Reasoning and the Semantic Web

Hassan Aït-Kaci

ANR Chair of Excellence

Université Claude Bernard Lyon 1

Constraint Event-Driven Automated Reasoning Project

CEDAR

Outline

- Constraint Logic Programming
- What is unification?
- Semantic Web objects
- Graphs as constraints
- $\mathcal{OWL}$ and $\mathcal{DL}$-based reasoning
- Constraint-based Semantic Web reasoning
- Recapitulation

Outline

- **Constraint Logic Programming**
- What is unification?
- Semantic Web objects
- Graphs as constraints
- $\mathcal{OWL}$ and $\mathcal{DL}$-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 $\mathcal{CLP}$ ***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 $X, Y, \ldots$);
- a set Φ of *formulae* (denoted $\phi, \phi', \ldots$) called *constraints*;
- a function **VAR**: $\Phi \mapsto \mathcal{V}$, giving for every constraint ϕ the set **VAR**(ϕ) of *variables constrained by* ϕ;
- 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 $\mathcal{R}$ ($r, r_1, \ldots$), 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 ϕ in Φ, *i.e.*, all $\mathcal{L}$-constraints;
  - all relational atoms $r(X_1, \ldots, X_n)$, where $X_1, \ldots, X_n \in \mathcal{V}$ are mutually distinct;

  and closed under $\&$ (conjunction) and $\rightarrow$ (implication);
- extending an interpretation $\mathcal{A}$ of $\mathcal{L}$ by adding relations: $r^{\mathcal{A}} \subseteq D^{\mathcal{A}} \times \ldots \times D^{\mathcal{A}}$ for each $r \in \mathcal{R}$.

Constraint Logic Programming Clause

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

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

where $(0 \leq m)$ and:

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

A constrained *resolvent* is a formula $\varrho \ \| \ \phi$, where ϱ is a (possibly empty) conjunction of relational atoms $r(X_1, \ldots, X_n)$—its *relational part*—and ϕ 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:

$$B_1 \ \& \ \ldots \ \& \ r(X_1, \ldots, X_n) \ \& \ \ldots B_k \ \| \ \phi$$

by the (renamed) program clause:

$$r(X_1, \ldots, X_n) \leftarrow A_1 \ \& \ \ldots \ \& \ A_m \ \| \ \phi'$$

is the new constrained resolvent of the form:

$$B_1 \ \& \ \ldots \ \& \ A_1 \ \& \ \ldots \ \& \ A_m \ \& \ \ldots B_k \ \| \ \phi \ \& \ \phi'.$$

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 $\mathcal{CLP}$!

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:

(n) Rule Name

$$\frac{\mathtt{Prior\ Form}}{\mathtt{Posterior\ Form}} \quad \text{if } \mathtt{Condition}$$

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
- $\mathcal{OWL}$ and $\mathcal{DL}$-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*;
- Σ_n sets of *constructors* of arity n $(n \geq 0)$;
- $\Sigma = \cup_{n\geq 0}\Sigma_n$ the constructor *signature*.

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

- a variable in $\mathcal{V}$; or,
- an element of Σ_0; or,
- an expression of the form $f(t_1, \ldots, t_n)$,
  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 σ and a FOT t, the σ-instance of t is the FOT:

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

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

$$\boxed{t \doteq t'}$$

What is unification?—FOT equation solving

A **solution**, if one exists, is any substitution σ such that:

$$\boxed{t\sigma = t'\sigma}$$

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

where: "σ_1 *is more general than* σ_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 ϕ

