Reprezentacja wiedzy

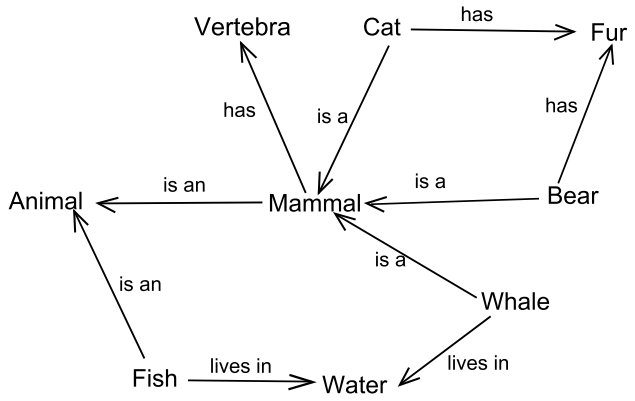
Format i sposób reprezentacji wiedzy o zagadnieniu są niezwykle istotne i mają bezpośredni wpływ na efektywność — lub w ogóle zdolność — znalezienia rozwiązania. Atrakcyjnym i ważnym historycznie schematem reprezentacji wiedzy w sztucznej inteligencji jest logika matematyczna. Nie tylko dostarcza ona rodzinę precyzyjnych języków z dobrze rozwiniętą składnią, zrozumiałą semantyką, ale również posiada aparat dowodzenia twierdzeń pozwalający prowadzić **wnioskowanie**, czyli wywodzenie nowych faktów i odpowiadanie na pytania.

W latach 60-tych XX-ego wieku panował wśród badaczy sztucznej inteligencji entuzjazm dotyczący możliwości wykorzystania logiki. Niestety, naturalny kompromis pomiędzy **ekspresyjnością** języka (*expressiveness*) a złożonością jego procedury dowodowej powodują, że w praktyce możliwości wykorzystania logiki do rozwiązywania problemów okazały się ograniczone. Popularny język logiki jakim jest rachunek predykatów pierwszego rzędu nie pozwala na wyrażanie wielu faktów o świecie rzeczywistym, a zarazem jego procedura dowodowa, o eksponencjalnej złożoności, nie pozwala rozwiązywać problemów typowych dla świata rzeczywistego.

Język logiki matematycznej pozostaje punktem wyjścia konstrukcji wielu schematów reprezentacji wiedzy, jednak stale poszukiwane są schematy bardziej ekspresyjne, ale pozwalające na implementację prostszych mechanizmów wnioskowania.

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.

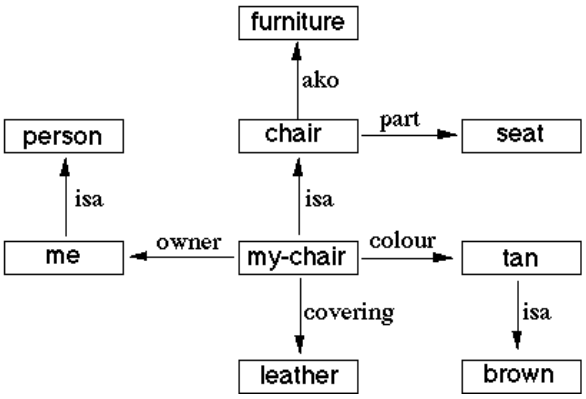
Powyższa sieć (źródło: Wikipedia) skonstruowana jest bez rozróżniania klas od indywiduów, konsekwencji w nazwach relacji (is a/is an), itp.

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.*



W praktyce sieci semantycznych te powyższe relacje są niekiedy używane w odmienny sposób, np. relacja ISA stosowana jest w roli AKO.

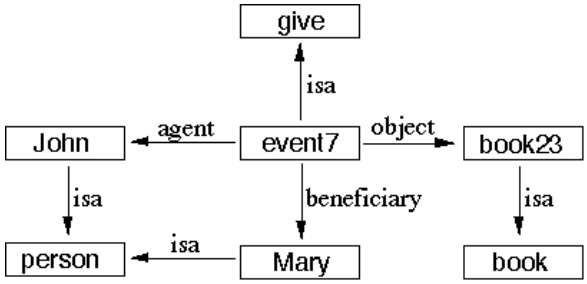
Sieci semantyczne: relacje binarne i reifikacja

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?

