aniž by se podrobně specifikovalo *jak* se to má spočítat.

Termy:

- jednoduché, tzn. konstanty a proměnné,
- složené, tvořené funktorem a n-ticí termů (např: `den(sobota)`, nebo `otec(karel_IV,zikmund)`).

Klauzule:

- atomické formule, které odpovídají složeným termům
  `P1`
- podmíněné příkazy ve tvaru implikace
  `A :- P1,P2,...,Pn`
- cílové klauzule, které mají tvar dotazu
  `?-C1,C2,...,Cn`.

Seznamy: `[H|T]`
Pro seznam `[a, e, i, o, u]` `H = a` a `T = [e, i, o, u]`.

```
member(X,[X|_].
member(X,[_|T]) :- member(X,T).
```

Petr Berka, 2004

Strategie Prologu

prohledávání do hloubky a mechanismus navracení *backtracking*

`?-C1,...,Cn`

Výpočet začíná pokusem splnit cíl `C1`. V případě, že se podařilo tento cíl splnit (např. nalezením **prvního** vyhovujícího faktu v databázi), systém pokračuje cílem `C2`. Ten se opět pokouší splnit (hledáním klauzulí jejichž hlava odpovídá cíli `C2` nebo hledáním faktů v databázi). Pokud se cíl `C2` nepodařilo splnit, program se *navrací* k cíli `C1` a pokouší se ho splnit jiným způsobem (např. nalezením **jiného** vyhovujícího faktu v databázi) tak, aby se eventuelně podařilo splnit i cíl `C2`. Podařilo-li se splnit cíl `C2`, pokračuje se cílem `C3`, atd.

```
hanoj(N) :-
    presun_vez(N,a,b,c).

presun_vez(0,_,_,_).
presun_vez(N,X,Y,Z) :-
    M = N-1,
    presun_vez(M,X,Z,Y),
    tah(X,Z),
    presun_vez(M,Y,X,Z).

tah(X,Z) :-
    write("Presun disk z "),write(X),
    write(" na "),write(Z),nl.
```

Petr Berka, 2004

Prolog jako expertní systém

Pravidla jako Prologovské klauzule:

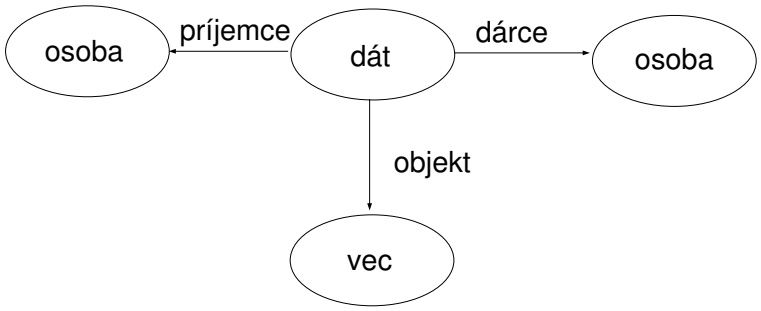
```
savec(X) :- dava_mleko(X).
savec(X) :- ma_srst(X).
dravec(X) :- savec(X),zere_maso(X).
kocka(X) :- dravec(X),mnouka(X).
pes(X) :- dravec(X),steka(X).
pes(X) :- dravec(X),chodi_na_voditku(X).
```

Odpovědi na otázky jako Prologovská fakta:

```
ma_srst(mici).
ma_srst(bobek).
ma_srst(sultan).
ma_srst(tyrl).
zere_maso(mici).
zere_maso(sultan).
zere_maso(tyrl).
mnouka(mici).
dava_mleko(stracena).
steka(sultan).
chodi_na_voditku(tyrl).
```

