TECHNISCHE UNIVERSITÄT DRESDEN

DALHOUSIE UNIVERSITY

Notation3 Logic: A Practical Introduction

Dörthe Arndt and William Van Woensel

Outline

N3 and the Web

Reasoning with N3

Applications

Outline

N3 and the Web

Reasoning with N3

Applications

Semantic Web

Idea: provide a machine-understandable version of the Web

Why? A Semantic Web enables computers to **use** the Web for you.

How? Logical Representation of knowledge in graphs.

Resource Description Framework (RDF)

Simple triples:

`:William :likes :spaghetti.` -> *“William likes spaghetti”*

Blank nodes:

`_:x :likes :spaghetti.` -> *“Someone likes spaghetti.”*

Notation3 Logic

Extension of RDF

- Rules
- Quotations
- Built-ins

Rules

```
{:William :likes :spaghetti.}=>{:William :likes :pizza.}.
```

"If William likes spaghetti, he also likes pizza."

Universal Variables

```
{?x :likes :spaghetti.}=>{?x :likes :pizza.}.
```

"If someone likes spaghetti, this person also likes pizza."

Quotations of formulae

```
:Doerthe :thinks {:William :likes spaghetti, :pizza}.
```

"Doerthe thinks that William likes spaghetti and pizza."

N3 quotation vs RDF-star

N3 quotation is for graphs:

```
:Doerthe :thinks {:William :likes spaghetti, :pizza}.
```

RDF-star is for triples:

```
:Doerthe :thinks <<:William :likes spaghetti>>.
```

RDF-star and N3 quotation are compatible, some reasoners support both.

Built-ins

Predicates with special meanings

"Traditional" N3 built-ins:

https://www.w3.org/2000/10/swap/doc/CwmBuiltins

"New" built-ins are currently discussed.

Outline

N3 and the Web

Reasoning with N3

Applications

Reasoners

Classical Reasoners: EYE and Cwm

Recent developments: jen3, N03, ...

We use **EYE** in our examples

Queries and Closure

N3 reasoners can give either

the **deductive closure** of an N3 graph or the **result of a query**

Deductive Closure

Given:

```
:William :likes :spaghetti.
{?x :likes :spaghetti.} => {?x :likes :pizza}.
```

Closure:

```
:William :likes :spaghetti.
{?x :likes :spaghetti.} => {?x :likes :pizza}.
:William :likes :pizza.
```

The **deductive closure** is the set of **all triples** which can be **derived** from a dataset

Query result

Given:

Facts and rules: `William :likes :spaghetti.`

`Query: {?x :likes :spaghetti.} => {?x :likes :pizza}.`

Result:

`:William :likes :pizza.`

A query is a special **rule**. Query reasoning provides all the **results of the rule marked as query**. The reasoning process can include other rules as well.

Example

Example execution in EYE (https://github.com/josd/eye)

From now on we will only consider deductive closures.

N3 Online editor

We provide an N3 online editor: http://ppr.cs.dal.ca:3002/n3/editor/

Spaghetti-example: http://ppr.cs.dal.ca:3002/n3/editor/s/4yxZnYnt

Existential rules

```
:William :likes :spaghetti.

{?x :likes :spaghetti.} => {?x :loves _:z}.

:William :loves _:z.
```

Link to example: http://ppr.cs.dal.ca:3002/n3/editor/s/E303UvWf

Rule-producing rules

```
:Researcher rdfs:subClassOf :Person.

{?x rdfs:subClassOf ?y.} => {{?z a ?x}=>{?z a ?y}}.

{?z a :Researcher}=>{?z a :Person}.
```

Link to example: http://ppr.cs.dal.ca:3002/n3/editor/s/XK5icom2

Forward vs. Backward Chaining (in EYE)

In N3 you can indicate how a rule should be applied:

Forward-chaining:

```
{?x :likes :Spaghetti}=>{?x :likes :Pizza}.
```

Backward-chaining:

```
{?x :likes :Pizza}<={?x :likes :Spaghetti}.
```

Link: http://ppr.cs.dal.ca:3002/n3/editor/s/rQEyElOC

Negation predicates

Different predicates which can express (scoped) negation as failure.

Example: **log:collectAllIn**

Built-in that collects all occurrences of a pattern in a list.

