UNICAL - A.A. 2006-2007

Gestione della Conoscenza

Prof. Massimo Ruffolo

Ing. Marco Manna

Capitolo 6

- …
- Modelli basati sulla DLP
  - La Programmazione Logica Disgiuntiva
  - Il linguaggio **Onto*DLP***
  - Sintassi e Semantica
- …

Modelli basati sulla DLP

La Programmazione Logica Disgiuntiva

Enumerazione

- Una classe **A** può essere descritta *dall'enumerazione* di un numero finito di nominali a_1, …, a_n
- Sintassi ***DL***
  - $A \equiv \{a_1, \ldots, a_n\}$
- Modello ***DLP***
  - $A(a_1)$.
  - . . .
  - $A(a_n)$.

Restrizioni di proprietà (1)

- Una classe **A** può essere descritta come *restrizione* su un insieme di Individui con ruolo **R** su **almeno** un individuo della classe **C**
- Sintassi ***DL***
  - $\mathbf{A} \equiv \exists \mathbf{R}.\mathbf{C}$
- Modello ***DLP***
  - `A(X) :- R(X,Y), C(Y).`

Restrizioni di proprietà (2)

- Una classe **A** può essere descritta come *restrizione* su un insieme di Individui con ruolo **R** su **soli individui** di classe **C**
- Sintassi ***DL***
  - $\mathbf{A} \equiv \forall \mathbf{R}.\mathbf{C}$
- Modello ***DLP***
  - `A(X) :- R(X,_), not notA(X).`
  - `notA(X) :- R(X,Y), not C(Y).`

Restrizioni di proprietà (3)

- Una classe **A** può essere descritta come *restrizione* su un insieme di Individui con ruolo **R** **sul solo individuo** a
- Sintassi ***DL***
  - $\mathbf{A} \equiv \forall \mathbf{R}.\{\mathbf{a}\}$
- Modello ***DLP***
  - `A(X) :- R(X,_), not notA(X).`
  - `notA(X) :- R(X,Y), Y<>a.`

Restrizioni di proprietà (4)

- Una classe **`A`** può essere descritta come *restrizione* su un insieme di Individui con ruolo **`R`** su **al massimo `n` individui**
- Sintassi ***DL***
  - $\mathtt{A} \equiv \leq \mathtt{n}\mathtt{R}$
- Modello ***DLP***
  - `A(X) :- R(X,_), #count{Y: R(X,Y)} <= n.`

Restrizioni di proprietà (5)

- Una classe **A** può essere descritta come *restrizione* su un insieme di Individui con ruolo **R** su **almeno n individui**
- Sintassi ***DL***
  - $\mathbf{A} \equiv \geq \mathbf{n}\mathbf{R}$
- Modello ***DLP***
  - `A(X) :- R(X,_), #count{Y: R(X,Y)} >= n.`

Restrizioni di proprietà (6)

- Una classe **A** può essere descritta come *restrizione* su un insieme di Individui con ruolo **R** su **esattamente n individui**
- Sintassi ***DL***
  - **A ≡ =nR**
- Modello ***DLP***
  - `A(X) :- R(X,_), #count{Y: R(X,Y)} = n.`

Intersezione

- Una classe **A** può essere descritta come *intersezione* di un numero finito di classi $C_1, \ldots, C_n$
- Sintassi ***DL***
  - $A \equiv C_1 \sqcap \ldots \sqcap C_n$
- Modello ***DLP***
  - `A(X) :- C1(X), ... , Cn(X).`

Unione

- Una classe **A** può essere descritta come *unione* di un numero finito di classi C_1, ..., C_n
- Sintassi ***DL***
  - $A \equiv C_1 \sqcup \ldots \sqcup C_n$
- Modello ***DLP***
  - `A(X) :- C1(X).`
  - `...`
  - `A(X) :- Cn(X).`

Complemento

- Una classe **A** può essere descritta come *complemento* di un'altra classe **B**
- Sintassi ***DL***
  - $\mathbf{A} \equiv \neg \mathbf{B}$
- Modello ***DLP***
  - `A(X) :- thing(X), not B(X).`