(2) Decompose

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

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

(3) Fail

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

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

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

(5) Erase

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

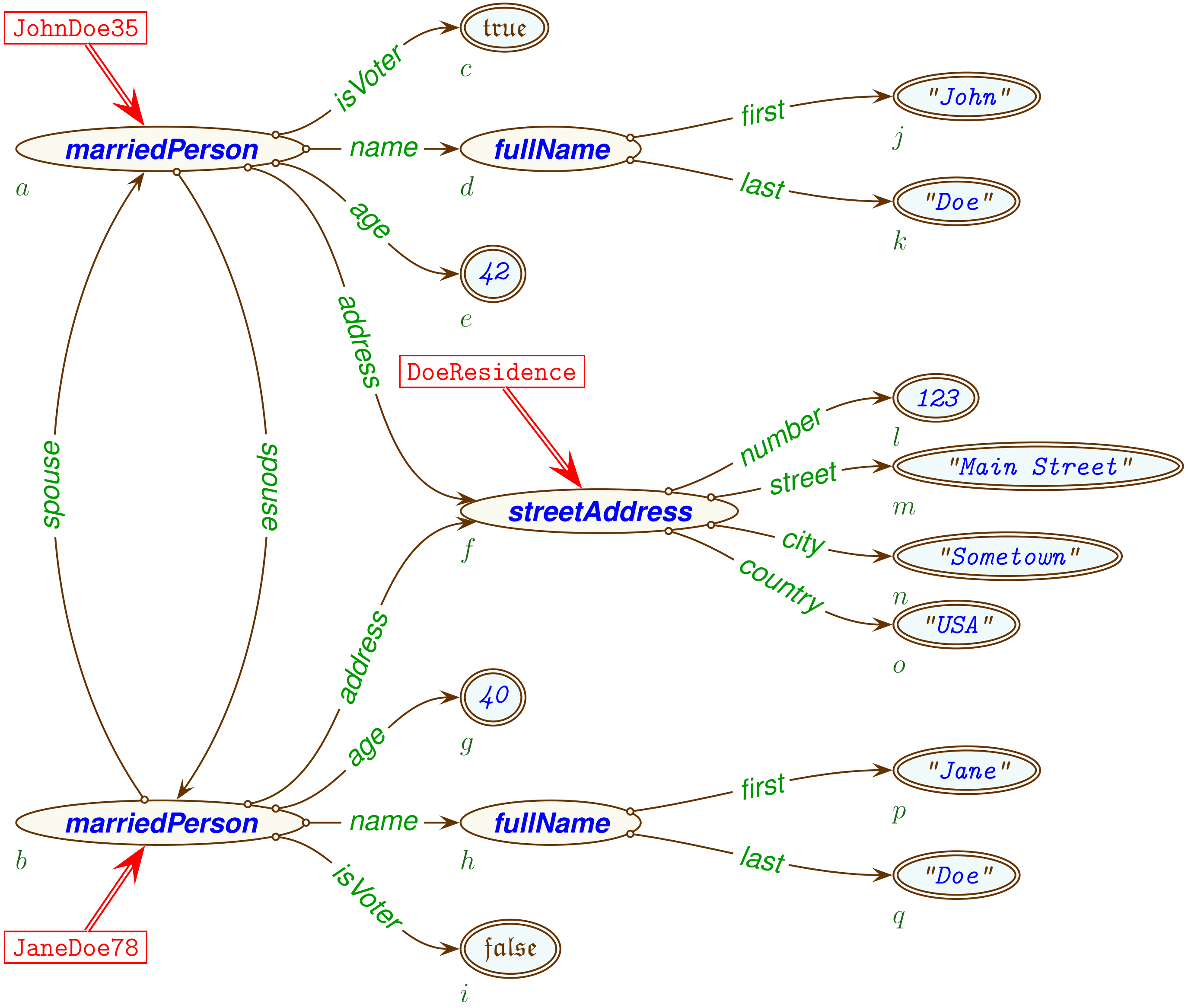
(6) Cycle

$$\frac{\phi \;\&\; X \doteq t}{\bot} \qquad \begin{array}{ll} \text{if} & X \in \mathcal{V} \\ \text{and} & t \notin \mathcal{V} \\ \text{and} & X \text{ occurs in } t \end{array}$$

Outline

- Constraint Logic Programming
- What is unification?
- **Semantic Web objects**
- Graphs as constraints
- $\mathcal{OWL}$ and $\mathcal{DL}$-based reasoning
- Constraint-based Semantic Web reasoning
- Recapitulation

