GC-SROIQ(C): Expressive Constraint Modelling and grounded circumscription for SROIQ

Arjun Bhardwaj
IIT Mandi, India
bhardwaj.arjun.14@gmail.com

Sanjeetha
AIDB Lab, IIT Madras, India
geetha@cse.iitm.ac.in

Abstract
Developments in semantic web technologies have promoted ontological encoding of knowledge from diverse domains. However, modelling many practical domains requires more expressive representations schemes than what the standard description logics(DLs) support. We extend the DL SROIQ with constraint networks and grounded circumscription. Applications of constraint modelling include embedding ontologies with temporal or spatial information, while grounded circumscription allows defeasible inference and closed world reasoning. This paper overcomes restrictions on existing constraint modelling approaches by introducing expressive constructs. Grounded circumscription allows concept and role minimization and is decidable for DL.

We provide a general and intuitive algorithm for the framework of grounded circumscription that can be applied to a whole range of logics. We present the resulting logic: GC-SROIQ(C), and describe a tableau decision procedure for it.

1 Introduction
Unrestricted combination of temporal constructs and circumscriptive schemes with DLs can lead to undecidability. In this paper, we describe the extension of SROIQ with grounded circumscription and constraint networks. This approach allows the creation of ontologies with temporal constraint information and closed world reasoning.

Open world assumption of Semantic Web and the monotonic nature of the underlying Description logics(DLs) makes it difficult to model many practical domains. Earlier attempts to embed temporal information (Lutz, Wolter, and Zakharyaschev 2008) into DLs have only been realized for less expressive logics like ALC. The work by (Lutz and Milicic 2007) augments ALC with ω-admissible Constraint Systems to form a decidable DL. The ω-admissible systems identified were the Allen’s relations(for temporal intervals) and RCC8 relations(for spatial regions). Circumscription is a non-monotonic framework that allows closed world reasoning and default reasoning. However, minimization of roles is undecidable even for simpler DLs and tableau procedures exist only for concept minimization in logics with finite model property. The paper (Sengupta, Krisnadhi, and Hitzler 2011) uses grounded circumscription, using which both concept and role minimization in DLs is decidable. However, a tableau procedure exists only for ALC and involves special rules to ensure the construction of a grounded model.

In this paper we introduce the concept of constraint individual to allow expressive constraint modelling. We allow simple hierarchies in concrete roles, non functional concrete roles and more expressive constructs to represent constraint relations, which are no longer restricted to Path Normal Form(PNF). Similar to SROIQ, number restriction constructs are introduced for the concrete roles of GC-SROIQ(C). We modify the algorithm presented in (Sengupta, Krisnadhi, and Hitzler 2011) and present an iterative algorithm tableau which makes use of GCIs (TBox axioms) to ensure that the tableau (model) constructed is minimal and grounded.

The paper aims for smooth integration of ω-admissible constraint systems and grounded circumscription into SROIQ. The resulting logic is able to represent the following: Hearts normally occur on left side, a happy man is one whose parents as well as children are alive, in parties dessert is normally served at the end of the course, etc.

In this paper, we extend SROIQ with constraint networks to form SROIQ(C) and present a tableau decision procedure for common inference problems. We then describe a general iterative algorithm for grounded circumscription with SROIQ(C) to define a decision procedure for inference problems in GC-SROIQ(C). We provide sketches of proofs and examples in the supplementary materials.

2 The Logic SROIQ(C)
We describe SROIQ(C) : the extension of SROIQ (Horoeks, Kutz, and Sattler 2006) with constraint networks. In this section, we describe the constructs of the logic, their semantics and provide terminating tableau decision procedure for SROIQ(C)-KB satisfiability. We begin by defining constraint systems and the property they must satisfy to be used in SROIQ(C).

2.1 Constraint Systems
We use the notion of a constraint system as defined in (Lutz and Milicic 2007). Let Var be a countably infinite set of variables and Rel a finite set of relation symbols. These relations relate the elements of a concrete domain. A Rel-constraint is an expression of the form (v r v'), where v, v' ∈ Var, r ∈ Rel. A Rel-network is a finite set of Rel-constraints. For a Rel-network N, let V_N denote the variables used in the network. N is complete if for every v, v' ∈ V_N, there is exactly one constraint (v r v') ∈ N. Let M be the set of all complete networks, possible according to the semantics of the domain being modelled. A network N' is a model of a network N, if N' ∈ M and there is a mapping τ : V_N → V_{N'} such that (v r v') ∈ N implies (τ(v) r τ(v')) ∈ N'. A constraint system C is defined as ≡ ⟨Rel, M⟩. A network N is satisfiable in C if M contains a model of N.

Given any finite constraint networks N, M, their intersection is defined as follows:

\[ I_{N,M} := \{(v r v') \mid v, v' \in V_N \cap V_M, (v r v') \in N\} \]

C has the patchwork property if for finite and satisfiable networks M, N with complete I_{M,N} = I_{N,M}, N ∪ M is satisfiable. C has the compactness property if the following
holds: a network $N$ with infinite $V_N$ is satisfiable in $C$, if and only if, for every finite $V \subseteq V_N$, the network $N \mid V$ is satisfiable in $C$. We say that $C$ is $\omega$-admissible iff the following holds

1. Satisfiability in $C$ is decidable
2. $C$ has patchwork property
3. $C$ has the compactness property

To ensure decidability, only $\omega$-admissible constraint system are permitted to be used in $\text{SROIQ}(C)$.

### 2.2 Syntax

Let $N_C$ be the set of concept names, $N_{a,R}$ abstract roles, $N_{c,R}$ concrete roles, $N_{d,I}$ names of abstract individuals, $Nom$ the set of nominals and $N_{c,I} \subset Var$ names of constraint individuals. A $\text{SROIQ}(C)$ KB consists of a tuple $(A, \mathcal{T}, R)$ where $A, \mathcal{T}, R$ are respectively the ABox, TBox and RBox.

