Provozně ekonomická fakulta

2014, Brno

Expertní systémy

(Pravidlové, nepravidlové, tvorba ES)

Expertní systém

- **Definice ES** (Feigenbaum): expertní systémy jsou počítačové programy, simulující rozhodovací činnost experta při řešení složitých úloh a využívající vhodně zakódovaných, explicitně vyjádřených znalostí, převzatých od experta, s cílem dosáhnout ve zvolené problémové oblasti kvality rozhodování na úrovni experta.
- **Charakteristické rysy ES:**
  - oddělení znalostí a mechanismu jejich využívání,
  - rozhodování za neurčitosti,
  - schopnost vysvětlování.

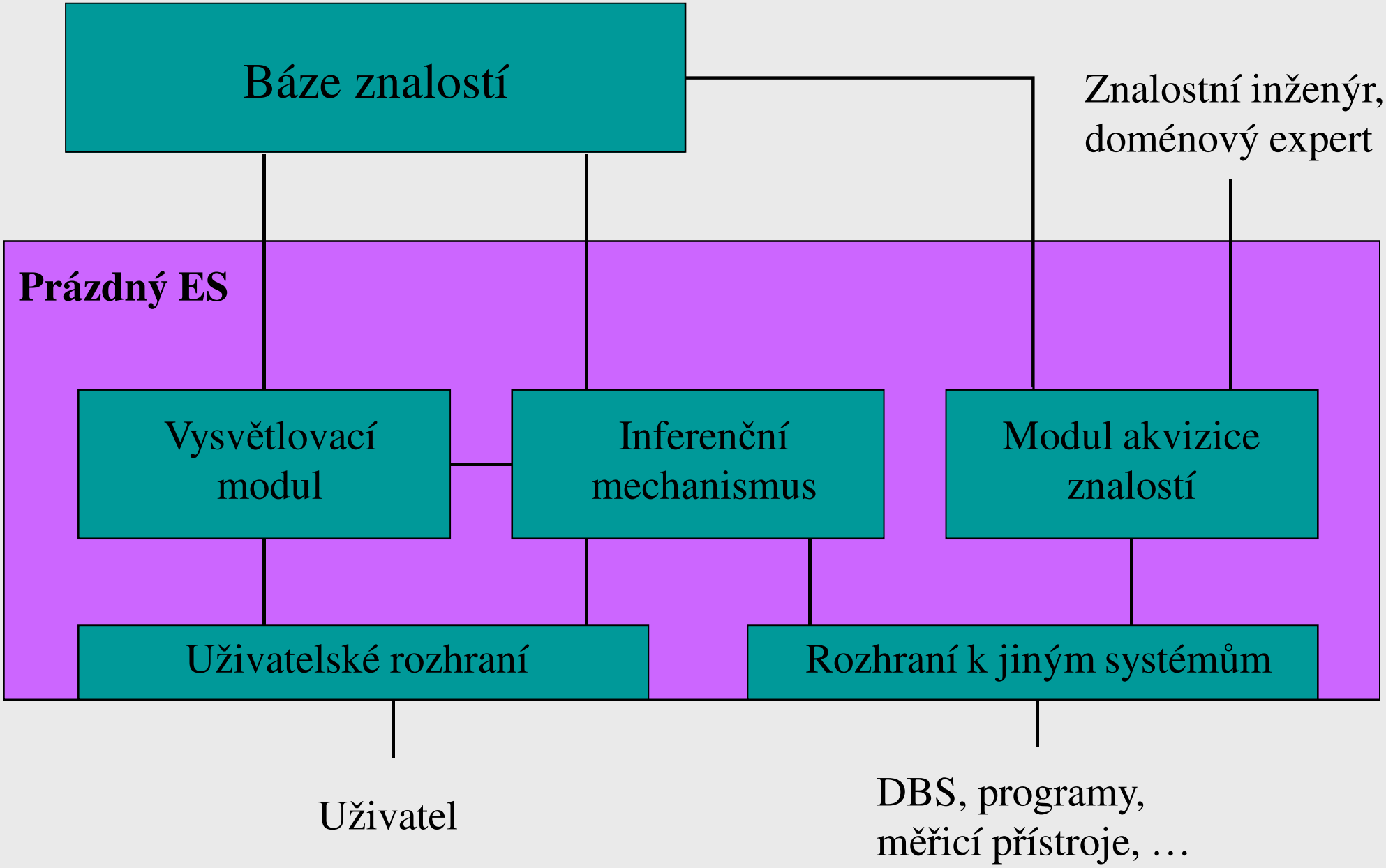
Expertní systémy a znalostní systémy

- **Znalostní systém** (*knowledge-based system*) je podle staršího pojetí obecnější pojem než expertní systém. Expertní systémy tedy lze chápat jako zvláštní typ znalostních systémů, který se vyznačuje používáním expertních znalostí a některými dalšími rysy, jako je např. vysvětlovací mechanismus.
- V poslední době dochází ke stírání rozdílů mezi těmito pojmy.

