Representation and Concretization of Underdetermined Data

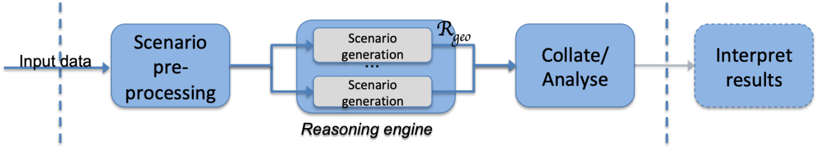
Vegar Skaret

August 24, 2020

Outline

Geological Assistant

- Research project in the SIRIUS center started after discovering a lack of tooling in the petroleum industry
- Assist the analysis process of oil and gas exploration
  - Help coping with uncertainty and huge state space



- Scenario pre-processing:
  - Generate concrete states from underdetermined geological input data

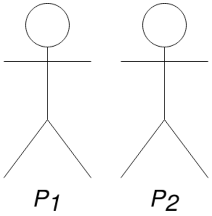
Outline

Underdetermined Data

- Represents a world of objects with attributes
- Some concrete values are missing
- Limited range of values based on domain knowledge
- Thus possible to generate a finite set of instantiations
- Relevant when
  - Discrete data types with finite ranges
  - Difficult to gather necessary data to make a conclusion

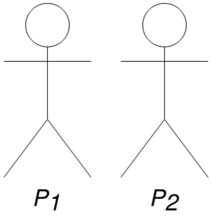
Underdetermined Couple

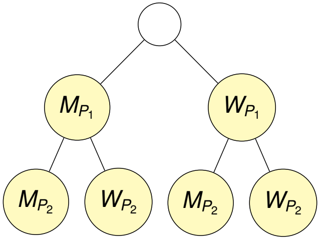
- Known facts about the world:
  - A married couple P_1 and P_2
- Domain knowledge:
  - Assume a person P_i can only have the gender man (M_{P_i}) or woman (W_{P_i})



Underdetermined Couple

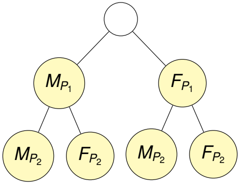
- Known facts about the world:
  - A married couple P_1 and P_2
- Domain knowledge:
  - Assume a person P_i can only have the gender man (M_{P_i}) or woman (W_{P_i})





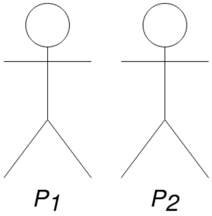
To Concretize Underdetermined Data

- To assign values to attributes
- Represented as a tree where each edge is a value assignment



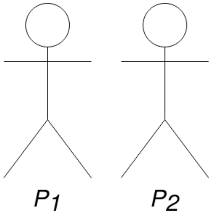
Underdetermined Couple 1990

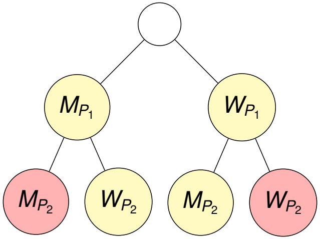
- Known facts:
  - A married couple P_1 and P_2
- Domain knowledge for 1990:
  - A person P_i can only have the gender man (M_{P_i}) or woman (F_{P_i})
  - Same sex marriage not allowed



Underdetermined Couple 1990

- Known facts:
  - A married couple P_1 and P_2
- Domain knowledge for 1990:
  - A person P_i can only have the gender man (M_{P_i}) or woman (F_{P_i})
  - Same sex marriage not allowed





Outline

Research Questions

1. How can underdetermined worlds be represented within the Web Ontology Language (OWL)?
2. How can all of the legal concretizations of a world be generated using OWL?
3. How can a representation of a world be translated from one language to another, specifically from OWL to Maude?

- Methodology for representing and solving a problem in the Geological Assistant

Outline

Representation

- Objects: Class
  - Identifier: Individual name
  - Attributes: Class
  - Values: Class
  - Assigned values: Class assertions
- Binary relations: Properties
- Dependencies: Axioms with properties and value classes

- Concrete semantics: no more class assertions

Attribute Modeling

- Attribute represented as a class A equivalent to the disjoint union of its values V_i:

$$A \equiv V_1 \sqcup V_2 \sqcup \; \ldots \; \sqcup V_n,$$

$$V_1 \sqcap V_2 \sqsubseteq \bot, \; V_1 \sqcap V_3 \sqsubseteq \bot, \; \ldots, \; V_{n-1} \sqcap V_n \sqsubseteq \bot$$

- Concrete semantics:
  - At least one value must be assigned
  - At most one value can be assigned

Object Modeling 1

- An object O is represented as the intersection of all of its attribute classes A_i:

$$O \equiv \bigsqcap_{i=1}^{n} A_i$$

- Concrete semantics:
  - An object needs to have all of its attributes

Object Modeling 2

- In addition, an object O is the union of the negation of all of the other attribute classes W_j in the domain:

$$O \sqsubseteq \bigsqcup_{j=1}^{m} \neg W_j$$

- Concrete semantics:
  - An object cannot be assigned any attribute it does not have

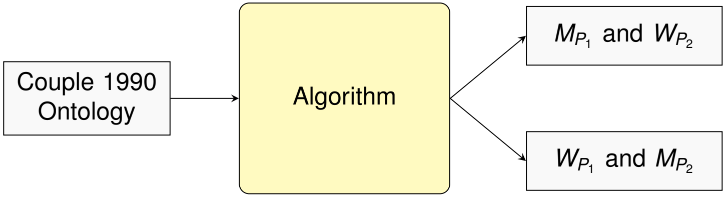
Dependencies in a World

- From 1990: $\mathit{Man} \sqsubseteq \forall \mathit{married}.\mathit{Woman}$
  - A man can only marry women

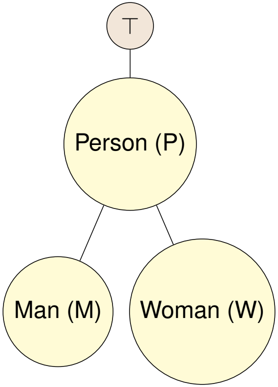
Outline

Concretization Algorithm

- Given the ontology, generate all maximal ABoxes
  - Maximal: any new class assertion will cause inconsistency
- A maximal ABox corresponds to a concretization



Ontology Class Hierarchy

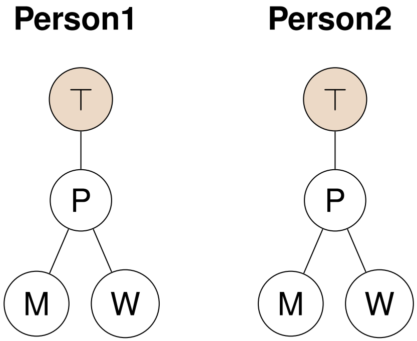


- Note: attribute class *Gender* removed for brevity
- Algorithm traverses the hierarchy breadth-first

