Nepravidlové a hybridní znalostní systémy

29. října 2024

Opakování – pravidlové znalostní systémy

- systémy porovnávání se vzory
- systémy založené na dopředném řetězení pravidel
- systémy založené na dopředném a zpětném řetězení pravidel
- systémy využívající algoritmus Rete, resp. Xrete
- systémy využívající produkční pravidla

Nepravidlové reprezentace znalostí

K nepravidlovým reprezentačním technikám patří:

- **rozhodovací stromy**
- **rámce**
- **sémantické sítě**
- **Petriho sítě**
- **objekty**

Rozhodovací stromy

Definice: **Rozhodovací stromy** (decision trees) jsou hierarchické nelineární systémy umožňující uložení znalostí. Jsou tvořeny uzly a hranami, uzly představují fáze rozhodovacího procesu:

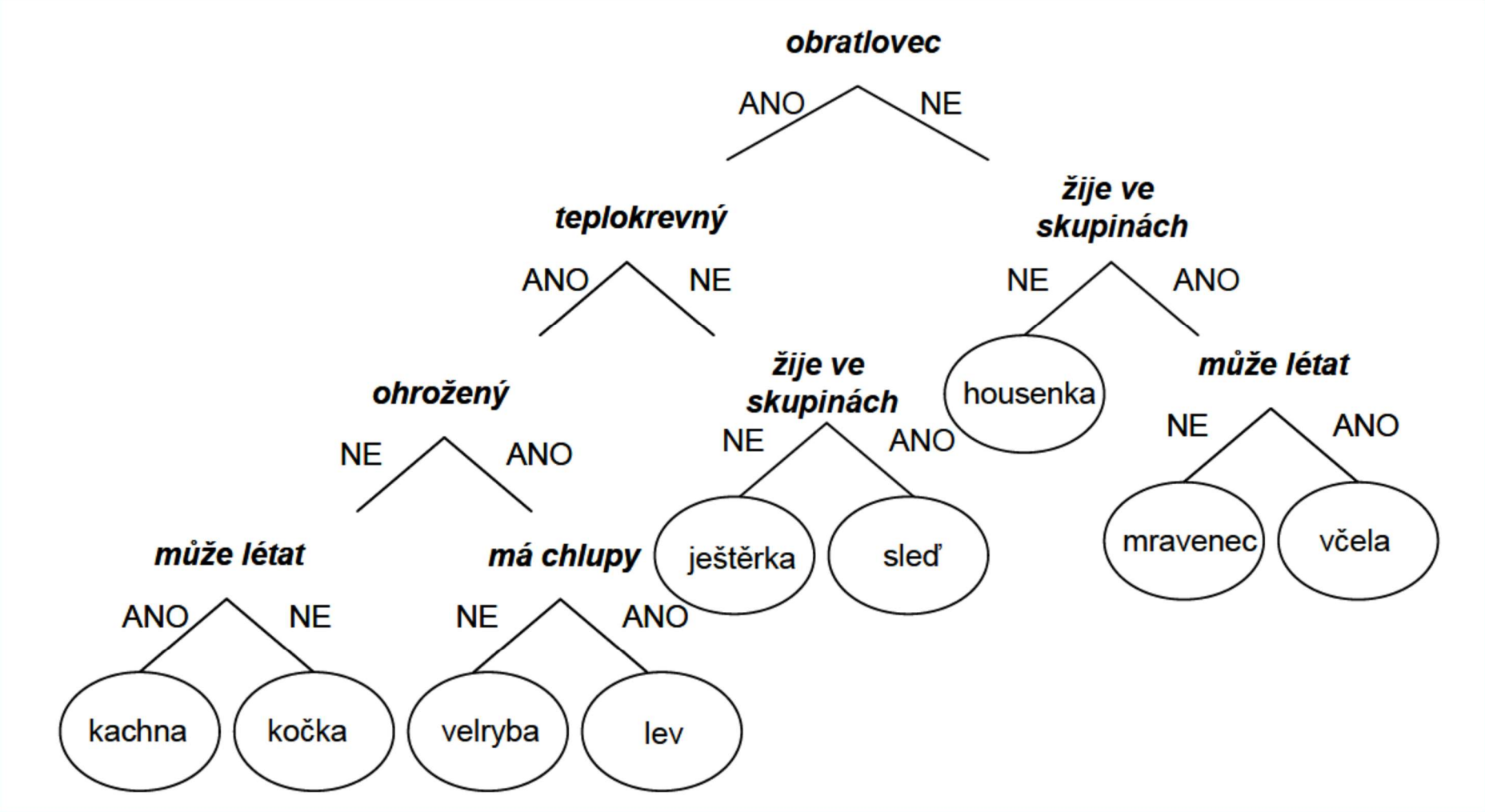
- rozhodovací uzly – deterministické, závislé na uživateli
- situační uzly – náhodné alternativy s pravděpodobnostmi

Hrany pak graficky zobrazují vazby mezi daty a díky grafické interpretaci jsou data uložená do stromu jednoduše čitelná.

Rozhodovací stromy musejí být co nejjednodušší, aby vyhledávání v nich bylo co nejrychlejší a nejefektivnější. Získané znalosti jsou vždy uloženy do struktury rozhodovacího stromu.

Optimální strategie vyhodnocování stromové struktury je dána vyhodnocováním jednotlivých uzlů od konců větví.

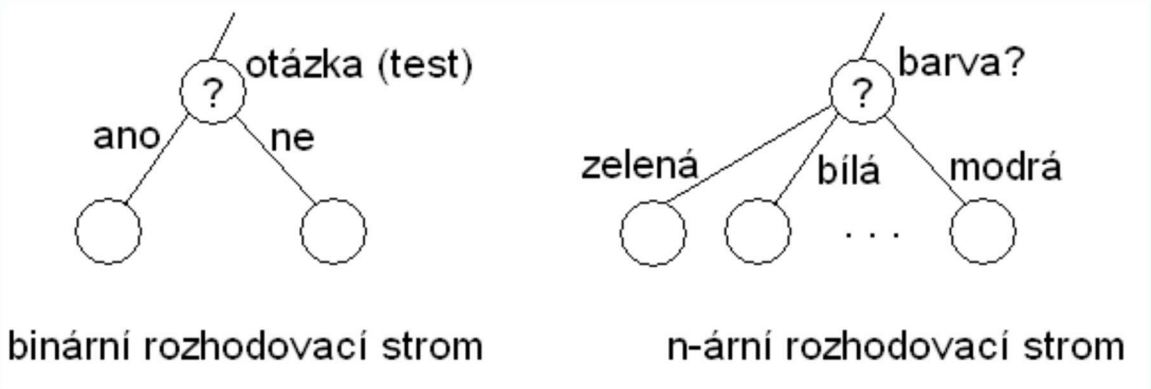
Příklad rozhodovacího stromu:



Rozhodovací strom je tvořen sadou hierarchicky uspořádaných rozhodovacích pravidel. Skládá se z kořene, který představuje celou množinu reprezentovaných objektů, vnitřních uzlů, z nichž vycházejí další hrany, a listových (terminálních) uzlů, které reprezentují zobrazované prvky.

Podle topologie (počtu větví vycházejících z uzlu) rozdělujeme stromy na:

- binární
- vícecestné (n-ární)



Podle hodnoty vstupních a výstupních proměnných se dělí na:

- kvalitativní (symbolické) – hodnoty proměnných jsou dány slovním vyjádřením, a to:
  - ordinální – mezi hodnotami platí vztahy <, =, > (např. malý < střední < velký)
  - nominální – jen výčet všech možností (např. vlasy – černé, hnědé, blond, rezavé)
- kvantitativní (numerické) – hodnoty proměnných jsou vyjadřovány číselnými hodnotami, které mohou být:
  - spojité
  - diskrétní

Podle typu závisle proměnné se pak rozdělují na:

- klasifikační a
- regresní

Podle počtu dělicích podmínek (počtu atributů v rozhodovacím uzlu) rozdělujeme stromy na:

- jednorozměrné (univariantní) – k dělení uzlu je použit pouze jeden atribut
- vícerozměrné (multivariantní) – k dělení uzlu se využívá více atributů (nejčastěji dva); mají však mnohem složitější sestavování a méně čitelnou strukturu rozhodovacího stromu

Podle způsobu vytváření (možností změny stromu) se stromy rozdělují na:

- neinkrementální – s novými vstupními daty musíme celý strom vytvářet znovu
- inkrementální – mají možnost změny struktury stromu podle nově přicházejících dat

Vstupní data pro rozhodovací stromy jsou zpravidla zadávána jako tabulka, ve které každý řádek představuje jeden příklad (např. konkrétní zvíře) a každý sloupec představuje jeden atribut (např. schopnost létat, osrstění atd.):