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



Semantic Web objects—*Objects are labelled graphs!*

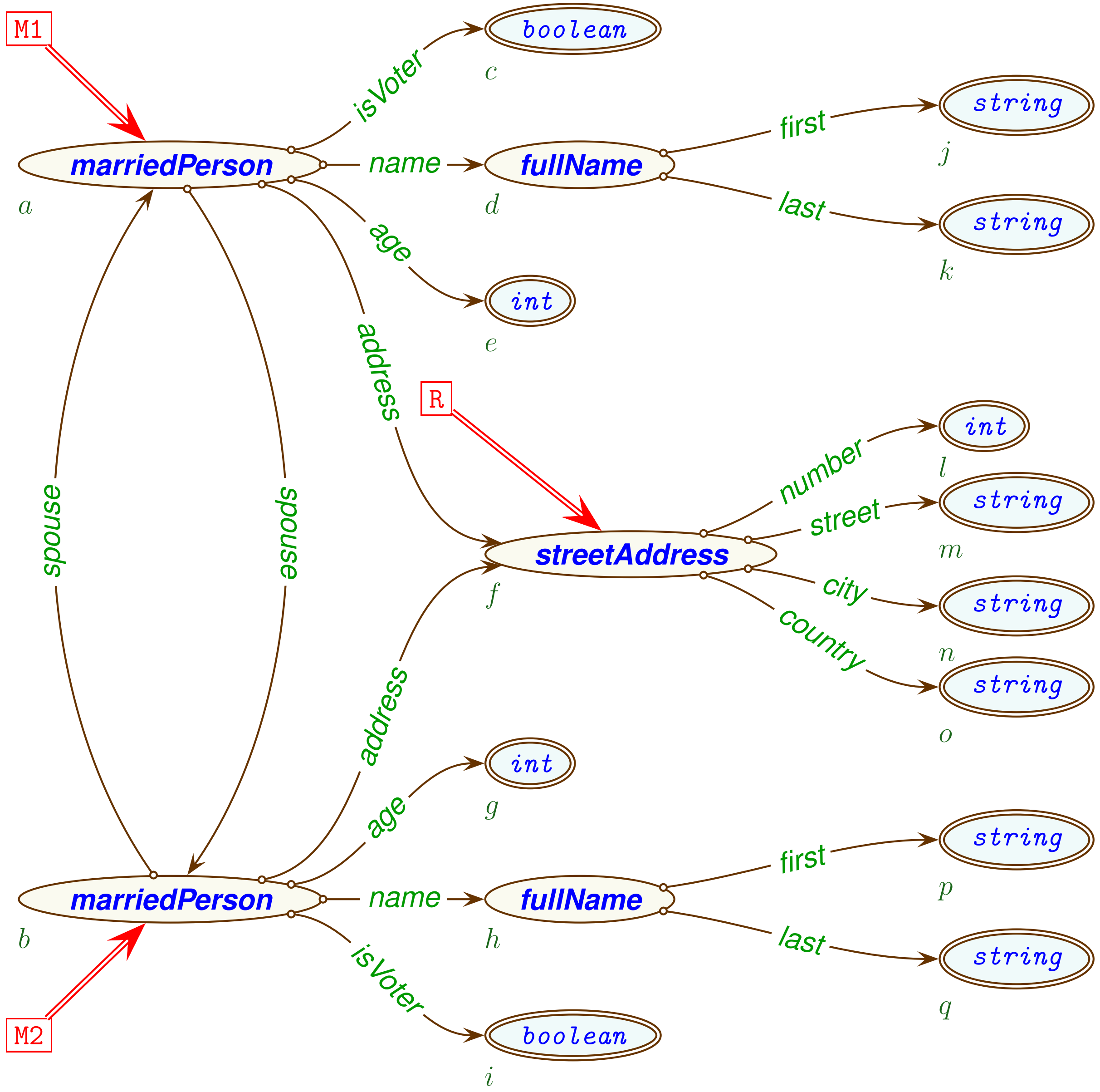
```
JohnDoe35 : marriedPerson ( name     => fullName
                                         ( first => "John"
                                         , last  => "Doe" )
                          , age      => 42
                          , address  => DoeResidence
                          , spouse   => 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**
- $\mathcal{OWL}$ and $\mathcal{DL}$-based reasoning
- Constraint-based Semantic Web reasoning
- Recapitulation

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

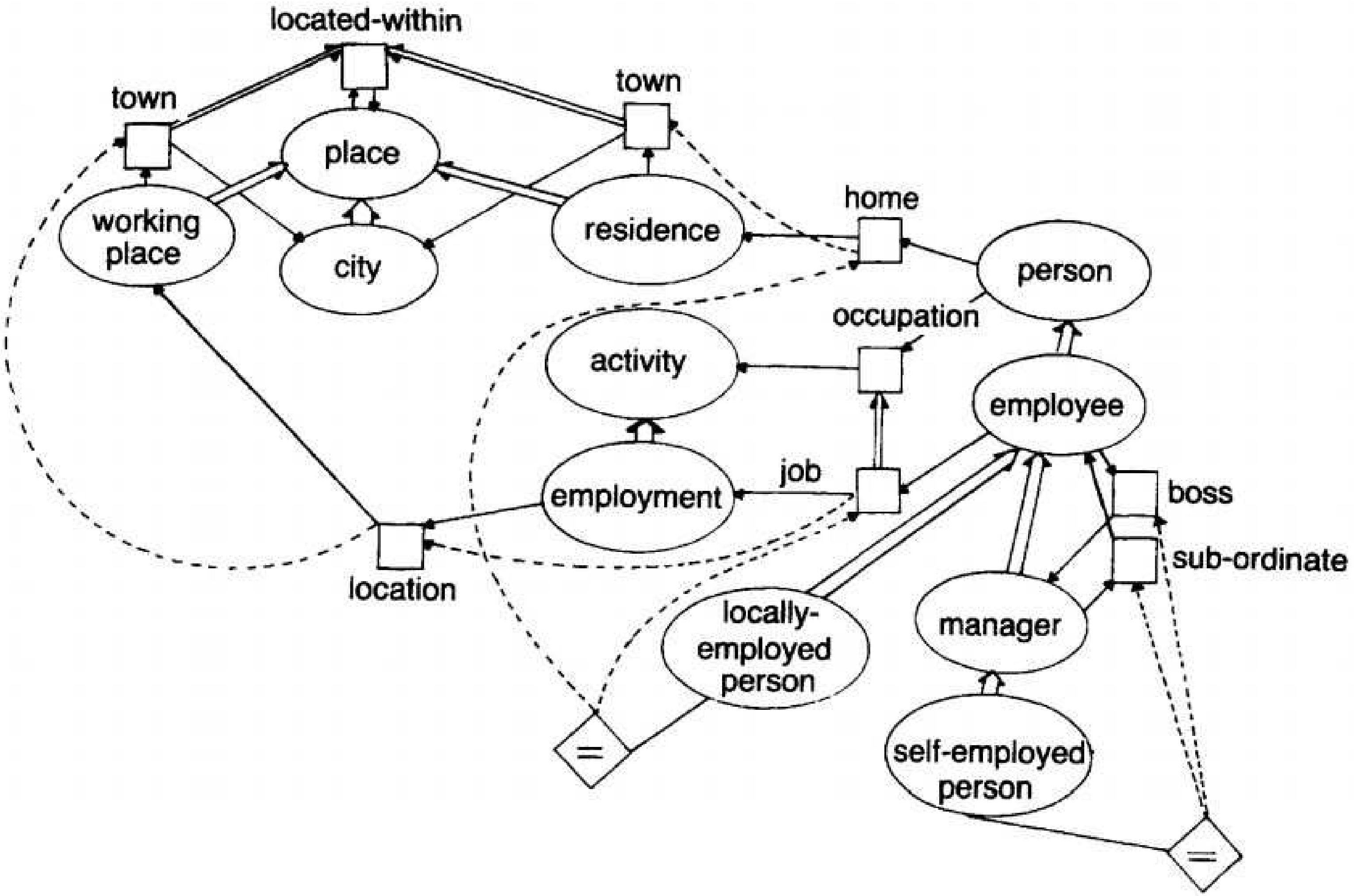


Fig. 1. Example of a KL-ONE semantic network.

Graphs as constraints—*Motivation*

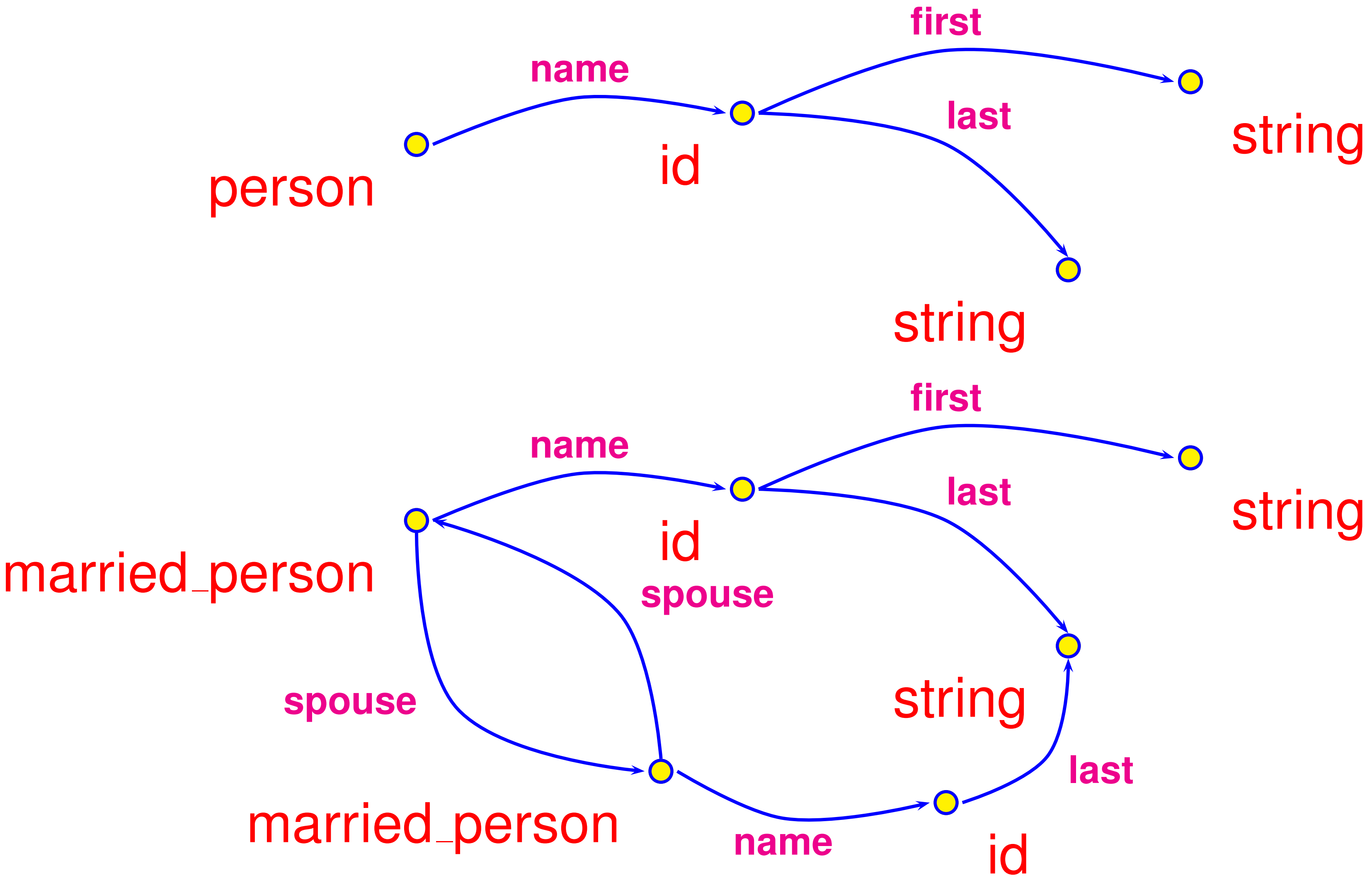
- **What:** a formalism for representing objects that is: intuitive (objects as labelled graphs), expressive (“real-life” data models), formal (logical semantics), operational (executable), & efficient (constraint-solving)
- **Why?** *viz.*, ubiquitous use of labelled graphs to structure information 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.*, ...

Graphs as constraints—*History*

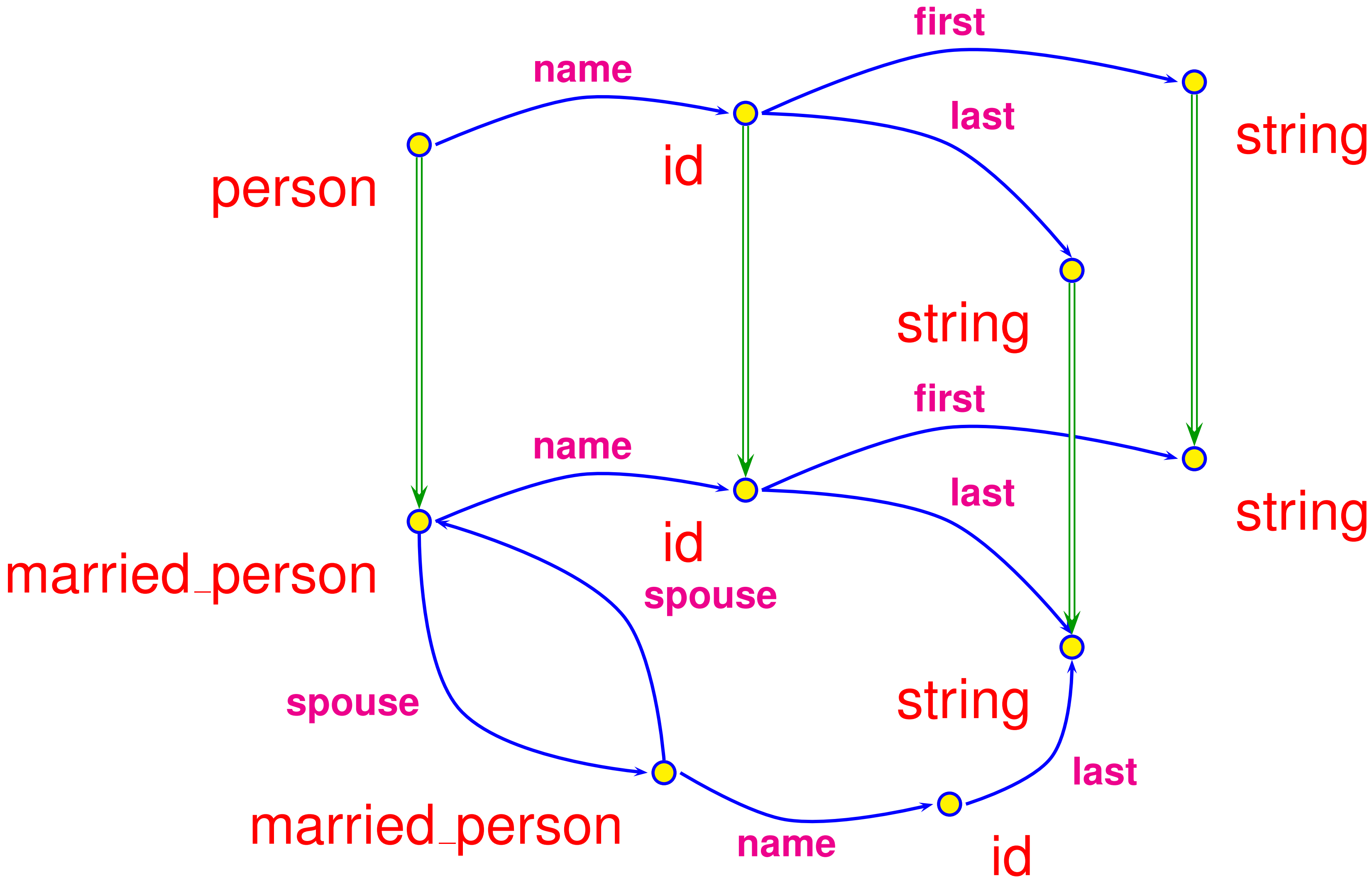
Viewing graphs as ***constraints*** stems from the work of:

- 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—$\mathcal{OSF}$ *term syntax*

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

