A Resolution Decision Procedure for SHOIQ

Yevgeny Kazakov and Boris Motik

The University of Manchester

August 20, 2006
SHOIQ is a Description Logic!
DESCRIPTION LOGICS:

- a family language for knowledge representation:

```
HappyFather ≡ Human ⊓ (⩾ 2 hasChild) ⊓
               ⊓ ∀ hasChild. (Famous ⊓ Rich)
```

Distinguished by:
- Formal semantics (set-theoretic)
- Decidability for key reasoning problems (satisfiability, subsumption, instance)

Related to:
- (Multi-) Modal Logics, Dynamic Logics
- Fragments of First-Order Logic (guarded, two-variable)
DESCRIPTION LOGICS:

- a family language for knowledge representation:

\[
\text{HappyFather} \equiv \text{Human} \sqcap (\geq 2 \text{hasChild}) \sqcap \\
\quad \sqcap \forall \text{hasChild}.(\text{Famous} \sqcup \text{Rich})
\]

- Distinguished by:
  - Formal semantics (set-theoretic)
  - Decidability for key reasoning problems (satisfiability, subsumption, instance)
DESCRIPTION LOGICS:

- a family language for knowledge representation:

```
HappyFather ≡ Human ⊓ (⩾ 2 hasChild) ⊓ ⊓ ∀hasChild.(Famous □ Rich)
```

- Distinguished by:
  - Formal semantics (set-theoretic)
  - Decidability for key reasoning problems (satisfiability, subsumption, instance)

- Related to:
  - (Multi-) Modal Logics, Dynamic Logics
  - Fragments of First-Order Logic (guarded, two-variable)
APPLICATION OF DESCRIPTION LOGICS

- Databases (Schema Integration)
- Ontologies (Knowledge Bases):
  - Rigorous description of terms in specific domains (Anatomy, Food, Cars)
  - Access information by performing queries:

\[ \text{?– Car} \sqsubseteq \exists \text{hasTransmission}\ .\text{Automatic} \sqsubseteq \]
\[ \sqsubseteq \exists \text{hasPart}\ .\left(\text{Engine} \sqsubseteq (\geq 6 \text{hasPart}\ .\text{Cylinder})\right) \]

- Semantic Web:
  - Ontology Web Language OWL (W3C standard)
  - Annotation of entries using "semantic" mark-up
  - Provide "the meaning" of entries

WHAT IS IT ABOUT SHOIQ?
**What is it About SHOIQ?**

- **DL SHOIQ** is a logical counterpart of **OWL DL**
- Development of **OWL DL**-ontologies requires **reasoning**:
  - computation of class trees (Heart ⊆ Organ)
  - evaluation of queries (\(?–\) Car ⊑ . . .)
- reasoning (**OWL**) = theorem proving (**SHOIQ**)

---

Roger F. Streater

The University of Manchester
What is it About SHOIQ?

- DL **SHOIQ** is a logical counterpart of **OWL DL**
- Development of **OWL DL**-ontologies requires reasoning:
  - computation of class trees (Heart ⊑ Organ)
  - evaluation of queries (? – Car △ ...)
- reasoning (**OWL**) = theorem proving (**SHOIQ**)
- Reasoning in **SHOIQ** can be reduced to $C^2$ (the two variable fragment with counting)
  - $C^2$ is decidable [Grädel et al., 1997]
  - $C^2$ is NExpTime-compete [Pacholski et al., 2000], [Pratt-Hartmann, 2005]
What is it about SHOIQ?

- DL **SHOIQ** is a logical counterpart of **OWL DL**
- Development of **OWL DL**-ontologies requires reasoning:
  - computation of class trees (Heart ⊑ Organ)
  - evaluation of queries (¬ Car ⊓ . . .)
- reasoning (**OWL**) = theorem proving (**SHOIQ**)
- Reasoning in **SHOIQ** can be reduced to $C^2$ (the two variable fragment with counting)
  - $C^2$ is decidable [Grädel et al., 1997]
  - $C^2$ is NExpTime-compete [Pacholski et al., 2000], [Pratt-Hartmann, 2005]
  - but these procedures are not practical (“guess-and-check”)
**What is it About SHOIQ?**

- **DL SHOIQ** is a logical counterpart of **OWL DL**
- Development of **OWL DL**-ontologies requires reasoning:
  - computation of class trees (Heart $\sqsubseteq$ Organ)
  - evaluation of queries ($\exists$ Car $\sqcap$ ...)
- reasoning (**OWL**) = theorem proving (**SHOIQ**)
- Reasoning in **SHOIQ** can be reduced to **$C^2$** (the two variable fragment with counting)
  - **$C^2$** is decidable [Grädel et al., 1997]
  - **$C^2$** is NExpTime-compete [Pacholski et al., 2000], [Pratt-Hartmann, 2005]
    - but these procedures are not practical (“guess-and-check”)
- [Horrocks & Sattler, 2005] – the first (and the only up until now) goal-directed procedure for **SHOIQ**
What is it About SHOIQ?

- DL **SHOIQ** is a logical counterpart of **OWL DL**
- Development of **OWL DL**-ontologies requires reasoning:
  - computation of class trees (Heart ⊑ Organ)
  - evaluation of queries (?− Car ⊓ . . .)
- reasoning (**OWL**) = theorem proving (**SHOIQ**)
- Reasoning in **SHOIQ** can be reduced to $C^2$ (the two variable fragment with counting)
  - $C^2$ is decidable [Grädel et al., 1997]
  - $C^2$ is NExpTime-compete [Pacholski et al., 2000], [Pratt-Hartmann, 2005]
  - but these procedures are not practical (“guess-and-check”)
- [Horrocks & Sattler, 2005] – the first (and the only up until now) goal-directed procedure for **SHOIQ**
- Now we can decide **SHOIQ** also by resolution!
**Why a Resolution-Based Decision Procedure for SHOIQ?**
Why a Resolution-Based Decision Procedure for SHOIQ?

- different from the tableau-based approach
  - search for proofs vs. search for models

[Tableau is good for reasoning with large schema (terminologies)
Resolution is useful for reasoning with large data (assertions)]

[Hustadt, Motik & Sattler, 2004]
Why a **Resolution-Based Decision Procedure for SHOIQ**?

