Through the Looking Glass, and what Horn Clause Programs Found There

Paul Tarau[0000−0001−7192−9421]

University of North Texas
paul.tarau@unt.edu

> "Well, now that we have seen each other," said the unicorn, "if you'll believe in me, I'll believe in you."
>
> Lewis Carroll, Through the Looking-Glass and What Alice Found There

Abstract. Dual Horn clauses mirror key properties of Horn clauses. This paper explores the "other side of the looking glass" to reveal some expected and unexpected symmetries and their practical uses.
We revisit Dual Horn clauses as enablers of a form of constructive negation that supports goal-driven forward reasoning and is valid both intuitionistically and classically. In particular, we explore the ability to falsify a counterfactual hypothesis in the context of a background theory expressed as a Dual Horn clause program.
With Dual Horn clause programs, by contrast to negation as failure, the variable bindings in their computed answers provide explanations for the reasons why a statement is successfully falsified. Moreover, in the propositional case, by contrast to negation as failure as implemented with stable models semantics in ASP systems, and similarly to Horn clause programs, Dual Horn clause programs have polynomial complexity.
After specifying their execution model with a metainterpreter, we devise a compilation scheme from Dual Horn clause programs to Horn clause programs, ensuring their execution with no performance penalty and we design the embedded SymLP language to support combined Horn clause and Dual Horn clause programs.
As a (motivating) application, we cast LLM reasoning chains into propositional Horn and Dual Horn clauses that work together to constructively prove and disprove goals and enhance Generative AI with explainability of reasoning chains.
Keywords: Dual Horn clauses; constructive negation; counterfactual reasoning; theory falsification; LLM generated logic programs; metainterpretation and compilation to Prolog.

1 Introduction

The concept of negation in logic programming traditionally relies on "negation as failure", which infers the negation of a proposition if the proposition itself cannot be

proven. This approach, while useful, often lacks the ability to provide clear explanations for why a proposition is considered false (given the absence of variable bindings). This is a problem, especially in complex scenarios where understanding the reasoning process is crucial.

In fact, negation as failure tacitly reflects a meta-level property about the proof procedure into the object-language, and as this can be iterated on, the resulting semantics is unavoidably subtle requiring carefully designed guardrails for both practical programming uses and theoretical foundations. Among them, while elegant and expressive, the stable-models semantics [11] based handling of negation in Answer Set Programming (ASP) is already NP-complete even in the propositional case [5] and must rely on grounding with exponential space complexity in the case of Datalog programs.

Curiosity about the possibility of a simpler and more informative negation mechanism in Logic Programming brings us to our "back to the future" revisiting of Dual Horn clauses.

Dual Horn clauses have emerged as an interesting set of propositional formulas in Schaefer's "*dichotomy theorem*", that classifies propositional formulas in P-complete vs. NP-complete classes [16], by falling in the P-complete class together with their Horn clause cousins.

Surprisingly, while their theoretical properties have been well known for very long time, we have found no evidence of their uses in programming or knowledge representation tasks and no mentions of uses in more expressive logic languages beyond the propositional case.

In part, this motivates our curiosity for revisiting them simply as logic programming expressiveness enhancers. Another motivation comes from the ability to generate interesting propositional Dual Horn clause programs with LLMs, when the problem we focus on involves forward reasoning from a fact to its consequences. This contrasts to Prolog's usual backward reasoning elaboration of a goal into progressively more solvable alternatives and subgoals.

As we will show next, it turns out that the significance of Dual Horn clauses centers around their ability to constructively handle negation, moving beyond the limitations of negation as failure. By enabling *goal-driven forward reasoning*, Dual Horn clauses support the exploration and falsification of hypotheses in the presence of negative factual information. This capability is particularly valuable in fields where reasoning about counterfactuals and understanding the underlying logic of decisions are important. With constructive logic in mind, we will also point out the properties shared between Horn clause and Dual Horn clause programs that also hold in Intuitionistic Logic when represented in implicational form (**section** 2).

After describing a metainterpreter, we devise a compilation scheme that transforms Dual Horn clause programs into conventional Horn clause programs allowing their execution with no performance penalty. This development ensures that the enhanced capabilities of Dual Horn clauses can be seamlessly integrated into existing logic programming systems, leading to a sketch of an embedded language accommodating Horn clauses and Dual Horn clause working together, `SymLP` (**section** 3). We will describe, using the notations of `SymLP` several examples of reasoning patterns including default

reasoning, theory falsification, decision on conflicting information and interactions with negation as failure (**section** 4).

We will also expand the utility of Dual Horn clauses by exploring their application in the context of Generative AI and Large Language Models (LLMs). By casting the reasoning steps of LLMs into propositional Horn and Dual Horn clauses, we provide a framework that not only supports the proving or disproving of goals but also enhances the explainability of the LLMs' decision-making processes. The ability to explain AI decisions transparently is crucial for building trust and for the practical deployment of AI systems in sensitive or impactful domains (**section** 5).

After discussing related work (**section** 6) we will conclude the paper and discuss future work (**section** 7).