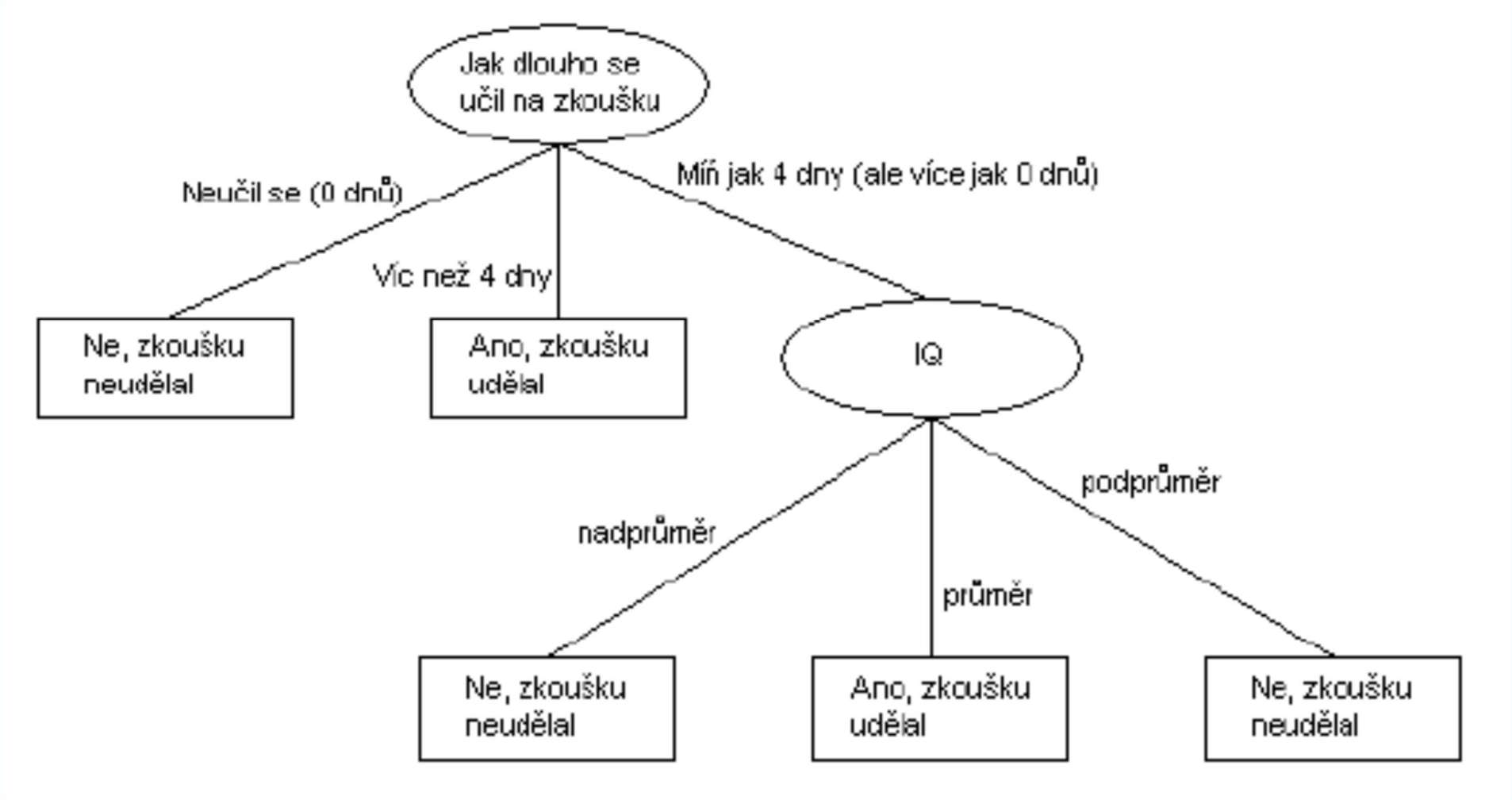
	teplokrevný	může létat	obratlovec	ohrožený	žije ve skupinách	má chlupy
kočka	ANO	NE	ANO	NE	NE	ANO
kachna	ANO	ANO	ANO	NE	ANO	NE
sleď	NE	NE	ANO	NE	ANO	NE
lev	ANO	NE	ANO	ANO	ANO	ANO
ještěrka	NE	NE	ANO	NE	NE	NE
velryba	ANO	NE	ANO	ANO	ANO	NE
mravenec	NE	NE	NE	NE	ANO	NE
včela	NE	ANO	NE	NE	ANO	ANO
housenka	NE	NE	NE	NE	NE	ANO

Příklad vytváření rozhodovacího stromu

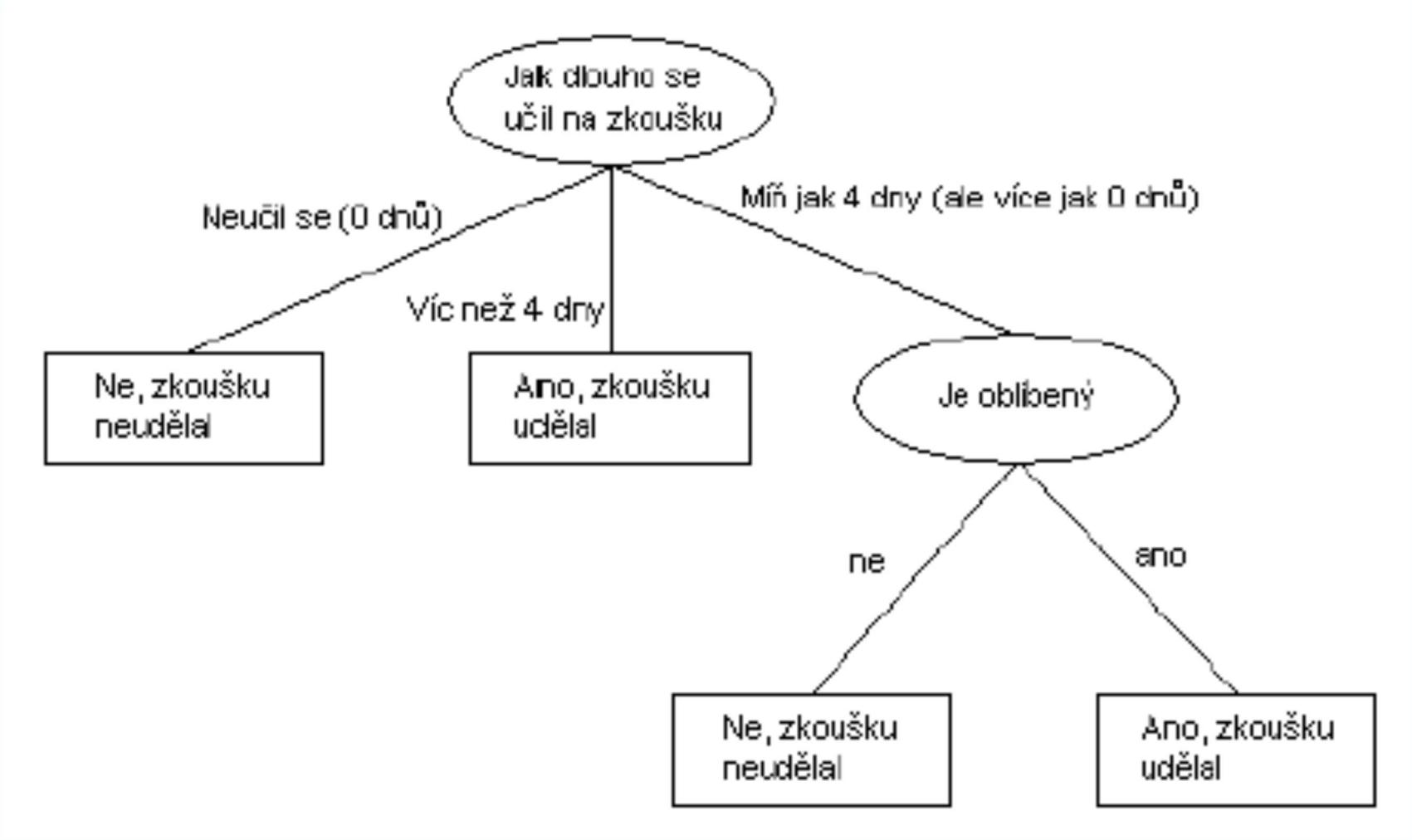
Mějme data o studentech zadaná následující tabulkou:

Student	Je oblíbený	IQ	Jak dlouho se učil na zkoušku	Průměr známek	Zkoušku udělal(á)
Anna K.	Ano	Průměrné	Více než 4 dny	1,3	Ano
Milan H.	Ne	Průměrné	Více než 4 dny	2,0	Ano
Ladislav G.	Ne	Nadprůměrné	Méně jak 4 dny	3,1	Ne
Jan D.	Ne	Podprůměrné	Více než 4 dny	3,2	Ano
Jan T.	Ano	Podprůměrné	Méně jak 4 dny	2,9	Ne
Buhumír F.	Ano	Průměrné	Méně jak 4 dny	2,5	Ano
Libuše R.	Ano	Průměrné	Méně jak 4 dny	3,0	Ano
Rudolf J.	Ano	Průměrné	Neučil se	1,4	Ne

Rozhodovací strom pro data uvedená v tabulce:



Alternativní rozhodovací strom pro data ve výše uvedené tabulce:

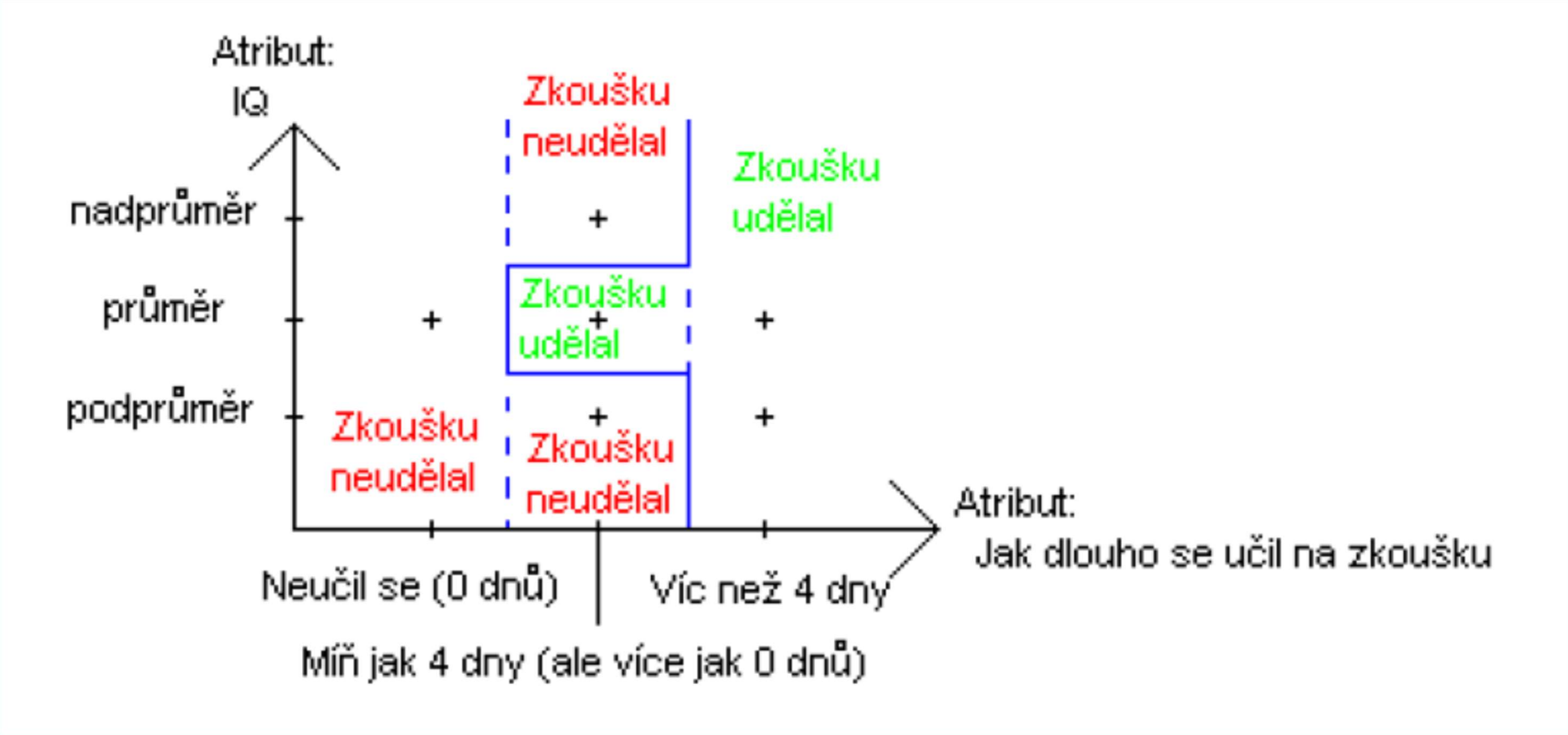


Z uvedených příkladů vyplývá, že pro jednu sadu dat lze vytvořit různé rozhodovací stromy, které rozdělují data do stejných tříd.

