22 nov 2021

**Required Expressivity**

Propositional logic and role quantifications provide basic expressivity

ALC is propositional logic with quantifiers (existential) or value restriction

But it is not always enough for a given application

In real life applications, we want to represent some additional properties

Description logics (DL) provide different new constructors available through a “mix-and-match” approach. More stuff, more slow program and another disadvantage is that that can be more way to express the same thing and make it more confusing

Some of this constructors, some for concepts, some for roles and additional axiom (constraints) that we can add to the KB

**Nominals *O* (symbol used for represent the nominal constructor)**

A nominal is a concept with exactly one element which is named

A concept: set of objects of the domain

Nominal: special concept that have one individual (speaking about want constant/element of the domain)

Syntax: {a}; a ∈ Ni

Semantics: ({a})i = {ai}

The interpretation of the single concept is the single concept that have the interpretation of that name

Interpretation maps individual name to objects of the domain

Examples: can do things that are not totally obvious

• EUCountry ⊑ {austria} ⊔ {belgium} ⊔ . . . ⊔ {sweden}

Concept of the consultancy of EU. This concept has to be contained in the set of constants.

Any country that is not in the list should not be an EU country (not so easy just with assertions)

With assertions will be

EUCountry(Austria)

EUCountry(Belgium)

…

T - {Austria,Belgium…} ⊑ ¬ EUCountry any other constant should not be an EU country

No other country outside the list is an european country

• French ⊑ ∃hasPresident.{macron}

Every french person have as president macron

We can express the ABox and much more

A(b) = {b} ⊑ A

*The concept A of b is equivalent to the GCI nominal b is a subconcept of the object A*

r(a,b) = {a} ⊑ ∃ r.{b}

**Cardinality Constraints Q, N**

2 variants Q, N

N: unqualified number restriction

Q: qualified number restrictions

N: always have T

A cardinality constraint limits the number of successors of an object

Limits how many successor an object can have

Syntax: ≥ n r.C; n ∈ N, r ∈ NR, C a concept

Semantics: (≥ n r.C)i ={δ | # { η∈Ci |(δ,η)∈ri } ≥ n}

Examples:

• {bai} ⊑ ¬(≥ 181hasStudent.⊤)

At most 181 students

• ≥ 3proficientIn.Language ⊑ Polyglot

If someone speak 3 or more languages is a polyglot

count the restriction

**Inverse Roles I**

Inverse roles allow to traverse the role relations backwards

Syntax: r-,r ∈ NR

Semantics: (r -)I = {(δ,η) | (η,δ) ∈ ri}

Examples:

• Male ⊓ ∃hasChild -.⊤ ⊑ Son

• ⊤ ⊑ ¬(≥ 3hasChild - .⊤)

No person can have more than 2 parents (inverse relation of child)

**Self**

The self constructor expresses a local reflexivity

Special operator that connect to itself

Syntax: ∃r.Self Semantics (∃r .Self )i = {δ | (δ, δ) ∈ ri }

the object is connected to itself through an r successor

Examples

• ⊤ ⊑ ¬(∃hasSibling.Self )

A person can not have itself as a sibilin

Sometimes we want to exclude some local expecity

• Freelance ⊑ ∃hasBoss.Self

Sometimes we want to make it positive

**Transitivity S**

A transitivity axiom restricts the interpretation of a role to be transitive

Axiom: trans(r),r ∈ NR I satisfies it iff ri is transitive

Example

• trans(hasDescendant)

A--hasDescendant-->B--hasDescendant-->C

than A--hasDescendant-->C

hasDescendant is transitive

• trans(hasPart)

If an object has a part and that part has a part than the last part is part of the object

**Role Hierarchies H**

Role inclusions are like GCIs but for roles express that one is a sub-role of another

