Automatic Web Form Construction via Compilation of Paraconsistent Entailment to Relational Databases

Timothy L. Hinrichs
University of Chicago

Jui-Yi Kao*
Stanford University

Michael R. Genesereth
Stanford University

Abstract

This paper presents the core algorithm for a tool implementing a declarative approach to web form development. Instead of writing Javascript to implement error detection and implied value computation, web developers write an ontology in classical logic that describes the relationships between web form fields, and the Javascript is generated automatically. To meet performance demands, the compilation of classical logic into Javascript uses relational databases as an intermediary; and, to address the well-known explosiveness problem of logical implication in the presence of contradictions (web form errors), our algorithm employs a paraconsistent form of logical implication.

Introduction

Modern web forms collect data from users on the World Wide Web using a combination of HTML to display widgets (*e.g.*, textboxes and checkboxes) and browser scripts (*e.g.*, Javascript, Flash) to identify errors and compute the values implied by user-provided data. Writing and maintaining HTML is easy, but writing and maintaining browser scripts is hard.

Several issues complicate the construction and maintenance of browser scripts for error-detection and implied-value computation. First, web forms allow users to enter data in whichever order they please, thereby requiring that the scripts make no assumption about which widgets have data at which points in time. Second, many web forms only highlight errors instead of forcing the user to fix those errors immediately. Because implied values are propagated through the form, the scripts must carefully control how errors propagate. Third, giving users real-time feedback requires scripts that run fast enough not to be noticed by users. Finally, even small changes to a form (*e.g.*, adding a widget or changing an error condition) may require substantial changes in the scripts, thus making the task of web form maintenance similar in difficulty to the task of construction.

Here we introduce a declarative approach to web form development that addresses the problems above. Instead of writing browser scripts directly, web developers write logical sentences that capture the constraints on a web form and invoke a compiler that automatically generates relational database queries implementing error-detection and implied-value computation (which can then be compiled into any browser scripting language). Implementing error-detection requires implementing logical-inconsistency detection (between the web form data and the constraints). Implementing implied-value computation requires implementing a paraconsistent notion of logical entailment, or else a single web form error causes all values to be implied for all widgets. Thus, the database queries produced by the compiler implement inconsistency detection and paraconsistent entailment, and because they are fixed for the entire lifetime of the web form, they enjoy polynomial parameterized complexity, helping meet the real-time performance demands of the web form domain.

While motivated by a real-world application, the contributions of this paper are theoretical. After discussing an example, we present a mathematical formalization of the web form domain. Then we describe a logical representation of web forms including complexity results and introduce the compilation problem studied in this paper. Subsequently, we describe a simple and easy-to-implement algorithm for solving the compilation problem and analyze its complexity. Finally, we report on related work and conclude. Proofs and additional details can be found in the extended version of this paper (Anonymous 2009).

Example

On its website, Toyota allows customers to investigate the various options available for each of its cars using web forms. Figure 1 shows some of the options available for the 2009 Toyota Sequoia. A Sequoia's grade represents its base-level options and can be either SR5, Limited, or Platinum; its transmission can be either 5 or 6 speed; and the drive can either be 4x2 or 4x4.

Suppose a user first selects a 5-speed transmission and then chooses the Limited grade. Because the Limited grade implies the 4x4 drive, the drive widget is automatically filled with 4x4. Because the Limited grade also implies the 6 speed transmission, the grade and transmission widgets are highlighted to indicate an error.

*This work was supported in part by the National Science Foundation under award number IIS-0841152.
Copyright © 2010, Association for the Advancement of Artificial Intelligence (www.aaai.org). All rights reserved.

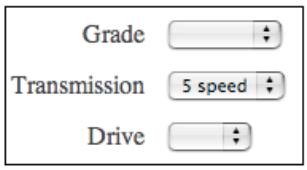

Figure 1: Example web form snippet

To construct this web form, a web developer would provide our compiler with the following logical constraints.

$(grade(limited) \vee grade(platinum)) \Rightarrow tran(6speed)$
$grade(limited) \Rightarrow drive(4x4)$

The compiler then constructs database queries for detecting errors and computing implied values for each widget. For example, the query for whether or not the grade widget is an error is given below by grade-err, and the query for the implied values of drive is given by drive-impl. Here we use DATALOG to represent the database queries.

```
grade-err     :-  not tran(6speed), grade(limited)
drive-impl(4x4) :-  grade(limited)
```

Web Form Formalization

The three crucial components of a web form are the widgets, the set of values that widgets can be assigned, and the ontology defining the set of permitted widget assignments. The definitions of interest in this paper, errors and implied values, are based on these building blocks.

Formally, W is a finite set of widgets, and U is a set of possible values. We use p, q, r for widgets and a, b, c for values. A *widget assignment* is a partial function $v : W \rightarrow 2^U$ mapping a widget to a set of values for some subset of all widgets[1]. $dom(v)$ is the set of widgets over which assignment v is defined. Assignment v_1 can be extended to v_2, written $v_1 \subseteq v_2$, if v_1's domain is a subset of v_2's domain and $v_1(x) = v_2(x)$ for all x in $dom(v_1)$. $v_1 \subset v_2$ means that $v_1 \subseteq v_2$ and there is an x in $dom(v_2)$ not in $dom(v_1)$.

An *ontology* O, defining the permissible web form completions, is a non-empty set of total widget assignments, *i.e.*, a non-empty set of widget assignments v such that $dom(v) = W$. An assignment v is *consistent* (with O) if there is a v' such that $v \subseteq v'$ and $v' \in O$; otherwise v is *inconsistent*. An assignment v is *minimally inconsistent* if v is inconsistent and there is no $v' \subset v$ that is also inconsistent. We say widget p is an *error* for assignment v if there is a minimally inconsistent $v' \subseteq v$ such that $p \in dom(v')$.

Having provided a definition for a web form error, we now provide a definition for an implied value. Value a is *positively implied* for widget p by consistent assignment v (written $v \Rightarrow p^+(a)$) if for every v' such that $v \subseteq v'$ and $v' \in O$, $a \in v'(p)$. Value a is *negatively implied* for widget p by consistent assignment v (written $v \Rightarrow p^-(a)$) if for every v' such that $v \subseteq v'$ and $v' \in O$, $a \notin v'(p)$. Note that these definitions apply only to consistent widget assignments because for inconsistent assignments, all values would be implied for all widgets.

For inconsistent assignments, appropriate definitions for implication are not so clear cut. We say value a is *positively implied* for widget p by assignment v if there is a $v' \subseteq v$ that is consistent and positively implies a according to the definition above. Similarly, value a is *negatively implied* for widget p by assignment v if there is a $v' \subseteq v$ that is consistent and negatively implies a according to the definition above. These definitions for implication reduce to the earlier ones for the case of consistent assignments but in the case of inconsistency do not always imply all values for all widgets.

[1]Widget assignments map a widget to a *set* of values to reflect the fact that some HTML widgets can be assigned multiple values.

Logical Representation

The mathematical objects defined in the previous section may be too large to conveniently write explicitly; in fact, the ontologies could be infinite. In this section we describe a logical representation for compactly describing web forms.

