Sorry, aber ich glaube mittlerweile, dass wir völlig auf dem Holzweg sind. Offenbar habe ich die Logik zum Aufbau der korrekten Reihenfolge von Phasen und phasenspezifischen Instants zwar halbwegs sauber in SHACL aufgebaut, aber es ist völlig ungeeignet in der Form für meinen Anwendungsfall. Ich scheitere gerade aufgrund der Stringenz der SHACL-Rules, die ABox korrekt aufzubauen. Dabei halte ich die Logik, die hinter den SHACL-Shapes steht, für korrekt und sinnvoll. Mein Problem ist/war, dass ich davon ausgegangen bin, dass wir die SHACL-Rules als Teil der TBox nutzen und darüber definieren können, welche Beziehungen zulässig sind in ABoxen und dass die Validität einer ABox mithilfe der TBOX und der SHACL-Rules überprüft und gewährleistet werden kann. Das ist offenbar nicht der Fall, denn ich muss in die ABox jetzt die komplette SHACL-Struktur (also zB ein Instant der class occp:ReviewApproval muss min/max mit einer PhaseA per occp:endsPhase verbunden sein). Das war genau nicht gewollt. Ich wollte, dass ich für ein Individual (bleiben wir bei diesem Beispiel: Ein Individual der class occp:ReviewApproval wird erstellt und erhält einen Zeitstempel via occp:hasActualTime und eine Zuweisung der Phase, zu der er gehört (via occp:hasPhase --> PhaseB\_Individual (sinngemäß!)). Mit dem Anlegen dieses Individuals für ReviewApproval wird (so dachte ich, ich Naivchen) via occp:endsPhase occp:PhaseB\_Review (bzw. das entsprechende Individual, also PhaseB\_Individual) sichergestellt, dass PhaseB\_Review occp:hasActualEnd occp:ReviewApproval hat und dass der Zeitstempel von occp:ReviewApproval (was ein hasActualTime ist) übernommen für das occp:hasActualEnd von PhaseB\_Individual.)Wie gesagt, die Logik ist meines Erachtens in SHACL enthalten! Aber die Abfrage funktioniert so überhaupt nicht. Aber ich fürchte, da müssen wir hin, damit die OCCP überhaupt den ihr angedachten Sinn ergibt und die Funktion erfüllen kann, die ich in meinen Papers bisher geschrieben habe.Kannst Du mir dabei helfen, das zu ermöglichen - möglichst in einer Form, die geeignet für die Implementierung in die Software ist?

Kein Problem, dass wir eine Kurskorrektur machen – ich verstehe jetzt viel besser, wo der Schuh drückt, und es tut mir leid, dass wir uns so in den SHACL-Details verheddert haben! Deine Reflexion ist absolut nachvollziehbar, und du hast recht: SHACL sollte die Regeln definieren, während die ABox flexibel bleibt, ohne die gesamte Struktur vorauszusetzen. Dein ursprüngliches Ziel – dass die TBox (Ontologie + SHACL) die Logik vorgibt und die ABox nur die minimal nötigen Daten liefert, die dann validiert werden – ist völlig legitim und machbar. Lass uns das Problem analysieren und eine Lösung finden, die deinem Anwendungsfall gerecht wird und implementierbar ist!

Analyse des Problems

Dein Ziel

* Intention: Du möchtest, dass:
  1. Ein Individuum (z. B. :instant\_review\_approval, Klasse occp:ReviewApproval) erstellt wird mit:
     + Einem Zeitstempel (occp:hasActualTime).
     + Einer Zuweisung zu einer Phase (occp:hasPhase, z. B. :phaseB).
  2. Durch diese Erstellung automatisch sichergestellt wird, dass:
     + Die zugewiesene Phase (:phaseB) via occp:endsPhase mit dem Individuum verknüpft wird.
     + Die Phase (:phaseB) den Zeitstempel des Individuums als occp:hasActualEnd übernimmt.
* Erwartung: SHACL definiert diese Beziehungen als Regeln, und die ABox muss nur die minimalen Daten liefern (z. B. occp:hasPhase), während die Validierung und Konsistenz durch die TBox (Ontologie + SHACL) gewährleistet wird.