Sémantické sítě

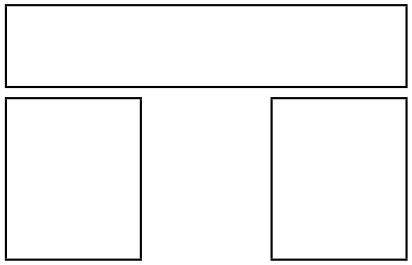
Původně nástroj pro reprezentaci významu výrazů (vět) v přirozeném jazyce (Quillian, 1968):

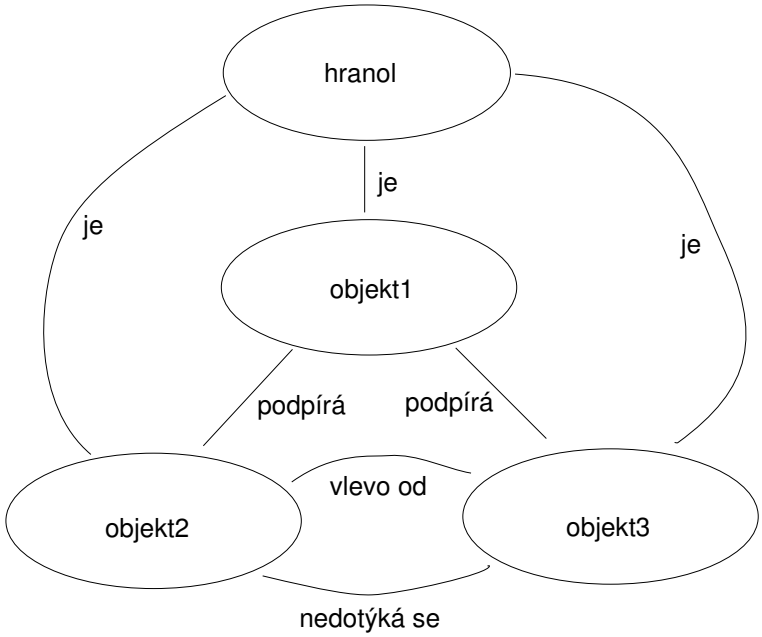


Později obecnější grafová reprezentace znalostí:

- **uzly** - objekty
- **hrany** - (libovolné) relace mezi objekty, k standardním relacím patří
  - A-KIND-OF (ako): pro vyjádření relace *specializace*
  - IS-A (isa): pro vyjádření relace *instance*

Příklad sémantické sítě

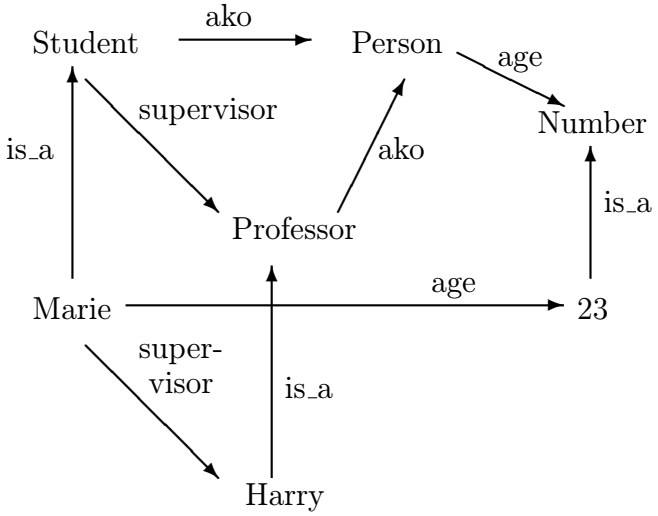




Petr Berka, 2004

Predikátová logika vs. Sémantické sítě

$(\forall x) student(x) \Rightarrow person(x)$
$(\forall x) professor(x) \Rightarrow person(x)$
$(\forall x) person(x) \Rightarrow (\exists y) number(y) \wedge age(x, y)$
$(\forall x) student(x) \Rightarrow (\exists y) professor(y) \wedge supervisor(x, y)$
$student(marie)$
$professor(harry)$
$number(23)$

