

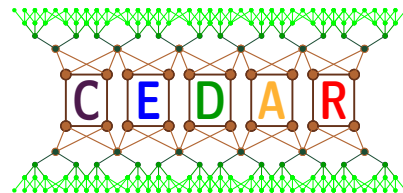
# Reasoning and the Semantic Web

Hassan Aït-Kaci

*ANR Chair of Excellence*

Université Claude Bernard Lyon 1

Constraint Event-Driven Automated Reasoning Project



## Outline

- ▶ **Constraint Logic Programming**
- ▶ **What is unification?**
- ▶ **Semantic Web objects**
- ▶ **Graphs as constraints**
- ▶ *OWL* and *DL*-based reasoning
- ▶ **Constraint-based Semantic Web reasoning**
- ▶ **Recapitulation**

## Outline

- ▶ **Constraint Logic Programming**
- ▶ What is unification?
- ▶ Semantic Web objects
- ▶ Graphs as constraints
- ▶ *OWL* and *DL*-based reasoning
- ▶ Constraint-based Semantic Web reasoning
- ▶ Recapitulation

## Constraint Logic Programming

---

In Prolog seen as a  $\mathcal{CLP}$  language, a clause such as:

```
append([],L,L).
```

```
append([H|T],L,[H|R]) :- append(T,L,R).
```

is construed as:

```
append(X1,X2,X3) :- true  
                    | X1 = [],      X2 = L, X3 = L.
```

```
append(X1,X2,X3) :- append(X4,X5,X6)  
                    | X1 = [H|T], X2 = L, X3 = [H|R],  
                    X4 = T,      X5 = L, X6 = R.
```

## Constraint Logic Programming Scheme

---

The *CLP scheme* requires a set  $\mathcal{R}$  of *relational symbols* (or, predicate symbols) and a *constraint language*  $\mathcal{L}$ .

The constraint language  $\mathcal{L}$  needs very little  
—*(not even syntax!)*:

- ▶ a *set*  $\mathcal{V}$  of *variables* (denoted as capitalized  $X, Y, \dots$ );
- ▶ a *set*  $\Phi$  of *formulae* (denoted  $\phi, \phi', \dots$ ) called *constraints*;
- ▶ a *function* **VAR**:  $\Phi \mapsto \mathcal{V}$ , giving for every constraint  $\phi$  the *set* **VAR**( $\phi$ ) of *variables constrained by*  $\phi$ ;
- ▶ a *family of interpretations*  $\mathcal{A}$  over some *domain*  $D^{\mathcal{A}}$ ;
- ▶ a *set* **VAL**( $\mathcal{A}$ ) of *valuations*—total functions  $\alpha : \mathcal{V} \mapsto D^{\mathcal{A}}$ .

## Constraint Logic Programming Language

---

Given a set of relational symbols  $\mathcal{R}$  ( $r, r_1, \dots$ ), a constraint language  $\mathcal{L}$  is extended into a language  $\mathcal{R}(\mathcal{L})$  of *constrained relational clauses* with:

- ▶ the set  $\mathcal{R}(\Phi)$  of formulae defined to include:
  - all formulae  $\phi$  in  $\Phi$ , *i.e.*, all  $\mathcal{L}$ -constraints;
  - all relational atoms  $r(X_1, \dots, X_n)$ ,  
where  $X_1, \dots, X_n \in \mathcal{V}$  are mutually distinct;

and closed under  $\&$  (conjunction) and  $\rightarrow$  (implication);

- ▶ extending an interpretation  $\mathcal{A}$  of  $\mathcal{L}$  by adding relations:  
 $r^{\mathcal{A}} \subseteq D^{\mathcal{A}} \times \dots \times D^{\mathcal{A}}$  for each  $r \in \mathcal{R}$ .

## Constraint Logic Programming Clause

---

We define a  $\mathcal{CLP}$  *constrained definite clause* in  $\mathcal{R}(\mathcal{L})$  as:

$$r(\vec{X}) \leftarrow r_1(\vec{X}_1) \ \& \ \dots \ \& \ r_m(\vec{X}_m) \ \parallel \ \phi,$$

where  $(0 \leq m)$  and:

- ▶  $r(\vec{X}), r_1(\vec{X}_1), \dots, r_m(\vec{X}_m)$  are relational atoms in  $\mathcal{R}(\mathcal{L})$ ; and,
- ▶  $\phi$  is a constraint formula in  $\mathcal{L}$ .

A *constrained resolvent* is a formula  $\varrho \ \parallel \ \phi$ , where  $\varrho$  is a (possibly empty) conjunction of relational atoms  $r(X_1, \dots, X_n)$ —its *relational part*—and  $\phi$  is a (possibly empty) conjunction of  $\mathcal{L}$ -constraints—its *constraint part*.

## Constraint Logic Programming Resolution

---

**Constrained *resolution*** is a reduction rule on resolvents that gives a sound and complete interpreter for *programs* consisting of a set  $\mathcal{C}$  of constrained definite  $\mathcal{R}(\mathcal{L})$ -clauses.

The reduction of a constrained *resolvent* of the form:

$$B_1 \ \& \ \dots \ \& \ r(X_1, \dots, X_n) \ \& \ \dots \ B_k \ \parallel \ \phi$$

by the (renamed) program clause:

$$r(X_1, \dots, X_n) \leftarrow A_1 \ \& \ \dots \ \& \ A_m \ \parallel \ \phi'$$

is the new constrained resolvent of the form:

$$B_1 \ \& \ \dots \ \& \ A_1 \ \& \ \dots \ \& \ A_m \ \& \ \dots \ B_k \ \parallel \ \phi \ \& \ \phi'.$$



## Why Constraints?

---

### Some important points:

- ▶ But... wait a minute: “*Constraints are logical formulae—so why not use only logic?*”

Indeed, constraints are logical formulae—and *that is good!*

But such formulae as factors in a conjunction *commute* with other factors, thus *freeing operational scheduling of resolvents*.

- ▶ A constraint is a formula *solvable by a specific solving algorithm* rather than general-purpose logic-programming machinery.
- ▶ Better: constraint solving *remembers proven facts* (proof memoizing).

**Such are key points exploited in *CLP!***

## Constraint Solving—Constraint Normalization

---

Constraint solving is conveniently specified using *constraint normalization rules*, which are semantics-preserving syntax-driven rewrite (meta-)rules.

Plotkin's SOS notation:

$$\begin{array}{c} (n) \text{ Rule Name} \\ \textit{Prior Form} \\ \hline \textit{Posterior Form} \end{array} \quad \text{if } \textit{Condition}$$

A normalization rule is said to be *correct* iff the prior form's denotation is equal to the posterior form's whenever the side condition holds.

## Constraint Normalization—Declarative Coroutining

---

Normalizing a constraint yields a **normal form**: a constraint formula that can't be transformed by any normalization rule.

Such may be either the **inconsistent constraint**  $\perp$ , or:

- ▶ a **solved form**—a normal form that can be immediately deemed consistent; or,
- ▶ a **residuated form**—a normal form but not a solved form.

A residuated constraint is a *suspended computation*; shared variables are **inter-process communication channels**: binding in one normalization process may trigger resumption of another residuated normalization process.

**Constraint residuation enables automatic coroutining!**

## Outline

- ▶ Constraint Logic Programming
- ▶ **What is unification?**
- ▶ Semantic Web objects
- ▶ Graphs as constraints
- ▶ *OWL* and *DL*-based reasoning
- ▶ Constraint-based Semantic Web reasoning
- ▶ Recapitulation

## What is unification?—First-order terms

---

The set  $\mathcal{T}_{\Sigma, \mathcal{V}}$  of *first-order terms* is defined given:

- ▶  $\mathcal{V}$  a countable set of *variables*;
- ▶  $\Sigma_n$  sets of *constructors* of arity  $n$  ( $n \geq 0$ );
- ▶  $\Sigma = \bigcup_{n \geq 0} \Sigma_n$  the constructor *signature*.

Then, a *first-order term* (FOT) is either:

- ▶ a variable in  $\mathcal{V}$ ; or,
- ▶ an element of  $\Sigma_0$ ; or,
- ▶ an expression of the form  $f(t_1, \dots, t_n)$ ,  
where  $n > 0$ ,  $f \in \Sigma_n$ , and  $t_i$  is a FOT, for all  $i \geq 1$ .