Sottoclassi

- Fra due descrizioni di classi **C** e **D** si può definire una relazione di sottoclasse
- Sintassi ***DL***
  - $C \sqsubseteq D$
- Modello ***DLP***
  - `D(X) :- C(X).`

Equivalenza

- Fra due descrizioni di classi **C** e **D** si può definire una relazione di equivalenza
- Sintassi ***DL***
  - **C ≡ D**
- Modello ***DLP***
  - **D(X) :- C(X).**
  - **C(X) :- D(X).**

Disgiunzione

- Fra due descrizioni di classi **C** e **D** si può dichiarare una relazione di disgiunzione
- Sintassi ***DL***
  - $C \sqcap D \equiv \bot$
- Modello ***DLP***
  - `C(X) v D(X) :- thing(X).`

  oppure
  - `C(X) :- thing(X), not D(X).`
  - `D(X) :- thing(X), not C(X).`

Sottoproprietà

- Una proprietà **R** può essere definita come sottoproprietà di un'altra proprietà **S**
- Sintassi ***DL***
  - $\mathbf{R} \sqsubseteq \mathbf{S}$
- Modello ***DLP***
  - `S(X,Y) :- R(X,Y).`

Dominio

- Di una proprietà **R** può essere specificato il dominio
- Sintassi ***DL***
  - $\top \sqsubseteq \forall$ **R.C** *(definizione del dominio* **D***)*
- Modello ***DLP***
  - `:- R(X, Y), not C(Y).`

Codominio

- Di una proprietà **R** può essere specificato il codominio
- Sintassi ***DL***
  - $\top \sqsubseteq \forall R^{-}.D$ *(definizione del codominio* **C***)*
- Modello ***DLP***
  - `:- R(X, Y), not D(X).`

Proprietà equivalente

- Una proprietà **R** può essere definita come equivalente a un'altra proprietà **S**
- Sintassi ***DL***
  - **R ≡ S**
- Modello ***DLP***
  - **R(X, Y) :- S(X, Y).**
  - **S(X, Y) :- R(X, Y).**

Proprietà inversa

- Data una proprietà **R** si può definire la proprietà inversa **S**
- Sintassi ***DL***
  - $S \equiv R^-$
- Modello ***DLP***
  - `S(Y, X) :- R(X, Y).`

Funzionalità

- Una proprietà **R** è funzionale se soddisfa il vincolo di cardinalità globale
- Sintassi ***DL***
  - $\top \sqsubseteq \leq 1 R$
- Modello ***DLP***
  - `:- R(X,_), #count{Y: R(X, Y)} > 1.`

Transitività

- In OWL è possibile dichiarare che una proprietà transitiva
- Sintassi ***DL***
  - ***Tr*(R)**
- Modello ***DLP***
  - **R(X, Z) :- R(X, Y), R(Y, Z).**

Simmetria

- In OWL è possibile dichiarare che una proprietà simmetrica
- Sintassi ***DL***
  - $R \sqsubseteq R^-$
- Modello ***DLP***
  - `R(Y, X) :- R(X, Y).`

Appartenenza a una classe

- In OWL è possibile specificare che un individuo **`a`** appartiene a una classe **`C`**
- Sintassi ***DL***
  - **`C(a)`**
- Modello ***DLP***
  - **`C(a).`**

Valori di proprietà

- In OWL è possibile specificare che una proprietà **`R`** di un individuo **`a`** ha valore **`b`**
- Sintassi ***DL***
  - **`R(a,b)`**
- Modello ***DLP***
  - **`R(a,b).`**

Identità (1)

- Il linguaggio **OWL** non assume che gli individui abbiano nome unico. Quindi è possibile asserire che due nomi fanno riferimento allo stesso
- Sintassi ***DL***
  - `a = b`
- Modello ***DLP***
  - `Non esprimibile in generale...`
  - `idem(a,b).`
  - `idem(X,Y) :- idem(Y,X).`
  - `idem(X,X) :- idem(X,_).`
  - `idem(Y,Y) :- idem(_,Y).`