Link: http://ppr.cs.dal.ca:3002/n3/editor/s/4wCFscuz

Link traversal

The built-in **log:semantics** allows us, to access the content behind a uri.

```
:doerthe foaf:knows git:william.n3 .

{   ?x foaf:knows ?y.
    ?y log:semantics ?content.
}=>{
    ?x :friendInfo ?content
    }.
```

Try it: http://ppr.cs.dal.ca:3002/n3/editor/s/uNpYRH12

Graph operations

The predicate **log:includes** allows us to search for patterns in graphs.

```
{:william :likes :Spaghetti, :Pizza, :Fish.}
                        log:includes {:william :likes :Pizza}.
```

Examples:

- http://ppr.cs.dal.ca:3002/n3/editor/s/HPX9ZCKY
- http://ppr.cs.dal.ca:3002/n3/editor/s/BFg3hpTT

Proofs

N3 reasoners produce **proofs**.

```
[] a r:Proof, r:Conjunction;
     r:component <#lemma1>;
     r:gives {:William :likes :pizza.}.

<#lemma1> a r:Inference;
     r:gives { :William :likes :pizza.};
     r:evidence ( <#lemma2> );
     r:rule <#lemma3>.

<#lemma2> a r:Extraction;
     r:gives { :William :likes :spaghetti.};
     r:because [ a r:Parsing; r:source <William.n3>].

<#lemma3> a r:Extraction;
     r:gives { {?x_0_1 :likes :spaghetti} => {?x_0_1 :likes :pizza}.};
     r:because [ a r:Parsing; r:source <spaghetti_rule.n3>].
```

Outline

N3 and the Web

Reasoning with N3

Applications

Using N3 as Lingua Franca for Clinical Decision Support

Interoperable Electronic Health Records (EHR)

- Open EHR standards: **HL7 FHIR** (RDF), openEHR

- Use biomedical ontologies to annotate data

⇒ N3 does not suffer impedance mismatch

- Natively refer to OWL classes, HL7 FHIR resources

- Records vs. facts:
  - *HL7 FHIR*: keeps records (Dr. X diagnosed Y with Z)
  - *RDF*: stating of absolute facts (Y has Z)

⇒ Use quoted graphs in N3

- Quote and describe graphs of statements

```
{ :patientY :has :viral_pneumonia }
    :diagnosed_by :doctorX ;
    :diagnosed_on "22-02-2022" .
```

Model-driven UIs using N3 + HL7 FHIR

```
diary_observations a fhir:PlanDefinition ;
    fhir:PlanDefinition.action ( [
    fhir:PlanDefinition.action.definitionUri :diagnose_cough_wheezing_stridor ] ... ) ]

:diagnose_cough_wheezing_stridor a fhir:ActivityDefinition ;
    fhir:ActivityDefinition.title "Is your cough, wheezing or stridor less, the same ... " ;
    fhir:ActivityDefinition.observationResultRequirement [
        fhir:ObservationDefinition.code :code_cough ; ...
        fhir:ObservationDefinition.permittedDataType :dt_int, :1_10_scale ] .
```

⇒ Generate UI for health data input[1]

- Applies validation constraints (input ranges, datatypes)
- Submits validated, self-contained EHR record
- Currently, supporting HTML+RDFa as UI format

1 http://ceur-ws.org/Vol-3055/paper4.pdf

Model-driven UIs using N3 + HL7 FHIR (2)

- UiTemplates
  - Select certain types of *health data descriptions*
  - Select appropriate *UI code* snippet (HTML+RDFa)
  - Instantiate the *UI code* from *health data description*

```
:tpl-int_range_field a tpl:UiTemplate ;
  tpl:select :select-int_range_field
  tpl:generate [
    tpl:ui :ui-int_input ;
    tpl:placeholders ('_code_' '_prefix_' '_id_' '_min_' '_max_' '_step_' '_suffix_');
    tpl:values ( ?code ?label ?id ?min ?max ?step ?range )
  ] .
```

```
{ ?req fhir:permittedDataType :dt_int .
  ?range!fhir:low fhir:value ?min .
  ?range!fhir:high fhir:value ?max .. }
```

Introspection and Rule Generation

- N3 rule introspects and outputs UI-generating rule

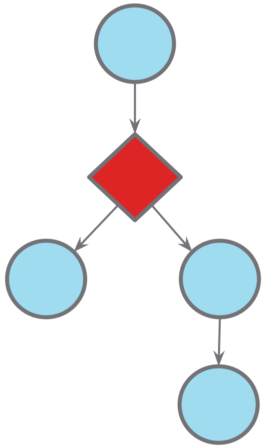
```
{ ?tpl a tpl:UiTemplate ;
    tpl:select [ rdf:value ?selection ] ;
    tpl:generate [
      tpl:ui ?element ; tpl:code ?ui_code ;
      tpl:placeholders ?placeholders ; tpl:values ?values ] .
  ( ?selection
   { ?element tpl:code ?ui_code .
     ( ?ui_code ?placeholders ?values ) string:replaceAll ?output }
  ) log:conjunction ?premise .

} => { ?premise => { ?element tpl:generated ?output } } .
```

