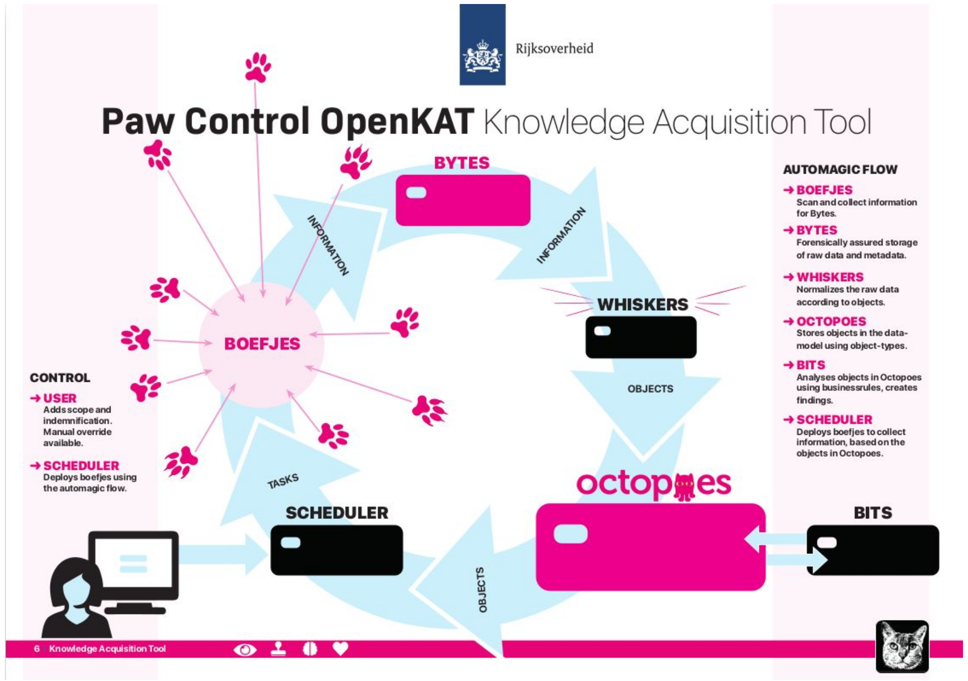
Investigating the use of Semantic Web Technologies in OpenKAT

Maurits Merks
Patrick Darwinkel
Perry van der Zande
Rachelle Bouwens

MSc Artificial Intelligence; MSc Information Science

Introduction

- Ministry of VWS is working on OpenKAT, a tool to monitor and scan networked entities for vulnerabilities
- Currently: No formal modelling of the data model and rules
- A formalized ontology can help with identifying inconsistencies



Paw Control OpenKAT Knowledge Acquisition Tool

CONTROL

→ USER
Adds scope and indemnification. Manual override available.

→ SCHEDULER
Deploys boefjes using the automagic flow.

AUTOMAGIC FLOW

→ BOEFJES
Scan and collect information for Bytes.

→ BYTES
Forensically assured storage of raw data and metadata.

→ WHISKERS
Normalizes the raw data according to objects.

→ OCTOPOES
Stores objects in the data-model using object-types.

→ BITS
Analyses objects in Octopoes using businessrules, creates findings.

→ SCHEDULER
Deploys boefjes to collect information, based on the objects in Octopoes.

6 Knowledge Acquisition Tool

About OpenKAT

- How it works
  1. Continuously scan the world through (security) tools
  2. Normalize the results from those tools into a common knowledge graph
  3. Reason about the graph and infer new objects and findings
  4. Repeat