Examples of FOTs:  $X$     $a$     $f(g(X, a), Y, h(X))$

(variables are capitalized as in Prolog).

## What is unification?—Substitutions & instances

---

A variable substitution is a **map**  $\sigma : \mathcal{V} \rightarrow \mathcal{T}_{\Sigma, \mathcal{V}}$  such that the set  $\{X \in \mathcal{V} \mid \sigma(X) \neq X\}$  is finite.

Given a **substitution**  $\sigma$  and a **FOT**  $t$ , the  $\sigma$ -instance of  $t$  is the FOT:

$$t\sigma = \begin{cases} \sigma(X) & \text{if } t = X \in \mathcal{V}; \\ a & \text{if } t = a \in \Sigma_0; \\ f(t_1\sigma, \dots, t_n\sigma) & \text{if } t = f(t_1, \dots, t_n). \end{cases}$$

**Unification** is the **process of solving an equation** of the form:

$$t \doteq t'$$

## What is unification?—FOT equation solving

---

A **solution**, if one exists, is any substitution  $\sigma$  such that:

$$t\sigma = t'\sigma$$

If solutions exist, there is always a **minimal solution** (*the most general unifier*): **mgu**( $t, t'$ ).

where: “ $\sigma_1$  *is more general than*  $\sigma_2$ ” iff  $\exists\sigma$  s.t.  $\sigma_2 = \sigma_1\sigma$

**Equation and solution example:**

$$f(g(X, b), X, g(h(X), Y)) \doteq f(g(U, U), b, g(V, a))$$

$$X \doteq b, Y \doteq a, U \doteq b, V \doteq h(b)$$

## What is unification?—Algorithms

---

FOT unification algorithms have been (re-)invented:

- ▶ J. Herbrand (PhD thesis—page 148, **1930**)
- ▶ J.A. Robinson (JACM **1965**)
- ▶ A. Martelli & U. Montanari (ACM TOPLAS **1982**)

But, rather than a monolithic algorithm, FOT unification is simply expressible as a set of syntax-driven **commutative and terminating constraint normalization rules!**



## What is unification?—Constraint normalization rules

---

### (1) Substitute

$$\frac{\phi \ \& \ X \doteq t}{\phi[X/t] \ \& \ X \doteq t}$$

if  $X$  occurs in  $\phi$

### (2) Decompose

$$\frac{\phi \ \& \ f(s_1, \dots, s_n) \doteq f(t_1, \dots, t_n)}{\phi \ \& \ s_1 \doteq t_1 \ \& \ \dots \ \& \ s_n \doteq t_n}$$

if  $f \in \Sigma_n$ , ( $n \geq 0$ )

### (3) Fail

$$\frac{\phi \ \& \ f(s_1, \dots, s_n) \doteq g(t_1, \dots, t_m)}{\perp}$$

if  $f \in \Sigma_n$ , ( $n \geq 0$ )  
and  $g \in \Sigma_m$ , ( $m \geq 0$ )  
and  $m \neq n$

## What is unification?—Constraint normalization rules

---

(4) Flip

$$\frac{\phi \ \& \ t \doteq X}{\phi \ \& \ X \doteq t} \quad \begin{array}{l} \text{if } X \in \mathcal{V} \\ \text{and } t \notin \mathcal{V} \end{array}$$

(5) Erase

$$\frac{\phi \ \& \ t \doteq t}{\phi} \quad \text{if } t \in \Sigma_0 \cup \mathcal{V}$$

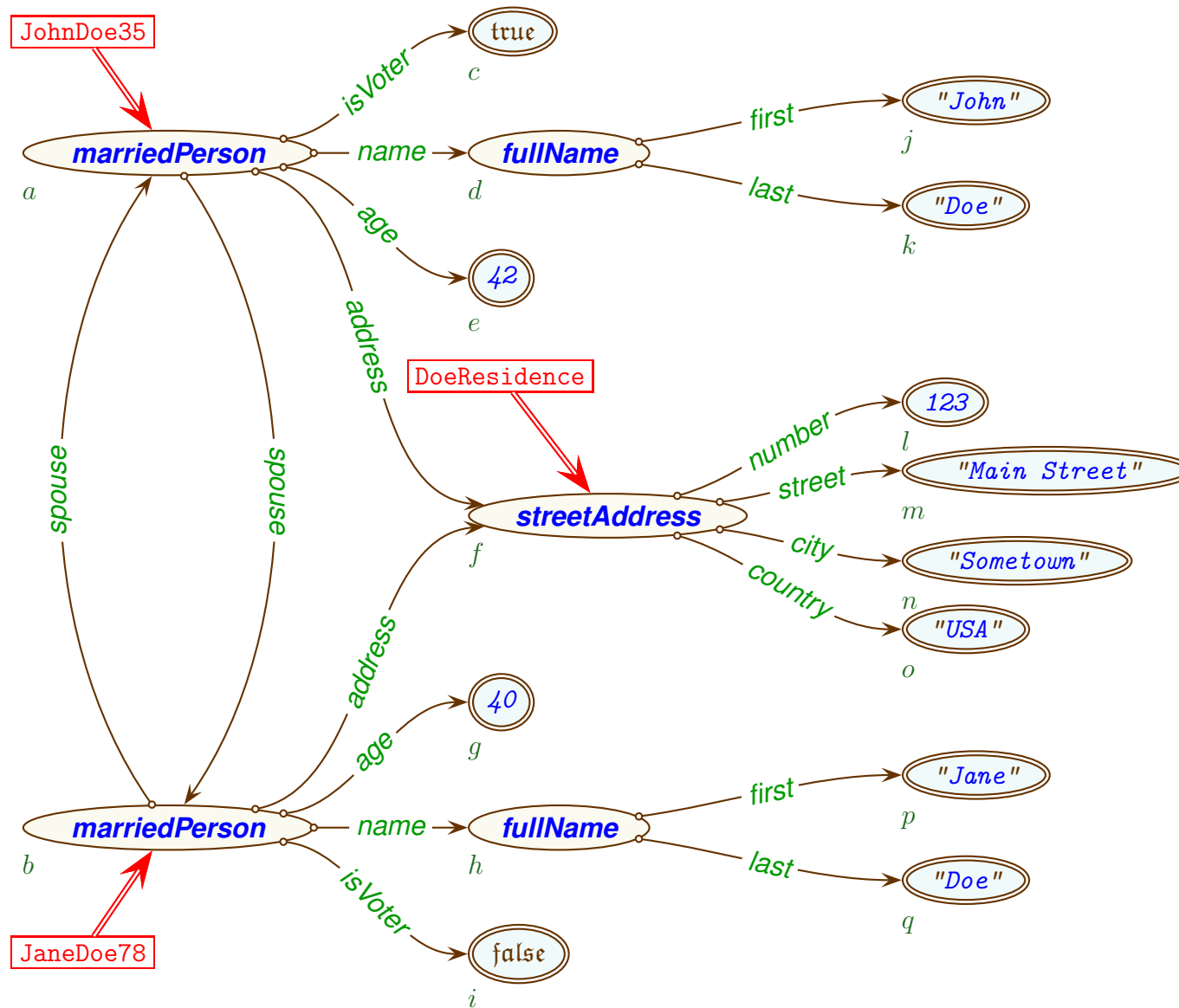
(6) Cycle

$$\frac{\phi \ \& \ X \doteq t}{\perp} \quad \begin{array}{l} \text{if } X \in \mathcal{V} \\ \text{and } t \notin \mathcal{V} \\ \text{and } X \text{ occurs in } t \end{array}$$

## Outline

- ▶ Constraint Logic Programming
- ▶ What is unification?
- ▶ **Semantic Web objects**
- ▶ Graphs as constraints
- ▶ *OWL* and *DL*-based reasoning
- ▶ Constraint-based Semantic Web reasoning
- ▶ Recapitulation

# Semantic Web objects—Objects are labelled graphs!



## Semantic Web objects—*Objects are labelled graphs!*

---

```
JohnDoe35 : marriedPerson ( name    => fullName
                             ( first => "John"
                               , last  => "Doe" )
                             , age    => 42
                             , address => DoeResidence
                             , spouse => JaneDoe78
                             , isVoter => true
                             )
```

