Knowledge-Based Systems

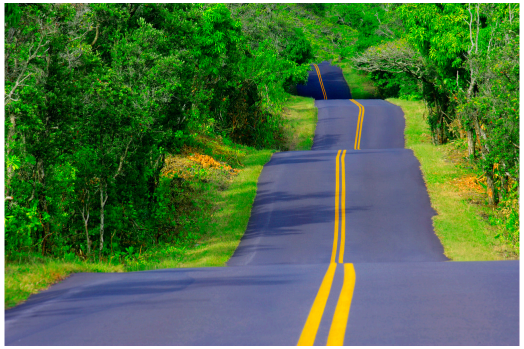
Description Logic Reasoning - Why and how did that happen (II)

Jeff Z. Pan

http://homepages.abdn.ac.uk/jeff.z.pan/pages/

UNIVERSITY OF ABERDEEN

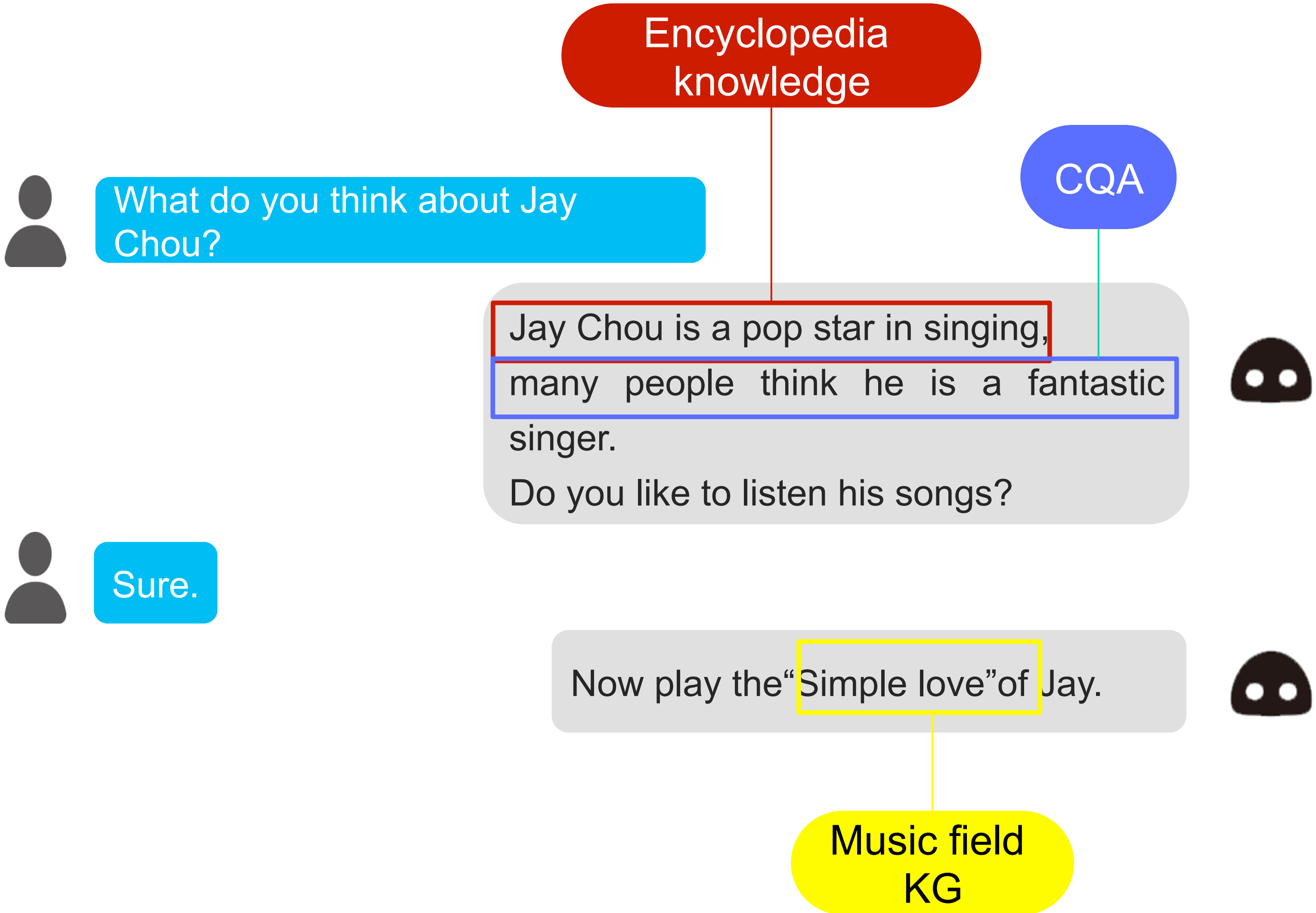
Roadmap



- Foundation
  - KR, ontology and rule; set theory
- Knowledge capture
- Knowledge representation
  - Ontology: Semantic Web standards RDF and OWL, Description Logics
  - Rule: Jess
- Knowledge reasoning
  - Ontology: formal semantics, tableaux algorithm
  - Rule: forward chaining, backward chaining
- Knowledge reuse and evaluation
- Meeting the real world
  - Jess and Java, Uncertainty, Invited talk

Microsoft CEO: Chatbots will 'fundamentally revolutionize' computing

Application of chatbot QA based on massive KB merging



Application of chatbot QA based on inconsistency checking in KB

Your friend Emily has just arrived

No, that’s Emma, not Emily

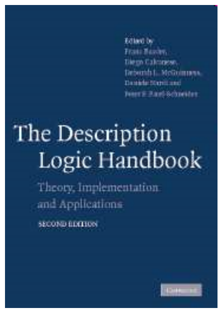
Ah, that’s Emma, but I remember Emma only has cats.
I see a dog with her now.