First, consider a logical representation of an ontology. Following (Kassoff and Genesereth 2007) we employ first-order logic with the following restrictions: monadic, function-free, equality-free, in the $\forall^*$ prefix. More precisely, the terms in the language are variables and object constants. Atomic sentences take the form $p(t)$ where p is a predicate and t is a term. Sentences are either atomic sentences or one of $\{\wedge, \vee, \neg, \Rightarrow, \Leftarrow, \Leftrightarrow\}$ applied to sentences in the usual way. Ground sentences contain no variables, and all variables in nonground sentences are implicitly universally quantified. We use MON to denote all such sentences. The semantics are standard (see (Enderton 2000) for details). The widgets of a web form correspond to the monadic predicates of the ontology description, and hence we call a consistent subset of MON a *logical ontology*. It is noteworthy that while an ontology from the previous section fixes the universe, a logical ontology allows for many different universes—a consequence of first-order model theory. While seemingly problematic, logical ontologies obviate the need for web developers to fix ahead of time the set of values that users are allowed to enter, and as we shall see results in a version of active domain semantics.

Second, consider a logical representation of a widget assignment. A *logical widget assignment* is consistent set of individual widget assignments. An *individual widget assignment* for widget p is a consistent conjunction of ground literals where for every object constant a either $p(a)$ or $\neg p(a)$ is included in the conjunction. This representation, while perhaps more complicated than one might expect, supports the upcoming result that our logical representation is faithful to the mathematical definition of web forms. In (Anonymous 2009), we illustrate why three obvious, simpler representations are inadequate. If Λ is a logical widget assignment, then Λ_P denotes the subset of Λ consisting of the individual widget assigments for the set of widgets P.

The Example section showed a web form and a logical ontology for that form. The widget assignment for Figure 1 follows.

$tran(5speed) \wedge \neg tran(6speed) \wedge$
$\neg tran(limited) \wedge \neg tran(sr5) \wedge \neg tran(platinum) \wedge$
$\neg tran(4x2) \wedge \neg tran(4x4)$

As one would hope, our logical representation implies a simple, logical definition for a web form error. That definition is based on inconsistency and utilizes the active domain semantics, *i.e.*, where the universe is the set of objects appearing in the logical ontology and widget assignment.

Proposition 1. *Suppose Λ represents widget assignment v, and Δ represents ontology O where the universe is $Objs[\Delta \cup \Lambda]$. Widget p is an error for v under O if and only if there is a minimal subset of Λ including p's individual widget assignment that is inconsistent with Δ.*

Third, consider a logical representation of widget assignment implication. It turns out that implication corresponds to a paraconsistent notion of entailment—in particular a variation of the well-known existential entailment relation. Formally, if Δ is a logical ontology and Λ is a logical widget assignment, we say that Λ strictly (existentially) entails ϕ with respect to Δ, written $\Lambda \models_E^\Delta \phi$, if and only if there is a subset Λ_0 of Λ such that $\Lambda_0 \cup \Delta$ is consistent and $\Lambda_0 \cup \Delta \models \phi$. Strict existential entailment corresponds in a natural and precise way to widget assignment implication.

Proposition 2. *Suppose Λ represents widget assignment v, and Δ represents ontology O where the universe is $Objs[\Delta \cup \Lambda] \cup \{a\}$. $v \Rightarrow p^{\pm}(a)$ under ontology O if and only if $\Lambda \models_E^\Delta \pm p(a)$*[2]*.*

Strict entailment is coNP-hard and is contained in PSPACE. Restricting the logical ontology to a single variable and no object constants while simultaneously forcing it to be written in clausal form fails to drop the complexity to P. Thus even with reasonably strong restrictions the optimal algorithm for strict entailment runs in exponential time.

Theorem 1 (Strict Entailment Complexity). *Suppose Δ is in* MON*, and Λ is a finite set of ground sentences. $\Lambda \models_E^\Delta p(a)$ is Π_2^{p}-hard and included in Σ_3^{p}. If the number of variables appearing in Δ is bounded by a constant, strict entailment is both Σ_1^{p} and Π_1^{p}-hard and is included in Σ_2^{p}. If Δ is in clausal form, contains a single variable, and includes no object constants, strict entailment is Π_1^{p}-hard.*

Web Form Compilation

Modern web forms detect errors (identify minimal inconsistencies) and calculate implied values (compute strict entailment) as fast as users fill in data. To generate such forms, we utilize an algorithm for constructing relational database queries that compute strict entailment. In the process that algorithm generates database manipulation statements that identify minimal inconsistencies. Hence, the remainder of the paper focuses on a compilation problem for strict entailment and includes a discussion of minimal inconsistency detection in due course.

The compilation problem consists of two transformations. The *ontology transformation*, denoted α, starts with an ontology and constructs a set of database queries that implement strict entailment assuming that they are evaluated over the appropriate database. The *data transformation* operation, denoted β, constructs the requisite database from a logical widget assignment and logical ontology. The ontology transformation operation is run once to construct the web form, whereas the data transformation operation is run on the browser each time the widget assignment changes.

We utilize the well-known equivalence of evaluating database queries on a database and evaluating first-order formulae on an interpretation.

Definition 1 (Web Form Constraint Compiler). *A web form constraint compiler is a pair of functions $\langle \alpha, \beta \rangle$ where α maps a logical ontology to a set of first-order formulae and β maps a logical widget assignment and a logical ontology to a first-order interpretation. $\langle \alpha, \beta \rangle$ is a compiler if for any finite logical ontology Δ and predicate p, there are sentences $\phi_p^+(x)$ and $\phi_p^-(x)$ in $\alpha[\Delta]$ such that for any finite logical widget assignment Λ and any object constant a,*

$$\Lambda \models_E^\Delta \pm p(a) \text{ if and only if } \models_{\beta[\Lambda,\Delta]} \phi_p^{\pm}(a)$$

Algorithms

Ontology Transformation To transform a given ontology, our algorithm follows a four-step process: compute the resolution closure, compute the contrapositives of each clause in the closure, augment each contrapositive with a consistency check, and invoke predicate completion.

For example, consider a simple logical ontology.

$$\forall x.(p(x) \vee q(x)) \wedge (\neg q(x) \vee \neg r(x))$$

The resolution closure adds a single clause: $p(x) \vee \neg r(x)$. Computing the contrapositives and appending consistency checks is straightforward and produces the following rules. (Here we show just the contrapositives with positive heads.)

$$\begin{aligned} p(x) &\Leftarrow \neg q(x) \wedge consistent_{\{q\}} \\ q(x) &\Leftarrow \neg p(x) \wedge consistent_{\{p\}} \\ r(x) &\Leftarrow p(x) \wedge consistent_{\{p\}} \\ p(x) &\Leftarrow r(x) \wedge consistent_{\{r\}} \end{aligned}$$

The consistency checks ensure that witnesses for entailment are consistent with the entire ontology and are discussed below. Predicate completion then constructs the first-order formula defining strict entailment for each predicate ρ: the disjunction of all the rules with ρ in the head.

$$\begin{aligned} \phi_p^+(x) &\equiv (\neg q(x) \wedge consistent_{\{q\}}) \vee \\ &\quad\ (r(x) \wedge consistent_{\{r\}}) \\ \phi_q^+(x) &\equiv \neg p(x) \wedge consistent_{\{p\}} \\ \phi_r^+(x) &\equiv p(x) \wedge consistent_{\{p\}} \end{aligned}$$

The definitions for ϕ_p^-, ϕ_q^-, and ϕ_r^- are constructed similarly from the contrapositives with negative heads. Algorithm 1, named ONTCOMPILE, formalizes the algorithm outlined here.