- different from the tableau-based approach
  - search for proofs vs. search for models
- likely to behave differently for different types of problems:
  - Tableau is good for reasoning with large schema (terminologies)
  - Resolution is useful for reasoning with large data (assertions) [Hustadt, Motik & Sattler, 2004]
**DESCRIPTION LOGICS: SYNTAX**

**Axioms**

```
Researcher ≡ Human ⊓ ∀produce.Paper
Researcher (Rob)
```
<table>
<thead>
<tr>
<th>Axioms</th>
<th>Terminology</th>
<th>Assertions</th>
</tr>
</thead>
<tbody>
<tr>
<td>Researcher ≡ Human □ ∀ produce.Paper</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Researcher (Rob)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
**Description Logics: Syntax**

**Axioms**

\[
\text{Researcher} \equiv \text{Human} \sqcap \forall \text{produce} . \text{Paper}
\]

\[
\text{Researcher} (\text{Rob})
\]

- Basic building blocks of DLs:
  - Concept names
  - Role names
  - Individuals
  - Operators
Basic building blocks of DLs:

- Concept names – sets: \( \text{Researcher}, \text{Human}, \text{Paper} \)
- Role names
- Individuals
- Operators

Terminology:
- \( \text{Researcher} \equiv \text{Human} \)
- \( \forall \text{produce. Paper} \)

Assertions:
- \( \text{Researcher (Rob)} \)
**Description Logics: Syntax**

<table>
<thead>
<tr>
<th>Axioms</th>
</tr>
</thead>
<tbody>
<tr>
<td>Researcher ≡ Human ⊓ ∀produce.Paper</td>
</tr>
<tr>
<td>Researcher (Rob)</td>
</tr>
</tbody>
</table>

- Basic building blocks of DLs:
  - Concept names – sets: Researcher, Human, Paper
  - Role names – binary relations: produces
  - Individuals
  - Operators

Terminology

Assertions
Description Logics and Ontologies

Description Logics: Syntax

Axioms

\[ \text{Researcher} \equiv \text{Human} \sqcap \forall \text{produce}. \text{Paper} \]

Researcher(\text{Rob})

Basic building blocks of DLs:

- Concept names – sets: Researcher, Human, Paper
- Role names – binary relations: produces
- Individuals – constants: Rob
- Operators

Terminology

Assertions
Description Logics and Ontologies

Description Logics: Syntax

Axioms

Researcher \equiv Human \sqcap \forall\text{produce}.\text{Paper} \quad \Rightarrow \quad \text{Ternology}

Researcher (Rob) \quad \Rightarrow \quad \text{ Assertions}

- Basic building blocks of DLs:
  - Concept names – sets: Researcher, Human, Paper
  - Role names – binary relations: produces
  - Individuals – constants: Rob
  - Operators – logical constructors: \( C_1 \sqcap C_2, \forall r.C, A \equiv C \)
Description Logics: Semantics

Axioms

Researcher ≡ Human □ ∀ produces. Paper

Researcher (Rob)

- Basic building blocks of DLs:
  - Concept names: Researcher, Human, Paper
  - Role names: produces
  - Individuals: Rob
  - Operators: \( C_1 □ C_2, \ ∀ r. C, \ A ≡ C \)
Description Logics: Semantics

Axioms

\[ \text{Researcher} \equiv \text{Human} \sqcap \forall \text{produces.Paper} \]

\[ \text{Researcher}(\text{Rob}) \]

Basic building blocks of DLs:

- Concept names
- \( \sim \) unary atoms:
- Role names
- Individuals
- Operators

\[ C_1 \sqcap C_2, \quad \forall r.C, \quad A \equiv C \]

\text{Rob}
**Description Logics: Semantics**

**Axioms**

\[
\text{Researcher} \equiv \text{Human} \sqcap \forall \text{produces}.\text{Paper}
\]

\[
\text{Researcher}(\text{Rob})
\]

- Basic building blocks of DLs:
  - Concept names
    \( \rights Consequently unary atoms: \)
    \( \text{Researcher}(x), \text{Human}(x), \text{Paper}(x) \)
  - Role names
    \( \rights Consequently binary atoms: \)
    produces
    \( \text{produces}(x,y) \)
  - Individuals
    \( \text{Rob} \)
  - Operators
    \( C_1 \sqcap C_2, \)
    \( \forall r. C, \)
    \( A \equiv C \)
## Description Logics: Semantics

### Axioms

<table>
<thead>
<tr>
<th>Axiom</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>Researcher $\equiv$ Human $\sqcap$ $\forall_{provides}.Paper$</td>
<td>Researcher(x), Human(x), Paper(x)</td>
</tr>
<tr>
<td>Researcher(Rob)</td>
<td></td>
</tr>
</tbody>
</table>

- Basic building blocks of DLs:
  - Concept names
    - $\rightarrow$ unary atoms: Researcher(x), Human(x), Paper(x)
  - Role names
    - $\rightarrow$ binary atoms: produces(x,y)
  - Individuals
    - $\rightarrow$ constants: Rob
  - Operators
    - $C_1 \sqcap C_2$, $\forall r.C$, $A \equiv C$
Description Logics: Semantics

Axioms

Researcher ≡ Human ⊓ ∀ produces.Paper

Researcher (Rob)

Basic building blocks of DLs:

- Concept names
  - Unary atoms: Researcher(x), Human(x), Paper(x)
- Role names
  - Binary atoms: produces(x,y)
- Individuals
  - Constants: Rob
- Operators
  - Constructors:
    - $C_1 \sqcap C_2$, $\forall r.C$, $A \equiv C$
    - $C_1(x) \land C_2(x)$, $\forall y.[r(x, y) \rightarrow C(y)]$, $\forall x.[A(x) \equiv C(x)]$
Description Logics: Semantics

Axioms

\[ \text{Researcher}(x) \equiv \text{Human}(x) \land \forall y. [\text{produces}(x, y) \rightarrow \text{Paper}(y)] \]
\[ \text{Researcher}(\text{Rob}) \]

Basic building blocks of DLs:

- Concept names
  - unary atoms: \( \text{Researcher}(x), \text{Human}(x), \text{Paper}(x) \)
- Role names
  - binary atoms: \( \text{produces}(x, y) \)
- Individuals
  - constants: \( \text{Rob} \)
- Operators
  - constructors: \( C_1(x) \land C_2(x), \forall y. [r(x, y) \rightarrow C(y)], \forall x. [A(x) \equiv C(x)] \)
**HIERARCHY OF DLs**

- Basic Description Logic $\mathcal{ALC}$: $\cap, \cup, \neg, \forall r.C, \exists r.C, \sqsubseteq$
- Transitive Roles: $\text{Transitive}(r)$
HIERARCHY OF DLs

- Basic Description Logic $\mathcal{ALC}$: $\sqcap, \sqcup, \neg, \forall r.C, \exists r.C, \sqsubseteq$

- Transitive Roles: $\text{Transitive}(r)$

- Role Hierarchies: $r_1 \sqsubseteq r_2$

- Inverse Roles: $r_2^{-}$

- Qualified Number Restrictions: $(\geq n r.C), (\leq n r.C)$

$= \mathcal{SHIQ}$
HIERARCHY OF DLs

- Basic Description Logic $\mathcal{ALC}$: $\cap, \cup, \neg, \forall r.C, \exists r.C, \sqsubseteq$
- Transitive Roles: $\text{Transitive}(r)$
- Role Hierarchies: $r_1 \sqsubseteq r_2$
- Inverse Roles: $r_2^-$
- Qualified Number Restrictions: $(\geq n \cdot r.C), (\leq n \cdot r.C)$

$\mathcal{SHIQ} = \mathcal{SHOIQ}$
**Expressive Power of SHOIQ**

- Cardinality restrictions: $|C| \leq n$
  - $C \subseteq \{i_1\} \cup \{i_2\} \cup \cdots \cup \{i_n\}$
  - $|C| \leq n$
**Expressive Power of SHOIQ**

- **Cardinality restrictions:** $|C| \leq n$, $|C| \geq n$
  - $C \subseteq \{i_1\} \cup \{i_2\} \cup \cdots \cup \{i_n\}$, $|C| \leq n$
  - $C \supseteq \{i_1\} \cup \{i_2\} \cup \cdots \cup \{i_n\}$, $|C| \geq n$
  - $\{i_p\} \cap \{i_q\} \subseteq \bot$, $p < q$

---

Yevgeny Kazakov and Boris Motik
A Resolution Decision Procedure for SHOIQ
**Expressive Power of SHOIQ**

- **Cardinality restrictions:** \(|C| \leq n, |C| \geq n\)
- **Large cardinality restrictions:**
  - \(C_0 \sqsupseteq \{i\}\)
  - \(C_0 \sqsubseteq (\geq 2 \times r \times C_1)\)

\[
C_0 \sqsupseteq \{i\} \quad |C_0| \geq 1
\]
\[
C_0 \sqsubseteq (\geq 2 \times r \times C_1) \quad |C_1| \geq 2
\]
**Expressive Power of SHOIQ**

- Cardinality restrictions: \(|C| \leq n, |C| \geq n|
- Large cardinality restrictions:
  - \(C_0 \supseteq \{i\}\)
  - \(C_0 \sqsubseteq (\geq 2r.C_1)\)
  - \(C_1 \sqsubseteq (\geq 2r.C_2)\)
  - \(\ldots\)
  - \(C_{n-1} \sqsubseteq (\geq 2r.C_n)\)

\[\begin{align*}
  |C_0| & \geq 1 \\
  |C_1| & \geq 2 \\
  \ldots \\
  |C_{n-1}| & \leq 2^{2r} n \\
\end{align*}\]
**Expressive Power of SHOIQ**

- **Cardinality restrictions:** $|C| \leq n$, $|C| \geq n$

- **Large cardinality restrictions:**
  - $C_0 \supseteq \{i\}$
  - $C_0 \subseteq (\geq 2 r \cdot C_1)$
  - $C_1 \subseteq (\geq 2 r \cdot C_2)$
  
  \[ \cdots \]
  - $C_{n-1} \subseteq (\geq 2 r \cdot C_n)$
  - $\top \subseteq (\leq 1 r^{-} \cdot \top)$

  ![Diagram showing the relationships between concepts and cardinality restrictions.]

- **Huge cardinality restrictions:**
  - $|C| \geq 2^n$
  - $|C| \leq 2^n$
**Expressive Power of SHOIQ**

- **Cardinality restrictions:** $|C| \leq n$, $|C| \geq n$
- **Large cardinality restrictions:** $|C| \geq 2^n$

- $C_0 \supseteq \{i\}$
- $C_0 \sqsubseteq (\geq 2 \cdot r \cdot C_1)$
- $C_1 \sqsubseteq (\geq 2 \cdot r \cdot C_2)$
  ...
- $C_{n-1} \sqsubseteq (\geq 2 \cdot r \cdot C_n)$
- $\top \sqsubseteq (\leq 1 \cdot r^\top \cdot \top)$

$|C_0| \geq 1$
$|C_1| \geq 2$
...
$|C_n| \geq 2^n$
**Expressive Power of SHOIQ**

- **Cardinality restrictions:** \(|C| \leq n, |C| \geq n\)
- **Large cardinality restrictions:** \(|C| \geq 2^n, |C| \leq 2^n\)

\[\begin{align*}
C_0 & \supseteq \{i\} \\
C_0 & \subseteq (\geq 2 r \cdot C_1) \\
C_1 & \subseteq (\geq 2 r \cdot C_2) \\
& \quad \ldots \\
C_{n-1} & \subseteq (\geq 2 r \cdot C_n) \\
T & \subseteq (\leq 1 r^{-1} \cdot T)
\end{align*}\]

\[\begin{align*}
C_0 & \subseteq \{i\} \\
C_1 & \subseteq (\geq 1 r^{-1} \cdot C_0) \\
C_2 & \subseteq (\geq 1 r^{-1} \cdot C_1) \\
& \quad \ldots \\
C_n & \subseteq (\geq 1 r^{-1} \cdot C_{n-1}) \\
T & \subseteq (\leq 2 r \cdot T)
\end{align*}\]
**Expressive Power of SHOIQ**