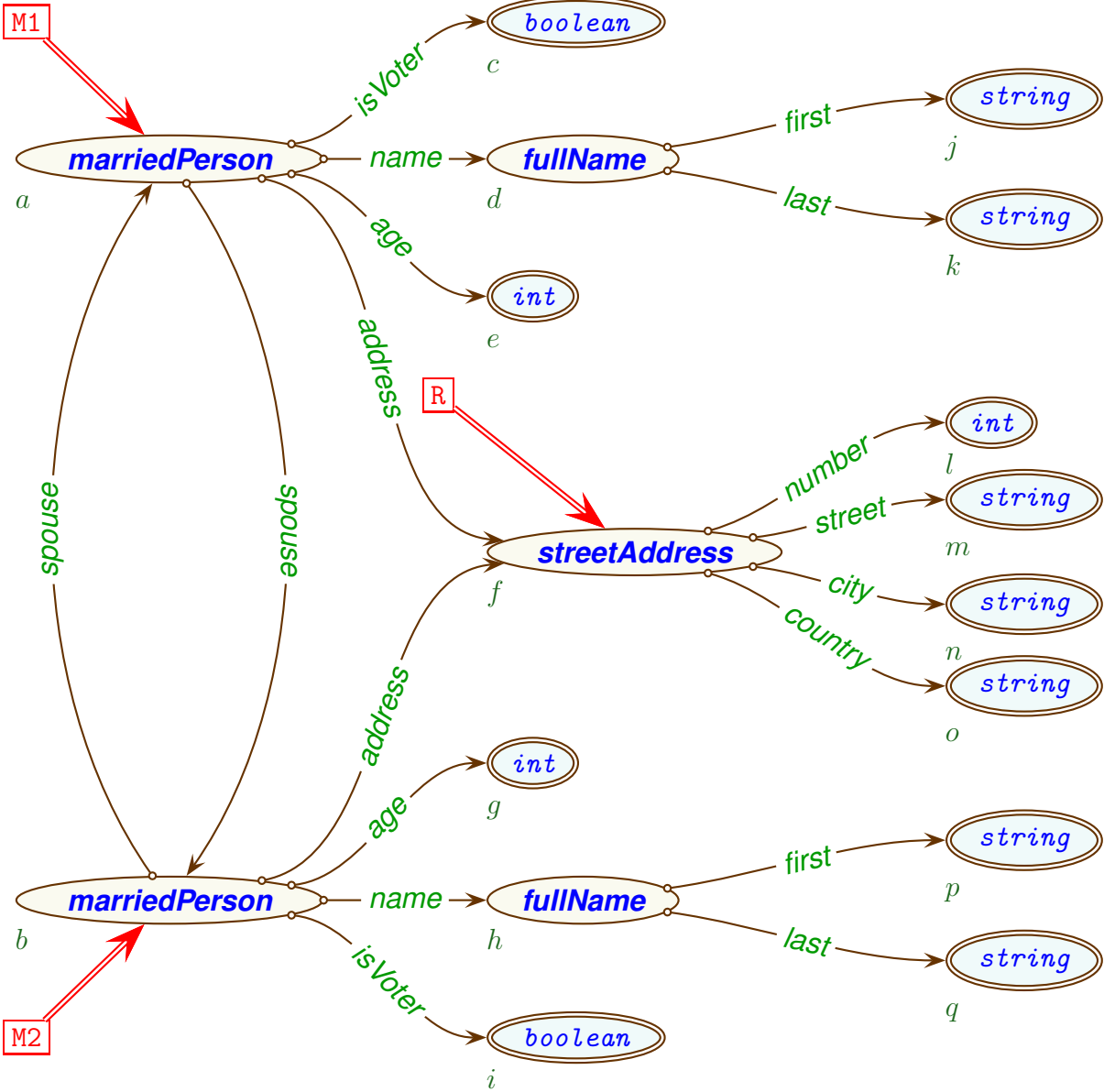
## Semantic Web objects—*Objects are labelled graphs!*

---

```
JaneDoe78 : marriedPerson ( name      => fullName
                             ( first => "Jane"
                               , last  => "Doe" )
                             , age     => 40
                             , address => DoeResidence
                             , spouse  => JohnDoe35
                             , isVoter => false
                             )
```

```
DoeResidence : streetAddress ( number => 123
                               , street => "Main Street"
                               , city   => "Sometown"
                               , country => "USA"
                               )
```

# Semantic Web types—Types are labelled graphs!



## Semantic Web types—Types are labelled graphs!

---

```
M1 : marriedPerson ( name    => fullName
                    ( first => string
                      , last  => string )
                    , age     => int
                    , address => R
                    , spouse  => M2
                    , isVoter => boolean
                    )
```



## Semantic Web formalisms—*Types are labelled graphs!*

---

```
M2 : marriedPerson ( name    => string
                    ( first => string
                      , last => string )
                    , age     => int
                    , address => R
                    , spouse  => M1
                    , isVoter => boolean
                    )
```

```
R : streetAddress ( number => int
                  , street => string
                  , city   => string
                  , country => string
                  )
```

## Outline

- ▶ Constraint Logic Programming
- ▶ What is unification?
- ▶ Semantic Web objects
- ▶ **Graphs as constraints**
- ▶ *OWL* and *DL*-based reasoning
- ▶ Constraint-based Semantic Web reasoning
- ▶ Recapitulation

# Original motivation: *Formalize this?*—ca. 1982

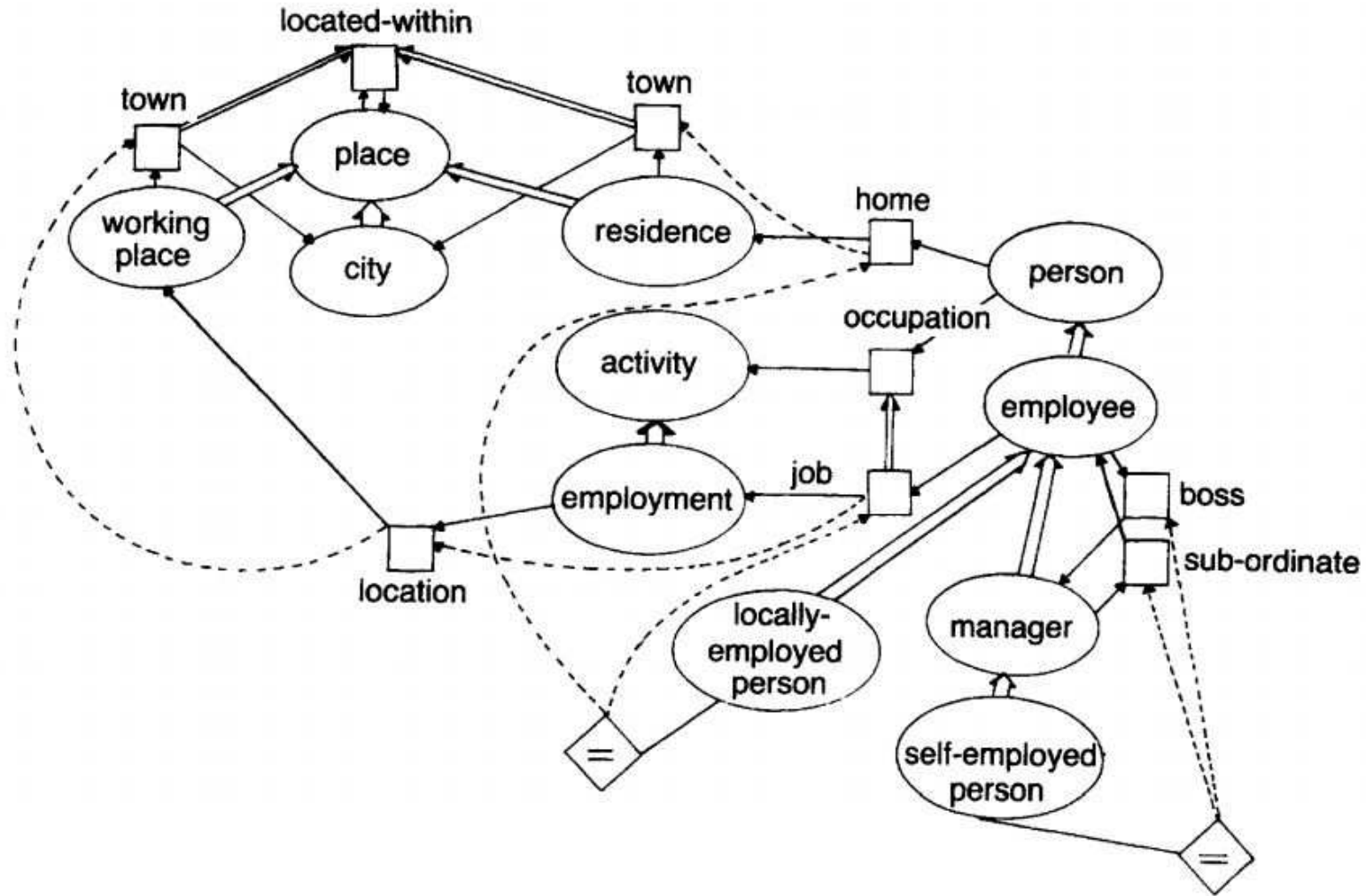


Fig. 1. Example of a KL-ONE semantic network.