That’s not her dog, but Lily’s.

I see, the girl next to Emma is Lily.

Lecture Outline

- Motivation
- Introduction to tableaux algorithms
- Some detailed discussions on tableaux algorithms
- Practical



[Section: 9.3.2.1]

Motivations:

- How to perform DL reasoning based on formal semantics
  – The first sound and complete algorithm for expressive DLs
- So far
  – we only introduced one expansion rule
  – we only allows simply class axioms

Expansion Rule for Simple Axioms

- Simple axioms
  - $A \sqsubseteq C$ where A is a name class
  - No cycles involve A
    - ✘ such as $A \sqsubseteq \exists R.A$
- **Expansion rule for simple axioms**
  - If A is in L(x) and $A \sqsubseteq C$ is in O
  - Then add C into L(x)

How about Class Descriptions

- Given the following ontology
  1. MadCow $\sqsubseteq$ Herbivore
  2. MadCow $\sqsubseteq \exists$eat.SheepBrain
  3. Herbivore $\sqsubseteq \forall$eat.Plant
  4. SheepBrain $\sqsubseteq \neg$Plant
  5. MadCow(mc)

- Q: Can you use the tableaux algorithm to check if the above ontology is consistent

Lecture Outline

- Motivation
- Introduction to more expansion rules in tableaux algorithms
  - The big picture
- Some detailed discussions on tableaux algorithms
- Practical

Expansion Rules for Some Constructors

$x \bullet \{C_1 \sqcap C_2, \ldots\}$	$\rightarrow_\sqcap$	$x \bullet \{C_1 \sqcap C_2, C_1, C_2, \ldots\}$
$x \bullet \{C_1 \sqcup C_2, \ldots\}$	$\rightarrow_\sqcup$	$x \bullet \{C_1 \sqcup C_2, C, \ldots\}$ for $C \in \{C_1, C_2\}$
$x \bullet \{\exists R.C, \ldots\}$	$\rightarrow_\exists$	$x \bullet \{\exists R.C, \ldots\}$ $R \downarrow$ $y \bullet \{C\}$
$x \bullet \{\forall R.C, \ldots\}$ $R \downarrow$ $y \bullet \{\ldots\}$	$\rightarrow_\forall$	$x \bullet \{\forall R.C, \ldots\}$ $R \downarrow$ $y \bullet \{C, \ldots\}$

UNIVERSITY OF ABERDEEN

Expansion Rules for Some Constructors

$x \bullet \{C_1 \sqcap C_2, \ldots\}$	$\rightarrow_\sqcap$	$x \bullet \{C_1 \sqcap C_2, C_1, C_2, \ldots\}$
$x \bullet \{C_1 \sqcup C_2, \ldots\}$	$\rightarrow_\sqcup$	$x \bullet \{C_1 \sqcup C$
$x \bullet \{\exists R.C, \ldots\}$	$\rightarrow_\exists$	$x \bullet$ R $y \bullet \{$
$x \bullet \{\forall R.C, \ldots\}$ R $y \bullet \{\ldots\}$	$\rightarrow_\forall$	$x \bullet \{\forall R.C, \ldots\}$ R $y \bullet \{C, \ldots\}$

If x is an instance of C1 ⊓C2,
Then x must be an instance of C1, and x must be an instance of C2

Expansion Rules for Some Constructors

$x \bullet \{C_1 \sqcap C_2, \ldots\}$	$\rightarrow_{\sqcap}$	$x \bullet \{C_1 \sqcap C_2, C_1, C_2, \ldots\}$
$x \bullet \{C_1 \sqcup C_2, \ldots\}$	$\rightarrow_{\sqcup}$	$x \bullet \{C_1 \sqcup C_2, C, \ldots\}$ for $C \in \{C_1, C_2\}$
$x \bullet \{\exists R.C, \ldots\}$	$\rightarrow_{\exists}$	
$x \bullet \{\forall R.C, \ldots\}$ R $y \bullet \{\ldots\}$		$y \bullet \{C, \ldots\}$

If x is an instance of C1 ⊔ C2,
Then x must be either an instance of C1, or x is an instance of C2

Expansion Rules for Some Constructors

$x \bullet \{C_1 \sqcap C_2, \ldots\}$	$\rightarrow_{\sqcap}$	$x \bullet \{C_1 \sqcap C_2, C_1, C_2, \ldots\}$
$x \bullet \{C_1 \sqcup C_2, \ldots\}$	$\rightarrow_{\sqcup}$	$x \bullet \{C_1 \sqcup C_2, C, \ldots\}$ for $C \in \{C_1, C_2\}$
$x \bullet \{\exists R.C, \ldots\}$	$\rightarrow_{\exists}$	$x \bullet \{\exists R.C, \ldots\}$ R $y \bullet \{C\}$
$x \bullet \{\forall R.C,$ R $y \bullet$		

If x is an instance of ∃R.C,
Then x must have an R relation to an instance y of C; hence, we create a new R-successor y that is labeled C