- **Cardinality restrictions:** $|C| \leq n$, $|C| \geq n$
- **Large cardinality restrictions:** $|C| \geq 2^n$, $|C| \leq 2^n$

- **C**
  - $C_0 \supseteq \{i\}$
  - $C_0 \subseteq (\geq 2 \cdot r \cdot C_1)$
  - $C_1 \subseteq (\geq 2 \cdot r \cdot C_2)$
  - $\ldots$
  - $C_{n-1} \subseteq (\geq 2 \cdot r \cdot C_n)$
  - $T \subseteq (\leq 1 \cdot r^{-1} \cdot T)$

- **C**
  - $C_0 \subseteq \{i\}$
  - $C_1 \subseteq (\geq 1 \cdot r^{-1} \cdot C_0)$
  - $C_2 \subseteq (\geq 1 \cdot r^{-1} \cdot C_1)$
  - $\ldots$
  - $C_n \subseteq (\geq 1 \cdot r^{-1} \cdot C_{n-1})$
  - $T \subseteq (\leq 2 \cdot r \cdot T)$
**Expressive Power of SHOIQ**

- Cardinality restrictions: $|C| \leq n$, $|C| \geq n$
- Large cardinality restrictions: $|C| \geq 2^n$, $|C| \leq 2^n$
- Huge cardinality restrictions: $|C| \geq 2^{2^n}$, $|C| \leq 2^{2^n}$
**Expressive Power of SHOIQ**

- **Cardinality restrictions:** $|C| \leq n$, $|C| \geq n$
- **Large cardinality restrictions:** $|C| \geq 2^n$, $|C| \leq 2^n$
- **Huge cardinality restrictions:** $|C| \geq 2^{2^n}$, $|C| \leq 2^{2^n}$

\[ B_n \sqcap \cdots \sqcap B_0 \subseteq \{i\} \]
\[ \top \subseteq (\geq 1 \, r^- . \top) \]
\[ \top \subseteq (\leq 2 \, r \, . \top) \]
\[
\begin{align*}
B_0 & \subseteq \forall r. \neg B_0 \\
\neg B_0 & \subseteq \forall r. B_0 \\
B_{i+1} & \sqcap B_i \subseteq \forall r. B_{i+1} \\
\neg B_{i+1} & \sqcap B_i \subseteq \forall r. \neg B_{i+1} \\
B_{i+1} & \sqcap \neg B_i \subseteq \forall r. [ (\neg B_{i+1} \sqcap B_i) \sqcup ( B_{i+1} \sqcap \neg B_i) ] \\
\neg B_{i+1} & \sqcap \neg B_i \subseteq \forall r. [ ( B_{i+1} \sqcap B_i) \sqcup (\neg B_{i+1} \sqcap \neg B_i) ] \\
\end{align*}
\]

- bits "count" over $r$

\[ B_n \sqcap \cdots \sqcap B_1 \sqcap B_0 = 0 \]
\[ B_n \sqcap \cdots \sqcap B_1 \sqcap \neg B_0 = 1 \]
\[ B_n \sqcap \cdots \sqcap \neg B_1 \sqcap B_0 = 2 \]
\[ \ldots \]
\[ B_n \sqcap \cdots \sqcap \neg B_1 \sqcap \neg B_0 = 2^n \]
Resolution-Based Procedures: The Basic Principles

- Invented by Joyner Jr. (1976)
- Allows one to use existing automated theorem provers (SPASS, VAMPIRE) as decision procedures
- The general idea is as follows:
  1. Define a clause class for the target fragment
  2. Show that this class is closed under inferences
  3. Show the class is finite for a fixed signature
- Many decision procedures are based on this principle:
  - clause classes $\mathcal{E}$, $\mathcal{S}^+$, $\mathcal{E}^+$, etc. [Fermüller et al., 1993]
  - modal logics [Schmidt, 1997], [Hustadt, 1999],
  - fragments of first-order logic [Bachmair et al., 1993], [Ganzinger & de Nivelle, 1999].
How to Turn Resolution Into a Decision Procedure?

- Tweak the parameters of a prover (ordering and selection function) so that the size of clauses does not grow.
How to Turn Resolution Into a Decision Procedure?

- Tweak the parameters of a prover (ordering and selection function) so that the size of clauses does not grow
- Problematic situations:

Example

\[
\begin{align*}
A(c) & \quad \sim \quad A(c) \quad \neg A(x) \lor A(f(x)) \\
A \sqsubseteq \exists r. A & \quad \Rightarrow \\
A(f(c)) & \\
A(f(f(c))) & \\
\vdots & \\
\end{align*}
\]

- Problem: the depth grows
HOW TO TURN RESOLUTION INTO A DECISION PROCEDURE?

- Tweak the parameters of a prover (ordering and selection function) so that the size of clauses does not grow.
- Problematic situations:

**Example**

\[
\begin{align*}
A(c) & \quad \leadsto \\
A \sqsubseteq \exists r. A & \\
\underline{A(c)} & \quad \neg A(x) \lor A(f(x)) \\
\underline{A(f(c))} & \\
\underline{A(f(f(c)))} & \\
\vdots & \\
\end{align*}
\]

- Problem: the depth grows.
- The reason: the selected literal is not the deepest one.
How to Turn Resolution Into a Decision Procedure?

- Tweak the parameters of a prover (ordering and selection function) so that the size of clauses does not grow
- Problematic situations:

**Example**

\[
\begin{align*}
A(c) & \quad \rightsquigarrow \quad \overline{A(c)} \quad \neg A(x) \quad \vee A(f(x)) \\
A \sqsubseteq \exists r.A & \\
A(f(c)) & \\
A(f(f(c))) & \\
& \ldots
\end{align*}
\]

- Problem: the depth grows
- The reason: the selected literal is not the deepest one
- Solution: resolve on the deepest literal
How to Turn Resolution Into a Decision Procedure?