An $\mathcal{OSF}$ term is an expression of the form:

$$\boxed{X : s(\ell_1 \Rightarrow t_1, \ldots, \ell_n \Rightarrow t_n)}$$

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 $X : s$)
- $\{\ell_1, \ldots, \ell_n\} \subseteq \mathcal{F}$ are features
- $t_1, \ldots, t_n$ are $\mathcal{OSF}$ terms

Graphs as constraints—$\mathcal{OSF}$ *term syntax example*

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

Lighter notation (showing only shared variables):

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

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

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

- $X : s$
- $X.\ell \doteq X'$
- $X \doteq X'$

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

$$\phi_1 \quad \& \quad \ldots \quad \& \quad \phi_n$$

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

An $\mathcal{OSF}$ term $t = X : s(\ell_1 \Rightarrow t_1, \ldots, \ell_n \Rightarrow t_n)$ is dissolved into an $\mathcal{OSF}$ clause $\phi(t)$ as follows:

$$\boxed{\begin{array}{lllllllll} \varphi(t) & \overset{\text{DEF}}{=\!=} & X : s & \& & X.\ell_1 \doteq X_1 & \& & \ldots & \& & X.\ell_n \doteq X_n \\ & & & \& & \varphi(t_1) & \& & \ldots & \& & \varphi(t_n) \end{array}}$$

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

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

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