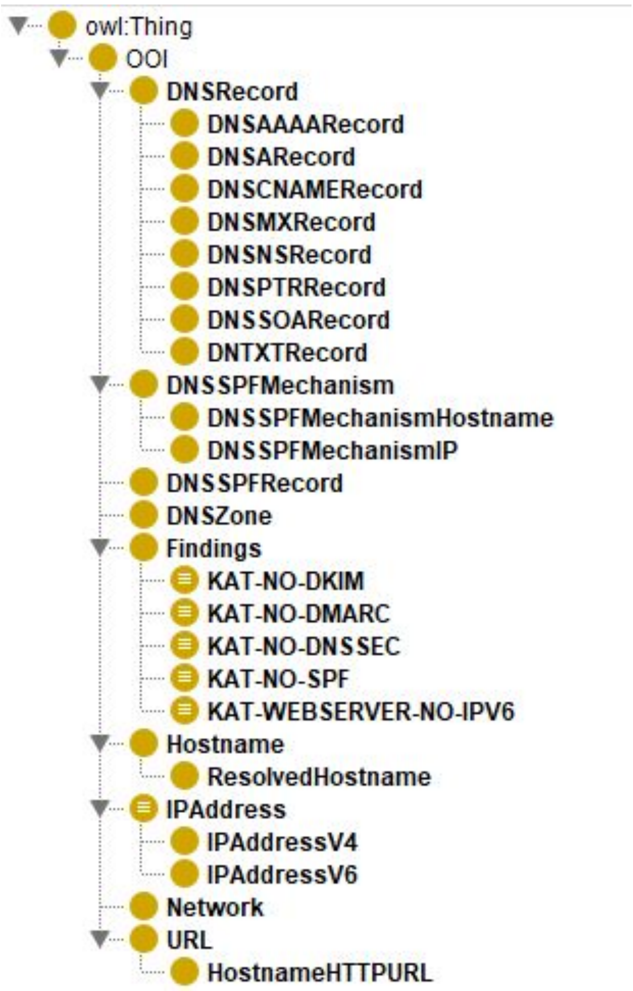
Research questions

- Can we reach feature-parity on a subset of KAT with Semantic Web technologies?
- What (if any) problems arise when modelling KAT with Semantic Web technologies?

Ontology: Classes

- Reflect the network components found in `mispo.es`
  - DNS Records
  - IP Addresses
  - Hostnames

- Additional classes made for findings



- owl:Thing
  - OOI
    - DNSRecord
      - DNSAAAARecord
      - DNSARecord
      - DNSCNAMERecord
      - DNSMXRecord
      - DNSNSRecord
      - DNSPTRRecord
      - DNSSOARecord
      - DNTXTRecord
    - DNSSPFMechanism
      - DNSSPFMechanismHostname
      - DNSSPFMechanismIP
    - DNSSPFRecord
    - DNSZone
    - Findings
      - KAT-NO-DKIM
      - KAT-NO-DMARC
      - KAT-NO-DNSSEC
      - KAT-NO-SPF
      - KAT-WEBSERVER-NO-IPV6
    - Hostname
      - ResolvedHostname
    - IPAddress
      - IPAddressV4
      - IPAddressV6
    - Network
    - URL
      - HostnameHTTPURL

Ontology: Properties

Object Properties:

- Standard Networking
  - hasHostname
  - hasDNSRecord
  - And more!
- In total, 17 object properties
  - Some are not present in a network: noDNSAAAARecordFound

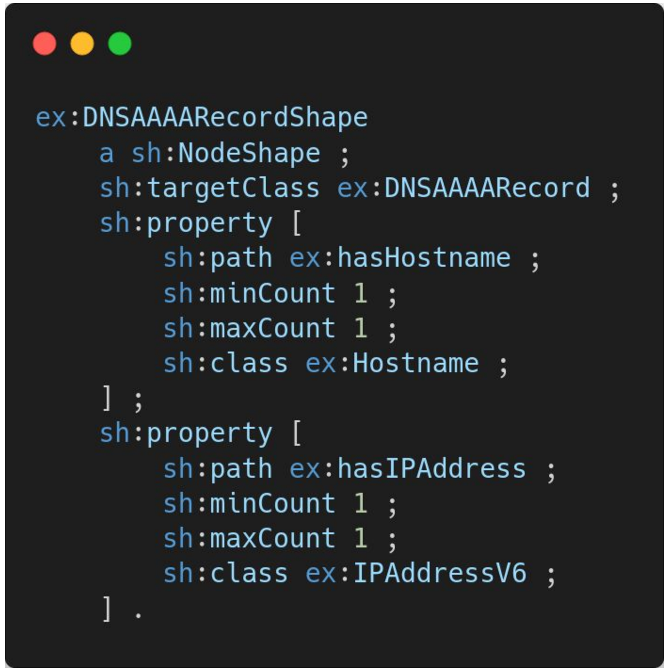
Data Properties:

- IP address values, Port numbers…
- Not all the information on a network
- But enough to make discoveries
- 17 data properties

SHACL

- SHACL shapes were created for each class
- Shapes enforce conditions to prevent mistakes