- Tweak the parameters of a prover (ordering and selection function) so that the size of clauses does not grow
- Problematic situations:

Example

\[
\neg A(c) \quad \leadsto \quad \neg A(c) \lor \neg R(x, y) \lor A(y) \\
A \sqsubseteq \forall R.A \\
\neg R(c, y_1) \lor A(y_1) \\
\neg R(c, y_1) \lor \neg R(y_1, y_2) \lor A(y_2) \\
\ldots
\]

- Problem: variables got duplicated
HOW TO TURN RESOLUTION INTO A DECISION PROCEDURE?

- Tweak the parameters of a prover (ordering and selection function) so that the size of clauses does not grow
- Problematic situations:

**EXAMPLE**

\[
\begin{align*}
A(c) & \quad \leadsto \quad A(c) \quad \neg A(x) \vee \neg R(x, y) \vee A(y) \\
A \sqsubseteq \forall R.A & \quad \Rightarrow \quad \neg R(c, y_1) \vee A(y_1) \\
& \quad \Rightarrow \quad \neg R(c, y_1) \vee \neg R(y_1, y_2) \vee A(y_2) \\
& \quad \ldots
\end{align*}
\]

- Problem: variables got duplicated
- The reason: the unified expression does not contain all variables of the clause
**HOW TO TURN RESOLUTION INTO A DECISION PROCEDURE?**

- Tweak the parameters of a prover (ordering and selection function) so that the size of clauses does not grow
- Problematic situations:

  **EXAMPLE**

  \[
  \begin{align*}
  A(c) & \rightarrow \neg A(x) \lor \neg R(x, y) \lor A(y) \\
  A & \sqsubseteq \forall R.A \\
  & \rightarrow \neg R(c, y_1) \lor A(y_1) \\
  & \rightarrow \neg R(c, y_1) \lor \neg R(y_1, y_2) \lor A(y_2) \\
  \ldots
  \end{align*}
  \]

  - Problem: variables got duplicated
  - The reason: the unified expression does not contain all variables of the clause
  - Solution: resolve on the expression with all variables
How to Turn Resolution Into a Decision Procedure?

- Tweak the parameters of a prover (ordering and selection function) so that the size of clauses does not grow.
- Problematic situations: depth or no. of variables grows.
- Decidability is typically a consequence that all expressions in clauses are covering:
HOW TO TURN RESOLUTION INTO A DECISION PROCEDURE?

- Tweak the parameters of a prover (ordering and selection function) so that the size of clauses does not grow.
- Problematic situations: depth or no. of variables grows.
- Decidability is typically a consequence that all expressions in clauses are covering:
  
  - every functional term of an expression contains all variables

**EXAMPLE**

<table>
<thead>
<tr>
<th>Expression</th>
<th>Coverage</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \neg A(x) \lor r(x, f(x, y)) )</td>
<td>term ( f(x, y) ) is covering</td>
</tr>
<tr>
<td>( \neg A(x) \lor x \equiv c )</td>
<td>term ( c ) is not covering</td>
</tr>
</tbody>
</table>
**Difficulties with SHOIQ in resolution**

**Example**

<table>
<thead>
<tr>
<th></th>
<th></th>
<th>1. ( \neg O(x) \lor x \simeq i )</th>
<th>2. ( \neg O(x) \lor r(x, f(x)) )</th>
<th>3. ( \neg O(x) \lor O(f(x)) )</th>
<th>4. ( \neg r(x, y) \lor x \simeq g(y) )</th>
</tr>
</thead>
<tbody>
<tr>
<td>O ( \subseteq {i} )</td>
<td>( \rightsquigarrow )</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>O ( \subseteq \exists r. O )</td>
<td>( \rightsquigarrow )</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>T ( \subseteq \leq 1 r^{-}.T )</td>
<td>( \rightsquigarrow )</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

\( i \)
DIFFICULTIES WITH \textit{SHOIQ} IN RESOLUTION

\textbf{Example}

\begin{align*}
\mathcal{O} \subseteq \{i\} & \implies 1. \neg O(x) \lor x \simeq i \quad \text{not covering} \\
\mathcal{O} \subseteq \exists r \mathcal{O} & \implies 2. \neg O(x) \lor r(x, f(x)) \\
\top \subseteq \leq 1 r^- \top & \implies 3. \neg O(x) \lor O(f(x)) \\
\top \subseteq \leq 1 r^- \top & \implies 4. \neg r(x, y) \lor x \simeq g(y) \quad \text{not covering}
\end{align*}
DIFFICULTIES WITH \textsc{SHOIQ} IN RESOLUTION

**Example**

\[ \begin{align*}
\text{O} & \subseteq \{i\} \\
\text{O} & \subseteq \exists r . \text{O} \\
\top & \subseteq \leq 1 \, \text{r} . \top
\end{align*} \]

\[ \begin{align*}
&\sim \Rightarrow 1. \neg \text{O}(x) \lor x \simeq i \\
&\sim \Rightarrow 2. \neg \text{O}(x) \lor r(x, f(x)) \\
&\sim \Rightarrow 3. \neg \text{O}(x) \lor \text{O}(f(x)) \\
&\sim \Rightarrow 4. \neg r(x, y) \lor x \simeq g(y)
\end{align*} \]

OR[1; 3] : 5. \( \neg \text{O}(x) \lor f(x) \simeq i \)

OR[2; 4] : 6. \( \neg \text{O}(x) \lor x \simeq g(f(x)) \)

OP[5; 6] : 7. \( \neg \text{O}(x) \lor x \simeq g(i) \)
Difficulties with **SHOIQ** in resolution

**Example**

\[
\begin{align*}
\mathcal{O} &\subseteq \{i\} \\
\mathcal{O} &\subseteq \exists r.\mathcal{O} \\
\mathcal{T} &\subseteq \leq 1 r.\mathcal{T}
\end{align*}
\]

\[\begin{align*}
1. \neg \mathcal{O}(x) \lor x \equiv i &\lessgtr\ 1. \\
2. \neg \mathcal{O}(x) \lor r(x, f(x)) &\lessgtr\ 2. \\
3. \neg \mathcal{O}(x) \lor \mathcal{O}(f(x)) &\lessgtr\ 3. \\
4. \neg r(x, y) \lor x \equiv g(y) &\lessgtr\ 4.
\end{align*}\]

\[\text{OR}[1;3]: 5. \neg \mathcal{O}(x) \lor f(x) \equiv i\]

\[\text{OR}[2;4]: 6. \neg \mathcal{O}(x) \lor x \equiv g(f(x))\]

\[\text{OP}[5;6]: 7. \neg \mathcal{O}(x) \lor x \equiv g(i) \lessgtr\ \text{of the same form}\]
**DIFFICULTIES WITH SHOIQ IN RESOLUTION**

