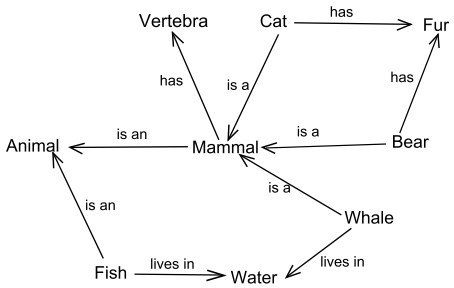
Sieci semantyczne

Sieci semantyczne są grafowym schematem reprezentacji wiedzy:



Fish are animals.
Mammals are animals.
Mammals have vertebra.
Whales are mammals.
Cats are mammals.
Bears are mammals.
Fish live in the water.
Whales live in the water.
Cats have fur.
Bears have fur.

Sieć zawiera węzły odpowiadające pojęciom danej dziedziny problemowej, i łuki odpowiadające związkom (relacjom) zachodzącym pomiędzy tymi pojęciami:

- sieć jest czytelna — ludzie często wyrażają informacje graficznie,
- sieć jest elastyczna — możemy wprowadzać informacje w dowolnej formie.

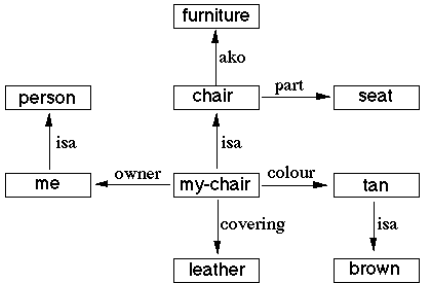
Powyższa sieć (źródło: Wikipedia) skonstruowana jest bez rozróżniania klas od indywiduów, konsekwencji w nazwach relacji (is a/is an), itp.

Sieci semantyczne: relacje ISA i AKO

Aby prawidłowo odróżnić w sieciach semantycznych klasy obiektów od indywiduów, oraz wyrazić różne zależności między nimi, stosuje się pewne standardowe relacje:

- ISA (ang. *is a*) jest relacją pomiędzy indywiduum a jego klasą
- HASA (ang. *has a*) jest relacją część-całość, alternatywnie: PART
- AKO (ang. *a kind of*) jest relacją pomiędzy podklasą a nadklasą, zapisywane często również jako: SUBCLASS, albo SS (subset)

Przykład: *I own a tan leather chair.*



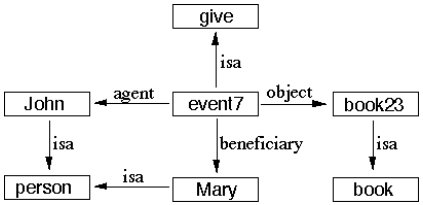
Sieci semantyczne: relacje binarne i reifikacja

Można traktować informacje zawarte w sieci semantycznej jako zbiór (koniunkcję) formuł logicznych. Formuły wyrażają bezpośrednio zachodzenie relacji pomiędzy obiektami (termami). Zauważmy, że w powyższych przykładach wszystkie relacje (i odpowiadające im formuły) są relacjami binarnymi (dwuargumentowymi). To jest podstawowa cecha sieci semantycznych.

Jak można wyrazić relację złożoną za pomocą zestawu relacji binarnych?

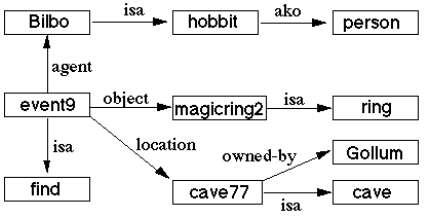
W niektórych przypadkach złożona relacja naturalnie dekomponuje się na składowe binarne. Jednak nie zawsze tak się uda. W pozostałych przypadkach stosuje się zabieg **reifikacji**, czyli przekształcenia relacji w obiekty.

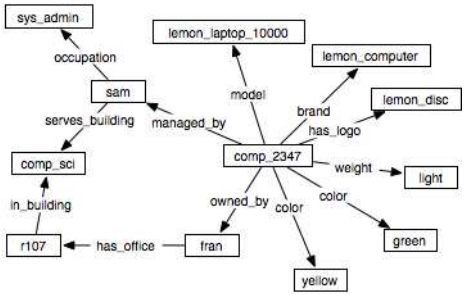
Przykład: *John gives the book to Mary.*



Sieci semantyczne: przykłady

Bilbo finds the magic ring in Gollum's cave.



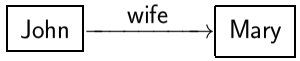


Odczytaj informacje z tej sieci!
Spróbuj przekształcić obiekty reifikowane na relacje złożone.