$\mathcal{R}$ consists of $R_h, R_{ch}, R_a, R_{ch}$ is a set of concrete RIA’s of the form $g \subseteq g'$ where $g, g' \in N_{c,R}; R_h, R_a$ and the notions of Inverse, Simple and Universal roles follow from $\text{SROIQ}$. However $\mathcal{R}_{ch}$ has extra assertions of the form $\text{Fxn}(S)$ and $\text{Fxn}(g)$ where $S$ is a simple role and $g$ a concrete role. A role chain is an expression of the form $R_1 \ldots R_n$ with $n \geq 1$ and each $R_i \in N_{c,R}$. A path is a sequence $R_1, \ldots, R_n$ consisting of simple roles $R_1, \ldots, R_n \in R_{ch}$ and a concrete role $g \in N_{c,R}$.

Unless mentioned otherwise, assume the following for $r, r' \in \text{Rel}(C)$: $g, g' \in N_{c,R}; R, R' \in N_{a,R} \cup N_{c,R}$; $S, S'$ are simple abstract roles; $C, D \in N_C; a, b \in Nom; i \in N_{d,I}, a, b \in N_{a,I}$; $U$ is a path; $G, G'$ are paths, role chains or $c$. The same applies to respective symbols with subscripts. The set of $\text{SROIQ}(C)$ concepts is defined recursively as follows:

$$C := A \mid \top \mid (C \cap D) \mid (C \sqcup D) \mid \neg C \mid \exists R.C \mid \forall R.C \mid \geq n.S.C \mid \leq n.S.C \mid \exists S:\text{Self} \mid \leq c.n.g \mid \geq c.n.g \mid \exists c.U_1.U_2.r \mid \exists c.U_1.r \mid \exists c.\{i\}.r \mid \forall c.U_1,U_2.r \mid \forall c.\{i\}.r \mid \forall c.\{i\}.r \mid$$

A $\text{SROIQ}(C)$ TBox contains GCI’s of the form $C \sqsubseteq D$, where $C, D$ are concepts.

A $\text{SROIQ}(C)$ ABox is extended to include assertions of the form $(i_1 \ r_2) \wedge g(a, i_1)$, where $a \in N_{a,I}, i_1, i_2 \in N_{d,I}, r \in \text{Rel} \cup \text{Rel}$ is the set of relations defined for the constraint system. For every $i \in N_I$ there must be some $g, a$ such that $ABox$ contains $g(a, i)$. We assume Unique Name Assumption(UNA) for constraint individuals, but not named individuals of abstract domain i.e $N_{a,I}$.

### 2.3 Semantics

An interpretation $\mathcal{I}$ is a tuple $(\Delta_X, \mathcal{I}, M_X)$, where $\Delta_X$ is the abstract domain, $\mathcal{I}$ is the interpretation function, and $M_X \in \mathfrak{M}$ is a complete constraint network of the constraint system $C$. The $\text{SROIQ}(C)$ constructs which are part of $\text{SROIQ}(C)$ have the usual interpretation using $(\Delta_X, \mathcal{I})$. We present here, the interpretations of the new constructs.

For every concrete role $g, g^2 \subseteq \Delta_X \times V_{M_X}$ The interpretation function is extended to concrete concepts as follows:

$$\exists U_1, U_2.r^2 := \{d \in \Delta^2 \mid \exists x_1 \in U_1^2(d) \text{ and } x_2 \in U_2^2(d) \text{ such that } (x_1 r x_2) \in M_I\},$$

$$\forall U_1, U_2.r^2 := \{d \in \Delta^2 \mid \forall x_1 \in U_1^2(d) \text{ and } x_2 \in U_2^2(d), \text{ we have } (x_1 r x_2) \in M_I\},$$

$$\exists c.U_1, \{i\}.r^2 := \{d \in \Delta^2 \mid \exists x_1 \in U_1^2(d) \text{ and } i \in N_{d,I} \text{ such that } (x_1 r x_2) \in M_I\},$$

$$\exists c\{i\}, U_2.r^2 := \{d \in \Delta^2 \mid \exists x_1 \in N_{d,I} \text{ and } x_2 \in U_2^2(d) \text{ such that } (i r x_2) \in M_I\},$$

$$\forall c\{i\}, U_2.r^2 := \{d \in \Delta^2 \mid \forall x_1 \in U_1^2(d) \text{ and for } i \in N_{d,I} \text{ we have } (x_1 r x_2) \in M_I\},$$

$$\exists c\{i\}, U_2.r^2 := \{d \in \Delta^2 \mid \forall x_1 \in U_1^2(d) \text{ and for } i \in N_{d,I} \text{ and all } x_1 \in U_1^2(d) \text{ we have } (i r x_1) \in M_I\},$$

$$\leq c.n(g)^2 := \{x \mid \# \{c(x, c) \in g^2 \text{ and } c \in V_{M_X} \} \leq n\},$$

$$\geq c.n(g)^2 := \{x \mid \# \{c(x, c) \in g^2 \text{ and } c \in V_{M_X} \} \geq n\}$$

The paths $U_i = R_i \ldots R_k g_i, i = 1, 2$ and $d \in \Delta_X, U_1^2(d)$ is defined as

$$\{x \in V_{M_X} \mid \exists e_1, \ldots, e_{k+1} \mid d = e_1, (e_i, e_{i+1}) \in R_1^2 \text{ for } 1 \leq i \leq k, \text{ and } g^2(e_{k+1}) = x\}$$

$\mathcal{I}$ satisfies (is a model of) the RIA $g_1 \sqsubseteq g_2$ iff $g_1^2 \subseteq g_2^2$. For the new Abox assertions:

$$\mathcal{I} \vdash g(a, i) \quad \text{if } a^2 \subseteq \Delta^2, i \in V_{M_X} \text{ and } (a^2, i) \in g^2$$

$$\mathcal{I} \vdash (i_1 \ r_2) \quad \text{if } i_1, i_2 \in V_{M_X}, r \in \text{Rel} \cup \text{Rel}$$

The constraint individuals provide a way to directly address the variables of the constraint network, which would be formed by the tableau algorithm. It allows us to explicitly name concrete nodes of the completion system.

A knowledge base $(\mathcal{T}, \mathcal{R}, A)$ is satisfiable(consistent) if there exists an interpretation which is a model for each of $\mathcal{T}, \mathcal{R}$ and $A$.

