Reasoning and the Semantic Web

Hassan Aït-Kaci

ANR Chair of Excellence
Université Claude Bernard Lyon 1

Constraint Event-Driven Automated Reasoning Project
Reasoning and the Semantic Web

Outline

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

Outline

- Constraint Logic Programming
- What is unification?
- Semantic Web objects
- Graphs as constraints
- OWL and DL-based reasoning
- Constraint-based Semantic Web reasoning
- Recapitulation
In Prolog seen as a $\mathcal{CLP}$ language, a clause such as:

\[
\text{append}([], L, L).
\]
\[
\text{append}([H|T], L, [H|R]) :- \text{append}(T, L, R).
\]

is construed as:

\[
\text{append}(X_1, X_2, X_3) :- \text{true}
\]
\[
| \begin{align*}
X_1 &= [], & X_2 &= L, & X_3 &= L. \\
\end{align*}
\]
\[
\text{append}(X_1, X_2, X_3) :- \text{append}(X_4, X_5, X_6)
\]
\[
| \begin{align*}
X_1 &= [H|T], & X_2 &= L, & X_3 &= [H|R], \\
X_4 &= T, & X_5 &= L, & X_6 &= R. \\
\end{align*}
\]
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 $X, Y, \ldots$);
- a set $\Phi$ of formulae (denoted $\phi, \phi', \ldots$) called constraints;
- a function $\text{VAR}: \Phi \mapsto \mathcal{V}$, giving for every constraint $\phi$ the set $\text{VAR}(\phi)$ of variables constrained by $\phi$;
- a family of interpretations $\mathcal{A}$ over some domain $D^\mathcal{A}$;
- a set $\text{VAL}(\mathcal{A})$ 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 $\phi$ in $\Phi$, 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 $r \in \mathcal{R}$. 
We define a \(\text{CLP} \) constrained \textit{definite clause} in \(\mathcal{R}(\mathcal{L})\) as:

\[
r(\vec{X}) \leftarrow r_1(\vec{X}_1) \land \ldots \land r_m(\vec{X}_m) \parallel \phi,
\]

where \((0 \leq m)\) and:

\begin{itemize}
  \item \(r(\vec{X}), r_1(\vec{X}_1), \ldots, r_m(\vec{X}_m)\) are relational atoms in \(\mathcal{R}(\mathcal{L})\); and,
  \item \(\phi\) is a constraint formula in \(\mathcal{L}\).
\end{itemize}

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

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

by the (renamed) program clause:

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

is the new constrained resolvent of the form:

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

$\begin{align*}
\text{(n) Rule Name} \\
\text{Prior Form} \quad \text{if Condition} \\
\text{Posterior Form}
\end{align*}$

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** $\bot$, 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!**
Reasoning and the Semantic Web

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

- $V$ 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 $V$; or,
- an element of $\Sigma_0$; 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).
A variable substitution is a map $\sigma : \mathcal{V} \rightarrow \mathcal{T}_{\Sigma, \mathcal{V}}$ such that the set $\{X \in \mathcal{V} | \sigma(X) \neq X\}$ is finite.

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

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

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

$$t \doteq t'$$
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**): $\text{mgu}(t, t')$.

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

\[ \phi \ & \ X \doteq t \]

\[ \phi[X/t] \ & \ X \doteq t \]

if \( X \) occurs in \( \phi \)

(2) Decompose

\[ \phi \ & \ f(s_1, \ldots, s_n) \doteq f(t_1, \ldots, t_n) \]

\[ \phi \ & \ s_1 \doteq t_1 \ & \ \ldots \ & \ s_n \doteq t_n \]

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

(3) Fail

\[ \phi \ & \ f(s_1, \ldots, s_n) \doteq g(t_1, \ldots, t_m) \]

\[ \bot \]

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
\[ \phi \land t \doteq X \]
\[ \phi \land X \doteq t \]
if \( X \in \mathcal{V} \)
and \( t \notin \mathcal{V} \)

(5) Erase
\[ \phi \land t \doteq t \]
\[ \phi \]
if \( t \in \Sigma_0 \cup \mathcal{V} \)

(6) Cycle
\[ \phi \land X \doteq t \]
\[ \bot \]
if \( X \in \mathcal{V} \)
and \( t \notin \mathcal{V} \)
and \( X \) occurs in \( t \)
Reasoning and the Semantic Web

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!
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!

marriedPerson
  a
  marriedPerson
  b

fullName
  c
  d

streetAddress
  e
  f
  g
  h
  i
  j
  k
  l
  m
  n
  o
  p
  q

isVoter
  M1
  M2

isVoter
  boolean
  string
  int
  string
  int
  string
  string
  string
  boolean
  string
  string
  string
  string

name

first

last

address

number

first

last

name

age

street

city

country

j

k
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
                 )
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—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:

\[
X : s(\ell_1 \Rightarrow t_1, \ldots, \ell_n \Rightarrow t_n)
\]