Sieci semantyczne: odpowiadanie na pytania

Rozważmy przykład zdania i odpowiadającej mu sieci semantycznej:

Mary is John's wife.



Sieć wyraża w pewnym języku formalnym informacje wcześniej zawarte w oryginalnej wypowiedzi języka naturalnego. Od systemu sztucznej inteligencji oczekiwalibyśmy, że posiadając pewną wiedzę, będzie w stanie odpowiadać na dotyczące jej pytania.

Inaczej mówiąc, jak zaimplementować wnioskowanie dla sieci semantycznych?

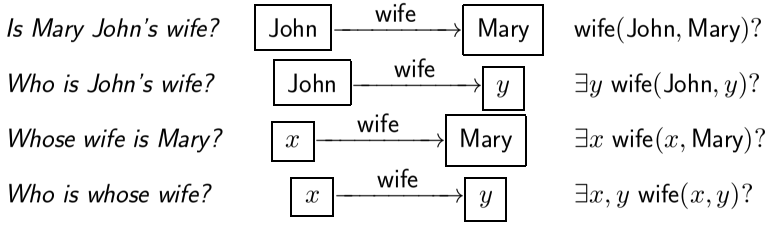
Na przykład:

Is Mary John's wife?
Who is John's wife?
Whose wife is Mary?
Who is whose wife?

Sieci semantyczne: dopasowanie

Wnioskowanie w sieciach semantycznych może być zaimplementowane przez:

1. wyrażenie pytania w postaci oddzielnej, zapytaniowej, sieci semantycznej,
2. próbę dopasowania sieci zapytaniowej do sieci faktowej,
3. w przypadku braku dopasowania, odpowiedź jest negatywna,
4. w przypadku uzyskania dopasowania, odpowiedź jest pozytywna.



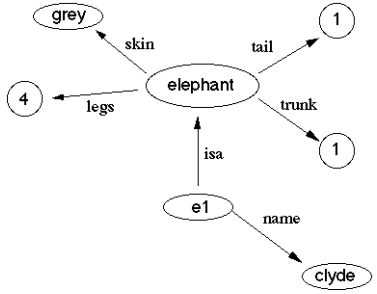
Formułowanie pytań w sieciach semantycznych wymaga użycia zmiennych domyślnie kwantyfikowanych kwantyfikatorem egzystencjalnym, podobnie jak w logice predykatów. Jeśli chcemy uzyskać wartość odpowiedzi w ostatnich trzech pytaniach, to musimy otrzymać od mechanizmu wnioskowania obiekty dopasowane do obiektów-zmiennych.

Sieci semantyczne: dziedziczenie

Rozważmy inny przykład:

Elephants have four legs.
Elephants have one trunk.
Elephants have one tail.
Elephants have grey skin.
This elephant's name is clyde.

Jakiego koloru skórę ma clyde?



W tej sieci mamy wiedzę ogólną o słoniach połączoną z wiedzą o pewnym słoniowym indywiduum o imieniu clyde. Tworzenie sieci zapytaniowych i dopasowanie ich do sieci faktowej daje odpowiedzi na pojedyncze fakty. Można rozszerzyć ten mechanizm przez wykorzystanie semantyki relacji *isa* i *ako*, ponieważ indywidua danej klasy normalnie posiadają własności wyrażone dla klasy, jak również wszystkich klas nadrzędnych.

Takie rozszerzenie nazywamy **dziedziczeniem**, i w powyższym przypadku pozwala ono np. uzyskać odpowiedź, że clyde ma skórę koloru szarego.

Sieci semantyczne: wiedza domyślna

Wiedza ogólna o klasach jest przykładem wiedzy **domyślnej** (*default*). Umożliwia ona wnioskowanie **niemonotoniczne**, specjalnie implementowane w niektórych systemach logicznych. W sieciach semantycznych pojawia się ono naturalnie dzięki dziedziczeniu.

Gdyby w przykładzie o słoniach zdanie: *This elephant's name is clyde.* zastąpić zdaniem: *This pink elephant's name is clyde.*
to do sieci przybyłaby dodatkowa krawędź: 

.

Wtedy odpowiedź na pytanie: 

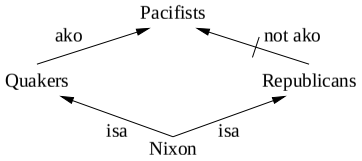
mogłaby być uzyskana bez dziedziczenia przez dopasowanie $z =$ pink. Taka odpowiedź normalnie ma priorytet, tzn. wyklucza uzyskanie odpowiedzi na to samo pytanie przez dziedziczenie.