Identità (2)

- Analogamente è possibile asserire che due nomi fanno riferimento al individui distinti
- Sintassi ***DL***
  - $\mathtt{a} \neq \mathtt{b}$
- Modello ***DLP***
  - `Non esprimibile in generale...`
  - `distinct(a,b).`
  - `distinct(X,Y) :- distinct(Y,X).`

Identità (3)

- È anche possibile asserire che **n** individui sono tutti distinti fra loro
- Sintassi ***DL***
  - **a1 ≠ … ≠ an**
- Modello ***DLP***
  - **Non esprimibile in generale...**
  - **distinct(a1,a2).**
  - …
  - **distinct(a1,an).**
  - **distinct(Y,Z) :- distinct(X,Y), distinct(X,Z).**

Esempi (I)

- *DL* **GENITORE ≡ PERSONA ⊓ ∃GenDi**
  - *DLP* `GENITORE(X) :- PERSONA(X), GenDi(X, _).`
- *DL* **GenDi : PERSONA → PERSONA**
  - *DLP* `:- GenDi(X, Y), not PERSONA(Y).`
  - *DLP* `:- GenDi(X, Y), not PERSONA(X).`
- *DL* **DONNA ≡ PERSONA ⊓ FEMMINA**
  - *DLP* `DONNA(X) :- PERSONA(X), FEMMINA(X).`
- *DL* **UOMO ≡ PERSONA ⊓ ¬FEMMINA**
  - *DLP* `UOMO(X) :- PERSONA(X), not FEMMINA(X).`
- *DL* **MADRE ≡ GENITORE ⊓ FEMMINA**
  - *DLP* `MADRE(X) :- GENITORE(X), FEMMINA(X).`

Esempi (II)

- *DL* **PADRE ≡ GENITORE ⊓ ¬FEMMINA**
  - *DLP* `PADRE(X) :- GENITORE(X), not FEMMINA(X).`
- *DL* **STATO ≡ {au, ch, de, es, fr, it, uk}**
  - *DLP* `STATO(au). STATO(ch). … STATO(uk).`
- *DL* **CittDi : PERSONA → STATO**
  - *DLP* `:- CittDi(X, Y), not STATO(Y).`
  - *DLP* `:- CittDi(X, Y), not PERSONA(X).`
- *DL* **ITAL ≡ PERSONA ⊓ ∃CittDi.{it}**
  - *DLP* `ITAL(X) :- PERSONA(X), CittDi(X, it).`
- *DL* **BRIT ≡ PERSONA ⊓ ∃CittDi.{uk}**
  - *DLP* `BRIT(X) :- PERSONA(X), CittDi(X, uk).`

Modelli basati sulla DLP

Il linguaggio Onto*DLP*

Classi

- Una classe **A** può essere definita come segue
- Sintassi ***DL***
  - **A**
- Modello **Onto*DLP***
  - `class A().`

Enumerazione

- Una classe **A** può essere descritta *dall'enumerazione* di un numero finito di nominali a_1, …, a_n
- Sintassi ***DL***
  - $A \equiv \{a_1, \ldots, a_n\}$
- Modello **Onto*DLP***
  - `class A().`
  - a_1`:A().`
  - `...`
  - a_n`:A().`

Restrizioni di proprietà (1)

- Una classe **A** può essere descritta come *restrizione* su un insieme di Individui con ruolo **R** su **almeno** un individuo della classe **C**
- Sintassi ***DL***
  - $\mathbf{A} \equiv \exists \mathbf{R}.\mathbf{C}$
- Modello **Onto*DLP*** *(collections)*
  - `collections class A().`
  - `X:A() :- R(X,Y), Y:C().`

Restrizioni di proprietà (2)