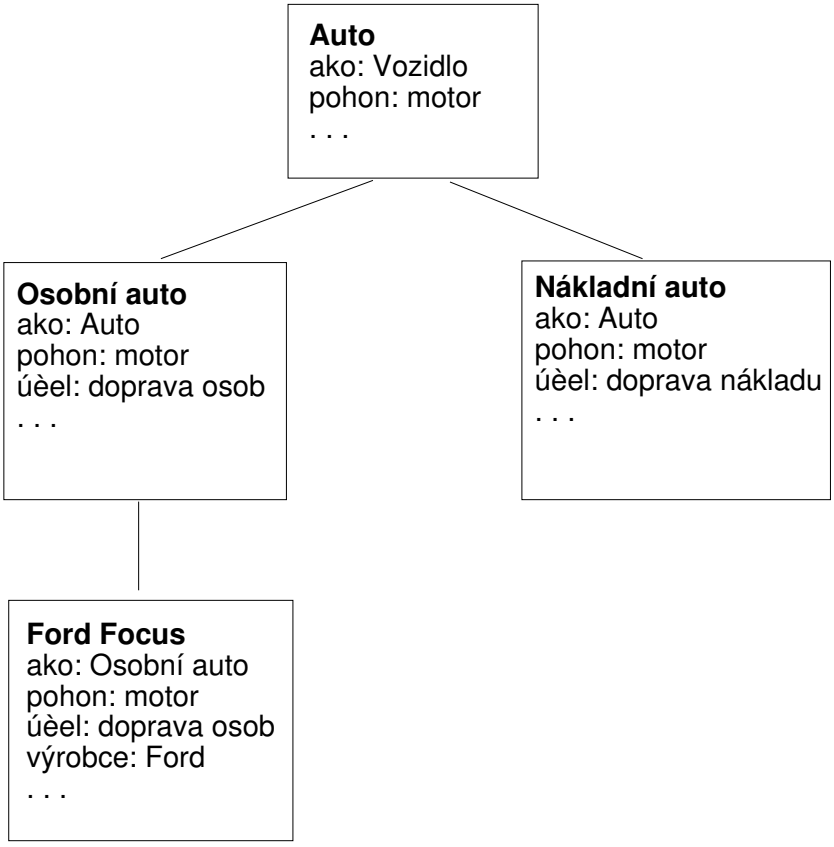


Petr Berka, 2004

Rámce

Datové struktury umožňující reprezentovat stereotypní situace:

- postupné vyplňování stránek
- předdefinované hodnoty (defaults)
- položky (standartně ako, isa, part_of) a meta-položky
- hierarchie (generalizace/specializace), dědičnost, zapouzdřenost



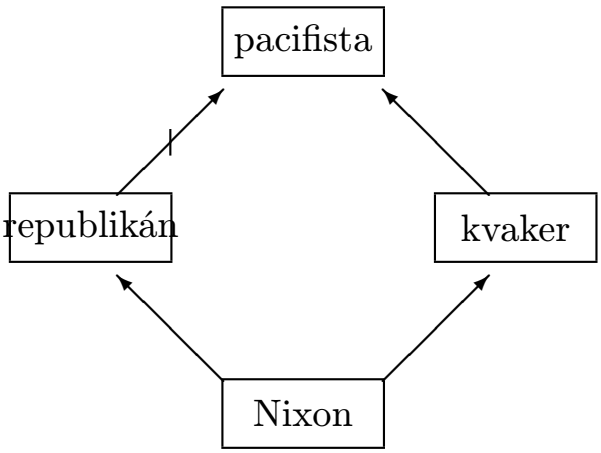
Příklady rámců

Nexpert Object, *PC-Kappa* pro osobní počítače nebo *CLOS* (Common Lisp Object System), pro pracovní stanice:

```
(defclass auto (vozidlo)
    (pohon :initform 'motor))
(defclass osobní_auto (auto)
    (účel :initform 'přeprava_osob))
```

syntaxe Common Lisp Object System (CLOS)