Základní složky ES

- báze znalostí,
- inferenční mechanismus,
- I/O rozhraní (uživatelské, vývojové, vazby na jiné systémy),
- vysvětlovací modul,
- modul pro akvizici znalostí.

Architektura ES



Báze znalostí a báze faktů

- **Báze znalostí** obsahuje znalosti z určité domény a specifické znalosti o řešení problémů v této doméně.
- **Báze faktů** se vytváří v průběhu řešení konkrétního problému a obsahuje data k řešenému problému.
- **Prostředky reprezentace znalostí:**
  - matematická logika,
  - pravidla (rules),
  - sémantické sítě (semantic nets),
  - rámce a scénáře (frames and scripts),
  - objekty (objects).

Inferenční mechanismus

- **Inferenční mechanismus** obsahuje obecné (doménově nezávislé) algoritmy schopné řešit problémy na základě manipulace se znalostmi z báze znalostí.

- Typický inferenční mechanismus je založen na
  - inferenčním pravidle pro odvozování nových poznatků z existujících znalostí,
  - strategii prohledávání báze znalostí.

Neurčitost v expertních systémech

- Neurčitost se může vyskytovat jednak v bázi znalostí a jednak v bázi faktů.
- **Zdroje neurčitosti:**
  - nepřesnost, nekompletnost, nekonzistence dat,
  - vágní pojmy,
  - nejisté znalosti.
- **Prostředky pro zpracování neurčitosti:**
  - Bayesovský přístup, Bayesovské sítě
  - faktory jistoty,
  - Dempster-Shaferova teorie,
  - fuzzy logika.

Typy ES:

- ***Problémově orientovaný ES:*** báze znalostí obsahuje znalosti z určité domény.
- ***Prázdný ES (shell):*** báze znalostí je prázdná.
- ***Diagnostický ES:*** jeho úkolem je určit, která z hypotéza z předem definované konečné množiny cílových hypotéz nejlépe koresponduje s daty týkajícími se daného konkrétního případu.
- ***Plánovací ES:*** obvykle řeší takové úlohy, kdy je znám cíl řešení a počáteční stav a je třeba s využitím dat o konkrétním řešeném případu nalézt posloupnost kroků, kterými lze cíle dosáhnout.

Tvorba ES

- Tvorba ES zahrnuje procesy:
  - akvizice znalostí (získání a reprezentace znalostí),
  - návrh uživatelského rozhraní,
  - výběr hardwaru a softwaru,
  - implementace,
  - validace a verifikace.
- Vytvářením ES se zabývá **znalostní inženýrství** (*knowledge engineering*). V procesu tvorby ES představuje úzké místo akvizice znalostí (*knowledge acqusition bottleneck*). Toto úzké místo pomáhají překonat metody **strojového učení** (*machine learning*).

Nástroje pro tvorbu expertních systémů

- **Prázdné expertní systémy:**
  EXSYS, FLEX, G2, HUGIN, M4, ...

- **Speciální programová prostředí:**
  CLIPS, OPS5, Lisp, Prolog, ...

- **Obecná programová prostředí:**
  Pascal, Delphi, C, C++Builder, ...

Aplikace ES

- Aby mělo smysl použít expertní systém pro řešení nějakého problému, musejí být splněny dvě podmínky:
- Musí se jednat o problém složitý rozsahem nebo neurčitostí vztahů, pro nějž exaktní metoda řešení buď není k dispozici, nebo není schopna poskytnout řešení v požadované době.
- Efekty plynoucí z použití expertního systému musejí převyšovat vynaložené náklady. To znamená, že by mělo jít o problém s opakovanou potřebou řešení a značnými finančními dopady, pro nějž lidští experti jsou drazí nebo omezeně dostupní.

Historie vývoje ES

- Poté, co při řešení praktických problémů selhaly obecné metody řešení, byla pochopena nutnost využívat specifické (expertní) znalosti z příslušné problémové domény.
- **Etapy vývoje:**
  - 1965-70 počáteční fáze (Dendral)
  - 1970-75 výzkumné prototypy (MYCIN, PROSPECTOR, HEARSAY II)
  - 1975-80 experimentální nasazování
  - 1981 komerčně dostupné systémy

1. generace ES

- Charakteristické rysy 1. generace ES:
  - jeden způsob reprezentace znalostí,
  - malé schopnosti vysvětlování,
  - znalosti pouze od expertů.

2.generace ES

- Charakteristické rysy 2.generace ES:
  - modulární a víceúrovňová báze znalostí,
  - hybridní reprezentace znalostí,
  - zlepšení vysvětlovacího mechanismu,
  - prostředky pro automatizované získávání znalostí.
- V rámci 2. generace ES se také objevují ***hybridní systémy***, v nichž se klasické paradigma expertních systémů kombinuje s dalšími přístupy, jako jsou neuronové sítě a evoluční metody.

