

Using Standard Description Logic Reasoning for SAT Planning: Exploring Role-Based Encodings of the Wolf–Goat–Cabbage Riddle

Michael Wessel

Claude (Anthropic)

March 2026

Abstract

SAT planning reduces planning problems to satisfiability tests: a plan of length n exists if and only if the conjunction of start state, goal state, and action axioms is satisfiable. The classical propositional encoding suffers from *symbol proliferation*—every combination of object and location requires its own propositional variable. We investigate whether Description Logic (DL) roles can yield a more compact encoding, using the well-known Wolf–Goat–Cabbage river crossing puzzle as a case study. Starting from an existing encoding with atomic concept names in KRSS syntax for the RACER reasoner, we systematically explore several role-based reformulations, documenting the trade-offs between symbol economy, reasoning performance, and semantic correctness. Our journey reveals fundamental tensions between compactness and tableau efficiency in DL-based SAT planning.

1 Introduction

The Wolf–Goat–Cabbage (WGC) riddle is a classical river crossing puzzle: a ferryman must transport a wolf, a goat, and a cabbage across a river. The boat holds at most the ferryman plus one object. The wolf and goat cannot be left alone (the wolf eats the goat), nor can the goat and cabbage (the goat eats the cabbage).

Following the SAT planning paradigm [1], the problem can be encoded as a satisfiability test. States are connected by a functional *next* relation; axioms constrain which successor states are legal. A plan of length n exists iff the conjunction of axioms, start state, and goal state is satisfiable for a chain of n states.

Wessel [2] presented an encoding using the RACER Description Logic reasoner, employing atomic concept names in KRSS syntax. While effective, this encoding uses over 20 atomic concept names for object–location combinations (e.g., `goat-a`, `wolf-boat-from-a-to-b`), exhibiting the symbol proliferation typical of propositional SAT encodings.

In this paper, we explore whether DL *roles* can provide a more compact representation, and document the challenges encountered along the way.

2 The Original Encoding

2.1 Logical Signature

Each object (goat, wolf, cabbage, ferryman, boat) can be in one of four locations: riverbank A , riverbank B , in the boat going from A to B , or in the boat going from B to A . This yields four atomic concept names per object, for example:

`goat-a`, `goat-b`, `goat-boat-from-a-to-b`, `goat-boat-from-b-to-a`

The role *next* is declared as a *feature* (functional role), connecting each state to its unique successor.

2.2 Successor State Axioms

The boat cycles deterministically through the four locations:

$$\text{boat-a} \sqsubseteq \forall \text{next}. \text{boat-from-a-to-b}$$

$$\text{boat-from-a-to-b} \sqsubseteq \forall \text{next}. \text{boat-b}$$

and so on. In RACER KRSS syntax:

```
(implies boat-a (all next boat-from-a-to-b))
(implies boat-from-a-to-b (all next boat-b))
(implies boat-b (all next boat-from-b-to-a))
(implies boat-from-b-to-a (all next boat-a))
```

2.3 No-Teleportation Axioms

Objects can only move by boarding the boat. From a riverbank, an object either stays or boards:

$$\text{goat-a} \sqsubseteq \forall \text{next}. \text{goat-a} \sqcup \forall \text{next}. \text{goat-boat-from-a-to-b}$$

From transit, it arrives at the opposite bank:

$$\text{goat-boat-from-a-to-b} \sqsubseteq \forall \text{next}. \text{goat-b}$$

The same pattern is repeated for wolf, cabbage, and ferryman (16 axioms total).

2.4 Boat Capacity, Safety, and Integrity

Capacity: When the boat travels from A to B , the ferryman is aboard, exactly one of {wolf, goat, cabbage} is aboard, and no two passengers ride together. When returning from B to A , the ferryman may return alone.

Safety: Wolf and goat, or cabbage and goat, on the same riverbank require the ferryman's presence:

$$\text{wolf-a} \sqcap \text{goat-a} \sqsubseteq \text{ferryman-a}$$

Disjointness: Each object is at exactly one location:

```
(disjoint goat-a goat-b goat-boat-from-a-to-b goat-boat-from-b-to-a)
```

Persistence: Objects do not disappear—every state specifies each object's location:

$$\text{goat} \sqsubseteq \text{goat-a} \sqcup \text{goat-b} \sqcup \text{goat-boat-from-a-to-b} \sqcup \text{goat-boat-from-b-to-a}$$

Coupling: An object on the boat implies the boat is in transit:

$$\text{goat-boat-from-a-to-b} \sqsubseteq \text{goat} \sqcap \text{boat-from-a-to-b}$$