The SWI-Prolog code described in the paper is available online[1].

2 Background on Dual Horn clauses

A *Horn clause* is a disjunction of literals with at most one positive literal. A *Definite Horn clause* is a disjunction of literals with exactly one positive literal. A *Dual Horn clause* is a disjunction of literals with at most one negative literal. A *Definite Dual Horn clause* is a disjunction of literals with exactly one negative literal. *A Horn clause Program* is a conjunction of Horn clauses. A *Dual Horn clause Program* is a conjunction of Dual Horn clauses.

The following formulas describe Horn clauses (1) and (2), vs. Dual Horn clauses (3) and (4). Note the disjunctive forms as expressed, for instance, in resolution theory and the implicational forms as expressed in Prolog or Answer Set Programming (ASP) programs.

Horn

$$p_0 \vee \neg p_1 \vee \ldots \vee \neg p_n. \tag{1}$$

$$p_0 \leftarrow p_1 \wedge \ldots \wedge p_n. \tag{2}$$

Dual Horn

$$\neg p_0 \vee p_1 \vee \ldots \vee p_n. \tag{3}$$

$$p_0 \rightarrow p_1 \vee \ldots \vee p_n. \tag{4}$$

Similarly, the two forms make sense when extending Definite Horn Horn clauses with integrity constraints called *denials* [9] (see (5) and (6)), asserting that at least one p_i must be false and their duals (see (7) and (8)), asserting that at least one p_i must be true.

Horn

$$\neg p_1 \vee \ldots \vee \neg p_n. \tag{5}$$

[1] `https://github.com/ptarau/TypesAndProofs/tree/master/symlp`

$$false \leftarrow p_1 \wedge \ldots \wedge p_n. \tag{6}$$

Dual Horn

$$p_1 \vee \ldots \vee p_n. \tag{7}$$

$$true \rightarrow p_1 \vee \ldots \vee p_n. \tag{8}$$

While the usual assumption about the underlying logic in Prolog and SAT solvers is classical logic (CL) or a fairly close intermediate logic in the case of ASP systems, the proof-theoretical semantics [14] of Horn clause Logic in implicational form has been known to be compatible also with its reading in Intuitionistic Logic (IL). We will keep this in mind when covering some key properties of both Horn clause and Dual Horn clause programs.

Proposition 1 – *(1) implies (2) in IL and (1) is equivalent to (2) in CL.*
– *(3) implies (4) in IL and (3) is equivalent to (4) in CL.*
– *(5) implies (6) in IL and (5) is equivalent to (6) in CL.*
– *(7) and (8) are equivalent both in IL and CL.*

Proofs of the CL statements are trivial using De Morgan and definition of material implication. The implicational forms in IL are weaker given that De Morgan holds only one way. Proofs in IL are otherwise easy using a sequent calculus system like [7] or its corresponding sound and complete theorem prover [18].

Proposition 2 *Satisfiability of Horn clause and Dual Horn clause formulas with or without integrity constraints is P-complete.*

This follows from Schaefer's dichotomy theorem classifying propositional formulas [16] in P-complete vs. NP-complete types. Note that finding a renaming that might turn a set of clauses into a set of Horn or Dual Horn clauses are also polynomial [3].

Example 1 *We start with a small Dual Definite program, adopting Prolog-like syntax, with* $\rightarrow$ *represented as "*`=>`*" and* $\vee$ *represented as ";".*

```
p => q ; r.
q => r ; s.
r => false.
s => false.
```

Note also that "`s => false`" represents a negated fact (in either CL or IL), the same way as "`s :- true`" would represent a positive fact. Let's proceed with `p` as our goal, similarly as if we would evaluate a Horn clause program. Assuming that `p` were true, we would infer that at least one of `q` , `r` and `s` should be true.That reduces to the truth of `r` and `s` and finally the truth of `s`. But `s` implies `false`, which then backpropagates, falsifies `s` and `r` and then falsifies the initial goal `p`. Note that this reasoning is also intuitionistically valid similarly to its Horn clause counterpart.

We conclude from this example that we have a goal oriented *falsification* process for Dual Horn programs that mimics *verification* in Horn Programs via SLD-resolution. It is easy to see that programs with variable bindings generated via unification will work also in a similar way.

More generally, the falsification process relies on the following fact:

Proposition 3 *The relations (9) and (10) hold both in IL and CL.*

$$(p_0 \leftarrow p_1 \wedge \ldots \wedge p_n) \wedge p_1 \wedge \ldots \wedge p_n \rightarrow p_0 \tag{9}$$

$$(p_0 \rightarrow p_1 \vee \ldots \vee p_n) \wedge \neg p_1 \wedge \ldots \neg p_n \rightarrow \neg p_0 \tag{10}$$

Thus, based on (10), to falsify p_0 we need to falsify all the disjuncts p_i that are consequences of p_0, the essence of a goal-driven falsification process, operationally similar to SLD-resolution proof procedure on Horn clause programs as illustrated by (9).

