NLPSolver

Natural Language to Formal Logic via LLMs

Semantic Parsing → First-Order Logic → Theorem Proving

https://github.com/tammet/nlpsolver/tree/main/llmpipe

The Problem

Given: natural language text + a question

Goal: derive the correct answer using logical reasoning

Challenge: bridge the gap between ambiguous English and precise logic

Example: "Elephants are animals. John is an elephant. Is John an animal?" → True

Requires: coreference resolution, quantifier scoping, default reasoning,
world state tracking, measurement comparison, ...

Pipeline Overview

English text

→ Stage 1 (LLM): English → Atomic Semantic Units (ASU JSON)
→ Stage 2 (LLM): ASU JSON → First-Order Logic (JSON)
→ logconvert: FOL → CNF clause list (GK format)
→ GK prover: clause list + axioms → proof / answer
→ Post-processing: proof → English answer

LLMs: GPT-5.1, Claude Sonnet 4.6, Gemini 2.0 Flash, DeepSeek V3.2
All produce the same Stage 1 / Stage 2 JSON format

Why Two LLM Stages?

Stage 1: linguistic analysis

- Coreference resolution, entity identification, tense detection
- World state tracking (W0, W1, W2, ...)
- Action decomposition (verb root, roles, mode)

Stage 2: logical encoding

- Predicates, quantifiers, implications
- Event reification (Davidsonian events)
- Defeasible reasoning markers (normally, typically)

Separation of concerns: Stage 1 handles language, Stage 2 handles logic

The GK Theorem Prover

First-order logic prover with confidence reasoning, based on the classical FOL high-performance reasoner GKC.

JSON-LD-LOGIC input format (JSON-based clause lists)

Key features:

- Survives contradictions in logic
- Confidence propagation through proof chains: both and (decreasing) and or (increasing)
- Defeasible reasoning via $block mechanism
- Built-in equality, arithmetic ($less, $sum, ...)
- Unique name assumption for integers and #:-prefixed distinct symbols
- Multiple proof strategies: negative_pref, unit, query_focus, etc

The UDP Pipeline (Reference)

Inital old alternative pipeline using dependency parsing (not LLMs)

UDPipe for syntactic analysis → rule-based logic conversion

Same GK prover backend

Similar, but less detailed / thought through logical encoding than the LLM pipeline logical encoding

Used as reference/comparison for LLM pipeline accuracy

Stage 1 — Atomic Semantic Units

Input: raw English text

Output: LLM giving JSON with sentence packages, each containing ASU units

LLM input: prompt constructed from a large instruction file + a large example file + actual English text to analyze

Each ASU has:

- unit_id, text (simplified), type (situation/real/strict_rule/normal_rule/query)
- entities: concrete or generic, with category (person/animal/place/measure/...)
- actions: verb root, mode (event/habitual/capability), roles
- Optional: time, time_prep, state_tense, pre_state/next_state, confidence, definites

Stage 1 — Example

Input: "Elephants are animals. John is an elephant. Is John an animal?"


```
[{"raw": "Elephants are animals.",
  "units": [{"unit_id": "S1", "text": "Elephants are animals.",
             "type": "strict_rule",
             "entities": [{"id": "elephants", "type": "generic", "category": "animal"},
                          {"id": "animals", "type": "generic", "category": "animal"}]}]},
 {"raw": "John is an elephant.",
  "units": [{"unit_id": "S2", "text": "John 1 is an elephant.",
             "type": "real",
             "entities": [{"id": "John 1", "type": "concrete", "category": "animal"}]}]},
 {"raw": "Is John an animal?",
  "units": [{"unit_id": "S3", "text": "Is John 1 an animal?",
             "type": "query",
             "entities": [{"id": "John 1", "type": "concrete"},
                          {"id": "animal", "type": "generic"}]}]}]
```

Stage 2 — First-Order Logic Predicates

Core: isa, has_property, is_rel2, have, has_part, can

Gradable: has_degree_property, has_degree_rel2

Event reification:

isa(activity,E), has_type(E,VERB), has_actor(E,X), has_target(E,Y),
has_destination(E,Y), has_recipient(E,Y), has_source(E,Y),
has_location(E,Y,PREP), has_time(E,T,PREP), has_instrument(E,Y), ...

Measurements: =($measure_of(attr,entity,world), $measure(number,unit))

Structure: forall, exists, implies, …, normally, confidence numbers

Worlds: holds(W,F), next(W1,W2),...