**EXAMPLE**

\begin{align*}
\text{O} \subseteq \{ i \} & \quad \leadsto \quad 1. \neg \text{O}(x) \lor x \approx i \\
\text{O} \subseteq \exists r \cdot \text{O} & \quad \leadsto \quad 2. \neg \text{O}(x) \lor r(x, f(x)) \\
\top \subseteq \leq 1 r^- . \top & \quad \leadsto \quad 3. \neg \text{O}(x) \lor \overline{\text{O}(f(x))} \\
\top \subseteq \leq 1 r^- . \top & \quad \leadsto \quad 4. \neg r(x, y) \lor x \approx g(y)
\end{align*}

\text{OR}[1; 3] : 5. \neg \text{O}(x) \lor f(x) \approx i \\
\text{OR}[2; 4] : 6. \neg \text{O}(x) \lor x \approx g(f(x)) \\
\text{OP}[5; 6] : 7. \neg \text{O}(x) \lor x \approx g(i) \quad \text{of the same form} \\
\quad \ldots \ 8. \neg \text{O}(x) \lor x \approx g(g(i)) \quad \text{produces deeper} \\
\quad \ldots \ 9. \neg \text{O}(x) \lor x \approx g(g(g(i))) \quad \text{clauses}
**Difficulties with SHOIQ in resolution**

### Example

| ∀x ∈ {i} | ⊇ | 1. ¬O(x) ∨ x ≃ i |
| O ⊇ ∃r.O | ⊇ | 2. ¬O(x) ∨ r(x, f(x)) |
| ⊤ ⊇ ≤ 1 r−.⊤ | ⊇ | 3. ¬O(x) ∨ O(f(x)) |

**OR[1; 3]:** 5. ¬O(x) ∨ f(x) ≃ i

**OR[2; 4]:** 6. ¬O(x) ∨ x ≃ g(f(x))

**OP[5; 6]:** 7. ¬O(x) ∨ x ≃ g(i)

**add new:** 8. ¬O(x) ∨ i ≃ g(i)  consequence of 1 and 7
DIFFICULTIES WITH \textbf{SHOIQ} IN RESOLUTION

\begin{example}

\begin{align*}
\text{O} & \subseteq \{i\} \quad \leadsto \quad 1. \neg \text{O}(x) \lor x \equiv i \\
\text{O} & \subseteq \exists r. \text{O} \quad \leadsto \quad 3. \neg \text{O}(x) \lor r(x, f(x)) \\
\top & \subseteq \leq 1 r^-. \top \quad \leadsto \quad 4. \neg r(x, y) \lor x \equiv g(y)
\end{align*}

\textbf{REDUNDANCY FOR CLAUSES}

A clause is redundant if it follows from smaller clauses

\begin{align*}
\text{OR}[1; 3] : 5. & \neg \text{O}(x) \lor f(x) \equiv i \\
\text{OR}[2; 4] : 6. & \neg \text{O}(x) \lor x \equiv g(f(x)) \\
\text{OP}[5; 6] : 7. & \neg \text{O}(x) \lor x \equiv g(i) \quad \text{follows from 1 and 8} \\
8. & \neg \text{O}(x) \lor i \equiv g(i) \quad \text{consequence of 1 and 7}
\end{align*}
\end{example}
**DIFFICULTIES WITH \textit{SHOIQ} IN RESOLUTION**

**EXAMPLE**

\[\begin{align*}
\text{O} \subseteq \{i\} & \quad \Rightarrow \quad 1. \neg \text{O}(x) \lor x \simeq i \\
\text{O} \subseteq \exists r. \text{O} & \quad \Rightarrow \quad 3. \neg \text{O}(x) \lor \neg \exists r. \text{O} \lor \neg \exists f. \text{O} \\
\top \subseteq \leq 1 r^- \cdot \top & \quad \Rightarrow \quad 5. \neg \text{O}(x) \lor f(x) \simeq i \\
\top \subseteq \leq 1 r^- \cdot \top & \quad \Rightarrow \quad 7. \neg \text{O}(x) \lor x \simeq g(i)
\end{align*}\]

**REDUNDANCY FOR CLAUSES**

A clause is redundant if it follows from smaller clauses

OR[1; 3] : 5. \(\neg \text{O}(x) \lor f(x) \simeq i\)

OR[2; 4] : 6. \(\neg \text{O}(x) \lor x \simeq g(f(x))\)

OP[5; 6] : 7. \(\neg \text{O}(x) \lor x \simeq g(i)\) follows from 1 and 8 larger than 1,
**DIFFICULTIES WITH **\textit{SHOIQ} **IN RESOLUTION**

**EXAMPLE**

\[
\begin{align*}
\emptyset & \subseteq \{i\} \\
\emptyset & \subseteq \exists r. \emptyset \\
\top & \subseteq \leq 1 \text{ } r^-. \top
\end{align*}
\]

\[
\begin{align*}
1. \neg O(x) \lor x \simeq i & \Rightarrow \quad 2. \neg O\big((\neg x \lor f(x))\big) \\
3. \neg O\big((\neg x \lor f(x))\big) & \Rightarrow \quad 4. \neg O\big((\neg x \lor f(x))\big)
\end{align*}
\]

**REDUNDANCY FOR CLAUSES**

A clause is redundant if it follows from smaller clauses

\[
\begin{align*}
\text{OR}[1;3] & : 5. \neg O(x) \lor f(x) \simeq i \\
\text{OR}[2;4] & : 6. \neg O(x) \lor x \simeq g(f(x)) \\
\text{OP}[5;6] & : 7. \neg O(x) \lor x \simeq g(i)
\end{align*}
\]

follows from 1 and 8 larger than 1, but not larger than 8!