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 OSF terms
Graphs as constraints—OSF term syntax example

\[ X : person(name \Rightarrow N : \top (first \Rightarrow F : string), \]
\[ \quad \text{name} \Rightarrow M : \text{id}(last \Rightarrow S : \text{string}), \]
\[ \quad \text{spouse} \Rightarrow P : person(name \Rightarrow I : \text{id}(last \Rightarrow S : \top), \]
\[ \quad \quad \text{spouse} \Rightarrow X : \top). \]

Lighter notation (showing only shared variables):\n
\[ X : person(name \Rightarrow \top (first \Rightarrow \text{string}), \]
\[ \quad \text{name} \Rightarrow \text{id}(last \Rightarrow S : \text{string}), \]
\[ \quad \text{spouse} \Rightarrow person(name \Rightarrow \text{id}(last \Rightarrow S), \]
\[ \quad \quad \text{spouse} \Rightarrow X)). \]
Graphs as constraints—OSF clause syntax

An OSF constraint is one of:

\[ X : s \]
\[ X.\ell \cong X' \]
\[ X \cong X' \]

where \( X \) (\( X' \)) is a variable (i.e., a node), \( s \) is a sort (i.e., a node’s type), and \( \ell \) is a feature (i.e., an arc).

An OSF clause is a conjunction of OSF constraints—i.e., a set of OSF constraints

\[ \phi_1 \& \ldots \& \phi_n \]
Graphs as constraints—\textit{From OSF terms to OSF clauses}

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

\[
\phi(t) \overset{\text{def}}{=} X : s \land X.\ell_1 \dashv X_1 \land \ldots \land X.\ell_n \dashv X_n \\
\land \phi(t_1) \land \ldots \land \phi(t_n)
\]

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

\[
t = X: \text{person}(\text{name} \Rightarrow N: \top (\text{first} \Rightarrow F: \text{string}), \\
\quad \text{name} \Rightarrow M: \text{id}(\text{last} \Rightarrow S: \text{string}), \\
\quad \text{spouse} \Rightarrow P: \text{person}(\text{name} \Rightarrow I: \text{id}(\text{last} \Rightarrow S: \top), \\
\quad \text{spouse} \Rightarrow X: \top))
\]

\[
\varphi(t) = X: \text{person} \quad \& \quad X. \text{name} \triangleq N \quad \& \quad N: \top \\
\quad \& \quad X. \text{name} \triangleq M \quad \& \quad M: \text{id} \\
\quad \& \quad X. \text{spouse} \triangleq P \quad \& \quad P: \text{person} \\
\quad \& \quad N. \text{first} \triangleq F \quad \& \quad F: \text{string} \\
\quad \& \quad M. \text{last} \triangleq S \quad \& \quad S: \text{string} \\
\quad \& \quad P. \text{name} \triangleq I \quad \& \quad I: \text{id} \\
\quad \& \quad I. \text{last} \triangleq S \quad \& \quad S: \top \\
\quad \& \quad P. \text{spouse} \triangleq X \quad \& \quad X: \top
\]
Graphs as constraints—Basic OSF term normalization

(1) Sort Intersection
\[ \phi \land X : s \land X : s' \quad \Rightarrow \quad \phi \land X : s \land s' \]

(2) Inconsistent Sort
\[ \phi \land X : \perp \quad \Rightarrow \quad X : \perp \]

(3) Variable Elimination
\[ \phi \land X = X' \quad \text{if} \quad X \neq X' \quad \text{and} \quad X \in \text{Var}(\phi) \]

\[ \phi[X'/X] \land X = X' \]

(4) Feature Functionality
\[ \phi \land X.\ell = X' \land X.\ell = X'' \]

\[ \phi \land X.\ell = X' \land X' = X'' \]
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

\[
X : \text{intern} \\
\quad (\text{roommate} \Rightarrow S : \text{intern}(\text{rep} \Rightarrow S), \\
\quad \text{advisor} \Rightarrow \text{don}(\text{assistant} \Rightarrow A, \\
\quad \quad \quad \text{secretary} \Rightarrow S), \\
\quad \text{helper} \Rightarrow \text{simon}(\text{spouse} \Rightarrow A)) \\
\&
\]

\[
X = Y \\
\&
\]

\[
E = S
\]
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 yields a powerful means for effectively using ontologies
Reasoning and the Semantic Web

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

- All the proposed DL Knowledge Base formalisms in the OWL family use tableau-based methods for reasoning
- Tableaux methods work by building models explicitly via formula expansion rules
- This limits 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—DL dialects