Questions: question(BODY) or ask(VAR, BODY)

Stage 2 — Example

Input: prompt constructed from a large instruction file + a large example file + stage 1 from previous slide

```
["and",
 ["@id", "S1",
   ["holds", "W0",
     ["forall", "X",
       ["implies", ["isa","elephant","X"], ["isa","animal","X"]]]]],
 ["@id", "S2",
   ["holds", "W0", ["isa","elephant","John 1"]]],
 ["@id", "S3",
   ["question", ["isa","animal","John 1"]]]]
```

Clausification + several conversions (logconvert)

Converts FOL to CNF (Conjunctive Normal Form) for the prover

Key transformations (but there are more):

- Skolemization: existentials → Skolem constants/functions (sk0_activity)
- $ctxt injection: [$ctxt, tense, world, location, knower] on every predicate
- Population facts: synthetic witnesses for rule variables ($some_elephant)
- Bridge axioms: $theof1 definite functions, $measure_of measurements
- Question encoding: $defq biconditional for wh-questions
- Meta-predicate normalization, tense stripping, entity ID normalization

Clausified Output — Simple Example

"Elephants are animals. John is an elephant. Is John an animal?"

```
Elephants are animals.
  isa(elephant,X) => isa(animal,X)

John is an elephant.
  isa(elephant,john)

Is John an animal?
  $defq0 => isa(animal,john)
  isa(animal,john) => $defq0
  question: $defq0

[generated: population facts]
  -isa(elephant,$some_not_elephant)
  isa(animal,$some_animal)
```

The $ctxt Context Term

Every stative/event predicate gets a context argument:

```
pred(args..., $ctxt(tense, world, location, knower))
```

- Tense: present, past, future (relative to world state)
- World: W0, W1, W2, ... (narrative progression)
- Location, Knower: optional scoping parameters

Enables: temporal reasoning, world state tracking

Example: has_type(sk0, travel, $ctxt(past, W0, ?:Fv1, ?:Fv2))

GK Prover — What It Does

Input: clause list (CNF) + background axioms (axioms_std.js)

Boolean questions: proof by contradiction

- Assume negation → derive contradiction → True
- Cannot derive contradiction → check if negation provable → False
- Can derive both positive and negative with some confidences C1 and C2: subtract confidences

Wh-questions:

- find values satisfying $defq biconditional → $ans(value)
- can derive both positive and negative for values found, with some confidences C1 and C2: subtract confidences

Confidence propagation: rules with confidence < 1.0 produce qualified answers

Defeasible reasoning: $block mechanism for default exceptions

Auto strategy selection: negation preferred, query preferred, unit limits etc etc

Proof — "Is John an animal?" → True

English rendering:

```
(1) if X is an elephant then X is an animal     [sentence 1]
(2) John is an elephant                          [sentence 2]
(3) John is an animal                            [from 1, 2]
(4) if John is an animal then answer holds       [sentence 3]
(5) assume for contradiction: the answer is no   [from question]
(6) Contradiction                                [from 3, 4, 5]
```

Answer: True.

Wh-Question — "Who is an animal?"

English rendering:

```
(1) if X is an elephant then X is an animal     [sentence 1]
(2) John is an elephant                          [sentence 2]
(3) John is an animal                            [from 1, 2]
(4) if X is an animal then X is an answer        [from question]
(5) John is an answer                            [from 3, 4]
```

Answer: John.

Wh-Question — Stage 2 + Clauses (JSON)

"Elephants are animals. John is an elephant. Who is an animal?"

```
Stage 2:
["and",
 ["@id","S1",["holds","W0",["forall","X",["implies",["isa","elephant","X"],["isa","animal","X"]]]]],
 ["@id","S2",["holds","W0",["isa","elephant","John 1"]]],
 ["@id","S3",["ask","X",["isa","animal","X"]]]]

Clausified:
  ["-isa","elephant","?:X"], ["isa","animal","?:X"]]          // rule
  ["isa","elephant","john"]                                    // fact
  {"@question": ["isa","animal","?:X"]}                       // query
```

Wh-Question — Proof with JSON

```
(1) if X is an elephant then X is an animal        [sentence 1]
    [["-isa","elephant","?:X"], ["isa","animal","?:X"]]

(2) John is an elephant                            [sentence 2]
    ["isa","elephant","john"]

(3) John is an animal                              [from 1, 2]
    ["isa","animal","john"]

(4) if X is an animal then X is an answer          [from question]
    [["-isa","animal","?:X"], ["$ans","?:X"]]

(5) John is an answer                              [from 3, 4]
    ["$ans","john"]
```

Answer: John.

Defeasible Reasoning — Penguins

"Birds normally fly. Penguins are birds. Penguins do not fly.
Tweety is a penguin. Robin is a bird. Who can fly?"