Při vytváření rozhodovacího stromu z předem připravených dat se postupuje nejčastěji metodou rozděl a panuj. Data se postupně rozdělují na menší a menší skupiny, ve kterých převládají více a více příklady z jedné třídy, až nakonec zůstanou příklady čistě jenom z jedné třídy (listy). Takový postup se označuje jako TDIDT (top down induction of decision trees).

Rozhodovací stromy postupně rozdělují prostor atributů nadrovinami rovnoběžnými s osami souřadné soustavy, což velmi zjednodušuje jejich interpretaci. Rozdělení prostoru dat podle rozhodovacího stromu na prvním obrázku zobrazíme následujícím obrázkem:

Rozdělení prostoru dat nadrovinami podle prvého rozhodovacího stromu:



Algoritmus vytváření rozhodovacího stromu – algoritmus TDIDT (Top-Down Induction of Decision Trees)

1. Nechť D je množina dat (příkladů) s atributy A, které nabývají hodnot $Hod(A)$, pro která se má vytvořit rozhodovací strom. Zvolí se jeden atribut jako kořen podstromu. $f_A(r) = a \in A$.
2. Data D se rozdělí v tomto uzlu na podmnožiny podle hodnot zvoleného atributu (D_1, D_2, …). Přidá se uzel pro každou z podmnožin: $V = V \cup \{v_1, v_2, \dots\}$, $E = E \cup \{(r, v_1), (r, v_2), \dots\}$, $\forall v \in \{v_1, v_2, \dots\} \rightarrow f_H\big((r, v)\big) = h \in Hod\big(f_A(r)\big)$
3. Existuje-li uzel v_i, pro který nepatří všechna data D_i do téže třídy $t \in T$, tak se spustí na D_i algoritmus TDIDT a spojí se výsledný rozhodovací strom pro D_i s právě vytvářeným tak, že $v_i = r_{D_i}$. Jinak pro každý vrchol v_j, pro který patří všechna data D_j do téže třídy $t \in T$ přidej zobrazení $f_T\big(v_j\big) = t$.

Alternativa vytváření stromu – algoritmus ID3 (Iterative Dichotomiser 3)

Rámce

Rámce (frames) jsou struktury pro reprezentaci stereotypních situací a odpovídajících stereotypních činností (scénářů). Tento prostředek reprezentace vychází z poznatku, že lidé používají pro analyzování a řešení nových situací rámcové struktury znalostí získaných na základě předchozích zkušeností.

Rámce mají reprezentovat obecné znalosti o třídách objektů, znalosti pravdivé pro většinu případů. Mohou existovat objekty, které porušují některé vlastnosti popsané v obecném rámci.

Rámce jsou preferovaným schématem reprezentace v modelovém a případovém usuzování (model-based reasoning, case-based reasoning).

Příklady jazyků pro reprezentaci rámců: KRYPTON, FRL, KSL.

Struktura rámce

Rámec je tvořen jménem a množinou *atributů*.

Atribut (*rubrika, slot*) může dále obsahovat položky (*links*, *facets*), jako např. aktuální hodnotu (*current*), implicitní hodnotu (*default*), rozsah možných hodnot (*range*).

Dalšími položkami atributu (rubriky, slotu) mohou být speciální procedury jako např. *if-needed*, *if-changed*, *if-added*, *if-deleted*. Tyto procedury jsou automaticky aktivovány, jestliže nastanou příslušné situace.

Př.: Typy událostmi řízených procedur v systému FLEX:

- *launches* (aktivují se při vytváření instance rámce)
- *watchdogs* (aktivují se při přístupu k aktuální hodnotě slotu)
- *constraints* (aktivují se před změnou hodnoty slotu)
- *demons* (aktivují se po změně hodnoty slotu)

Vztahy mezi rámci

Mezi rámci mohou existovat vztahy dědičnosti, které umožňují distribuovat informace bez nutnosti jejich zdvojování. Rámec může být specializací jiného obecnějšího rámce (vztah typu *specialization-of*) a současně může být zobecněním jiných rámců (vztah typu *generalization-of*).

Příklady vztahů mezi rámci v systému FLEX:

- *Rodič – potomek* (*is-a, is-an, is-a-kind-of*)

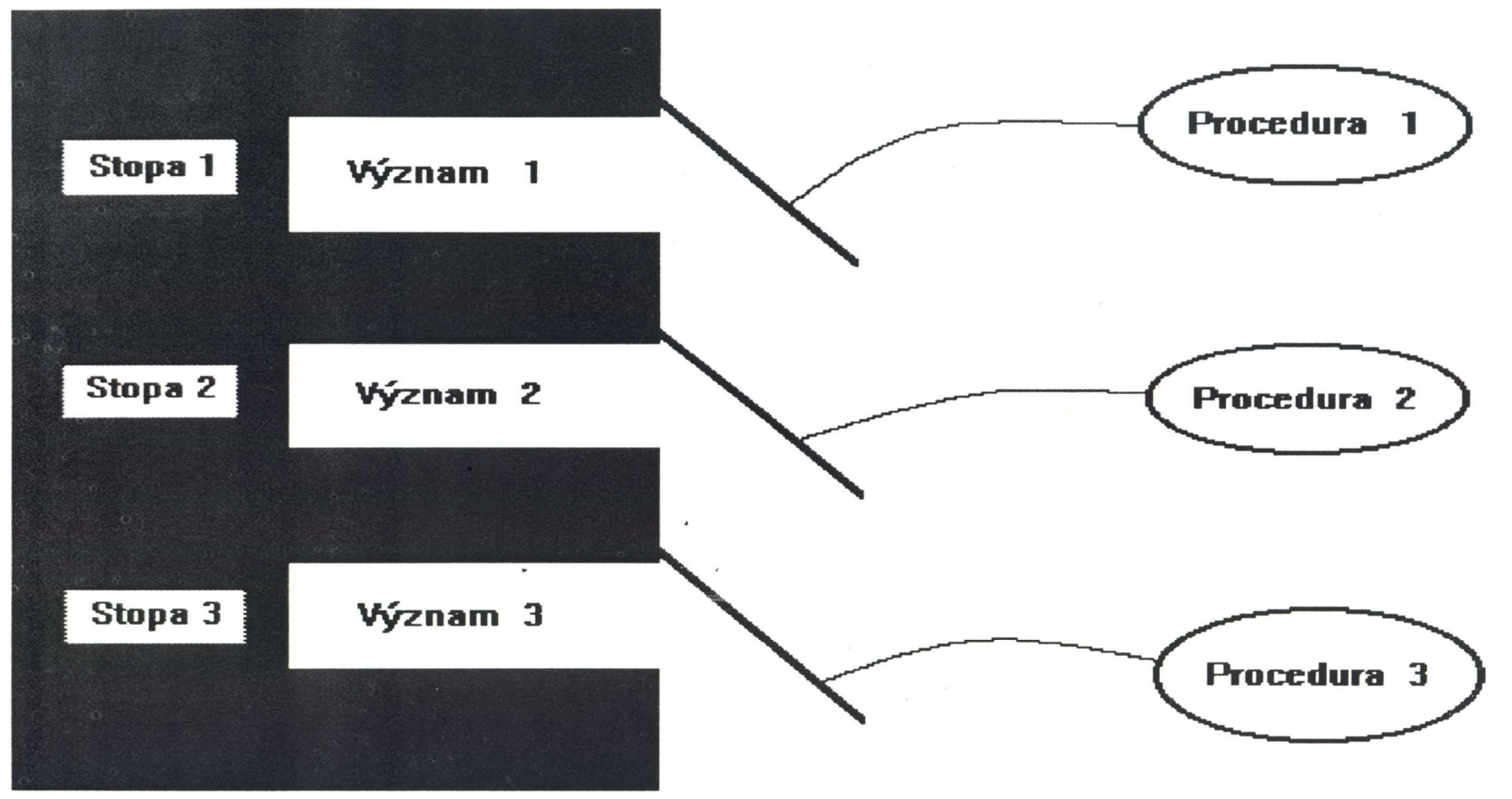
  Tento vztah může být typu 1:1, 1:*n*, *n*:1. Dědění některého atributu může být pro určitý rámec potlačeno.