Rest of the 1990 Ontology

- Object axioms:
  - $Person \equiv Man \sqcup Woman$
  - $Man \sqcap Woman \sqsubseteq \bot$
- Dependency axioms:
  - $Man \sqsubseteq \forall married.Woman$
  - $Woman \sqsubseteq \forall married.Man$
- Initial assertions:
  - $\boxed{married(1,2)}$

Algorithm Initial State

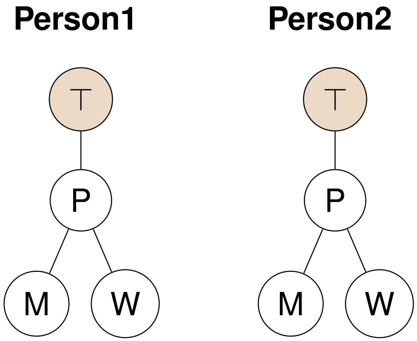


- Instantiations (ABox)
  1. married(1, 2)

Node Coloring Semantics

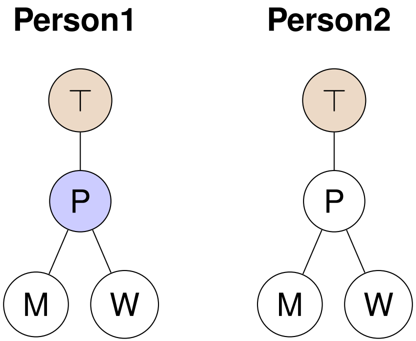
- ○ Class not yet reached
- 🟤 Root of the class hierarchy
- 🔵 Current class assertion
- 🟡 Ontology consistent after assertion
- 🔴 Ontology inconsistent
- ◍ Traversal without class

Algorithm Example Run



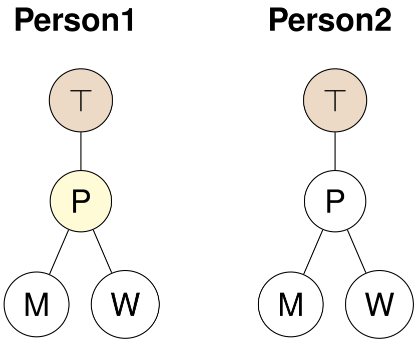
- Instantiations (ABox)
  1. married(1, 2)

Example Algorithm Run



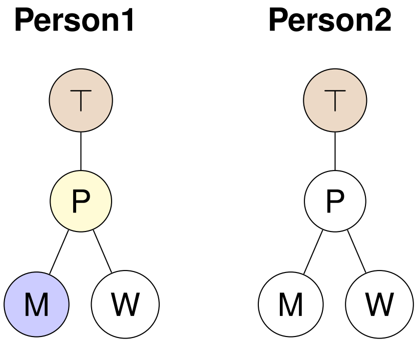
- Instantiations
  1. married(1, 2), P(1)

Example Algorithm Run



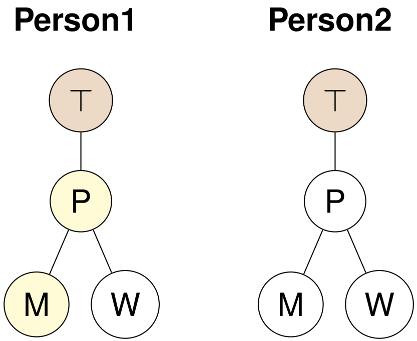
- Instantiations
  1. married(1, 2), P(1)

Example Algorithm Run



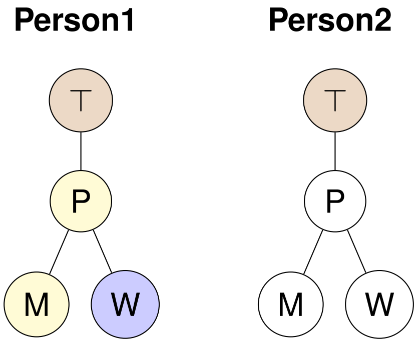
- Instantiations
  1. married(1, 2), P(1), M(1)

Example Algorithm Run



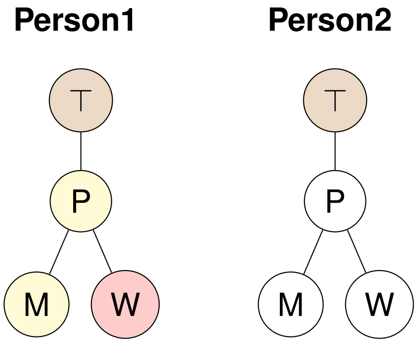
- Instantiations
  1. married(1, 2), P(1), M(1)

Example Algorithm Run



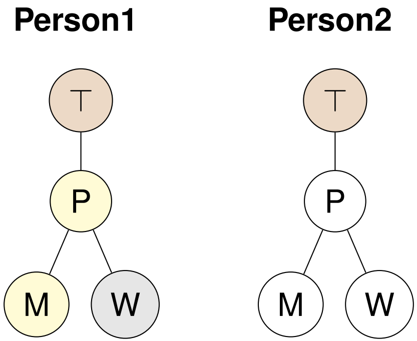
- Instantiations
  1. married(1, 2), P(1), M(1), W(1)

Example Algorithm Run



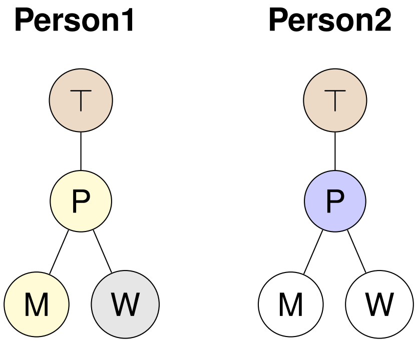
- Instantiations
  1. married(1, 2), P(1), M(1), W(1)

Example Algorithm Run



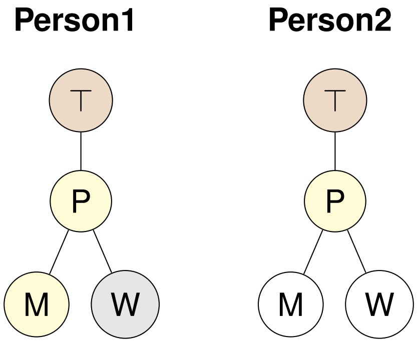
- Instantiations
  1. married(1, 2), P(1), M(1)

Example Algorithm Run



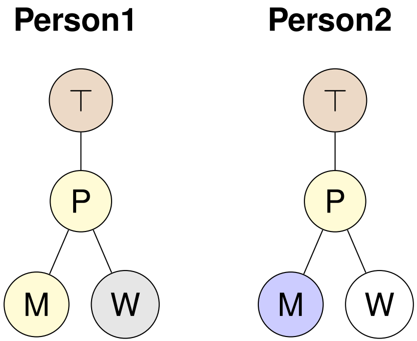
- Instantiations
  1. married(1, 2), P(1), M(1), P(2)