Rámce v systému *Nexpert Object* umožňují používat (násobné) dědění položek i jejich hodnot. Možnost násobného dědění hodnot položek může vést k inkonsistencím:



Příklad Meta-položek

Systém Nexpert Object umožňuje v meta-položkách definovat

- způsob, jakým se systém ptá uživatele na hodnotu
- inferenční prioritu
- prioritu dědění
- typ dědičnosti (lze dědit vlastnosti, hodnoty i metapoložky,
- pořadí zdrojů (Order of Sources):
  - zpětné řetězení,
  - dědění dolů,
  - dědění vzhůru.
  - dotaz uživateli,
  - inicializace na počátku konzultace,
  - zjišťování při běhu (default),
  - načtení z databáze,
  - zavolání externí procedury.
- akce po změně hodnoty (If Change)

Pravidla

Pravidlo v systému R1/XCON

```
IF
    The current context is assigning devices to Unibus models and
    There is an unassigned dual-port disk drive and
    The type of controller it requires is known and
    There are two such controllers, neither of which has any devices
       assigned to it, and
    The number of devices that these controllers can support is known
THEN
    Assign the disk drive to each of the controllers,
      and
    Note that the two controllers have been associated and that each
      supports one drive
```

Pravidlo v systému MYCIN

```
IF
    The site of the culture is blood, and
    The identity of the organism is not known with certainty, and
    The stain of the organism is gramneg, and
    The morfology of the organism is rod, and
    The patient has been seriously burned
THEN
    There is a weakly suggestive evidence (.4) that the identity
       of the organism is pseudomonas
```

Pravidla

Pravidla jakožto IF-THEN struktury jsou dobře známa z programovacích jazyků, použití pravidel vychází z implikací ve výrokové logice.

procedurální sémantika:
jestliže *situace* pak *akce*

deklarativní sémantika:
jestliže *předpoklad* pak *závěr*
(případně jestliže *předpoklad* pak *závěr* a *akce*)

Vyjadřovací síla:

- `IF a THEN b ELSE c` lze zapsat jako dvě pravidla

```
IF     a  THEN b
IF NOT a  THEN c
```

- `IF a OR b ELSE c` lze zapsat jako dvě pravidla

```
IF a  THEN c
IF b  THEN c
```

Atributy a výroky

Situace, předpoklad a závěr pravidel jsou kombinace tvrzení o stavu světa:

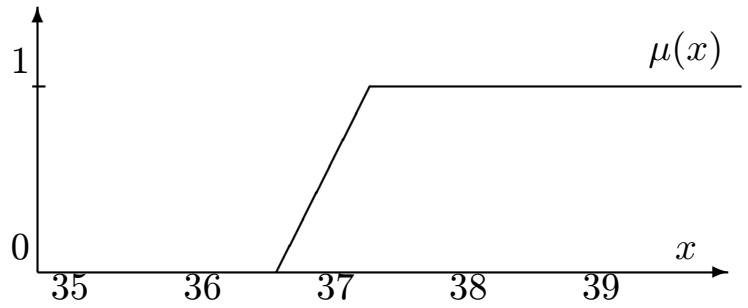
- výrok `auto má červenou barvu`
- atribut, hodnota `barva_auta = červená`
- objekt, atribut, hodnota `auto: barva = červená`

typy atributů:

- kategoriální - tvrzení tvořena hodnotami atributů
  - binární (např. žena)
  - nominální (např. barva vlasů)
  - ordinální (např. dosažené vzdělání)
- numerické (např. věk) - tvrzení tvořena intervaly hodnot

Atributy v systému NEST

Atribut	výrok
binární	True, False
nominální	hodnoty atributu
numerický	fuzzy intervaly



Fuzzy interval zvýšená teplota

- nominální atributy jednoduché vs. množinové
- atributy případu vs. atributy prostředí