Výhody a nevýhody ES

- **Výhody ES:**
  - schopnost řešit složité problémy,
  - dostupnost expertíz a snížené náklady na jejich provedení,
  - trvalost a opakovatelnost expertízy,
  - trénovací nástroj pro začátečníky,
  - uchování znalostí odborníků odcházejících z organizace.
- **Nevýhody ES:**
  - nebezpečí selhání ve změněných podmínkách,
  - neschopnost poznat meze své použitelnosti.

Tvorba expertního systému

Znalostní inženýrství

- Problematikou tvorby expertních (znalostních) systémů se zabývá *znalostní inženýrství* (*knowledge engineering*).
- Znalostní inženýrství má mnoho společných rysů se softwarovým inženýrstvím. Odlišnosti se týkají typu, povahy a množství reprezentovaných znalostí.
- U softwarového inženýrství se jedná o dobře definované algoritmické znalosti. Povahu a množství těchto znalostí potřebných pro řešení daného problému lze předem dobře odhadnout.
- U znalostního inženýrství se jedná o extenzivní, nepřesné a špatně definované znalosti, jejichž povahu a množství lze předem velmi špatně odhadnout. To způsobuje potíže v počátečních etapách vývoje ES při odhadu potřebného úsilí a při tvorbě návrhu.

Životní cyklus expertního systému

- Model životního cyklu ES kombinuje *rychlé prototypování* a *inkrementální vývoj* a obsahuje tyto etapy:
    1. Analýza problému.
    2. Specifikace požadavků.
    3. Předběžný návrh.
    4. Počáteční (rychlé) prototypování a vyhodnocování.
    5. Konečný návrh.
    6. Implementace (získávání a reprezentace znalostí).
    7. Validace a verifikace (testování).
    8. Změny návrhu.
    9. Údržba.
- Etapy 6, 7 a 8 se iteračně opakují pro jednotlivé části (subsystémy) expertního systému.

Analýza problému

- Cílem analýzy je posoudit vhodnost aplikace znalostních technik pro řešení daného problému.

- Kritéria pro toto posouzení mohou být rozdělena do dvou skupin:
  - Vhodnost aplikace
  - Dostupnost zdrojů

Vhodnost aplikace

- Problém skutečně existuje?
- Jsou pro něj vhodné znalostní techniky?
  - Mohou být replikovány lidské znalosti řešení problému?
  - Znalosti jsou převážně heuristické?
  - Jsou tyto znalosti dobře chápány a akceptovány?
  - Expertízy se často mění (nejsou konstantní)?
  - Vstupní data jsou nekompletní nebo nepřesná?
  - Znalostní přístup k řešení je lepší než jiné prostředky?

  Odpovědi na tyto otázky mají různou váhu a nemusejí být všechny kladné. Musejí být posuzovány jako celek s ohledem na konkrétní podmínky).
- Je znalostní přístup oprávněn z hlediska nákladů a přínosů?

Dostupnost zdrojů

- Má projekt manažerskou podporu?
  - dostatek času
  - potřebné prostředky a školení
  - disponibilita expertů
- Je podpora ze strany expertů?
- Jsou experti kompetentní?
- Jsou experti komunikativní?
- Jsou experti fyzicky dostupní?
- Jsou k dispozici jiné zdroje znalostí?

Struktura specifikace požadavků

- Úvod
  - Charakteristika problému, profil uživatelů, cíle projektu.
- Funkce expertního systému.
  - Vstupy a výstupy systému, pomocné funkce, implementační priority.
- Omezení
  - Hardwarová omezení, externí rozhraní, kompatibilita s předchozími produkty, rychlost, spolehlivost, udržovatelnost, bezpečnost, identifikace chyb.
- Závěrečné požadavky
  - Metody validace a verifikace, požadavky na dokumentaci, jiné požadavky.

Předběžný návrh

- Výběr paradigmatu reprezentace znalostí.
  - Pravidla nebo logika – vhodné pro mělké znalosti.
  - Struktury (rámce objekty, sémantické sítě) – vhodné pro hluboké a strukturálně provázané znalosti.
  - Hybridní systémy – spojení strukturálních znalostí se schopností inference.
- Výběr metod usuzování (souvisí s volbou reprezentace).
- Výběr nástrojů (komerční nebo zákaznický systém?).
- Výběr lidských zdrojů (znalostní inženýři, vedoucí týmu, experti).
- Požadavky na vývojový tým (dány zejména složitostí a rozsahem systému).