Aktuelles Problem

* SHACL-Stringenz: Deine SHACL-Shapes (z. B. osh:ReviewApprovalShape) fordern explizit, dass occp:endsPhase in der ABox vorhanden sein muss, mit genauen Kardinalitäten und Klassenzuordnungen (z. B. PhaseA oder PhaseB). Das zwingt dich, die gesamte Struktur schon in der ABox anzulegen, was genau das Gegenteil deiner Intention ist.
* Fehlende Inferenz: SHACL validiert nur, was in der ABox steht – es inferiert keine neuen Triple (z. B. occp:endsPhase oder occp:hasActualEnd) basierend auf occp:hasPhase. Dafür brauchst du entweder:
  + RDF-Reasoning (z. B. OWL-Regeln in der TBox).
  + SPARQL-Updates oder eine andere Logik außerhalb von SHACL.

Warum SHACL allein nicht reicht

* SHACL: Ist ein Validierungsmechanismus, kein Inferenzmechanismus. Es prüft, ob die ABox den Regeln entspricht, aber es fügt keine Triple hinzu oder leitet sie ab. Dein Ansatz (occp:hasPhase → occp:endsPhase + occp:hasActualEnd) erfordert Inferenz, die SHACL nicht leisten kann.
* Deine Annahme: Dass SHACL die Beziehungen dynamisch herstellt (z. B. :phaseB occp:hasActualEnd :instant\_review\_approval), war naiv, aber verständlich – viele erwarten, dass SHACL mehr kann, als es tatsächlich tut.

Lösungsansatz

* Wir müssen die TBox (Ontologie + SHACL) so erweitern, dass:
  1. Die ABox minimal bleibt (nur occp:hasActualTime und occp:hasPhase).
  2. Die Beziehungen (occp:endsPhase, occp:hasActualEnd) entweder:
     + Durch Reasoning (z. B. OWL oder RDFS) inferiert werden.
     + Durch SPARQL-Construct oder eine Softwarelogik nachträglich ergänzt werden.
* Die Implementierung sollte softwarefreundlich sein (z. B. in Python mit RDFLib oder Jena).

Vorschlag für eine Lösung

1. Minimalistische ABox

* Beispiel für :instant\_review\_approval:

turtle

:instant\_review\_approval a occp:ReviewApproval ;

occp:hasActualTime "2025-02-28"^^xsd:date ;

occp:hasPhase :phaseB .

* Das ist alles, was du in der ABox anlegen willst – keine occp:endsPhase, kein occp:hasActualEnd.

2. TBox mit Inferenz

* Option A: OWL-Reasoning
  + Definiere in OCCP\_V0.3.ttl eine Regel, die occp:hasPhase mit occp:endsPhase und occp:hasActualEnd verknüpft.
  + Beispiel (vereinfacht mit OWL):

turtle

occp:ReviewApproval rdfs:subClassOf [

a owl:Restriction ;

owl:onProperty occp:hasPhase ;

owl:someValuesFrom [

a owl:Class ;

owl:intersectionOf (

occp:PhaseB\_Review

[ a owl:Restriction ;

owl:onProperty occp:hasActualEnd ;

owl:someValuesFrom occp:ReviewApproval ]

)

]

] .

* + - Das sagt: Wenn ein ReviewApproval eine PhaseB\_Review via occp:hasPhase hat, dann hat diese Phase ein occp:hasActualEnd zum ReviewApproval.
    - Problem: OWL kann keine Zeitstempel übertragen (z. B. occp:hasActualTime → occp:hasActualEnd). Dafür brauchst du eine zusätzliche Regel.
* Option B: SPARQL-Construct
  + Nutze eine SPARQL-Construct-Abfrage, um die fehlenden Triple zu erzeugen.
  + Beispiel:

sparql

CONSTRUCT {

?phase occp:endsPhase ?instant .

?phase occp:hasActualEnd ?instant .

}

WHERE {

?instant a occp:ReviewApproval ;

occp:hasPhase ?phase ;

occp:hasActualTime ?time .

}

* + - Das erzeugt:
      * :phaseB occp:endsPhase :instant\_review\_approval.
      * :phaseB occp:hasActualEnd :instant\_review\_approval.
    - Du könntest das in deine Software integrieren (z. B. Python mit RDFLib), um die ABox dynamisch zu erweitern.