- Example: DNSAAAARecord
  - Has object properties:
    - hasHostname
    - hasIPAddress



```
ex:DNSAAAARecordShape
    a sh:NodeShape ;
    sh:targetClass ex:DNSAAAARecord ;
    sh:property [
        sh:path ex:hasHostname ;
        sh:minCount 1 ;
        sh:maxCount 1 ;
        sh:class ex:Hostname ;
    ] ;
    sh:property [
        sh:path ex:hasIPAddress ;
        sh:minCount 1 ;
        sh:maxCount 1 ;
        sh:class ex:IPAddressV6 ;
    ] .
```

SWRL

- Write rules as first-order logic

-
```
Hostname(?h) ^ DNSRecord(?r) ^ IPAddress(?a)
^ hasIPAddress(?r, ?a) ^ hasHostname(?r, ?h)
-> resolvesToAddress(?h, ?a)
```

Results

1. Weakened inference due to OWA
2. Inferring existence of new individuals
3. Limited flexibility in rules
4. Primary keys and distributed ontologies
5. Modelling scan levels

1. Weakened inference due to OWA

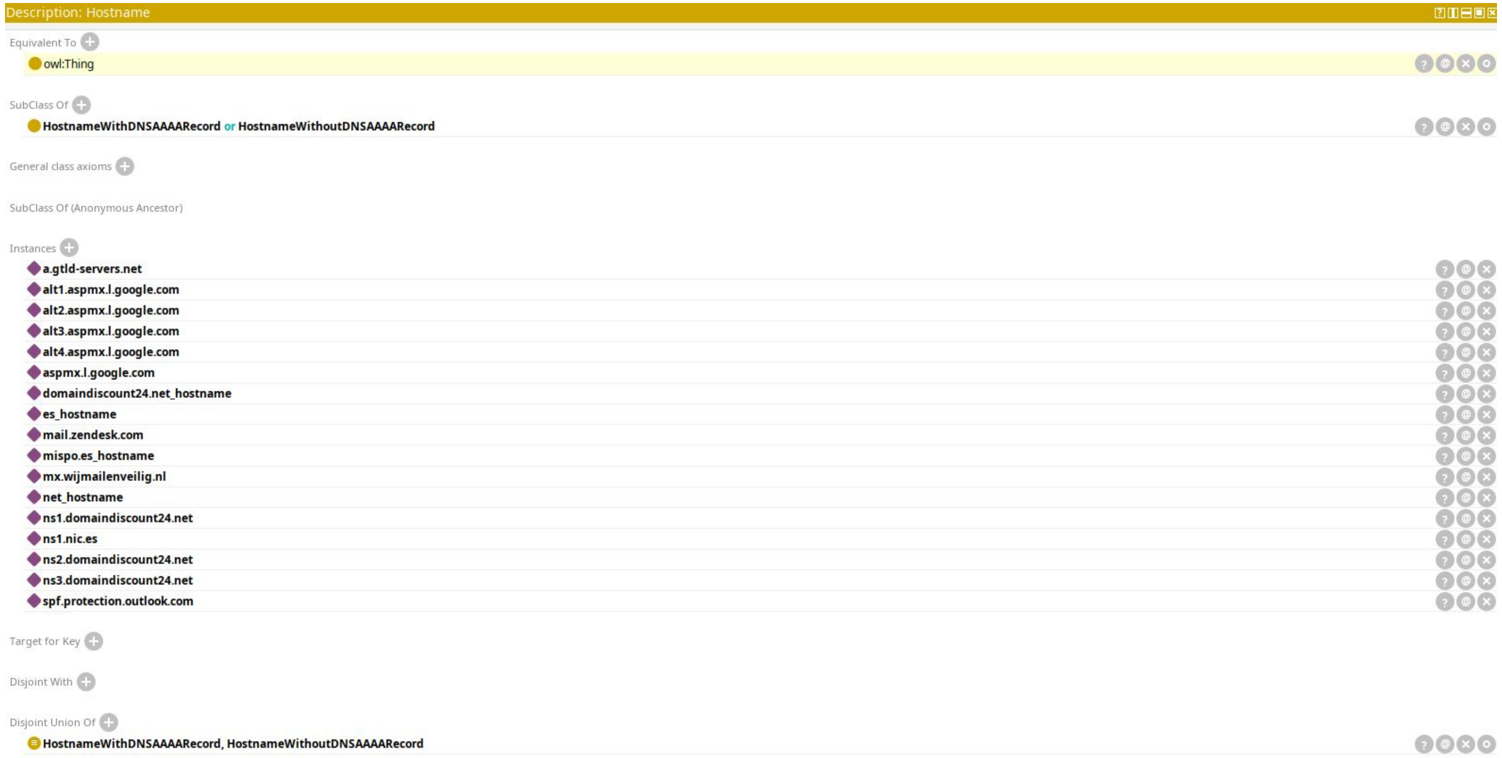
- Open World Assumption in OWL

The absence of evidence is not evidence of absence

Basically, you can't use classical logical negation in OWL (requires closed world)