Example Algorithm Run



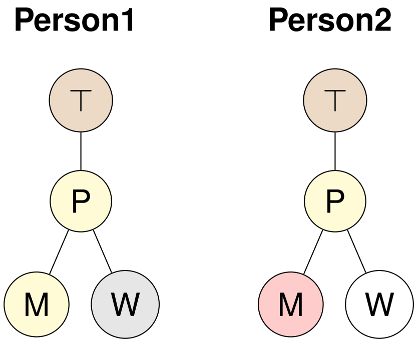
- Instantiations
  1. married(1, 2), P(1), M(1), P(2)

Example Algorithm Run



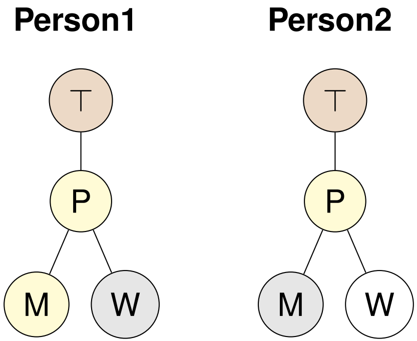
- Instantiations
  1. married(1, 2), P(1), M(1), P(2), M(2)

Example Algorithm Run



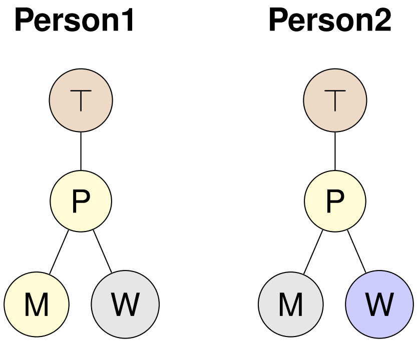
- Instantiations
    1. married(1, 2), P(1), M(1), P(2), M(2)

Example Algorithm Run



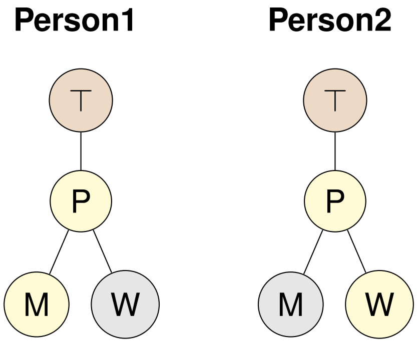
- Instantiations
  1. married(1, 2), P(1), M(1), P(2)

Example Algorithm Run



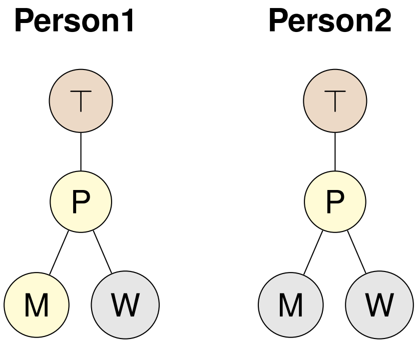
- Instantiations
  1. married(1, 2), P(1), M(1), P(2), W(2)

Example Algorithm Run



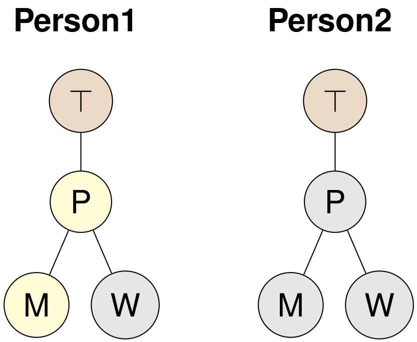
- Instantiations
  1. married(1, 2), P(1), M(1), P(2), W(2)

Example Algorithm Run



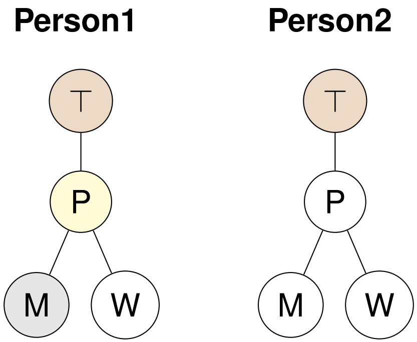
- Instantiations
  1. married(1, 2), P(1), M(1), P(2), W(2)
  2. married(1, 2), P(1), M(1), P(2)

Example Algorithm Run



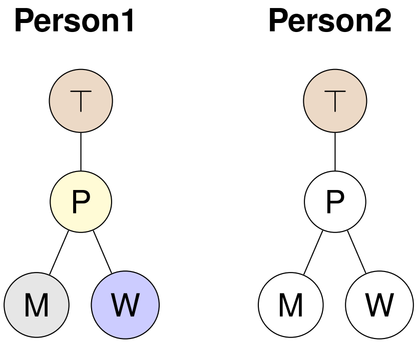
- Instantiations
  1. married(1, 2), P(1), M(1), P(2), W(2)
  2. married(1, 2), P(1), M(1)

Example Algorithm Run



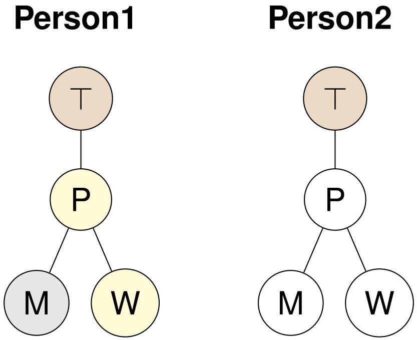
- Instantiations
  1. married(1, 2), P(1), M(1), P(2), W(2)
  2. married(1, 2), P(1)

Example Algorithm Run



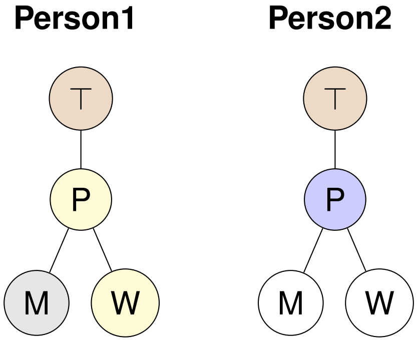
- Instantiations
  1. married(1, 2), P(1), M(1), P(2), W(2)
  2. married(1, 2), P(1), W(1)

Example Algorithm Run



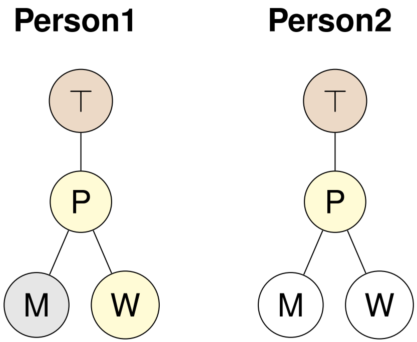
- Instantiations
  1. married(1, 2), P(1), M(1), P(2), W(2)
  2. married(1, 2), P(1), W(1)