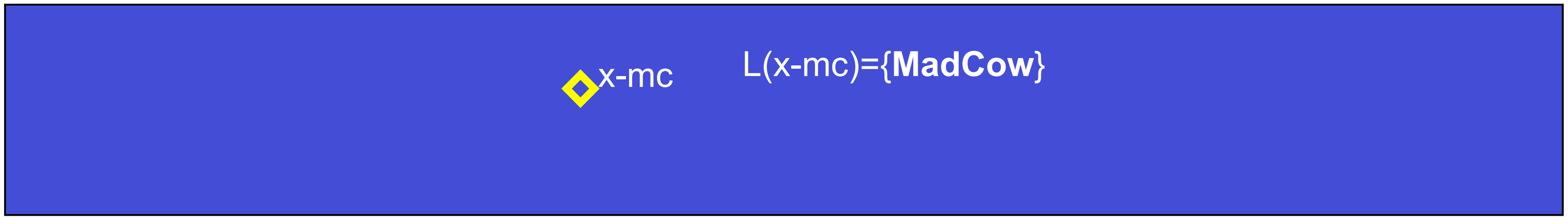
Expansion Rules for Some Constructors

$x \bullet \{C_1 \sqcap C_2, \ldots\}$	$\rightarrow_{\sqcap}$	$x \bullet \{C_1 \sqcap C_2, C_1, C_2, \ldots\}$
$x \bullet \{C_1 \sqcup C_2, \ldots\}$	$\rightarrow_{\sqcup}$	$x \bullet \{C_1 \sqcup C_2, C, \ldots\}$
$x \bullet \{\exists$		
$x \bullet \{\forall R.C, \ldots\}$ R $y \bullet \{\ldots\}$	$\rightarrow_{\forall}$	$x \bullet \{\forall R.C, \ldots\}$ R $y \bullet \{C, \ldots\}$

If x is an instance of ∀R.C,
Then every R-successor y of x must be an instance of C; hence, we add C into the label of all y

Example: Using Expansion Rules

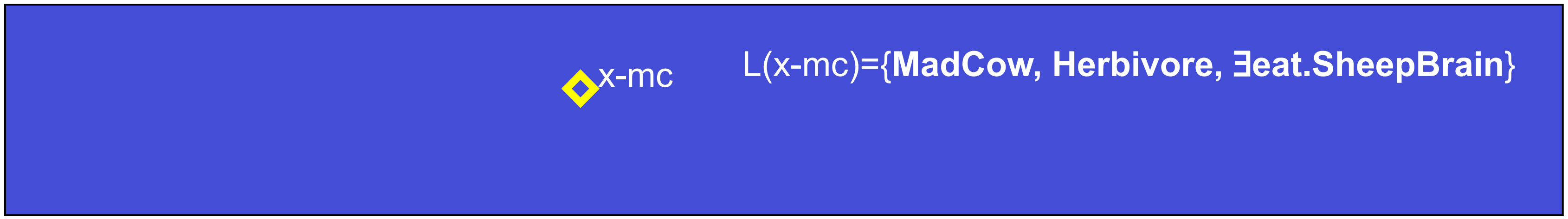
- Check if the following ontology is consistent
  1. MadCow $\sqsubseteq$ Herbivore
  2. MadCow $\sqsubseteq \exists$eat.SheepBrain
  3. Herbivore $\sqsubseteq \forall$eat.Plant
  4. SheepBrain $\sqsubseteq \neg$ Plant
  5. MadCow(mc)



6. Initialise the tableau: L(x-mc1)={**MadCow**}

Example: Using Expansion Rules

- Check if the following ontology is consistent
  1. MadCow $\sqsubseteq$ Herbivore
  2. MadCow $\sqsubseteq$ ∃eat.SheepBrain
  3. Herbivore $\sqsubseteq$ ∀eat.Plant
  4. SheepBrain $\sqsubseteq$ ¬Plant
  5. MadCow(mc)
  6. Initialise the tableau: L(x-mc)={MadCow}



7. Expand L(x-mc): L(x-mc)={MadCow**, Herbivore, ∃eat.SheepBrain**} //simple expansion rule on axioms 1 and 2)

UNIVERSITY OF ABERDEEN

Example: Using Expansion Rules

- Check if the following ontology is consistent
  1. MadCow $\sqsubseteq$ Herbivore
  2. MadCow $\sqsubseteq \exists$eat.SheepBrain
  3. Herbivore $\sqsubseteq \forall$eat.Plant
  4. SheepBrain $\sqsubseteq \neg$Plant
  5. MadCow(mc)
  6. Initialise the tableau: L(x-mc)={MadCow}
  7. Expand L(x-mc): L(x-mc)={MadCow, Herbivore, $\exists$eat.SheepBrain}

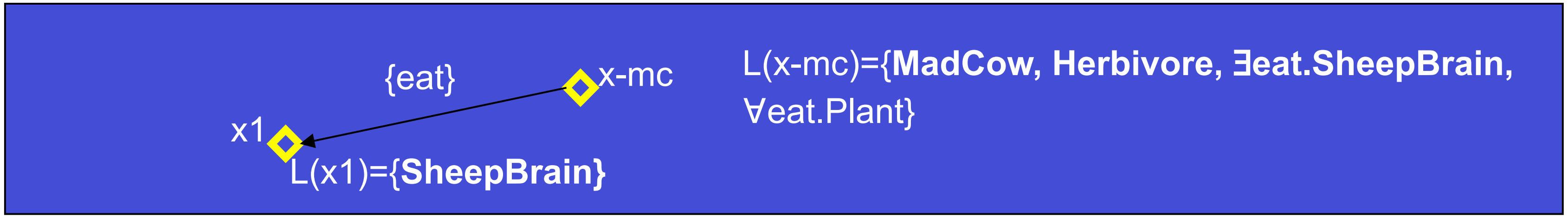
x-mc L(x-mc)={**MadCow, Herbivore, $\exists$eat.SheepBrain,** $\forall$eat.Plant}