2.5 States, Start, and Goal

Every state either is a goal or has a successor, and specifies all objects:

$$\text{state} \sqsubseteq (\text{goal} \sqcup \exists \text{next}. \text{state}) \sqcap \text{goat} \sqcap \text{wolf} \sqcap \text{cabbage} \sqcap \text{ferryman} \sqcap \text{boat}$$

$$\text{start} \sqsubseteq \text{state} \sqcap \text{boat-a} \sqcap \text{goat-a} \sqcap \text{wolf-a} \sqcap \text{cabbage-a} \sqcap \text{ferryman-a}$$

$$\text{goal} \sqsubseteq \text{state} \sqcap \text{boat-b} \sqcap \text{goat-b} \sqcap \text{wolf-b} \sqcap \text{cabbage-b} \sqcap \text{ferryman-b}$$

2.6 Performance Characteristics

This encoding uses exclusively *atomic concept names* on the left-hand side of GCIs (General Concept Inclusions), enabling effective *absorption* by the tableau reasoner. Absorbed GCIs are handled as lazy unfolding rules rather than global constraints checked against every individual—a significant performance advantage.

The encoding works well in RACER: ABox consistency checking completes quickly, and individual type retrieval yields the expected two solutions to the puzzle (corresponding to transporting the cabbage or the wolf after the initial goat crossing).

The price is symbol proliferation: 5 objects $\times$ 4 locations = 20 atomic concept names, plus 5 generic object concepts and supporting names.

3 Attempt 1: Object-Centric Location Roles

3.1 Idea

Replace the 20 atomic object–location concepts with 5 *functional roles* (features) mapping states to locations, plus 4 shared location concepts:

$$\begin{aligned} \text{goat-a} &\rightsquigarrow \exists \text{goat-loc. a} \\ \text{goat-b} &\rightsquigarrow \exists \text{goat-loc. b} \\ \text{goat-boat-from-a-to-b} &\rightsquigarrow \exists \text{goat-loc. from-a-to-b} \end{aligned}$$

The primitive vocabulary shrinks from 20+ symbols to 9 (5 roles + 4 location concepts). Disjointness of locations combined with functional roles gives per-object location uniqueness for free:

```
(define-primitive-role goat-loc :feature t)
(disjoint a b from-a-to-b from-b-to-a)
```

3.2 Axiom Rewriting

All axioms are rewritten using role expressions. For example, the boat cycle:

$$\exists \text{boat-loc. a} \sqsubseteq \forall \text{next. } \exists \text{boat-loc. from-a-to-b}$$

Coupling becomes:

$$\exists \text{goat-loc. from-a-to-b} \sqsubseteq \exists \text{boat-loc. from-a-to-b}$$

3.3 Performance Problem

This encoding was too slow for practical use. The root cause: every axiom with $\exists r. C$ on the left-hand side is a *General Concept Inclusion* (GCI) that cannot be absorbed into the concept hierarchy. The original encoding had atomic concept names on the left—cheaply absorbed. The role-based encoding has ~ 30 non-absorbable GCIs, each requiring global checking against every individual in the tableau.

Additionally, the `:feature` declarations on all 5 location roles imposed expensive merge operations whenever the tableau created multiple fillers for the same role.

4 Attempt 2: Defined Concept Abbreviations

4.1 Idea

Introduce named concepts as abbreviations for the role expressions:

```
(define-concept goat-at-a (some goat-loc a))
(define-concept goat-at-b (some goat-loc b))
;; ... 20 definitions total
```

Then use these names in axioms: (`implies goat-at-a (all next goat-at-a) ...`). This gives the reasoner atomic-like concept names for internal optimization while preserving the role-based semantics.

4.2 Problem

This reintroduces symbol proliferation. We now have 20 defined concept names—the same count as the original’s 20 primitive concepts. While they are *defined from* the 9-symbol role vocabulary rather than independently primitive, the practical effect is the same: 20 names that the user must manage. The compactness goal is defeated.

5 Attempt 3: Dropping Features

5.1 Idea

Remove the `:feature` declarations from the location roles to eliminate merge operations. Use (`at-most 1 role`) in the state axiom instead—scoped to state individuals rather than globally declared:

```
(define-primitive-role goat-loc) ;; NOT a feature

(implies state (and ...
  (at-most 1 goat-loc)
  (at-most 1 wolf-loc)
  ...))
```

5.2 Problem

Still too slow. The (`at-most 1`) restriction is semantically equivalent to a feature constraint—the reasoner must still track cardinality and perform merge operations. The GCI bottleneck ($\exists r. C$ on axiom left-hand sides) remains unchanged. The **at-most** restrictions were subsequently removed entirely, relying on the axiom structure to naturally produce at most one filler per role (the tableau is lazy and only creates fillers when forced by existential restrictions).