- Una classe **A** può essere descritta come *restrizione* su un insieme di Individui con ruolo **R** su **soli individui** di classe **C**
- Sintassi ***DL***
  - $\mathbf{A} \equiv \forall \mathbf{R}.\mathbf{C}$
- Modello **Onto*DLP*** *(collections)*
  - `collections class A().`
  - `X:A() :- R(X,_), not notA(X).`
  - `notA(X) :- R(X,Y), not C_rel(Y).`
  - `C_rel(X) :- X: C().`

Restrizioni di proprietà (3)

- Una classe **A** può essere descritta come *restrizione* su un insieme di Individui con ruolo **R** **sul solo individuo** a
- Sintassi ***DL***
  - $\mathbf{A} \equiv \forall \mathbf{R}.\{a\}$
- Modello **Onto*DLP*** *(collections)*
  - `collections class A().`
  - `X:A() :- R(X,_), not notA(X).`
  - `notA(X) :- R(X,Y), Y<>a.`

Restrizioni di proprietà (4)

- Una classe **A** può essere descritta come *restrizione* su un insieme di Individui con ruolo **R** su **al massimo n individui**
- Sintassi ***DL***
  - $\mathbf{A \equiv \leq n R}$
- Modello **Onto*DLP*** *(collections)*
  - `collections class A().`
  - `X:A() :- R(X,_), #count{Y: R(X,Y)} <= n.`

Restrizioni di proprietà (5)

- Una classe **A** può essere descritta come *restrizione* su un insieme di Individui con ruolo **R** su **almeno n individui**
- Sintassi ***DL***
  - **A ≡ ≥nR**
- Modello **Onto*DLP*** *(collections)*
  - `collections class A().`
  - `A(X) :- R(X,_), #count{Y: R(X,Y)} >= n.`

Restrizioni di proprietà (6)

- Una classe **A** può essere descritta come *restrizione* su un insieme di Individui con ruolo **R** su **esattamente n individui**
- Sintassi ***DL***
  - **A ≡ =nR**
- Modello **Onto*DLP*** *(collections)*
  - `collections class A().`
  - `A(X) :- R(X,_), #count{Y: R(X,Y)} = n.`

Intersezione

- Una classe **A** può essere descritta come *intersezione* di un numero finito di classi C_1, ..., C_n
- Sintassi ***DL***
  - $A \equiv C_1 \sqcap \ldots \sqcap C_n$
- Modello **Onto*DLP*** *(collections)*
  - `collections class A().`
  - `X:A() :- X:C1(), ... , X:Cn().`

Unione

- Una classe **A** può essere descritta come *unione* di un numero finito di classi C_1, …, C_n
- Sintassi ***DL***
  - $A \equiv C_1 \sqcup \ldots \sqcup C_n$
- Modello **Onto*DLP*** *(collections)*
  - `collections class A().`
  - `X:A() :- X:C1().`
  - `...`
  - `X:A() :- X:Cn().`

Complemento

- Una classe **A** può essere descritta come *complemento* di un'altra classe **B**
- Sintassi ***DL***
  - **A** ≡ ¬**B**
- Modello **Onto*DLP*** *(collections)*
  - `collections class A().`
  - `X:A() :- thing(X), not B_rel(X).`
  - `B_rel(X) :- X: B().`

Sottoclassi

- Fra due descrizioni di classi **C** e **D** si può definire una relazione di sottoclasse
- Sintassi ***DL***
  - $C \sqsubseteq D$
- Modello **Onto*DLP***
  - `class C isa {D}.`

Equivalenza

- Fra due descrizioni di classi **C** e **D** si può definire una relazione di equivalenza
- Sintassi ***DL***
  - **C ≡ D**
- Modello ***DLP***
  - **Non esprimibile (non sono ammessi cicli nella gerarchia 'isa')**

Disgiunzione

- Fra due descrizioni di classi **`C`** e **`D`** si può dichiarare una relazione di disgiunzione
- Sintassi ***DL***
  - $C \sqcap D \equiv \bot$
- Modello **Onto*DLP*** *(collections)*
  - `collections class C().`
  - `collections class D().`
  - `X:C() :- thing(X), not D_rel(X).`
  - `X:D() :- thing(X), not C_rel(X).`
  - `C_rel(X) :- X:C().`
  - `D_rel(X) :- X:D().`