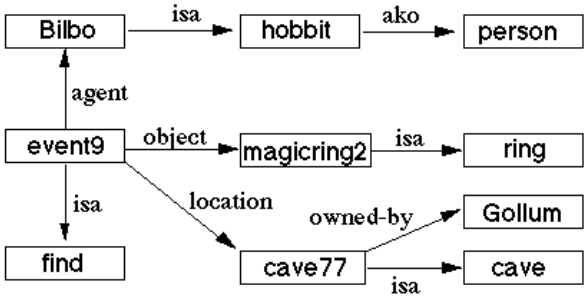
W niektórych przypadkach złożona relacja naturalnie dekomponuje się na składowe binarne. Jednak nie zawsze jest to możliwe. W pozostałych przypadkach stosuje się zabieg **reifikacji**, czyli przekształcenia relacji w obiekty.

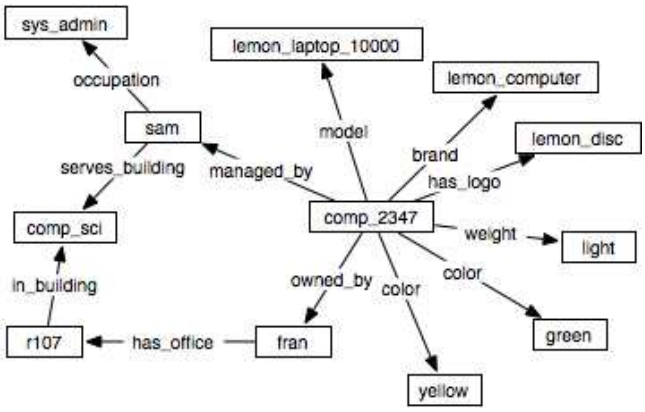
Przykład: *John gives the book to Mary.*



Sieci semantyczne: przykłady

Bilbo finds the magic ring in Gollum's cave.



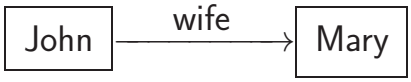


Odczytaj informacje z tej sieci!
Spróbuj przekształcić obiekty reifikowane na relacje złożone.

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?

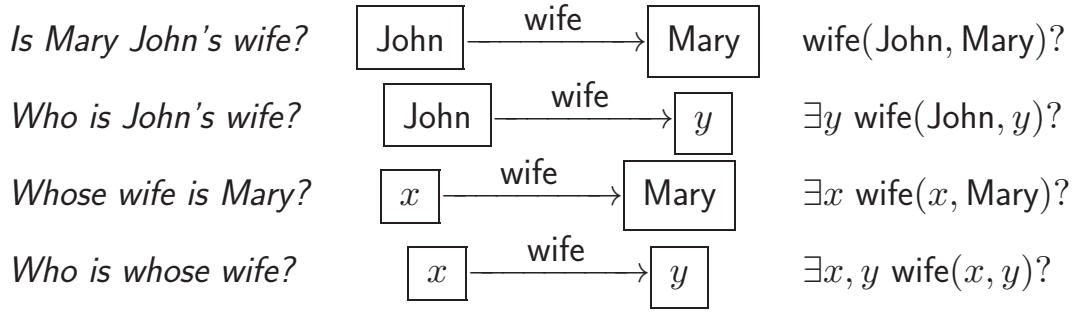
Na przykład:

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.

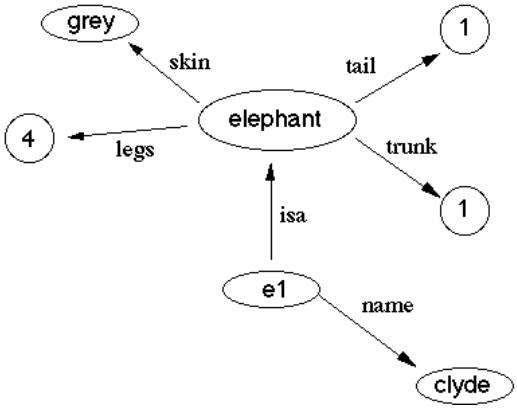


Formułowanie pytań w sieciach semantycznych wymaga użycia zmiennych domyślnie kwantyfikowanych kwantyfikatorem egzystencjalnym, podobnie jak w logice predykatów. Jeśli chcemy uzyskać wartość odpowiedzi w ostatnich trzech pytaniach, to musimy otrzymać od mechanizmu wnioskowania obiekty dopasowane do obiektów-zmiennych.

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 proste 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: $\boxed{\text{e1}} \xrightarrow{\text{skin}} \boxed{\text{pink}}$.