$$\begin{aligned} \varphi(t) = X : person \;\; & \& \;\; X.name \doteq N \;\; \& \;\; N : \top \\ & \& \;\; X.name \doteq M \;\; \& \;\; M : id \\ & \& \;\; X.spouse \doteq P \;\; \& \;\; P : person \\[1ex] & \& \;\; N.first \doteq F \;\; \& \;\; F : string \\ & \& \;\; M.last \doteq S \;\; \& \;\; S : string \\[1ex] & \& \;\; P.name \doteq I \;\; \& \;\; I : id \\ & \& \;\; I.last \doteq S \;\; \& \;\; S : \top \\ & \& \;\; P.spouse \doteq X \;\; \& \;\; X : \top \end{aligned}$$

Graphs as constraints—*Basic $\mathcal{OSF}$ term normalization*

(1) Sort Intersection

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

(3) Variable Elimination

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

if $X \neq X'$
and $X \in \textbf{\textit{Var}}(\phi)$

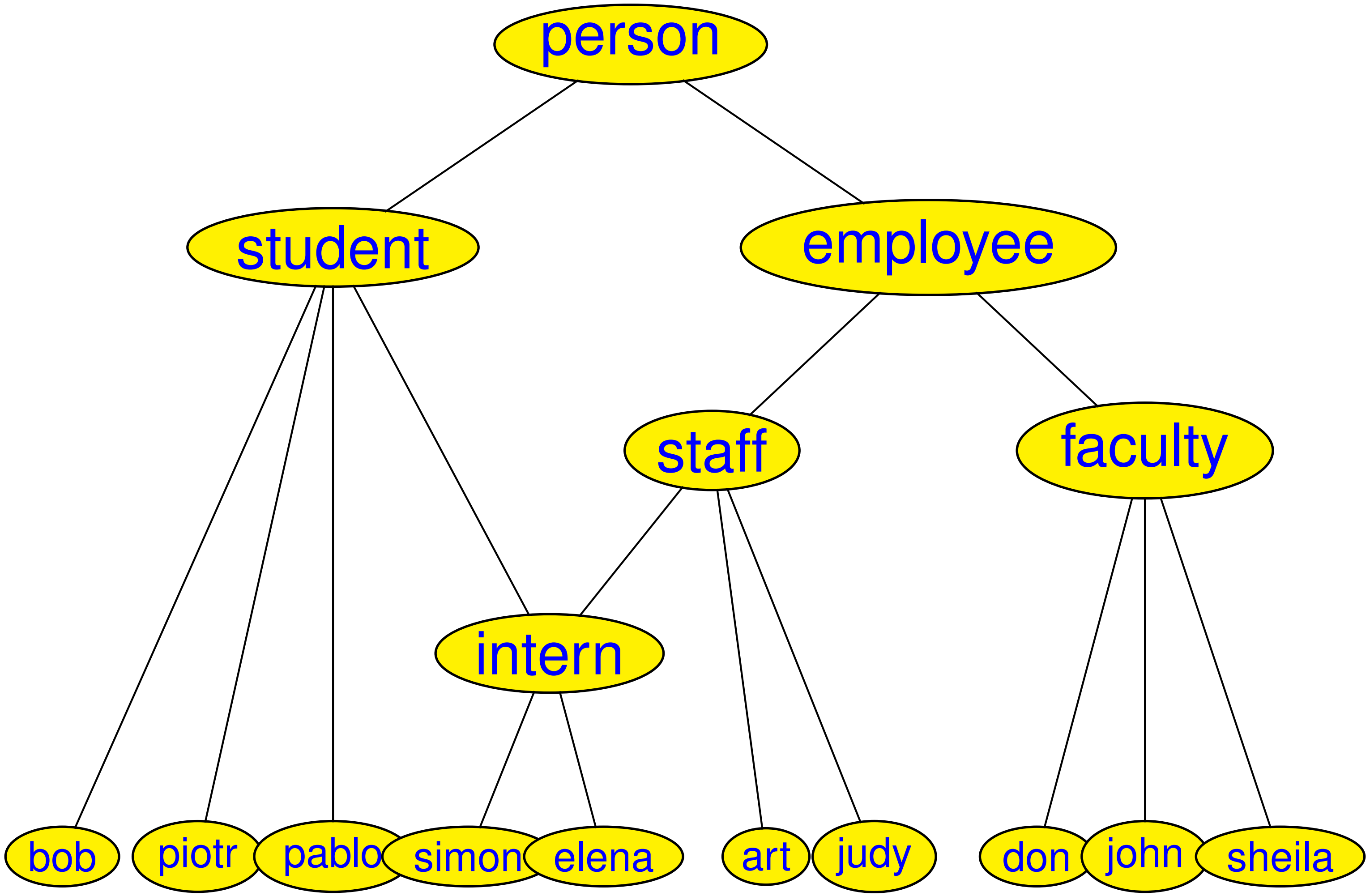
(2) Inconsistent Sort

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

(4) Feature Functionality

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

Graphs as constraints—$\mathcal{OSF}$ *unification as* $\mathcal{OSF}$ *constraint normalization*



Graphs as constraints—$\mathcal{OSF}$ *unification as* $\mathcal{OSF}$ *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—$\mathcal{OSF}$ *unification as* $\mathcal{OSF}$ *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 $\mathcal{OSF}$ terms***

Basic $\mathcal{OSF}$ 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.*, “$\mathcal{OSF}$ 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
- **$\mathcal{OWL}$ and $\mathcal{DL}$-based reasoning**
- Constraint-based Semantic Web reasoning
- Recapitulation

Semantic Web formalisms—$\mathcal{OWL}$ *speaks*