Kritéria pro výběr komerčního shellu

- Paradigma reprezentace znalostí a usuzování.
- Flexibilita.
  - Uživatelsky definované funkce, externí rutiny, vestavěné funkce, podpora datových struktur.
- Speciální požadavky.
  - Časové usuzování, operace v reálném čase, zpracování neurčitosti, přístup k externímu softwaru, grafika, okna.
- Pomocné funkce.
  - Editor znalostní báze, trasování, vysvětlování, testovací a verifikační pomůcky, grafická prezentace znalostní báze.
- Výkon.
- Podpora výrobce.
  - Dokumentace, on-line help, podpora horkou linkou, školení, konzultace.
- Náklady

Rychlé prototypování

- Rychlé prototypování (*rapid prototyping*) využívá prostředky jako Lisp, Prolog a/nebo komerčně dostupné prázdné ES (*shells*) s cílem rychle vytvořit fungující prototyp finálního systému.
- Na základě vyhodnocení prototypu musejí být všechna předběžná rozhodnutí potvrzena nebo změněna.
- Počáteční prototyp by měl mít dobré uživatelské rozhraní a rozumně robustní podmnožinu znalostí, aby zamýšlení uživatelé mohli posoudit jeho aplikovatelnost.
- Prototyp může být sice po vyhodnocení dále modifikován, ale doporučuje se jeho opuštění a započetí implementace ES na základě konečného návrhu od počátku.

Získávání znalostí

- Získávání znalostí (*knowledge acquisition*) je klíčovou operací implementace ES a představuje nejdelší a nejpracnější část vývoje ES.
- Akvizice znalostí je proces zjišťování (*elicitation*) ze zdrojů (expertů, textů, dat, obrázků, …) a jejich reprezentace v bázi znalostí.
- Proces naplňování báze znalostí probíhá inkrementálně (*incremental development*). Postupně jsou implementovány zvládnutelné a relativně ucelené části znalostí (subsystémy). Po implementaci každé části probíhá testování, na jehož základě mohou být provedeny případné změny v návrhu.

Způsoby získávání znalostí

- Získávání znalostí od expertů formou spolupráce mezi znalostními inženýry a experty
  - 1 : 1 (nejčastější případ)
  - 1 : *n*
  - *m* : 1
  - *m* : *n*
- Automatizované získávání znalostí (strojové učení)
  - od expertů
  - z textů
  - z dat (*data mining*)

Proces získávání znalostí od expertů

- Obvykle se proces dělí do tří fází:
  - Seznámení s problémem, získání základních znalostí (spolupráce nejen s expertem ale také se zadavatelem a uživatelem)
  - Získávání obecných znalostí.
  - Získávání specifických znalostí.
- Práce s jedním expertem:
  - obvykle formou interview;
  - nebezpečí zavlečení chybné expertízy
- Práce se skupinou expertů:
  - panelová diskuse, brainstorming;
  - nižší riziko chybných expertíz;
  - náročnější na přípravu a průběh,
  - nebezpečí konfliktů mezi experty

Příprava interview

- Optimalizace interview:
  - pečlivé naplánování a efektivní řízení průběhu
- Plánování interview:
  - místo konání – v počáteční fázi na pracovišti experta, později (je-li to možné) na pracovišti znalostního inženýra
  - doba trvání – kolem 2 hodin, rozhodně ne více než 3
  - cíle interview – stanoveny na základě přehledu výsledků předchozího sezení
- S plánem interview je třeba experta předem seznámit.

Techniky získávání znalostí od experta

- Nestrukturované interview (běžný rozhovor, vhodné pro počáteční fázi)
- Strukturované interview (kladení cílených dotazů, získání detailního pohledu)
- Myšlení nahlas (expert popisuje svá myšlenkové pochody a chování při řešení problému)
- Pokus o řešení problému pod dohledem experta s cílem vcítit se do jeho myšlenkových pochodů
- Metoda repertoárové tabulky (repertory grid)
  - sloupce odpovídají objektům z dané oblasti
  - řádky odpovídají konstruktům; každý konstrukt je tvořen dvěma mezními (nejlépe protikladnými) vlastnostmi objektů
  - políčka tabulky obsahují číselná ohodnocení příslušnosti objektu k jednomu či druhému pólu

Problémy práce s experty

- Paradox znalostního inženýrství:
  - Čím více se experti stávají kompetentními, tím méně jsou schopni popsat znalost, kterou používají při řešení problémů.
- Typy problémových expertů:
  - expert obávající se ztráty postavení po zavedení ES
  - cynický expert
  - velekněz oboru
  - paternalistický expert
  - nekomunikativní expert
  - lhostejný expert
  - pseudovzdělanec v umělé inteligenci

Nepravidlové a hybridní expertní systémy

Nepravidlové reprezentace znalostí

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

Sémantická síť

- ***Sémantická síť*** (***semantic net***) je ohodnocený orientovaný graf. Uzly reprezentují objekty a hrany představují vztahy mezi objekty. Místo pojmu sémantická síť se 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 si dát pozor na interpretaci vztahu *is-a*, který může mít např. tyto významy: *je instancí, je prvkem, je podmnožinou, je podtřídou, je ekvivalentní s.*

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.

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 se pokoušejí 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 slotu mohou být speciální procedury, jako např. ***if-needed***, ***if-changed***, ***if-added***, ***if-deleted***. Tyto procedury jsou automaticky aktivovány, jestliže nastanou příslušné situace.
- 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.

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.