1. Weakened inference due to OWA

› Negation is very hard



Description: Hostname

Equivalent To
- owl:Thing

SubClass Of
- HostnameWithDNSAAAARecord or HostnameWithoutDNSAAAARecord

General class axioms

SubClass Of (Anonymous Ancestor)

Instances
- a.gtld-servers.net
- alt1.aspmx.l.google.com
- alt2.aspmx.l.google.com
- alt3.aspmx.l.google.com
- alt4.aspmx.l.google.com
- aspmx.l.google.com
- domaindiscount24.net_hostname
- es_hostname
- mail.zendesk.com
- mispo.es_hostname
- mx.wijmailenveilig.nl
- net_hostname
- ns1.domaindiscount24.net
- ns1.nic.es
- ns2.domaindiscount24.net
- ns3.domaindiscount24.net
- spf.protection.outlook.com

Target for Key

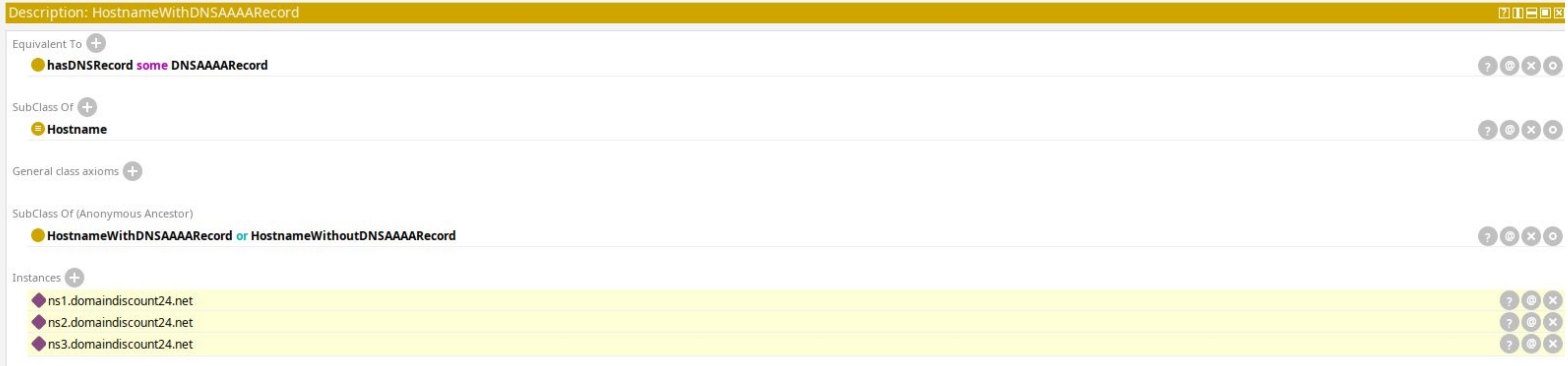
Disjoint With

Disjoint Union Of
- HostnameWithDNSAAAARecord, HostnameWithoutDNSAAAARecord

1. Weakened inference due to OWA

- Negation is very hard

Correct and useful:



Description: HostnameWithDNSAAAARecord

Equivalent To
hasDNSRecord some DNSAAAARecord

SubClass Of
Hostname

General class axioms

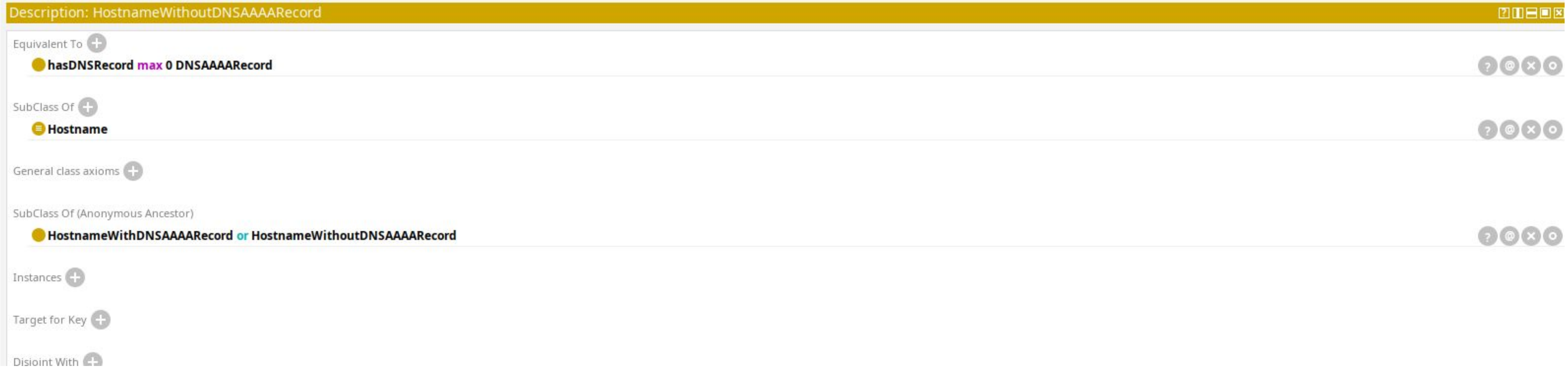
SubClass Of (Anonymous Ancestor)
HostnameWithDNSAAAARecord or HostnameWithoutDNSAAAARecord

Instances
ns1.domaindiscount24.net
ns2.domaindiscount24.net
ns3.domaindiscount24.net

1. Weakened inference due to OWA

- Negation is very hard

Correct but useless:



Description: HostnameWithoutDNSAAAARecord

Equivalent To

● hasDNSRecord max 0 DNSAAAARecord

SubClass Of

● Hostname

General class axioms

SubClass Of (Anonymous Ancestor)

● HostnameWithDNSAAAARecord or HostnameWithoutDNSAAAARecord

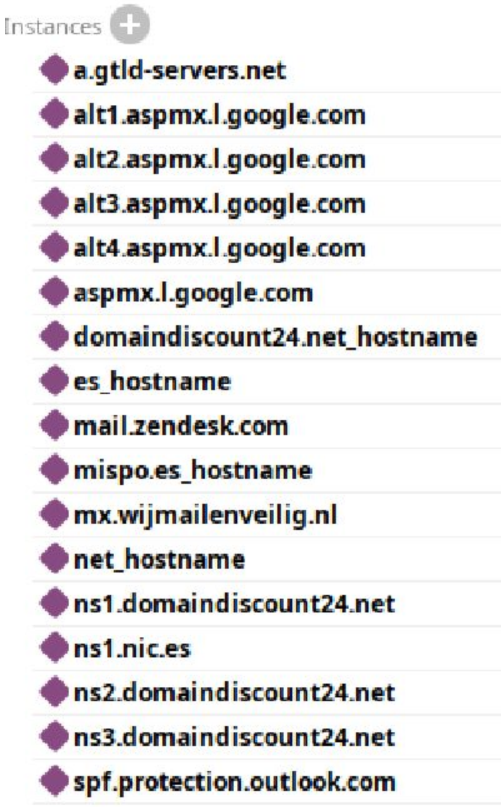
Instances

Target for Key

Disjoint With

1. Weakened inference due to OWA

› Where did all the other instances go?



Instances

- a.gtld-servers.net
- alt1.aspmx.l.google.com
- alt2.aspmx.l.google.com
- alt3.aspmx.l.google.com
- alt4.aspmx.l.google.com
- aspmx.l.google.com
- domaindiscount24.net_hostname
- es_hostname
- mail.zendesk.com
- mispo.es_hostname
- mx.wijmailenveilig.nl
- net_hostname
- ns1.domaindiscount24.net
- ns1.nic.es
- ns2.domaindiscount24.net
- ns3.domaindiscount24.net
- spf.protection.outlook.com

1. Weakened inference due to OWA

- Problem when modelling Findings:
  - KAT-MISSING-DKIM
  - KAT-MISSING-SPF
  - KAT-MISSING-DMARC

- These are based on the absence of records

1. Weakened inference due to OWA

- Problem when modelling Findings:
  - KAT-MISSING-DKIM
  - KAT-MISSING-SPF
  - KAT-MISSING-DMARC

- These are based on the absence of records

- …..actually a deeper problem with KAT!