What language(s) do $\mathcal{OWL}$'s speak?—a confusing growing crowd of strange-sounding languages and logics:

- $\mathcal{OWL}$, $\mathcal{OWL}$ Lite, $\mathcal{OWL}$ DL, $\mathcal{OWL}$ Full
- $\mathcal{DL}$, $\mathcal{DLR}$, ...
- $\mathcal{AL}$, $\mathcal{ALC}$, $\mathcal{ALCN}$, $\mathcal{ALCNR}$, ...
- $\mathcal{SHIF}$, $\mathcal{SHIN}$, $\mathcal{CIQ}$, $\mathcal{SHIQ}$, $\mathcal{SHOQ}(D)$, $\mathcal{SHOIQ}$, $\mathcal{SRIQ}$, $\mathcal{SROIQ}$, ...

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 $\mathcal{OWL}$ 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

Tableaux style $\mathcal{DL}$ reasoning ($\mathcal{ALCNR}$)

CONJUNCTIVE CONCEPT:

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

DISJUNCTIVE CONCEPT:

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

UNIVERSAL ROLE:

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

EXISTENTIAL ROLE:

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

MIN CARDINALITY:

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

MAX CARDINALITY:

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

Understanding $\mathcal{OWL}$ *speak*—$\mathcal{OSF}$ *vs.* $\mathcal{DL}$

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

ergo:

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

Reasoning with knowledge expressed as constrained ($\mathcal{OSF}$) *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
- $\mathcal{OWL}$ and $\mathcal{DL}$-based reasoning
- **Constraint-based Semantic Web reasoning**
- Recapitulation