In ONTCOMPILE, RES[Δ] denotes the resolution and factoring closure of the clausal form of Δ, and PREDS[$\phi(\bar{x})$] denotes the predicates occurring in $\phi(\bar{x})$. The semantics for $consistent^{\Delta}_{\text{PREDS}[\phi(\bar{x},\bar{y})]}$ follows.

[2]Here we use $+p(a)$ to mean $p(a)$ and $-p(a)$ to mean $\neg p(a)$. The sign of $\pm p(a)$ is the same as the sign of $p^{\pm}(a)$.

Algorithm 1 ONTCOMPILE[Δ]

Assumes: Δ is a clause set.
Outputs: A set of first-order equivalences.

1: Δ := RES[Δ]
2: $\Gamma_p^s := \emptyset$ for all predicates p and $s \in \{+, -\}$
3: **for all** contrapositives d in $\bigcup_i \{p_i(x) \vee \neg p_i(x)\} \cup \Delta$ **do**
4: write d as $\pm p(x) \Leftarrow \phi(x, \bar{y})$
5: $\Gamma_p^{\pm} := \{\exists \bar{y}.\phi(x, \bar{y}) \wedge consistent^{\Delta}_{\text{PREDS}[\phi(x,\bar{y})]}\} \cup \Gamma_p^{\pm}$
6: **end for**
7: **print** $\phi_p^s \equiv \bigvee \Gamma_p^s$ for all predicates p and $s \in \{+, -\}$

Definition 2 ($consistent_P^{\Theta}$). *Suppose Λ is a logical widget assignment. For the sentence set Θ and predicate set P, $consistent_P^{\Theta}$ is true (with respect to Λ) if and only if $\Lambda_P \cup \Theta$ is consistent.*

The database queries produced by ONTCOMPILE are evaluated over the database constructed by the data transformation operation described in the next section. Given a widget assignment Λ and an ontology Δ, our data transformation operation, denoted β, performs two tasks: (1) representing Λ as a first-order interpretation and (2) extending that interpretation to include definitions for the necessary consistency checks. Before discussing β in detail, we analyze the complexity of ONTCOMPILE as well as the complexity of its output. The first result ensures that ONTCOMPILE paired with β is sound and complete.

Theorem 2 (Soundness and Completeness). *The pair $\langle$ONTCOMPILE$, \beta\rangle$ is a web form constraint compiler for* MON *ontologies.*

The complexity of ONTCOMPILE is polynomial in the size of the clause set produced by converting to clausal form and computing the resolution closure.

Proposition 3 (Resolution Complexity). *The output complexity for resolution for* MON *premises is* EXPSPACE*-hard and included in* 2EXPSPACE*. When the premises are in clausal form, contain one variable, and include no object constants, the output complexity is* EXPSPACE*-complete.*

More important than the performance of the compiler is the performance of the database queries that compiler produces for implementing strict entailment. When the ontology starts in clausal form, the cost of computing strict entailment is bounded above by doubly exponential time, but it is unknown if this bound is tight; thus, it is unknown whether or not our implementation of strict entailment is optimal in the general case. But for a special case, we have shown the computation of strict entailment to be singly exponential and therefore optimal.

Theorem 3 (Optimality). *When the logical ontology is in clausal form, contains one variable, and includes no object constants, strict entailment computation runs in singly exponential time, an optimal algorithm assuming* $\mathrm{P} \neq \mathrm{coNP}$.

Additionally, because compilation builds the ontology into the procedure for computing strict entailment, the only change from query to query is the widget assignment; thus, the ontology is fixed and the parameterized complexity of strict entailment computation is polynomial in the input.

Proposition 4 (Polynomial Parameterized Complexity). *The parameterized complexity of computing strict entailment is polynomial because the ontology is fixed by compilation.*

Data Transformation Recall that our data transformation operation β performs two tasks when given a widget assignment Λ and an ontology Δ: represent Λ as a first-order interpretation, and extend that interpretation to include definitions for the necessary $consistent_P^{\Delta}$. Our implementation of β proceeds in three steps. The first step, denoted β_1, builds an interpretation $\mathcal{I}$ that maps each predicate p occurring in Λ to the set of values to which p is assigned, *i.e.*, $\mathcal{I}(p) = \{a \mid p(a)$ is a positive conjunct occurring in $\Lambda\}$. The second step, denoted β_2, extends the result of step one to include definitions for $inconsistent_P^{\Delta}$ (whose semantics is exactly the opposite of $consistent_P^{\Delta}$). The third step, denoted β_3, extends the result of step two to include definitions for the necessary $consistent_P^{\Delta}$. It is noteworthy that only the second step, β_2, requires information about Δ, which means that our data transformation operation takes the form $\beta_3[\beta_2[\beta_1[\Lambda], \Delta]]$.

The first step is straightforward as illustrated above, but the next two steps, which jointly construct the needed $consistent_P^{\Delta}$ definitions, is not as straightforward and occupy the remainder of this section. This two-step approach to defining $consistent_P^{\Delta}$ has two benefits. First, by including interpretations for $inconsistent_P^{\Delta}$, it is straightforward to identify errors on the web form. A widget w is an error exactly when there is a minimal P including w such that $inconsistent_P^{\Delta}$ is true. Thus, each time a user changes the web form, errors can be identified once β_2 has been executed. Second, the presence of $inconsistent_P^{\Delta}$ definitions makes the construction of $consistent_P^{\Delta}$ definitions independent of Δ. Thus, of the three steps, β_2 (where the $inconsistent_P^{\Delta}$ definitions are constructed) is the only one that requires Δ. Moreover as shown below, it is possible to employ compilation to construct a β_2 specialized to any given Δ, thereby avoiding the costs of interpreting Δ during execution. We denote such a β_2 as β_2^{Δ}; thus, our data transformation operation actually takes the form $\beta_3[\beta_2^{\Delta}[\beta_1[\Lambda]]]$.

As mentioned above, implementing the β_3 operation is simple since the result of $\beta_2^{\Delta}[\beta_1[\Lambda]]$ includes truth values for $inconsistent_Q^{\Delta}$ for at least all those Q that are minimally inconsistent.

$$\beta_3[\mathcal{I}] = \{consistent_P^{\Delta} \rightarrow true \mid \text{there is no } Q \text{ such that } Q \subseteq P \text{ and } \models_{\mathcal{I}} inconsistent_Q^{\Delta}\} \cup \mathcal{I}$$

More complicated is the compilation algorithm that constructs β_2^{Δ} from a given ontology Δ. Algorithm 2, named DATACOMPILE, includes the core algorithm for compilation and bears strong similarities to our ontology compilation algorithm (ONTCOMPILE found in Algorithm 1). DATACOMPILE constructs a sentence σ_P that implements an inconsistency check for certain sets of predicates P. The definition for β_2^{Δ}, shown below, defines the truth value for

$inconsistent_P^\Delta$ as the truth value of σ_P evaluated over the current widget assignments.

$$\beta_2[\mathcal{I}] = \{inconsistent_P \rightarrow true \mid \models_{\mathcal{I}} \sigma_P\} \cup \mathcal{I}$$

Algorithm 2 DATACOMPILE[Δ]

Assumes: Δ is a clause set.
Outputs: A set of first-order equivalences.
1: $\Gamma_P := \emptyset$ for all predicate sets P
2: **for all** clauses c in RES[Δ] **do**
3: $\quad \Gamma_{\text{PREDS}[c]} := \{\exists^*.\neg c\} \cup \Gamma_{\text{PREDS}[c]}$
4: **end for**
5: **print** $\sigma_P \equiv \bigvee \Gamma_P$ for all predicate sets P

DATACOMPILE is sound and complete, and its complexity is polynomial in the size of the resolution closure. The data transformation operation $\beta_3[\beta_2^\Delta[\beta_1[\Lambda]]]$ is also sound and complete.