### 3 Tableau Algorithm for SROIQ(C)

The Tableau algorithm generates a completion system $S = (\mathcal{G}, N, Q)$, where $\mathcal{G} = (V_a^S, V_c^S, E_a^S, E_c^S, L, M_S, \neq)$ is a completion graph, $N$ is a finite constraint network with $V_N = V_c^S$ and $Q$ is the constraint template set. $V_a^S$ is the set of abstract nodes, $V_c^S$ is the set of concrete nodes, $M_S$ relates each of the concrete nodes of $S$ to a set of markers, $E_a^S$ is Set of abstract edges of the form $(a, b)$, where $a, b \in V_a^S$; $E_c^S$ is Set of concrete edges of the form $(a, x)$, where $a \in V_a^S, x \in V_c^S$.

The completion system is a finite graphical representation of a (possibly infinite) tableau or a model, with both nodes labelled with concepts, and edge labelled with roles, using the labelling function $L^S$.

#### 3.1 Preliminaries

Here we discuss the terminology required to introduce the tableau algorithm. Let $KB$ be a $\text{SROIQ}(C)$ knowledge base consisting of $(A, R, T)$. Common inference problems can be converted to KB satisfiability as shown in (Horrocks, Kutz, and Sattler 2006). The $\text{SROIQ}(C)$ tableau algorithm assumes that
the SROIQ Rₐ assertions and the universal role have been reduced. Further, ABox assertions except the newly proposed assertions have been internalized and the RIA’s have been compiled into Automatons for complex roles. The Rₐ assertions Fxml(S) is converted to T ⊨≤ 1S, while Fxml(g) is converted to T ⊨≤ 1g.

Further, all the concepts must be in NNF. In NNF, the negation appears only in front of “primary” concepts. The set of primary concepts consists of all atomic concepts of NC, ∃S. Self, (∃s, U₁, U₂,r₁), (∃s, U₁, {i}, r₁), (∃s, {i}, U₁,r₁). For the new constructs, we define : NNF(¬(≤ ng)) = (≥ (n + 1)g), NNF(¬(≤ {i}, r₁)) = (≤ ng), NNF(¬(≤ 0g)) = ⊥, NNF(¬(≤ {i}, U₁,r₁)) = ∪ r₁ ∈ Rel,r₁′ ≠ ∃U₁, U₂,r₁′ and NNF(¬(∀U₁, {i}, r₁)) = ∩ r₁ ∈ Rel,r₁′ ≠ ∃U₁, U₂,r₁′. The semantics of primary concepts are inforced by means of completion rules and special clash conditions, mentioned later.

If (x, y) ∈ Eₘ or Eₙ, then y is called a successor of x, and x is called a predecessor of y. Ancestor is the transitive closure of predecessor, and descendant is the transitive closure of successor. For x, y ∈ Vₘ, R ∈ NₐR, y is called an R-successor of a node x if, for some R′ ∈ NₐR with R′ ⊆ R, R′ ∈ Lₘ(x, y). Similarly, for x ∈ Vₙ, y ∈ Vₙ, R ∈ NₐR, y is called an g-successor of a node x if, for some g′ ∈ NₐR with g′ ⊆ g, g′ ∈ Lₙ(x, y). A node y is called a R-forward-neighbour of a node x, if y is a R-successor of x or if x is a Inv(R)-successor of y. For a path U = R₁...Rₙg, a node a ∈ Vₙ is called a U-successor or forward-neighbour of a node a ∈ Vₙ if there exist e₁,...,e_k+₁ ∈ Vₙ such that e₁ is the R₁ forward neighbour of a, e_k+₁ is the Rₙ forward neighbour of e_k, and e_k ∈ Rₙg. If c is Uₙ = R₁...Rₙg successor of a, then we define Path(a, c, U) = Edge(a, e₁) ∪ Edge(e_k, c). If x is the successor of y, then Edge(x, y) = R and Edge(y, x) = Rev(R), where R ∈ Eₙ((a, b)) and Rev is a marker to indicate direction of traversal of the edge.

Markers and Internal Constructs In order to ensure the semantics of the constructs new to SROIQ, the completion rules break these down into internal constructs of the form ∃ₜ, Uₜ,q or ∀ₜ, Uₜ,q, where q is a marker. These internal constructs are not available for modelling knowledge in KB. If ∃ₜ,q ∈ Lₘ(a) for a ∈ Vₙ, then the completion rules insert the marker q into Mₖ(a), the marker set of node a. A marker q is a tuple of the form (qₛ, (G₁, G₂, r, v, E)), where qₛ ∈ Q is a marker symbol and (G₁, G₂, r, v, E) constitutes the constraint information embedded into the marker. We define Symbol(qₛ) = qₛ, and Info(qₛ) = (G₁, G₂, r, v, E). Here, G₁, G₂ may be paths, simple role chains or v ∈ R or v = {s, e} with s being the “start” symbol and e being the “end” symbol. The start and end symbols indicate the direction of the constraint relation, if the constraint (c₁r₁c₂) is seen as an arc pointing from c₁ to c₂. Consider an internal construct I with q, described above, as its marker. Let E be the original construct which was decomposed by completion rules to form the internal construct I, amongst possibly others. For aₚ,q-a ∈ V₈, let I ∈ Lₘ(a), E ∈ Lₘ(aₚ,q). Let the path U = G₁G₂. Then, the concrete construct E must contain U as one of the paths. Further, U is split into G₁, G₂ such that a is the G₁ successor of aₚ, Info(qₛ) encodes the information about where the concrete node c₁ with q in its Mₘ is located, relative to a. If c₁ and c₂ participate in a constraint relation, the information about the nature of the relation, and the location of c₂ relative to a is also encoded.

The constraint template set Q is a set of expressions of the form (qₛ₁ r₁ qₛ₂) (the positive template) or ¬(qₛ₁ r₁ qₛ₂) (the negative template), with qₛ₁, qₛ₂ ∈ Mₘ and r₁ ∈ Rel. Let c₁, c₂ be any two concrete nodes with qₛ₁ ∈ Mₘ(c₁), qₛ₂ ∈ Mₘ(c₂). If (Symbol(qₛ₁), Symbol(qₛ₂)) ∈ Q, then the completion rules add a constraint (c₁r₁c₂) to N. In comparison, if ¬(Symbol(qₛ₁), Symbol(qₛ₂)) ∈ Q, then the presence of (c₁r₁c₂) ∈ N leads to a clash. The negative template is used with clash conditions and completion rules to ensure the semantics of concepts of the form ¬∃U₁, U₂,r or ¬∃U₁, {i}, r or ¬∃{i}, U₁,r.