- *Rámec – instance rámce* (*is-an-instance-of*)

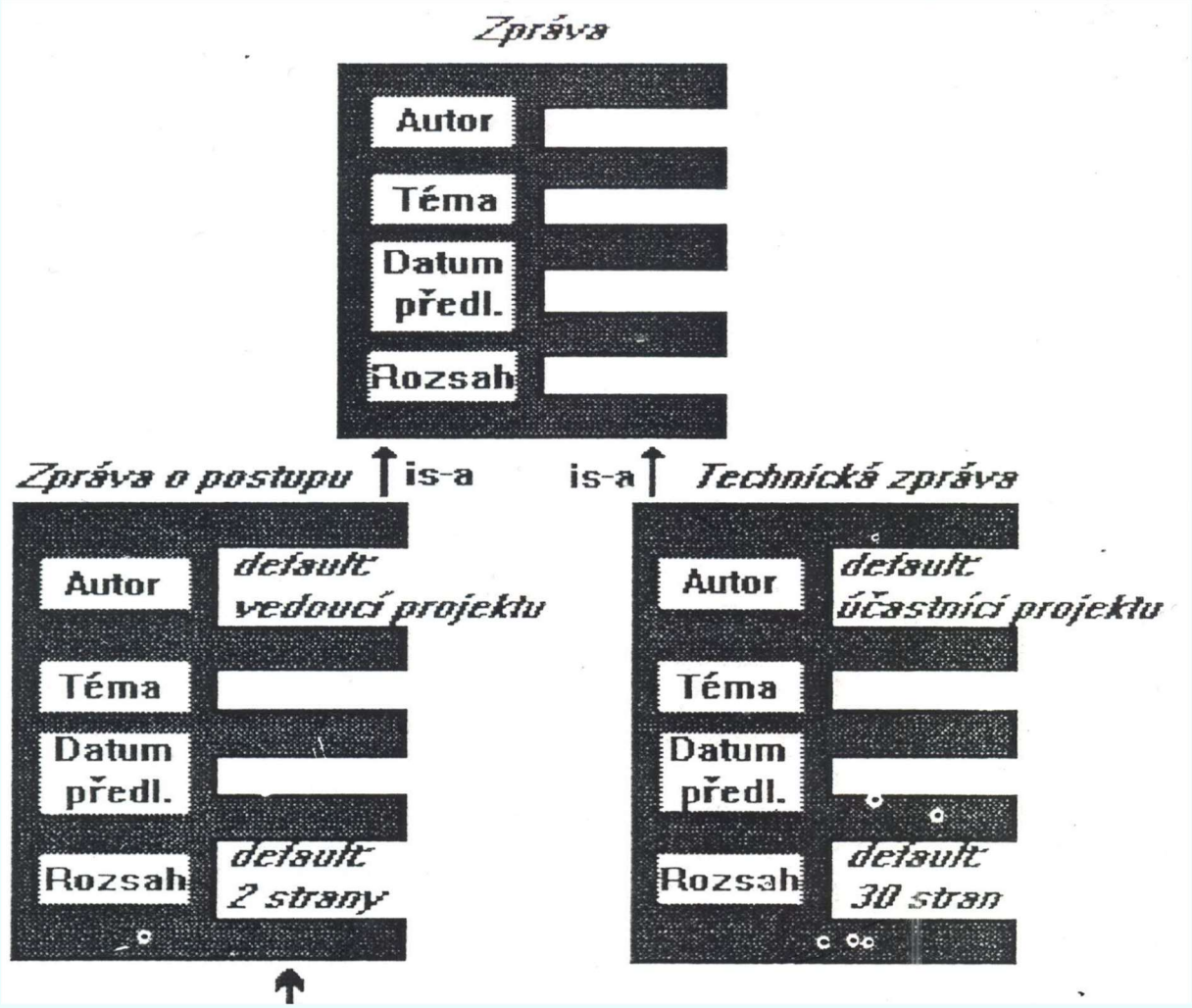
  Tento vztah je typu 1:1. Přitom je navíc možné dědění nějakého specifického atributu od nějakého specifického rámce.

- *Vlastnictví rámce* – atributem rámce může být jiný rámec.

Reprezentace rámcových systémů – jeden prvek

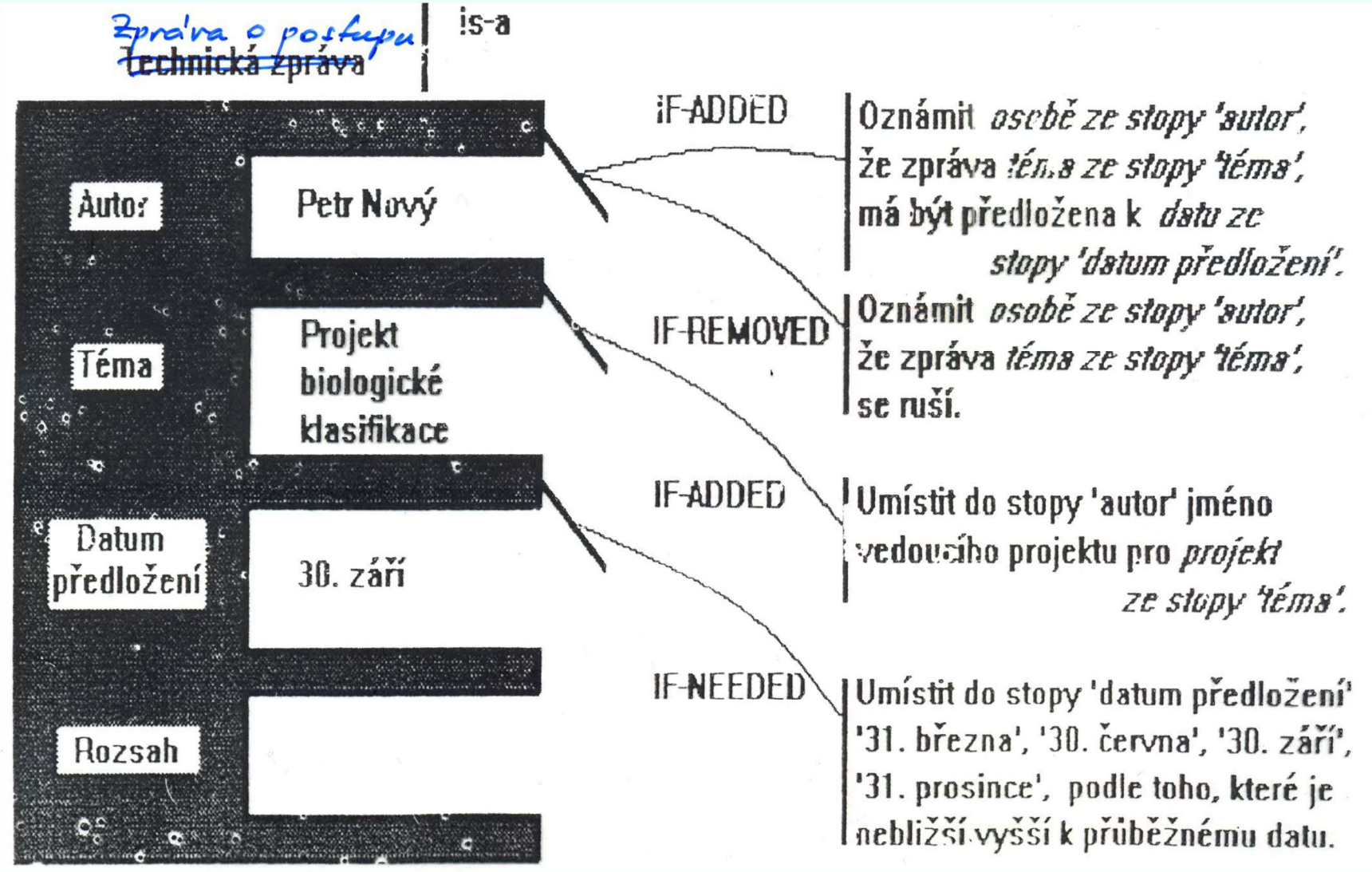


Příklad 1: Hierarchie rámců

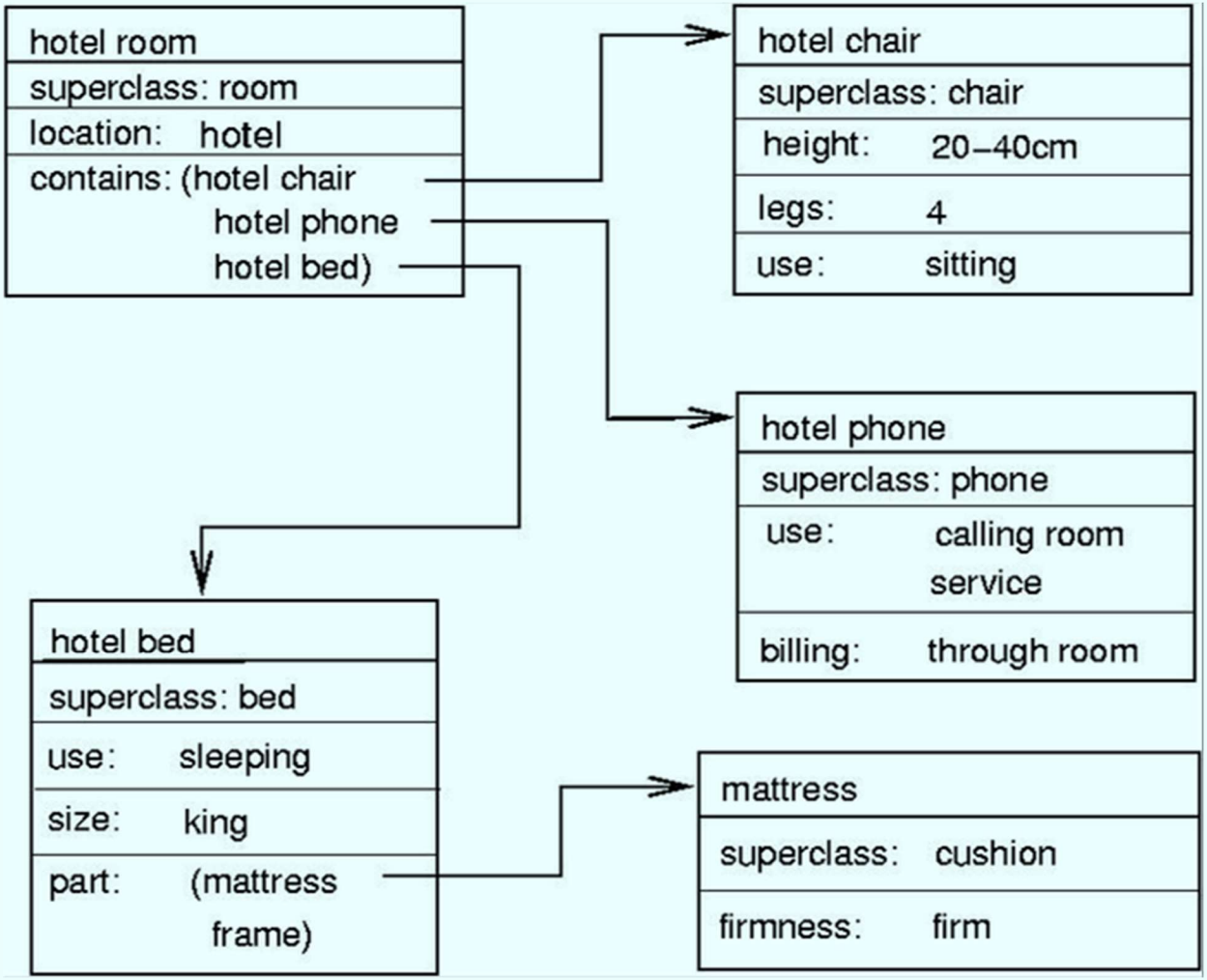


(pokračování)