Theorem 4 (Soundness and Completeness of DATACOMPILE). *Suppose Δ is a logical ontology, Λ is a logical widget assignment, and P is a predicate set. $\Lambda_P \cup \Delta$ is inconsistent if and only if there is some $Q \subseteq P$ such that $\sigma_Q \in$ DATACOMPILE[Δ] and $\models_{\beta_1[\Lambda]} \sigma_Q$.*

Theorem 5 (Soundness and Completeness of β). *Suppose Δ is a logical ontology, and Λ is a logical widget assignment. If $consistent_P^\Delta$ occurs in* ONTCOMPILE[Δ] *then*

$$\models_{\beta_3[\beta_2^\Delta[\beta_1[\Lambda]]]} consistent_P^\Delta \textit{ iff } \Delta \cup \Lambda_P \textit{ is consistent}$$

Related Work

Related work touches on three topics: web application frameworks, inconsistency tolerance for classical logic, and knowledge compilation.

Commercial web application frameworks, designed to alleviate some of the more tedious aspects of web development (*e.g.*, Ruby On Rails, PhP), allow developers to declaratively specify relatively simple constraints on individual widgets that are then compiled into client-side error-detection code. For example, a textbox may be specified to only accept numbers, and frameworks will automatically construct client-side code that checks that type constraint. In contrast, frameworks today do not allow developers to declaratively specify constraints on more than one widget. For example, a web form soliciting a user's city and state would ideally inform the user when the specified city does not exist in the specified state, but such constraints can only be implemented imperatively in today's frameworks. Thus, the work reported here generalizes the class of declarative constraints handled by today's web application frameworks.

Inconsistency tolerance for classical logic has received significant attention over the last decade (see (Besnard and Hunter 2008)). In contrast to some of that work, the problem addressed in this paper involves detecting but not repairing inconsistencies, *e.g.*, (Everaere, Konieczny, and Marquis 2008; Benferhat, Lagrue, and Rossit 2007; Subrahmanian and Amgoud 2007). Second, our work focuses on a fragment of first-order logic (Besnard and Hunter 2005) that is not propositional, *e.g.*, (Efstathiou and Hunter 2008), yet retains decidability. Third, instead of establishing the relationships between all possible arguments with an argument tree, *e.g.*, (Efstathiou and Hunter 2008; Besnard and Hunter 2008), we find two arguments for each conclusion: one supporting and one undermining.

For knowledge compilation, our work is differentiated from most because it addresses inconsistency tolerance. The other efforts we are aware of addressing both inconsistency and compilation (Flouris et al. 2006; Huang, van Harmelen, and ten Teije 2005; Gomez, Chesnevar, and Simari 2008; Hinrichs, Kao, and Genesereth 2009) fail to address real-time performance demands or fail to capitalize on the properties of the web form domain. Ignoring inconsistency tolerance, our work differs from the existing knowledge compilation work because we utilize a logical language that is not propositional logic, *e.g.*, (Darwiche and Marquis 2002; Selman and Kautz 1996; Cadoli and Mancini 2002; Besnard and Hunter 2006) yet is incomparable to similar work in the description logic community (because we allow non-Horn sentences) (Nagy, Lukacsy, and Szeredi 2006; Calvanese et al. 2008; 2007; Lutz, Toman, and Wolter 2008; 2009; Kontchakov et al. 2009).

Despite minor technical differences, the line of research on description logic knowledge compilation is especially pertinent because it involves the conversion of classical logic into relational databases as a way of implementing description logic reasoners capable of handling large ABoxes. In their terminology, the web form's constraints correspond to a TBox, the web form data corresponds to an ABox (though we include negative literals), and we only consider (positive and negative) instance queries. Our algorithms infuse the TBox into all possible instance queries at compilation-time, and our algorithms infuse the TBox into the ABox each time the web form data changes. In short, our work is a form of combined FO-rewriting, as opposed to [query] FO-rewriting (Kontchakov et al. 2009).

Conclusion and Future Work

This paper introduces the web form compilation problem and algorithms for solving it. Those algorithms construct relational database queries and data manipulation statements that implement error-detection and implied-value computation. Because the result of compilation is fixed for the lifetime of the web form, the parameterized complexity of detecting errors and computing implied values is polynomial, a near necessity for achieving the required real-time performance demands. Additionally, we identified a conditions under which our algorithms are optimal for non-parameterized complexity and leave (sub-) optimality results for the general case to future work. Our aggressive compilation techniques are appropriate for the web form domain because the compiler runs once to produce a result that may be used thousands or millions of times. We have prototyped our approach and given it the name PLATO, available at `http://tlh.cs.uchicago.edu:5000/plato/`.

Our long-term goal is to provide web developers with a practical tool for building and maintaining web forms; thus, we have begun our investigation with a simple monadic,

first-order language for expressing constraints with strong compilation properties. In the future we plan to extend this language, using the compilation approaches developed in the description logic community as a baseline, adapting them to paraconsistent entailment. We will also explore languages that can be compiled into our monadic language.

Additionally, we plan to investigate how to automatically extract web form constraints from back-end database integrity constraints, and how to automatically employ AJAX to check web form constraints that rely on the data contained within those databases. We also plan to integrate our approach into existing web application frameworks, *e.g.*, Ruby on Rails, thereby giving developers the ability to employ both declarative and imperative styles of development. Such an integration would allow entire web applications to be built using declaratively-specified web forms. We expect that web applications built within such a framework would have improved security because both server-side and client-side error-detection could be generated automatically, thereby eliminating parameter tampering exploits algorithmically, one of the most prominent vulnerabilities on the web today.

Acknowledgements

Thanks to Stuart Kurtz, Matthias Baaz, Eyal Amir, and Alexander Razborov for discussions about the complexity results in this paper.

References

Anonymous. 2009. Automatic web form generation via compilation of paraconsistent entailment to relational databases. Technical report, University X.

Benferhat, S.; Lagrue, S.; and Rossit, J. 2007. An egalitarist fusion of incommensurable ranked belief bases under constraints. In *Proceedings of the AAAI Conference on Artificial Intelligence*, 367–372.

Besnard, P., and Hunter, A. 2005. Practical first-order argumentation. In *Proceedings of the AAAI Conference on Artificial Intelligence*, 590–595.

Besnard, P., and Hunter, A. 2006. Knowledgebase compilation for efficient logical argumentation. In *Proceedings of the International Conference on Principles of Knowledge Representation and Reasoning*, 123–133.

Besnard, P., and Hunter, A. 2008. *Elements of Argumentation*. MIT Press.

Cadoli, M., and Mancini, T. 2002. Knowledge compilation = query rewriting + view synthesis. In *Proceedings of the ACM Symposium on Principles of Database Systems*, 199–208.

Calvanese, D.; Giacomo, G. D.; Lembo, D.; Lenzerini, M.; and Rosati, R. 2007. Eql-lite: Effective first-order query processing in description logics. In *Proceedings of the International Joint Conference on Artificial Intelligence*, 274–279.

Calvanese, D.; Giacomo, G. D.; Lembo, D.; Lenzerini, M.; Poggi, A.; Rosati, R.; and Ruzzi, M. 2008. Data integration through dl-litea ontologies. In *Proceedings of the International Workshop on Semantics in Data and Knowledge Bases*.