Blocking For internal constructs I₁, I₂ with marker symbols qₛ₁, I₁, I₂, if they differ only in qₛ₁, qₛ₂, and if Info(qₛ₁) = Info(qₛ₂), for a, b ∈ Vₘ Lₘ(a) = Lₘ(b) if there exists an injective mapping π : Lₘ(a) → Lₘ(b) such that I ∈ Lₘ(a) iff I(π) ∈ Lₘ(b), and I = π(I) (if I is not an internal construct), I = π(I) (otherwise).

An non-nominal abstract node of S is called a blockable node. For a path/role chain G = R₁R₂...g, we define Inv(G) = Inv(g) ... Inv(R₂)Inv(R₁), a node a₂ is a strict descendant of a₁, if it is a descendant of a₁ and can be reached from a₁ without encountering a nominal node on the connecting path.

If blocking does not occur, the rules can be applied in a way to produce an infinite structure composed of repeating units. Let a, b, aₚ, bₚ ∈ Vₙ. The pair (b₂, b) potentially repeats the pair (aₚ, a) if aₚ, aₚ are the ancestors of b and all the following hold:
1. aₚ is a predecessor of a and b is a predecessor of b
2. aₚ, a, b, b₂, and all nodes on the path from a to b are blockable
3. Lₘ(a) = Lₘ(b) and Lₘ(aₚ) = Lₘ(bₚ)
4. Lₘ((aₚ, a)) = Lₘ((b₂, b))

However, this condition alone is not sufficient for blocking, and we perform further test to ensure that an infinitely repeating structure is possible. Having identified a potential repeating unit, if required, we attempt to create new units underneath the existing one (described by a, b) by selectively applying completion rules. We define the following terms:

\[ \text{intc}(a, b) = \{ c \mid c \in Vₙ; c \text{ is a strict descendant of } a \text{ but not of } b \} \]

\[ \text{cPaths}(a) = \{ U \mid \existsₜ U,a,q \in Lₘ(a); \forallₜ U,a,q \in Lₘ(a); \text{ or} \}
\]
\[ I \in Lₘ(a), \text{Info}(I) = (G₁, G₂, r, v, (∃ₜ, Vₙ), U₁, U₂,r),
\]
\[ U = \text{Inv}(G₁)U₁ (i f \ \text{v} = s), U = \text{Inv}(G₁)U₁ (i f \ \text{v} = e) \}

\[ \text{cNodes}(a, U) = \{ c \mid c \in Vₙ; c \text{ is } U\text{-successor of } a; c \text{ not a strict descendant of } a \text{ and the path connecting } a \text{ to } c \text{ doesn’t pass through nominal nodes} \}
\]
\[ \text{cNodes}(a, U) = \{ c \mid c \in \text{cNodes}(a, U), U \in \text{cPaths}(a) \}
\]

For the above mentioned a and b, we define:

\[ \text{extc}(a, b) = \{ c \mid c \in Vₙ; c \text{ has constraint with some } c' \in \text{intc}(a, b); c \notin \text{cNodes}(a) \cup \text{cNodes}(b) \} \]
intc(a,b) are the concrete nodes that “belong” to a repeating unit. Constraints can be formed between nodes of intc(a,b), and between these and (1) those nodes which would have been the same for all repeating units (extc(a,b)) (2) those which vary for each unit (cNodes(s), assuming the repeating unit is headed by s). These are defined below.

We attempt to create new nodes Vc and Vp (descendants of a) such that there exists an injective mapping \( \phi : V_a \cup V_c \rightarrow \text{intc}(a,b) \) such that for \( a, b \in V_a \) and \( c, c' \in V_c : L^S(a) = L^S(\phi(a)), L^S(b) = L^S(\phi(b)), L^S(\{a, c\}) = L^S(\{\phi(a), \phi(c)\}), M^S(\{c\}) = M^S(\{\phi(c)\}) \). One unit is said to have been formed this way. The mapping ensures that after stacking a new unit below the original unit defined by a, b, we now have a nodes \( b_{i+1}, b_{i} \in V_a \) such that \( b_{i+1}, b_{i} \) potentially repeats \( (b_{i+1}) \). We stack new units underneath existing ones repeatedly till we have a unit in which \( s_{n+1}, s_n \) potentially repeats \( (s_{n+1}) \). And all cNodes(s) are descendants of a. At this stage, bs is potentially blocked by bn-1, bn is (label) blocked by bn-1, if all the following hold:

- \( b_s \) is potentially blocked by \( b_{n-1} \)
- \( cPaths(b_{n-1}) = cPaths(b_n) \) and for all \( U \in cPaths(b_n) \) : \#(cNodes(b_{n-1}, U)) \#(cNodes(b_n, U))
- There exists an injective mapping \( f : cNodes(b_n) \rightarrow cNodes(b_{n-1}) \) such that for all \( U \in cPaths(b_n) \) and all \( c \in cNodes(b_n, U) : Path(b, c, U) = Path(a, f(c), U) \)
- constraint networks \( N_c = cNodes(a) \cup extc(a,b) \) and \( N_b = cNodes(b) \cup extc(a,b) \)
- are complete and isomorphic i.e. for \( c_1, c_2 \in cNodes(b) \) and \( c_3, c_4 \in extc(a,b) : (c_1 r c_2) \iff (f(c_1) r f(c_2)) \) and w.l.o.g. \( (c_3 r c_3) \iff (f(c_1) r c_3) \)

The notion of indirectly blocked is the same as in SROIQ. If however, the test fails and label blocking is not established, we remove all nodes below b and all corresponding edges and constraints are also removed.