## Graphs as constraints—*Motivation*

---

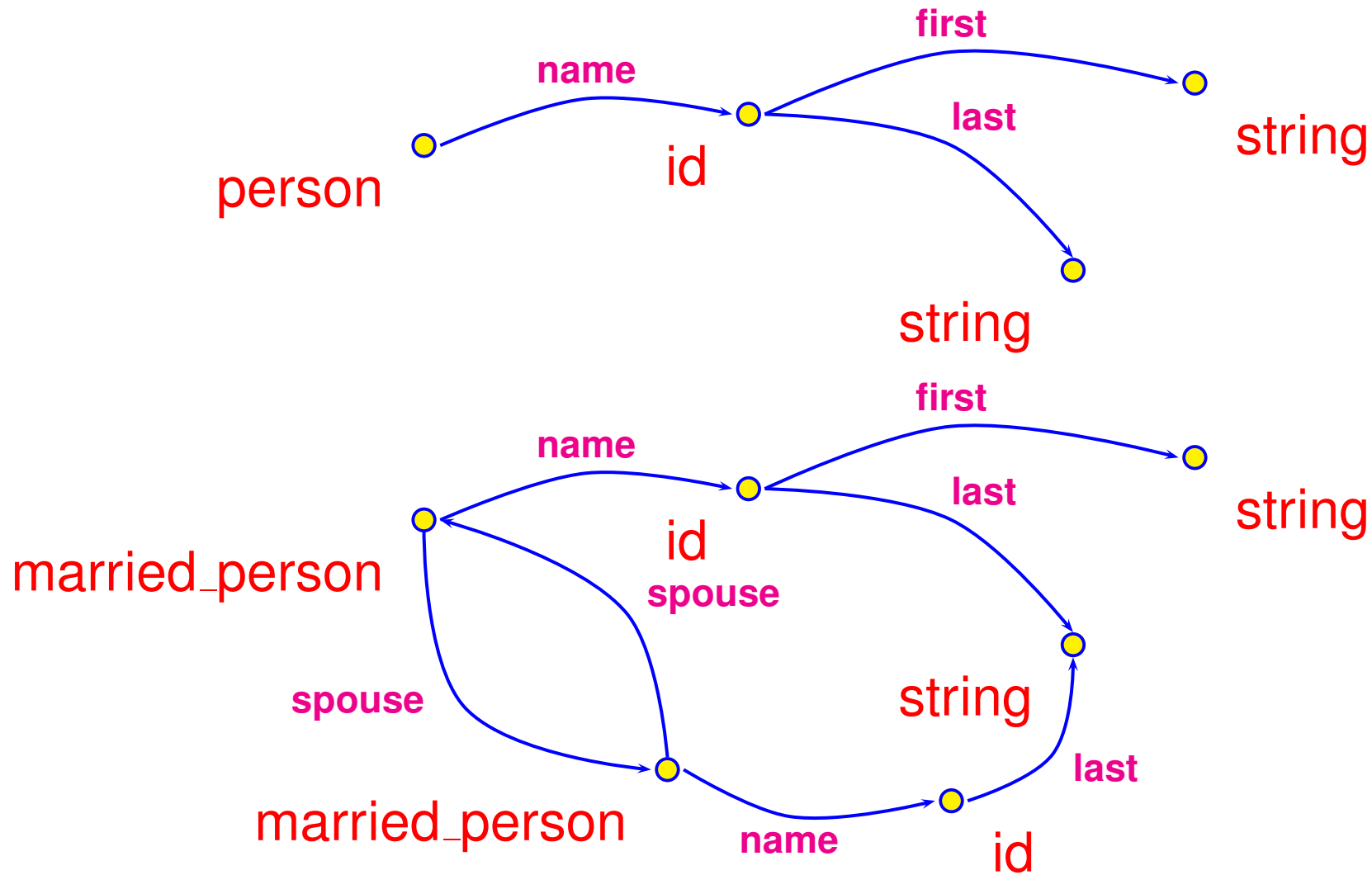
- ▶ **What:** a formalism for representing objects that is:
  - intuitive** (objects as labelled graphs), **expressive** (“real-life” data models), **formal** (logical semantics), **operational** (executable), & **efficient** (constraint-solving)
- ▶ **Why?** *viz.*, ubiquitous use of labelled graphs to structure information **naturally** as in:
  - object-orientation, knowledge representation,
  - databases, semi-structured data,
  - natural language processing, graphical interfaces,
  - concurrency and communication,
  - XML, RDF, the “Semantic Web,” *etc.*, ...

Viewing graphs as **constraints** stems from the work of:

- ▶ Hassan Aït-Kaci (since 1983)
- ▶ Gert Smolka (since 1986)
- ▶ Andreas Podelski (since 1989)
- ▶ Franz Baader, Rolf Backhofen, Jochen Dörre, Martin Emele, Bernhard Nebel, Joachim Niehren, Ralf Treinen, Manfred Schmidt-Schauß, Remi Zajac, . . .

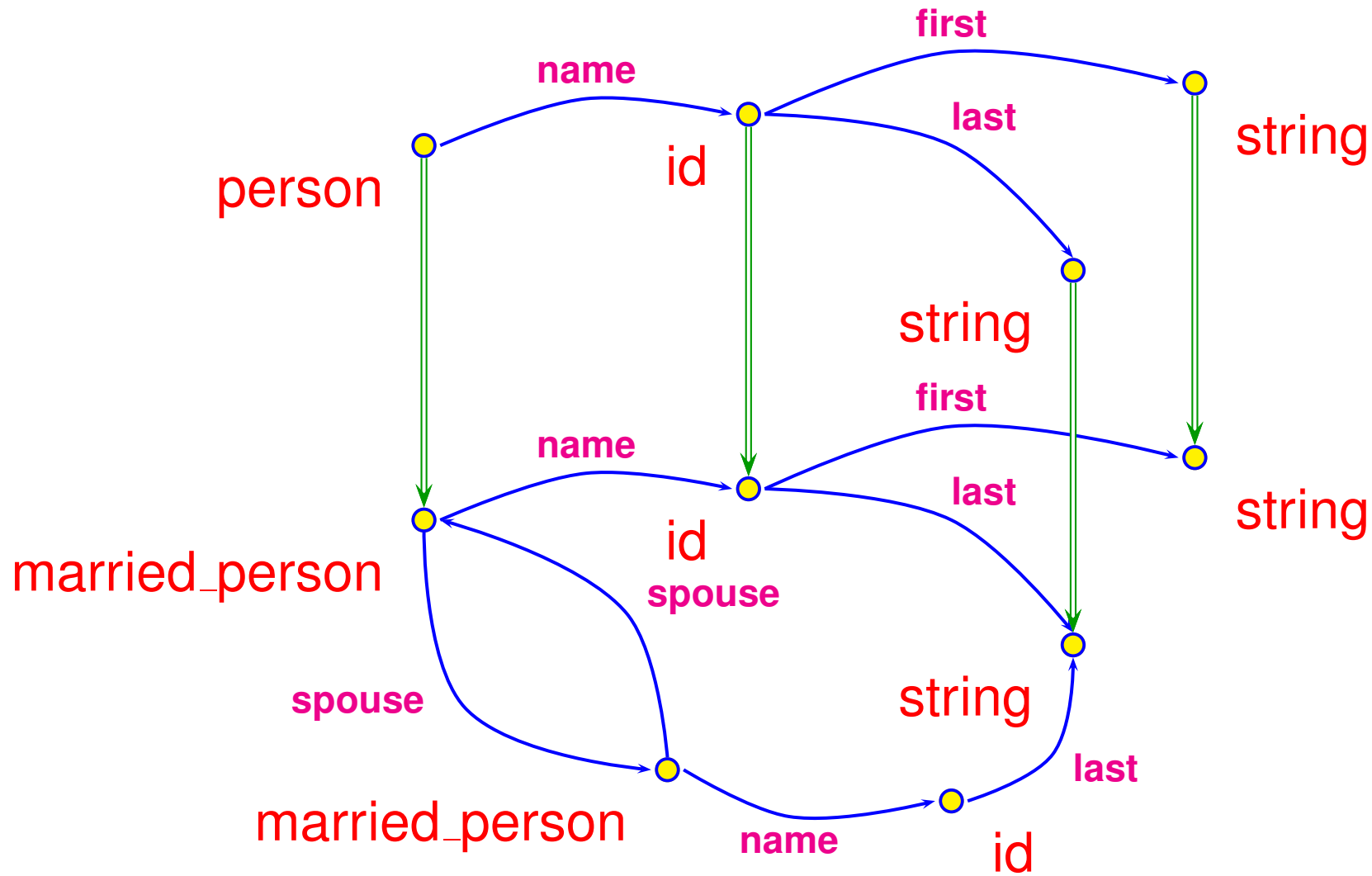
# Graphs as constraints—*Inheritance as graph endomorphism*