$\mathcal{LIFE}$—Rules + constraints for Semantic Web reasoning

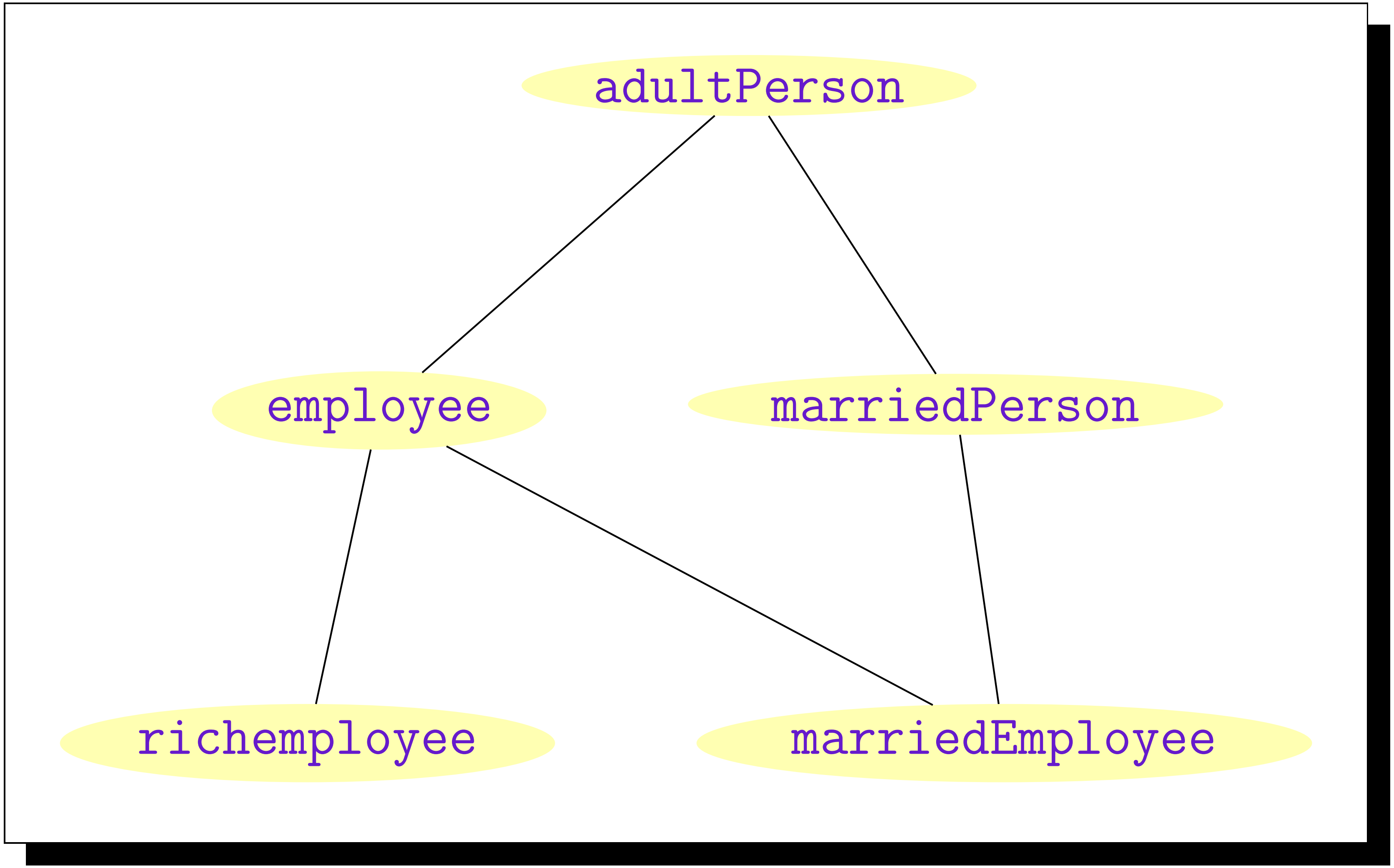
$\mathcal{LIFE}$—$\mathcal{L}$ogic, $\mathcal{I}$nheritance, $\mathcal{F}$unctions, and $\mathcal{E}$quations

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

$\mathcal{LIFE}$ is a $\mathcal{CLP}$ system over $\mathcal{OSF}$ constraints and functions over them (rewrite rules); namely:

$$\boxed{\mathcal{LIFE} = \mathcal{CLP}(\mathcal{OSF} + \mathcal{FP})}$$

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 $\mathcal{LIFE}$ 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 $\mathcal{LIFE}$ sort hierarchy

$$:: \; M : \texttt{marriedPerson} \; (\; \texttt{spouse} \Rightarrow P : \texttt{marriedPerson} \;)$$

$$\mid \; P.\texttt{spouse} = M.$$

$$:: \; R : \texttt{richEmployee} \; (\; \texttt{institution} \Rightarrow I \; , \; \texttt{salary} \Rightarrow S \;)$$

$$\mid \; \texttt{stockValue}(I) = V \; , \; \texttt{hasShares}(R, I, N) \; , \; S + N * V \geq 200000.$$

Proof “memoizing”

$\mathcal{OSF}$ 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!**

- $\mathcal{OSF}$-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, $\mathcal{OSF}$ ***constraint proof rules ensure that:***

no type constraint is ever proved twice

Proof "memoizing"

$\mathcal{OSF}$ type constraints are incrementally "*memoized*" as they are verified:

sorts act as (instantaneous!) proof caches!

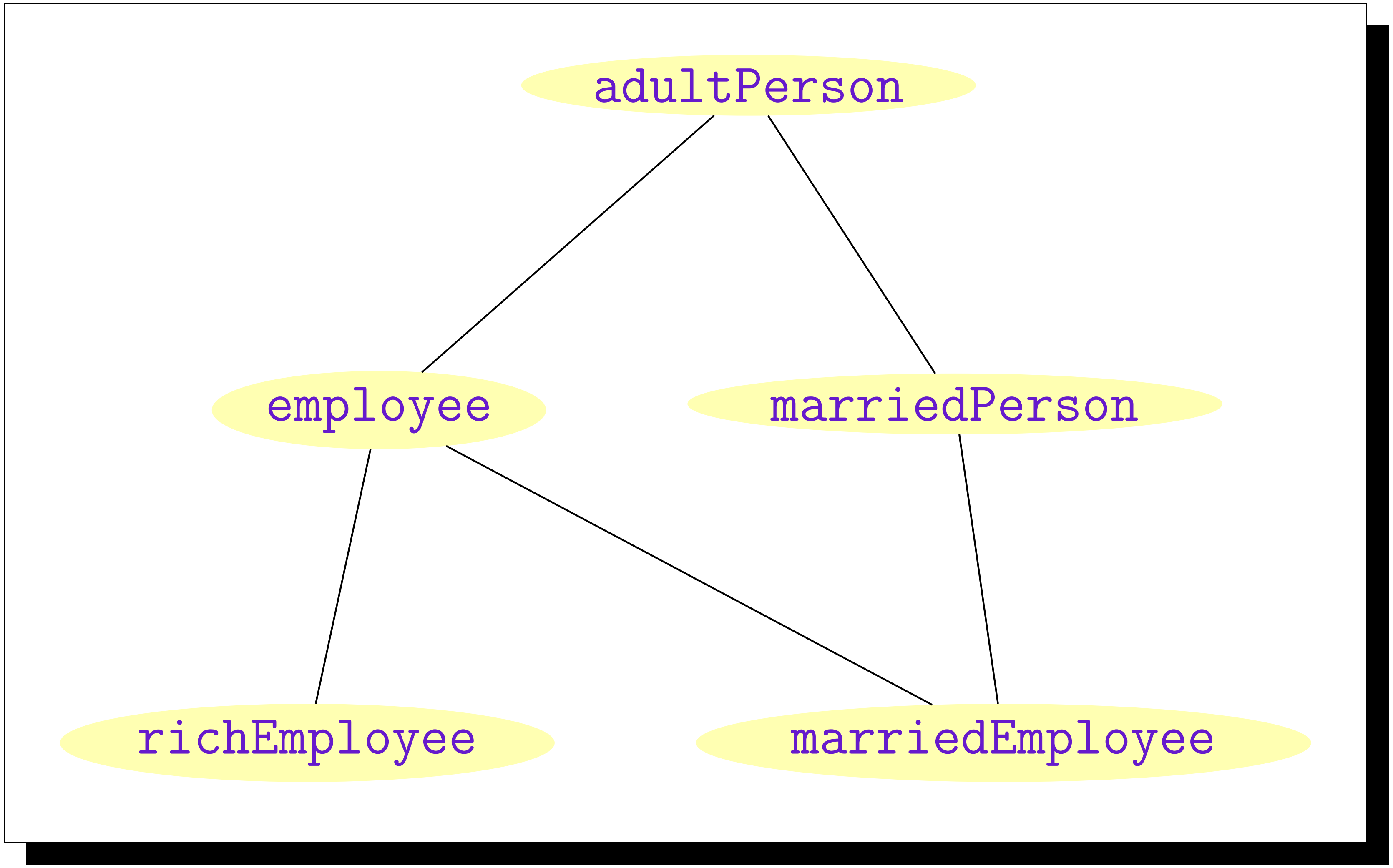
whereas in logic having proven $s(X)$ is not *"remembered"* in any way (*e.g.*, Prolog)