8. Expand L(x-mc): L(x-mc)={MadCow, Herbivore, $\exists$eat.SheepBrain, **$\forall$eat.Plant**} //simple expansion rule on axiom 3)

Example: Using Expansion Rules

- Check if the following ontology is consistent

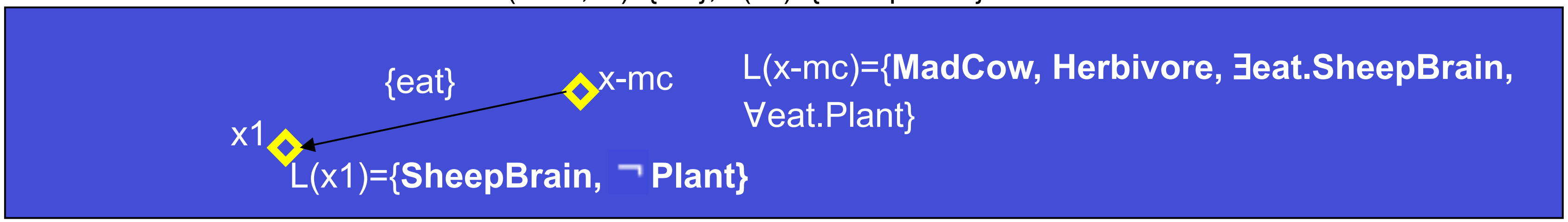
1. MadCow $\sqsubseteq$ Herbivore
2. MadCow $\sqsubseteq \exists$eat.SheepBrain
3. Herbivore $\sqsubseteq \forall$eat.Plant
4. SheepBrain $\sqsubseteq \neg$ Plant
5. MadCow(mc)
6. Initialise the tableau: L(x-mc)={MadCow}
7. Expand L(x-mc): L(x-mc)={MadCow, Herbivore, ∃eat.SheepBrain}
8. Expand L(x-mc): L(x-mc)={MadCow, Herbivore, ∃eat.SheepBrain, ∀eat.Plant}



9. Create node x1. L(x-mc,x1)={**eat**}, L(x1)={**SheepBrain**} //∃-expansion rule on x-mc

Example: Using Expansion Rules

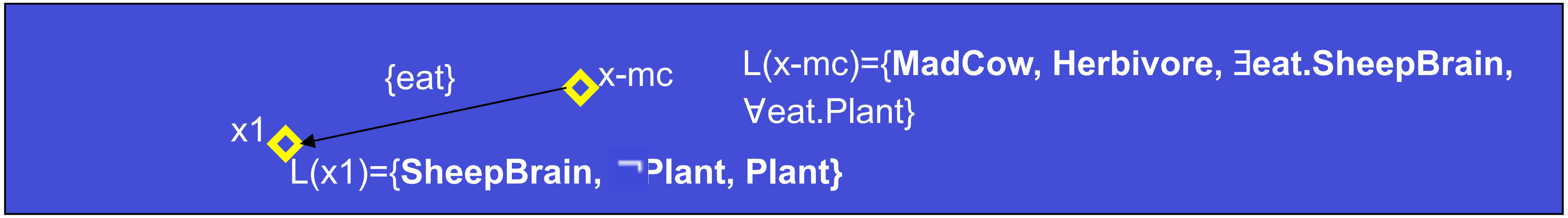
- Check if the following ontology is consistent
  1. MadCow ⊑ Herbivore
  2. MadCow ⊑ ∃eat.SheepBrain
  3. Herbivore ⊑ ∀eat.Plant
  4. SheepBrain ⊑ ¬Plant
  5. MadCow(mc)
  6. Initialise the tableau: L(x-mc)={MadCow}
  7. Expand L(x-mc): L(x-mc)={MadCow, Herbivore, ∃eat.SheepBrain}
  8. Expand L(x-mc): L(x-mc)={MadCow, Herbivore, ∃eat.SheepBrain, ∀eat.Plant}
  9. Create node x1. L(x-mc,x1)={eat}, L(x1)={SheepBrain}



10. Expand L(x1). L(x1)={SheepBrain,¬ **Plant**} //simple expansion rule on axiom 4

Example: Using Expansion Rules

- Check if the following ontology is consistent
  1. MadCow ⊑ Herbivore
  2. MadCow ⊑ ∃eat.SheepBrain
  3. Herbivore ⊑ ∀eat.Plant
  4. SheepBrain ⊑ ¬ Plant
  5. MadCow(mc)
  6. Initialise the tableau: L(x-mc)={MadCow}
  7. Expand L(x-mc): L(x-mc)={MadCow, Herbivore, ∃eat.SheepBrain}
  8. Expand L(x-mc): L(x-mc)={MadCow, Herbivore, ∃eat.SheepBrain, ∀eat.Plant}
  9. Create node x1. L(x-mc,x1)={eat}, L(x1)={SheepBrain}
  10. Expand L(x1). L(x1)={SheepBrain, ¬ **Plant**}



11. Expand L(x1). L(x1)={SheepBrain, ¬ Plant, **Plant**} //∀-expansion rule on x-mc
12. There is a contradiction, so the ontology is inconsistent