Stage 1 output (key parts):

S1: "Birds can fly" — type: normal_rule, confidence: 0.95

S2: "Penguins are birds" — type: strict_rule

S3: "Penguins cannot fly" — type: strict_rule

S4: "Tweety is a penguin" — type: real

S5: "Robin is a bird" — type: real

S6: "Who can fly?" — type: query

Penguins — Stage 2 JSON

```
["and",
 ["@id","S1",
   ["and",
     ["holds","W0",
       ["forall","X",["implies",["isa","bird","X"],["normally",["can","X","fly"]]]]],
     ["@p","S1",0.95]]],
 ["@id","S2",
   ["holds","W0",["forall","X",["implies",["isa","penguin","X"],["isa","bird","X"]]]]],
 ["@id","S3",
   ["holds","W0",["forall","X",["implies",["isa","penguin","X"],["not",["can","X","fly"]]]]]],
 ["@id","S4",["holds","W0",["isa","penguin","Tweety 1"]]],
 ["@id","S5",["holds","W0",["isa","bird","Robin 2"]]],
 ["@id","S6",["ask","X",["can","X","fly"]]]]
```

Penguins — Clausified Logic

```
Birds can fly.
  isa(bird,X) => can(X,fly)  [block(-can(X,fly))]  @0.9

Penguins are birds.
  isa(penguin,X) => isa(bird,X)

Penguins cannot fly.
  isa(penguin,X) => -can(X,fly)

Tweety is a penguin.    isa(penguin,tweety)
Robin is a bird.        isa(bird,robin)

Who can fly?            question: can(X,fly)
```

Penguins — Proof (English)

```
(1) if X is a bird then normally X can fly     [sentence 1, confidence 90%]
(2) Robin is a bird                             [sentence 2]
(3) normally Robin can fly                      [from 1, 2, confidence 90%]
(4) if X can fly then X is an answer            [from question]
(5) Robin is an answer                          [from 3, 4, confidence 90%]

Exceptions checked: "Robin cannot fly" — not holding ✓

Why NOT Tweety?
  isa(penguin,tweety) → isa(bird,tweety) → normally can(tweety,fly)
  BUT: isa(penguin,tweety) → -can(tweety,fly)  [strict, blocks the default]
```

Answer: Robin (confidence 0.9).

Penguins — Proof with JSON (1/2)

```
(1) if X is a bird then normally X can fly          [sentence 1, @0.9]
    [["-isa","bird","?:X"],
     ["can","?:X","fly",["$ctxt","?:Y","?:Z","?:U","?:V"]],
     ["$block",["$","bird",1],
       ["$not",["can","?:X","fly",["$ctxt","?:Y","?:Z","?:U","?:V"]]]]]

(2) Robin is a bird                                  [sentence 2]
    ["isa","bird","robin"]

(3) normally Robin can fly                           [from 1, 2, @0.9]
    ["can","robin","fly",["$ctxt","?:X","?:Y","?:Z","?:U"]],
     ["$block",["$","bird",1],
       ["$not",["can","robin","fly",["$ctxt","?:X","?:Y","?:Z","?:U"]]]]]
```

Penguins — Proof with JSON (2/2)

```
(4) if X can fly then X is an answer                  [from question]
    [["-can","?:X","fly",["$ctxt","?:Y","?:Z","?:U","?:V"]],
     ["$ans","?:X","?:Y","?:Z","?:U","?:V"]]

(5) Robin is an answer, except when Robin cannot fly [from 3, 4, @0.9]
    [["$ans","robin","?:X3","?:Y3","?:Z3","?:U3"],
     ["$block",["$","bird",1],
       ["$not",["can","robin","fly",
         ["$ctxt","?:X3","?:Y3","?:Z3","?:U3"]]]]]

Exceptions checked: "Robin cannot fly" — no evidence found ✓

Tweety blocked by strict rule:
    isa(penguin,X) => -can(X,fly)    [confidence 1.0, defeats 0.9 default]

Answer: Robin (confidence 0.9).
```

Complex Syntax — Relative Clauses + Adjectives

"John bought a blue car. Mary saw the car which John bought.
Did Mary see a blue car?"