### Tableaux style DL reasoning (ALCNR)

<table>
<thead>
<tr>
<th>Concept Type</th>
<th>Rule</th>
<th>Result</th>
</tr>
</thead>
</table>
| **Conjunctive Concept** | $\begin{cases} \text{if } x : (C_1 \cap C_2) \in S \\
|                        | \text{and } \{x : C_1, x : C_2\} \notin S \end{cases}$               | $S \cup \{x : C_1, x : C_2\}$         |
| **Disjunctive Concept** | $\begin{cases} \text{if } x : (C_1 \cup C_2) \in S \\
|                        | \text{and } x : C_i \notin S \ (i = 1, 2) \end{cases}$               | $S \cup \{x : C_i\}$                  |
| **Universal Role**     | $\begin{cases} \text{if } x : (\forall R.C) \in S \text{ and } y : R_S[x] \\
|                        | \text{and } y : C \notin S \end{cases}$                             | $S \cup \{y : C\}$                    |
| **Existential Role**   | $\begin{cases} \text{if } x : (\exists R.C) \in S \text{ s.t. } R \models \bigwedge_{i=1}^m R_i \\
|                        | \text{and } z : C \in S \Rightarrow z \notin R_S[x] \text{ and } y \text{ is new} \end{cases}$ | $S \cup \{x R_i y\}_{i=1}^m \cup \{y : C\}$ |
| **Min Cardinality**   | $\begin{cases} \text{if } x : (\geq n.R) \in S \text{ s.t. } R \models \bigwedge_{i=1}^m R_i \\
|                        | \text{and } |R_S[x]| \neq n \text{ and } y_i \text{ is new} \ (0 \leq i \leq n) \end{cases}$ | $S \cup \{x R_i y\}_{i=1}^{m,n} \cup \{y_i \neq y_j\}_{1 \leq i < j \leq n}$ |
| **Max Cardinality**   | $\begin{cases} \text{if } x : (\leq n.R) \in S \text{ s.t. } R \models \bigwedge_{i=1}^m R_i \\
|                        | \text{and } |R_S[x]| > n \text{ and } y, z \in R_S[x] \text{ and } y \neq z \notin S \text{ and } y \neq z \notin S \end{cases}$ | $S \cup S[y/z]$                        |
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*
Reasoning and the Semantic Web

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—Logic, Inheritance, Functions, and Equations

$\text{CLP}(\chi)$—Constraint, Logic, Programming, parameterized over is a constraint system $\chi$

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

$$\text{LIFE} = \text{CLP}(\text{OSF} + \text{FP})$$
A multiple-inheritance hierarchy
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 LITE

employee <:成人Person.
marrriedPerson <:成人Person.
richEmployee <:employee.
marrriedEmployee <:employee.
marrriedEmployee <:marredPerson.

::成人Person (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 \( \text{LIFE} \) sort hierarchy

\[
\begin{align*}
\text{adultPerson} &:\quad \langle \text{id} \Rightarrow \text{name},
\quad \text{dob} \Rightarrow \text{date},
\quad \text{age} \Rightarrow \text{int},
\quad \text{ssn} \Rightarrow \text{string} \rangle \\
\text{ageInYears} &\equiv \text{ageInYears}(\text{P}), \quad \text{A} \geq 18.
\end{align*}
\]

\[
\begin{align*}
\text{employee} &:\quad \langle \text{position} \Rightarrow \text{title},
\quad \text{institution} \Rightarrow \text{string},
\quad \text{supervisor} \Rightarrow \text{employee},
\quad \text{salary} \Rightarrow \text{int} \rangle \\
\text{higherRank} &\equiv \text{higherRank}(\text{E.position}, \text{T}), \quad \text{E.salary} \geq \text{S}.
\end{align*}
\]
A relationally and functionally constrained \texttt{LIFE} sort hierarchy

:: $M : \text{marriedPerson}$ (spouse $\Rightarrow P : \text{marriedPerson}$ )
  \hspace{1cm} | \hspace{1cm} P\.spouse = M.

:: $R : \text{richEmployee}$ (institution $\Rightarrow I$
  \hspace{2cm}, salary $\Rightarrow S$)
  \hspace{1cm} | \hspace{1cm} \text{stockValue}(I) = V$, hasShares($R, I, N$), $S + N \times V \geq 200000$. 
OSF constraints as syntactic variants of logical formulae:

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

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

**Coreferences** are equations: \( X \equiv Y \iff [X] = [Y] \)

So …

Why not use (good old) logic proofs instead?
Proof “memoizing”

But: \textbf{model equivalence} \neq \textbf{proof equivalence}!

- \textit{OSF}-unification proves sort constraints by reducing them monotonically w.r.t. the sort ordering
- \textit{ergo}, once \( X : s \) has been proven, the proof of \( s(X) \) is recorded as \textit{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, \textit{OSF constraint proof rules ensure that}:

  \textbf{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**: $\text{type#(condition)}$ means “constraint condition attached to sort type”
Proof “memoizing”—Example hierarchy reminded
Proof “memoizing”

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

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*
Reasoning and the Semantic Web

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** ≠ **efficient**
  
  – **undecidable** ≠ **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

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

http://cedar.liris.cnrs.fr