---



# Graphs as constraints—*Inheritance as graph endomorphism*

---



## Graphs as constraints—*OSF term syntax*

---

Let  $\mathcal{V}$  be a countable set of **variables**, and  $\mathcal{S}$  a lattice of **sorts**.

An **OSF term** is an expression of the form:

$$X : s(l_1 \Rightarrow t_1, \dots, l_n \Rightarrow t_n)$$

where:

- ▶  $X \in \mathcal{V}$  is the **root variable**
- ▶  $s \in \mathcal{S}$  is the **root sort**
- ▶  $n \geq 0$  (if  $n = 0$ , we write  $X : s$ )
- ▶  $\{l_1, \dots, l_n\} \subseteq \mathcal{F}$  are features
- ▶  $t_1, \dots, t_n$  are *OSF* terms



## Graphs as constraints—*OSF* term syntax example

---

$$\begin{aligned} X : & \text{person}(\text{name} \Rightarrow N : \top(\text{first} \Rightarrow F : \text{string}), \\ & \text{name} \Rightarrow M : \text{id}(\text{last} \Rightarrow S : \text{string}), \\ & \text{spouse} \Rightarrow P : \text{person}(\text{name} \Rightarrow I : \text{id}(\text{last} \Rightarrow S : \top), \\ & \text{spouse} \Rightarrow X : \top)). \end{aligned}$$

Lighter notation (showing only shared variables):

$$\begin{aligned} X : & \text{person}(\text{name} \Rightarrow \top(\text{first} \Rightarrow \text{string}), \\ & \text{name} \Rightarrow \text{id}(\text{last} \Rightarrow S : \text{string}), \\ & \text{spouse} \Rightarrow \text{person}(\text{name} \Rightarrow \text{id}(\text{last} \Rightarrow S), \\ & \text{spouse} \Rightarrow X)). \end{aligned}$$

## Graphs as constraints— $\mathcal{OSF}$ clause syntax

---

An  $\mathcal{OSF}$  constraint is one of:

- ▶  $X : s$
- ▶  $X.l \doteq X'$
- ▶  $X \doteq X'$

where  $X$  ( $X'$ ) is a **variable** (*i.e.*, a **node**),  $s$  is a **sort** (*i.e.*, a **node's type**), and  $l$  is a **feature** (*i.e.*, an **arc**).

An  $\mathcal{OSF}$  clause is a **conjunction** of  $\mathcal{OSF}$  constraints—*i.e.*, a **set of  $\mathcal{OSF}$  constraints**

$$\phi_1 \ \& \ \dots \ \& \ \phi_n$$

## Graphs as constraints—From $\mathcal{OSF}$ terms to $\mathcal{OSF}$ clauses

---

An  $\mathcal{OSF}$  term  $t = X : s(l_1 \Rightarrow t_1, \dots, l_n \Rightarrow t_n)$  is **dissolved** into an  $\mathcal{OSF}$  clause  $\phi(t)$  as follows:

$$\phi(t) \stackrel{\text{DEF}}{=} X : s \quad \& \quad X.l_1 \doteq X_1 \quad \& \quad \dots \quad \& \quad X.l_n \doteq X_n \\ \quad \quad \quad \& \quad \phi(t_1) \quad \quad \quad \& \quad \dots \quad \quad \quad \& \quad \phi(t_n)$$

where  $X_1, \dots, X_n$  are the root variables of  $t_1, \dots, t_n$ .

## Graphs as constraints—Example of $\mathcal{OSF}$ term dissolution

---

$$t = X : \text{person}(\text{name} \Rightarrow N : \top(\text{first} \Rightarrow F : \text{string}), \\ \text{name} \Rightarrow M : \text{id}(\text{last} \Rightarrow S : \text{string}), \\ \text{spouse} \Rightarrow P : \text{person}(\text{name} \Rightarrow I : \text{id}(\text{last} \Rightarrow S : \top), \\ \text{spouse} \Rightarrow X : \top))$$
$$\begin{aligned} \varphi(t) = & X : \text{person} \quad \& \quad X.\text{name} \doteq N \quad \& \quad N : \top \\ & \quad \& \quad X.\text{name} \doteq M \quad \& \quad M : \text{id} \\ & \quad \& \quad X.\text{spouse} \doteq P \quad \& \quad P : \text{person} \\ & \quad \& \quad N.\text{first} \doteq F \quad \& \quad F : \text{string} \\ & \quad \& \quad M.\text{last} \doteq S \quad \& \quad S : \text{string} \\ & \quad \& \quad P.\text{name} \doteq I \quad \& \quad I : \text{id} \\ & \quad \& \quad I.\text{last} \doteq S \quad \& \quad S : \top \\ & \quad \& \quad P.\text{spouse} \doteq X \quad \& \quad X : \top \end{aligned}$$