W ogólności wiedza domyślna podlega normalnemu procesowi dziedziczenia. Gdyby klasa Człowiek miała własność typowy-wzrost (średni), to dla jakiegoś anonimowego człowieka mógłby on wynosić np. 170cm, ale dla podklasy Mężczyzna raczej 180cm, a dla podklasy Mężczyzna-koszykarz pewnie 190cm.

Sieci semantyczne: dziedziczenie wielokrotne

Można byłoby zadać sobie pytanie, czy indywiduum w sieci semantycznej może należeć do więcej niż jednej klasy przez relację *isa* (lub łańcuch *isa-ako**). Gdyby tak było, to mechanizm wnioskowania z dziedziczeniem mógłby teoretycznie uzyskiwać różne odpowiedzi przez różne ścieżki dziedziczenia.

Popularnym przykładem w wielu podręcznikach jest zagadnienie czy Nixon[1] był pacyfistą (∼1980). Wiadomo o nim, że był kwakrem[2] i jednocześnie republikaninem.[3]



Ze względu na problemy wielokrotnego dziedziczenia, w niektórych systemach obiektowych jest ono wykluczone.

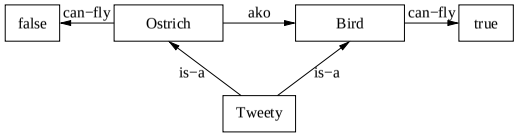
[1] Richard M. Nixon — prezydent U.S.A. w latach 1969–1974. Po wygraniu wyborów na drugą kadencję w 1973r. był zamieszany w następstwa afery Watergate i zrezygnował z urzędu prezydenta pod groźbą usunięcia.

[2] Quakers jest nazwą grupy stowarzyszeń religijnych wywodzących się z XVII-wiecznej Anglii i działających na całym świecie, m.in. w Stanach Zjednoczonych. Głosili m.in. skromność ubioru i odmowę udziału w wojnach.

[3] Partia Republikańska w Stanach Zjednoczonych jest symbolem konserwatyzmu, poglądów wolnorynkowych, prywatnej własności, ograniczenia roli związków zawodowych i interwencjonizmu państwowego, za to silnej armii.

Sieci semantyczne: wnioskowanie z dziedziczeniem

Chcielibyśmy, aby algorytm wnioskowania z dziedziczeniem sam rozwiązywał istniejące kolizje, o ile to tylko możliwe. Rozważmy przykład:



Tweety jest strusiem, i jednocześnie ptakiem. Pytanie: czy potrafi fruwać? Zdolność fruwania jest cechą ptaków, ale nie strusi. Ponieważ fakt, że tweety jest strusiem jest bardziej szczegółowy, więc wydaje się, że kwestię fruwania powinna rozstrzygać domyślna wiedza o strusiach.

W ogólności definiuje się **odległość inferencyjną** klas w taksonomii. Klasa C jest dalej niż klasa B od klasy A, jeśli ścieżka dziedziczenia z A do C biegnie przez B. Algorytm wnioskowania rozstrzyga wielokrotne dziedziczenie na korzyść klasy bliższej. Ponieważ jednak taka odległość wprowadza tylko porządek częściowy, rozwiązuje ona problem tweety, ale nie rozwiązuje problemu Nixona.

Sieci semantyczne: formalizacja

Jednym z problemów sieci semantycznych jest brak standardowego katalogu relacji (linków). Można wprowadzać dowolne relacje i stosować dowolne nazwy. Utrudnia to zrozumienie nieznanej sieci (przez komputer), sprawdzenie jej poprawności, itp. Aby rozwiązać ten problem wprowadzono pewne standardy.

typ linku	znaczenie	przykład
$A \xrightarrow{isa} B$	$A \in B$	tweety $\subseteq$ Ostrich
$A \xrightarrow{ako} B$	$A \subseteq B$	Ostrich $\subseteq$ Bird
$A \xrightarrow{R} B$	$R(A, B)$	tweety $\xrightarrow{\text{can-fly}}$ false
$A \xrightarrow{\boxed{R}} B$	$\forall x\ x \in A \Rightarrow R(x, B)$	Bird $\xrightarrow{\#\text{legs}}$ 2
$A \xrightarrow{\boxed{\boxed{R}}} B$	$\forall x \exists y\ x \in A \Rightarrow y \in B \wedge R(x, y)$	Bird $\xrightarrow{\text{parent}}$ Bird