3 Symmetric Logic Programming (SymLP) : the two sides of the mirror, together

The next step is to design a mechanism for the "safe cohabitation" of Horn clauses and Dual Horn clauses in the same program. To ensure that their predicates defining rules and facts are disjoint (for instance, to avoid contradictions) we will place the Horn clause component in module `true` and the Dual Horn clause component in module `false`. Note that using modules, is mostly a syntactic simplification for implementing them in Prolog, as any mechanism ensuring that the predicate symbols are distinct would do (e.g., by prefixing predicate names with symbols `true` and `false`).

3.1 Syntax

To embed the SymLP language that combines Horn clauses and Dual Horn clauses into Prolog, we will use a few operators. We use "`+`" to mark facts that are true and "`-`" to mark facts that are false. We will use "`=>`" to represent implication and "`<=`" to represent reverse implication. Note also that `-p` can be seen as a shortcut for `p=>false` and that similarly, `+p` can be seen as a shortcut for `p<=true`. We will borrow from Prolog the usual notation for conjunction "`,`" and disjunction "`;`".

3.2 Implementing Symmetric Logic Programs

Next, we will describe a surprisingly simple mechanism to implement a goal-oriented execution mechanism for Dual Horn programs, inspired by the SLD-resolution mechanism of Prolog.

3.3 The dual use meta-interpreter

We start with a simple difference-list based metainterpreter that will turn out to work for both Horn clause and Dual Horn clause programs.

```
metaint([]).          % no more goals left, succeed
metaint([G|Gs]):-     % unify the first goal with the head of a clause
   cls([G|Bs],Gs),    % build a new list of goals from the body of the
                      % clause extended with the remaining goals as tail
   metaint(Bs).       % interpret the extended body
```

The following examples show that the representation provided as the predicate `cls/2` works in both cases.

Example 2 *Simple Horn program with Prolog-like notation:*

```
p <= q,r.
q <= r,s.
r <= true,
s <= true.
```

Difference list representation of clauses:

```
cls([ p,q,r      |Tail],Tail).
cls([ q,r,s      |Tail],Tail).
cls([ r          |Tail],Tail).
cls([ s          |Tail],Tail).
```

Successful verification:

```
?- metaint([p]).
true
```

Example 3 *Dual Horn program with* `=>` *as implication and* `;` *as disjunction.*

```
p => q ; r.
q => r ; s.
r => false.
s => false.
```

We can use the same *difference list representation of Dual Horn clauses:*

```
cls([ p,q,r      |Tail],Tail).
cls([ q,r,s      |Tail],Tail).
cls([ r          |Tail],Tail).
cls([ s          |Tail],Tail).
```

Successful falsification is now computed as well as:

```
?- metaint([p]).
true
```

Proposition 4 *Falsification of a goal of a Dual Horn clause program by the metainterpreter succeeds if and only if the proof of the goal with a corresponding Horn Clause program succeeds.*

Proof*: By flipping each literal in formula (3), formula (1) is obtained, which is a Horn clause. Note that success of the interpreter on (1) corresponds to its success on (3).*

Note that the meta-intepreter also accommodates terms with variables, in which case variable bindings will be informative about *why something is successfully falsified*.

3.4 The Compilation to Prolog clauses

Observing that after translation to a uniform clause representation the same meta-interpreter works for Horn programs and their duals, suggests that a simple compilation scheme from Dual Horn clauses to Horn clauses must exist, and when extended to the case of SymLP programs, it will just keep Horn clauses invariant.

We will implement it here using Prolog's `term_expansion`[2] that overloads the Prolog reader with a call to a `compile_clauses(SymLPclause,EquivalentPrologclause)`. The key idea is to convert Dual Horn clauses occurring in a Prolog program to corresponding Horn clauses that, when executed, follow the semantics specified by the dual-use meta-interpreter.

We will distinguish successful falsification from successful proof of the "compiled" Horn program simply by placing the result of the `term_expansion/2` of Horn clauses into the module `true` and the result for Dual Horn clauses into the module `false`.

After defining "`<=`" (reverse implication) and "`=>`" (implication) as operators, the predicate `compile_clauses` will be called at term expansion time to convert the Dual Horn clauses to Horn clauses to be placed in module "`false`". Note also that in the bodies of Dual Horn clauses disjunctions will be converted to conjunctions. Facts marked with "`-`" will be placed in module `false`. The action on Horn clauses introduced by "`<=`" is just replacing "`<=`" with "`:-`" to be then added together with facts marked with "`+`" to the module "`true`".

```
:- multifile(term_expansion/2).

:-op(1199,xfx,(=>)).
:-op(1199,xfx,(<=)).
```

```
compile_clauses(C,_):-var(C),!,fail.

compile_clauses((H<=B),true:(H:-B)):-!,nonvar(H),nonvar(B).
compile_clauses((+H),true:H):-!,nonvar(H).

compile_clauses((H=>B),R):-nonvar(H),nonvar(B),!,dual2clause((H=>B),R).
compile_clauses((-H),false:H):-nonvar(H).
```