Sottoproprietà

- Una proprietà **R** può essere definita come sottoproprietà di un'altra proprietà **S**
- Sintassi ***DL***
  - $R \sqsubseteq S$
- Modello **Onto*DLP***
  - `relation R isa {S}.`

Dominio

- Di una proprietà **R** può essere specificato il dominio
- Sintassi ***DL***
  - $\top \sqsubseteq \forall$ **R.C** *(definizione del dominio* **D***)*
- Modello **Onto*DLP***
  - **relation R(..., cod: D).**

Codominio

- Di una proprietà **R** può essere specificato il codominio
- Sintassi ***DL***
  - $\top \sqsubseteq \forall R^{-}.D$ *(definizione del codominio* **C***)*
- Modello **Onto*DLP***
  - `relation R(dom: D, ...).`

Proprietà equivalente

- Una proprietà **R** può essere definita come equivalente a un'altra proprietà **S**
- Sintassi ***DL***
  - **R ≡ S**
- Modello ***DLP*** or **Onto*DLP***
  - **R(X, Y) :- S(X, Y).**
  - **S(X, Y) :- R(X, Y).**

Proprietà inversa

- Data una proprietà **R** si può definire la proprietà inversa **S**
- Sintassi ***DL***
  - **S ≡ R⁻**
- Modello ***DLP*** or **Onto*DLP***
  - **S(Y, X) :- R(X, Y).**

Funzionalità

- Una proprietà **R** è funzionale se soddisfa il vincolo di cardinalità globale
- Sintassi ***DL***
  - $\top \sqsubseteq \leq 1 R$
- Modello ***DLP*** or **Onto*DLP***
  - `:- R(X,_), #count{Y: R(X, Y)} > 1.`

Transitività

- In OWL è possibile dichiarare che una proprietà transitiva
- Sintassi ***DL***
  - ***Tr*(R)**
- Modello ***DLP*** or **Onto*DLP***
  - **R(X, Z) :- R(X, Y), R(Y, Z).**

Simmetria

- In OWL è possibile dichiarare che una proprietà simmetrica
- Sintassi ***DL***
  - $\mathbf{R \sqsubseteq R^-}$
- Modello ***DLP*** or **Onto*DLP***
  - `R(Y, X) :- R(X, Y).`

Appartenenza a una classe

- In OWL è possibile specificare che un individuo **`a`** appartiene a una classe **`C`**
- Sintassi ***DL***
  - `C(a)`
- Modello ***DLP***
  - `a:C().`

Valori di proprietà

- In OWL è possibile specificare che una proprietà **`R`** di un individuo **`a`** ha valore **`b`**
- Sintassi ***DL***
  - **`R(a,b)`**
- Modello ***DLP*** or **Onto*DLP***
  - **`R(a,b).`**

Identità (1)

- Il linguaggio **OWL** non assume che gli individui abbiano nome unico. Quindi è possibile asserire che due nomi fanno riferimento allo stesso
- Sintassi ***DL***
  - `a = b`
- Modello ***DLP*** or **Onto*DLP***
  - `Non esprimibile in generale...`
  - `idem(a,b).`
  - `idem(X,Y) :- idem(Y,X).`
  - `idem(X,X) :- idem(X,_).`
  - `idem(Y,Y) :- idem(_,Y).`

Identità (2)

- Analogamente è possibile asserire che due nomi fanno riferimento al individui distinti
- Sintassi ***DL***
  - `a ≠ b`
- Modello ***DLP*** or **Onto*DLP***
  - `Non esprimibile in generale...`
  - `distinct(a,b).`
  - `distinct(X,Y) :- distinct(Y,X).`

Identità (3)