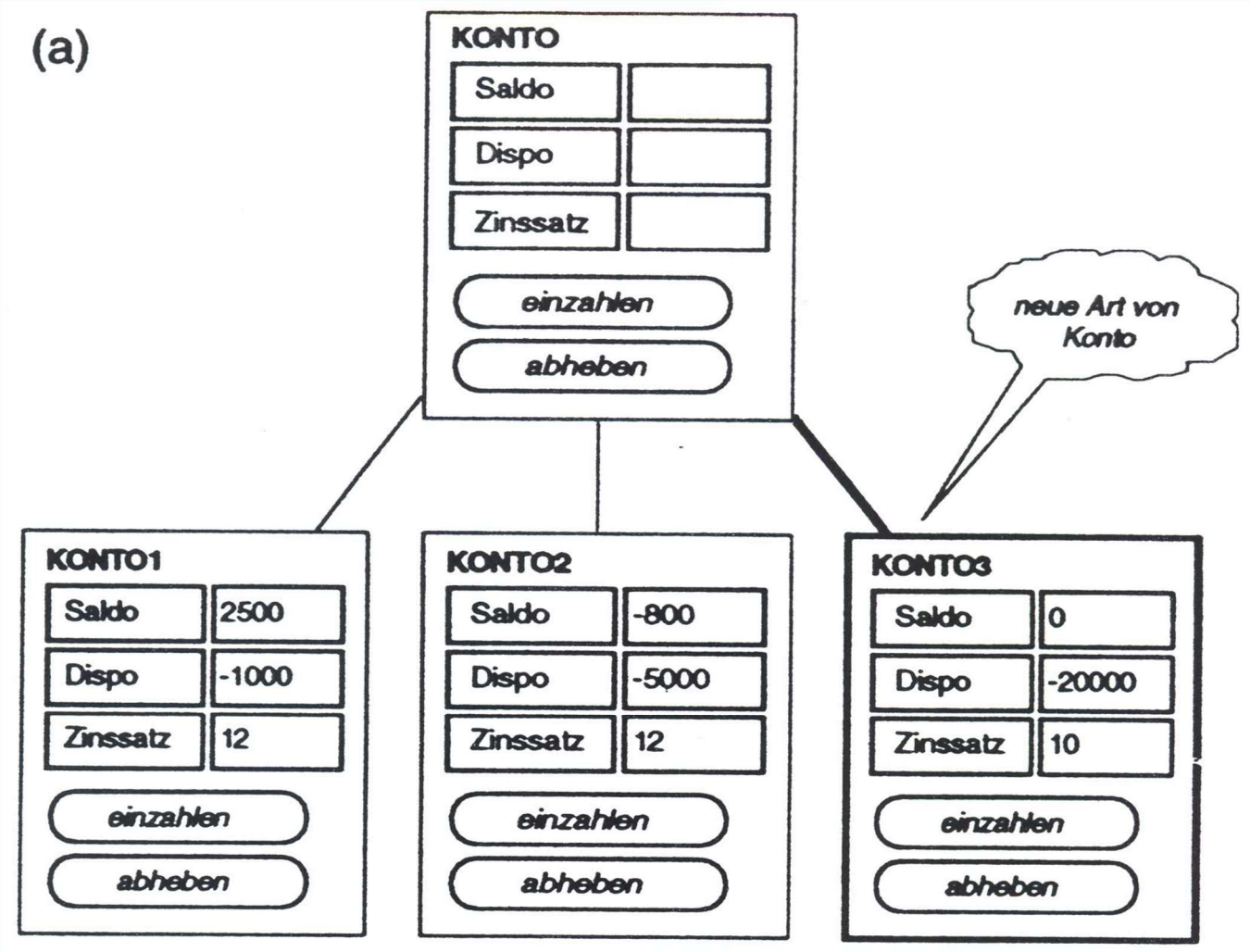
5. Nepravidlové a hybridní znalostní systémy



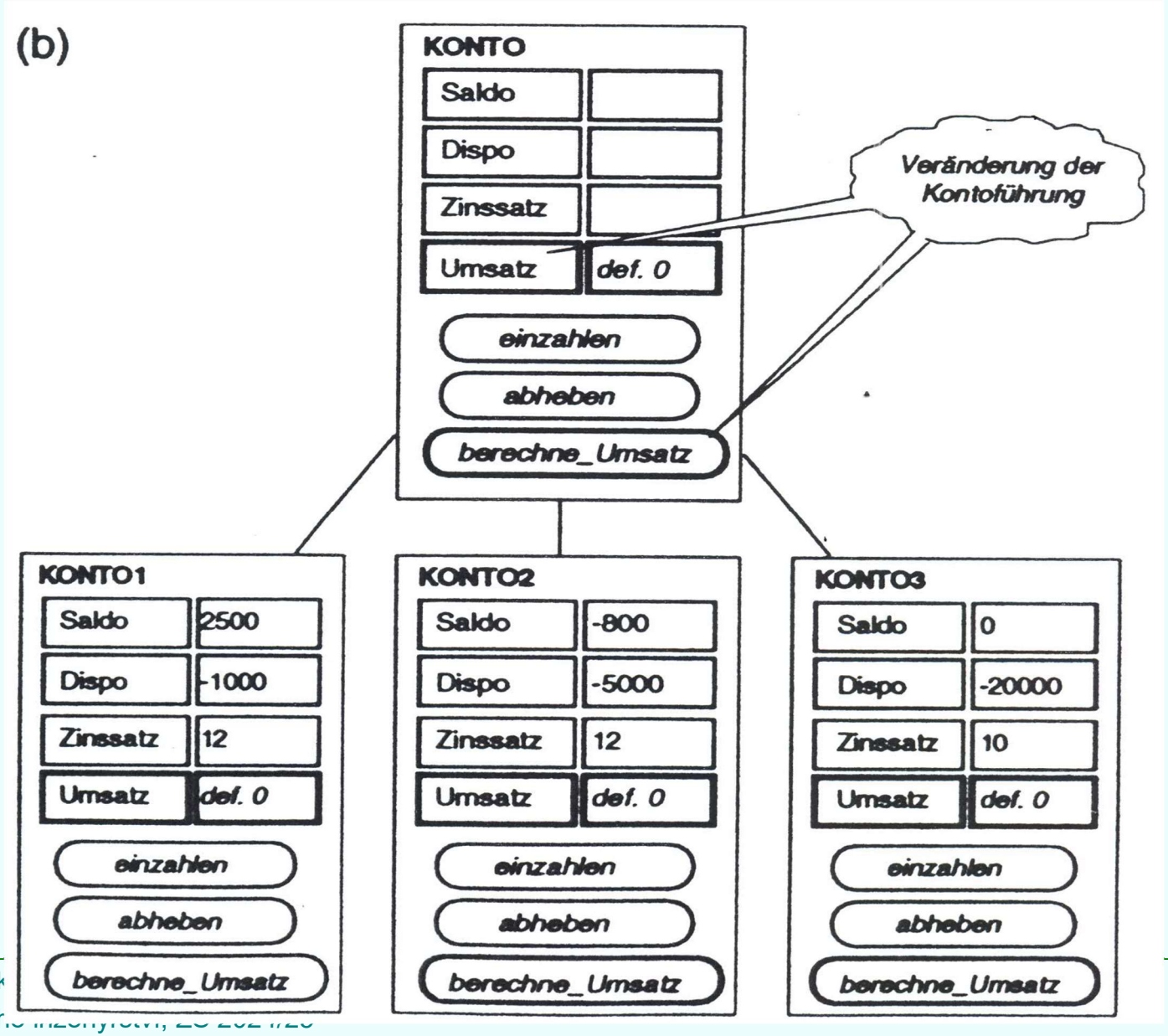
Příklad 2: Reprezentace ubytování v hotelu



Příklad 3: Reprezentace bankovních kont



5. Nepravidlové a hybridní znalostní systémy



Výhody a nevýhody rámcových systémů

Výhody:

- snazší usuzování řízené očekáváním (na základě využití démonů),
- organizace znalostí (větší strukturovanost a organizace než v sémantických sítích),
- samořízení (schopnost rámců určit svou vlastní aplikovatelnost v dané situaci),
- uchovávání dynamických hodnot (ve slotech rámců); výhodné při simulaci, plánování, diagnostice, … .

Nevýhody:

- potíže s odlišností objektů od prototypu,
- obtížné přizpůsobení novým situacím,
- obtížný popis detailních heuristických znalostí.

Sémantická (asociativní) síť

Sémantická (asociativní) síť **(*semantic or associative net*)** je ohodnocený orientovaný graf. Uzly reprezentují objekty a hrany představují vztahy mezi objekty. Místo pojmu sémantická síť se někdy také používá pojem *asociativní síť*.

Sémantická síť poskytuje vyšší úroveň porozumění akcím, příčinám a událostem, které se vyskytují v odpovídající doméně. To umožňuje úplnější usuzování znalostního systému o problémech z této domény.