[2] `https://www.swi-prolog.org/pldoc/man?predicate=term_expansion/2`

```
dual2clause((H=>false),false:(H)):-!.
dual2clause((H=>B),false:(H:-CB)):-disj2conj(B,CB).
dual2clause((-H),false:H).

disj2conj((A;B),(CA,CB)):-nonvar(A),nonvar(B),!,
    disj2conj(A,CA),
    disj2conj(B,CB).
disj2conj(A,A).
```

Example 4 *As the Horn clauses get compiled simply by replacing* `<=` *with* `:-`*, we will illustrate next what happens to a Dual Horn clause program:*

```
:-include('compile_clauses.pro').

p => q ; r.
q => r ; s.
r => false.
s => false.
```

becomes:

```
% in module false
p :- q , r.
q :- r , s.
r :- true.
s :- true.
```

Then, querying it with:

```
?- false:p.
true
```

the success confirms that p *is indeed falsifiable.*

4 Reasoning patterns expressed in SymLP

We will next overview a few reasoning patterns expressed as SymLP programs.

4.1 Default reasoning, a dual view

Example 5 *Dually to the usual way to express defaults and exceptions, we just declare which birds are not challenged when defining which of them can fly.*

```
:-include('compile_clauses.pro').

fly(X) <= bird(X),false:challanged_bird(X).
```

```
+bird(tweety).
+bird(chicken_little).
+bird(eagle_joe).
+bird(humming_jenny).

-challanged_bird(eagle_joe).
-challanged_bird(humming_jenny).
```

```
?- true:fly(X).
X = eagle_joe ; X = humming_jenny.
```

4.2 Dual Horn programs and the logic of theory falsification

Falsifiability of a theory has been known for a long time [15] as instrumental to make the theory predictive and testable, and thus useful in practice. In terms of Dual Horn programs, this is expressed by saying that something should fail if its consequences fail and by observing that falsity propagates back from false facts to rules that rely to them. This suggest a proof procedure to be applied to Dual Horn clauses described in formula (4).

Example 6 *We will clarify this by working out an example of "theory falsification"*[3].

```
:-include('compile_clauses.pro').

'Negative gravity fields are possible' =>
    'Planets and stars would disperse' ;
    'Atmospheres of planets would be pushed away from their surfaces' ;
    'Unresolvable paradoxes in physics'.

'Planets and stars would disperse' => false.
```

```
'Unresolvable paradoxes in physics' =>
   'Relativity theory is incorrect' ;
   'Quantum field theory is incorrect'.

'Relativity theory is incorrect' => false.
'Quantum field theory is incorrect' => false.

'Atmospheres of planets would be pushed away from their surfaces' => false.
```

The result of the "execution" of this Dual Horn clause program could be expressed by a query goal of the form:

```
?- false:'Negative gravity fields are possible'.
true
```

[3] obtained by edits of an LLM request to explain problems with assuming the existence of negative gravity fields

confirming that "`Negative gravity fields are possible`" has been successfully falsified in the context of our accepted background knowledge about physics.

4.3 Combining positive and negative advice in a SymLP program

The next example of SymLP program will have calls across the `true` and `false` modules. Note that the `include` statement will trigger the compilation process, as explained in subsection 3.4.

Example 7 *Combining nuances of (a fictional example) of advice on stocks.*

```
:-include('compile_clauses.pro').

cautious_buy(X)<=recommended(X),safe(X).

safe(X) <= false:volatile(X).
safe(X) <= false:overvalued(X).
safe(X) <= true:stable(X).

+recommended(qqq).
+recommended(bitcoin).
+recommended(apple).
+recommended(meta).
+recommended(berkshire).

+stable(att).
+stable(berkshire).

volatile(X) => big_price_changes_last_month(X).

-big_price_changes_last_month(apple).
-big_price_changes_last_month(meta).
-big_price_changes_last_month(comcast).

-overvalued(qqq).
```

After combining positive and negative facts and inferences drawn from them in Horn module `true` *and Dual Horn module* `false` *the call to* `cautious_buy(X)` *in module* `true` *will generate the following answers:*

```
?- true:cautious_buy(X).
X = qqq ;
X = apple ;
X = meta ;
X = berkshire.
```

At this point, one might want to ask the legitimate question:

> Why would we represent negation by explicitly listing negative facts, knowing that something like `not(white(swan,X))` will include not just "black swans" but also an infinite set of unrelated entities (e.g., "globular galaxies") ?

First, like in the stock market advising in Example 7, decisions are justified by a small set of positive or negative facts from where a decision process initiates. Next, reasoning with Machine Learning datasets (including those used in Inductive Logic Programming) relies on finite sets of positive and negative facts. In particular, in Generative AI, a way to control hallucinations is by implementing, as part of a a multi-agent framework, generation of positive and negative facts and rules relying on them, as we will show in section 5.