- È anche possibile asserire che **n** individui sono tutti distinti fra loro
- Sintassi ***DL***
  - **a1 ≠ … ≠ an**
- Modello ***DLP*** or **Onto*DLP***
  - **Non esprimibile in generale...**
  - **distinct(a1,a2).**
  - …
  - **distinct(a1,an).**
  - **distinct(Y,Z) :- distinct(X,Y), distinct(X,Z).**

Modelli basati sulla DLP

Sintassi e Semantica

Onto*DLP*

- Estensione della Programmazione Logica Disgiuntiva *(DLP)* arricchita con le nozioni basilari della programmazione orientata agli oggetti
- Permette, infatti, di specificare oggetti, classi *(eventualmente collocate in una gerarchia di ereditarietà attraverso la relazione ISA)*, e relazioni tra gli oggetti
- Consente, inoltre, l'asserzione di assiomi, ovvero proprietà che devono essere verificate per ogni elemento del dominio di conoscenza. Il soddisfacimento di tali assiomi assicura la consistenza del dominio rappresentato
- Il ragionamento sul dominio di conoscenza può essere effettuato attraverso moduli di ragionamento *(veri e propri programmi logici)*

Classi (1)

- Uno dei più potenti meccanismi di astrazione per la rappresentazione della conoscenza è la classificazione, cioè l'identificazione di categorie di oggetti (o classi) sulla base dell'osservazione di differenze di rilievo tra le entità stesse.
- Supponiamo di voler modellare il dominio degli esseri viventi, e supponiamo che siano state individuate quattro classi di individui: le persone, gli animali, il cibo e i luoghi.
- Queste classi possono essere definite in **Onto*DLP*** come segue

```
class persona.
class animale.
class cibo.
class luogo.
```

Classi (2)

- Proprietà comuni a tutti gli individui di una classe espresse attraverso la dichiarazione di attributi
- Un attributo è una coppia *(nome-attributo, tipo-attributo)*
  - *nome-attributo* è il nome della proprietà
  - *tipo-attributo* è la classe della proprietà
- Classi built-in
  - **`integer`** (intero positivo)
  - **`string`** (stringa)
- Esempio

```
class persona(nome: string, eta: integer,
  luogoDiNascita: luogo).
```

Classi (3)

- Specificare, come tipo degli attributi, classi definite dall'utente, consente la definizione di *oggetti complessi*
- Gli attributi modellano proprietà che caratterizzano **tutti gli individui** di una data classe
- Proprietà che possono essere presenti solo in corrispondenza di alcuni individui devono essere modellate attraverso l'uso delle relazioni
- Esempio

```
class luogo(nome: string).
class cibo(nome: string, origine: luogo).
class animale(nome: string, eta: integer, specie:
  string).
```

Oggetti (1)

- Ogni dominio di conoscenza è costituito da una popolazione di individui, chiamati oggetti o istanze
- Ogni individuo in **Onto*DLP*** appartiene ad una classe ed è univocamente identificato da una costante detta **object identifier** *(oid)* o *surrogato*
- **Onto*DLP*** permette la rappresentazione di oggetti tramite l'utilizzo di un tipo speciale di fatti logici
- Esempio

```
roma: luogo(nome: "Roma").
```

Oggetti (2)

- Quando dichiariamo un'istanza specifichiamo un **oid** *(in questo caso roma)* e un valore per tutti gli attributi *(in questo caso il nome dell'istanza è la stringa "Roma")*
- L'oid **`roma`** può essere usato per riferire questo posto
- In **Onto*DLP*** è permesso anche dichiarare istanze senza fornire esplicitamente un **oid**. In tal caso è il sistema che si occupa della generazione automatica di un apposito **oid** per la nuova istanza

Oggetti (3)

- Modo compatto per rappresentare più individui di una stessa classe
- OID espliciti

```
instanceof luogo {
  roma:(nome: "Roma").
  lazio:(nome: "lazio").
  europa:(nome: "Europa").}.
```

- OID impliciti

```
instanceof luogo {
  (nome: "Roma").
  (nome: "lazio").
  (nome: "Europa").}.
```

Oggetti (4)