Example Algorithm Run



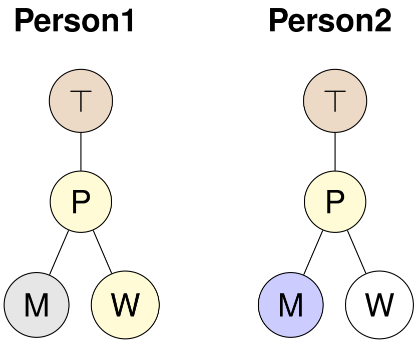
- Instantiations
  1. married(1, 2), P(1), M(1), P(2), W(2)
  2. married(1, 2), P(1), W(1), P(2)

Example Algorithm Run



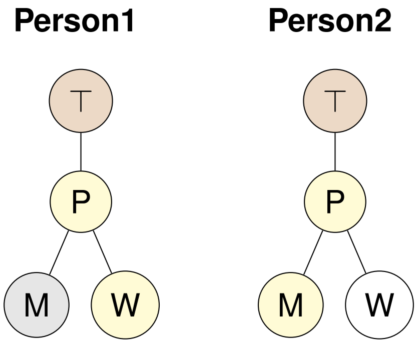
- Instantiations
  1. married(1, 2), P(1), M(1), P(2), W(2)
  2. married(1, 2), P(1), W(1), P(2)

Example Algorithm Run



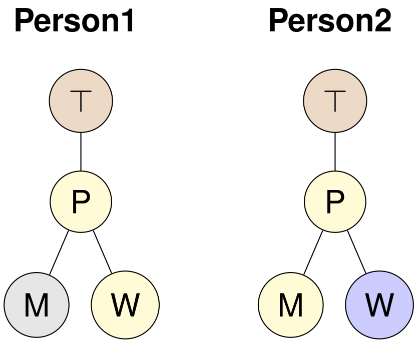
- Instantiations
  1. married(1, 2), P(1), M(1), P(2), W(2)
  2. married(1, 2), P(1), W(1), P(2), M(2)

Example Algorithm Run



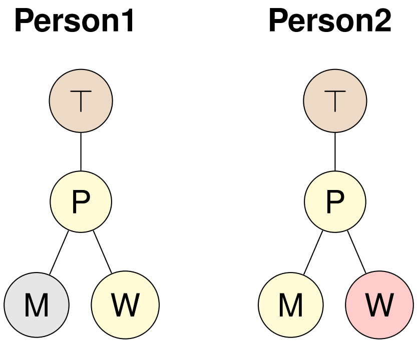
- Instantiations
  1. married(1, 2), P(1), M(1), P(2), W(2)
  2. married(1, 2), P(1), W(1), P(2), M(2)

Example Algorithm Run



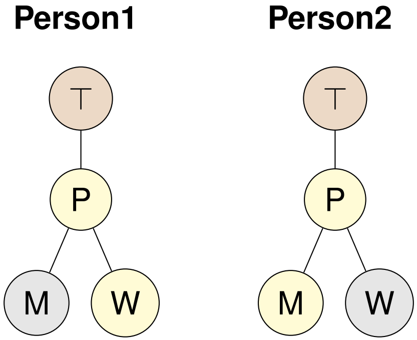
- Instantiations
  1. married(1, 2), P(1), M(1), P(2), W(2)
  2. married(1, 2), P(1), W(1), P(2), M(2), W(2)

Example Algorithm Run



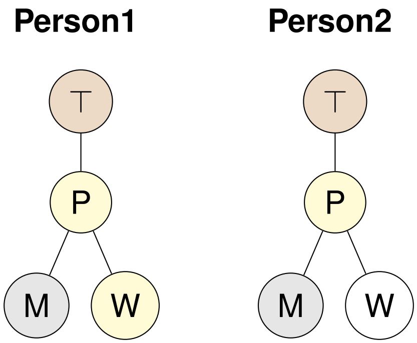
- Instantiations
  1. married(1, 2), P(1), M(1), P(2), W(2)
  2. married(1, 2), P(1), W(1), P(2), M(2), W(2)

Example Algorithm Run



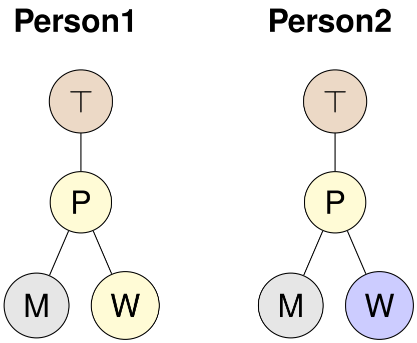
- Instantiations
  1. married(1, 2), P(1), M(1), P(2), W(2)
  2. married(1, 2), P(1), W(1), P(2), M(2)

Example Algorithm Run



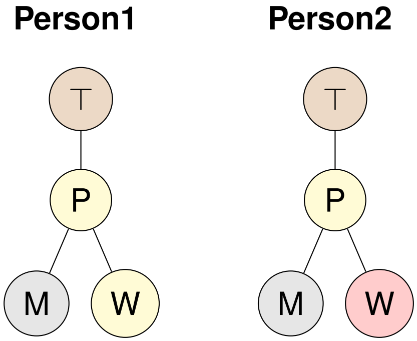
- Instantiations
  1. married(1, 2), P(1), M(1), P(2), W(2)
  2. married(1, 2), P(1), W(1), P(2), M(2)
  3. married(1, 2), P(1), W(1), P(2)

Example Algorithm Run



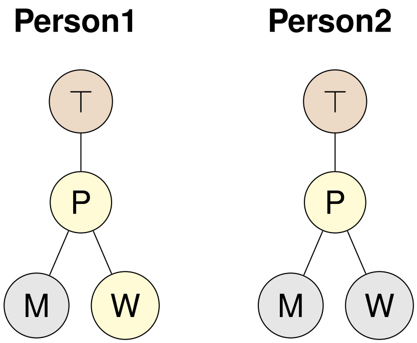
- Instantiations
  1. married(1, 2), P(1), M(1), P(2), W(2)
  2. married(1, 2), P(1), W(1), P(2), M(2)
  3. married(1, 2), P(1), W(1), P(2), W(2)