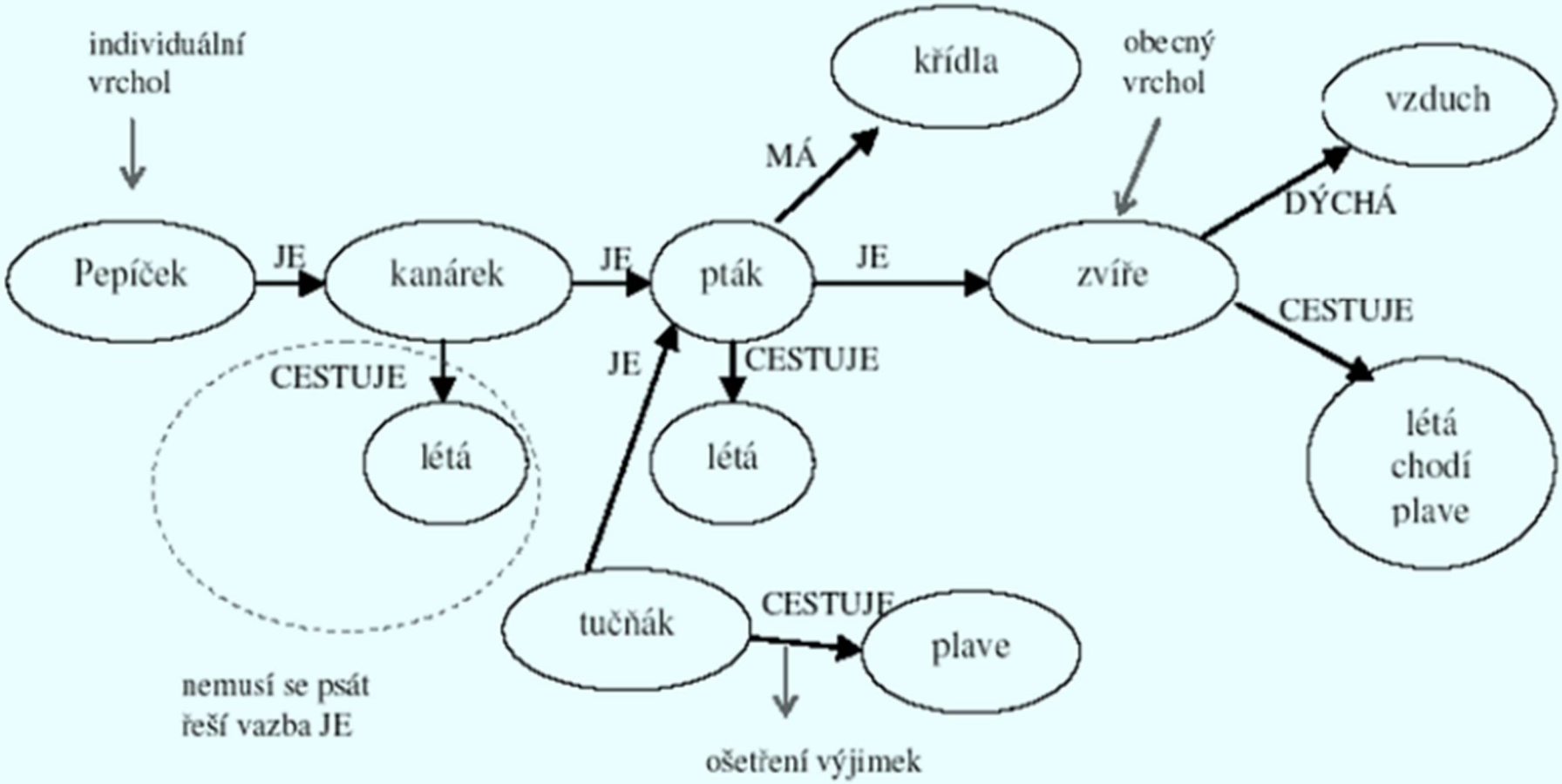
Vztahy v sémantické síti

Sémantická síť umožňuje reprezentaci fyzikálních, kauzálních a taxonomických vztahů a podporuje dědičnost a tranzitivitu.

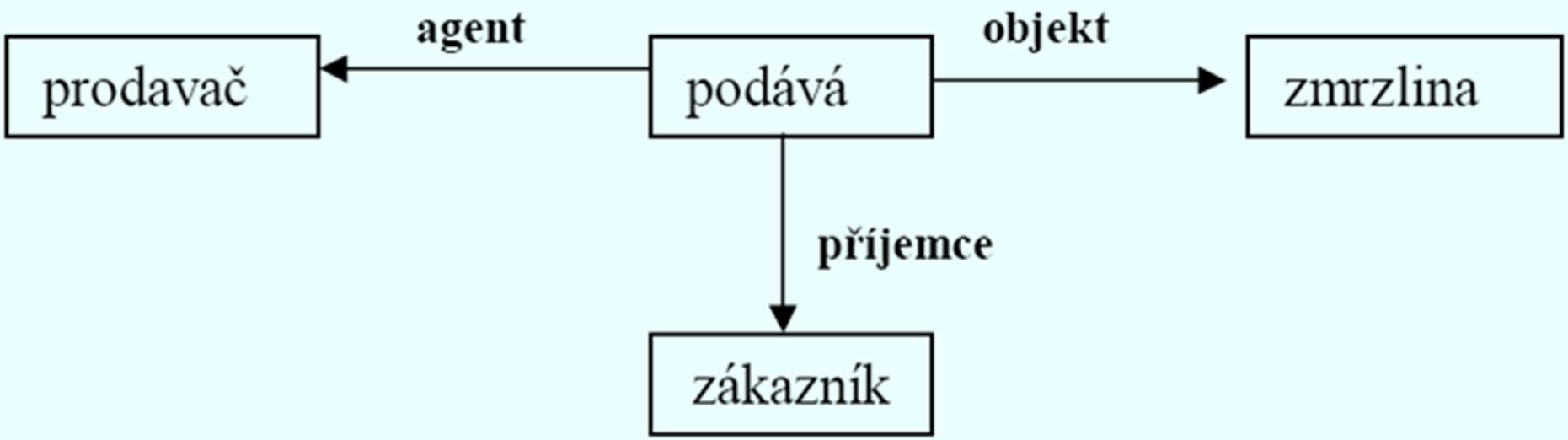
Příklady vztahů v sémantické síti: *is-a*, *has-a*, *part-of*, *number-of*, *connected-to*, *causes*, …

Je nutné dát si pozor na interpretaci vztahu *is-a*, který může mít např. tyto významy: *je instancí, je prvkem, je podmnožinou, je podtřídou, je ekvivalentní s … .*

Příklad 1: Reprezentace kanárka Pepíčka



Příklad 2: Reprezentace větné sémantiky

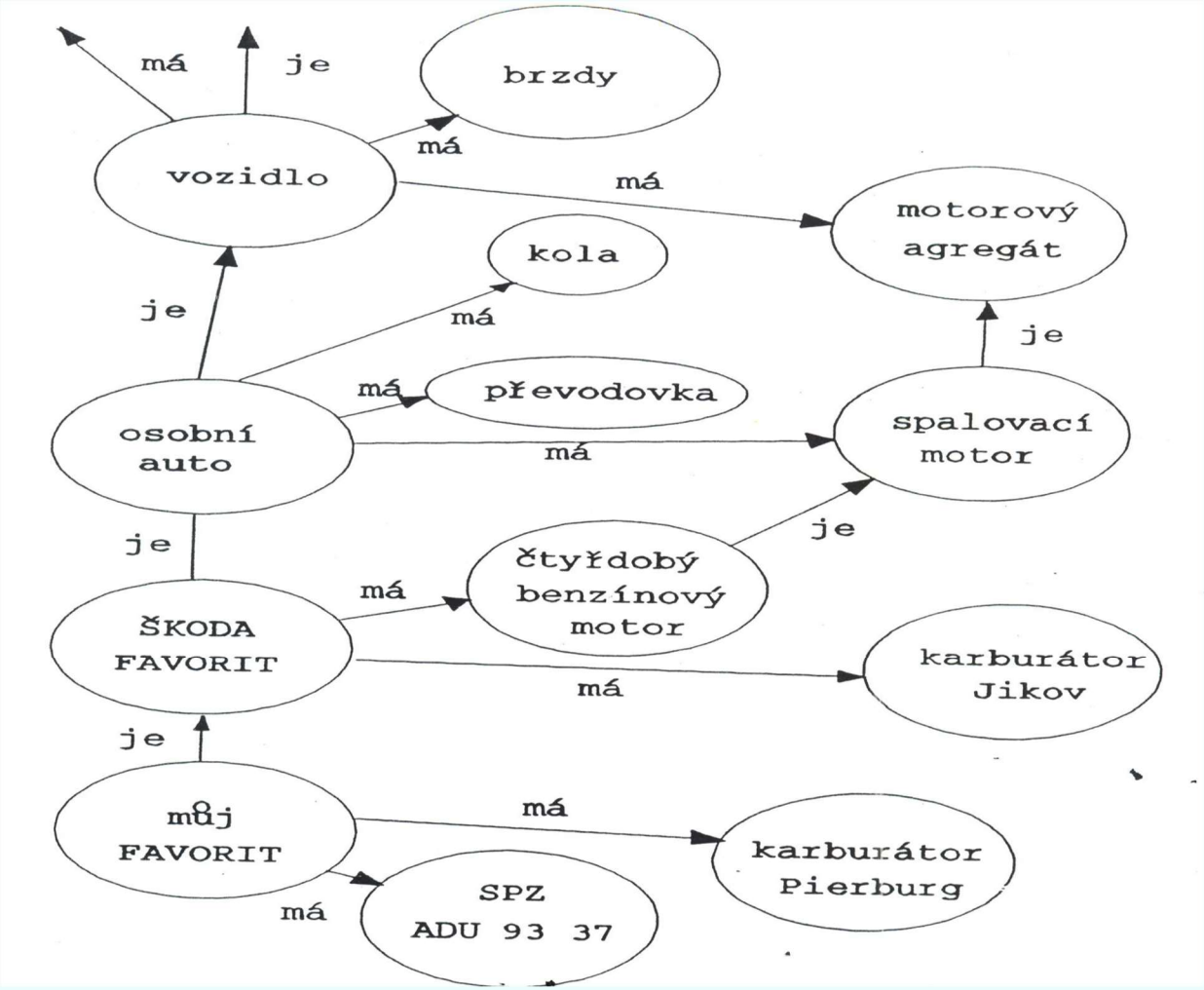


Porovnej: Reprezentace „Favorita“ rámci a sémantickou sítí:

a) rámci

můj Škoda Favorit Škoda Favorit osobní auto vozidlo	můj Škoda Favorit má SPZ ADU 93 37 karburátor Pierburg čtyřdobý benzínový motor karburátor Jikov kola převodovku spalovací motor motorový agregát brzdy

b) sémantickou sítí



Výhody a nevýhody sémantických sítí

Výhody:

- explicitní a jasné vyjádření,
- redukce doby hledání (pro dotazy typu dědičnosti nebo rozpoznávání).