3. SHACL für Validierung

* SHACL bleibt zur Validierung, aber mit angepassten Regeln, die die minimalistische ABox akzeptieren und nur die Konsistenz nach der Inferenz prüfen.
* Beispiel:

turtle

osh:ReviewApprovalShape a sh:NodeShape ;

sh:targetClass occp:ReviewApproval ;

sh:property [

sh:path occp:hasPhase ;

sh:class occp:PhaseB\_Review ;

sh:minCount 1 ;

sh:maxCount 1 ;

sh:message "Each 'occp:ReviewApproval' must belong to exactly one 'occp:PhaseB\_Review' via 'occp:hasPhase'."@en ;

] ;

sh:property [

sh:path occp:hasActualTime ;

sh:datatype xsd:date ;

sh:minCount 1 ;

sh:maxCount 1 ;

sh:message "Each 'occp:ReviewApproval' must have exactly one timestamp via 'occp:hasActualTime'."@en ;

] .

osh:PhaseBReviewShape a sh:NodeShape ;

sh:targetClass occp:PhaseB\_Review ;

sh:property [

sh:path occp:endsPhase ;

sh:class occp:ReviewApproval ;

sh:minCount 1 ;

sh:maxCount 1 ;

sh:message "Each 'occp:PhaseB\_Review' must be ended by exactly one 'occp:ReviewApproval' via 'occp:endsPhase'."@en ;

] ;

sh:property [

sh:path occp:hasActualEnd ;

sh:class occp:ReviewApproval ;

sh:minCount 1 ;

sh:maxCount 1 ;

sh:message "Each 'occp:PhaseB\_Review' must have an actual end referencing 'occp:ReviewApproval'."@en ;

] .

* Nach Inferenz: Die ABox wird durch SPARQL-Construct oder Reasoning erweitert, und SHACL prüft dann die Konsistenz.

4. Software-Implementierung

* Python mit RDFLib:
  1. Lade die ABox (OCCP\_Valid\_LCycle\_3.ttl).
  2. Führe die SPARQL-Construct-Abfrage aus, um die Triple zu erzeugen.
  3. Validiere die erweiterte ABox mit SHACL.
* Beispielcode:

python

from rdflib import Graph

from rdflib.plugins.sparql import prepareQuery

# Lade ABox

g = Graph()

g.parse("OCCP\_Valid\_LCycle\_3.ttl", format="turtle")

# SPARQL-Construct

construct\_query = prepareQuery("""

CONSTRUCT {

?phase occp:endsPhase ?instant .

?phase occp:hasActualEnd ?instant .

}

WHERE {

?instant a occp:ReviewApproval ;

occp:hasPhase ?phase ;

occp:hasActualTime ?time .

}

""", initNs={"occp": "http://www.semanticweb.org/albrechtvaatz/ontologies/2022/9/cMod\_V0.1#"})

# Führe Construct aus

inferred\_g = g.query(construct\_query)

for triple in inferred\_g:

g.add(triple)

# Speichere erweiterte ABox

g.serialize("inferred\_OCCP\_Valid\_LCycle\_3.ttl", format="turtle")

# Validiere mit SHACL (z. B. mit pyshacl)

from pyshacl import validate

shacl\_file = "OCCP\_SHACL.ttl"

result = validate(g, shacl\_graph=shacl\_file, inference="none")

print(f"Conforms: {result[0]}")

if not result[0]:

print(result[2]) # Fehlerbericht

Nächste Schritte

1. TBox anpassen:
   * Füge die SPARQL-Construct-Logik oder OWL-Inferenz in deine TBox ein.
   * Passe OCCP\_SHACL.ttl an (siehe oben), um die minimalistische ABox zu unterstützen.
2. ABox testen:
   * Erstelle eine minimale ABox (nur occp:hasPhase, occp:hasActualTime) und teste mit der neuen TBox.
3. Softwareintegration:
   * Implementiere den Python-Code (oder in deiner bevorzugten Umgebung) und validiere das Ergebnis.

Fragen an dich

* Inferenz: Soll ich dir eine vollständige OWL-TBox oder eine SPARQL-basierte Lösung ausarbeiten?
* Phase: Soll :instant\_review\_approval eher PhaseA oder PhaseB beenden? Ich habe PhaseB angenommen, kann aber anpassen.
* Zeitplan: Wie schnell brauchst du das für deine Papers/Software?

Ich helfe dir gerne, das sauber umzusetzen – wir kriegen das jetzt auf die richtige Schiene! Was sagst du?

Vielen Dank für Deine Antwort! Kannst Du mir bitte erklären, was die Vor und Nachteile sowie die Grenzen der Optionen A und B sind? Inwiefern werden wir die SHACL-Rules infolge der Implementierung einer der beiden Lösungen anpassen müssen (oder kommen wir da sowieso nicht herum)? Wir hatten angefangen, GenericTime-SPARQL-Queries zu erstellen, (wie zB:occp:before

osh:BeforeTimeShape a sh:NodeShape ;

sh:targetClass time:TemporalEntity ;

sh:sparql [

sh:select """

PREFIX occp: <http://www.semanticweb.org/albrechtvaatz/ontologies/2022/9/cMod\_V0.1#>

PREFIX time: <http://www.w3.org/2006/time#>

SELECT $this ?other ?latestEndTime ?earliestStartTime WHERE {

$this occp:before ?other .