4.4 Negation as failure to prove and affirmation as failure to disprove

Assuming that we have compiled our SymLP e program into the modules `true` and `false`, negation as failure (`unverifiable/1`) and its dual (`unfalsifiable`) are implemented as follows:

```
unverifiable(X):-not(true:X).
unfalsifiable(X):-not(false:X).
```

Note that combining modules `true` and `false` into one program has the same complexity as Horn clause programs. Thus it has a single minimal model that can be computed in polynomial time in the propositional case. This holds true also if `not/1` calls between modules `true` and `false` result in a stratified compiled program [8]. Otherwise, we are back to the usual pitfalls of Prolog's negation as failure. Therefore, a semantically safe and simple use of Prolog's `not/1` would be to only apply it to predicates in modules `true` and `false` from *outside* (e.g., from Prolog's module "`user`").

Example 8 *Using Prolog's* `not/1` *in a SymLP program.*

```
:-include('compile_clauses.pro').

exonerated(X) <= suspect(X),false:proven_guilty(X).

investigated(X) <= suspect(X),not(false:proven_guilty(X)).

+suspect(alice).
+suspect(bob).

proven_guilty(X) => found_of(X,dna) ; found_of(X,fingerprints).

-found_of(alice,_anything).
```

Not falsifiable *that Bob is* `proven_guilty` *only entails that he is still investigated (vs. exonerated if not* proven *guilty in the case of Alice).*

```
?- exonerated(X).
X = alice.

?- investigated(X).
X = bob.
```

Let us just mention that, similarly to ASP and s(CASP), concepts related to epistemic modalities emerge, when combining the independent opinions originating on the two sides:

- things that are clearly unknown: failure to prove and failure to disprove
- things that are strongly accepted as known: successfully proven and failing to disprove
- things that are very likely to be impossible: successfully falsified and failing to prove.

5 LLM generated Dual Horn programs

Given the ability to steer an LLM to explore recursively a given topic while staying focussed on the objective specified by an initiator goal [20,19], the LLM can be used to generate large sets of high quality positive or negative facts as well as rules describing their inference steps, to be all exported as a Prolog program.

Example 9 *The following is a complete example of a propositional Dual Horn clause program*[4] *generated from a recursively explored (up to depth = 2) initiator goal asking to falsify the misguided belief about: "*escalation risks after use of tactical nuclear weapons*":*

```
'escalation risks after use of tactical nuclear weapons'=>
    'Global nuclear war';
    'Uncontrollable retaliation cycles'.
'Global nuclear war'=>
    'Widespread radioactive fallout',
    'Massive civilian casualties',
    'Long-term environmental damage',
    'Global economic collapse',
    'Irreversible climate change',
    'Extensive agricultural failure'.
'Widespread radioactive fallout'=>
    'Long-term environmental damage';
    'Massive civilian casualties'.
```

```
'Long-term environmental damage'=>
    'Persistent ecosystem disruption',
    'Chronic health conditions',
    'Irreversible soil contamination',
    'Permanent loss of biodiversity';
    'Widespread radiation exposure';
    'Persistent ecological disruption';
```