Merging and pruning. The merging carried out here is similar to [Horrocks, Kutz, and Sattler 2006]. Additionally, the following steps must be carried out when merging abstract node y into x : Any concrete nodes \( c \notin N_c \) are simply removed from \( V_S^a \) as well as from \( N_c \). If \( y \in N_c \) and \( i \in N_c \), g-successor of \( y \), then we remove \( (y,i) \) from:

1. If \( (x,i) \in E^S_x \), add \( y \) to \( L^S((x,i)) \)
2. Else, create a new edge between \( x \) and \( i \)

The case of merging a concrete node c1 into another concrete node c2

1. For \( a \in V_a^S \) such that \( (a, c_1) \in E^c \) (incoming edge)
   (a) If \( (a, c_2) \in E^c \), then set \( L^S((a, c_2)) = L^S((a, c_1)) \)
   (b) Remove \( (z, y) \) from \( E^S_y \)
2. Set \( M^S(c_2) = L^S(c_1) \cup M^S(c_2) \)
3. Rename c1 to c2 in all constraints in \( N \).
4. Add \( c_2 \neq c' \) for all \( c' \) such that \( c' \neq c_1 \)

Pruning is done similar to SROIQ, but when an abstract node is pruned, the concrete nodes (\( \notin N_c \)) successors of the nodes are also pruned from \( S \).

Clash Conditions. Apart from the the clash conditions of SROIQ, the completion system is said to contain a clash in either of the following cases

1. There exists \( a \in V_S^s \), such that \( S \leq a, ng \in L^S(a) \), and there exist \( c_1, \ldots, c_k \in V_S^s \) such that \( k > n, c_i \neq c_j \) for \( 1 \leq i < j \leq k \) and each \( c_i \) is a g successors of a for \( 1 \leq i < k \)
2. There exist \( c_1, c_2 \in V_S^s \), such that \( (c_1, c_2) \in N \) even though \( \neg(Symbol(q_1), r Symbol(q_2)) \in \mathcal{Q} \) for some \( q_1 \in M^S(c_1), q_2 \in M^S(c_2) \).
3. \( N \) is not satisfiable

3.2 Algorithm Initialization

If \( o_1, \ldots, o_l \in \text{Nom} \), then the tableau algorithm starts with the completion graph \( G = (\{r_0, \ldots, r_l\}, \emptyset, L^S, \emptyset) \) and \( L^S(r_i) = \{a_i\} \) for \( 0 \leq i \leq l \), with \( o_0 \) being a new nominal. For every ABox assertion of the form \( g(p, i) \), where \( p \in N_a, i \in N_c, \{p, o\} \subseteq o \) and \( o \in L^S(r) \), do the following: add a concrete node i to \( V_c \), and add an edge \( (i, r) \) to \( E_c \) and set \( L^S((r, i)) = \{i\} \).

For all such concrete nodes \( i_1, \ldots, i_k \in N_c \cap V_c^s \), add \( i_j \neq i \) for \( 1 \leq j < l \leq k \). Further, for every assertion of the form \( (i_1 r_i), (i_2 r_j) \), add \( (i_1 r_j) \) to \( N \).

G is then expanded by repeatedly (and non-deterministically) applying the expansion(completion) rules. This stops if either a clash occurs or if no more rules are applicable, in which case \( S \) is said to be complete. The completion system \( S \), if complete and clash-free, can be unraveled to form an augmented tableau for the Knowledge Base \( \langle A, T, R \rangle \). The algorithm turns a complete and clash-free completion system, iff the KB is consistent.

3.3 The Completion Rules

Assume \( a, b \in V_a^c, c \in V_c^c, q \in Q \) and \( q' \) are new in \( S \).

- **R \exists U**: If \( \exists U, q, q_1, q_2 \in L^S(a) \) and \( a \) is not indirectly blocked, Then, if not already added, add \( \exists U, q_1 \in L^S(a) \) and \( \exists U, q_2 \in L^S(a) \) and \( (q_1, r q_2) \) to \( Q \) where \( q_1 = (q_1, (U_1, r, s, q, \exists U_1, U_2, r)) \), \( q_2 = (q_2, (U_2, r, e, s, \exists U_1, U_2, r)) \).

- **R \exists i**: If \( \exists U, i, \{i\} r \in L^S(a) \) and \( a \) is not indirectly blocked, Then, if not already present, add \( \exists U, q_1 \in L^S(a) \) and add \( (q_1, r q_2) \) to \( Q \) where \( q_1 = (q_1, (U_1, r, s, \exists U_1, U_2, r)) \), \( q_2 = (q_2, (U_2, r, e, s, \exists U_1, U_2, r)) \).

- **R \exists i**: If \( \neg(\exists U, i, \{i\} r \in L^S(a) \) and \( a \) is not indirectly blocked, Then, if not already present, add \( \forall U, q_1 \in L^S(a) \) and \( \forall U, q_2 \in L^S(a) \) and \( (q_1, r q_2) \) to \( Q \) where \( q_1 = (q_1, (U_1, r, s, \neg \exists U_1, U_2, r)) \), \( q_2 = (q_2, (U_2, r, e, s, \neg \exists U_1, U_2, r)) \).

- **R \forall U**: If \( \forall U, U_1, U_2, r \in L^S(a) \) is not indirectly blocked, Then, if not already present, add \( \forall U, q_1 \in L^S(a) \) and \( \forall U, q_2 \in L^S(a) \) and \( (q_1, r q_2) \) to \( Q \) where \( q_1 = (q_1, (U_1, r, s, \forall U_1, U_2, r)) \), \( q_2 = (q_2, (U_2, r, e, s, \forall U_1, U_2, r)) \).

- **R \forall i**: If \( \forall U, \{i\} r \in L^S(a) \) is not indirectly blocked, Then, if not already present, add \( \forall U, q_1 \in L^S(a) \) and \( \forall U, q_2 \in L^S(a) \) and \( (q_1, r q_2) \) to \( Q \) where \( q_1 = (q_1, (U_1, r, s, \forall U_1, \{i\} r)) \), \( q_2 = (q_2, (U_2, r, e, s, \forall U_1, \{i\} r)) \).
$q_2 = (q'_2, (e, r, e, \{∀_e U_1, \{i\}, r \}))$. Analogously for $∀_{C}(i_1), U_1.r$.

- $R \exists_{\text{int}}$: If $∃_{U} q \in L^S(a), q = (q_a, (G_1, U_1, r, v), E)$, $U_1 = RU_2$, $a$ is not blocked, and there are no $R$ forward neighbours of $a$ with $∃_{U_2} q' \in L^S(b)$ such that $q' = (q_a, (G_1 R, U_2, r, v), E)$; then create new abstract node $b$ with $L^S(b) = \{∃_{U_2} q'\}$ and $L^{S}(a, b) = \{R\}$.