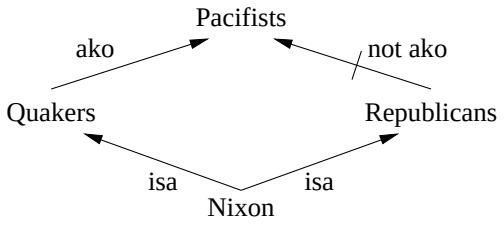
Wtedy odpowiedź na pytanie: $\boxed{\text{e1}} \xrightarrow{\text{skin}} \boxed{z}$ mogłaby być uzyskana bez dziedziczenia przez dopasowanie $z = \text{pink}$. Taka odpowiedź normalnie ma priorytet, tzn. wyklucza uzyskanie odpowiedzi na to samo pytanie przez dziedziczenie.

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.

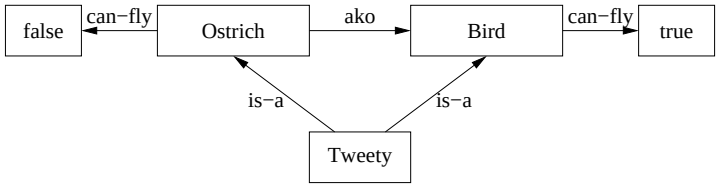
[1] Richard M. Nixon — prezydent U.S.A. w latach 1969–1974. Po wygraniu wyborów na drugą kadencję w 1973r. był zamieszany w następstwa afery Watergate i zrezygnował z urzędu prezydenta pod groźbą usunięcia.

[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: standaryzacja

Jednym z problemów sieci semantycznych jest brak standardowego katalogu relacji (linków). Można wprowadzać dowolne relacje i stosować dowolne nazwy. Utrudnia to zrozumienie nieznanej sieci, 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
$A \xrightarrow{R} B$	$R(A, B)$	clyde $\xrightarrow{\text{skin}}$ pink
$A \xrightarrow{\boxed{R}} B$	$\forall x\ x \in A \Rightarrow R(x, B)$	Bird $\xrightarrow{\boxed{\#\text{legs}}}$ 2
$A \xrightarrow{\boxed{\boxed{R}}} B$	$\forall x \exists y\ x \in A \Rightarrow y \in B \wedge R(x, y)$	Bird $\xrightarrow{\boxed{\boxed{\text{parent}}}}$ Bird

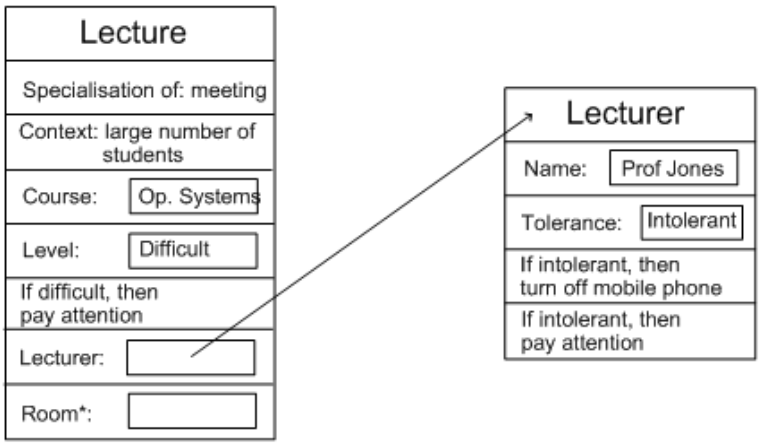
Jednak nadmierna rozbudowa mechanizmów sieci semantycznych, i wprowadzanie kolejnych rodzajów węzłów i łuków sieci niweluje jedną zasadniczą zaletę, jaką jest czytelność reprezentacji graficznej.

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.



System reprezentacji wiedzy oparty na ramkach jest bardzo ogólny, a jednocześnie brak mu formalnej definicji określającej semantykę. Znacznie utrudnia to praktyczne zastosowania komputerowego przetwarzania wiedzy i wnioskowania. Mimo to system ramek stał się podstawą standardu OKBC (*Open Knowledge Base Connectivity*) mającego na celu ujednolicenie interfejsu dostępu do baz wiedzy.

Systemy ramek a modelowanie obiektowe

Systemy ramek są prekursorem systemów obiektowych. Zasadniczym ich podobieństwem jest hierarchiczna budowa oparta na kategoryzacji.

Jednak pomiędzy systemami ramek a systemami obiektowymi są 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. W systemie ramek nie ma sztywnego podziału na ramki-klasy i ramki-obiekty, ponieważ ramka może być obiektem pewnej klasy, i jednocześnie klasą dla pewnej grupy obiektów.