Nevýhody:

- neexistence interpretačních standardů,
- nebezpečí chybné inference,
- nebezpečí kombinatorické exploze.

Petriho sítě (Petri nets)

Byly vytvořeny roku 1962 Carlem Adamem Petrim

Definovány jako orientovaný bipartitní graf s ohodnocením, skládá se z míst, přechodů a hran

Místa obsahují libovolný počet prvků (tokenů)

Rozložení tokenů se nazývá značení (marking)

Hrany spojují místa s přechody –

- místo, z něhož vede hrana do přechodu – vstupní místo
- místo, do něhož vede hrana z přechodu – výstupní místo

Přechody „odpalují“ tokeny ze vstupních míst do výstupních

- odpálení je „uschopněno“, pokud je v každém ze vstupních míst alespoň jeden token

Základní Petriho sítě:

Petriho sítě obsahují **místa**, **přechody** a **hrany**. Hrany jsou pouze mezi místy a přechody, nikoliv mezi dvěma místy nebo dvěma přechody. Místa, ze kterých vedou hrany do přechodu, jsou nazývána *vstupní místa* tohoto přechodu; místa, do kterých vedou hrany z přechodu, jsou nazývána *výstupní místa* tohoto přechodu.

Místa mohou obsahovat libovolný počet **teček** (někdy též *značek* nebo *tokenů*; anglicky: *tokens*). Rozložení značek mezi místy v síti je nazýváno **značení** (anglicky: *marking*). Přechody mohou tečky ze vstupních míst takzvaně **odpalovat** (anglicky: *firing*) do míst výstupních. Přechod je **uschopněn** (anglicky: *enabled*) a může odpálit, pokud je v každém ze vstupních míst alespoň tečka. Když přechod odpálí, odebere tečky z jeho vstupních míst, provede nějaké výpočetní úlohy, a vloží zvolený počet teček do každého výstupního místa. Tento proces činí automaticky v každém jednotlivém kroku.

Výpočet Petriho sítě je nedeterministický. Což znamená následující:

1. více přechodů může být uschopněno současně a libovolný může odpálit,
2. žádný přechod není nutno odpálit – odpalují se dle libosti mezi časy 0 až nekonečno nebo vůbec nikdy. Je tedy možné, že se neodpálí vůbec nic.

Protože je odpalování nedeterministické, jsou Petriho sítě vhodné např. pro modelování souběžného chování distribuovaných systémů nebo systémů hromadné obsluhy.

Poznámka: Stručný a velmi pěkný popis Petriho sítí (i když ne vyčerpávající) lze nalézt na adrese:

https://www.fd.cvut.cz/department/k611/pedagog/K611THO_soubory/webskriptum/11_petriNet.html

Formální definice

Petriho síť je pětice (S, T, F, M_0, W), kde:

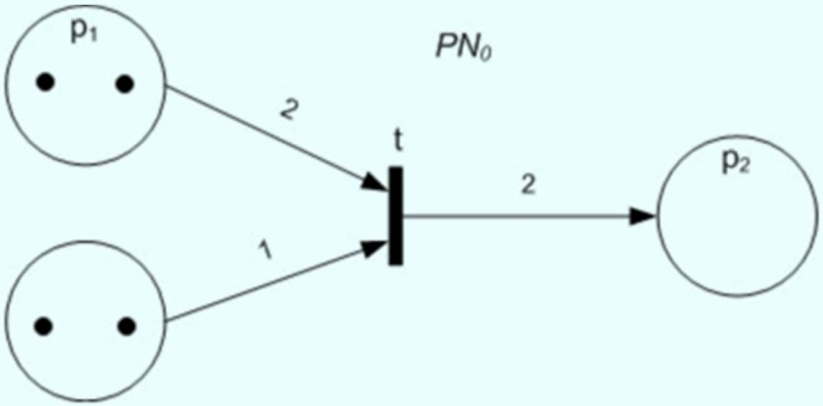
- S je konečná množina *míst*.
- T je konečná množina *přechodů*.
- F je konečná množina *hran*, kde žádná hrana nemůže spojovat dvě místa nebo dva přechody, formálněji: $F \in (S \times T) \cup (T \times S)$.
- $M_0 : S \rightarrow \mathbb{N}$ je *počáteční označkování*, kde v každém místě $s \in S$ je $n \in \mathbb{N}$ teček.
- $W : F \rightarrow \mathbb{N}^+$ je množina *vážených hran*, která přiřazuje každé hraně $f \in F$ nějaké číslo $n \in \mathbb{N}^+$ označující kolik teček je odebráno z místa tímto přechodem, nebo alternativně: kolik teček je přechodem produkováno a vloženo do každého výstupního místa.

Pozn.: Existuje více variant formálních definic – některé nemají vážené hrany, ale povolují více hran mezi stejným místem a přechodem.

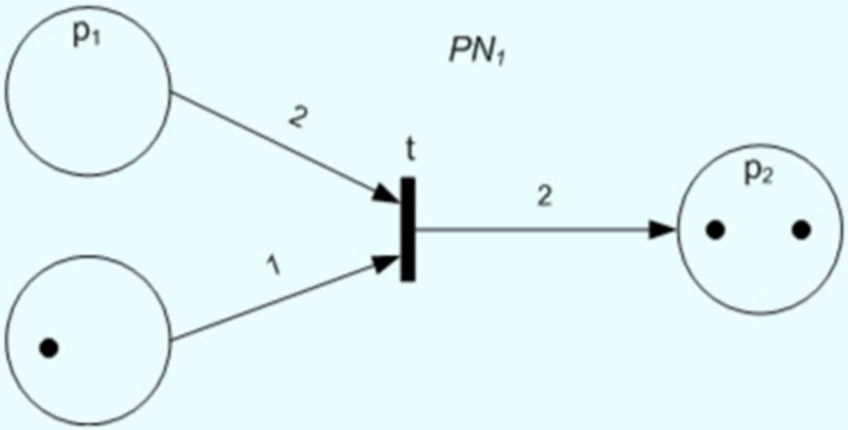
Typy Petriho sítí

- C/E (Condition/Event) Petriho sítě,
- P/T (Place/Transitions) Petriho sítě,
- P/T Petriho sítě s inhibičními hranami,
- P/T Petriho sítě s prioritami,
- TPN Časované (Timed) Petriho sítě,
- CPN Barevné (Coloured, barvené) Petriho sítě,
- HPN Hierarchické (Hierarchical) Petriho sítě,
- OOPN Objektové (Object Oriented) Petriho sítě.

Příklad P/T sítě:



1. odpálení je „uschopněno“



2. stav po „odpálení“

Výhody a nevýhody Petriho sítí

Výhody:

- srozumitelné a jednoduše analyzovatelné,
- možnost grafického vyjádření,
- možnost graficky simulovat dynamické chování modelu.

Nevýhody:

- použitelné jen pro jednodušší úlohy (modely),
- každý reálný systém je možné modelovat mnoha různými způsoby,
- dnes mnoho typů sítí, otázka volby typu sítě složitá.

Objekty

Objekty podobně jako rámce sdružují deklarativní znalosti a procedurální znalosti.

Objekt je programová struktura, obsahující jak data, tak metody (procedury), které s těmito daty pracují. Data objektu jsou přístupná pouze prostřednictvím metod objektu. Tato vlastnost se označuje jako zapouzdření (encapsulation). Objekt je instance třídy.

Třída je skupina objektů, které mají stejné vlastnosti (datové složky) a stejné chování (metody).

Příklady objektových jazyků:

- jazyky čistě objektové (Smalltalk, Actor, CLOS, …)
- jazyky podporující OOP, ale umožňující programovat i neobjektově (Java, C++, C#, Python, PHP, Perl, …)

TOP programovací jazyky:

Pořadí pro:	2017	2018	2019	2020	2021
1.	Python	JavaScript	Java	Java	Python
2.	Java	Python	JavaScript	Python	C#
3.	R	C#	C++	C	Java
4.	JavaScript	Java	Python	Ruby	C
5.	Swift	PHP	C#	JavaScript	Ruby
6.	C++	Go	PHP	C#	Objective-C
7.	C#	Swift	Perl	PHP	Perl
8.	PHP	Rust	SQL	Perl	PHP
9.	SQL	Kotlin		Objective-C	JavaScript
10.	Go	C & C++		SQL	

Základními vlastnostmi objektů jsou:

Abstrakce od detailů světa a vnitřní implementace objektu. Objekt zachycuje jen ty rysy reality, které jsou podstatné pro řešení úlohy.

Zapouzdření vnitřních atributů a metod. Objekt je představován černou skříňkou. Navenek poskytuje bez ohledu na vnitřní implementaci jen jasně definované rozhraní. To zajišťuje stabilní chování objektu, které je závislé jen na vstupních proměnných.

Dědičnost – objekty (potomci) dědí své vlastnosti (atributy, metody) od svého rodiče nebo i více rodičů současně. Mohou je doplňovat, nebo i transformovat, například změnit vnitřní implementaci dané metody. Hierarchie objektů je dána uspořádáním jejich vzorů – tříd.

Polymorfismus pro různé potomky téže třídy může být vnitřní implementace a provádění metod různé, byť navenek se objekty různých tříd tváří stejně v relaci ke svým předkům. Potomek umí minimálně totéž, co jeho rodič.

Vztahy mezi třídami

Dědičnost:

Od jedné třídy (bázové, rodičovské) můžeme odvodit třídu jinou (odvozenou, dceřinou).