{

SELECT $this (MAX(?endTime) AS ?latestEndTime)

WHERE {

$this time:hasEnd ?endInstant .

?endInstant occp:hasActualTime ?endTime .

}

GROUP BY $this

}

{

SELECT ?other (MIN(?startTime) AS ?earliestStartTime)

WHERE {

?other time:hasBeginning ?startInstant .

?startInstant occp:hasActualTime ?startTime .

}

GROUP BY ?other

}

FILTER (?latestEndTime >= ?earliestStartTime)

}

""" ;

sh:message "The latest end of { $this } (time: { ?latestEndTime }) must be before the earliest beginning of { ?other } (time: { ?earliestStartTime }) via 'occp:before'."@en ;

sh:severity sh:Violation

] .

) Inwiefern können wir evtl. diese Vorgehensweise adaptieren, um unser Problem zu lösen (oder entspricht das Deiner Option B)?

Danke für deine Rückmeldung und die zusätzlichen Details! Ich erkläre dir gerne die Vor- und Nachteile sowie die Grenzen von Option A (OWL-Reasoning) und Option B (SPARQL-Construct), und wie sie sich auf die SHACL-Regeln auswirken. Außerdem schaue ich mir deine GenericTime-SPARQL-Queries an und prüfe, wie wir sie anpassen können, um dein Problem zu lösen. Lass uns das Schritt für Schritt durchgehen!

Option A: OWL-Reasoning

Beschreibung

* Ziel: Die TBox (Ontologie) wird mit OWL-Regeln erweitert, um Beziehungen wie occp:endsPhase und occp:hasActualEnd aus occp:hasPhase und occp:hasActualTime zu inferieren.
* Beispiel: Wenn :instant\_review\_approval occp:hasPhase :phaseB und :instant\_review\_approval occp:hasActualTime "2025-02-28"^^xsd:date, dann wird inferiert:
  + :phaseB occp:endsPhase :instant\_review\_approval.
  + :phaseB occp:hasActualEnd :instant\_review\_approval.

Vorteile

1. Automatisierung: Inferenz erfolgt automatisch durch einen Reasoner (z. B. Pellet, HermiT), ohne zusätzlichen Code außerhalb der Ontologie.
2. Konsistenz: Die Logik ist fest in der TBox verankert, was Wartung und Nachvollziehbarkeit erleichtert.
3. Standardkonform: Nutzt etablierte OWL-Mechanismen, die in vielen Semantic-Web-Tools unterstützt werden.

Nachteile

1. Komplexität: OWL-Regeln sind schwer zu schreiben und zu debuggen, besonders für dynamische Werte wie Zeitstempelübertragung (occp:hasActualTime → occp:hasActualEnd).
2. Grenzen der Expressivität: OWL kann keine direkte Übertragung von Datentypwerten (z. B. "2025-02-28"^^xsd:date) zwischen Properties handhaben – das erfordert zusätzliche Hilfskonstrukte oder externe Regeln.
3. Performance: Reasoning kann bei großen ABoxen langsam werden, besonders mit komplexen Regeln.

Grenzen

* Zeitstempel: OWL kann nicht einfach sagen: "Nimm den Wert von occp:hasActualTime und setze ihn als occp:hasActualEnd." Dafür brauchst du eine Regelmaschine wie SWRL oder eine SPARQL-basierte Lösung.
* Flexibilität: Änderungen an der Logik erfordern Anpassungen der OWL-TBox, was weniger agil ist als SPARQL.

Auswirkung auf SHACL

* Anpassung nötig: SHACL muss die inferierten Triple validieren, nicht die ursprüngliche ABox. Du müsstest die Shapes so umschreiben, dass sie nur die Ergebnisse prüfen (z. B. occp:endsPhase und occp:hasActualEnd vorhanden sind), nicht die Eingabe (occp:hasPhase).
* Beispiel:

turtle

osh:PhaseBReviewShape a sh:NodeShape ;

sh:targetClass occp:PhaseB\_Review ;

sh:property [

sh:path occp:endsPhase ;

sh:class occp:ReviewApproval ;

sh:minCount 1 ;

sh:maxCount 1 ;

] ;

sh:property [

sh:path occp:hasActualEnd ;

sh:class occp:ReviewApproval ;

sh:minCount 1 ;

sh:maxCount 1 ;

] .