Pravidla v systému NEST

Pravidla s prioritami

IF předpoklad THEN závěr A akce

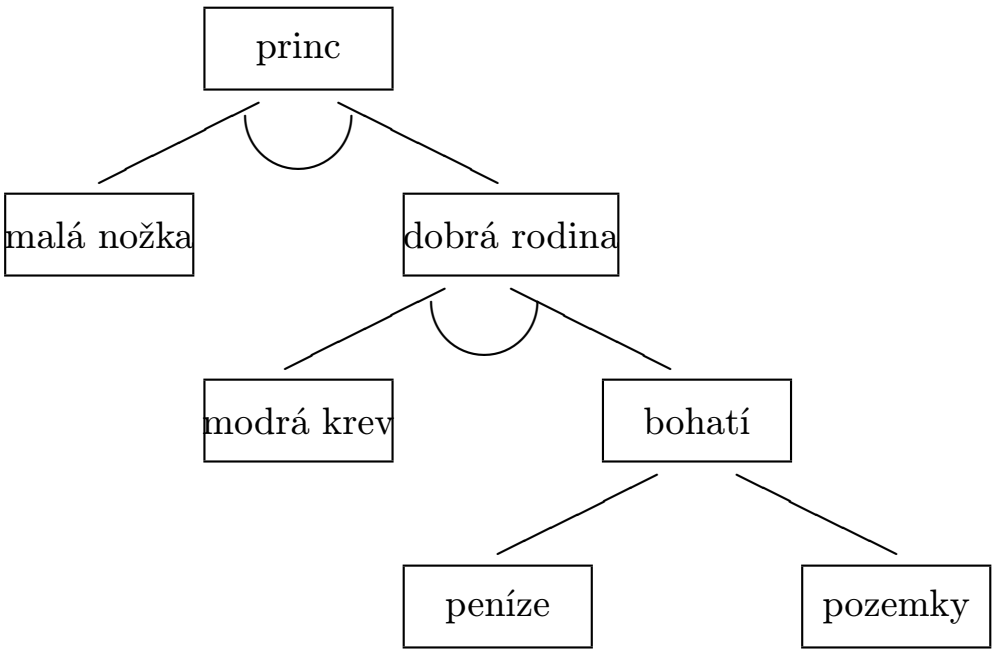
kde předpoklad je disjunktivní forma (disjunkce konjunkcí) literálů, závěr je seznam literálů a akce je seznam akcí (externích programů)

Typy pravidel:

- *kompozicionální* - každý literál v závěru doplněn vahou
  `IF noha(malá) AND dobrá_rodina THEN princ[3,000]`
- *apriorní* - kompozicionální pravidla bez předpokladu
  `Apriori THEN princ[-1,000]`
- *logická* - nekompozicionální pravidla bez vah
  `IF NOT(dobrá_rodina) THEN NOT(princ)`

Petr Berka, 2004

Inferenční síť



AND-OR graf

- *dotazy* - tvrzení, která se nevyskytují v závěrech pravidel
- *cíle* - tvrzení, která se nevyskytují v přepokladech pravidel
- *mezilehlé uzly* - ostatní tvrzení

Rámce a pravidla

častá kombinace v komerčních systémech:

- rámce (objekty) pro vyjádření statických znalostí
- pravidla pro odvozování

Příkladem generativního systému může být produkční systém (jazyk) Ops5. Prostředkem pro reprezentaci znalostí jsou v Ops5 objekty (rámce) a pravidla. Objekty se dělí do tříd, každá třída má svoje atributy. Ke vzniku, zániku a modifikaci objektů dochází působením (produkčních) pravidel. Objekty jsou během práce systému uloženy v pracovní paměti.