Lecture Outline

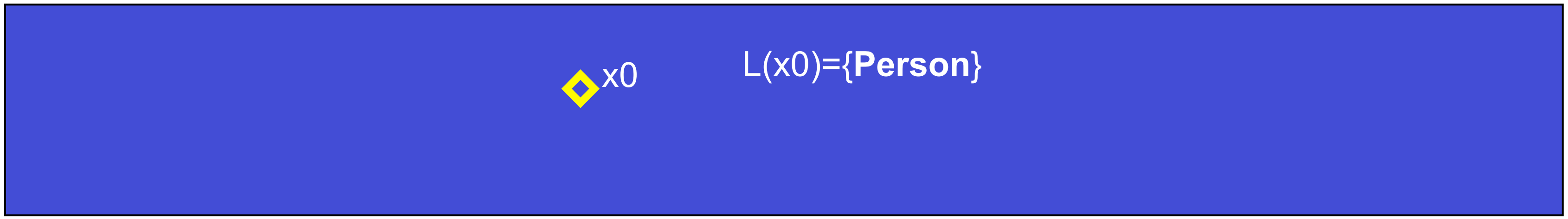
- Motivation
- Introduction to tableaux algorithms
- Some detailed discussions on tableaux algorithms
- Practical

Blocking: Ensuring Termination

- Expansion can be applicable forever
  - We need to block the expansion on e.g. cyclic axioms
- Blocking
  - Condition: $L(y) \subseteq L(x)$ for some ancestor x (blocking node) and predecessor y (blocked node)
  - Intuitively, this means that the same constraints have been dealt with before

Example: Blocking

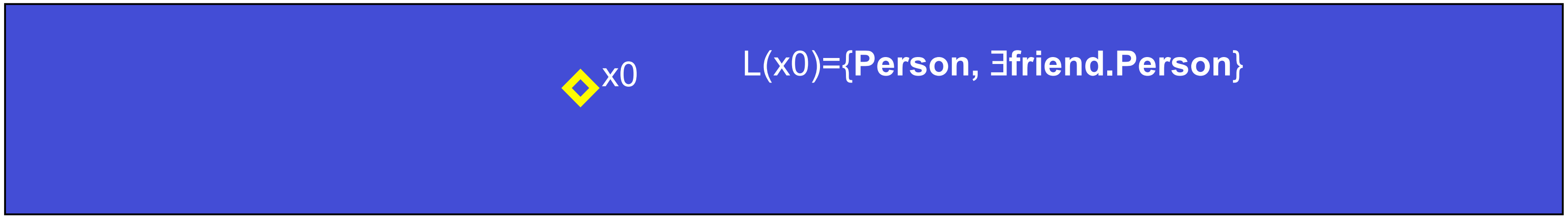
- Example:
  - Given the ontology
    1. Person $\sqsubseteq$ ∃friend.Person
  - Check if Person is satisfiable
- Construct a tableau
  2. Initialise the tableau: L(x0)={Person}



Example: Blocking

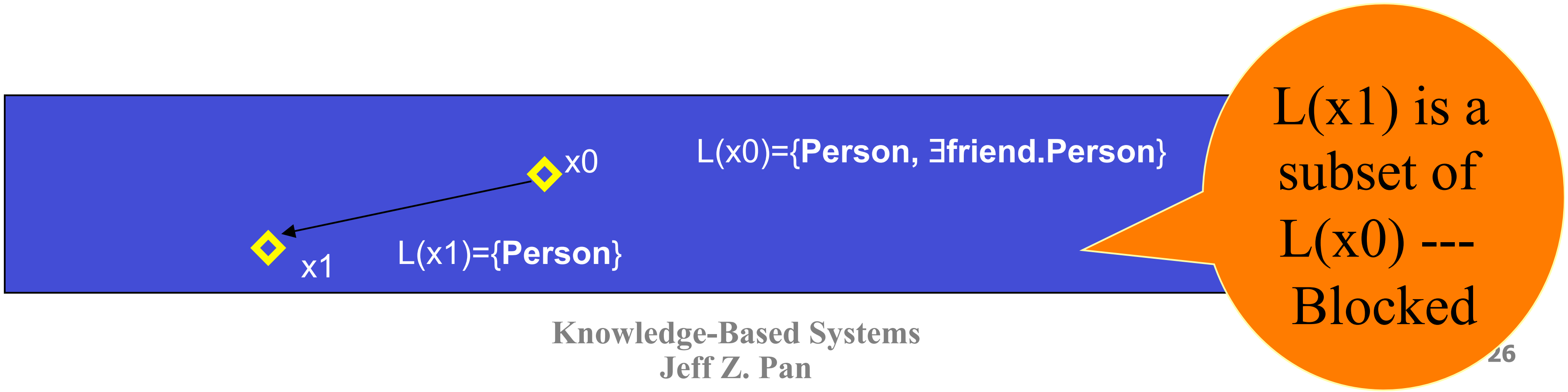
- Example:
  - Given the ontology
    1. Person $\sqsubseteq$ ∃friend.Person
  - Check if Person is satisfiable
- Construct a tableau

2. Initialise the tableau: L(x0)={Person}
3. L(x0)={Person, ∃friend.Person} //simple axiom expansion



Example: Blocking

- Example:
  - Given the ontology
    1. Person $\sqsubseteq$ ∃friend.Person
  - Check if Person is satisfiable
- Construct a tableau
  2. Initialise the tableau: L(x0)={Person}
  3. L(x0)={Person, ∃friend.Person} //simple axiom expansion
  4. L(x1)={Person}, L(x0,x1)=friend //∃-expansion



UNIVERSITY OF ABERDEEN

Tableau and Interpretation

- Tableau
  - $L(x0)=$ {Person, $\exists$friend.Person}
  - $L(x0,x1)=$ {friend}, $L(x1)=$ {Person}
- We can construct an interpretation: Note that blocked nodes are not included in the interpretation
  - Δ^I ={x0}
  - $Person^I$ = {x0}
  - $friend^I$ ={<x0,x0>}