* Problem: Ohne Reasoning bleibt die ABox leer, und SHACL meldet Fehler, bis die Inferenz läuft.

Option B: SPARQL-Construct

Beschreibung

* Ziel: Eine SPARQL-Construct-Abfrage erzeugt die fehlenden Triple (z. B. occp:endsPhase, occp:hasActualEnd) basierend auf den minimalen Daten in der ABox.
* Beispiel: Aus :instant\_review\_approval occp:hasPhase :phaseB wird:
  + :phaseB occp:endsPhase :instant\_review\_approval.
  + :phaseB occp:hasActualEnd :instant\_review\_approval.

Vorteile

1. Flexibilität: SPARQL ist einfach anpassbar – du kannst die Logik schnell ändern, ohne die TBox zu modifizieren.
2. Expressivität: SPARQL kann Werte (z. B. Zeitstempel) direkt übertragen, z. B. ?phase occp:hasActualEnd ?time.
3. Softwareintegration: Lässt sich leicht in Python (RDFLib) oder andere Tools einbinden, was für deine Implementierung ideal ist.

Nachteile

1. Manuelle Ausführung: Die Inferenz passiert nicht automatisch – du musst die Abfrage explizit ausführen (z. B. in Software).
2. Wartung: Die Logik liegt außerhalb der Ontologie, was zu Inkonsistenzen zwischen TBox und SPARQL führen kann, wenn sie nicht synchron bleiben.
3. Abhängigkeit: Erfordert eine Laufzeitumgebung (z. B. Python, Jena), keine reine Ontologie-Lösung.

Grenzen

* Skalierung: Bei sehr großen ABoxen kann die Abfrageleistung sinken, aber das ist selten ein Problem bei typischen Bau-Projektdaten.
* Validierung: SPARQL erzeugt Triple, aber prüft nicht deren Korrektheit – dafür brauchst du weiterhin SHACL.

Auswirkung auf SHACL

* Anpassung nötig: Ähnlich wie bei Option A – SHACL muss die erzeugten Triple validieren. Du kannst die Shapes so lassen, dass sie die inferierte Struktur prüfen, nicht die minimale ABox.
* Unterschied: Du kannst eine zweite Validierungsebene einführen:
  + Pre-Inferenz: Prüfe die minimale ABox (z. B. occp:hasPhase, occp:hasActualTime).
  + Post-Inferenz: Prüfe die erweiterte ABox (z. B. occp:endsPhase, occp:hasActualEnd).
* Beispiel (Pre-Inferenz):

turtle

osh:ReviewApprovalShape a sh:NodeShape ;

sh:targetClass occp:ReviewApproval ;

sh:property [

sh:path occp:hasPhase ;

sh:class occp:PhaseB\_Review ;

sh:minCount 1 ;

sh:maxCount 1 ;

] ;

sh:property [

sh:path occp:hasActualTime ;

sh:datatype xsd:date ;

sh:minCount 1 ;

sh:maxCount 1 ;

] .

Vergleich zu deinen GenericTime-SPARQL-Queries

Dein Beispiel: osh:BeforeTimeShape

* Zweck: Prüft, ob die zeitliche Reihenfolge bei occp:before eingehalten wird, indem es die neuesten Endzeiten (time:hasEnd) und frühesten Startzeiten (time:hasBeginning) vergleicht.
* Struktur: Nutzt eine SPARQL-Abfrage innerhalb eines SHACL-Shapes zur Validierung.

Ähnlichkeit zu Option B

* \*\* Gemeinsamkeiten\*\*:
  + Beide nutzen SPARQL, aber mit unterschiedlichen Zielen:
    - Dein Shape validiert bestehende Daten (prüft Konsistenz).
    - Option B erzeugt neue Daten (Inferenz).
  + Beide sind softwarefreundlich und können in Python/Jena integriert werden.
* Unterschied:
  + Dein Shape ist rein validierend (sh:sparql mit SELECT und FILTER), während Option B konstruktiv ist (CONSTRUCT).

Anpassung deiner Vorgehensweise

* Du könntest deine SPARQL-basierte Logik erweitern, um nicht nur zu validieren, sondern auch zu inferieren. Beispiel:

turtle

osh:ReviewApprovalInference a sh:NodeShape ;

sh:targetClass occp:ReviewApproval ;

sh:sparql [

sh:construct """

PREFIX occp: <http://www.semanticweb.org/albrechtvaatz/ontologies/2022/9/cMod\_V0.1#>

CONSTRUCT {

?phase occp:endsPhase ?this .