## Graphs as constraints—Basic OSF term normalization

---

### (1) Sort Intersection

$$\frac{\phi \ \& \ X : s \ \& \ X : s'}{\phi \ \& \ X : s \wedge s'}$$

### (2) Inconsistent Sort

$$\frac{\phi \ \& \ X : \perp}{X : \perp}$$

### (3) Variable Elimination

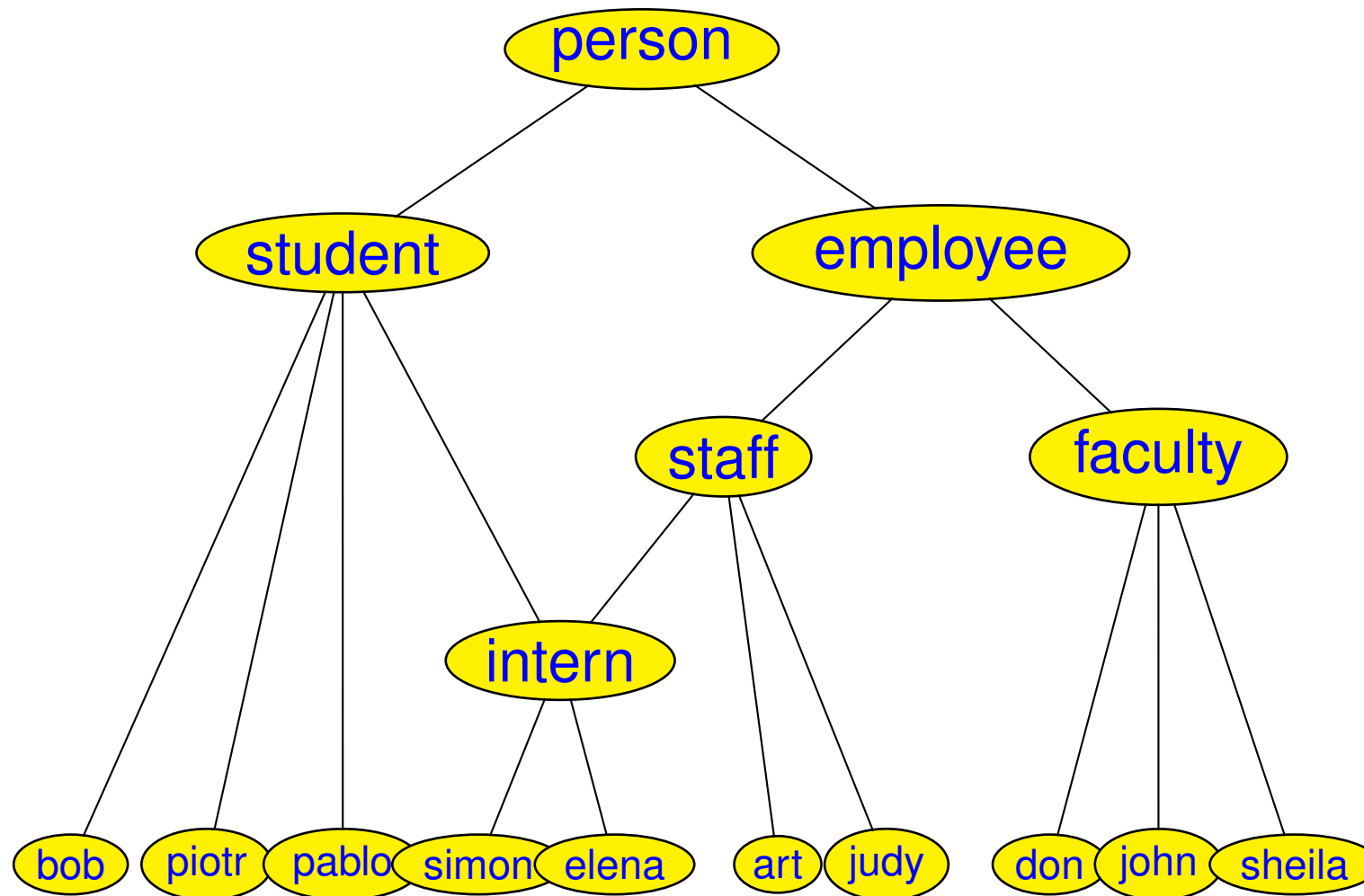
$$\frac{\phi \ \& \ X \doteq X'}{\phi[X'/X] \ \& \ X \doteq X'} \quad \text{if } X \neq X' \text{ and } X \in \mathbf{Var}(\phi)$$

### (4) Feature Functionality

$$\frac{\phi \ \& \ X.l \doteq X' \ \& \ X.l \doteq X''}{\phi \ \& \ X.l \doteq X' \ \& \ X' \doteq X''}$$

## Graphs as constraints—*OSF* unification as *OSF* constraint normalization

---



## Graphs as constraints—*OSF unification as OSF constraint normalization*

---

X : student

(roommate => person(rep => E : employee),  
advisor => don(secretary => E))

&

Y : employee

(advisor => don(assistant => A),  
roommate => S : student(rep => S),  
helper => simon(spouse => A))

&

X = Y

## Graphs as constraints—*OSF unification as OSF constraint normalization*

---

```
X : intern
  (roommate => S : intern(rep => S),
   advisor => don(assistant => A,
                 secretary => S),
   helper => simon(spouse => A))
```

&

```
X = Y
```

&

```
E = S
```



## Graphs as constraints—*Extended OSF terms*

---

Basic *OSF* terms may be extended to express:

- ▶ Non-lattice sort signatures
- ▶ Disjunction
- ▶ Negation
- ▶ Partial features
- ▶ Extensional sorts (*i.e.*, denoting elements)
- ▶ Relational features (*a.k.a.*, “roles”)
- ▶ Aggregates (*à la* monoid comprehensions)
- ▶ Regular-expression feature paths
- ▶ Sort definitions (*a.k.a.*, “*OSF* theories”—“ontologies”)

## Order-sorted featured graph constraints—(Summary)

---

We have overviewed a formalism of objects where:

- ▶ “real-life” objects are viewed as logical constraints
- ▶ objects may be approximated as set-denoting constructs
- ▶ object normalization rules provide an efficient operational semantics
- ▶ consistency extends unification (and thus matching)
- ▶ this enables rule-based computation (whether rewrite or logical rules) over general graph-based objects
- ▶ this yield a powerful means for effectively using ontologies

## Outline

- ▶ Constraint Logic Programming
- ▶ What is unification?
- ▶ Semantic Web objects
- ▶ Graphs as constraints
- ▶ ***OWL* and *DL*-based reasoning**
- ▶ Constraint-based Semantic Web reasoning
- ▶ Recapitulation

## Semantic Web formalisms—*OWL speaks*

---

*What language(s) do OWL's speak?—a confusing growing crowd of strange-sounding languages and logics:*

- *OWL, OWL Lite, OWL DL, OWL Full*
- *DL, DLR, ...*
- *AL, ALC, ALCN, ALCNR, ...*
- *SHIF, SHIN, CIQ, SHIQ, SHOQ(D), SHOIQ, SRIQ, SROIQ, ...*

*Depending on whether the system allows:*

- concepts, roles (inversion, composition, inclusion, ...)
- individuals, datatypes, cardinality constraints
- various combination thereof

## Semantic Web formalisms— $\mathcal{DL}$ dialects

---

For better or worse, the W3C has married its efforts to  $\mathcal{DL}$ -based reasoning systems:

- ▶ All the proposed  $\mathcal{DL}$  Knowledge Base formalisms in the *OWL* family use *tableaux-based methods* for reasoning
- ▶ *Tableaux methods work* by building models explicitly *via* formula expansion rules
- ▶ This limits  $\mathcal{DL}$  reasoning to finite (*i.e.*, decidable) models
- ▶ Worse, *tableaux methods* only work for small ontologies: they *fail to scale up* to large ontologies

# Semantic Web formalisms— $\mathcal{DL}$ dialects

## Tableaux style $\mathcal{DL}$ reasoning ( $\mathcal{ALCN}\mathcal{R}$ )

### CONJUNCTIVE CONCEPT:

$$\left[ \begin{array}{l} \text{if } x : (C_1 \sqcap C_2) \in S \\ \text{and } \{x : C_1, x : C_2\} \not\subseteq S \end{array} \right] \frac{S}{S \cup \{x : C_1, x : C_2\}}$$

### EXISTENTIAL ROLE:

$$\left[ \begin{array}{l} \text{if } x : (\exists R.C) \in S \text{ s.t. } R \stackrel{\text{DEF}}{=} (\prod_{i=1}^m R_i) \\ \text{and } z : C \in S \Rightarrow z \notin R_S[x] \\ \text{and } y \text{ is new} \end{array} \right] \frac{S}{S \cup \{x R_i y\}_{i=1}^m \cup \{y : C\}}$$

### DISJUNCTIVE CONCEPT:

$$\left[ \begin{array}{l} \text{if } x : (C_1 \sqcup C_2) \in S \\ \text{and } x : C_i \notin S \ (i = 1, 2) \end{array} \right] \frac{S}{S \cup \{x : C_i\}}$$

### MIN CARDINALITY:

$$\left[ \begin{array}{l} \text{if } x : (\geq n.R) \in S \text{ s.t. } R \stackrel{\text{DEF}}{=} (\prod_{i=1}^m R_i) \\ \text{and } |R_S[x]| \neq n \\ \text{and } y_i \text{ is new } (0 \leq i \leq n) \end{array} \right] \frac{S}{S \cup \{x R_i y_j\}_{i,j=1,1}^{m,n} \cup \{y_i \neq y_j\}_{1 \leq i < j \leq n}}$$

### UNIVERSAL ROLE:

$$\left[ \begin{array}{l} \text{if } x : (\forall R.C) \in S \\ \text{and } y \in R_S[x] \\ \text{and } y : C \notin S \end{array} \right] \frac{S}{S \cup \{y : C\}}$$

### MAX CARDINALITY:

$$\left[ \begin{array}{l} \text{if } x : (\leq n.R) \in S \\ \text{and } |R_S[x]| > n \text{ and } y, z \in R_S[x] \\ \text{and } y \neq z \notin S \end{array} \right] \frac{S}{S \cup S[y/z]}$$

## Understanding *OWL speak*—*OSF* vs. *DL*

---

Understanding *OWL* amounts to reasoning with knowledge expressed as *OWL sentences*. Its *DL* semantics relies on *explicitly* building models using *induction*.