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 (Borland Pascal, Object Pascal, C++, …)

Vztahy mezi třídami

- ***Dědičnost*:**

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

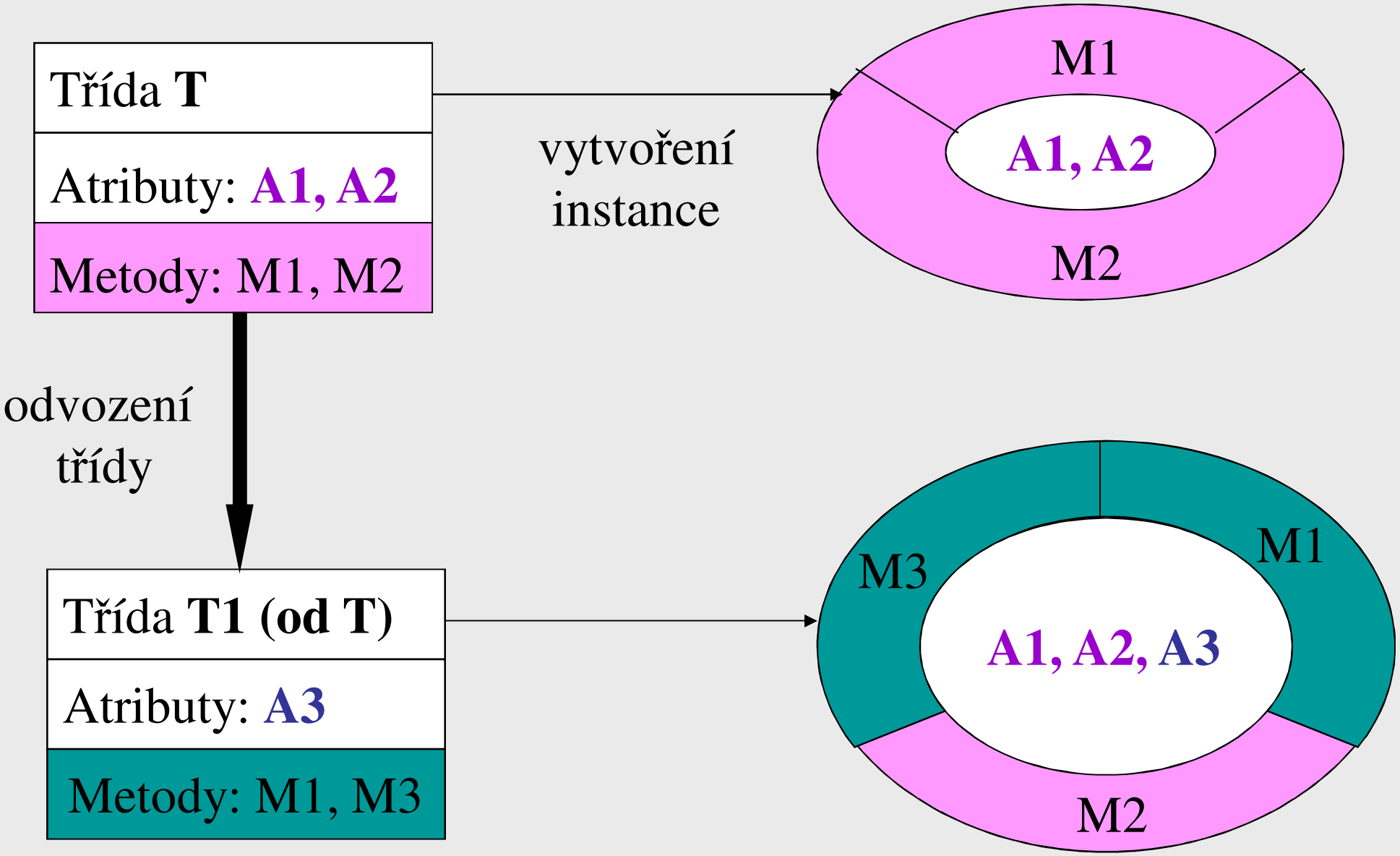
  Dceřinná třída dědí všechny složky své rodičovské třídy a k nim může přidat svoje vlastní. Zděděné metody je možno předefinovat.