6 Attempt 4: Location-Centric Roles

6.1 Idea

Invert the role perspective: instead of objects tracking their locations (*goat-loc*, *wolf-loc*, ...), locations track which objects are at them. Use 4 roles (*at-a*, *at-b*, *at-ab*, *at-ba*) and 5 object concepts (*goat*, *wolf*, *cabbage*, *ferryman*, *boat*):

$$\begin{aligned} \text{goat-a} &\rightsquigarrow \exists \text{at-a. goat} \\ \text{wolf-boat-from-a-to-b} &\rightsquigarrow \exists \text{at-ab. wolf} \end{aligned}$$

The symbol count is unchanged (9), but axioms that reference a single location now use a single role throughout. For example, the safety constraint:

$$\exists \text{at-a. wolf} \sqcap \exists \text{at-a. goat} \sqsubseteq \exists \text{at-a. ferryman}$$

All three expressions reference the same role *at-a*, which may benefit the reasoner’s internal indexing.

6.2 Open-World Semantics and (all next ...)

A subtle but important issue arose with the state axiom. The original formulation used $\exists next.state$ to ensure non-goal states have a successor:

```
(implies state (and (or goal (some next state)) ...))
```

Under open-domain semantics, $\exists next.state$ does *not* necessarily constrain the explicitly modeled successor (linked via `(related s1 s2 next)`). The reasoner may create a fresh anonymous individual to satisfy the existential, leaving the asserted successor unconstrained.

The fix: use $\forall next.\cdot$ to push object-persistence constraints to the modeled successor. Since *next* is a feature, $\forall next.C$ applies to the unique successor. Additionally, declaring `:domain state :range state` on the *next* role ensures that all individuals connected by *next* are classified as states:

```
(define-primitive-role next :feature t :domain state :range state)

(implies state (and (or goal (some next top))
                    (all next (or (some at-a goat) ...))
                    ...))
```

6.3 The Uniqueness Problem

Without uniqueness enforcement, the encoding is unsound.

The location-centric approach cannot use features on the *at* – *X* roles because multiple objects legitimately share a location (e.g., wolf, cabbage, and ferryman can all be at riverbank *A*).

Without uniqueness constraints, the safety axioms become toothless. Consider state *s*₂ where the boat departs from *A* with the ferryman. If the goat stays at *A* alongside the wolf, safety requires $\exists at-a.ferryman$. Without uniqueness, the reasoner simply creates an *additional at-a* filler of type *ferryman*—even though the ferryman is already on the boat ($\exists at-ab.ferryman$). The ferryman appears at two locations simultaneously, trivially satisfying the safety constraint.

In the original encoding, this was prevented by:

```
(disjoint ferryman-a ferryman-b
         ferryman-boat-from-a-to-b ferryman-boat-from-b-to-a)
```

6.4 Solution: Per-Object Location Exclusion Axioms

Since per-object uniqueness cannot be achieved via features or `disjoint` on atomic names (we don’t have them), we add explicit exclusion axioms. For each object and each location, being at that location excludes all others:

$$\exists at-a.goat \sqsubseteq \neg \exists at-b.goat \sqcap \neg \exists at-ab.goat \sqcap \neg \exists at-ba.goat$$

In RACER syntax:

```
(implies (some at-a goat)
  (and (not (some at-b goat))
        (not (some at-ab goat))
        (not (some at-ba goat)))))
```

This requires 5 objects $\times$ 4 locations = 20 exclusion axioms—effectively re-encoding the original’s disjointness declarations as GCIs.

6.5 Readable Labels

Since the role-based encoding produces types like $\exists at-a. \text{goat}$ rather than atomic names, ABox queries would not yield human-readable labels. We add 20 labeling axioms that tag states with familiar names:

```
(implies (some at-a goat) goat-a)
(implies (some at-ab wolf) wolf-boat-from-a-to-b)
;; ... etc.
```

These are cheap GCIs: the right-hand side is an atomic concept name (no fillers created, just tagging).