- $R \forall_{\text{int}}$: If $∀_{U} q \in L^S(a), q = (q_a, (G_1, U_1, r, v), E)$, $a$ is not indirectly blocked, $U_1 = RU_2$ and there is some $R$ forward neighbour $b$ of $a$, with $∀_{U_2} q' \notin L^S(b)$ where $q' = (q_a, (G_1 R, U_2, r, v), E)$; then set $L^S(b) = L^S(b) \cup \{∀_{U_2} q'\}$.

- $R \exists_{g}$: If $∃_{g} q \in L^S(a), q = (q_a, (G, g, r, v), E)$ such that $a$ is not blocked, and there is no $g$ successor of $a$, $c$ with $q_c \in M^S(c)$; then, create new concrete node $c$ with $M^S(c) = \{(q_a, (G, g, r, v), E))\}$ and $L^S(c) = \{g\}$.

- $R \forall_{g}$: If $∀_{g} q \in L^S(a), q = (q_a, (G, g, r, v), E)$ such that $a$ is not blocked, and there is some $g$ successor of $a$, $c$ with $q_c \in M^S(c)$; then add $(q_a, (G, g, r, v), E))$ to $M^S(c)$.

- $R \exists_{c}$: If $∃_{c} q \in \mathcal{Q}$ and there are nodes $c_1, c_2$ such that $q_1 \in M^S(c_1), q_2 \in M^S(c_2)$. Symbol$(q_1) = q'_1$ and Symbol$(q_2) = q'_2$. Then add $(c_1 \triangleright c_2)$ to $\mathcal{N}$.

- $R \forall_{c}$: If $∀_{c} q \in L^S(a)$ and there are not $n$ successors of $a, c_1, \ldots, c_n$ such that $c_i \neq c_j$ for $1 \leq i < j \leq n$. Then create $n$ new concrete nodes $c_1, \ldots, c_n$ with $L^S(a, c_i) = \{g\}$ and $c_i \neq c_j$ for $1 \leq i < j \leq n$.

- $R \exists_{c}$: If $∃_{c} q \in L^S(a)$, $a$ is not indirectly blocked, and there exist more than $n$ successors of $a$, and there are two $g$-successors $c_1, c_2$ of $a$ without $c_i \neq c_2$; then, if $c_1 \in N_{c_1}$, then Merge$(c_2, c_1)$, else if $c_2 \in N_{c_1}$, then Merge$(c_1, c_2)$.

- $\text{Complete}$: If $a_{n+1}$ is potentially blocked by $a_1 \ldots a_n$, guess a completion for $c_{a_1,a_{n+1}}$, and $i_n \triangleright c_{a_1,a_{n+1}}, 1 \leq i \leq n$.

**Proposition 1.** The KB satisfiability of a SROIQ(C) KB is sound, terminating and complete.

**Algorithm 1:** InitTab Computes the Grounded Model if it exists

| Input: | KB $K$ with circumscriptive pattern $(M_c, V)$ i.e. $(K, M_c, V)$ |
| Output: | Grounded model $GM$ if it exists |
| 1 | For $C \in M_c$, assert $C \subseteq \text{Nom}$ |
| 2 | For $R \in M_c$, assert $\exists R, \top \subseteq \text{Nom}$ |
| 3 | For $R \in M_c$, assert $\exists R, \top \subseteq \text{Nom}$ |
| 4 | Based on the inference task, activate the appropriate clash checks |
| 5 | Run the SROIQ(C) tableau algorithm |
| 6 | if Clashes occur then |
| 7 | return |
| 8 | else |
| 9 | return Grounded Model $GM$ |

**Proposition 2.** If InitTab produces a complete and clash-free completion system, then the resulting completion system is a grounded model for the GC-SROIQ(C) KB.

**MinTab:** The algorithm is executed on success of InitTab. Let this grounded model be $GM$.

The grounded model $GM$ is given as input to minTab. minTab extracts the extensions of minimized predicates from the completion system. Let the concept extensions be $C_i^{ext}$ for $1 \leq i \leq n$. Let the role extensions be $R_j^{ext}$ for $1 \leq j \leq m$. From the extension of the role $R_j$, we extract $R_j^{ext,dom}$: the domain for the extension, $R_j^{ext,range}$: the range for the extension. For every $p \in R_j^{ext,dom}$, we define $R_j^{ext,range,p} = \{p' \mid (p, p') \in R_j^{ext,dom}\}$. Next perform the following for the original KB $(K, M_c, V)$:

1. Assert $C_i \subseteq C_i^{ext}$ for concepts
2. Assert $\exists R_j \top \subseteq R_j^{ext,range}$ for range of the role $R_j$
3. Assert $\exists R_j \top \subseteq R_j^{ext,range}$ for domain of the role $R_j$
4. Assert for all $p \in R_j^{ext,dom}$: $R_j \{p\} \subseteq R_j^{ext,range,p}$
5. Based on the inference task, have the appropriate clash conditions.
6. Activate the preference clash check.
7. Run the SROIQ(C) tableau algorithm

Let \( C^E \) be the extension of a concept \( C \in M \) extracted from a (possibly incomplete) completion system which is currently being operated upon by completion rules. Similarly, we extract \( R^E \). A preference clash is said to occur if for all \( P \in M \), \( P^{ext} = P^E \). It may be noted that for both algorithms, the added clashes are in addition to the ones internal to the tableau algorithms. If \( \text{minTab} \) produces a complete and clash-free tableau, then this completion system is fed as an input to \( \text{minTab} \) again. This continues until no more models can be obtained. This model is the grounded circumscription model.

**Proposition 3.** If \( \text{MinTab} \) produces a complete and clash-free completion system, then the resulting completion system represents a model preferred over the input model w.r.t. the circumscription pattern.