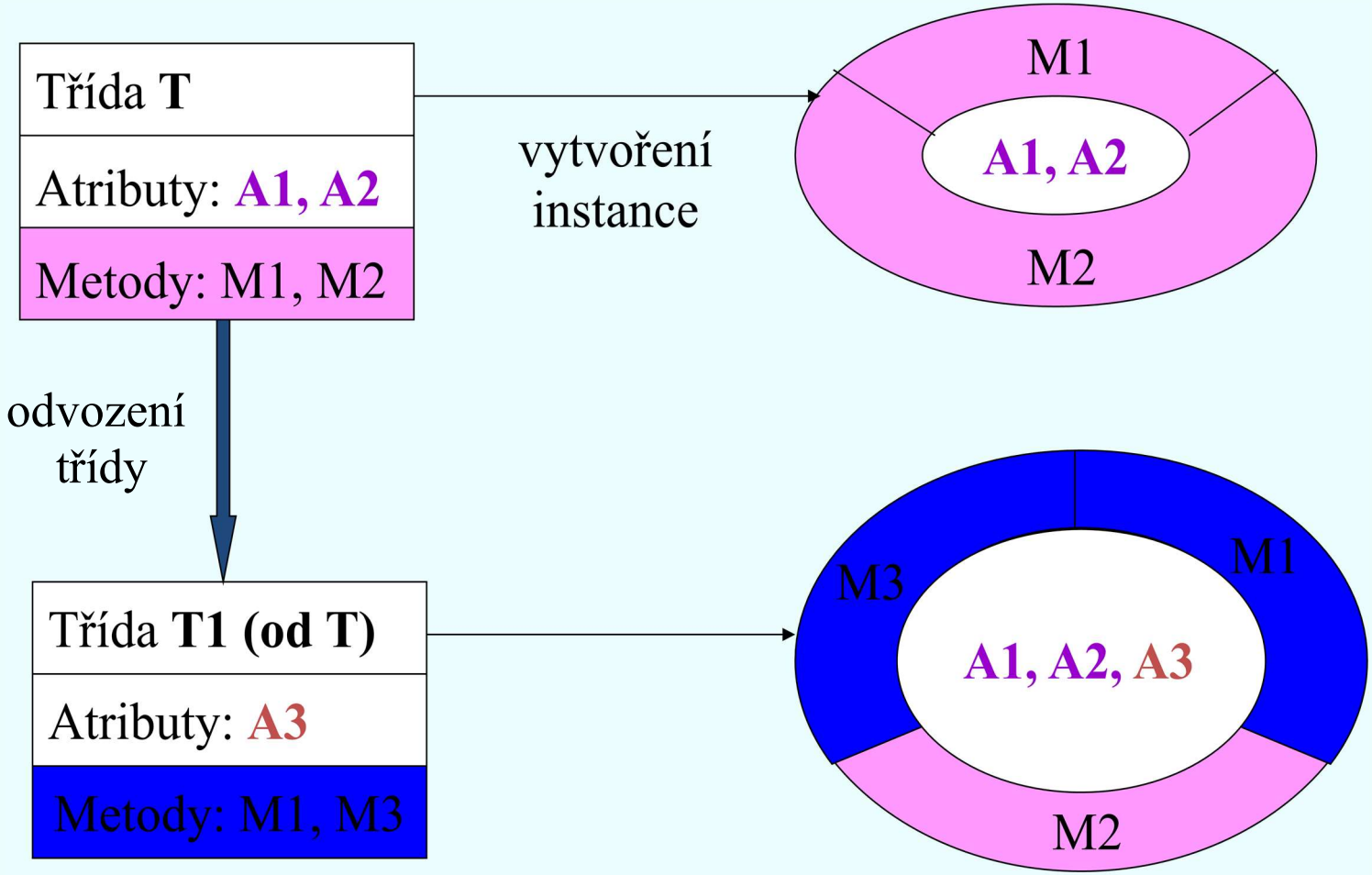
Dceřiná třída dědí všechny složky své rodičovské třídy a k nim může přidat svoje vlastní. Zděděné metody je pak možno předefinovat.

Objekt třídy předek může být v programu zastoupen objektem třídy potomek (může se jednat i o nepřímého potomka).

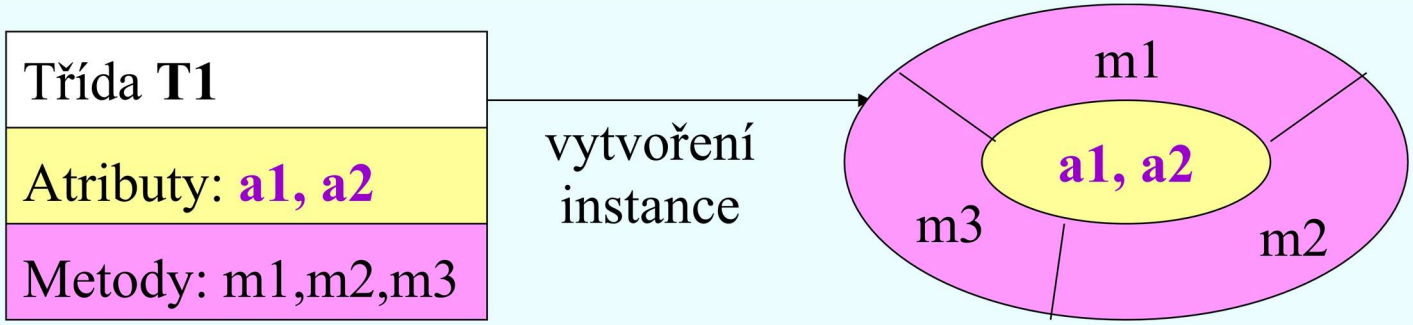
Vlastnictví (skládání):

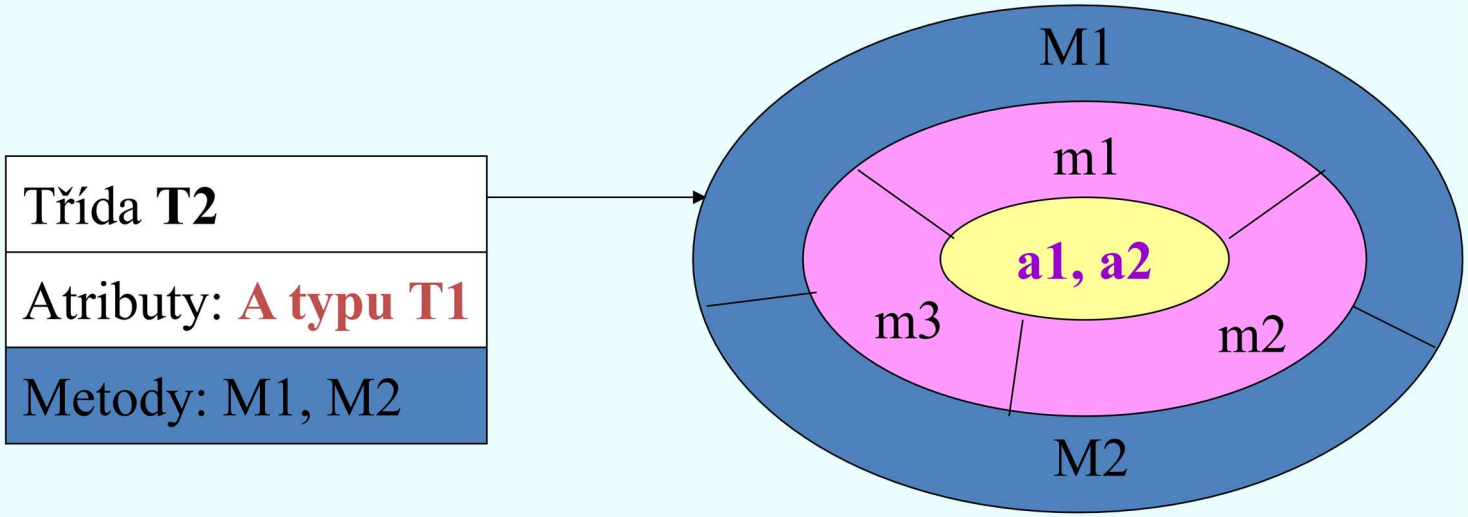
Složkou třídy může být jiná třída.

Třídy, objekty a dědičnost



Skládání tříd a objektů





Komunikace mezi objekty

Objekty spolu komunikují tak, že si navzájem posílají zprávy; obvykle to znamená, že jeden objekt (odesílatel zprávy) volá metodu jiného objektu (příjemce zprávy).

Časná vazba (*early binding*) – příjemce zprávy je určen v okamžiku kompilace.

Pozdní vazba (*late binding*) – příjemce zprávy je určen až za běhu programu; pozdní vazbou se realizuje ***polymorfismus*** (mnohotvarost).

Výhody a nevýhody objektů

***Výhody*:**

- abstrakce
- zapouzdření (ukrývání informace)
- dědičnost
- polymorfismus
- znovupoužitelnost kódu

***Nevýhody* (podobné jako u rámců):**

- jak se vypořádat s odchylkami od normy ?
- jak zohlednit nové dosud neuvažované situace ?

Hybridní systémy

Zatímco u 1. generace znalostních (expertních) systémů byl v rámci jednoho systému používán pouze jeden způsob reprezentace znalostí, znalostní systémy 2. generace obvykle používají hybridní (kombinované) reprezentace znalostí.

Hybridní reprezentace, které jsou dostupné v nejčastěji používaných prostředcích pro vývoj ZS, kombinují pravidlově, rámcově a objektově orientované techniky. Umožňují tzv. *modelový* přístup k tvorbě systému a usnadňují vývoj modelů.

Hybridní systémy, kombinující rámce a pravidla, používají např. rámce pro reprezentaci strukturních znalostí a pravidla pro usuzování o těchto znalostech. Rámce také mohou být využity pro implementaci sémantických sítí.

Příklady hybridních systémů

ACQUIRE	prázdný ZS, pravidla, rámce a objekty
CLIPS	programové prostředí, objekty a pravidla
FLEX	programové prostředí implementované v Prologu, pravidla a rámce, dopředné a zpětné řetězení
G2	objektově orientované prostředí, pravidla, modely a procedury, diagnostické a řídicí aplikace
Rete++	programové prostředí, pravidla a objekty, dopředné a zpětné řetězení
Rtworks	programové prostředí, objekty a pravidla
XpertRule	programové prostředí, rozhodovací stromy a tabulky příkladů