A-Box Reasoning –
A little Bit Different View

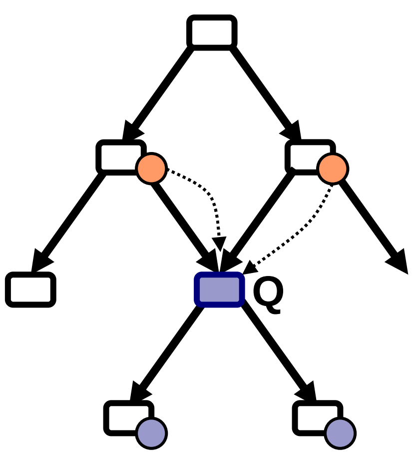
Motivation

- Applications
  - Configuration of complex components (e.g. PC with its components)
  - Query refinement
- Characteristics
  - Interactive
  - Knowledge (i.e. A-Box) will be “completed” incrementally

ABOX Reasoning

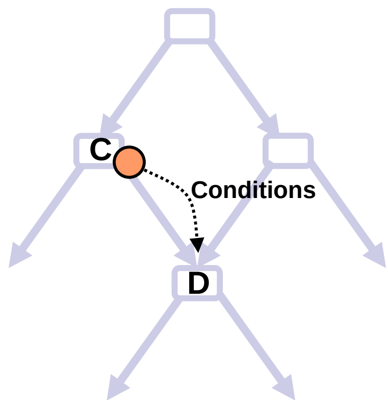
- Instance Retrieval
  - “Tell me all instances which are subsumed by a query Q?”
  - Needs inferences
  - Optimized with traditional database queries (only for role-free A-Boxes)
- Instance Realization
  - “To which classes do an instance I belong to?”
- …

Note: Retrieval needs Realization



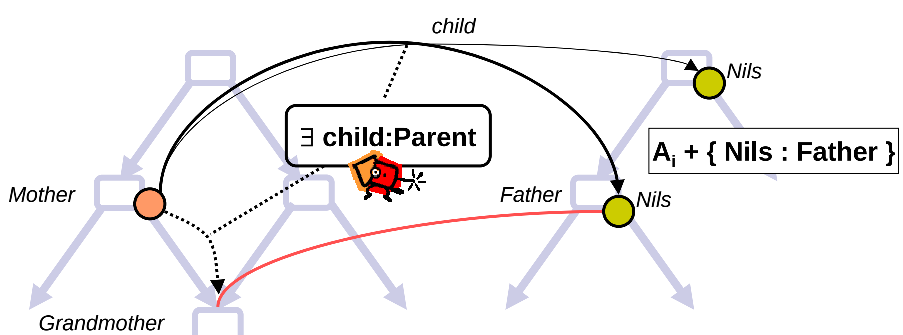
- Retrieval Process
  1. Classify Query Q
  2. Select Instances from subsumed classes
  3. Realize instances from direct parents, if the belongs to Q
- Cmp. Instance Store for role-free A-Boxes

Idea: “Decision Tree” for Instances



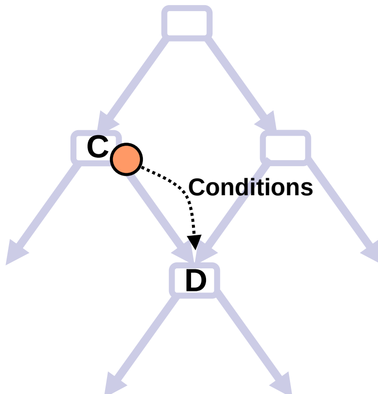
- Specify conditions, which allow to realize instance from class C to D
- Incremental behavior: Check conditions, when new information arrives
- Analogies
  - Data-driven vs. goal-driven inference method (tableaux method)
  - Bottom-up vs. Top-down
  - Constraint Propagation

Example: Specialization of a role filler



- See ABOX as a sequences of statements
  - A_i = { Anja : Woman, child(Anja, Nils), Nils: Man }
  - A_{i+1} = A_i + { Nils : Father }
  - Conclude in A_{i+1}: Anja : Grandmother

It is (theoretical) possible?