W typowych systemach obiektowych właściwości (atrybuty i medoty) są lokalne względem klasy, np. właściwości o tej samej nazwie mogą występować w różnych klasach nie mając ze sobą nic wspólnego. W klasycznym systemie ramek właściwości (relacje) są reprezentowane przez odpowiednie ramki, a więc są globalne. W niektórych systemach ramek właściwości mogą nie mieć oddzielnej reprezentacji w postaci ramek, ale ich nazwy nadal są globalne.

Semantyka sieci semantycznych i systemów ramek

Nieco wbrew własnej nazwie, sieci semantyczne nie mają precyzyjnie określonej semantyki. Jest to zarówno zaleta jak i wada. W prostych zastosowaniach elastyczność języka opisowego, implementowanego indywidualnie w każdym systemie, daje więcej swobody i możliwości niż np. język logiki matematycznej. W systemach większej skali brak precyzji opisu semantycznego staje się barierą.

Sieci semantyczne pojawiły się w latach 60-tych XX wieku. W latach 70-tych utraciły popularność na rzecz systemów reprezentacji wiedzy opartych na ramkach, które jednak bynajmniej nie miały silniejszej podbudowy semantycznej.

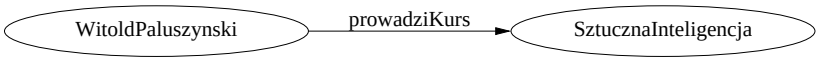
Jednak w XXI-ym wieku sieci semantyczne pojawiły się ponownie, tym razem ze wsparciem semantyki, standardów, oraz języków i systemów programowania. Jak również nadbudowy w postaci ogólniejszego i mocniejszego systemu reprezentacji wiedzy jaką jest logika opisowa. Oraz całego szeregu zastosowań praktycznych związanych z rozwojem Internetu.

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.

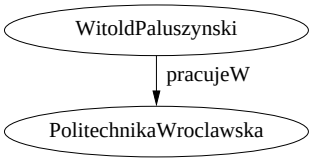
Stwierdzenia jako trójki

- Trójkę (x, P, y) można uważać za formułę logiczną $P(x, y)$, gdzie binary predykat P wiąże obiekt x z obiektem y.

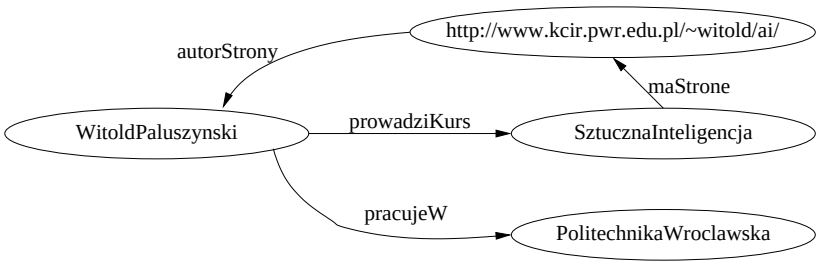
- Trójkę można również uważać za skierowany graf z etykietowanymi węzłami i łukami:
  - skierowany **od** zasobu podmiotu (obiektu) stwierdzenia,
  - skierowany **do** przedmiotu (wartości) stwierdzenia,
  - wartość stwierdzenia może być innym zasobem lub literałem.