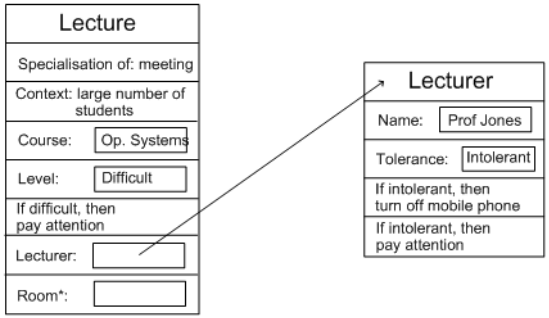
Jednak nadmierna formalizacja sieci semantycznych, i wprowadzanie kolejnych mechanizmów dla węzłów i łuków sieci niweluje zasadniczą zaletę, jaką jest czytelność reprezentacji graficznej.

Systemy ramek

System ramek (*frame system*) składa się z kolekcji ramek opisujących elementy modelowanej dziedziny. Ramka zawiera zbiór **atrybutów** (ang. *slots*) reprezentujących jej właściwości. Atrybuty ramki w polskiej literaturze bywają nazywane **klatkami** (K.Goczyła) lub **szufladkami** (W.Duch).

Ramki mogą reprezentować pojęcia z dziedziny — mają wtedy charakter klasy — jak również indywidualne obiekty. Ramka może posiadać dwa rodzaje atrybutów: własne (*own*) albo szablonowe (*template*). Atrybuty własne należą do danej ramki, a ich wartości są prywatne dla ramki. Atrybuty szablonowe danej klasy stają się atrybutami własnymi wszystkich jej instancji. Ramki mogą dziedziczyć od siebie zarówno atrybuty własne, jak i szablonowe. Ramka, która nie posiada atrybutów szablonowych, jest obiektem.

Atrybut może posiadać wartość, która jest wartością dosłowną (*literal*), odnośnikiem-relacją do innej ramki, oraz pewne **cechy** (*facets*). Te cechy mogą określać wartość domyślną, więzy takie jak: liczbę wartości (atrybut jedno- lub wielowartościowy, min i max wartości), typ i zakres wartości, lub zbiór dopuszczalnych wartości, dołączone procedury (np. if-needed, if-added, if-removed), atrybuty odwrotne, itp.



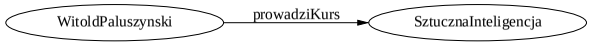
Systemy ramek są prekursorem systemów obiektowych, jednak są między nimi istotne różnice. Na przykład, system programowania obiektowego definiuje hierarchię klas z metodami, która pozwala na tworzenie obiektów i dziedziczenie przez nie zarówno struktury obiektu, jak i metod. W systemie ramek nie ma zasadniczej różnicy pomiędzy klasami a obiektami, więc cała taksonomia jest dostępna dla programu w czasie wykonania.

Język RDF

Podstawowy element składowy: trójka **obiekt-atrybut-wartość**: [4]

- Nazywa się to **stwierdzeniem** (*statement*).
- Przykład stwierdzenia:

  Witold Paluszyński prowadzi kurs Sztuczna Inteligencja.
- Graf RDF reprezentujący powyższe stwierdzenia:



Podstawowe pojęcia RDF:

- zasoby (*resources*),
- właściwości (*properties*),
- stwierdzenia (*statements*).

[4] Uwaga: często stosowana jest alternatywna (miejscami myląca) terminologia: podmiot-predykat-przedmiot (*subject-predicate-object*), a w polskiej literaturze również: podmiot-orzeczenie-dopełnienie [K.Goczyła]. Ponieważ rzadko powoduje to nieporozumienia, trzeba pogodzić się z praktyką mieszania tej terminologii, i nie przywiązywać zbyt wielkiej wagi do użytego w danym kontekście słowa.

Zasoby: URL, URI, IRI

- Można myśleć o zasobach jako obiektach, o których chcemy mówić:
  - np.: ludzie, miejsca, miasta, naukowcy, studenci, uczelnie, itp.
- Każdy zasób ma URI (*Universal Resource Identifier*).
- URI może być:
  - adresem URL (internetowym), lub
  - jakimś innym unikalnym identyfikatorem.
- W tych rozważaniach będziemy przyjmowali adresy URL jako URI.
  IRI są zinternacjonalizowaną wersją URI.
- Zalety korzystania z URI:
  - globalny, uniwersalny w skali świata, unikalny schemat nazewnictwa,
  - częściowo rozwiązuje problem homonimii (wieloznaczności identycznych nazw) rozproszonych reprezentacji danych.