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

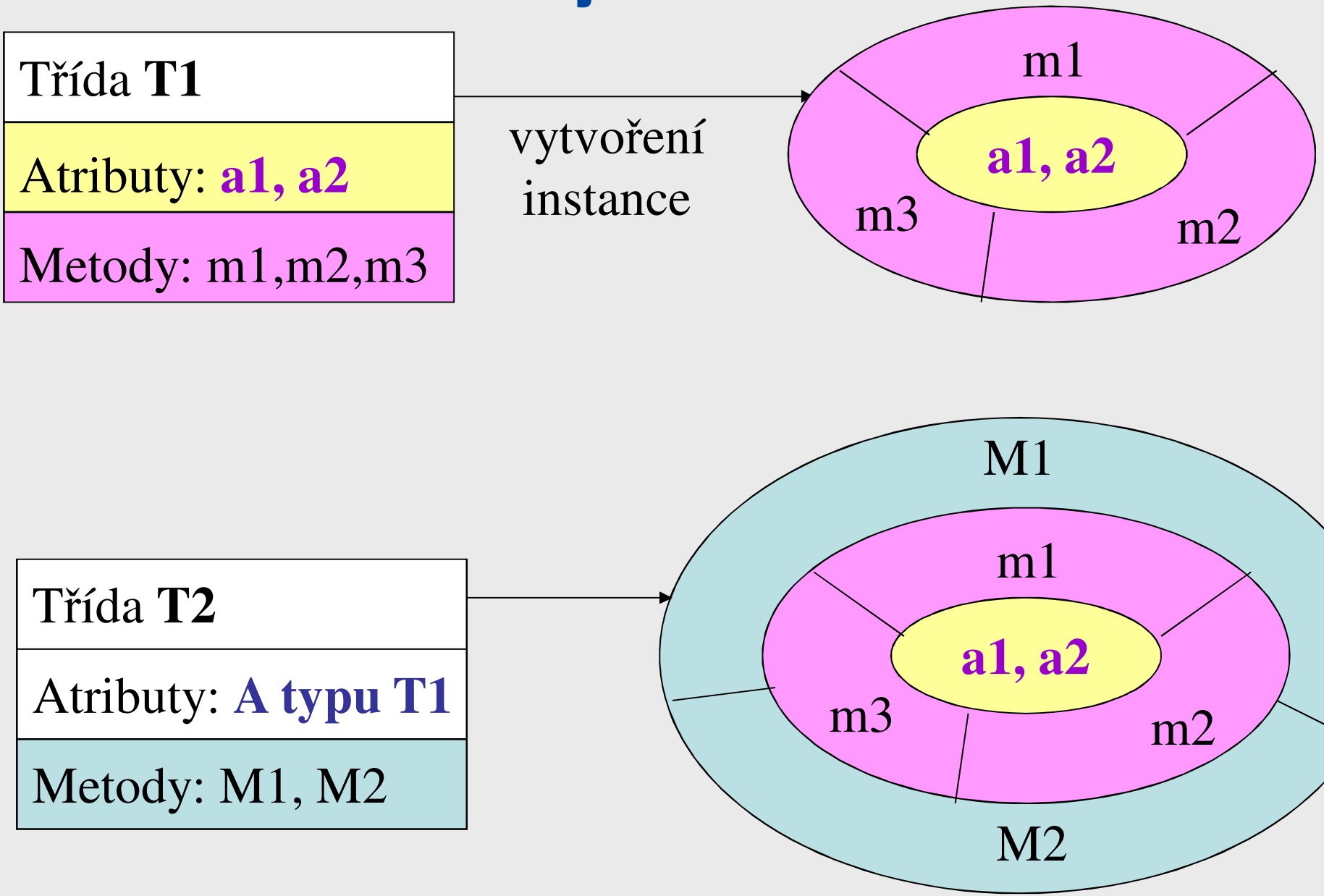
- ***Vlastnictví* (*skládání*):**

  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 (odesilatel 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; pomocí pozdní vazby 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 expertních systémů byl v rámci jednoho systému používán pouze jeden způsob reprezentace znalostí, expertní systémy 2.generace obvykle používají hybridní (kombinované) reprezentace znalostí.
- Hybridní reprezentace, které jsou dostupné v nejčastěji používaných prostředcích pro vývoj ES, 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 strukturálních znalostí a pravidla pro usuzování o těchto znalostech. Rámce také mohou být využity pro implementaci sémantických sítí.

Příklady hybridních systémů

• **ACQUIRE**	• prázdný ES, 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 řídící 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

Pravidlové expertní systémy

Tvary pravidel

- Pravidla (*rules*) mohou mít například takovéto tvary:

  IF předpoklad THEN závěr

  IF situace THEN akce

  IF podmínka THEN závěr AND akce

  IF podmínka THEN důsledek1 ELSE důsledek2

- V předpokladové části (***antecedentu***) se mohou vyskytnout spojky AND a OR, v důsledkové části (***konsekventu***) se může vyskytnout spojka AND. Součástí pravidla může být také tzv. ***kontext***, ve kterém má být pravidlo uvažováno.
- Jiný způsob zápisu pravidla *if E then H*: $E \rightarrow H$
- (E – evidence, H – hypothesis).
- Pravidlo $E \rightarrow H$ neznamená totéž, co implikace $E \Rightarrow H$.

Pravidlové (produkční) systémy

- Většina znalostních systémů je založena na pravidlech, nebo kombinuje pravidla s jiným způsobem reprezentace.
- Pravidlové systémy se od klasických logických systémů odlišují ***nemonotonním uvažováním*** a možností ***zpracování neurčitosti***.
- Neurčitost se může vyskytnout jednak v předpokladech pravidla, jednak se může týkat pravidla jako celku.