Example Algorithm Run



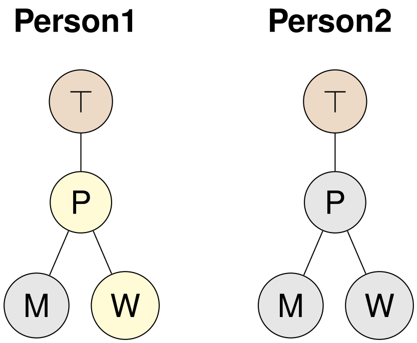
- Instantiations
  1. married(1, 2), P(1), M(1), P(2), W(2)
  2. married(1, 2), P(1), W(1), P(2), M(2)
  3. married(1, 2), P(1), W(1), P(2), W(2)

Example Algorithm Run



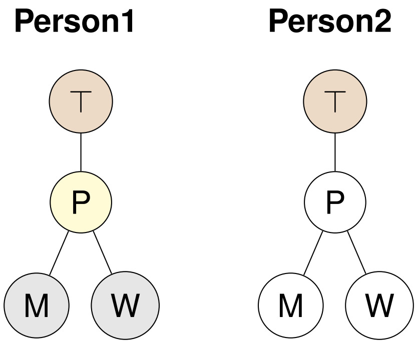
- Instantiations
  1. married(1, 2), P(1), M(1), P(2), W(2)
  2. married(1, 2), P(1), W(1), P(2), M(2)
  3. married(1, 2), P(1), W(1), P(2)

Example Algorithm Run



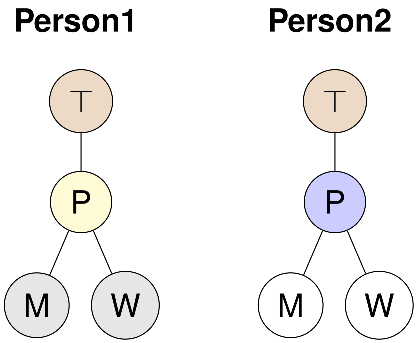
- Instantiations
  1. married(1, 2), P(1), M(1), P(2), W(2)
  2. married(1, 2), P(1), W(1), P(2), M(2)
  3. married(1, 2), P(1), W(1)

Example Algorithm Run



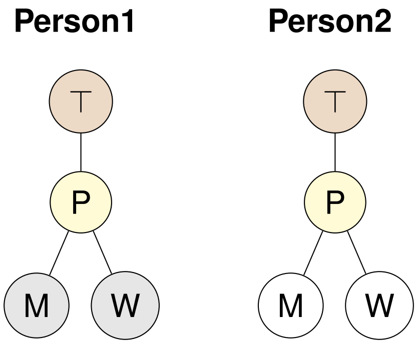
- Instantiations
  1. married(1, 2), P(1), M(1), P(2), W(2)
  2. married(1, 2), P(1), W(1), P(2), M(2)
  3. married(1, 2), P(1)

Example Algorithm Run



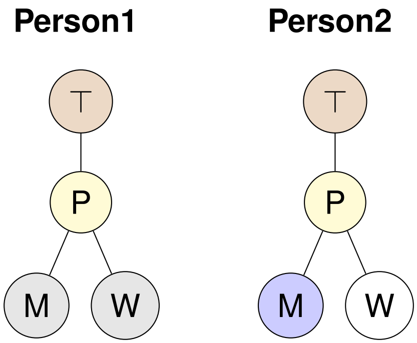
- Instantiations
  1. married(1, 2), P(1), M(1), P(2), W(2)
  2. married(1, 2), P(1), W(1), P(2), M(2)
  3. married(1, 2), P(1), P(2)

Example Algorithm Run



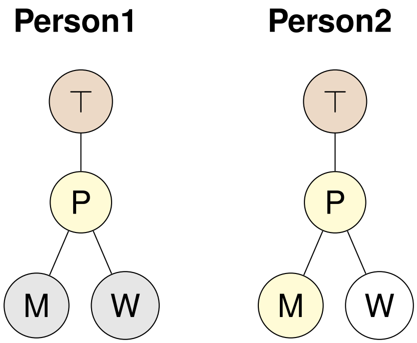
- Instantiations
  1. married(1, 2), P(1), M(1), P(2), W(2)
  2. married(1, 2), P(1), W(1), P(2), M(2)
  3. married(1, 2), P(1), P(2)

Example Algorithm Run



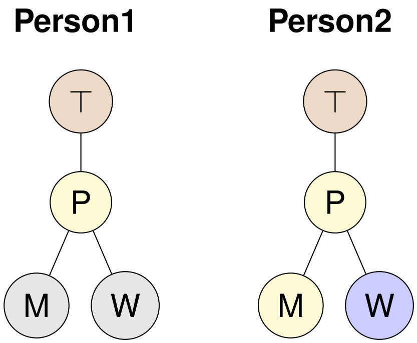
- Instantiations
  1. married(1, 2), P(1), M(1), P(2), W(2)
  2. married(1, 2), P(1), W(1), P(2), M(2)
  3. married(1, 2), P(1), P(2), M(2)

Example Algorithm Run



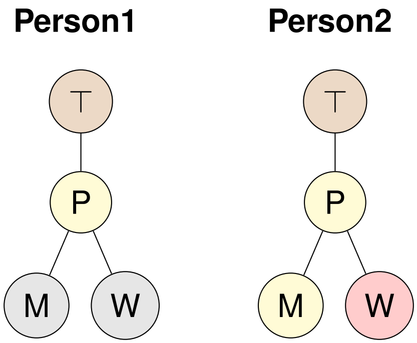
- Instantiations
  1. married(1, 2), P(1), M(1), P(2), W(2)
  2. married(1, 2), P(1), W(1), P(2), M(2)
  3. married(1, 2), P(1), P(2), M(2)

Example Algorithm Run



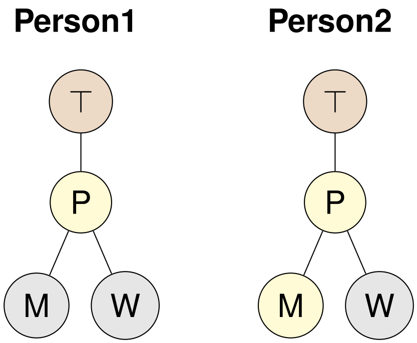
- Instantiations
  1. married(1, 2), P(1), M(1), P(2), W(2)
  2. married(1, 2), P(1), W(1), P(2), M(2)
  3. married(1, 2), P(1), P(2), M(2), W(2)

Example Algorithm Run



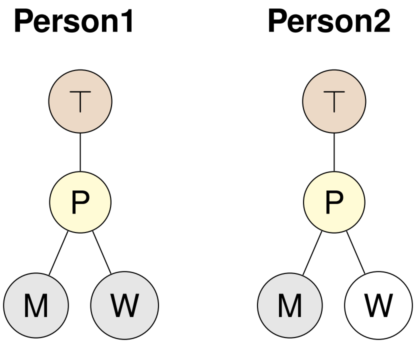
- Instantiations
  1. married(1, 2), P(1), M(1), P(2), W(2)
  2. married(1, 2), P(1), W(1), P(2), M(2)
  3. married(1, 2), P(1), P(2), M(2), W(2)