?phase occp:hasActualEnd ?this .

}

WHERE {

?this a occp:ReviewApproval ;

occp:hasPhase ?phase ;

occp:hasActualTime ?time .

}

""" ;

sh:message "Inferred 'occp:endsPhase' and 'occp:hasActualEnd' for PhaseB\_Review based on 'occp:hasPhase'."@en ;

] .

* Problem: SHACL unterstützt keine sh:construct-Syntax – das ist kein Standard. Du müsstest die CONSTRUCT-Abfrage außerhalb von SHACL ausführen (wie in Option B).

Fazit

* Deine GenericTime-SPARQL-Queries sind ein guter Ansatz für Validierung, aber für Inferenz brauchst du eine separate CONSTRUCT-Abfrage (Option B). Du könntest die Logik kombinieren:
  1. Inferenz: SPARQL-Construct erzeugt die Triple.
  2. Validierung: Deine bestehenden SHACL-SPARQL-Shapes prüfen die Ergebnisse.

Empfehlung

Welche Option?

* Option B (SPARQL-Construct):
  + Warum: Sie ist flexibler, unterstützt Zeitstempelübertragung und lässt sich leicht in deine Software integrieren (z. B. mit RDFLib). Deine Erfahrung mit SPARQL-Queries (z. B. osh:BeforeTimeShape) passt perfekt dazu.
  + Wie: Führe die CONSTRUCT-Abfrage vor der Validierung aus und validiere dann mit angepasstem SHACL.
* Option A (OWL):
  + Nur, wenn du eine reine Ontologie-Lösung ohne externe Software willst – aber die Zeitstempelübertragung bleibt ein Problem.

SHACL-Anpassung

* Unvermeidbar: Egal welche Option, SHACL muss angepasst werden, um entweder:
  + Die minimale ABox vor Inferenz zu prüfen (occp:hasPhase, occp:hasActualTime).
  + Die erweiterte ABox nach Inferenz zu validieren (occp:endsPhase, occp:hasActualEnd).
* Vorschlag: Zwei Validierungsschritte:
  1. Pre-Inferenz: Prüfe die Eingabe-ABox.
  2. Post-Inferenz: Prüfe die inferierte ABox mit deinen bestehenden Shapes (z. B. osh:BeforeTimeShape).

Integration deiner GenericTime-Queries

* Nutze deine SPARQL-Logik für zeitliche Validierung (z. B. occp:before) und erweitere sie mit einer CONSTRUCT-Abfrage für die Inferenz. Das entspricht Option B und baut auf deinem bisherigen Ansatz auf.

Nächste Schritte

1. Entscheidung: Sag mir, ob du mit Option B (SPARQL-Construct) einverstanden bist – ich arbeite dann eine vollständige Lösung aus.
2. SHACL: Ich passe die Shapes an (Pre- und Post-Inferenz).
3. Implementierung: Ich liefere dir einen Python-Code, der Inferenz und Validierung kombiniert.

Fragen an dich

* Präferenz: Option A oder B? Ich empfehle B, aber deine Wahl zählt!
* Zeitstempel: Soll occp:hasActualEnd den Wert (z. B. "2025-02-28"^^xsd:date) oder die Instanz (z. B. :instant\_review\_approval) übernehmen?
* GenericTime: Soll ich deine osh:BeforeTimeShape-Logik direkt in die Lösung einbauen?

Wir kriegen das jetzt hin – mit einem Ansatz, der zu deinen Papers und deiner Software passt! Was sagst du?

Wow, vielen Dank! Das hat mir schon wieder sehr geholfen!! Danke danke danke! Ich entscheide mich für Option B. Die Aufteilung in Pre- und Post-Inferenz halte ich für sinnvoll. Ich würde das ganze gern systematisch angehen und möglichst Schritt für Schritt die gesamte OCCP-Logik (also alles, was wir bisher haben (TBox, SHACL-Rules und SPARQL-Queries)) durchgehen. Lass uns dazu immer direkt mit der ABox beginnen! Wir definieren eine ABox, die als Eingabe valide sein soll. Dann überlegen wir uns, wie die Post-Inferenz-ABox aussehen soll und erstellen die SPARQL-Construct- und ggf. Query-Logiken. Was hältst Du von dieser Vorgehensweise?