Darwiche, A., and Marquis, P. 2002. A knowledge compilation map. *Journal of Artificial Intelligence Research* 17:229–264.

Efstathiou, V., and Hunter, A. 2008. Algorithms for effective argumentation of classical propositional logic. In *Proceedings of the Symposium on Foundations of Information and Knowledge Systems*.

Enderton, H. 2000. *A Mathematical Introduction to Logic*. Academic Press.

Everaere, P.; Konieczny, S.; and Marquis, P. 2008. Conflict-based merging operators. In *Proceedings of the International Conference on Principles of Knowledge Representation and Reasoning*, 348–357.

Flouris, G.; Huang, Z.; Pan, J. Z.; Plexousakis, D.; and Wache, H. 2006. Inconsistencies, negations and changes in ontologies. In *Proceedings of the AAAI Conference on Artificial Intelligence*, 1295–1300.

Gomez, S. A.; Chesnevar, C. I.; and Simari, G. R. 2008. An argumentative approach to reasoning with inconsistent ontologies. In *Proceedings of the KR Workshop on Knowledge Representation and Ontologies*, 11–20.

Hinrichs, T. L.; Kao, J. Y.; and Genesereth, M. R. 2009. Inconsistency-tolerant reasoning with classical logic and large databases. In *Proceedings of the Symposium of Abstraction, Reformulation, and Approximation*.

Huang, Z.; van Harmelen, F.; and ten Teije, A. 2005. Reasoning with inconsistent ontologies. In *Proceedings of the International Joint Conference on Artificial Intelligence*.

Kassoff, M., and Genesereth, M. R. 2007. PrediCalc: A logical spreadsheet management system. *Knowledge Engineering Review* 22(3):281–295.

Kontchakov, R.; Lutz, C.; Toman, D.; Wolter, F.; and Zakharyaschev, M. 2009. Combined fo rewritability for conjunctive query answering in dl-lite. In *Proceedings of the International Workshop on Description Logic*.

Lutz, C.; Toman, D.; and Wolter, F. 2008. Conjunctive query answering in el using a database system. In *Proceedings of the OWL: Experiences and Directions*.

Lutz, C.; Toman, D.; and Wolter, F. 2009. Conjunctive query answering in the description logic el using a relational database system. In *Proceedings of the International Joint Conference on Artificial Intelligence*, 2070–2075.

Nagy, Z.; Lukacsy, G.; and Szeredi, P. 2006. Translating description logic queries to prolog. In *Proceedings of the Symposium on Practical Aspects of Declarative Languages*, 168–182.

Selman, B., and Kautz, H. 1996. Knowledge compilation and theory approximation. *Journal of the ACM* 43(2):193–224.

Subrahmanian, V. S., and Amgoud, L. 2007. A general framework for reasoning about inconsistency. In *Proceedings of the International Joint Conference on Artificial Intelligence*, 599–604.

Appendix

Web Form Formalization

There is an alternative formalization for the web form domain that requires one simple change to the definitions presented earlier. The change concerns the definition of a widget assignment's extension. Originally, assignment v_1 could be extended to v_2, written $v_1 \subseteq v_2$, if v_1's domain were a subset of v_2's domain and $v_1(x) = v_2(x)$ for all x in $dom(v_1)$. This assumes that extension is defined at the level of widgets, but alternatively we can define extension in terms of widget values. Assignment v_1 can be extended to v_2, written $v_1 \subseteq' v_2$, if v_1's domain is a subset of v_2's domain and if $b \in v_1(p)$ then $b \in v_2(p)$ for all b. The original definition was chosen to ensure that our logical representation was faithful. (See the next subsection for more details on that.) As evidenced by the following proposition, it is arguable, however, that the alternative definition is more natural.

Proposition 5. *Suppose we define widget assignment extension using $\subseteq'$. Widget p is an error for assignment v if and only if there is a value a such that $a \in v(p)$ and $v \Rightarrow p^-(a)$.*

Proof. ($\Rightarrow$) Suppose p is an error for v. Then there is a $v' \subseteq v$ that is minimally inconsistent such that $p \in dom(v')$. Because v' is minimal, reducing it to $v'' \subset v'$ results in a consistent assignment. In particular, let v'' be identical to v' except that for some $a \in v'(p)$, $a \notin v''(p)$. Because v' is inconsistent but v'' is consistent, none of the extensions of v'' in O can include a in p's assignment; thus, $v'' \Rightarrow p^-(a)$. Since $v'' \subseteq v' \subseteq v$ and v'' is consistent, by definition $v \Rightarrow p^-(a)$.

($\Leftarrow$) Suppose there is an a such that $v \Rightarrow p^-(a)$. Then there is a consistent $v' \subseteq v$ such that $v' \Rightarrow p^-(a)$. Consider then the assignment v'' identical to v' except that a is added to $v''(p)$. Notice then that v'' is inconsistent, and if it is minimally inconsistent, then p is an error by definition. Otherwise, v'' is inconsistent but not minimal, meaning that something must be removed to achieve minimality. But because v' is consistent and v'' is inconsistent, a cannot be removed from $v''(p)$ to achieve inconsistency; thus, there must be a $v''' \subset v''$ where $a \in v'''(p)$ that is minimally inconsistent, making p an error by definition. □

Logical Representation

We use a fragment of first-order logic (function-free, quantifier-free, equality in obeyance with the unique names assumption) to represent each ontology (a set of functions from widgets W to sets of values from a universe U). Moreover, we use logic to represent each widget assignment (a partial function from W to a set of values from U). We would like it to be the case that our representation is faithful, *i.e.*, that the logic problems addressed in the body of the paper solve the real mathematical versions of those problems. Below we define precisely the relationships between ontologies/logical ontologies and widget assignments/logical widget assignments, and then demonstrate that our representations are faithful.

We begin by discussing Herbrand interpretations, which for the quantifier-free fragment of first-order logic obeying the unique names assumption are sufficient for determining entailment. Often the universe of the Herbrand interpretations of a sentence set Δ is exactly the ground terms of Δ, but because here Δ is to be combined with a logical widget assignment Λ, we understand there to be many different universes for Herbrand interpretations of Δ, all of which include all ground terms from Δ. Formally, a Herbrand interpretation for a function-free language $\mathcal{L}$ is a two-tuple $\langle U, \mathcal{I} \rangle$ where U is a finite set of object constants including all those in $\mathcal{L}$ and $\mathcal{I}$ is a function that maps a predicate of arity n from $\mathcal{L}$ to a subset of U^n.

Herbrand models can be viewed as representations of total widget assignments. The Herbrand model M represents the widget assignment comprised of exactly those assignments $p \rightarrow \{a \mid \models_M p(a)\}$ where p is defined in M. We denote this mapping from M to a widget assignment as $Rep[M]$. The ontology described by Δ for universe U (denoted $Rep_U[\Delta]$) is the set of all $Rep[M]$ where M is a Herbrand model of Δ with universe U. Similarly, we use $Rep[\Lambda]$ to denote the widget assignment described by Λ: the set of all $p \rightarrow \{a \mid p(a)$ occurs in $\Lambda\}$.

Notice there is a one-to-one correspondence between logical widget assignments and widget assignments (and between Herbrand models and total widget assignments). We use Rep^{-1} to denote the inverse of Rep for these cases. Also notice that every subset of a logical widget assignment is itself a logical widget assignment, a fact used in the proofs below. Finally, notice also that there is a many-to-many relationship between logical ontologies and ontologies, but that given a logical ontology and a universe, there is a unique corresponding ontology. The following lemma exploits these relationships to provide alternative ways for computing the semantics of widget assignments and logical widget assignments.