### 5 Related Work

Previous attempts at extending DLs with concrete domains include the tableau for \( \mathcal{ALC} \). Unrestricted concrete domain addition to DLs quickly leads to undecidability \cite{Lutz2002}. Constraint systems are a subset of concrete domain. The paper by \cite{LutzMilicic2007} provided a decidable tableau algorithm for \( \mathcal{ALC} \) extended with constraint systems by using \( \omega \)-admissible systems \( t\text{-}\text{OWL} \). \cite{MileaFrasincarKaymak2012} motivates the need to embed ontologies with temporal information. It provides constructs using the \( SHIN(D) \) DL. Circumscription is a well-studied technique \cite{Lifschitz1996}. An analysis of the complexity of circumscription in DLs can be found in \cite{BonattiLutzWolter2014}. A tableau procedure is provided for concept minimization in \( \mathcal{ALCO} \) \cite{BonattiLutzWolter2014}. However, circumscription has been studied only for the DLs with finite model property. Work by \cite{GrimmHitzler2012} allows preferential firing of default rules. The grounded closed world reasoning technique introduced in \cite{SenguptaKrisnadhiHitzler2011} is successful in capturing the essence of circumscription. For grounded circumscription, both concept and role minimization is decidable, as long as the underlying language is decidable. It has applications for defeasible inference and modelling exceptions to concept subsumption rules and has a general approach to introduce closed world reasoning to DLs. Though the work done in \cite{SenguptaKrisnadhiHitzler2011} can be potentially adapted to higher DLs like SROIQ, a tableau decision procedure exists only for \( \mathcal{ALC} \). Further the tableau proposed involved special rules to ensure that the tableau constructed a grounded model.

### 6 Illustrative Example

#### Lucky and Unlucky Grandfathers

We present a sample \( \mathcal{GC-SROIQ}(C) \) KB. An element in the concept \textit{LuckyGrandFather} is a person who is alive at the same time as one of his grandchildren. An element in the concept \textit{UnluckyGrandFather} is a person who is dead before any of his grandchildren were born.

- \( hC \subseteq hP^- \), \( \top \subseteq_c 1hLTime \)
- \( GF \equiv \exists hC.(\exists hC, \top), GF \equiv LGF \sqcup UGF \)
- \( RGF \equiv \exists_j(hLTime), (hC hC hLTime) \cdot overlaps \)
- \( UGF \equiv \forall_j(hLTime), (hC hC hLTime) \cdot before \)
- \( hLTime(john, i_j), hC(john, steve), hC(steve, bob) \)

Here \( hC \) is \textit{hasChild}, \( hP \) is \textit{hasParent}, \( hLTime \) is \textit{hasLifeTime}. The abstract roles are \( hC \) and \( hP \). The concrete role \( hLTime \) is functional. Here \( GF \) is \textit{GrandFather}, \( LGF \) is \textit{LuckyGrandFather}, \( UGF \) is \textit{UnluckyGrandFather}. We minimize the predicates \( UGF \) and \( hC \).

The query \( \exists (hP hLTime). (hC hLTime) \cdot overlaps \) returns true. Due to minimization of \( hC \), \( steve \) is the only child of \( john \), likewise for \( bob \) and \( steve \). The minimization of \( UGF \) puts \( john \) in \( LGF \). If we add the assertions \( hLTime(bob, i_b) \) and \( i_j \) before \( i_b \), the same query returns false.

### 7 Conclusion and Future Work

We have presented a decidable inference procedure for the expressive DL \( \mathcal{GC-SROIQ}(C) \). It has expressive constraint modelling features and can be used to perform closed world reasoning. Many tasks still remain, including finding the complexity of the presented logic, optimizing the tableau decision procedure, extending circumscription to prioritized circumscription, amongst others. Discovering measures to counter the non-determinism associated with the inference procedure would ease the practical applications of the logic.

### References

- Bonatti, Lutz, and Wolter 2014: Bonatti, P. A.; Lutz, C.; and Wolter, F. 2014. The complexity of circumscription in dls. CoRR abs/1401.3476.
- Grimm and Hitzler: Grimm, S., and Hitzler, P. Defeasible inference with circumscribed owl.
- Horrocks, Kutz, and Sattler 2006: Horrocks, I.; Kutz, O.; and Sattler, U. 2006. The even more irresistible SROIQ. In In KR, 57–67. AAAI Press.
- Lifschitz 1996: Lifschitz, V. 1996. Circumscription.
- Lutz and Milicic 2007: Lutz, C., and Milicic, M. 2007. A tableau algorithm for description logics with concrete domains and general tboxes. J. Autom. Reasoning 38(1-3):227–259.
- Lutz, Wolter, and Zakaryaschev 2008: Lutz, C.; Wolter, F.; and Zakaryaschev, M. 2008. Temporal description logics: A survey. In 15th International Symposium on Temporal Representation and Reasoning, TIME 2008, Université du Québec à Montréal, Canada, 16-18 June 2008, 3–14.
- Lutz 2002: Lutz, C. 2002. Description logics with concrete domains-a survey. In Advances in Modal Logic, 265–296.
- Milea, Frasincar, and Kaymak 2012: Milea, V.; Frasincar, F.; Kaymak, U. 2012. towl: A temporal web ontology language. IEEE Transactions on Systems, Man, and Cybernetics, Part B 42(1):268–281.
- Sengupta, Krisnadhi, and Hitzler 2011: Sengupta, A. A.; and Hitzler, P. 2011. Local closed world semantics: Grounded circumscription for owl. In The Semantic Web - ISWC 2011 - 10th International Semantic Web Conference, Bonn, Germany, October 23-27, 2011, Proceedings, Part I, volume 7031 of Lecture Notes in Computer Science, 617–632. Springer.
8 Appendix: Knowledge Modelling Examples

Docks and Ships  Sep1, Sep2 ... Sep30 all be constraint individuals. As intervals, they represent the days of September month. The ABox assertion, helps enforce the semantics: (Sep1 meets Sep2) ... (Sep29 meets Sep30). Ship ⊑ ∃c Ship are elements of the abstract domain.