[4] Ready to be tried out online with the DeepLLM system at `https://deepllm.streamlit.app/`

```
'Massive civilian casualties'=>
    'Severe humanitarian crises',
    'Overwhelmed medical systems',
    'Long-term psychological impacts',
    'Economic destabilization';
    'Widespread environmental destruction';
    'Long-term radiation effects'.
'Widespread environmental destruction'=>
    'Long-term habitat loss',
    'Severe biodiversity decline',
    'Persistent soil contamination',
    'Irreversible climate impacts',
    'Chronic water shortages'.
```

```
'Long-term radiation effects'=>
    'Genetic mutations',
    'Chronic health disorders',
    'Environmental contamination',
    'Agricultural degradation'.
'Widespread radiation exposure'=>
    'Genetic mutations increase',
    'Agricultural crop failure',
    'Chronic health deterioration',
    'Wildlife population decline'.
```

```
'Persistent ecological disruption'=>
    'Species extinction rates',
    'Reduced agricultural productivity',
    'Chronic health conditions',
    'Loss of biodiversity'.
```

```
'Global economic collapse'=>
    'Widespread humanitarian crisis';
    'Irreversible environmental damage';
    'Severe job losses',
    'Market instability',
    'Investment crashes',
    'Supply chain disruptions',
    'Increased poverty rates',
    'Currency devaluation'.
```

```
'Widespread humanitarian crisis'=>
    'Massive refugee movements',
    'Severe food shortages',
    'Critical medical deficiencies',
    'Intensified civil unrest'.
```

```
'Irreversible environmental damage'=>
    'Long-term habitat destruction',
    'Permanent loss of biodiversity',
    'Severe air quality deterioration',
    'Unrecoverable soil degradation',
    'Extinction of critical species',
    'Irreversible climate alterations'.
```

```
'Irreversible climate change'=>
    'Widespread ecological collapse';
    'Permanent agricultural devastation'.
'Widespread ecological collapse'=>
    'Massive species extinction',
    'Severe food shortages',
    'Chronic water scarcity',
    'Intensified natural disasters'.
```

```
'Permanent agricultural devastation'=>
    'Widespread famine crisis',
    'Long-term soil infertility',
    'Severe food shortages',
    'Collapse of ecosystems'.
'Extensive agricultural failure'=>
    'Widespread famine crisis';
    'Severe ecological disruption'.
```

```
'Widespread famine crisis'=>
    'Severe malnutrition rates',
    'Increased mortality rates',
    'Economic instability',
    'Social unrest';
'Severe ecological disruption'=>
    'Biodiversity loss',
    'Habitat destruction',
    'Soil degradation',
    'Water resource depletion'.
```

```
'Uncontrollable retaliation cycles'=>
    'Global security destabilization',
    'Widespread humanitarian crises',
    'Escalated military conflicts',
    'Severe economic disruptions'.
'Global security destabilization'=>
    'Unpredictable military responses';
    'Widespread humanitarian crises'.
```

```
'Unpredictable military responses'=>
    'Widespread civilian casualties',
    'Severe economic disruptions',
    'Long-term environmental damage',
    'Sudden geopolitical shifts'.
```

```
'Widespread humanitarian crises'=>
    'Massive displacement waves',
    'Severe resource shortages',
    'Intensified disease outbreaks',
    'Heightened conflict incidents';
    'Global economic collapse';
    'Massive refugee movements';
    'Massive refugee movements',
    'Intensified food shortages',
    'Overwhelmed medical systems',
    'Increased child mortality'.
```

```
'Massive refugee movements'=>
    'Overburdened local resources',
    'Increased social tensions',
    'Strained healthcare systems',
    'Economic destabilization',
    'Environmental degradation'.
```

```
'Escalated military conflicts'=>
    'Global humanitarian crises';
    'Widespread ecological disasters'.
'Global humanitarian crises'=>
    'Widespread famine outbreaks',
    'Mass displacement waves',
    'Severe medical shortages',
    'Intensified poverty levels'.
'Widespread ecological disasters'=>
    'Long-term habitat destruction',
    'Severe biodiversity loss',
    'Persistent soil contamination',
    'Irreversible water pollution'.
'Severe economic disruptions'=>
    'Global market collapse';
    'Widespread humanitarian crises';
'Global market collapse'=>
    'Widespread unemployment surge',
    'Investment capital evaporation',
    'Consumer spending plummet',
    'International trade paralysis',
    'Financial sector instability'.
```

```
-'Persistent ecosystem disruption'.
-'Chronic health conditions'.
-'Irreversible soil contamination'.
-'Permanent loss of biodiversity'.
-'Severe humanitarian crises'.
-'Overwhelmed medical systems'.
-'Long-term psychological impacts'.
-'Economic destabilization'.
-'Long-term habitat loss'.
-'Severe biodiversity decline'.
-'Persistent soil contamination'.
-'Irreversible climate impacts'.
-'Chronic water shortages'.
-'Genetic mutations'.
-'Chronic health disorders'.
-'Environmental contamination'.
-'Agricultural degradation'.
```

```
-'Genetic mutations increase'.
-'Agricultural crop failure'.
-'Chronic health deterioration'.
-'Wildlife population decline'.
-'Species extinction rates'.
-'Reduced agricultural productivity'.
-'Loss of biodiversity'.
-'Severe food shortages'.
-'Critical medical deficiencies'.
-'Intensified civil unrest'.
-'Long-term habitat destruction'.
-'Severe air quality deterioration'.
-'Unrecoverable soil degradation'.
-'Extinction of critical species'.
-'Irreversible climate alterations'.
-'Massive species extinction'.
-'Chronic water scarcity'.
```

```
-'Intensified natural disasters'.
-'Long-term soil infertility'.
-'Collapse of ecosystems'.
-'Severe malnutrition rates'.
-'Increased mortality rates'.
-'Economic instability'.
-'Social unrest'.
-'Biodiversity loss'.
-'Habitat destruction'.
-'Soil degradation'.
-'Water resource depletion'.
-'Widespread civilian casualties'.
```

```
-'Sudden geopolitical shifts'.
-'Massive displacement waves'.
-'Severe resource shortages'.
-'Intensified disease outbreaks'.
-'Heightened conflict incidents'.
-'Severe job losses'.
-'Market instability'.
-'Investment crashes'.
-'Supply chain disruptions'.
-'Increased poverty rates'.
-'Currency devaluation'.
-'Overburdened local resources'.
-'Increased social tensions'.
-'Strained healthcare systems'.
-'Environmental degradation'.
-'Widespread famine outbreaks'.
-'Mass displacement waves'.
-'Severe medical shortages'.
-'Intensified poverty levels'.
```

```
-'Severe biodiversity loss'.
-'Irreversible water pollution'.
-'Widespread unemployment surge'.
-'Investment capital evaporation'.
-'Consumer spending plummet'.
-'International trade paralysis'.
-'Financial sector instability'.
-'Intensified food shortages'.
-'Increased child mortality'.
```

Note that the "-" operator marking negation is interpreted here as an unwanted outcome, from which, a rational agent would propagate back the denial of the initiator goal, implying that "*escalation risks after use of tactical nuclear weapons*" is something to avoid, given its consequences.