Example: Class Subsumption Checking

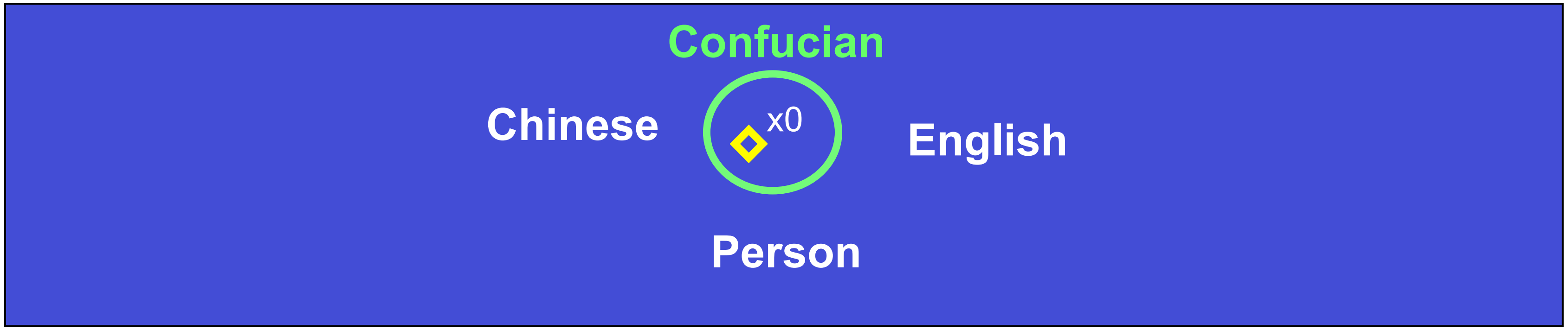
- Question: Given the following ontology O,
  - Class (Chinese partial Person)
  - Class (English partial Person)
  - Class (Confucian partial Chinese)
  - Class (Confucian partial English)
- O |= Confucian $\sqsubseteq$ Person?

Example: Class Subsumption Checking

- Check if Confucian $\sqsubseteq$ Person
  1. Chinese $\sqsubseteq$ Person
  2. English $\sqsubseteq$ Person
  3. Confucian $\sqsubseteq$ Chinese $\sqcap$ English

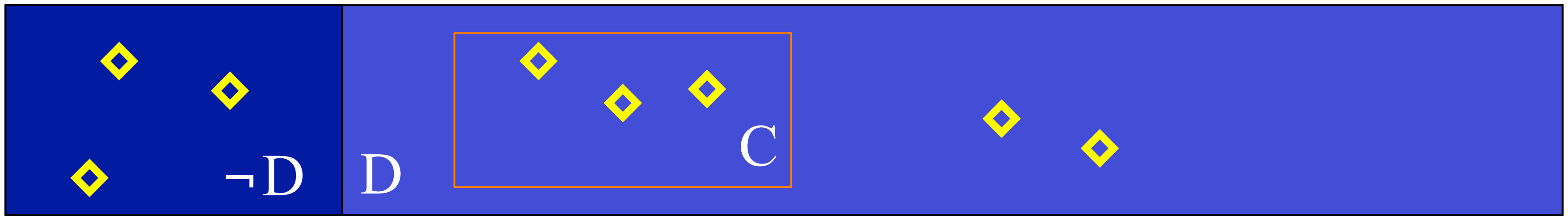
Can we simply build an interpretation?

No.



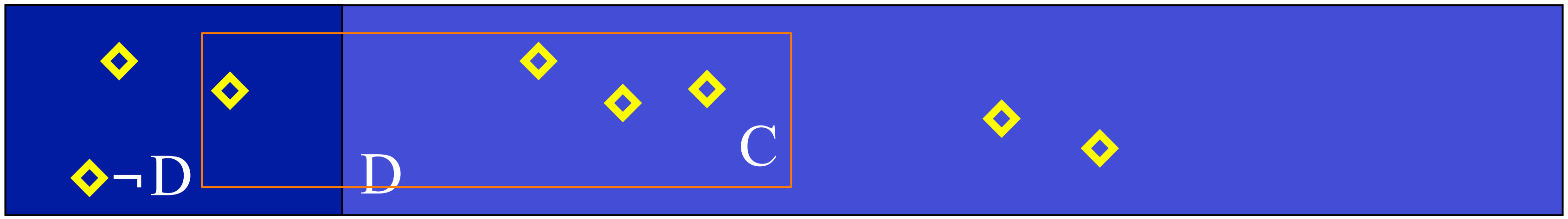
From Subsumption Checking to Class Unsatisfiability Checking

- Some reasoning services can be reduced to each other
- E.g. class subsumption checking to class (un)satisfiability checking
  - $O \models C \sqsubseteq D$ iff $C \sqcap \neg D$ is unsatisfiable ?



From Subsumption Checking to Class Unsatisfiability Checking

- Some reasoning services can be reduced to each other
- E.g. class subsumption checking to class (un)satisfiability checking
  - O |= $C \sqsubseteq D$ iff $C \sqcap \neg D$ is unsatisfiable ?