8. \(\neg O(x) \lor i \simeq g(i)\)
Difficulties with *SHOIQ* in resolution

**Example**

\[ \begin{align*}
\text{O} & \subseteq \{i\} \quad \sim \Rightarrow \\
\text{O} & \subseteq \exists r. \text{O} \quad \sim \Rightarrow \\
\top & \subseteq \leq 1 r^-.\top \quad \sim \Rightarrow
\end{align*} \]

A clause is redundant if it follows from *smaller* clauses

OR[1; 3] : 5. \( \neg \text{O}(x) \lor f(x) \simeq i \)

OR[2; 4] : 6. \( \neg \text{O}(x) \lor x \simeq g(f(x)) \)

OP[5; 6] : 7. \( \neg \text{O}(x) \lor x \simeq g(i) \)
**DIFFICULTIES WITH SHOIQ IN RESOLUTION**

**EXAMPLE**

<table>
<thead>
<tr>
<th>Clause</th>
<th>Rule</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>( O \subseteq {i} )</td>
<td>( \leadsto )</td>
<td>1. ( \neg O(x) \lor x \approx i )</td>
</tr>
<tr>
<td>( O \subseteq \exists r.O )</td>
<td>( \leadsto )</td>
<td>2. ( \neg O(x) \lor r(x, f(x)) )</td>
</tr>
<tr>
<td>( T \subseteq \leq 1 r^{-}.T )</td>
<td>( \leadsto )</td>
<td>3. ( \neg O(x) \lor O(f(x)) )</td>
</tr>
</tbody>
</table>

**REdundancy for Clauses**

A clause is redundant if it follows from smaller clauses.

<table>
<thead>
<tr>
<th>Rule</th>
<th>Clause</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>OR[1; 3]</td>
<td>5. ( \neg O(x) \lor f(x) \approx i )</td>
<td></td>
</tr>
<tr>
<td>OR[2; 4]</td>
<td>6. ( \neg O(x) \lor x \approx g(f(x)) )</td>
<td></td>
</tr>
<tr>
<td>OP[5; 6]</td>
<td>7. ( \neg O(x) \lor x \approx g(i) ) wait a bit...</td>
<td></td>
</tr>
<tr>
<td>OR[7; 3]</td>
<td>8. ( \neg O(x) \lor f(x) \approx g(i) )</td>
<td></td>
</tr>
</tbody>
</table>
DIFFICULTIES WITH \textsc{SHOIQ} IN RESOLUTION

EXAMPLE

\begin{align*}
O & \subseteq \{i\} & \leadsto & & 1. \neg O(x) \lor x \simeq i \\
O & \subseteq \exists r. O & \leadsto & & 2. \neg O(x) \lor \neg r(x, f(x)) \\
\top & \subseteq \leq 1 r^-. \top & \leadsto & & 3. \neg O(x) \lor O(f(x)) \\
\end{align*}

REDUNDANCY FOR CLAUSES

A clause is redundant if it follows from smaller clauses

OR[1;3]: 5. \neg O(x) \lor f(x) \simeq i

OR[2;4]: 6. \neg O(x) \lor x \simeq g(f(x))

OP[5;6]: 7. \neg O(x) \lor x \simeq g(i) \quad \text{wait a bit...}

OR[7;3]: 8. \neg O(x) \lor f(x) \simeq g(i)

add: 9. \neg O(x) \lor i \simeq g(i) \quad \text{consequence of 5 and 8}
DIFFICULTIES WITH \textit{SHOIQ} IN RESOLUTION

\begin{example}
\begin{align*}
\text{O} & \subseteq \{i\} \\
\text{O} & \subseteq \exists r . \text{O} \\
\text{T} & \subseteq \leq 1 r^{-}.\text{T}
\end{align*}
\end{example}

\text{REDUNDANCY FOR CLAUSES}

A clause is redundant if it follows from smaller clauses

\begin{align*}
\text{OR}[1; 3] & : 5. \neg \text{O}(x) \lor f(x) \equiv i \\
\text{OR}[2; 4] & : 6. \neg \text{O}(x) \lor x \equiv g(f(x)) \\
\text{OP}[5; 6] & : 7. \neg \text{O}(x) \lor x \equiv g(i) \\
\text{OR}[7; 3] & : 8. \neg \text{O}(x) \lor f(x) \equiv g(i) \tag{follows from 5 and 9 larger than 5, and larger than 9!} \\
\text{OR}[7; 3] & : 9. \neg \text{O}(x) \lor i \equiv g(i) \tag{consequence of 5 and 8}
\end{align*}
### Difficulties with SHOIQ in Resolution

#### Example

<table>
<thead>
<tr>
<th>Clause</th>
<th>Resolution Steps</th>
<th>Redundancy for Clauses</th>
</tr>
</thead>
<tbody>
<tr>
<td>( O \subseteq { i } )</td>
<td>( \neg O(x) \lor x \simeq i )</td>
<td>A clause is redundant if it follows from smaller clauses</td>
</tr>
<tr>
<td>( O \subseteq \exists r. O )</td>
<td>( \neg O(x) \lor r(x, f(x)) )</td>
<td></td>
</tr>
<tr>
<td>( T \subseteq \leq 1 r^{-}. T )</td>
<td>( \neg O(x) \lor \top )</td>
<td></td>
</tr>
</tbody>
</table>

#### Resolution Steps

- \( \neg O(x) \lor x \simeq i \) \( \Rightarrow \) 1.
- \( \neg O(x) \lor r(x, f(x)) \) \( \Rightarrow \) 2.
- \( \neg O(x) \lor r(x, y) \lor x \simeq g(y) \) \( \Rightarrow \) 3.
- \( \neg O(x) \lor f(x) \simeq i \) \( \Rightarrow \) 4.
- \( \neg O(x) \lor x \simeq g(f(x)) \) \( \Rightarrow \) 5.
- \( \neg O(x) \lor x \simeq g(f(x)) \) \( \Rightarrow \) 6.
- \( \neg O(x) \lor x \simeq g(i) \) \( \Rightarrow \) 7.
- \( \neg O(x) \lor f(x) \simeq g(i) \) \( \Rightarrow \) 8.