Example: consider the $\mathcal{OSF}$ constraint conjunction:

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

Notation: $\texttt{type\#}(\texttt{condition})$ means "*constraint* `condition` *attached to sort* `type`"

Proof “memoizing”—Example hierarchy reminded



Proof “memoizing”

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

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 $\mathcal{OSF}$ constraint-solving enables:

using rules over ontologies

as well as, *conversely*,

enhancing ontologies with rules

Indeed, with $\mathcal{OSF}$:

- ***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
- $\mathcal{OWL}$ and $\mathcal{DL}$-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!

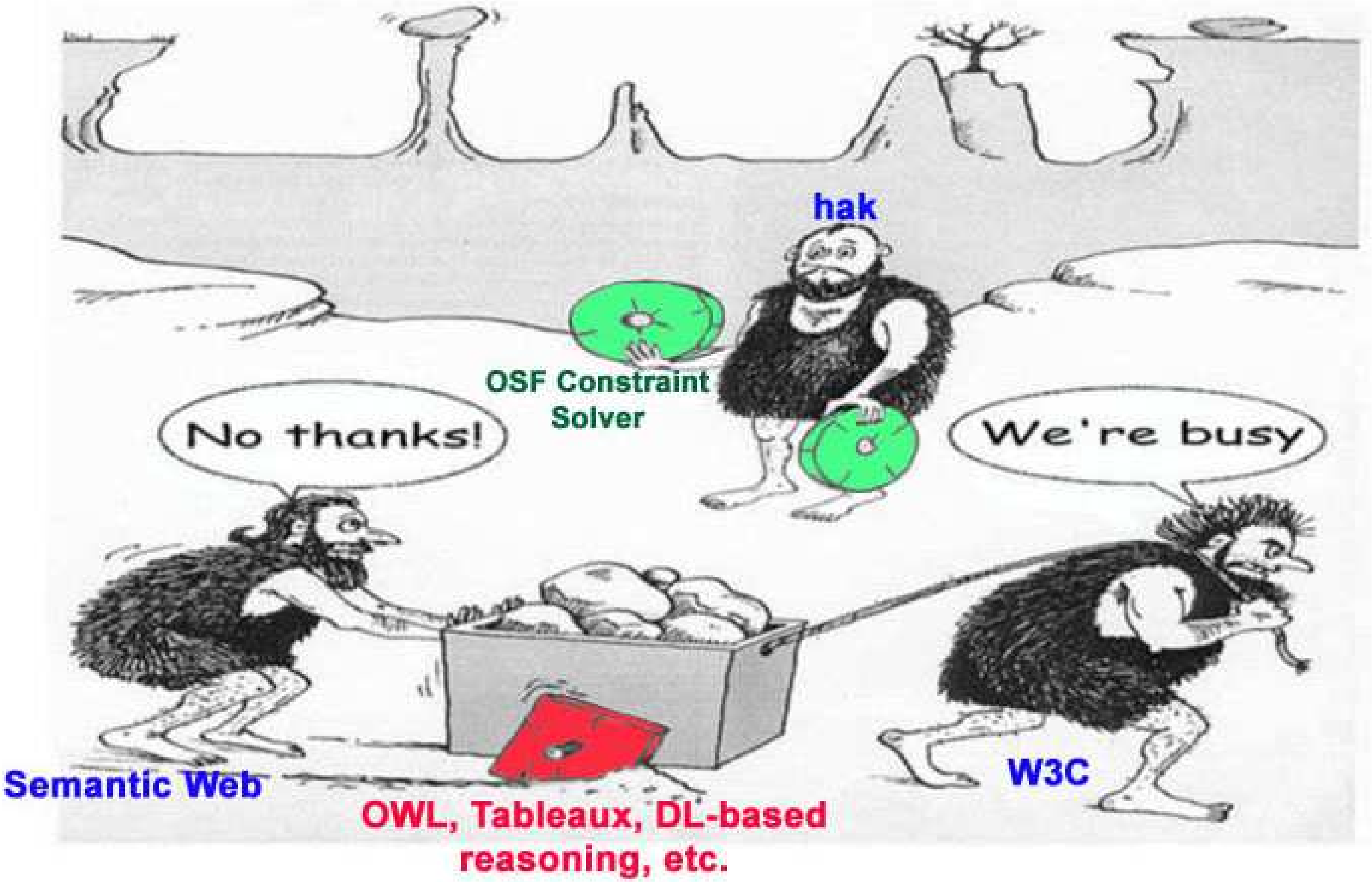
Recapitulation—*what you must remember from this talk. . . (ctd)*

Proposal: take heed of the following facts:

- **Linked data** represents all information as interconnected sorted labelled $\mathcal{RDF}$ graphs—it has become a universal *de facto* knowledge model standard
- **Differences between $\mathcal{DL}$ and $\mathcal{OSF}$ can come handy**:
  - $\mathcal{DL}$ is expansive—therefore, expensive—and can only describe finitely computable sets; whereas,
  - $\mathcal{OSF}$ is contractive—therefore, efficient—and can also describe recursively-enumerable sets
- **$\mathcal{CLP}$-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

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

http://cedar.liris.cnrs.fr