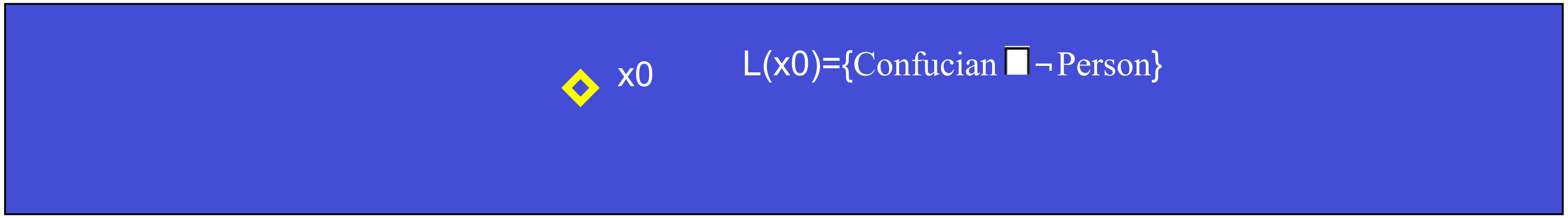


Example: Class SubsumptionChecking

- Question: Given the following ontology O,
  - Class (Chinese partial Person)
  - Class (English partial Person)
  - Class (Confucian partial Chinese)
  - Class (Confucian partial English)
- To check O |= Confucian $\sqsubseteq$ Person, we need to check if (Confucian $\sqcap \neg$Person) is unsatisfiable

Example: Class Subsumption Checking

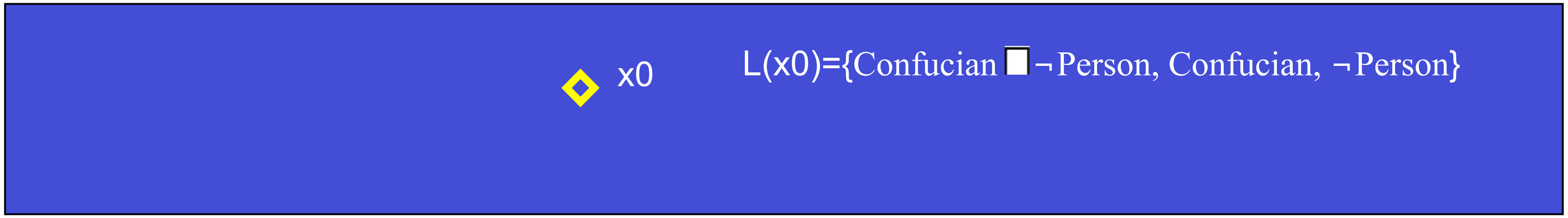
- Check if Confucian $\sqsubseteq$ Person
  1. Chinese $\sqsubseteq$ Person
  2. English $\sqsubseteq$ Person
  3. Confucian $\sqsubseteq$ Chinese $\sqcap$ English



4. Initialise the tableau: L(x0)={**Confucian** $\sqcap$ **¬Person**}

Example: Class Subsumption Checking

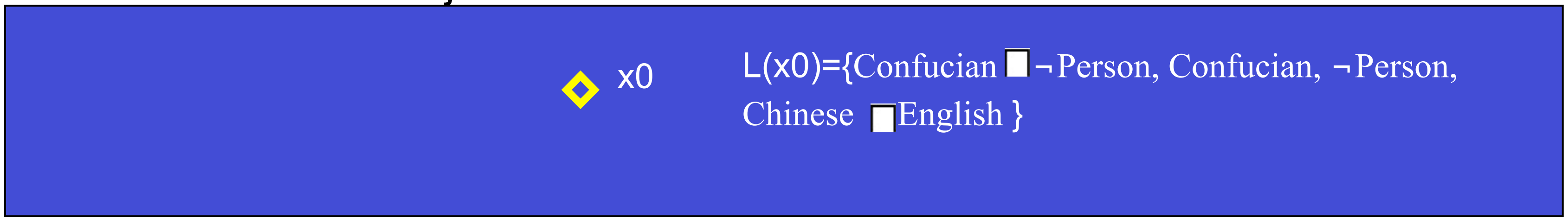
- Check if Confucian ⊑ Person
  1. Chinese ⊑ Person
  2. English ⊑ Person
  3. Confucian ⊑ Chinese ⊓ English
  4. Initialise the tableau: L(x0)={Confucian ⊓¬Person}



5. Expand L(x0): L(x0)={Confucian ⊓ ¬Person, **Confucian, ¬Person**}// ⊓-expansion rule

Example: Class Subsumption Checking

- Check if Confucian ⊑ Person
  1. Chinese ⊑ Person
  2. English ⊑ Person
  3. Confucian ⊑ Chinese ⊓ English
  4. Initialise the tableau: L(x0)={Confucian ⊓¬Person}
  5. Expand L(x0): L(x0)={Confucian ⊓ ¬Person, Confucian, ¬Person}



6. Expand L(x0): L(x0)={Confucian ⊓ ¬Person, Confucian, ¬Person, **Chinese ⊓English** }//simple expansion rule on axiom 3

UNIVERSITY OF ABERDEEN

Example: Class Subsumption Checking

- Check if Confucian Person
  1. Chinese ⊑ Person
  2. English ⊑ Person
  3. Confucian ⊑ Chinese ⊓ English
  4. Initialise the tableau: L(x0)={Confucian ⊓ ¬Person}
  5. Expand L(x0): L(x0)={Confucian ⊓ ¬Person, Confucian, ¬Person}
  6. Expand L(x0): L(x0)={Confucian ⊓ ¬Person, Confucian, ¬Person, Chinese ⊓ English }

x0 L(x0)={Confucian ⊓ ¬Person, Confucian, ¬Person, Chinese ⊓ English, Chinese, English }

7. Expand L(x0): L(x0)={Confucian ⊓ ¬Person, Confucian, ¬Person, Chinese ⊓ English, **Chinese, English** }// ⊓-expansion rule on axiom 3