Właściwości

- Właściwości opisują binarne relacje między innymi zasobami:
  - np.: „prowadzi kurs", „kieruje", „tytuł", itd.
- Właściwości są obywatelami pierwszej klasy, tzn. są również traktowane jako zasoby, mogą mieć różne charakterystyki, i tworzą własną taksonomię.
- Właściwości jako zasoby są również identyfikowane przez URI.

Stwierdzenia

- Stwierdzenia stwierdzają posiadanie właściwości przez zasoby, a dokładniej: związek pewnej pary zasobów pewną relacją (binarną).
- Stwierdzenie jest trójką: obiekt-atrybut-wartość
  - Składa się z zasobu, właściwości i wartości.
- Wartościami mogą być zasoby lub literały.
  - Literały są wartościami atomowymi (typu string).

Trzy reprezentacje stwierdzeń

Stwierdzenie możemy reprezentować jako:

- trójkę obiekt-atrybut-wartość,
- elementarny graf z dwoma węzłami połączonymi łukiem skierowanym,
- zapis tekstowy, zwany **serializacją**.

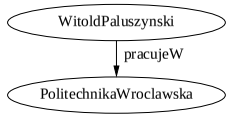
Zatem zbiór stwierdzeń, wyrażający pewien zasób wiedzy może być postrzegany jako:

- zbiór trójek obiekt-atrybut-wartość,
- graf zwany **siecią semantyczną**,
- dokument (np. plik) zawierający serializację zbioru trójek.

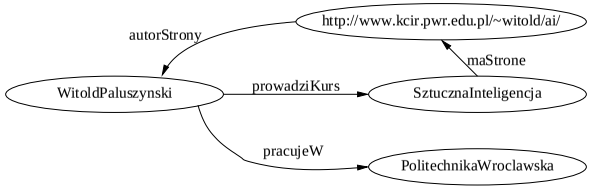
Stwierdzenia jako trójki

- Trójkę (x, P, y) można uważać za formułę logiczną $P(x, y)$, gdzie binary predykat P wiąże obiekt x z obiektem y.
- Trójkę można również uważać za skierowany graf z etykietowanymi węzłami i łukami:
  - skierowany **od** zasobu podmiotu (obiektu) stwierdzenia,
  - skierowany **do** przedmiotu (wartości) stwierdzenia,
  - wartość stwierdzenia może być innym zasobem lub literałem.
- W RDF zarówno zasoby jak i właściwości muszą być identyfikowane przez URI. Możliwe jest jednak stosowanie przestrzeni nazw, skracających zapis.

```
(foaf:Person#WitoldPaluszynski,
 dbpedia-owl:employer,
 http://www.pwr.edu.pl/)
```



Zbiór trójek jako sieć semantyczna



Sieci semantyczne są elastycznym i ekspresyjnym narzędziem reprezentacji wiedzy. Ich grafowa wersja jest bardzo zrozumiała, ale przetwarzanie reprezentacji graficznych przez komputery nie jest efektywne.

Istnieją reprezentacje tekstowe sieci semantycznych. Jednak z nich jest oparta na XML, zwana RDF/XML. Jednak nie jest ona częścią modelu danych RDF.

Zapis stwierdzeń w RDF/XML

```
<rdf:RDF
  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:myonto="http://www.kcir.pwr.edu.pl/~witold/myonto-ns">

  <rdf:Description rdf:about="http://www.kcir.pwr.edu.pl/~witold/">
    <myonto:author rdf:resource="#Witold Paluszynski"/>
  </rdf:Description>
</rdf:RDF>
```

- Dokument RDF jest reprezentowany przez element XML ze znacznikiem rdf:RDF
- Zawartością tego elementu jest pewna liczba opisów (*descriptions*), które wykorzystują znaczniki rdf:Description
- W powyższym opisie, dotyczącym zasobu `http://www.kcir.pwr.edu.pl/~witold/`:
  - właściwość jest używana jako znacznik elementu,
  - wartość własności może być dana przez zawartość elementu (literał), lub jak w tym przypadku, wskazywana przez atrybut rdf:resource.

Ogólnie, w serializacji RDF/XML każdy opis wyraża fakt o zasobie, identyfikowanym na jeden z 3 sposobów:

- przez atrybut rdf:about, z odniesieniem do istniejącego zasobu,
- przez atrybut rdf:ID, z utworzeniem nowego zasobu,
- bez nazwy, tworząc nowy zasób (anonimowy).