Lemma 1. *Consider just the class of Herbrand interpretations. Suppose v is a widget assignment, Δ is a logical ontology, and U is a universe.*

$$\{v' \mid v \subseteq v' \text{ and } v' \in Rep_U[\Delta]\}$$
$$= \{Rep[M] \mid \models_M \Delta \cup Rep^{-1}[v] \text{ with universe } U\}$$

Suppose Λ is a logical widget assignment, Δ is a logical ontology, and U is a universe.

$$\{M \mid \models_M \Delta \cup \Lambda \text{ and } M \text{ has universe } U\}$$
$$= \{Rep^{-1}[v'] \mid Rep[\Lambda] \subseteq v' \text{ and } v' \in Rep_U[\Delta]\}$$

Proof. For part one, consider any M with universe U satisfying $\Delta \cup Rep^{-1}[v]$. Since M satisfies Δ, $Rep[M]$ belongs to $Rep_U[\Delta]$; thus, we need only show that $v \subseteq Rep[M]$. M was chosen to satisfy $Rep^{-1}[v]$, and hence if $b \in v(p)$, then $p(b) \in Rep^{-1}[v]$ and $\models_M p(b)$. Then by definition $b \in Rep[M](p)$. Since the argument holds for an arbitrary p and b, it holds for all p and b; thus, $v \subseteq Rep[M]$.

Now consider any v' such that $v \subseteq v'$ and $v' \in Rep_U[\Delta]$. Since $v' \in Rep_U[\Delta]$, $Rep^{-1}[v']$ is a Herbrand interpretation with universe U satisfying Δ. Since $v \subseteq v'$, $b \in v(p)$ implies $b \in v'(p)$, and consequently that $p(b) \in Rep^{-1}[v]$ implies $p(b) \in Rep^{-1}[v']$. Hence, the interpretation $Rep^{-1}[v']$

satisfies $Rep^{-1}[v]$. Since it satisfies Δ and $Rep^{-1}[v]$ individually, it satisfies their union.

For part two, use v to denote $Rep[\Lambda]$. Then we can write part 1 as

$$\{Rep[M] \mid \models_M \Delta \cup \Lambda \text{ with universe } U]\}$$
$$= \{v' \mid Rep[\Lambda] \subseteq v' \text{ and } v' \in Rep_U[\Delta]\}$$

We can obviously apply Rep^{-1} to both sides of the equality (specifically to $Rep[M]$ and v'), to produce the equation we desire. Hence, part two is implied by part one. □

Our representation is faithful under the active domain semantics, *i.e.*, when the universe under consideration at any point in time is the set of object constants occurring in $\Delta \cup \Lambda$, denoted $Objs[\Delta \cup \Lambda]$.

Proposition 6. *Suppose Δ is a logical ontology and Λ is a logical widget assignment. Widget p is an error for widget assignment $Rep[\Lambda]$ under ontology $Rep_{Objs[\Delta\cup\Lambda]}[\Delta]$ if and only if there is a minimal subset of Λ including p's individual widget assignment that is inconsistent with Δ.*

Proof. ($\Leftarrow$) Suppose there is a minimal $\Lambda_0 \subseteq \Lambda$ such that Λ_0 is inconsistent with Δ. Because any subset of a logical widget assignment is a logical widget assignment, Λ_0 represents some logical widget assignment v. Also note that $Rep[\Lambda_0] \subseteq Rep[\Lambda]$. Since there is no M satisfying $\Delta \cup \Lambda_0$, there is certainly no satisfying Herbrand interpretation. Consequently, there is no total widget assignment $v' \supseteq v$ belonging to any of the ontologies described by Δ; hence, v is inconsistent (under $Rep_{Objs[\Delta\cup\Lambda]}[\Delta]$ in particular). Moreover, since Λ_0 is minimal, v must also be minimally inconsistent. Finally, since p's individual widget assignment is included in Λ_0, $p \in dom(v)$, ensuring that p is an error for $v = Rep[\Lambda]$ and therefore $Rep[\Lambda]$.

($\Rightarrow$) Suppose p is an error for $Rep[\Lambda]$ under ontology $Rep_{Objs[\Delta\cup\Lambda]}[\Delta]$. Then there is a minimally inconsistent $v \subseteq Rep[\Lambda]$ where $p \in dom(v)$. By construction $Rep^{-1}[v] = \Lambda_0$ consists of the individual widget assignments for all widgets $q \in dom(v)$. Since $p \in dom(v)$, p's individual widget assignment belongs to Λ_0. Moreover, $\Lambda_0 \subseteq \Lambda$ again by the definition of Rep. Since v is minimally inconsistent under $O = Rep_{Objs[\Delta\cup\Lambda]}[\Delta]$, there is no total widget assignment $v' \supseteq v$ included in O. There is therefore no Herbrand interpretation over universe $Objs[\Delta \cup \Lambda]$ satisfying $\Lambda_0 \cup \Delta$. Herbrand's theorem applies, ensuring that there is no interpretation whatsoever satisfying $\Lambda_0 \cup \Delta$. (To expand the universe to include objects from Λ not in Λ_0, add tautologies that include those objects.) Thus Λ_0 is inconsistent with Δ and includes p's individual widget assignment. Further, since v is minimal, every $\Lambda_1 \subset \Lambda$ is consistent with Δ; hence Λ_0 is minimal. □

Proposition 7. *Let Δ be a logical ontology and Λ be a logical widget assignment. $\Lambda \models_E^{\Delta} [\neg]p(a)$ if and only if $Rep[\Lambda] \Rightarrow p^{\pm}(a)$ under ontology $Rep_U[\Delta]$ for $U = Objs[\Delta \cup \Lambda] \cup \{a\}$.*

Proof. ($\Rightarrow$) Suppose $\Lambda \models_E^{\Delta} p(a)$. Then there is a $\Lambda_0 \subseteq \Lambda$ that together with Δ is consistent and entails $p(a)$. Thus all interpretations that satisfy $\Delta \cup \Lambda_0$ also satisfy $p(a)$. Because all subsets of our logical widget assignment language are themselves widget assignments, Λ_0 represents some widget assignment. Use v to denote $Rep[\Lambda_0]$, and U to denote $Objs[\Delta \cup \Lambda_0] \cup \{a\}$. Notice that it is safe to assume U includes all objects in Λ since we can trivially add tautologies to Δ that include all objects in Λ without changing Δ's semantics. First we show that v is consistent with respect to $Rep_U[\Delta]$ and that $v \subseteq Rep[\Lambda]$. Second we show for all $v' \supseteq v$ where $v' \in Rep_U[\Delta]$ that $a \in v'(p)$. This suffices to conclude that $Rep[\Lambda] \Rightarrow p^+(a)$ under ontology $Rep_U[\Delta]$. The negative case follows similarly.