Similar initiator queries can cover recursive descent in consequences of things like unobservable / unmeasurable claims, undesirable outcomes of planned actions, implications of hidden legalese in contracts as well as untruthful advertisements or political persuasion hyperbolae.

By default, the DeepLLM system [20] generates propositional programs together with their unique minimal model[5] and its low complexity polynomial solver can support scaling to very large programs aggregating positive and negative knowledge snippets consisting of facts and rules of Horn and Dual Horn programs. By asking an LLM to decompose recursively a task into subtasks organized as an AND-OR tree, the generated Prolog file will be a Horn clause program. Dually, by asking an LLM to explore

[5] `https://github.com/ptarau/recursors`

consequences of an undesirable state of the world or of a hypothesis that one would want to reject, the results will take the shape of a Dual Horn clause program.

This abundance of positive and negative information offers a fully explainable and semantically straightforward alternative to default reasoning and alleviates the contrast between the underlying open vs. closed world assumptions, given that arbitrary positive or negative information and constructive inference based on it is available on demand.

Explicit reasoning with negative information is also relevant for *unlearning* algorithms (in particular in-context unlearning) that ensure removal of stale information (e.g., who is the president of USA), untruthful content or copyrighted, toxic, dangerous, and otherwise harmful content (e.g., instructions on how to make napalm) [13].

6 Related Work

Horn clause formulas and Dual Horn clause formulas (called weakly negative and, respectively, weakly positive in [16]) are proven in his "dichotomy theorem" (under the assumption $P \neq NP$) to be among the classes of propositional formulas that are in P. A graph-based linear algorithm exists for for Horn clause formulas, described in [6] and given the linear renaming algorithm of propositional variables in [3], the same applies to Dual Horn formulas. As a follow-up of [6], the Hornlog system, a logic programming alternative to Prolog covering (besides definite programs) the handling of denial integrity constraints has been implemented [10].

A salient question one might ask is "*If Dual Horn clauses have been known for a long time, why they haven't been widely used until now?*".

A possible explanation is that the practical usefulness of negation as failure in Prolog and the related theoretical ramifications, culminating with the stable model semantics [11] and the emergence of Answer Set Programming as an alternative logic programming execution model have provided enough expressive power to deal with more subtle nuances of negative information.

Another is that constructive negation algorithms, going back to [2,17] and present in goal-directed ASP systems [1], have provided, in the form of constraints on variable bindings, similar explanations covering the reasons of negative outcomes. Also, a compilation scheme from normal logic programs to definite programs under given stratification constraints has been devised in [12]. Note however that constructive negation expressed as a set of "is different from" constraints on a set of variables is fundamentally weaker than inferring in Dual Horn clause logic what the actual values should be based on known negative facts. More precisely, in the case of classical constructive negation, these results were expressed as disjunction of conjunctions of the form `X\=T` where `T` is a (usually ground) term, thus describing what `X` cannot be, while in the case of Dual Horn programs the result is a positive binding of the form `X=T` saying what `X` actually is. Thus, while the classical "constructive" negation in combination with constraint solvers provides an efficient filtering mechanism over a potentially infinite set of terms, it shares with negation as failure the fact that it can only reject bindings constructed elsewhere in the program but it cannot actually *construct* variable bindings as its name would (somewhat inadvertently) suggest.

7 Conclusion and Future Work

We have revisited Dual Horn clauses as a "back to the future" endeavor, motivated by their syntactic simplicity and straightforward semantics. As a result, we have devised a compilation scheme that integrates them into Prolog programs. This enables our SymLP embedded language as a practical programming tool that reasons with explicitly specified positive and negative facts and rules. In case of the possibly very large propositional programs generated by LLMs, the low polynomial complexity[6] ensures tractability – a key requirement for practical applications.

To some extent, the utility of Dual Horn program relies on the assumption that full knowledge of positive and negative facts can be acquired given their encapsulation in LLMs or traditional knowledge bases. This leaves open the possibility of unknown facts (e.g., those reached at the DeepLLM recursion depth limit) that can be seen as *abducibles*, i.e., verifiable if passing integrity constraints on the Horn clause side (formula (6)) or falsifiable if passing the corresponding constraints on the Dual Horn side (formula (8)). Future work will be needed to study these in full detail, while aware that their presence, in the propositional case will keep complexity polynomial.

This also opens the possibility to rely on soft-unification [4], relevant especially in the presence of LLM-generated facts stored as embeddings into a vector database. For instance, closeness via cosine-similarity to positive or negative facts could decide rules on which side would adopt these facts as abducibles, another future work direction worth to be explored.