Example: Class Subsumption Checking

- Check if Confucian Person

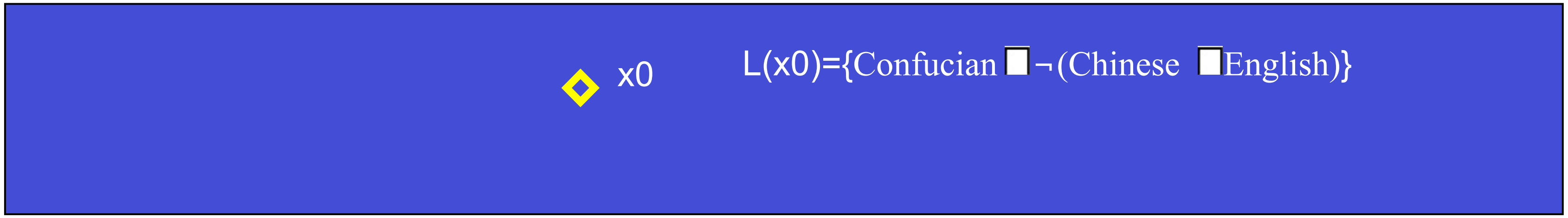
1. Chinese $\sqsubseteq$ Person
2. English $\sqsubseteq$ Person
3. Confucian $\sqsubseteq$ Chinese $\sqcap$ English
4. Initialise the tableau: L(x0)={Confucian $\sqcap$ ¬Person}
5. Expand L(x0): L(x0)={Confucian $\sqcap$ ¬Person, Confucian, ¬Person}
6. Expand L(x0): L(x0)={Confucian $\sqcap$ ¬Person, Confucian, ¬Person, Chinese $\sqcap$ English }
7. Expand L(x0): L(x0)={Confucian $\sqcap$ ¬Person, Confucian, ¬Person, Chinese $\sqcap$ English, Chinese, English }

x0 L(x0)={Confucian $\sqcap$ ¬Person, Confucian, ¬**Person**, Chinese $\sqcap$ English, Chinese, English, **Person** }

7. Expand L(x0): L(x0)={Confucian $\sqcap$ ¬Person, Confucian, ¬**Person**, Chinese $\sqcap$ English, Chinese, English, **Person** }// simple expansion rule on axiom 3. There is a contradiction, so Confucian $\sqcap$ ¬Person is **unsatisfiable**.
8. So Confucian $\sqsubseteq$ Person is **true**

Example: Class Subsumption Checking

- Check if Confucian $\sqsubseteq$ Chinese $\sqcap$ English
  1. Chinese $\sqsubseteq$ Person
  2. English $\sqsubseteq$ Person
  3. Confucian $\sqsubseteq$ Chinese $\sqcap$ English



- Now the question is how do we handle ¬ (Chinese $\sqcap$ English)

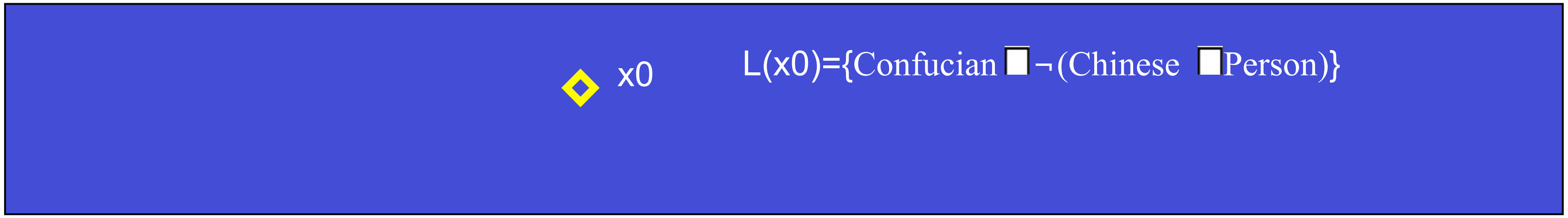
NNF: Negated Normal Form

- Negated Normal Form (NNF)
  - If a class is in NNF, negations only appear in front of named classes
  - E.g., ¬Person is in NNF
  - However, ¬(Chinese ⊓ English) is not in NNF
- In tableau algorithm, all the input classes should be in NNF
  - We can make use of the following table to transform inputs into NNF

$$
\begin{aligned}
\neg\neg C &\equiv C \\
\neg(C \sqcap D) &\equiv \neg D \sqcup \neg C \\
\neg(C \sqcup D) &\equiv \neg D \sqcap \neg C
\end{aligned}
\qquad
\begin{aligned}
\neg\exists r.C &\equiv \forall r.\neg C \\
\neg\forall r.C &\equiv \exists r.\neg C \\
\neg\leqslant nr.C &\equiv \geqslant(n+1)r.C \\
\neg\geqslant(n+1)r.C &\equiv \leqslant nr.C
\end{aligned}
$$

Example: Class Subsumption Checking

- Check if Confucian $\sqsubseteq$ Chinese $\sqcap$ English
    1. Chinese $\sqsubseteq$ Person
    2. English $\sqsubseteq$ Person
    3. Confucian $\sqsubseteq$ Chinese $\sqcap$ English



- Now the question is how do we handle $\neg$ (Chinese $\sqcap$ English)
- According to the table in the previous slide, it is equivalent to
    - $\neg$Chinese $\sqcup$ $\neg$Person
    - Q: Can you try to finish this subsumption checking?

Class Instance Checking

- Question: given the following ontology O,
  - $OldLady \sqsubseteq \forall hasPet.Cat$
  - OldLady(Minnie)
  - hasPet(Minnie, Tom)

- Does O entail Individual (Tom type (Cat)) ?

Practical

- Ontology reasoning in tableau algorithm

Summary

- More expansion rules for the tableau algorithm
  - cyclic axioms
  - class descriptions
  - reasoning tasks reduction
    - class instance checking
    - class subsumption checking