- Le istanze di classe possono essere definite sia utilizzando una notazione non posizionale in cui gli attributi, dati dalla coppia nome-valore, vengono elencati non necessariamente nell'ordine dello schema di classe, sia attraverso una notazione posizionale, in cui vengono elencati solo i valori degli attributi, seguendo però strettamente l'ordine degli attributi nello schema
- Esempio

```
class persona(nome: string, eta: integer,
  luogoDiNascita: luogo).
john: persona(luogoDiNascita: milano,
  nome: "John", eta: 34).
john: persona("John", 34,  milano).
```

Ereditarietà tra Classi (1)

- Specializzazione/Generalizzazione
- Tale meccanismo consente di organizzare i concetti in tassonomie. Nei linguaggi orientati agli oggetti, lo strumento dell'ereditarietà è il mezzo per definire tale meccanismo
- **Onto*DLP*** fornisce supporto all'ereditarietà attraverso la relazione binaria **`isa`**.
- Esempio

```
class studente isa {persona} (matricola: string, scuola:
   string, tutor: persona).
class impiegato isa {persona} (salario: integer, skill:
   string, azienda: string, tutor: impiegato).
```

- **Persona** è un concetto più generale *(superclasse)* di **studente** e **impiegato** che sono specializzazioni di persona *(sottoclassi)*

Ereditarietà tra Classi (2)

- Una conseguenza importante (ed utile) di queste dichiarazioni è che ogni istanza (propria) di **impiegato** verrà automaticamente considerata una istanza della classe **persona** *(l'opposto, ovviamente, non vale)*
- Esempio
  - ```
    al: studente (nome: "Alfred", eta: 20,
    luogoNascita: roma, matricola: "100",
    scuola: "Cambridge", tutor: hanna).
    ```
  - ```
    jack: impiegato(nome: "Jack", eta: 54,
    luogoNascita: roma, salario: 1000,
    skill: "Java programmer", azienda: "SUN",
    tutor: betty).
    ```
- Tali istanze sono considerate, automaticamente, istanze della classe persona

Ereditarietà tra Classi (3)

- **Onto*DLP*** supporta l'ereditarietà multipla
- Una classe può essere la specializzazione di un numero qualsiasi di superclassi, ed ereditare, conseguentemente, tutti gli attributi di queste ultime
- Esempio

```
class studente_impiegato isa {studente, impiegato} (
   oreLavorative: integer).
```

- Tale classe è sottoclasse sia di studente che di impiegato *(anche di persona)*
- L'attributo ***tutor*** è definito sia in *studente* che in *impiegato* con tipo persona e impiegato. Questa situazione di conflitto è risolta applicando un semplice criterio. Il tipo dell'attributo "confliggente" è dato dall'**intersezione** *(nel senso dell'insieme delle istanze condivise)* tra i tipi dell'attributo nelle superclassi *(nell'esempio persona ed impiegato)*. Questa scelta è, ragionevolmente, sicura, infatti, essa garantisce che tutte le istanze della sottoclasse (in questo caso studente-impiegato), anche quando ereditate, verifichino gli schemi di classe
- La classe built-in `object` che è (automaticamente) superclasse di tutte le classi

Relazioni (1)

- Una relazione può essere dichiarata in **Onto*DLP*** utilizzando una sintassi simile a quella delle classi
- Esempio

```
relation amico(pers1: persona, pers2: persona).
relation haVissuto (per: persona, lu: luogo,
  periodo: string).
```

- Come nelle classi, l'insieme degli attributi di una relazione è chiamato schema
- La cardinalità dello schema è detta arità
- Lo schema di una relazione definisce la struttura delle sue istanze *(tuple)*

Relazioni (2)

- Esempio di tuple

```
haVissuto(per: john, lu: roma, periodo: "due anni").
```

- Le tuple di una relazione, dunque, così come avviene per le istanze di una classe, sono definite asserendo opportuni fatti logici
- Similmente alle istanze di classe, inoltre, è possibile dichiarare tuple utilizzando sia la notazione non posizionale sia quella posizionale
- Diversamente dalle istanze di classe, però, le tuple non hanno oid e non sono quindi direttamente referenziabili