Example Algorithm Run



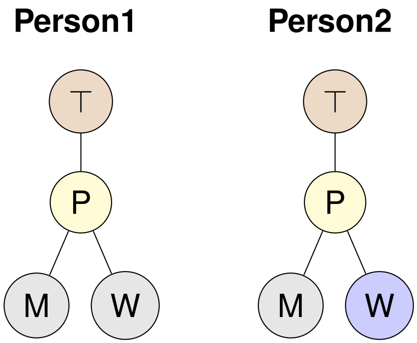
- Instantiations
  1. married(1, 2), P(1), M(1), P(2), W(2)
  2. married(1, 2), P(1), W(1), P(2), M(2)
  3. married(1, 2), P(1), P(2), M(2)

Example Algorithm Run



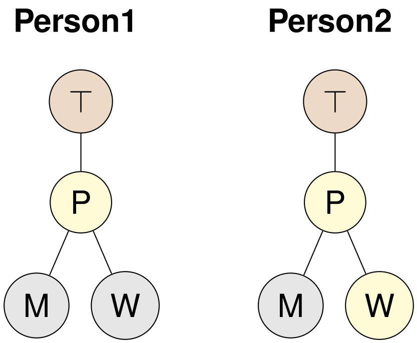
- Instantiations
  1. married(1, 2), P(1), M(1), P(2), W(2)
  2. married(1, 2), P(1), W(1), P(2), M(2)
  3. married(1, 2), P(1), P(2)

Example Algorithm Run



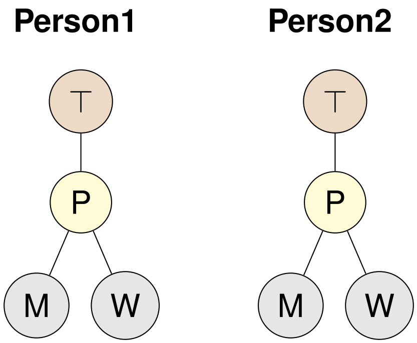
- Instantiations
  1. married(1, 2), P(1), M(1), P(2), W(2)
  2. married(1, 2), P(1), W(1), P(2), M(2)
  3. married(1, 2), P(1), P(2), W(2)

Example Algorithm Run



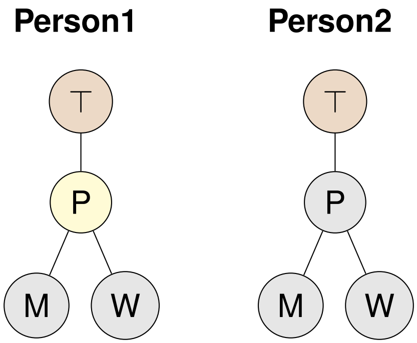
- Instantiations
  1. married(1, 2), P(1), M(1), P(2), W(2)
  2. married(1, 2), P(1), W(1), P(2), M(2)
  3. married(1, 2), P(1), P(2), W(2)

Example Algorithm Run



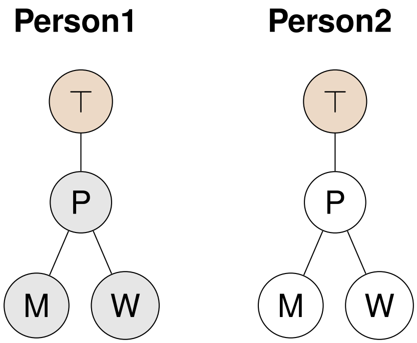
- Instantiations
  1. married(1, 2), P(1), M(1), P(2), W(2)
  2. married(1, 2), P(1), W(1), P(2), M(2)
  3. married(1, 2), P(1), P(2)

Example Algorithm Run



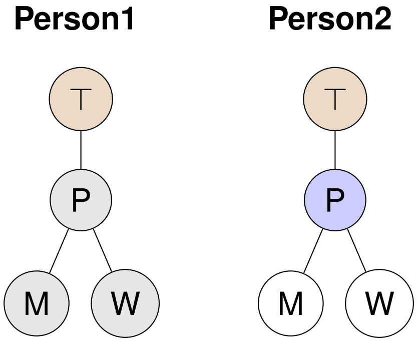
- Instantiations
  1. married(1, 2), P(1), M(1), P(2), W(2)
  2. married(1, 2), P(1), W(1), P(2), M(2)
  3. married(1, 2), P(1)

Example Algorithm Run



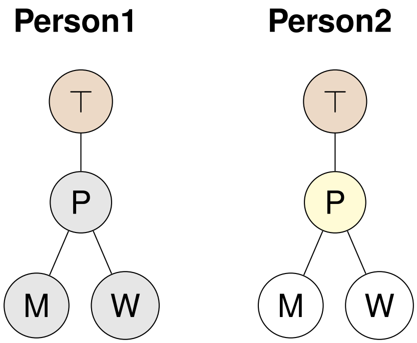
- Instantiations
  1. married(1, 2), P(1), M(1), P(2), W(2)
  2. married(1, 2), P(1), W(1), P(2), M(2)
  3. married(1, 2)

Example Algorithm Run



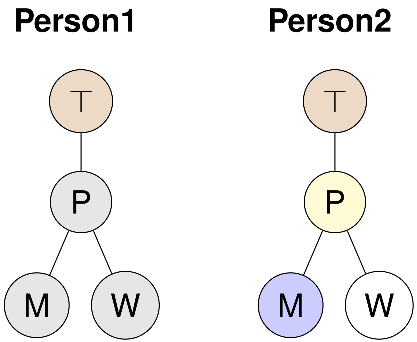
- Instantiations
  1. married(1, 2), P(1), M(1), P(2), W(2)
  2. married(1, 2), P(1), W(1), P(2), M(2)
  3. married(1, 2), P(2)

Example Algorithm Run



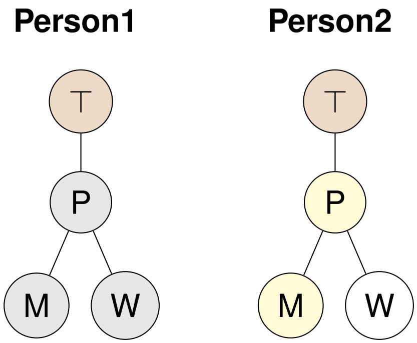
- Instantiations
  1. married(1, 2), P(1), M(1), P(2), W(2)
  2. married(1, 2), P(1), W(1), P(2), M(2)
  3. married(1, 2), P(2)