The concrete roles and the relation between them is given by: dayOfArrival ⊑ dayAtDock, dayOfMaintainence ⊑ dayAtDock, dayOfArrival ⊑ dayAtDock. We can declare these to be functional i.e. Ship ⊑ c 1.dayOfArrival, Ship ⊑ c 1.dayOfDeparture, T ⊑ ≤ 1.dayOfMaintainence. Note that we have not made the role dayOfDockUsage functional, but we can code the knowledge that every ship uses the dock for at least a day using: Ship ⊑ ∃c 1.dayOfDockUse. Further, we encode the information that these days occur in a particular order i.e. T ⊑ (∀c dayOfArrival, dayOfMaintainence.meets) ∪ (∀c dayOfArrival, dayOfMaintainence.equals), similarly, we order day of departure to be same as day of maintainece, or come meet the day of maintainece. We can ensure that the day of Dock usage all align with the intervals(Sep1 ... Sep30) we have declared, by the following restriction: Ships ⊑ (∀c dayOfArrival, Sep1.equals)∪ ... (∀c dayOfArrival, Sep30.equals). Similarly for dayOfMaintainence and dayOfDeparture. The functional nature of these concrete roles allowed us to enforce the alignments.

We can have default rules like: normally ships come before Sep10. This is ensured by minimizing the abnormality predicate Ab1 in Ship ⊑ (∀c dayOfArrival, {Sep10}.before) ∪ Ab1.

Normally Ships stay at the dock for more than 2 days : Ship ⊑ (≥c 2.dayAtDock) ⊑ AB2, where Ab2 is the predicate to be minimized.

Interesting queries can be made to the KB:
To check if a particular ship X came to the dock after a fixed date : \{X\} ⊑ ∀c dayOfArrival, {Sep25}. after

Lucky and Unlucky Grandfathers  We present a sample GC-SROIQ(C) KB. An element in the concept LuckyGrandFather is a person who is alive at the same time as one of his grandchildren. An element in the concept UnluckyGrandFather is a person who is dead before any of his grandchildren were born.

\begin{align*}
hC & \subseteq hP^−, T \subseteq ≤ 1hLTime \\
GF & \equiv ∃hC.(∃hC.T), GF \equiv LGF \sqcup UGF \\
LGF & \subseteq ∃c(hLTime), (hC hC hLTime).overlaps \\
UGF & \subseteq ∀c(hLTime), (hC hC hLTime).before \\
hLTime(john, i_j), hC(john, steve), hC(steve, bob)
\end{align*}

Here hC is hasChild, hP is hasParent, hLTime is hasLifeTime. The abstract roles are hC and hP. The concrete role hLTime is functional. Here GF is GrandFather, LGF is LuckyGrandFather, UGF is UnluckyGrandFather. We minimize the predicates UGF and hC.

The query ∃(hP hLTime), (hC hLTime).overlaps returns true. Due to minimization of hC, steve is the only child of john, likewise for bob and steve. The minimization of UGF puts john in LGF. If we add the assertions hLTime(bob, i_b) and i_b before i_b, the same query returns false.

Professors and tenures  Professor A had exactly 3 tenures, at least one of which was before the tenure of his student B. \{A\} ⊑ Professor ∩ (∃c timeOfTenure, hasStudent timeOfTenure.before) ∩ (≤c 3timeOfTenure) ∩ (≥c 3timeOfTenure)

Old Battleship  This representation is possible because of the fact that we do not require the concrete domain concepts to follow PNF. Suppose we wish to convey that battleship X is so old that its missiles were made before its crew was born. \{X\} ⊑ ∀c hasMissile.timeOfManufacture, hasCrewMember lifeTime.before
9 Appendix : Sample Completion Systems

9.1 Example 1

Drawing Conventions:
The names Abstract nodes are highlighted in yellow.
The names Concrete nodes are highlighted in orange.

dotted line indicates a constraint relation between concrete nodes
dark lines are abstract role edges.
light lines represent concrete role edges.
a long curved line joining nodes drawn in triangles indicates a blocking relation

a star represents a concrete node.
a elaborate star represents a constraint individual.
a dot represent an abstract node.
a dot in a circle represents a nominal node.
a filled triangle represents the blocker node.
an empty triangle represents the blocked node.

label sets of nodes are illustrated using curly brackets alongside the nodes.

Elaboration on diagram For clarity, the internal constructs are not shown in the diagram. $N_{aR} = \{P, Q, R, T\}$, the set of abstract roles. $N_{cR} = \{g, g', g''\}$, the set of concrete roles. $\{e_1, e_2, e_3, e_4\} \subseteq \text{Rel}$, the set of constraint relations. $\text{Nom} = \{o_1, o_2\}$. $N_I = \{N_1, N_2\}$, such that $o_1 = \{N_1\}$, $o_2 = \{N_2\}$

completion graph $G = (V^S_a, V^S_c, L^S, E^S_a, E^S_c, \notin)$, where :

$V^S_a = \{r_3, a_p, a, a_1, a_2, b_p, b, r_1, a_3\}$

$V^S_c = \{c_1, c_3, c_4, c_5, c_6, c_7, c_8, i_1\}$

$\mathcal{N} = \{(c_1 e_3 c_3), (c_8 e_1 c_7), (c_4 e_4 c_6), (c_5 e c_6), (c_3 e_2 i_1)\}$
The KB contains the following axioms: $o_2 \sqsubseteq \exists R. G$

$$o_1 \sqsubseteq \exists g. Rg.r \sqsubseteq (\exists Q.C \cap (\geq 1.g) \cap (\forall e.g, QPQg'.e)$$

$$C \sqsubseteq \exists P'(\exists Q.D)$$

$$D \sqsubseteq (\exists g. Rg'.e_1) \cap (\forall R.E) \cap (\exists g', \{i_1\} \cdot e_2)$$

$$E \sqsubseteq (\exists S.o_1) \cap (\forall e.g, TRg'.e_4) \sqsubseteq G$$

The ABox has these axioms: $(N_2)$

### 9.2 Example 2

The KB for the completion graph shown in the figure is:

$$(\exists HLg'.c_1(e, HLg', r, s), \exists g.g(e, g, r, e)$$

The contents of the constraint template set at this time are:

$$N_{aR} = \{H, L, S, G\}, N_{cR} = \{h, g', g''\}, \{r, r', r''\} \in \text{Rel}.$$ The example shows the internal constructs also in the label sets. The KB for the completion graph shown in the figure is:

$$o_1 \sqsubseteq \exists HLg'.g.r$$

$$\exists L. T \sqsubseteq (\exists g, Sg.r) \cap (\forall e.g, Sg.r')$$

$$\exists S. T \sqsubseteq (\exists GHLg'.g''', r''', e)$$

$o_1$ is a nominal set and $x_1$ is the first node created during initialization of the completion graph.