Inna serializacja RDF: N-Triples

Model danych RDF jest najlepiej reprezentowany grafami. Jednak przydatna i często niezbędna jest ich reprezentacja tekstowa, zwana serializacją. Dotychczas, oprócz formatu zapisu RDF/XML, stosowana była nieformalnie notacja: (R,P,V). Istnieją jednak bardziej sformalizowane konwencje, ukierunkowane zarówno na czytelność jak i przetwarzanie maszynowe.

Jeden z takich formatów, zwany N-Triples, polega na zapisie trzech elementów trójki RDF w kolejności podmiot-predykat-przedmiot, zakończonej kropką, po jednej trójce w wierszu. Każdy z elementów trójki zapisywany jest w postaci w pełni kwalifikowanych, nieskróconych URI, zapisywanych w nawiasach kątowych <>, według schematu:

```
<http://domain/ns#res> <http://domain/ns#prop> <http://domain/ns#val> .
```

Nawet powyższy schemat trudno zapisać w wymagany sposób, w jednym wierszu. Jak widać, ten format średnio nadaje się do prezentacji tekich jak niniejsza. Natomiast bardzo dobrze nadaje się dla przeszukiwania i porównywania tekstowego.

N-Triples: przykład

Dla trójki reprezentowanej przez poniższy zapis RDF/XML:

```
<rdf:RDF
  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:myonto="http://www.kcir.pwr.edu.pl/~witold/myonto-ns">

  <rdf:Description rdf:about="http://www.kcir.pwr.edu.pl/~witold/">
    <myonto:author rdf:resource="#Witold Paluszynski"/>
  </rdf:Description>
</rdf:RDF>
```

reprezentacja N-Triples ma postać (w jednym wierszu):

```
<http://www.kcir.pwr.edu.pl/~witold/>
<http://www.kcir.pwr.edu.pl/~witold/myonto-ns#autor>
"#Witold Paluszynski"
.
```

Serializacja RDF — Turtle

Innym formatem zapisu tekstowego RDF jest Turtle (*Terse RDF Triple Language*). Podstawowa gramatyka Turtle jest podobna do N-Triples (w rzeczywistości oba te formaty są podzbiorami ogólnej notacji N3 (*Notation3*)), ale bardziej zorientowana na skróty, czytelność, i wygodę.

W notacji Turtle zasoby mogą być zapisywane w postaci *qnames*, czyli `ns:id`, gdzie ns jest symbolem przestrzeni nazw, a id identyfikatorem zasobu. Przestrzenie nazw związane są w Turtle z definiującymi je URI za pomocą deklaracji `@prefix`.

```
@prefix myonto <http://www.kcir.pwr.edu.pl/~witold/myonto-ns#>

<http://www.kcir.pwr.edu.pl/~witold/"> myonto:author "#Witold Paluszynski" .
```

przykłady notacji Turtle dla kontynuacji ;.

Typy danych

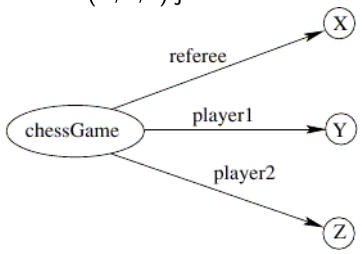
- Typy danych stosowane są w językach programowania, aby umożliwić interpretację.
- W RDF w tym celu stosowane są literały typowane:

```
(#Witold Paluszynski,
  http://www.kcir.pwr.edu.pl/~witold/myonto/roomNumber,
  "307"^^http://www.w3.org/2001/XMLSchema#integer)
```

- Zapis ^^ wskazuje typ literału
- w dokumentach RDF dozwolone jest korzystanie z wszelkich zewnętrznych typów danych.
- W praktyce najczęściej wykorzystywany jest system typów XML Schema, który definiuje szeroki wachlarz typów danych. Na przykład: Boolean, liczby całkowite, zmiennoprzecinkowe, czas, daty, itp.

Krytyczne spojrzenie na RDF: predykaty binarne

- RDF używa tylko binarnych właściwości.
  - Jest to ograniczenie, ponieważ często używamy predykatów z więcej niż 2 argumentami.
  - Ale można je zasymulować predykatami binarnymi.
- Przykład: referee(X,Y,Z)
  X jest sędzią meczu szachowego pomiędzy graczami Y i Z.
  - Wprowadzamy nowy pomocniczy zasób chessGame oraz predykaty binarne: ref, player1 i player2
  - Możemy teraz wyrazić referee(X,Y,Z) jako:



Krytyczne spojrzenie na RDF: właściwości

- Właściwości są specjalnym rodzajem zasobów.
- Właściwości mogą występować jako obiekty w trójkach obiekt-atrybut-wartość (stwierdzeniach).
- Możliwość ta oferuje dużą elastyczność.
- Ale to jest niezwykłe dla języków modelowania i języków programowania OO.
- Może to być mylące dla programistów modelowania semantycznego.