Example Algorithm Run



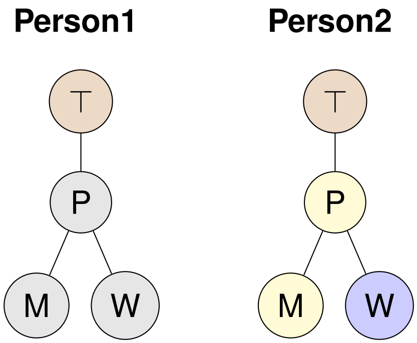
- Instantiations
  1. married(1, 2), P(1), M(1), P(2), W(2)
  2. married(1, 2), P(1), W(1), P(2), M(2)
  3. married(1, 2), P(2), M(2)

Example Algorithm Run



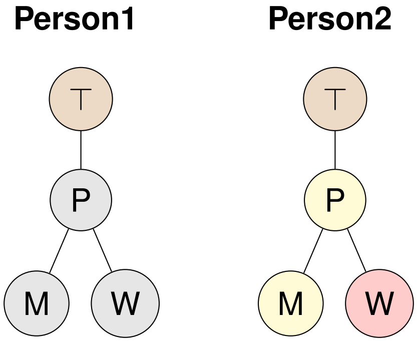
- Instantiations
  1. married(1, 2), P(1), M(1), P(2), W(2)
  2. married(1, 2), P(1), W(1), P(2), M(2)
  3. married(1, 2), P(2), M(2)

Example Algorithm Run



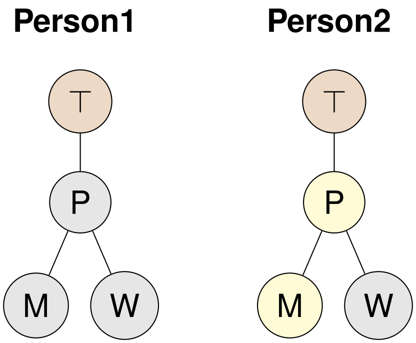
- Instantiations
  1. married(1, 2), P(1), M(1), P(2), W(2)
  2. married(1, 2), P(1), W(1), P(2), M(2)
  3. married(1, 2), P(2), M(2), W(2)

Example Algorithm Run



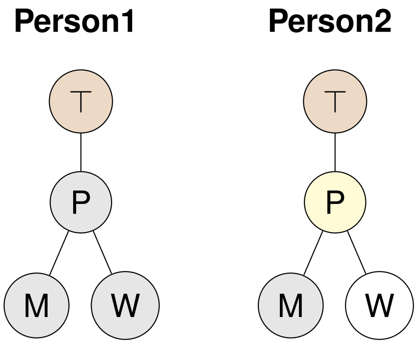
- Instantiations
  1. married(1, 2), P(1), M(1), P(2), W(2)
  2. married(1, 2), P(1), W(1), P(2), M(2)
  3. married(1, 2), P(2), M(2), W(2)

Example Algorithm Run



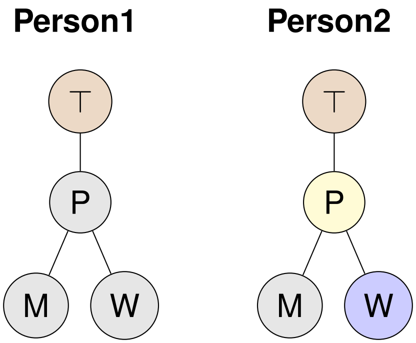
- Instantiations
  1. married(1, 2), P(1), M(1), P(2), W(2)
  2. married(1, 2), P(1), W(1), P(2), M(2)
  3. married(1, 2), P(2), M(2)

Example Algorithm Run



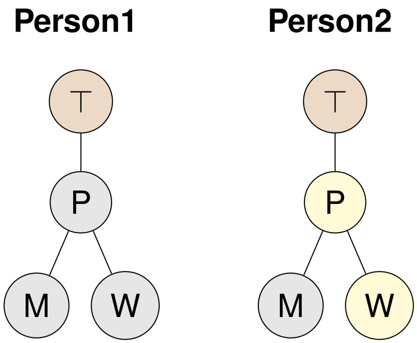
- Instantiations
  1. married(1, 2), P(1), M(1), P(2), W(2)
  2. married(1, 2), P(1), W(1), P(2), M(2)
  3. married(1, 2), P(2)

Example Algorithm Run



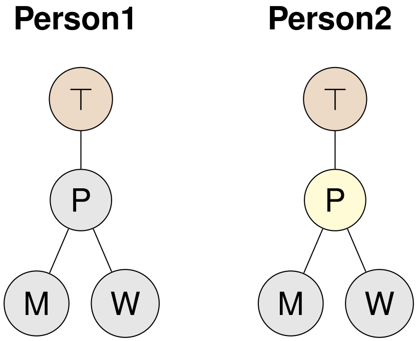
- Instantiations
  1. married(1, 2), P(1), M(1), P(2), W(2)
  2. married(1, 2), P(1), W(1), P(2), M(2)
  3. married(1, 2), P(2), W(2)

Example Algorithm Run



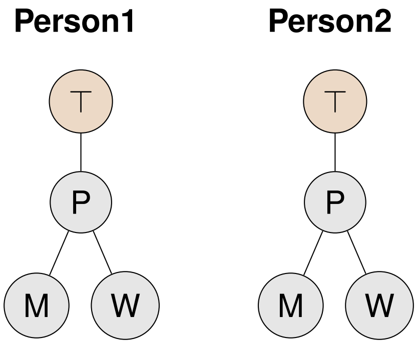
- Instantiations
  1. married(1, 2), P(1), M(1), P(2), W(2)
  2. married(1, 2), P(1), W(1), P(2), M(2)
  3. married(1, 2), P(2), W(2)

Example Algorithm Run



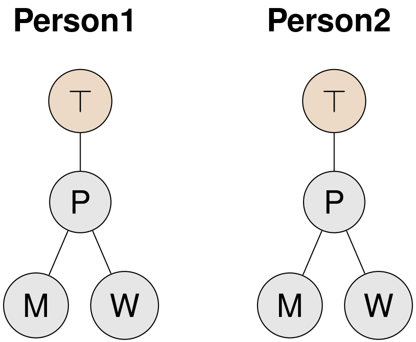
- Instantiations
  1. married(1, 2), P(1), M(1), P(2), W(2)
  2. married(1, 2), P(1), W(1), P(2), M(2)
  3. married(1, 2), P(2)

Example Algorithm Run



- Instantiations
  1. married(1, 2), P(1), M(1), P(2), W(2)
  2. married(1, 2), P(1), W(1), P(2), M(2)
  3. married(1, 2)

Example Algorithm Run



- Instantiations
  1. P(1), P(2), married(1, 2), M(1), W(2)
  2. P(1), P(2), married(1, 2), W(1), M(2)

Concretization Summary

How can all the legal concretizations of an underdetermined world be generated using OWL?

