**Project 1: OWL Cheat Sheet Documentation**

**OWL 2 Web Ontology Language Overview**

**The OWL 2 Web Ontology Language is a highly expressive ontology language for the Semantic Web. It facilitates the definition and sharing of complex ontologies over the web. Building on OWL 1, it introduces enhancements that make it backward compatible while offering new features such as property chains, profiles for specific use cases (OWL 2 EL, QL, RL), and improved datatype expressiveness.**

**Part 1: OWL 2 Direct Semantics Role Constraints**

1. **Objective:**
   * **The goal of Part 1 is to explain why combinations of role constraints result in unsatisfiability or inconsistency in OWL ontologies.**
   * **This involves testing specific combinations of properties (Functional, Inverse Functional, Symmetric, Asymmetric, Transitive, Reflexive, Irreflexive) to understand which combinations lead to logical contradictions or failures in OWL reasoning engines like HermiT.**
2. **Explanation of Role Combinations and Test Results:**
   * **Functional + Transitive: The conflict arises from the fact that Functional requires unique assignments while Transitive allows cascading relationships. HermiT flags this combination as X^NS, leading to undecidability or potential reasoning failure.**
   * **Transitive + Irreflexive: The issue occurs because Transitivity can imply indirect self-relations (e.g., if xRy and yRz, then xRz), while Irreflexive explicitly forbids any self-relations. This conflict results in unsatisfiability, flagged as X^UNSAT by the reasoner.**
   * **Symmetric + Asymmetric: This leads to a contradiction, as Symmetry implies if xRy then yRx, while Asymmetry forbids such relationships. This test resulted in unsatisfiability and flagged X^UNSAT.**
3. **Heuristics Emerged from Table:**
   * **Transitivity combines best with Symmetry or Reflexivity.**
   * **Asymmetry only combines usefully with Functional, Inverse Functional, and Irreflexivity.**
   * **Reflexivity should not be combined with Asymmetry.**
   * **Irreflexivity should not be combined with Transitivity.**

**Part 2: OWL SubPropertyOf Testing and Results**

1. **Objective:**
   * **The focus of Part 2 is exploring combinations of role constraints spread across parent-child object property relationships (using owl:subPropertyOf).**
   * **The challenge is understanding how these combinations lead to unsatisfiability or undecidability and creating tables that show results from the reasoner based on these constraints.**
2. **Tested Object Properties and Characteristics:**
   * **A: Irreflexive (Inverse of Ai)**
   * **B: Asymmetric (SubProperty of A, Inverse of Bi)**
   * **Ai: Asymmetric (Inverse of A)**
   * **Bi: No characteristics (SubProperty of Ai, Inverse of B)**
   * **hasParent: Irreflexive**
   * **hasChild: None (SubProperty of hasParent)**
   * **hasRelation: Functional & Inverse Functional**
   * **isSiblingOf: Symmetric & Irreflexive (Inverse of isSiblingOf)**
   * **R2: Transitive**
   * **R3: Symmetric & Asymmetric (causing errors)**
   * **R4: Transitive & Symmetric (Inverse of R4)**
   * **R5: Transitive & Reflexive**
   * **R6: Reflexive**
   * **R7: Asymmetric & Irreflexive**
   * **R8: Symmetric & Irreflexive (Inverse of R8)**
   * **R9: Transitive**
3. **Documenting X^UNSAT and X^NS Testing Results:**
   * **Symmetric + Asymmetric (R3):**
     + **Test Results: Unsatisfiable as expected. The reasoner flagged this combination as conflicting due to Symmetric requiring mutual relationships (if xRy, then yRx) and Asymmetric prohibiting this.**
   * **Reflexive + Asymmetric (B/R6):**
     + **Test Results: Also resulted in unsatisfiability as Reflexive requires xRx, but Asymmetric prevents yRx. The reasoner flagged it appropriately.**
   * **Symmetric + Irreflexive (A or R8):**
     + **Test Results: This combination worked without errors, which aligns with expected behavior. However, in a few cases, such combinations can result in logical contradictions depending on how property chains are defined. The reasoner processed the combination successfully.**