1. Weakened inference due to OWA

- What if you disable all active scanning in a fresh install?
- Then create `mispo.es` objects manually

1. Weakened inference due to OWA

- What if you disable all active scanning in a fresh install?
- Then create `mispo.es` objects manually

- KAT concludes:
  - `Hostname mispo.es` has missing DKIM, SPF, and DMARC

1. Weakened inference due to OWA

- What if you disable all active scanning in a fresh install?
- Then create `mispo.es` objects manually

- KAT concludes:
  - `Hostname mispo.es` has missing DKIM, SPF, and DMARC

- …..but KAT didn't actually bother to scan anything
- If these records actually exist, then the claim is wrong

1. Weakened inference due to OWA

- Potential "solutions":
  - Assert explicit object properties such as `NoDNSAAAARecordFoundYet`
  - Use reasoner which treats negation-as-failure

2. Inferring new individuals

- SWT cannot infer existence of new individuals
  - KAT does, however

- But this is not necessarily a problem

2. Inferring new individuals

- `Hostname(?h) ^ DNSRecord(?r) ^ IPAddress(?a) ^ hasIPAddress(?r, ?a) ^ hasHostname(?r, ?h) -> resolvesToAddress(?h, ?a)`

- Above implements the `dns-resolving` bit
- Makes `ResolvedHostname` instance and class redundant!

3. Limited flexibility in rules

- `url-classification` essentially decomposes a URL into a schema, hostname, and port
- https://mispo.es -> https, mispo.es, 443
- ….but how to model in SWRL?

3. Limited flexibility in rules

- Built-in operators for strings and lists

```
› swrlb:stringConcat(?uri, "https", "://", "mispo.es")
› swrlb:stringConcat(?uri, ?scheme, "://", ?hostname)
```

- Resolver doesn't work with the second one
- Hardcoding makes it convoluted to implement
- Many KAT bits have to parse text

4. Primary keys and distributed ontologies

- KAT uses natural keys (functional determinants) to determine unique objects
- Nice for deduplication and storage optimization
- But what if class definitions change?

4. Primary keys and distributed ontologies

- KAT uses natural keys (functional determinants) to determine unique objects
- Nice for deduplication and storage optimization
- But what if class definitions change?

- Immutability of certain properties may make revisions/updates tricky
- KAT has no way to validate schema changes

4. Primary keys and distributed ontologies

- OWL references individuals by IRI
- But different IRI's may refer to the same real-world individual
- IRI's have no semantic or logical meaning
- This is perfectly fine in SWT

4. Primary keys and distributed ontologies

- Mixing ontologies (IRI's) is first-class feature of SWT
- ….provided they don't contradict each other
  - But SWT has integrated consistency checking!

4. Primary keys and distributed ontologies

- `kat:IPAddress`
  - `kat:IPAddressV4`
  - `kat:IPAddressV6`

- What if we have a third party ontology
- OpenHOND: Handige Objectgeoriënteerde Netwerk Dumper

4. Primary keys and distributed ontologies

- `kat:IPAddress`
  - `kat:IPAddressV4`
  - `kat:IPAddressV6`
  - `hond:IPAddressV8`

- Declare `hond:IPAddressV8` subclass of `kat:IPAddress` and disjoint with `kat:IPAddressV4` and `kat:IPAddressV6`

- All axioms and rules that apply to `kat:IPAddress` now also apply to `hond:IPAddressV8`

4. Primary keys and distributed ontologies

- `kat:IPAddress`
  - `kat:IPAddressV4`
  - `kat:IPAddressV6`
  - `hond:IPAddressV4`

- Declare `hond:IPAddressV4` equivalent to `kat:IPAddressV4`

- All revisions and changes that apply to `hond:IPAddressV4` now also apply to `kat:IPAddressV4`

4. Primary keys and distributed ontologies

- SWT provides a reliable way to integrate various (versioned) ontologies
- Inconsistencies and contradictions are immediately picked up by the reasoner

5. Modelling scan levels

- Basically, KAT's scan level inheritance can be modelled through SWT
- Actually quite straightforward to implement basic algorithm
- However, limitations in SWRL makes it convoluted
  - Such as lack of `min(a, b)` function

Conclusion

- It is possible to model KAT with SWT
  - (Based on the subset we attempted to implement)

- However, non-trivial:

  - Very specialized knowledge required
    - Goes beyond Python programming

  - Limited flexibility may make certain rules very convoluted to implement

  - Will have to decide how to tackle OWA

Future work?

- Rewrite findings to use positive evidence (OWA)
- Investigate atomic functions for e.g. string parsing
- Investigate Datalog/Prolog for reasoning
- Consider implementation-specific workarounds
  - See e.g. RDFox, which has negation-as-failure

Thank you for your attention

More details and explanations can be found in the research paper