Role inclusion

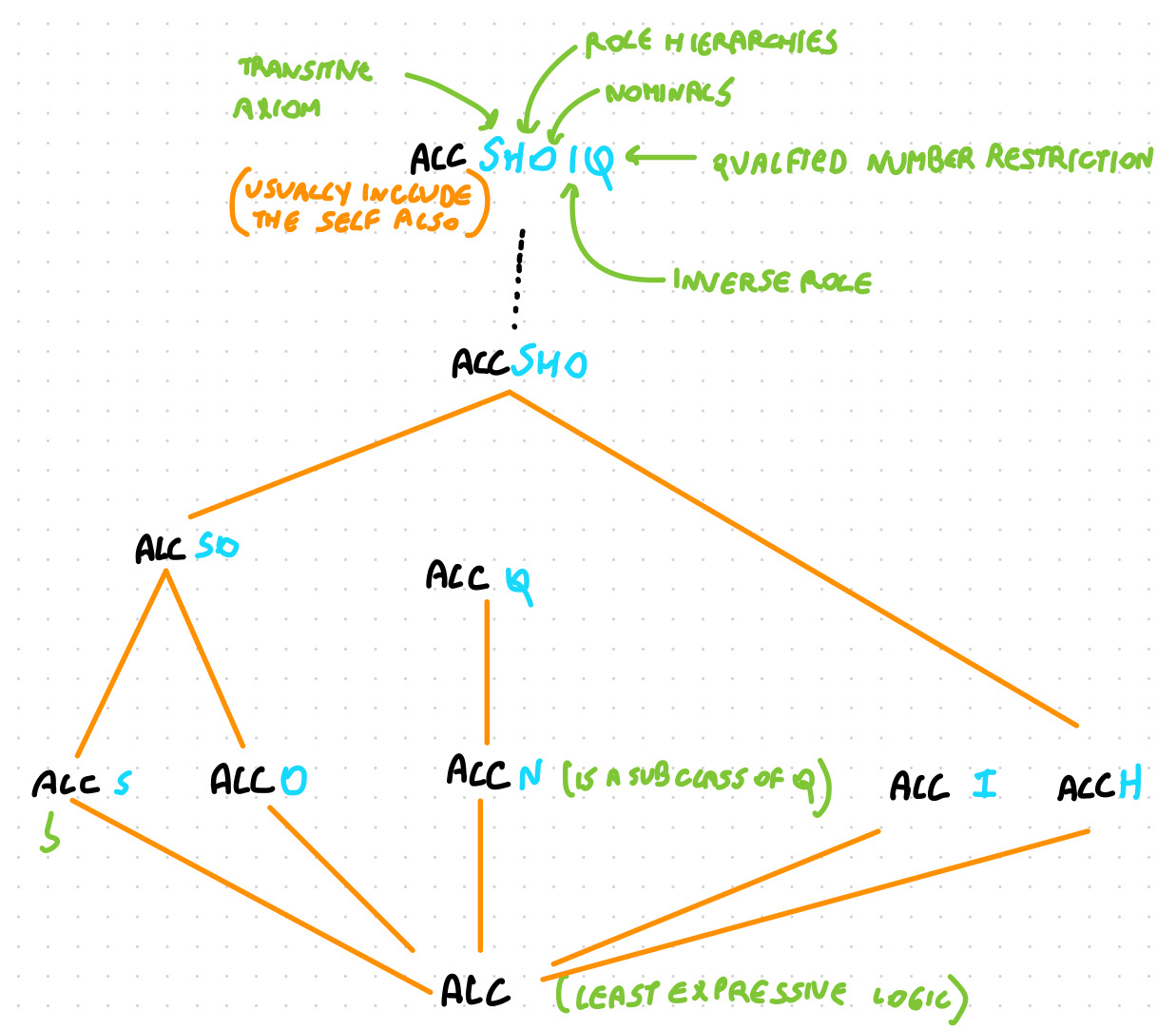
Axiom: r ⊑ s, r, s∈NR I satisfies it iff ri ⊆si

**Language Hierarchies**

different constructs: O, Q, N, I, SELF, S, H

The different constructors define a hierarchy of languages of varying expressivity and complexity

ALC: least expressive



try to keep it as low as possible. easier and faster, control less cases

Easier to implement if you have to control less cases

If we get to SHOIQ, we can't guarantee there is a finite model, we can build KB that are consistent but only have infinite models