Příklad pravidla:

```
(P  jak-se-do-lesa-vola
    ((message          ^type  acustic
                       ^source-id  <x>
                       ^dest-id  <y>
                       ^cont  <cont>)  <old>)
    (information-node  ^id  <y>
                       ^kind  forest)
-->
    ((MAKE message     ^type acustic
                       ^source-id  <y>
                       ^dest-id  <x>
                       ^cont  <cont>)  <new>)
    (REMOVE message    <old>) )
```

Nexpert Object

Třídy (rámce)

```
(@CLASS=  adept  (@PROPERTIES=  parents_status
                                size_of_feet
                                status))
```

Instance objektu

```
(@OBJECT= current_adept
          (@CLASSES=  adept)
          (@PROPERTIES=  evaluation
                         parents_status
                         size_of_feet
                         status))
```

Diagnostická pravidla s akcemi

```
(@RULE= r1
        (@LHS=
              (Is  (current_adept.size_of_feet) ("small"))
              (Yes (good_parents))  )
        (@HYPO=  adept_evaluation)
        (@RHS=
              (Let  (current_adept.status)  ("Prince"))
              (Show ("princ_ye.txt")
        ))
```

Petr Berka, 2004

CLIPS

C Language Integrated Production System je programovací prostředí, které podporuje pravidlové, objektově orientované i procedurální programování (NASA/Johnson Space Center, `http://www.jsc.nasa.gov/)`.

Rámce (objekty)

```
(deftemplate adept   (slot jmeno)
                     (slot velikost_nohy)
                     (slot dobra_rodina)
                     (slot status))
```

Instance objektu

```
(deffacts adepts     (adept (jmeno Honza)
                            (velikost_nohy velka)
                            (status nevim)) )
```

Produkční pravidla

```
(defrule princ
      ?a <- (adept (velikost_nohy mala) (dobra_rodina ano)
            (jmeno ?jmeno) (status nevim))
      =>
      (modify ?a (status "je princ"))
      (printout t  ?jmeno " je princ"  crlf))
```

Petr Berka, 2004

Pomocné prostředky reprezentace znalostí

- **Kontexty**

  popisují situace, kdy to, zda se bude vyhodnocovat nějaké tvrzení závisí na tom, že jiné tvrzení již bylo vyhodnoceno.

  v systému NEST konjunkce literálů, která určuje že aplikovatelné nějaké pravidlo nebo interitní omezení

- **Integritní omezení**

  vyjadřují pravdivé vazby mezi tvrzeními. Nepodílí se na odvozování, používají se pro kontrolu logické konistence průběhu konzultace

  v systému NEST mají podobu $Ant \rightarrow SUC(stupeň)$

- **Akce**

  vnější procedury, jejichž aktivace je vázána na průběh odvozování

  v systému NEST mohou být aktivovány po vyhodnocení tvrzení nebo po aplikování pravidla

- **Zdroje**

  definují způsob získání hodnoty atributu nebo váhy tvrzení

  v systému NEST může být zdrojem odpovědi na dotaz uživatel, soubor, vnitřní funkce nebo vnější procedura

Případy (case)

Případy mají podobu vyřešených problémů z dané aplikační oblasti:

```
BEGIN CASE CASE11
   TITLE
     Zásobník inkoustu je poškozen, způsobuje černé skvrny.
   DESCRIPTION
     Malé kulaté černé skvrny se objevují na přední nebo
     zadní straně papíru.
     Občas se objeví velké nesouvislé skvrny.
   QUESTIONS
     Máte problémy s kvalitou tisku?
           ANSWER: ano
           SCORING: (-)
     Jaká je kvalita tisku?
           ANSWER: černé skvrny
           SCORING: (default)
     Pomohlo vyčištění tiskárny?
           ANSWER: ne
           SCORING: (default)
   ACTIONS
     Zkontrolujte zásobník a vyměňte ho, je-li v něm málo
     náplně nebo je-li poškozen
   CREATION 29/7/91  14:19:22
   LAST_UPDATE 29/7/91  14:19:22
   LAST_USED 29/7/91  14:19:22
END CASE
```

Petr Berka, 2004

Přehled možností reprezentace

Petr Berka, 2004