(1) v is consistent with respect to $Rep_U[\Delta]$ if and only if there is a $v' \supseteq v$ where $v' \in Rep_U[\Delta]$. By the lemma, the set of all such v' is $\{Rep[M] \mid \models_M \Delta \cup Rep^{-1}[v] \text{ with universe } U\}$. Since Λ_0 is $Rep^{-1}[v]$ and $\Delta \cup \Lambda_0$ is consistent, $\Delta \cup Rep^{-1}[v]$ is also consistent; moreover, because it is quantifier-free and obeys the UNA, by Herbrand's theorem, $\Delta \cup Rep^{-1}[v]$ is satisfied by at least one Herbrand model where the universe is U. Thus there is at least one v' meeting the constraints above and hence v is consistent. Further, since $\Lambda_0 \subseteq \Lambda$ and v is $Rep[\Lambda_0]$, for every $b \in v(p)$ corresponding to $p(b) \in \Lambda_0$, we know $p(b) \in \Lambda$, and hence $b \in Rep[\Lambda](p)$. Thus by definition, $v \subseteq Rep[\Lambda]$.

(2) Consider any v' in the set $\{Rep[M] \mid \models_M \Delta \cup Rep^{-1}[v] \text{ with universe } U\}$. Notice that since $Rep^{-1}[v]$ is Λ_0, and $\Delta \cup \Lambda_0 \models p(a)$, we know that $Rep^{-1}[v']$ satisfies $p(a)$. By definition $a \in v'(p)$ if and only if $Rep^{-1}[v']$ satisfies $p(a)$. Thus, $a \in v'(p)$ holds for an arbitrary and therefore all v'.

($\Leftarrow$) Suppose that $Rep[\Lambda] \Rightarrow p^+(a)$ under ontology $Rep_U[\Delta]$ for $U = Objs[\Delta \cup \Lambda] \cup \{a\}$. Then there is a consistent $v \subseteq Rep[\Lambda]$ such that for all v' where $v \subseteq v'$ and $v' \in Rep_U[\Delta]$, we know $a \in v'(p)$. Use Λ_0 to denote $Rep^{-1}[v]$.

First we show that Λ_0 is consistent with Δ and that $\Lambda_0 \subseteq \Lambda$. Then we show that every interpretation that satisfies $\Lambda_0 \cup \Delta$ also satisfies $p(a)$. This suffices to ensure that Λ_0 is the witness for $\Lambda \models_E^{\Delta} p(a)$.

(1) By the lemma above the models satisfying $\Delta \cup \Lambda_0$ with universe U is the set $\{Rep^{-1}[v'] \mid v \subseteq v' \text{ and } v' \in Rep_U[\Delta]\}$. Since v is consistent, there is at least one such v', and hence the set of such models is nonempty, making $\Delta \cup \Lambda_0$ consistent. Furthermore, since $v \subseteq Rep[\Lambda]$, if $b \in v(p)$ then $b \in Rep[\Lambda](p)$ for every p and b. Consequently, we see for all p and b, if $p(b) \in \Lambda_0$ then $p(b) \in \Lambda$ and hence $\Lambda_0 \subseteq \Lambda$.

(2) By the lemma above, the Herbrand interpretations satisfying $\Delta \cup \Lambda_0$ with universe U is the set $\{Rep^{-1}[v'] \mid v \subseteq v' \text{ and } v' \in Rep_U[\Delta]\}$. Consider any such v'. Since $a \in v'(p)$, we know $\models_{Rep^{-1}[v']} p(a)$, and since the argument holds for an arbitrary v', it holds for all v' and hence all Herbrand interpretations with universe U satisfying $\Delta \cup \Lambda_0$ also satisfy $p(a)$. By Herbrand's theorem, the same holds of all interpretations. □

Recall that a logical widget assignment is a set of complete, consistent ground conjunctions. As mentioned pre-

viously, this representation has two properties used in the proofs above. First there is a one-to-one correspondence between logical widget assignments and widget assignments. Second, every subset of a logical widget assignment is itself a widget assignment. Here we explain why three simpler and more obvious representations fail to achieve these properties.

First, suppose we represent a widget assignment v as the set of all $p(a)$ such that $a \in v(p)$. This representation satisfies the subset property but not the one-to-one correspondence property. To see why, we demonstrate two widget assignments that differ but that cannot be represented differently in this language: $v_1(p) = \emptyset$ and $p \notin dom(v_2)$. Both of these assignments include no a such that $a \in v(p)$; thus, they cannot be differentiated by the representation language.

Second, suppose we represent a widget assignment as a complete, consistent set of literals, *i.e.*, we use just the set of literals occurring in the actual representation. This representation has a one-to-one correspondence but does not obey the subset requirement. To see why, just notice that the subset $\{p(a)\}$ of $\{p(a), \neg p(b)\}$ is not complete, and hence is not itself a logical widget assignment.

Third, suppose we represent a widget assignment as a set of conjunctions of ground atoms, *i.e.*, take our representation and drop out all the negative literals. This representation satisfies the subset property but not the one-to-one property. Consider the same two widget assignments: $v_1(p) = \emptyset$ and $p \notin dom(v_2)$. Again, there is no way to differentiate them because in both cases the conjunction for p would be empty.

While lacking the one-to-one property points to a basic inadequacy of representation, it is less clear why the subset property is crucial for faithfulness besides the fact that we use it in the proof. Here we provide a counterexample illustrating that for the second representation above, where a logical widget assignment is a complete, consistent set of literals, strict entailment can produce unsound results, *i.e.*, there is a Δ, Λ, and $p(a)$ such that $\Lambda \models_E^{\Delta} p(a)$ but $Rep[\Lambda]$ does not positively imply $p(a)$ under $Rep_U[\Delta]$ for any appropriate U.

Consider the following sentences.

$$\neg p(a)$$
$$\neg p(x) \Rightarrow q(x)$$

For Λ, we choose $\{p(a), \neg p(b)\}$. For the query, we choose $q(b)$. The witness for strict entailment is $\neg p(b)$, which obviously entails $q(b)$ and is consistent with Δ.

Now we show that $Rep[\Lambda]$ does not positively imply $p(a)$ under $Rep_U[\Delta]$ for any appropriate U. First notice that we can assume U is $\{a, b\}$ since U must contain all constants in Δ (and though b does not appear in Δ we can make it appear by adding $p(b) \vee \neg p(b)$ without changing the essence of the example).

Second, because the subset of Λ does not represent a single widget assignment, we need to argue for all possible widget assignments represented by $\neg p(b)$ that are subsets of $Rep[\Lambda]$. In this regard, notice that there are two possible definitions for $v \subseteq v'$: value-based or widget-based. The value-based version says that for all $p \in dom(v)$ and all $c \in v(p)$, $c \in v'(p)$. The widget-based version says that for all $p \in dom(v)$, $v(p) = v'(p)$. The value-based version allows extra values to be added to widget assignments, whereas the widget-based version only allows new widgets to be added. We consider each case separately.

For the value-based version of subset, any widget assignment v represented by $\{\neg p(b)\}$ has the property that $b \notin v(p)$ or that $p \notin dom(v)$. But in either case, one of the extensions to v includes b in $v(p)$, which can then be extended consistently so that $b \notin q$, thus demonstrating that $Rep[\Lambda]$ does not positively imply $q(b)$.

For the widget-based version of subset, consider any $v \subseteq Rep[\Lambda]$. Either $v(p) = Rep[\Lambda](p)$ or $dom(v) = \emptyset$. For the former case, v is inconsistent with Δ, ensuring v does not positively imply $q(b)$. For the latter case, one of the extensions for v makes $b \in v(p)$, which can be extended consistently so that $b \notin v(q)$, ensuring v again does not positively imply $q(b)$.