- Trivial languages (no union, no negation): sure
- Non-trivial languages: under investigation
- Depends on the expressiveness of the decision tree

Realization depends on the Concept Definition

- Necessary:

  $[X:C]! \leftarrow \text{condition}^{N}$

  "When can X be realized to C?"
- Sufficient:

  $[X:C]! \rightarrow \text{condition}_{S}$

  "What happen, when X is realized to C?"

Condition^{N} and Condition$_{S}$

- Conditions are generated from a concept term
- First Instance

  condition(C $\rightarrow$ D) = f(D)

- Later

  condition(C $\rightarrow$ D) = f(Δ)

  with D $\equiv$ C $\cap$ Δ; Δ is difference between C and D
  faster; more understandable and readable

Notation: Querying and Realizing

- Querying,
  - if an instance X belongs to class C
    [X : C]?
  - If an relation instance belongs to a relation R
    [R(X, Y)]?
- Realizing, (do it)
  - if an instance X belongs to class C
    [X : C]!
  - If an relation instance belongs to a relation R
    [R(X, Y)]!

Necessary conditions: When can X be realized to C?

Concept term	Condition^N f(.)	Remarks
X: $C \subseteq D$	noop	
X: $C \equiv D$	f(X:D)	
X: $C \supseteq D$	f(X:D)	
X: CN	[X : CN]?	CN primitive?
X: $D_1 \cap D_2$	f(X: D_1) $\oplus$ f(X: D_2)	
X: $D_1 \cup D_2$	f(X: D_1) $\otimes$ f(X: D_2)	DNF: two conditions
X: $\neg$CN	[X: CN_{free}]?	$CN_{free} \equiv \neg CN$; NNF required
X: $\exists$R.D (X: $\geq$1.R.D)	[R(X,Y)]? $\oplus$ f(Y:D)	Y must not be generated (like in the tableaux method)
X: $\forall$R.D	[X: CN_{free}]?	$CN_{free} \equiv \forall R.D$
X: $\forall$F.D (X: $\leq$1.R.D $\cap$ $\forall$R.E)	[F(X) = Y)]? $\oplus$ f(Y:D)	Features have an upper bound
X: $\geq$n.R.D	[R(X,$Y_{1..n}$) $\wedge$ $\forall Y_i \neq Y_j$]? $\oplus$ f($Y_{1..n}$:D)	$Y_{1..n}$ must not be generated
X: $\leq$n.R.D	[R(X,$Y_{1..n}$) $\wedge$ $\forall Y_i \neq Y_j$]? $\oplus$ f($Y_{1..n}$:D)	
X: $\leq$n.R.D $\cap$ $\forall$R.E	[R(X,$Y_{1..n}$) $\wedge$ $\forall Y_i \neq Y_j$]? $\oplus$ f($Y_{1..n}$:E)	$E \subseteq D$

Remarks (I)

- Using concept hierarchy to answer [X:C]?
  if $E \subseteq C$ and [X: E]?
- X: $C \subseteq D$ (= X: $C \equiv D \cap \Delta$)
  Δ is not known and (can not be) specified;
  X can only be realized to C if [X:E]! and $E \subseteq C$
- X: $\forall$R.D
  OWA prevents to deduce forall terms because every time an ABOX can be extended with a role filler R(X,Y) and Y: $\neg$D;
  X can only be realized to $CN_{free} \equiv \forall R.D$ with a free concept name CN_{free} in the TBOX
  if [X:E]? and $E \subseteq CN_{free}$
  - Note: Precompilation is needed; TBOX reasoning is needed a priori

Remarks (II)

- X: ¬CN
  Let the TBOX reasoning determine the conditions, if X belongs to a negated concept. X can only be realized to $CN_{free} \equiv \neg CN$ with a free concept name CN_{free} in the TBOX
  if [X:E]? and $E \subseteq CN_{free}$
  - Note: Concept terms has to be transformed to NNF
- X: ∃R.D (X: ≥1.R.D)
  Role filler must not be generated (like in the tableaux method), because only when the role filler is present in the ABOX some additional information can be associated to the role filler which can be used for the realization;
  BUT: then inconsistencies can not be detected; example
  ABOX = { X: ∀R.¬D , X: ∃R.D }

Sufficient conditions:
What happened when X is realized to C?