Ereditarietà tra relazioni

- Come avviene per le classi è possibile rappresentare legami di specializzazione/generalizzazione e quindi definire delle tassonomie, anche in corrispondenza delle relazioni
- **Onto*DLP***, infatti, consente l'utilizzo della relazione binaria **`isa`** anche all'interno di una definizione di relazione
- Esempio

```
relation possiedeCodiceIdentificativo (
   pers: persona, codice: string).
relation possiedeCodiceFiscale isa
   {possiedeCodiceIdentificativo}.
relation possiedeNumeroDiPatente isa
   {possiedeCodiceIdentificativo}
   (dataDiRilascioDelDocumento: string,
   validità: string).
```

Assiomi e Consistenza (1)

- Spesso è necessario imporre dei vincoli, condizioni addizionali sulle proprietà degli individui
- Tali asserzioni, possono essere modellate in utilizzando gli assiomi
- Un assioma serve a verifica la consistenza, e consente di definire condizioni che devo sempre essere soddisfatte *(almeno se quanto specificato è corretto)* dagli oggetti del dominio rappresentato.
- Esempio

```
relation collega (imp1: impiegato, imp2: impiegato).
::- collega(imp1: X1, imp2: X2),
     not collega(imp1: X2, imp2 : X1)
::- collega(imp1: X1, imp2: X2), X1 : impiegato(azienda: A),
     not X2: impiegato(azienda: A).
```

- Il primo assioma richiede che la relazione collega sia simmetrica
- Il secondo richiede che se due persone (X1 e X2) sono colleghi, e X1 lavora per una azienda A allora anche X2 deve lavorare per A

Assiomi e Consistenza (2)

- Si noti che gli assiomi in **Onto*DLP*** non consentono di derivare nuova conoscenza, ma semplicemente impongono delle condizioni che devono essere sempre vere
- Di fatto essi si comportano come i *constraint* nella DLP *(differiscono solo per il simbolo ::- che è utilizzato al posto di :-)*
- Se un assioma è violato, la rappresentazione del dominio è inconsistente *(contiene informazioni che contraddittorie o non ammissibili rispetto alla modellazione che è stata fatta della realtà)*

Moduli di Ragionamento (1)

- Data una base di conoscenza, può essere molto utile ragionare sui dati in essa descritti. I moduli di ragionamento sono i componenti del linguaggio **Onto*DLP*** che consentono di effettuare potenti inferenze logiche sulle informazioni presenti nelle ontologie
- In pratica, un modulo di ragionamento è un programma logico disgiuntivo concepito in modo da ragionare sui dati presenti nel dominio rappresentato. Ogni modulo di ragionamento in **Onto*DLP*** è identificato da un nome ed è definito da un insieme di regole logiche (eventualmente disgiuntive) e di vincoli di integrità
- Sintatticamente, il nome del modulo è preceduto dalla parola chiave `module`, mentre le regole logiche sono racchiuse da parentesi graffe (in questo modo sono raccolte insieme, ed identificate da un nome, tutte le regole che codificano la soluzione di un dato problema)

Moduli di Ragionamento (2)

- I moduli di ragionamento, dunque, isolano un insieme di regole logiche e di vincoli tra loro correlati. Ciò garantisce la possibilità di sfruttare una forma di programmazione modulare in cui i programmi logici possono essere utilizzati come librerie. Inoltre, all'interno dei moduli di ragionamento è possibile specificare un insieme di predicati ausiliari che non sono corredati da uno schema specifico e che sono locali al modulo
- Esempio

```
module giovaniTimidi
{
    giovaneTimido (N) :- P: persona(nome: N, eta: A), A < 18,
        #count{F: amico(pers1: P, pers2: F)} < 10.
}
```

- Tale modulo consente di individuare tutte le persone che hanno meno di 18 anni e hanno meno di 10 amici