#### Conclusion

- The saturation procedure terminates!
The idea is developed into a new simplification rule that introduces constants

\[
\begin{align*}
\alpha(x) \lor \bigvee_{i=1}^{n} f(x) & \simeq t_i \\
\alpha(x) \lor \bigvee_{i=1}^{n} f(x) & \simeq c_i \\
\alpha(x) \lor \bigvee_{j=1}^{n} c_i & \simeq t_j \\
1 \leq i \leq n
\end{align*}
\]

where \((i)\) \(c_i\) are fresh constants for \(t_i\) and \(\alpha\)
The idea is developed into a new simplification rule that introduces constants. The constants are reused when the rule has been applied to $\alpha(x)$ and $f(x)$ before.
The idea is developed into a new simplification rule that introduces constants. The constants are reused when the rule has been applied to $\alpha(x)$ and $f(x)$ before. There is a second variant of this rule for a different type of clauses.

**Nominal Generation 1**

\[
\begin{align*}
\alpha(x) \lor \bigvee_{i=1}^{n} f(x) &\simeq t_i \\
\alpha(x) \lor \bigvee_{i=1}^{k} f(x) &\simeq c_i \\
\alpha(x) \lor \bigvee_{j=1}^{n} c_i &\simeq t_j \\
1 \leq i \leq k
\end{align*}
\]

Where (i) $c_i$ are fresh constants for $t_i$ and $\alpha$, (ii) $k=n$ for the first application of rule for $\alpha(x)$ and $f(x)$, otherwise $k$ and $c_i$ are reused.

**Nominal Generation 2**

\[
\begin{align*}
\alpha(x) \lor \bigvee_{i=1}^{n} f(x) &\simeq t_i \lor \bigvee_{i=1}^{n} x \simeq c_i \\
\end{align*}
\]
Termination and Complexity Analysis

Every application of the rule can increase the number of constants by at most a polynomial factor.

Nominal Generation

\[
\alpha(x) \lor \bigvee_{i=1}^{n} f(x) \simeq t_i
\]

\[
\alpha(x) \lor \bigvee_{i=1}^{k} f(x) \simeq c_i
\]

\[
\alpha(x) \lor \bigvee_{j=1}^{n} c_i \simeq t_j
\]

1 \leq i \leq k

where (i) \( c_i \) are fresh constants for \( t_i \) and \( \alpha \), (ii) \( k=n \) for the first application of rule for \( \alpha(x) \) and \( f(x) \), otherwise \( k \) and \( c_i \) are reused.
Every application of the rule can increase the number of constants by at most a polynomial factor.

There are at most exponentially many applications possible (exponentially many pairs $\alpha(x)$ and $f(x)$).

**Nominal Generation**

\[
\frac{\alpha(x) \lor \bigvee_{i=1}^{n} f(x) \simeq t_i}{\alpha(x) \lor \bigvee_{i=1}^{k} f(x) \simeq c_i}
\]

\[
\alpha(x) \lor \bigvee_{i=1}^{n} c_i \simeq t_j \\
1 \leq i \leq k
\]

where (i) $c_i$ are fresh constants for $t_i$ and $\alpha$, (ii) $k=n$ for the first application of rule for $\alpha(x)$ and $f(x)$, otherwise $k$ and $c_i$ are reused.
Termi nation and Complexity Analysis

- Every application of the rule can increase the number of constants by at most a polynomial factor.
- There are at most exponentially many applications possible (exponentially many pairs $\alpha(x)$ and $f(x)$).
- Hence the procedure terminates, with the upper bound: $3\text{EXPTIME}$.

Nominal Generation

\[
\begin{align*}
\alpha(x) \lor \bigvee_{i=1}^{n} f(x) &\sim t_i \\
\alpha(x) \lor \bigvee_{i=1}^{k} f(x) &\sim c_i \\
\alpha(x) \lor \bigvee_{j=1}^{n} c_i &\sim t_j \\
1 \leq i \leq k
\end{align*}
\]

where (i) $c_i$ are fresh constants for $t_i$ and $\alpha$, (ii) $k=n$ for the first application of rule for $\alpha(x)$ and $f(x)$, otherwise $k$ and $c_i$ are reused.
Why Is It So Hard?

- In *SHOIQ* it is possible to express very large cardinality restrictions like $|C| \leq 2^{2^n}$, $|D| \geq 2^{2^m}$.
**Why Is It So Hard?**

- In **SHOIQ** it is possible to express very large cardinality restrictions like $|C| \leq 2^{2^n}$, $|D| \geq 2^{2^m}$.

- Hence, it is possible to encode **combinatorial constraints** involving very big numbers:

---

**Example**

$|A \uplus B| \leq 2^{2^n}$, $|A \uplus C| \geq 2^{2^m+k}$, $|B \uplus C| \geq 2^{2^k}$, $|C| \leq 2^n$
Why Is It So Hard?

- In SHOIQ it is possible to express very large cardinality restrictions like $|C| \leq 2^{2^n}$, $|D| \geq 2^{2^m}$.

- Hence, it is possible to encode combinatorial constraints involving very big numbers.

- Such problems (in particular, the pigeon hole principle) are known to be hard for resolution since it is not really capable to deal with numbers.
Conclusions

We have found a decision procedure for SHOIQ based on basic superposition calculus which runs in $3\text{ExpTime}$.

High complexity is due to combination of:
- nominals + number restrictions + inverse roles

The restriction of the procedure to simpler languages (SHOIQ, ALC) behaves like procedures known before

hence it exhibits “pay as you go” behaviour

The restricted version for SHIQ has proved itself in practice in system KAON2 \(^1\)

No additional degree of non-determinism is introduced by Nominal Generation rules

Future developments: Integration of algebraic reasoning into resolution?

\(^1\)http://www.kaon2.semanticweb.org
Thank You!
Comparison with the Tableau Procedure

- Constants introduced by Nominal Generation correspond (in some way) to “nominal nodes”.
- The exact number of different constants is not guessed, but equality constraints are generated.
- “Blocking” is native in resolution by subsumption deletion.
- No “yo-yo” effect in resolution, since deletion of clauses is permanent.

(A picture from the presentation by Horrocks & Sattler on “A Tableau Decision Procedure for SHOIQ” [2005])