Three linguistic constructions:

- "a blue car" — adjective attached to entity
- "the car which John bought" — relative clause (coreference to same car)
- Question combines both: "a blue car" (adjective) verified across sentences

Stage 1 resolves coreference: car 2 = the bought car = the blue car = the seen car

Blue Car — Clausified Logic

```
John bought a car.                              The car is blue.
  has_type(sk0, buy, $ctxt(past, W0))             has_property(blue, car_B, $ctxt(past, W1))
  has_actor(sk0, john, $ctxt(past, W0))
  has_target(sk0, car_B, $ctxt(past, W0))

Mary saw the car which John bought.
  has_type(sk1, see, $ctxt(past, W1))
  has_actor(sk1, mary, $ctxt(past, W1))
  has_target(sk1, car_B, $ctxt(past, W1))       ← same car_B!

Did Mary see a blue car?
  [biconditional: isa(car,X) & has_property(blue,X) &
   has_type(E,see) & has_actor(E,mary) & has_target(E,X)]

Answer: True.
```

Complex Syntax — Nested Relative Clauses

"A man who ate breakfast had a car which a woman bought.
The car was red. The man who ate breakfast had the red car
which a woman bought?"

Three complex constructions:

- "a man who ate breakfast" — relative clause identifying the man
- "a car which a woman bought" — relative clause identifying the car
- "the red car" — adjective carried across sentences

Question combines ALL: man (breakfast eater) + car (red, woman-bought)

Complex Syntax — Clausified Logic

```
A man who ate breakfast had a car which a woman bought.
  isa(man,man_A)  isa(car,car_B)  isa(woman,woman_C)
  isa(activity,sk0)  has_type(sk0,eat)  has_actor(sk0,man_A)
  isa(breakfast,sk1)  has_target(sk0,sk1)                    // man ate breakfast
  have(man_A, car_B)                                          // man had car
  isa(activity,sk2)  has_type(sk2,buy)  has_actor(sk2,woman_C)
  has_target(sk2,car_B)                                       // woman bought car

The car was red.
  has_property(red, car_B)

Question: all 14 conditions must hold simultaneously:
  man_A is person ∧ car_B is car ∧ car_B is red ∧ man_A has car_B ∧
  ∃E1[eat(E1) ∧ actor(E1,man_A) ∧ target(E1,breakfast)] ∧
  ∃E2[buy(E2) ∧ actor(E2,woman_C) ∧ target(E2,car_B)]
```

Complex Syntax — Proof (28 steps, by contradiction)

```
Assume answer is "no", derive contradiction:

(1)  Question: if all 14 conditions hold → answer      [question]
(2)  car B is red                                       [sentence 2]
(3)  Assume: answer is no                               [contradiction setup]
     ...resolution chain eliminates conditions one by one...
(9)  man A has car B                                    [sentence 1]
(11) act1 is an eat event                               [sentence 1]
(15) man A performs act1                                [sentence 1]
(17) act1 targets breakfast bre1                        [sentence 1]
(19) bre1 is a breakfast                                [sentence 1]
(21) act2 is a buy event                                [sentence 1]
(25) woman C performs act2                              [sentence 1]
(26) act2 does NOT target car B  ← from "no" assumption  [derived]
(27) act2 DOES target car B                             [sentence 1]
(28) Contradiction!

Answer: True.  All cross-references verified.
```

Movement and World States

"John travelled to the hallway. Mary journeyed to the bathroom. Where is John?"

World state progression:

W0 → W1: John travels to hallway

W1 → W2: Mary journeys to bathroom

Question asks about latest state (W2)

Background axioms:

- travel/journey/move → go (synonyms)
- go + destination + next(W,W2) → is_rel2(at, X, Dest, present, W2)
- Frame: is_rel2 persists across worlds (blocked when entity moved)

Movement — Clausified Logic

```
John travelled to the hallway.                    next(W0, W1)
  has_type(sk0, travel, $ctxt(past, W0))
  has_actor(sk0, john, $ctxt(past, W0))
  has_destination(sk0, hallway_B, $ctxt(past, W0))

Mary journeyed to the bathroom.                   next(W1, W2)
  has_type(sk1, journey, $ctxt(past, W1))
  has_actor(sk1, mary, $ctxt(past, W1))
  has_destination(sk1, bathroom_D, $ctxt(past, W1))

Where is John?
  is_rel2(at,john,Q1,$ctxt(?,W2,...)) => $defq0(at,Q1)
  question: $defq0(Rel, Q1)
```

Movement — Proof

```
(1)  Movement axiom: go + destination + actor + next → at   [background]
(2)  act1 is a travel event                                  [sentence 1]
(3)  travel → go                                             [synonym axiom]
(4)  act1 is a go event                                      [from 2, 3]
(6)  act1 goes to hallway B                                  [sentence 1]
(8)  John performs act1                                      [sentence 1]
(10) next(W0, W1)                                            [sentence 1]
(11) John is at hallway B at W1                              [from 1,4,6,8,10]
     is_rel2(at, john, hallway_B, $ctxt(present, W1, ...))
(16) W1 before W2 → persistence to W2                        [frame axiom]
(17) John is at hallway B at W2                              [persistence]
(18) if John is at X then [at, X] is the answer              [question]
(19) [at, hallway] is the answer
```

Answer: At the hallway.