Krytyczne spojrzenie na RDF: reifikacja

- Reifikacja jest innym dość mocnym mechanizmem.
- Może wydawać się nie na miejscu we w sumie prostym języku takim jak RDF.
- Tworzenie stwierdzeń o stwierdzeniach wprowadza poziom złożoności, który nie jest niezbędny do podstawowej warstwy Semantic Web.
- Mogłoby wydawać się bardziej naturalne umieszczenie tego mechanizmu w bardziej zaawansowanych warstwach, które zapewniają bogatsze funkcje reprezentacji.

Krytyczne spojrzenie na RDF: podsumowanie

- RDF jest dostosowany do przetwarzania maszynowego, jednak do czytania przez ludzi może być niezbyt zrozumiały.
- RDF ma swoje dziwactwa i ogólnie nie jest optymalnym językiem modelowania, ale:
  - jest już de facto standardem,
  - ma wystarczającą siłę wyrazu (przynajmniej dla budowania na nim dalszych warstw reprezentacji),
  - informacja jest jednoznacznie mapowana do modelu.

RDF Schema

- RDF jest uniwersalnym językiem, który pozwala użytkownikom opisywać zasoby przy pomocy własnych zestawów pojęć.
- RDF nie przyjmuje, ani nie definiuje semantyki konkretnej dziedziny.
- Użytkownik może to zrobić w RDF Schema przy użyciu:
  - klas i właściwości,
  - hierarchii klas i dziedziczenia,
  - hierarchii właściwości

Jednak nie będziemy tu zgłębiać języka RDF Schema. Semantykę dziedzin będziemy opisywali w inny sposób.

Język zapytań SPARQL

SPARQL (*Simple Protocol And RDF Query Language*) jest językiem zapytań RDF. Składniowo SPARQL przypomina nieco SQL, lecz w rzeczywistości język SPARQL nawiązuje do grafowego modelu danych RDF:

- SPARQL opiera się na dopasowaniu do wzorców-grafów.
- Najprostszym wzorcem-grafem jest trójka, podobna do trójki RDF ale z możliwością użycia zmiennej zamiast termu RDF na pozycji podmiotu, predykatu lub przedmiotu.
- Łączenie wzorców-trójek daje wzorzec-graf. Dokładne dopasowanie wzorca do grafu danych RDF jest niezbędne dla dopasowania wzorca.

Przykładowe zapytanie SPARQL

Przykład:

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
SELECT ?c
WHERE
{
    ?c rdf:type rdfs:Class .
}
```

Zapytanie pobiera wszystkie trójki, gdzie właściwością jest rdf:type a podmiotem jest rdfs:Class. Co oznacza, że pobiera wszystkie klasy.

Przykładowe zapytanie SPARQL (2)

Pobierz wszystkie instancje danej klasy, np. kurs (deklaracja prefiksów rdf, rdfs pominięte dla zwięzłości):

```
PREFIX wp-np: <http://www.kcir.pwr.edu.pl/~witold/ontologies/2015/2/NaukaPolska
SELECT ?i
WHERE
{
    ?i rdf:type wp-np:Kurs .
}
```

Należy nadmienić, że SPARQL nie wymaga, ani sam nie realizuje semantyki RDFS. Zatem, czy w odpowiedzi na powyższe zapytanie otrzymamy tylko instancje klasy `wp-np:Kurs`, czy również jej podklas, będzie zależeć od systemu realizującego dopasowanie wzorca i odpowiedź.

Struktura zapytania SELECT-FROM-WHERE

Podobnie jak w SQL, zapytania SPARQL mają strukturę SELECT-FROM-WHERE:

- SELECT określa projekcję: liczbę i kolejność pobieranych danych,
- FROM służy do określenia źródło przeszukiwania (opcjonalne),
- WHERE nakłada ograniczenia na możliwe rozwiązania w postaci szablonów, wzorców wykresów i ograniczeń logicznych.

Przykład: pobrać wszystkie numery pokoi pracowników:

```
SELECT ?x ?y
WHERE
{
    ?x wp-np:nr-pokoju ?y .
}
```

`?x` i `?y` są tu zmiennymi, a wzorzec `"?x wp-np:nr-pokoju ?y"` reprezentuje trójkę zasób-właściwość-wartość.

Domyślny *join*

Przykład: pobierz wszystkich wykładowców i ich numery pokoi:

```
SELECT ?x ?y
WHERE
{
    ?x rdf:type wp-np:prowadzacy ;
       wp-np:nr-pokoju ?y .
}
```

Powyższe zapytanie reprezentuje tzw. domyślny *join*: drugi wzorzec jest ograniczony tylko do tych trójek, których zasób jest w zmiennej `?x`.

Zwróćmy uwagę: używamy tutaj skróconej składni: średnik wskazuje że następująca trójka współdzieli podmiot z poprzednikiem. Ta składnia nazywa się *turtle*.

Poprzednie zapytanie jest równoważne następującej formie:

```
SELECT ?x ?y
WHERE
{
    ?x rdf:type wp-np:prowadzacy .
    ?x wp-np:nr-pokoju ?y .
}
```


Jawny *join*

Kolejny przykład: chcemy znaleźć nazwy wszystkich kursów prowadzonych przez wykładowcę z ID 411

```
SELECT ?n
WHERE
{
    ?x rdf:type wp-np:Kurs ;
       wp-np:prowadzacy :411 .
    ?c wp-np:nazwisko ?n .
    FILTER (?c = ?x) .
}
```

Taka forma zapytań reprezentuje tzw. jawny *join*.

Co to jest ontologia

Pojęcie ontologii pochodzi z filozofii (starożytnej) i ma wiele znaczeń. Słowo ontologia pochodzi od greckich słów: „*on*" (w dopełniaczu „*ontos*") oznaczającego ogólnie byt, i „*logos*" czyli nauki lub wiedzy.

Jedna z najczęściej cytowanych definicji ontologii w sensie reprezentacji wiedzy w sztucznej inteligencji (1992, Gruber):

> **Ontologia** jest jawną specyfikacją konceptualizacji.

Ta definicja może na pierwszy rzut oka przygnieść, lecz spróbujemy się z nią zaprzyjaźnić, a może nawet polubić.

Ontologia służy do tworzenia jawnych, i zrozumiałych dla wszystkich, opisów dowolnych dziedzin. Musi ona zawierać specyfikację:

- pojęć dotyczących danej dziedziny,
- atrybutów tych pojęć, ich własności, i związków między nimi,
- istniejących więzów na te atrybuty, własności, i związki,
- indywiduów istniejących w dziedzinie.

Wyszczególnienie, i nazwanie wszystkich tych elementów danej dziedziny nazywa się jej **konceptualizacją**.

Ontologia powinna określać dla danej dziedziny:

- terminologię uzgodnioną dla danej dziedziny,
- jednoznaczną zrozumiałość pojęć danej dziedziny.

Dlatego właśnie w największym skrócie ontologię danej dziedziny nazywa się jawną specyfikacją jej konceptualizacji.

Po co tworzyć ontologie

Taką rolę opisu znaczenia wszystkich pojęć pełniły jednak dotychczas słowniki (dla pojęć ogólnych z danego języka) i encyklopedie (dla pojęć szczególnych, indywiduów, nazw własnych, itp.).

Dlaczego chcemy tworzyć ontologie?

Wymienione opisy są tworzone w języku naturalnym, nie są całkowicie precyzyjne, natomiast odwołują się do często subtelnych znaczeń konstrukcji językowych, wiedzy ogólnej, a także ogólnie przyjmowanych założeń o wiedzy podstawowej (kulturze) jej czytelnika. Jeśli celem jest umożliwienie agentom sztucznie inteligentnym korzystanie z takowych opisów, to agent musiałby praktycznie mieć własny mechanizm myślenia (umysł) identyczny z mechanizmem myślenia człowieka, aby je tak samo rozumieć.

Aby umożliwić agentom sztucznie inteligentnym różnych poziomów inteligencji ich właściwe zrozumienie, opisy takie musimy stworzyć w jakimś formalizmie dostępnym dla takich agentów.

Po co tworzyć ontologie (cd.)

Inne ważne powody, dla których warto tworzyć ontologie (czyli konceptualizacje formalne), są:

- uzyskiwana dzięki nim jednoznaczność pojęć, standaryzacja,
- tworzenie jawnych zapisów pewnych założeń, które dotąd były domyślne, niejawne, i często niejasne,
- rozdzielenie wiedzy podstawowej o dziedzinie od wiedzy operacyjnej.

Tworzenie ontologii nie jest celem samym w sobie. Jest ono podobne do definiowania standardowej struktury danych do wykorzystania przez programy. Ontologie tworzone są dla zapewnienia możliwości budowy agentów software-owych umożliwiających analizę danych w różnych dziedzinach, wspomaganie podejmowania decyzji, itp.