*ergo:*

Inductive techniques are *eager* and (thus) *wasteful*

Reasoning with knowledge expressed as constrained (*OSF*) *graphs* relies on *implicitly* pruning inconsistent elements using *coinduction*.

*ergo:*

Coinductive techniques are *lazy* and (thus) *thrifty*

## Outline

- ▶ Constraint Logic Programming
- ▶ What is unification?
- ▶ Semantic Web objects
- ▶ Graphs as constraints
- ▶ *OWL* and *DL*-based reasoning
- ▶ **Constraint-based Semantic Web reasoning**
- ▶ Recapitulation



## *LIFE*—Rules + constraints for Semantic Web reasoning

---

*LIFE*—*L*ogic, *I*nheritance, *F*unctions, and *E*quations

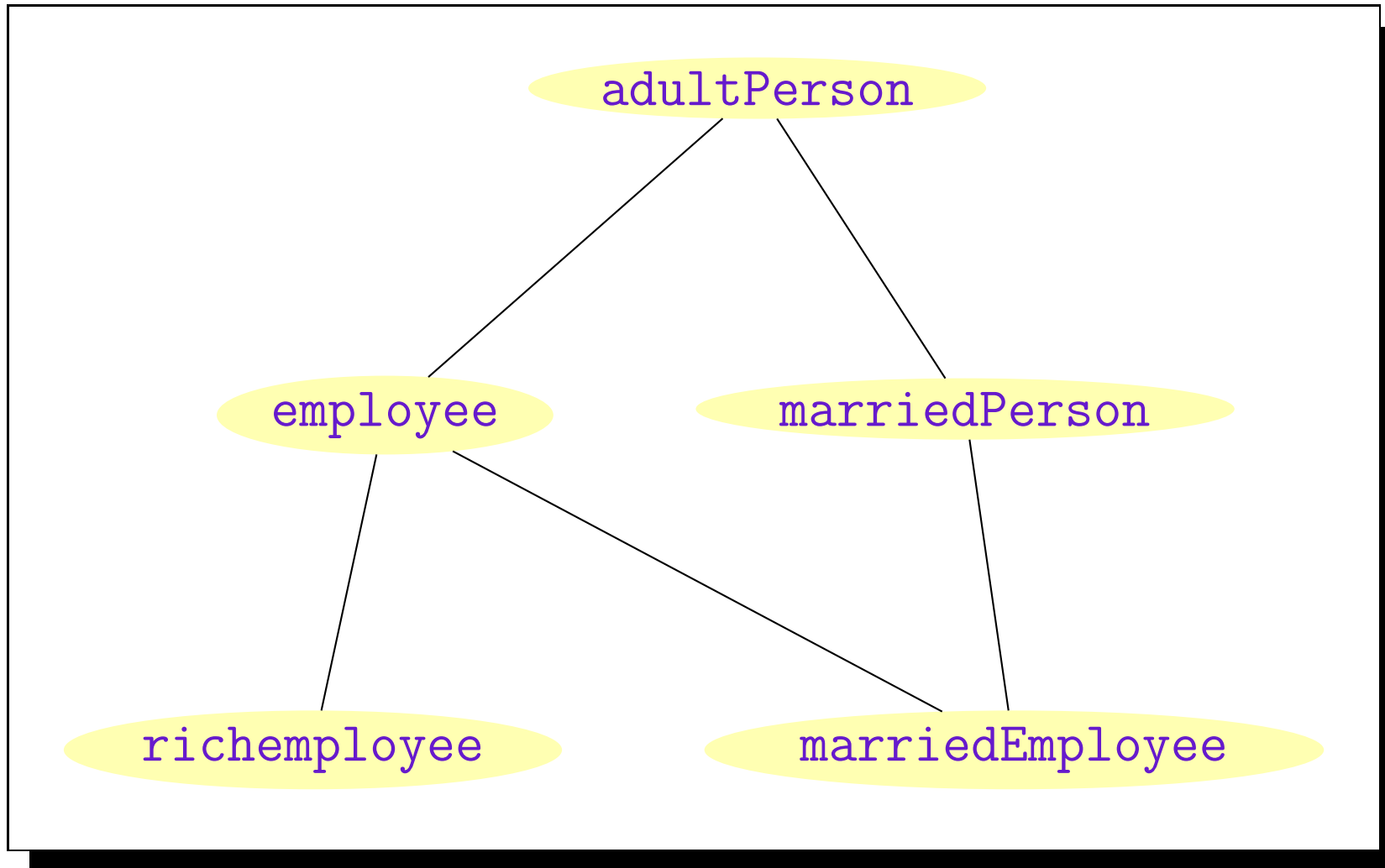
*CLP*( $\chi$ )—*C*onstraint, *L*ogic, *P*rogramming, parameterized over  
is a constraint system  $\chi$

*LIFE* is a *CLP* system over *OSF* constraints and functions  
over them (rewrite rules); namely:

$$LIFE = CLP(OSF + FP)$$

## *LIFE*—Rules + constraints for Semantic Web reasoning

---



**A multiple-inheritance hierarchy**

## The same hierarchy in Java

---

```
interface adultPerson {
    name id;
    date dob;
    int age;
    String ssn;
}
interface employee extends adultPerson {
    title position;
    String institution;
    employee supervisor;
    int salary;
}
interface marriedPerson extends adultPerson {
    marriedPerson spouse;
}
interface marriedEmployee extends employee, marriedPerson {
}
interface richEmployee extends employee {
}
```

## The same hierarchy in *LIFE*

---

```
employee <: adultPerson.  
marriedPerson <: adultPerson.  
richEmployee <: employee.  
marriedEmployee <: employee.  
marriedEmployee <: marriedPerson.  
  
:: adultPerson ( id ⇒ name  
                , dob ⇒ date  
                , age ⇒ int  
                , ssn ⇒ string ).  
  
:: employee ( position ⇒ title  
              , institution ⇒ string  
              , supervisor ⇒ employee  
              , salary ⇒ int ).  
  
:: marriedPerson ( spouse ⇒ marriedPerson ).
```

## A relationally and functionally constrained *LIFE* sort hierarchy

---

```
:: P : adultPerson ( id ⇒ name  
                    , dob ⇒ date  
                    , age ⇒ A : int  
                    , ssn ⇒ string )
```

```
| A = ageInYears(P), A ≥ 18.
```

---

```
:: employee      ( position ⇒ T : title  
                  , institution ⇒ string  
                  , supervisor ⇒ E : employee  
                  , salary ⇒ S : int )
```

```
| higherRank(E.position, T) , E.salary ≥ S.
```

## A relationally and functionally constrained *LIFE* sort hierarchy

---

```
:: M : marriedPerson ( spouse  $\Rightarrow$  P : marriedPerson )  
| P.spouse = M.
```

---

```
:: R : richEmployee ( institution  $\Rightarrow$  I  
                    , salary  $\Rightarrow$  S )  
| stockValue(I) = V , hasShares(R, I, N) , S + N * V  $\geq$  200000.
```

## Proof “memoizing”

---

*OSF* constraints as syntactic variants of logical formulae:

Sorts are unary predicates:  $X : s \iff [s]([X])$

Features are unary functions:  $X.f \doteq Y \iff [f]([X]) = [Y]$

Coreferences are equations:  $X \doteq Y \iff [X] = [Y]$

So ...