- 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 takich 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)
```

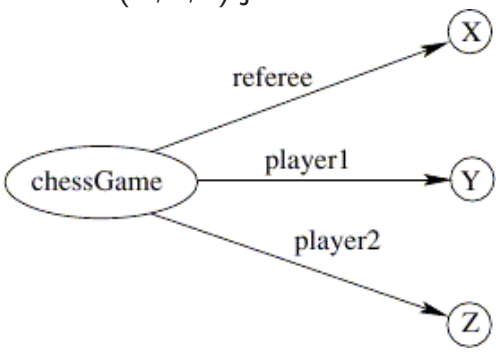
- Zapis ^^ wskazuje typ literału

- 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. Istnieje rozszerzenie języka RDF o nazwie RDF Schema wprowadzające:

- klasy i właściwości,
- hierarchię klas i dziedziczenia,
- hierarchię właściwości.

RDF Schema (zwany również RDFS, RDF(S), RDF-S, albo RDF/S) dostarcza podstawowych elementów do tworzenia opisów dziedzin problemowych o sile wyrazu istotnie większej niż RDF. Ważne jest żeby rozumieć, że RDF Schema nie jest odrębnym językiem tworzenia schematów dla dokumentów RDF, w taki sposób jak XML Schema jest językiem tworzenia schematów dla dokumentów XML.

Nie będziemy tu zgłębiać języka RDF Schema. Semantykę dziedzin będziemy opisywali w inny sposób. Warto dodać, że RDF Schema nie zyskał akceptacji takiej jak RDF, który jest podstawowym standardem inicjatywy Semantic Web.

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.

Co to jest ontologia (2)?

Ontologia jest jawnym, precyzyjnym, i kompletnym, opisem jakiejś części świata, zwanej dziedziną przedmiotową (lub problemową). Celem tworzenia takiego precyzyjnego opisu jest m.in.: uniknięcie nieporozumień, zapewnienie, że wszyscy agenci operujący w lub na danej dziedzinie, rozumieją jej elementy i własności w jednolity sposób. Ontologia musi zawierać specyfikację:

- terminologii uzgodnionej dla danej dziedziny,
- pojęć istniejących w, oraz dotyczących danej dziedziny,
- atrybutów tych pojęć, ich własności, i związków między nimi,
- istniejących więzów na te atrybuty, własności, i związki.

Powyższe elementy stanowią **terminologiczną** wiedzę o dziedzinie. Ontologia może również zawierać część **asercyjną** obejmującą:

- wiedzę o indywiduach/obiektach istniejących w dziedzinie.

Zestawienie, i opisy wszystkich tych elementów danej dziedziny bywa nazywane jej **konceptualizacją**. Dlatego w największym skrócie ontologię danej dziedziny nazywa się jawną specyfikacją jej konceptualizacji.

Po co tworzyć ontologie?

Rolę opisu znaczenia wszystkich pojęć pełnią dotychczas słowniki (listy słów danego języka), tezaurusy (listy związków między słowami: synonimów, antonimów, itp.), 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 właściwe zrozumienie takich opisów znaczeń, musimy je stworzyć w jakimś formalizmie dostępnym dla sztucznie inteligentnych 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.

Ontologie górne i dziedzinowe

Istnieje spora liczba mniej lub bardziej szczegółowo opracowanych ontologii. Rozpoczynając pracę z modelowaniem ontologicznym warto — wręcz należy — zapoznać się z istniejącymi opracowaniami, i jeśli tylko możliwe nie tworzyć nowych ontologii od zera, lecz wykorzystać te istniejące, często dostępne w Internecie on-line. Można podzielić istniejące ontologie świata rzeczywistego na dwie grupy:

Ontologie górne (*upper*) — opisują ogólne pojęcia świata rzeczywistego, wspólne dla wszelkich działań, nienależące do żadnej określonej dziedziny problemowej. Istnieją kontrowersje co do sensu albo możliwości tworzenia ontologii górnych. Jednocześnie, ontologie zaczynają mieć znaczenie komercyjne, zatem istnieje konkurencja w ich tworzeniu i promowaniu jako standardu.

Przykłady ontologii górnych: Dublin Core, Cyc/OpenCyc/ResearchCyc, GFO, SUMO, DOLCE, Wordnet, itp.

Ontologie dziedzinowe — wprowadzają ujednoliconą terminologię, systematykę, i definiują model danych określonej dziedziny problemowej. Znaczenie pojęć ontologii dziedzinowej jest specyficzne dla danej dziedziny. Mogą, ale nie muszą, odwoływać się do jakiejś ontologii górnej.

Przykłady ontologii dziedzinowych: Gene Ontology, SNOMED CT, ...

Język budowy ontologii OWL

OWL (*Ontology Web Language*) jest językiem tworzenia ontologii. W istocie OWL opiera się na grupie standardów, które definiują rodzinę języków. Te definicje są dość specyficzne. Nie ma jednej składni OWL ani nie ma jednej semantyki OWL. W tym wykładzie zapoznamy się dość pobieżnie z językiem OWL2, aktualną wersją standardu, pomijając przy tym wiele aspektów formalnych, a skupiając się na elementach praktycznych.