Inference v pravidlových systémech

- Inference je založena na pravidle ***modus ponens***:

$$\frac{E,\ E \rightarrow H}{H}$$

- To znamená, že jestliže platí předpoklad E a pravidlo E→H, pak platí závěr H. Modus ponens představuje přímé usuzování. Nepřímé usuzování je dáno pravidlem ***modus tollens***:

$$\frac{\neg H,\ E \rightarrow H}{\neg E}$$

- Řešení problému spočívá v nalezení řady inferencí (***inference chain***), které tvoří cestu od definice problému k jeho řešení

Základní strategie procesu usuzování:

- ***Usuzování řízené daty (dopředné řetězení, forward chaining):***
  - Začíná se všemi známými daty a postupuje k závěru. Je vhodné pro problémy zahrnující syntézu (navrhování, konfigurace, plánování, rozvrhování, …).
- ***Usuzování řízené cíli* (*zpětné řetězení, backward chaining*):**
  - Vybírá možný závěr a pokouší se dokázat jeho platnost hledáním dat, které jej podporují. Je vhodné pro diagnostické problémy, které mají malý počet cílových hypotéz.

Základní struktury v pravidlových systémech:

- ***Inferenční síť* (*inference network*):**
  - Závěry pravidel jsou fakta, která korespondují s předpoklady jiných pravidel. Znalostní bázi můžeme vizualizovat jako síť propojených pravidel a faktů.
- ***Systém porovnávání se vzorem (pattern-matching system):***
  - Závěry pravidel jsou obecnější a můžeme je chápat jako kolekce faktů, které mohou nebo nemusí korespondovat se vzory popsanými v předpokladech jiných pravidel.

Inferenční síť

- Inferenční síť může být reprezentována jako graf, jehož uzly jsou fakta a orientované hrany odpovídají pravidlům.
- Inferenční sítě jsou užitečné pro domény, kde počet možných řešení je limitován, jako jsou např. ***klasifikační nebo diagnostické problémy***. Tyto systémy jsou však méně flexibilní.
- Inferenční sítě se snadněji implementují a snadněji se v nich zajišťuje vysvětlování.

Systém porovnávání se vzorem

- Vztahy mezi fakty a pravidly se ustavují až při běhu na základě úspěšného porovnání faktů se vzory, které se nacházejí v levých částech pravidel. V případě shody všech vzorů v levé části pravidla s fakty v bázi faktů se mohou provést akce v pravé části pravidla (např. to může být zápis faktu do báze faktů nebo zrušení faktu v bázi faktů).
- Systémy založené na porovnání se vzorem se vyznačují vysokou flexibilitou a schopností řešit problémy. Jsou spíše aplikovatelné v doménách, kde počet možných řešení je vysoký nebo neomezený, jako je ***navrhování, plánování a syntéza***.V těchto doménách nejsou předdefinovány vztahy mezi fakty a pravidly.
- V těchto systémech se hůře zajišťuje podpora rozhodování za neurčitosti. V rozsáhlých aplikacích hrozí snížení efektivnosti při vyhledávání aplikovatelných pravidel.

Základní kroky dopředného řetězení

- ***Porovnání* (*matching*):**
  - Pravidla ze znalostní báze jsou porovnávána se známými fakty, aby se zjistilo, u kterých pravidel jsou splněné předpoklady.
- ***Řešení konfliktu* (*conflict resolution*):**
  - Z množiny pravidel se splněnými předpoklady se vybírá pravidlo podle priority a v případě více pravidel se stejnou prioritou podle nějaké strategie .
- ***Provedení* (*execution*):**
  - Provede se pravidlo vybrané v předchozím kroku. Důsledkem provedení pravidla může být přidání nového faktu do báze faktů, odstranění faktu z báze faktů, přidání pravidla do báze znalostí apod.
  - Obvykle je přitom uplatňována podmínka, že pravidlo může být aktivováno pouze jednou se stejnou množinou faktů.

Příklady strategií řešení konfliktu

- ***Strategie hledání do hloubky* (*depth strategy*):** preferována jsou pravidla používající aktuálnější data (data, která se v bázi faktů vyskytují kratší dobu).
- ***Strategie hledání do šířky* (*breath strategy*):** preferována jsou pravidla používající starší data.
- ***Strategie složitosti resp. specifičnosti* (*complexity strategy*):** preferována jsou speciálnější pravidla (pravidla mající více podmínek).
- ***Strategie jednoduchosti* (*simplicity strategy*):** preferována jsou jednodušší pravidla

Vhodné aplikace pro dopředné řetězení

- ***Monitorování a diagnostika řídicích systémů pro řízení procesů v reálném čase***, kde data jsou kontinuálně získávána a měněna a kde existuje málo předem určených vztahů mezi vstupními daty a závěry. V těchto aplikacích se z důvodu potřeby rychlé odezvy používá inferenční síť.
- ***Problémy zahrnující syntézu* (*navrhování, konfigurace, plánování, rozvrhování, …*).** V těchto aplikacích existuje mnoho potenciálních řešení a pravidla proto musejí vyjadřovat znalosti jako obecné vzory. Přesné vztahy (inferenční řetězce) tudíž nemohou být předem určeny a musejí být použity systémy porovnávání se vzorem.