6.6 Final Encoding

The final, working encoding uses:

- 4 location roles: *at-a*, *at-b*, *at-ab*, *at-ba* (non-features)
- 5 object concepts: **goat**, **wolf**, **cabbage**, **ferryman**, **boat** (pairwise disjoint)
- 1 functional role: *next* with domain/range **state**
- 20 per-object exclusion axioms for uniqueness
- 20 labeling axioms for readable output
- Standard axioms: boat cycle (4), capacity (2), no-teleportation (16), coupling (8), safety (4), state/start/goal (3)

The encoding produces correct results and runs at reasonable speed in RACER.

7 Discussion

7.1 Symbol Count Comparison

Table 1 summarizes the primitive vocabulary of each approach.

Approach	Roles	Concepts	Total Primitives	GCIs
Original (atomic)	1	20+	21+	0 (all absorbed)
Object-centric roles	6	4	10	~30
Defined abbreviations	6	24	30	~30
Location-centric (final)	5	5	10	~50

Table 1: Comparison of encoding approaches. GCI count refers to non-absorbable axioms (complex concept on LHS).

7.2 The Compactness–Performance Trade-off

Our experiments reveal a fundamental tension:

1. **Atomic concepts absorb; role expressions do not.** Axioms of the form $A \sqsubseteq C$ where A is atomic are absorbed into the concept hierarchy and handled as lazy unfolding. Axioms of the form $\exists r. C \sqsubseteq D$ are GCIs checked globally against every tableau individual—far more expensive.
2. **Features are expensive but necessary.** Functional role declarations enable automatic per-object uniqueness but impose merge operations. Dropping features requires explicit exclusion axioms (20 additional GCIs) to maintain soundness.
3. **Open-world semantics require care.** The use of $\exists next. state$ in the state axiom can lead to the creation of anonymous successor individuals rather than constraining the explicitly

asserted ones. Using $\forall next.$ with domain/range declarations ensures constraints apply to the modeled plan states.

4. **Location-centric roles cannot use features for uniqueness.** When multiple objects share a location role, features would incorrectly limit the role to a single filler. Per-object uniqueness must be encoded separately.

7.3 Lessons Learned

The original encoding, despite its symbol proliferation, is well-matched to tableau-based DL reasoning. Atomic concept names enable absorption, **disjoint** declarations are handled efficiently, and no GCIs are needed.

The role-based encodings achieve greater *ontological economy* (fewer primitive symbols), but at the cost of GCIs that the tableau must check globally. The location-centric variant adds locality benefits (same-role references within safety and capacity axioms) and produces correct results at reasonable speed, but requires 20 exclusion axioms and 20 labeling axioms—totaling 40 additional axioms—that partially offset the compactness gains.

The exercise demonstrates that in DL-based SAT planning, the choice of representation is not merely aesthetic: it has direct consequences for reasoning performance. The “right” encoding depends on the reasoner’s optimization strategies, particularly GCI absorption and feature handling.

8 Conclusion

We explored several role-based reformulations of a classical SAT planning problem in Description Logics, seeking to reduce symbol proliferation while maintaining reasoning efficiency. Our journey through object-centric roles, defined abbreviations, feature elimination, and location-centric roles revealed that:

- Compact role-based encodings are possible (9 primitive symbols vs. 20+), but produce non-absorbable GCIs that slow tableau reasoning.
- Uniqueness of object locations—trivially handled by **disjoint** in the atomic encoding—requires either features (expensive), cardinality restrictions (equally expensive), or explicit exclusion axioms (verbose) in role-based encodings.
- Open-world semantics demand careful use of $\forall next.$ rather than $\exists next.$ to ensure constraints apply to explicitly modeled successor states.
- The final location-centric encoding with exclusion axioms provides a working, reasonably fast alternative that demonstrates the use of DL roles for structuring planning state descriptions.

The original atomic encoding remains the pragmatic choice for performance-critical applications. However, the role-based approach offers conceptual clarity and extensibility—for instance, adding new objects or locations requires only new concept names, not a combinatorial explosion of new atomic symbols.

References

- [1] H. Kautz and B. Selman, “Planning as Satisfiability,” in *Proc. ECAI*, 1992, pp. 359–363.
- [2] M. Wessel, “SAT Planning in Description Logics: Solving the Classical Wolf Goat Cabbage Riddle,” 2014. Unpublished presentation.
- [3] V. Haarslev, K. Hidde, R. Möller, and M. Wessel, “The RACERPRO Knowledge Representation and Reasoning System,” *Semantic Web – Interoperability, Usability, Applicability*, vol. 3,

no. 3, pp. 267–277, IOS Press, 2012. <https://www.semantic-web-journal.net/content/racerpro-knowledge-representation-and-reasoning-system>