Ontologie OWL mogą być używane łącznie z informacjami zapisanymi w RDF, i sama ontologia OWL może być zapisana jako dokument RDF. Stosowane są również alternatywne składnie zapisu OWL, ułatwiające czytanie i omawianie fragmentów ontologii. Specyfikacja OWL wymaga, by każda implementacja operowała składnią RDF/XML, ale istnieje również składnia OWL/XML, a także przeznaczone do użytku przez człowieka składnie: Functional-Style, Turtle, i Manchester.

Wnioskowanie w OWL i logika

OWL jest językiem deklaratywnym pozwalającym opisać budowę, właściwości, i aktualny stan pewnej dziedziny problemowej. Ten opis składa się ze zbioru stwierdzeń zapisanych w OWL. Z tych stwierdzeń można **wywieść** dalsze informacje, opierając się na semantyce formalnej. Istnieją dwie semantyki OWL: **Semantyka Bezpośrednia** (*Direct Semantics*), i **Semantyka RDF** (*RDF-Based Semantics*). Takie wywody logiczne realizują narzędzia zwane **silnikami wnioskowania** (*reasoner*), których konstrukcja nie jest częścią specyfikacji OWL.

Podstawą teoretyczną języka OWL jest **logika opisowa** (*Description Logic, DL*), która jest nazwą rodziny formalizmów logicznych stanowiących rozstrzygalne podzbiory matematycznej logiki pierwszego rzędu. Ta rozstrzygalność gwarantuje możliwość implementacji, oraz jednoznaczność działania silników wnioskowania.

Specyfikacja OWL1 była oparta na dialekcie DL o nazwie $\mathcal{SHOIN}$. Język ten ma ekspresywność wyższą niż RDF Schema, ale w praktycznych zastosowaniach niewystarczającą. Nowsza specyfikacja OWL2 oparta jest na logice $\mathcal{SROIC}$, o większej ekspresywności, ale nadal pełnej rozstrzygalności. Związek OWL z logiką opisową oznacza, że konstrukcje danej wersji OWL odpowiadają odpowiednim konstrukcjom logiki. Dzięki temu znane teoretyczne własności DL obejmują OWL.

Modelowanie ontologiczne w OWL

OWL jest językiem reprezentacji wiedzy pozwalającym na wyrażanie, wymianę, i przetwarzanie wiedzy o danej dziedzinie problemowej. Proces tworzenia takiej reprezentacji wiedzy nazywamy modelowaniem, a jego efektem jest ontologia danej dziedziny.

Ontologie zapisane w OWL tworzą klasy, właściwości, indywidua, i wartości danych. W języku OWL występują:

- **encje** (*entities*) — elementy określające pojęcia modelowanej dziedziny (zarówno obiekty, ich własności, jak i relacje między nimi),
- **wyrażenia** (*expressions*) — kombinacje encji i konstruktorów OWL,
- **aksjomaty** (*axioms*) — podstawowe stwierdzenia tworzące ontologię.

Ontologia zbudowana z tych elementów składa się z szeregu **stwierdzeń** (*statements*), mających charakter aksjomatów logicznych. Zapis ontologii stwierdza prawdziwość tych aksjomatów. Ogólnie, dowolne stwierdzenie OWL może być prawdziwe albo fałszywe w kontekście danej ontologii. Może ono być albo obecne jawnie w ontologii albo wywiedzione jako jej konsekwencja.

Wnioskowanie logiczne w OWL

Zbiór stwierdzeń OWL może być spójny (posiada model) albo niespójny (nie posiada modelu). Co może być modelem danego zbioru stwierdzeń jest określone przez semantykę formalną OWL.

Silnik wnioskowania (*reasoner*) jest narzędziem do automatycznego obliczania konsekwencji posiadanego zbioru stwierdzeń. Interakcje pomiędzy aksjomatami ontologicznymi prowadzą czasami do nieoczekiwanych konsekwencji. Automatyczne wywodzenie konsekwencji przez silnik jest zarazem pomocą jak i utrudnieniem w modelowaniu ontologii. Jest pomocą ponieważ pozwala łatwo uzyskać wiele konsekwencji, i zauważyć zarówno pożądane jak i niepożądane efekty wprowadzanych stwierdzeń. Ale jest również utrudnieniem, ponieważ nie zawsze można łatwo przewidzieć konsekwencje aktualnie wprowadzanych stwierdzeń, które silnik obliczy dopiero później, po pojawieniu się innych stwierdzeń.

Na przykład, istnieje zasadnicza różnica pomiędzy semantyką bazy wiedzy a bazy danych. Fakt nieistniejący w bazie danych przyjmuje się zwykle jako fałszywy, tzw. **założenie świata zamkniętego** (*closed-world assumption*, CWA). Ontologia OWL przyjmuje **założenie świata otwartego** (*open-world assumption*, OWA).

Klasy i indywidua

W tym wykładzie przykładową dziedziną wykorzystaną do ilustracji mechanizmów modelowania OWL będzie funkcjonowanie nauki polskiej, a dokładniej pewien jej widok obejmujący naukowców, instytucje naukowe, i ich struktury.