Concept term	Condition_S g(.)	Remarks
[X : C]?	noop	
X: C ⊆ D	g(X:D)	
X: C ≡ D	g(X:D)	
X: C ⊇ D	noop	
X: CN	[X : CN]!	CN primitive!
X: $D_1 \cap D_2$	g(X: D_1), g(X: D_2)	
X: $D_1 \cup D_2$	Not possible!	INCOMPLETE!!!
X: ¬CN	[X: CN_{free}]?	$CN_{free} \equiv \neg CN$; NNF required
X: ∃R.D (X: ≥1.R.D)	[R(X,Y)]? → g(Y:D), otherwise nothing!	Inconsistency can not be detected
X: ∀R.D	[R(X,Y)]? → g(Y:D), otherwise nothing!	
X: ≥n.R.D	[R(X,Y)]? → g(Y:D), otherwise nothing!	
X: ≤n.R.D	[R(X,Y)]? → g(Y:D), otherwise nothing!	

Example

Grandmother ≡ Mother ∩ ∃child:Parent
Grandmother ⊆ ∃married-with:Grandfather

f,g

[X:Grandmother]! ← [X:Mother]? ⊕ [child(X,Y)]? ⊕ [Y:Parent]?

[X:Grandmother]! → [married-with(X,Y)]? ⊕ [Y:Grandfather]!

	ABOX	"Decision Tree"
0		[X:Grandmother]! ← [X:Mother]? ⊕ [child(X,Y)]? ⊕ [Y:Parent]? [X:Grandmother]! → [married-with(X,Y)]? ⊕ [Y:Grandfather]!
1	Anja:Mother	[X:Grandmother]! ← [X:Mother]? ⊕ [child(X,Y)]? ⊕ [Y:Parent]? [X:Grandmother]! → [married-with(X,Y)]? ⊕ [Y:Grandfather]!
2	Anja:Mother, married-with(Anja, Knut)	[X:Grandmother]! ← [X:Mother]? ⊕ [child(X,Y)]? ⊕ [Y:Parent]? [X:Grandmother]! → [married-with(X,Y)]? ⊕ [Y:Grandfather]!
3	Anja:Mother, married-with(Anja, Knut), child(Anja, Nils)	[X:Grandmother]! ← [X:Mother]? ⊕ [child(X,Y)]? ⊕ [Y:Parent]? [X:Grandmother]! → [married-with(X,Y)]? ⊕ [Y:Grandfather]!
4	Anja:Mother, married-with(Anja, Knut), child(Anja, Nils), Nils:Father	[X:Grandmother]! ← [X:Mother]? ⊕ [child(X,Y)]? ⊕ [Y:Parent]? [X:Grandmother]! → [married-with(X,Y)]? ⊕ [Y:Grandfather]!
5	Anja:GrandMother, married-with(Anja, Knut), child(Anja, Nils), Nils:Father	[X:Grandmother]! ← [X:Mother]? ⊕ [child(X,Y)]? ⊕ [Y:Parent]? [X:Grandmother]! → [married-with(X,Y)]? ⊕ [Y:Grandfather]!
6	Anja:GrandMother, married-with(Anja, Knut), child(Anja, Nils), Nils:Father, Knut:Grandfather	

TODO

- Concrete Domains
  - Seems to be simple because restricted to tests about integers and strings ?
- Roles
  - Hierarchy, Domain, Range
  - Inverse Relations
- Inconsistence checking possible with dummy (skolem) individuals?

Thank you for your attention

Requirements for "Decision Tree"

- Formalism for representing the decision tree
  - More powerful than normal decision trees?!
- Difference Operator for extracting the difference between parent C and child D