- Reasoner as an oracle:
    - Consistent ontology means assignments are legal
    - Verify value assignments
    - Verify attribute assignments
    - Verify object assignments
- Power set inspired traversal of class hierarchy
    - All possible combinations of class assertions
- Assertion order

Algorithm Correctness

- Soundness of results:
  - Only consistent maximal combinations
- Completeness of results:
  - All consistent maximal combinations

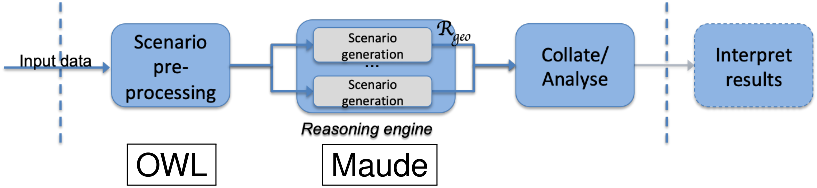
Outline

Conversion

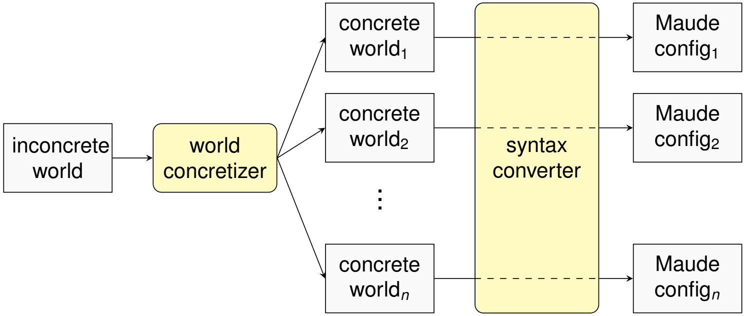
RQ3: Converting from OWL to Maude

- Influenced by the implementation of the Geological Assistant
- Inspired a tool independent representation of worlds
- Will briefly show
  - How the conversion fits into the pipeline
  - Mapping from OWL to Maude

Geological Assistant Pipeline



Maude Conversion



Mapping

OWL		World		Maude
Individual Name	$\rightarrow$	Object ID (*id*)	$\rightarrow$	Oid
Object/Data Properties	$\rightarrow$	Binary Relations	$\rightarrow$	Binary Operators
Value Class	$\rightarrow$	Value Name	$\rightarrow$	Object Attribute Value
Value Set Class	$\rightarrow$	Value Set Name	$\rightarrow$	Attribute Name
Object Class	$\rightarrow$	Object Name	$\rightarrow$	Object Name

Outline

Results

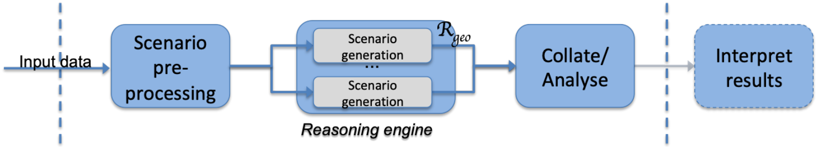
- Test results correct
- Long execution time
  - Poor algorithm complexity
- Room for optimizations
  - Methodology specific?
  - Implementation specific
- Helpful to review semantics of ontology

Related Work

- Configuration design
- Geological Assistant research

Summary

- Many possible states based on domain knowledge
- Generate states as a starting point for scenario generation
- Represent the facts and domain knowledge with OWL
- Utilize OWL reasoner's consistency checking



Appendix

Algorithm 1 Consistent Maximal ABox Generator

```
1: procedure NEXTINDIVIDUAL($individuals$, $comb$)
2:     $individuals_1 \leftarrow individuals$.copy()
3:     $u \leftarrow individuals_1.remove(0)$
4:     $queue \leftarrow$ QUEUEUPDATER(empty queue, $\top$)
5:     TREETRAVERSE($individuals_1$, $queue$, $u$, $comb$)
```

```
procedure TREETRAVERSE($individuals$, $queue_1$, $u$, $comb_1$)
    if $|queue_1| = 0$ then
        if $|individuals| > 0$ then
            NEXTINDIVIDUAL($individuals$, $comb_1$)
        else if $\neg(comb_1 \subseteq c)$, where $c \in combs$ then
            $combs$.add($comb_1$)
        return
    $class \leftarrow$ pop $queue_1$
    $axiom \leftarrow$ assert $class(u)$
    $axiomWasInOntology \leftarrow$ ontology.contains(axiom)
```

```
procedure …
    if $\neg$axiomWasInOntology then
        ontology.add(axiom)
        synchronize reasoner

    if ontology is consistent then
        $comb_2 \leftarrow comb_1$.copy().add(axiom)
        $queue_2 \leftarrow$ QUEUEUPDATER($queue_1$, class)
        TREETRAVERSE(individuals, $queue_2$, u, $comb_2$)

    if $\neg$axiomWasInOntology then
        ontology.remove(axiom)
        synchronize reasoner
    TREETRAVERSE(individuals, $queue_1$, u, $comb_1$)
```

```
1: procedure QUEUEUPDATER($queue$, $class$)
2:     $temp \leftarrow queue$.copy()
3:     for all ($sub \sqsubseteq class$), where $sub$ is a direct subclass of $class$ do
4:         add $sub$ to $temp$
5:     return $temp$
```

Proof Lemma 1

All combinations *c* stored in the variable combs at the end of a run of the algorithm are consistent.

Proof Lemma 2

At the end of the algorithm, there exists no combinations b and c such that $b \in combs$, $c \in combs$ and $b \subset c$.

Proof Lemma 3

Any consistent combination b, with no consistent combination c such that $b \subset c$, is stored in the variable combs.

Why does it work?

- Soundness: all combinations returned are legal and complete
  - Complete: only combinations that are not subset of others
  - Legal: only combinations part of consistent ontologies
- Completeness: all combinations that are legal and complete are returned
  - At any base case of the algorithm, the combination is not a subset of any combinations generated later
- Note: for the generated results to be correct, it is important that the model is semantically correct

Only Complete Combinations

- Two factors:
  - Order of combination generation
  - Subset check before storing in the base case
- Try all combinations with an assertion before trying all combinations without the assertion
- Makes sure no combination is a subset of a combination that is generated later

Order Example

- Given the example graph $\{P, M, F\}$, the combinations are generated in the following order
- $\{P, M, F\}$
- $\{P, M\}$
- $\{P, F\}$
- $\{P\}$
- $\{M, F\}$
- $\{M\}$
- $\{F\}$
- $\{\}$

Abstract Representation

- Representation of the married couple:
- $World = (objects, relations)$:
  - $Family = (\{P_1, P_2\}, \{married(1, 2)\})$
- $Object = (ID, \{attributes\})$:
  - $P_1 = (1, \{gender_1\})$
  - $P_2 = (2, \{gender_2\})$
- $attribute \Leftarrow \{value_1, ..., value_n\}$:
  - $sex \Leftarrow \{man, woman\}$