pattern-match with UiTemplate

replace placeholders with selector values

Evidence-based CDS using N3

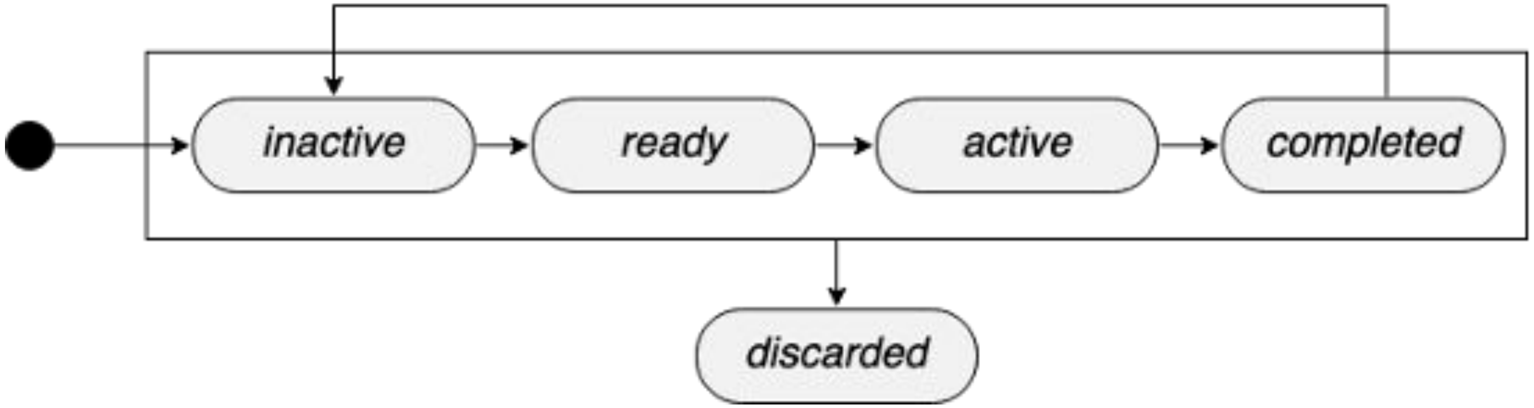
- Requires computerization of clinical guidelines
  - PROForma, Asbru, GLIF3, GASTON, CIG Ontology, ..
- Most CDS support a Task-Network Model (TNM)
  - Workflow with sequential, parallel, composite, decision, .. tasks
  - Typically have additional focus (intentions, temporal, ..)
- But, no formal execution semantics
  - Makes it difficult to re-use, extend the work



⇒ Use N3 rules to define a Finite State Machine
  - Make available as resource for others to re-use and build on
  - https://github.com/william-vw/glean

Evidence-based CDS: Finite State Machine

- Finite State Machine (FSM):
  - Tasks can be in a finite number of states (one at a time)



  - Transition between these states, depending on conditions
    - Decisional crite ria, other task states, temporal constraints, ..

High-level State Transition Formalism

```
{ { ?e1 :next ?e2 . ?e1 state:in :Completed } rdf:type state:Guard .
  ?e2 state:in :Inactive
} state:transit { ?e2 state:in :Ready } ;
  state:reason :readyNextOfCompletedEntity .

{ { ?composite a :CompositeTask ; state:in :Active ; :subTask ?sub .
    <> log:notIncludes { ?prev :next ?sub }
  } a state:Guard .
  ?sub state:in :Inactive
} state:transit { ?sub state:in :Ready } .
```

Using Linear Logic for State Transitions

- Reduce state-transition rules to linear logic rules in N3
  - After *active* => *completed*, task should no longer be in *inactive*
  - Linear logic [2] consumes premise after deriving conclusion
  - Possible to indicate “stable truth” that will not be consumed

Introspection and Rule Generation (2)

- N3 rule introspects and outputs new Linear Logic N3 rule

pattern-match with state-transition rule

```
{ ?premise state:transit ?conclusion ;
      log:includes { ?target state:in ?oldState . ?guard a state:Guard } .
  ?conclusion log:includes { ?target state:in ?newState } .
  ?premise state:reason ?reason .
  ( { ?guard a log:StableTruth } { ?target state:in ?oldState } )
     log:conjunction ?newPremise .
  ( ?conclusion
    { [ a state:Log ; state:target ?target ; state:reason ?reason ;
          state:from ?oldState ; state:to ?newState ; state:time ?now ] }
   ) log:conjunction ?newConclusion
} => { ?newPremise log:becomes ?newConclusion } .
```

construct new premise

construct new conclusion

Outline

N3 and the Web

Reasoning with N3

Applications

Can your use case benefit from N3?

Try it: http://ppr.cs.dal.ca:3002/n3/editor/

Join the W3C Community group: https://www.w3.org/community/n3-dev/