Measurements — Equality Contradiction

"Nile's length is 80 kilometers. The length of Nile is 90 kilometers?"

Encoding:

```
Assertion: =($measure_of(length, Nile, W0), $list(80000, #:meter))
Question:  =($measure_of(length, Nile, W0), $list(90000, #:meter)) ?
```

- `$measure_of(attr, entity, world)`: ground function term
- `$list(number, #:unit)`: structured value with UNA
- Canonical conversion: 80 km → 80000 meters
- `#:meter` is a distinct symbol (UNA in gk)

```
Prover: $measure_of terms unify (ground),
        $list decomposition: 80000 ≠ 90000 → contradiction
```

```
Answer: False.
```

"What" Questions — Class Answers

"Cats are afraid of wolves. Emily is a cat. Gertrude is a wolf.
What is Emily afraid of?"

Prover finds two answers:

- Emily afraid of Gertrude (concrete, confidence 0.98)
- Emily afraid of $some_wolf (population, confidence 0.98)

"What" questions prefer class answers over instances:

- Population fact isa(wolf, $some_wolf) generated for 'what' queries
- Tier filtering: population preferred over concrete
- Skolem function answers resolved to class via type tables

Answer: A wolf (confidence 0.98). Not "Gertrude"

"What" vs "Who"

Same facts, different question words:

"What is Emily afraid of?" → A wolf
"Who is afraid of wolves?" → Emily
"Who is an animal?" → John
"What is an animal?" → An elephant

The question word controls answer preference:

- "who" → prefer concrete entities (person names)
- "what" → prefer class/population answers

Where-Question with Defeasible Reasoning

"Bears eat berries in a forest. John is a bear. Where does John eat?"

Combines: default rule + event reification + location + where-question

```
Rule: isa(bear,X) → normally [∃E. activity(E) ∧ eat(E) ∧ actor(E,X)
                              ∧ target(E, berry) ∧ location(E, forest, in)]

Proof: John is a bear → normally John eats berries in a forest
  Confidence degrades through chain: 98% → 94% → 92%
  Where-question matches has_location with preposition expansion
  Skolem function sk2(john) = "the forest of John" → rendered as "the forest"

Answer: Likely in the forest (confidence 92%).
```

Pipeline Components

Component	Role
llmparse.py	Two-stage LLM parser, entity ID normalization
llmcall.py	LLM API wrapper (GPT/Claude/Gemini/DeepSeek), SQLite caching
logconvert.py	FOL → CNF orchestration, $ctxt injection, question dispatch
lc_clausify.py	Clausification, Skolemization, typed Skolem naming
lc_questions.py	Where/when/who/what question builders, population facts
lc_postprocess.py	Definites, measurements, less_measure, possessive have
lc_rewrites.py	Meta-predicate normalization, tense stripping
prover.py	GK interface, auto strategy selection
procproofs.py	Answer formatting, tier filtering, proof explanation
axioms_std.js	Movement, persistence, tense bridges, measurement comparison

Test Infrastructure

Ca 1500 test cases, of which 300+ representative test cases in testfixlog.txt with systematic debugging workflow

Tools:

- examine.py: runs all 5 pipelines in parallel, per-LLM log files
- test.py: automated test runner against tests_core.py
- Output levels: -logic, -explain, -debug, -details
- -gkin FILE: save prover input for standalone experimentation
- LLM response caching in SQLite (keyed on prompt + input)

Debug workflow: run examine → read all logs → compare LLMs → find root cause → fix → register

Current Status

Acceptably solved:

- Boolean, who, what, where, when questions
- Probabilistic reasoning with contradictory evidence
- Defeasible reasoning with exceptions (penguins, bears)
- Nested relative clauses, adjective propagation
- Multi-LLM support with consistent encoding
- Measurement comparison (equality, less_measure, unit conversion)
- World state tracking and movement

Open challenges:

- Frame problem: location persistence with biconditional questions
- Spatial containment hierarchy (room inside building etc)
- Efficient synonym / antonym handling
- Efficient similarities and analogues (queen is like king, but female; John has an apple → do not assume he has a pear)
- Long narrative event chains
- Complex causal reasoning
- Efficient simulation of the changing world
- Specialized guided proof strategies