***Why not use (good old) logic proofs instead?***

## Proof “memoizing”

---

But: **model equivalence  $\neq$  proof equivalence!**

- ▶ *OSF*-unification proves sort constraints by reducing them monotonically w.r.t. the sort ordering
- ▶ *ergo*, once  $X : s$  has been proven, the proof of  $s(X)$  is recorded as *the sort “s” itself!*
- ▶ if further down a proof, it is again needed to prove  $X : s$ , it is remembered as *X’s binding*
- ▶ **Indeed, *OSF constraint proof rules ensure that:***

***no type constraint is ever proved twice***



## Proof “memoizing”

---

*OSF* type constraints are incrementally “*memoized*” as they are verified:

***sorts act as (instantaneous!) proof caches!***

whereas in logic having proven  $s(X)$  is not “*remembered*” in any way (e.g., Prolog)

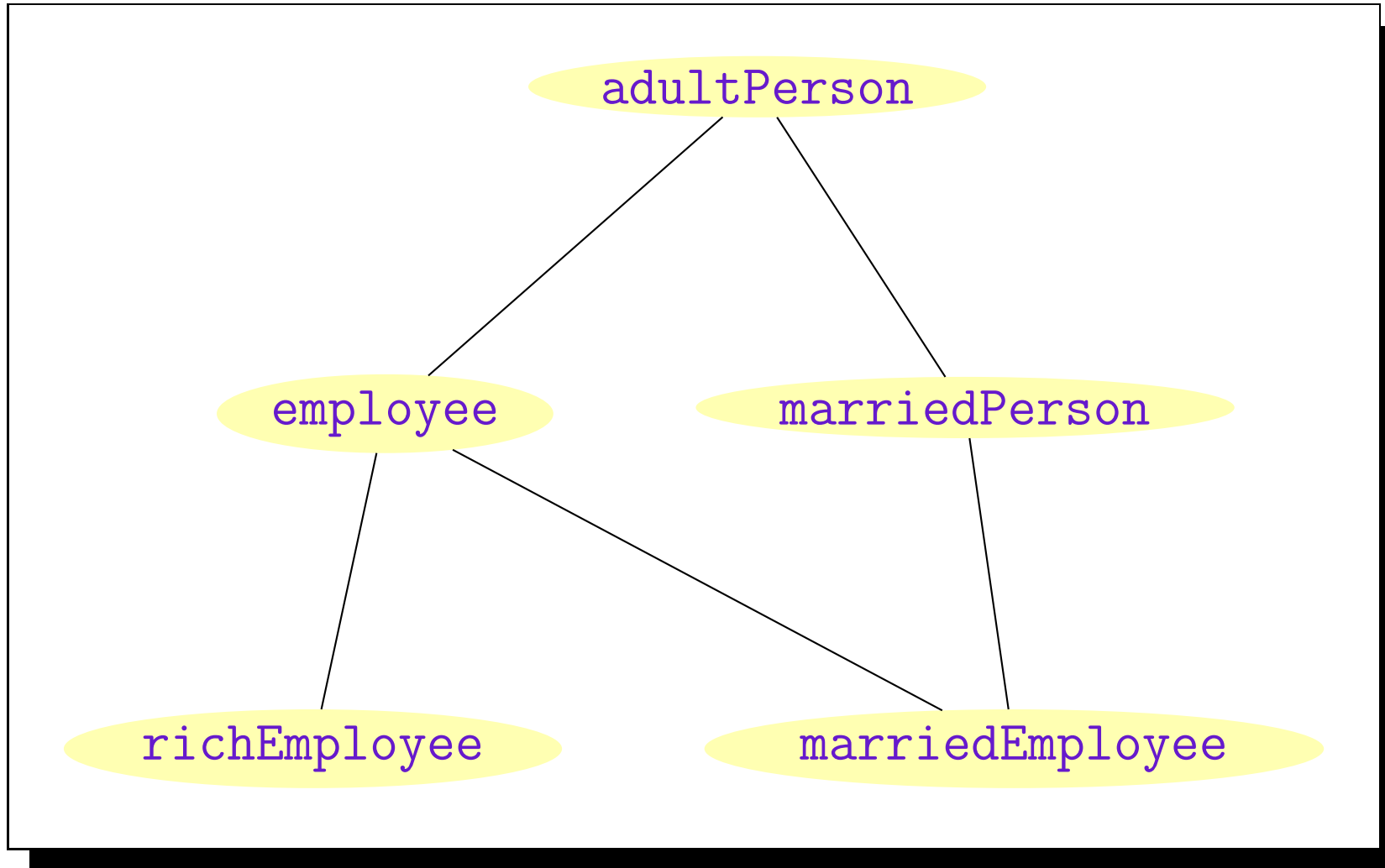
**Example:** consider the *OSF* constraint conjunction:

- $X : \text{adultPerson}(\text{age} \Rightarrow 25)$ ,
- $X : \text{employee}$ ,
- $X : \text{marriedPerson}(\text{spouse} \Rightarrow Y)$ .

**Notation:**  $\text{type}\#(\text{condition})$  means “*constraint condition attached to sort type*”

## Proof “memoizing”—Example hierarchy reminded

---



## Proof “memoizing”

---

1. proving:  $X : \text{adultPerson}(\text{age} \Rightarrow 25) \dots$
2. proving:  $\text{adultPerson}\#(X.\text{age} \geq 18) \dots$
3. proving:  $X : \text{employee} \dots$
4. proving:  $\text{employee}\#(\text{higherRank}(E.\text{position}, P)) \dots$
5. proving:  $\text{employee}\#(E.\text{salary} \geq S) \dots$
6. proving:  $X : \text{marriedPerson}(\text{spouse} \Rightarrow Y) \dots$
7. proving:  $X : \text{marriedEmployee}(\text{spouse} \Rightarrow Y) \dots$
8. proving:  $\text{marriedEmployee}\#(Y.\text{spouse} = X) \dots$

*Therefore, all other inherited conditions coming from a sort greater than marriedEmployee (such as employee or adultPerson) can be safely ignored!*

## Proof “memoizing”

---

This “*memoizing*” property of *OSF* constraint-solving enables:

***using rules over ontologies***

as well as, *conversely*,

***enhancing ontologies with rules***

Indeed, with *OSF*:

- ▶ ***concept ontologies may be used as constraints by rules*** for inference and computation
- ▶ ***rule-based conditions in concept definitions may be used*** to magnify expressivity of ontologies thanks to the ***proof-memoizing property of ordered sorts***

## Outline

- ▶ Constraint Logic Programming
- ▶ What is unification?
- ▶ Semantic Web objects
- ▶ Graphs as constraints
- ▶ *OWL* and *DL*-based reasoning
- ▶ Constraint-based Semantic Web reasoning
- ▶ **Recapitulation**

## Recapitulation—*what you must remember from this talk...*

---

- ▶ Objects are **graphs**
  - ▶ Graphs are **constraints**
  - ▶ Constraints are **good**: they provide both **formal** theory and **efficient** processing
  - ▶ **Formal Logic** is **not** all there is
  - ▶ even so: **model** theory  $\neq$  **proof** theory
  - ▶ indeed, due to its youth, much of W3C technology is often **naïve** in conception and design
- Ergo...** it is condemned to reinventing **[square!]** wheels as long as it does not realize that such issues have been studied in depth for the past 50 years in theoretical CS!

## Recapitulation—*what you must remember from this talk... (ctd)*

---

Pending issues re. “ontological programming”

▶ **Syntax:**

- What’s **essential**?
- What’s **superfluous**?

**Confusing notation** : XML-based cluttered verbosity

*ok, not for human consumption—but still!*

▶ **Semantics:**

- What’s a **model** good for?
- What’s (efficiently) **provable**?
- **decidable**  $\neq$  **efficient**
- **undecidable**  $\neq$  **inefficient**

▶ **Applications, maintenance, evolution, etc., ...**

▶ **Many, many, publications...** but no (real) field testing as yet!

## Recapitulation—*what you must remember from this talk... (ctd)*

---

**Proposal:** take heed of the following facts:

- ▶ **Linked data** represents all information as interconnected sorted labelled *RDF* graphs—it has become a universal *de facto* knowledge model standard
- ▶ **Differences between *DL* and *OSF* can come handy:**
  - *DL* is **expansive**—therefore, **expensive**—and can only describe finitely computable sets; whereas,
  - *OSF* is **contractive**—therefore, **efficient**—and can also describe recursively-enumerable sets
- ▶ ***CLP*-based graph unification reasoning = practical KR:**
  - ***structural***: objects, classes, inheritance
  - ***non-structural***: path equations, relational constraints, type definitions



# Innovation takes courage. . . (from Martin Wildberger's "Smarter Planet" Keynote, CASCON 2009)

*If I'd asked my customers what they wanted, they'd have said a faster horse!—Henry Ford*



Thank You For Your Attention !

For more information:

[hak@acm.org](mailto:hak@acm.org)

<http://cs.brown.edu/people/pvh/CPL/Papers/v1/hak.pdf>

<http://cedar.liris.cnrs.fr>