References

1. ARIAS, J., CARRO, M., SALAZAR, E., MARPLE, K., GUPTA, G.: Constraint answer set programming without grounding. Theory and Practice of Logic Programming **18**(3-4), 337–354 (2018). https://doi.org/10.1017/S1471068418000285
2. Chan, D.: Constructive Negation Based on the Completed Database. In: Kowalski, R.A., Bowen, K.A. (eds.) Logic Programming, Proceedings of the Fifth International Conference and Symposium, Seattle, Washington, USA, August 15-19, 1988 (2 Volumes). pp. 111–125. MIT Press (1988)
3. Chandru, V., Coullard, C.R., Hammer, P.L., Montañez, M., Sun, X.: On renamable Horn and generalized Horn functions. Annals of Mathematics and Artificial Intelligence **1**(1-4), 33–47 (sep 1990). https://doi.org/10.1007/BF01531069, https://doi.org/10.1007/BF01531069
4. Cingillioglu, N., Russo, A.: Learning invariants through soft unification. In: Larochelle, H., Ranzato, M., Hadsell, R., Balcan, M., Lin, H. (eds.) Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual (2020), https://proceedings.neurips.cc/paper/2020/hash/5d0d5594d24f0f955548f0fc0ff83d10-Abstract.html
5. Dantsin, E., Eiter, T., Gottlob, G., Voronkov, A.: Complexity and expressive power of logic programming. ACM Comput. Surv. **33**(3), 374–425 (sep 2001). https://doi.org/10.1145/502807.502810, https://doi.org/10.1145/502807.502810

[6] actually linear if using the graph-based algorithm of [6])

6. Dowling, W.F., Gallier, J.H.: Linear-Time Algorithms for Testing the Satisfiability of Propositional Horn Formulae. J. Log. Program. **1**(3), 267–284 (1984), https://doi.org/10.1016/0743-1066(84)90014-1
7. Dyckhoff, R.: Contraction-free sequent calculi for intuitionistic logic. Journal of Symbolic Logic **57**(3), 795–807 (1992). https://doi.org/10.2307/2275431
8. Eiter, T., Fink, M., Tompits, H., Woltran, S.: Complexity Results for Checking Equivalence of Stratified Logic Programs. In: IJCAI. pp. 330–335 (2007)
9. Eshghi, K., Kowalski, R.A.: Abduction Compared with Negation by Failure. In: Levi, G., Martelli, M. (eds.) Logic Programming, Proceedings of the Sixth International Conference, Lisbon, Portugal, June 19-23, 1989. pp. 234–254. MIT Press (1989)
10. Gallier, J.H., Raatz, S.: HORNLOG: A graph-based interpreter for general Horn clauses. The Journal of Logic Programming **4**(2), 119–155 (1987). https://doi.org/https://doi.org/10.1016/0743-1066(87)90015-X, https://www.sciencedirect.com/science/article/pii/074310668790015X
11. Gelfond, M., Lifschitz, V.: The stable model semantics for logic programming. In: Kowalski, R.A., Bowen, K.A. (eds.) Logic Programming, Proceedings of the Fifth International Conference and Symposium, Seattle, Washington, USA, August 15-19, 1988 (2 Volumes). pp. 1070–1080. MIT Press (1988)
12. Kanchanasut, K., Stuckey, P.: Eliminating negation from normal logic programs. In: Kirchner, H., Wechler, W. (eds.) Algebraic and Logic Programming. pp. 217–231. Springer Berlin Heidelberg, Berlin, Heidelberg (1990)
13. Liu, K.Z.: Machine unlearning in 2024 (Apr 2024), https://ai.stanford.edu/ kzliu/blog/unlearning
14. Miller, D.: A survey of the proof-theoretic foundations of logic programming. CoRR **abs/2109.01483** (2021), https://arxiv.org/abs/2109.01483
15. Popper, K.R.: The Logic of Scientific Discovery. Hutchinson, London (1934)
16. Schaefer, T.J.: The complexity of satisfiability problems. In: Proceedings of the Tenth Annual ACM Symposium on Theory of Computing. p. 216?226. STOC '78, Association for Computing Machinery, New York, NY, USA (1978). https://doi.org/10.1145/800133.804350, https://doi.org/10.1145/800133.804350
17. Stuckey, P.: Negation and Constraint Logic Programming. Information and Computation **118**(1), 12–33 (1995). https://doi.org/https://doi.org/10.1006/inco.1995.1048, https://www.sciencedirect.com/science/article/pii/S0890540185710486
18. Tarau, P.: Abductive Reasoning in Intuitionistic Propositional Logic via Theorem Synthesis. Theory and Practice of Logic Programming **22**(5), 693–707 (2022). https://doi.org/10.1017/S1471068422000254
19. Tarau, P.: Full Automation of Goal-driven LLM Dialog Threads with And-Or Recursors and Refiner Oracles arXiv:2306.14077 (Jun 2023). https://doi.org/10.48550/arXiv.2306.14077
20. Tarau, P.: System Description: DeepLLM, Casting Dialog Threads into Logic Programs. In: Gibbons, J., Miller, D. (eds.) Proceedings of 17th International Symposium on Functional and Logic Programming (FLOPS 2024). Springer LNCS 14659 (2024)