Algoritmus zpětného řetězení

1. Utvoř zásobník a naplň jej všemi koncovými cíli.
2. Shromáždi všechna pravidla schopná splnit cíl na vrcholu zásobníku. Je-li zásobník prázdný, pak konec.
3. Zkoumej postupně všechna pravidla z předchozího kroku.
   a) Jsou-li všechny předpoklady splněny, pak odvoď závěr (proveď pravidlo). Jestliže zkoumaný cíl byl koncový, pak jej odstraň ze zásobníku a vrať se na krok 2. Jestliže to byl podcíl (dílčí cíl), odstraň jej ze zásobníku a vrať se ke zpracování předchozího pravidla, které bylo dočasně odloženo.
   b) Jestliže fakty nalezené v bázi faktů nesplňují předpoklady pravidla, je zkoumání pravidla ukončeno.

c) Jestliže pro některý parametr předpokladu chybí hodnota v bázi faktů, zjišťuje se, zda existuje pravidlo, z něhož by mohla být tato hodnota odvozena. Pokud ano, parametr se vloží do zásobníku jako podcíl, zkoumané pravidlo se dočasně odloží a přejde se na krok 2. V opačném případě se tato hodnota zjistí od uživatele a pokračuje se v kroku 3.a) zkoumáním dalšího předpokladu.

4. Jestliže pomocí žádného ze zkoumaných pravidel nebylo možné odvodit hodnotu důsledku, pak daný cíl zůstává neurčen. Odstraní se ze zásobníku a pokračuje se krokem 2.

Vhodné aplikace pro zpětné řetězení

- Zpětné řetězení je vhodnější pro aplikace, mající mnohem více vstupů než možných závěrů.
- Dobrou aplikací pro zpětné řetězení je ***diagnostika***, kde člověk komunikuje se znalostním systémem a zadává data pomocí klávesnice. Většina diagnostických systémů byla implementována pomocí inferenční sítě, protože vztahy mezi fakty jsou obvykle dobře známy.
- Ideální pro zpětné řetězení jsou rovněž ***klasifikační problémy***. Tento typ aplikace může být implementován buď pomocí inferenční sítě nebo pomocí vzorů v závislosti na složitosti dat.

Algoritmus Rete

- Dopředný systém porovnávání se vzorem je velmi neefektivní. V každém cyklu se musejí opakovaně porovnávat všechna pravidla se všemi fakty v bázi faktů. Podle odhadu až 90% času práce produkčního systému je věnováno opakovanému porovnávání se vzorem. Přitom po provedení pravidla většina báze faktů a tudíž také jejich účinků na pravidla zůstává nezměněna.
- ***Rete*** je účinný porovnávací algoritmus redukující dobu porovnávání na základě síťové struktury, ve které jsou uloženy informace o ztotožnění podmínek s fakty v bázi faktů.

Síťová struktura pro Rete

- ***Uzly sítě*:** Startovací uzel a dále jeden uzel pro každou podmínku a konjunkci podmínek. Konjunktivní uzly s výstupním stupněm 0 korespondují s pravidly. S každým uzlem je spojena množina faktů, se kterými je podmínka ztotožněna.
- ***Hrany sítě*:** jsou označeny vazbami nebo vztahy proměnných vyskytujících se v počátečním uzlu hrany.
- Fakt, který je přidáván do báze faktů, je reprezentován příznakem. Tento příznak je zpočátku umístěn ve startovacím uzlu, a odtud se pak šíří po síti. Příznak může projít hranou, jestliže jeho argumenty splňují vztah spojený s hranou.
- Jestliže v uzlu, odpovídajícím pravidlu, všechny příznaky splňují podmínky pravidla, pak se pravidlo přidá do konfliktní množiny.
- Pokud je nějaký fakt z báze faktů odstraněn, pak se jeho příznak podobně šíří po síti, přičemž se mažou všechny jeho kopie vyskytující se v uzlech a z konfliktní množiny jsou pak také odstraněna odpovídající pravidla.

Příklady pravidlových systémů

• **ART**	• prázdný ES založený na Lispu, dopředné řetězení, algoritmus Rete
• **CLIPS**	• programové prostředí, dopředné řetězení, algoritmus Rete
• **EXSYS**	• prázdný expertní systém, dopředné a zpětné řetězení
• **M.4**	• programové prostředí, dopředné a zpětné řetězení, porovnávání se vzorem
• **ILOG-RULES**	• programové prostředí, dopředné řetězení, algoritmus Xrete
• **OPS5**	• programové prostředí, dopředné řetězení, algoritmus Rete

Výhody a nevýhody pravidlových systémů

- ***Výhody***:
  - modularita,
  - uniformita,
  - přirozenost.
- ***Možné nevýhody a problémy***:
  - nebezpečí nekonečného řetězení,
  - přidání nové rozporné znalosti,
  - modifikace existujících pravidel,
  - neefektivnost,
  - neprůhlednost,
  - pokrytí domény (existují domény vyžadující příliš mnoho pravidel).