```
ClassAssertion( :Uczelnia :PWr )
```

Powyższe stwierdzenie stwierdza przynależność indywiduum (:PWr) do klasy (:Uczelnia). Stwierdzenie zostało zapisane w tzw. **notacji funkcjonalnej** (*Functional-Style syntax*) OWL. Encje są ogólnie reprezentowane przez URI/IRI, jednak dla zwiększenia czytelności stosowane są skróty z prefiksami reprezentującymi przestrzenie nazw. W OWL można stosować pusty prefiks, który reprezentuje domyślną przestrzeń nazw.

```
SubClassOf( :Uczelnia :JednostkaNaukowa )
SubClassOf( :JednostkaNaukowa :NaukaPolska )
EquivalentClasses( :Uczelnia :SzkolaWyzsza )
```


Związki między klasami

Aksjomaty opisujące związki klas `SubClassOf` stosuje się typowo do opisania pełnej hierarchii klas występujących w danej dziedzinie, np.:

```
SubClassOf( :InstytucjaNaukowa :NaukaPolska )
SubClassOf( :SzkolaWyzsza :InstytucjaNaukowa )
SubClassOf( :InstytutPAN :InstytucjaNaukowa )
SubClassOf( :InstytutBranzowy :InstytucjaNaukowa )

SubClassOf( :JednostkaNaukowa :NaukaPolska )
SubClassOf( :Uczelnia :JednostkaNaukowa )
SubClassOf( :Wydzial :JednostkaNaukowa )
SubClassOf( :Instytut :JednostkaNaukowa )
SubClassOf( :Katedra :JednostkaNaukowa )

SubClassOf( :PracownikNaukowy :NaukaPolska )

SubClassOf( :Kurs :NaukaPolska )

...
```

Wnioskowanie w hierarchii klas

Hierarchia klas pozwala na wnioskowanie o własnościach indywiduów przez dziedziczenie. To wnioskowanie wykonuje silnik wnioskowania (*reasoner*). Dziedziczenie w hierarchii klas wykorzystuje fakt, że relacja bycia podklasą jest przechodnia. Jest ona również zwrotna, tzn. każda klasa jest swoją podklasą.

W hierarchii klas występują pewne dalsze związki, których silnik wnioskowania nie może stwierdzić sam, ale które może wykorzystać, na przykład równoważność albo rozłączność klas:

```
EquivalentClasses( :SzkolaWyzsza :Uczelnia )
DisjointClasses( :SzkolaWyzsza :InstytutPAN :InstytutBranzowy )
DisjointClasses( :Uczelnia :Wydzial :Instytut :Katedra )
```

W praktyce, o ile równoważność klas wprowadza w istocie alternatywną nazwę klasy (alias), która pojawia się jedynie właśnie przez swoją definicję, to rozłączność jest własnością, którą łatwo pominąć tworząc aksjomatykę dziedziny. Częstą przyczyną tego jest intuicyjne przyjmowanie rozłączności klas dla których nie jest jawnie zdefiniowany jakiś inny związek. Jednak posiadanie aksjomatów rozłączności klas pozwala wywieść wiele ważnych i potrzebnych własności.

Wyrażenia określające klasy

W aksjomatach dotyczących klas mogą pojawiać się nie tylko jawne identyfikatory klas, ale również wyrażenia określające klasy. Przykłady takich wyrażeń:

```
ObjectIntersectionOf( C1 C2 ... Cn )
ObjectUnionOf( C1 C2 ... Cn )
ObjectComplementOf( C )
ObjectOneOf( a1 a2 ... an )
```

Zauważmy, że aksjomat z konstrukcją `EquivalentClasses` gdzie jednym argumentem jest nazwa klasy a drugim wyrażenie określające klasę, pełni rolę definicji klasy, np.

```
EquivalentClasses( :JednostkaNaukowa
                   ObjectUnionOf( :SzkolaWyzsza :InstytutPAN
                                  :InstytutBranzowy ) )
```

Istnieje konstruktor `DisjointUnionOf` który pozwala zdefiniować klasę jako sumę mnogościową rozłącznych podklas:

```
DisjointUnionOf( CW C1 C2 ... Cn )
```

Powyższe stwierdzenie jest równoważne parze stwierdzeń:

```
DisjointClasses( C1 C2 ... Cn )
EquivalentClasses( CW ObjectUnionOf( C1 C2 ... Cn ) )
```

Właściwości obiektów klas

Właściwości obiektów klas (*object properties*) są definiowane dla klas, i służą do wyrażania związków pomiędzy indywiduami tych klas, np.:

```
ObjectPropertyAssertion( :PracujeW :WitoldP :PolitechnikaWroclawska )
ObjectPropertyAssertion( :ProwadziKurs :WitoldP :SztInt )
NegativeObjectPropertyAssertion( :ProwadziKurs :WitoldP :Java )
```

W OWL właściwości są zawsze binarne (właściwości unarne są modelowane jako klasy zdefiniowane lub określone wyrażeniem). Możemy zdefiniować **dziedzinę** i/lub **zakres** właściwości binarnej (dziedzina jest w rzeczywistości dziedziną pierwszego argumentu relacji, a zakres dziedziną drugiego), np.:

```
ObjectPropertyDomain( :ProwadziKurs :PracownikNaukowy )
ObjectPropertyRange( :ProwadziKurs :Kurs )
```


Warto zwrócić uwagę, że definiowanie dziedzin i zakresów relacji pełni w ontologiach trochę inną rolę niż np. w systemach baz danych. W tych ostatnich takie stwierdzenia pełnią rolę więzów i mogą być podstawą uznania pewnych stwierdzeń za fałszywe, np:

```
ObjectPropertyAssertion( :ProwadziKurs :DonaldTusk :Europeistyka )
```

Możnaby mieć wątpliwości czy wiedza, że `:DonaldTusk` jest politykiem nie powoduje sprzeczności z powyższym aksjomatem dziedziny własności `:ProwadziKurs`. Ponieważ zapisana ontologia wywodzi swoją semantykę z logiki (bez typów), powyższe stwierdzenie nie prowadzi do sprzeczności, o ile tylko nie jest jawnie sprzeczne z posiadaną wiedzą ogólną. Natomiast bezsprzecznie pozwoliłoby silnikowi wnioskowania na wywiedzenie, że `:DonaldTusk` jest pracownikiem nauki polskiej.

Hierarchie właściwości

Właściwości mogą tworzyć hierarchię pozwalające wnioskować przez dziedziczenia, podobne do hierarchii klas, np.:

```
SubObjectPropertyOf( :ProwadziEgzamin :ProwadziKurs )
```

Dziedziczenie właściwości między obiektami przebiega od właściwości podrzędnej do nadrzędnej. Z wiedzy o podrzędności właściwości możemy również wnioskować o dziedzinie i zakresie właściwości, tylko tutaj dziedziczenie przebiega od właściwości nadrzędnej do podrzędnej.

Na przykład, z powyższych faktów silnik wnioskowania może wywieść, że dziedziną właściwości `:ProwadziEgzamin` będzie `:PracownikNaukowy`).

Identyczność indywiduów

OWL nie czyni żadnych założeń co do unikalności nazw, zatem można (i należy) jawnie wyrażać fakty o posługiwaniu się różnymi nazwami (aliasami) wobec jednego obiektu, jak również o tym, że różne nazwy w istocie odnoszą się do różnych indywiduów:

```
SameIndividuals( :PWr :PolitechnikaWroclawska )
DifferentIndividuals( :PolitechnikaWroclawska :UniwersytetWroclawski )
```

Typy danych

Logiki opisowe

Logiki opisowe — notacja Manchester

notacja DL	notacja Manchester	konstrukcja OWL	przykład	komentarz
$C \sqcap D$	C and D	intersectionOf	Rodzic and Kobieta	matka
$C \sqcup D$	C or D	unionOf	Mężczyzna lub Kobieta	osoba
$\neg\, C$	not C	complementOf	not Rodzic	bez dzieci
$\forall R.C$	R only C	allValuesFrom	maDziecko only Kobieta	tylko córki
$\exists R.C$	R some C	someValuesFrom	maDziecko some Kobieta	ma córkę
$\exists R.\{a\}$	R value a	hasValue	maDziecko value kasia	ma dziecko Kasię
$\geq$	R min n	minCardinality	maDziecko min 3	
$\leq$	R max n	maxCardinality	maDziecko max 3	
$=$	R exactly n	cardinality	maDziecko exactly 3	
$\{a, b, c\}$	{a b c}	oneOf	{Włochy Niemcy Francja}	

Dodatkowo słowo kluczowe `that` jest synonimem `and`, co daje bardziej czytelne (po angielsku) wyrażenia, gdy prawy argument jest wyrażeniem, np. „Person that hasChild some Woman".

Notacja Manchester sprzyja beznawiasowemu zapisowi wyrażeń dzięki priorytetom:

najwyższy:	some, only, value, min, max, exactly, that
	not
	and
najniższy:	or

Przydatne materiały

- K.Goczyła — Ontologie w systemach informatycznych, EXIT, Warszawa 2011
- Ontology101
  http://protege.stanford.edu/publications/ontology_development/ontology101-noy-mcguinness.html
- OWL 2 Primer (Second Edition) W3C Recommendation 11 December 2012:
  http://www.w3.org/TR/owl2-primer/
- Pełny opis notacji Manchester:
  http://webont.org/owled/2008dc/papers/owled2008dc_paper_11.pdf