Paraconsistent Entailment Complexity

Here we give complexity results for two notions of paraconsistent entailment for quantifier-free, function-free, monadic first-order logic, denoted MON. The first is the well-known existential entailment: $\Delta \models_E \phi$ if there is a satisfiable $\Delta_0 \subseteq \Delta$ such that $\Delta_0 \models \phi$. The second is strict entailment, where the premises Γ are restricted to be a set of atoms. $\Gamma \models_E^{\Delta} \phi$ if there is a $\Gamma_0 \subseteq \Gamma$ such that $\Gamma_0 \cup \Delta$ is consistent and entails ϕ.

For each paraconsistent entailment relation we will consider several variations of MON that differ in two dimensions: variables and equality. We consider an arbitrary number of variables (denoted MON_*) and a constant number of variables (MON_k). Orthogonally, we consider MON with and without equality. When equality is allowed, we superscript the language with = to produce either $\text{MON}_k^=$ or $\text{MON}_*^=$.

We show complexity results for atomic queries. None of our results are tight, but because they place the entailment relations in the lower levels of the polynomial hierarchy, our results are sufficient to guarantee that the optimal algorithm is singly-exponential in time (barring a collapse of the lower levels of the polynomial hierarchy).

Existential Entailment

Proposition 8 ($\text{MON}_* \models_E$ hardness). *Suppose Δ is a finite set of quantifier-free, function-free, equality-free monadic first-order sentences with two relation constants. $\Delta \models_E p(a)$ is Π_2^{p}-hard.*

Proof. The following problem is Σ_2^{p}-hard. Given a first-order formula $\exists^*\forall^*\phi$ where ϕ is quantifier-free with at most one unary relation constant, no functions, no equality, determine if $\exists^*\forall^*\phi$ is satisfiable. First we claim that $\exists^*\forall^*\phi \models p(a)$ is Π_2^{p}-hard even if $\exists^*\forall^*\phi$ is known to be satisfiable. Second we claim that for satisfiable $\exists^*\forall^*\phi$, $\exists^*\forall^*\phi \models p(a)$ if and only if $\exists^*\forall^*\phi \models_E p(a)$. Third we claim that $\exists^*\forall^*\phi \models_E p(a)$ if and only if the skolemization $\forall^*\phi' \models_E p(a)$. Thus, we can embed a Π_2^{p}-hard problem in polynomial time within existential entailment where the premises are in quantifier-free (equivalently $\forall^*$), function-free, equality-free monadic first-order logic with two rela-

tion constants: the one appearing in ϕ and p. Now we prove the three claims.

Claim 1: $\exists^*\forall^*\phi \models p(a)$ for a fresh $p(a)$ is Π_2^{p}-hard even if $\exists^*\forall^*\phi$ is known to be satisfiable. Since satisfiability of $\exists^*\forall^*\phi$ is Σ_2^{p}-hard, unsatisfiability is Π_2^{p}-hard. We show that $\exists^*\forall^*\phi$ is unsatisfiable if and only if $\neg\exists^*\forall^*\phi \Rightarrow p(a) \models p(a)$. ($\Rightarrow$) If $\exists^*\forall^*\phi$ is unsatisfiable, $\neg\exists^*\forall^*\phi$ is true in all models, and every model that satisfies $\neg\exists^*\forall^*\phi \Rightarrow p(a)$ satisfies $p(a)$ and hence entailment holds. ($\Leftarrow$) Suppose entailment holds, then there is a resolution proof of the empty clause from (the clausal form of) $\exists^*\forall^*\phi \vee p(a)$ and $\neg p(a)$. If $p(a)$ does not appear in the proof, we have a proof of the unsatisfiability of $\exists^*\forall^*\phi$; otherwise, we can reorder the proof so that $p(a)$ is eliminated first (using pure literal elimination). The resulting proof includes within it a proof of the unsatisfiability of $\exists^*\forall^*\phi$ since the elimination of $p(a)$ makes no variable bindings.

Claim 2: for satisfiable $\exists^*\forall^*\phi$, $\exists^*\forall^*\phi \models p(a)$ if and only if $\exists^*\forall^*\phi \models_E p(a)$. ($\Rightarrow$) Suppose entailment holds. Then since the premise is satisfiable, it is the witness required for existential entailment. ($\Leftarrow$) Suppose existential entailment holds. Then since $p(a)$ is not a tautology, the only premise set that could entail it is $\{\exists^*\forall^*\phi\}$, and hence entailment must hold.

Claim 3: $\exists^*\forall^*\phi \models_E p(a)$ if and only if $\forall^*\phi' \models_E p(a)$ where $\forall^*\phi'$ is the skolemization of $\exists^*\forall^*\phi$. By skolemization, we know that the equivalence holds under traditional entailment. Moreover, skolemization preserves satisfiability; hence the two premises are equally satisfiable and equally entail $p(a)$; hence, the existential entailment of $p(a)$ for the two premises is the same. □

Proposition 9 ($\mathbf{MON}_k \models_E$ hardness). *Suppose Δ is a finite set of quantifier-free, function-free, equality-free monadic first-order sentences with at most a constant k number of variables. $\Delta \models_E p(a)$ is Σ_1^{p}-hard and Π_1^{p}-hard.*

Proof. Here we embed the satisfiability and unsatisfiability of propositional logic within existential entailment to arrive at the Σ_1^{p}-hardness and Π_1^{p}-hardness bounds, respectively. Propositional logic can be encoded easily in monadic logic using no variables, one object constant b, and one monadic predicate for each propositional constant. For any Herbrand interpretation, a monadic predicate p is assigned the object b if and only if the original proposition were true in the corresponding propositional interpretation. Below, we ignore such details.

We know that the satisfiability of a sentence ϕ in propositional logic is NP-hard (Σ_1^{p}-hard). We claim that ϕ is satisfiable if and only if $\{\phi, \phi \Rightarrow p\} \models_E p$, where p is fresh. ($\Rightarrow$) Suppose ϕ is satisfiable. Then both the premises are also satisfiable, and by a single application of modus ponens, the premises entail p. This ensures the premises existentially entail p. ($\Leftarrow$) Suppose existential entailment holds. Since p is fresh and not a tautology, the only premise subset that could entail it is $\{\phi, \phi \Rightarrow p\}$. Since that premise set is satisfiable, so are its members, and hence ϕ is satisfiable. Since both directions hold, we can embed propositional satisfiability within existential entailment (with a constant number of variables) and hence we have Σ_1^{p}-hardness.

We also know that the unsatisfiability of a propositional sentence ϕ is coNP-hard (Π_1^{p}-hard). Consequently, entailment $\phi \models p$ is coNP-hard, and so is entailment when the premise ϕ is known to be satisfiable. We claim that for satisfiable ϕ, $\phi \models p$ if and only if $\phi \models_E p$. ($\Rightarrow$) Suppose $\phi \models p$. Then since ϕ is satisfiable, it is the witness ensuring that $\phi \models_E p$. ($\Leftarrow$) Suppose $\phi \models_E p$. Then since p is not a tautology, the only subset that can entail it is $\{\phi\}$, and hence $\phi \models p$.

Now we prove that $\phi \models p$ is coNP-hard even when ϕ is satisfiable. We show that ϕ is unsatisfiable if and only if $\neg\phi \Rightarrow p \models p$ for a fresh p. ($\Rightarrow$) Suppose ϕ is unsatisfiable. Then in every model $\neg\phi$ is true, and in every model satisfying $\neg\phi \Rightarrow p$, p must be true since $\neg\phi$ is true; thus, p is entailed. ($\Leftarrow$) Suppose entailment holds. Then there is a resolution proof of the empty clause from $\phi \vee p$ and $\neg p$. For any such proof, if p does not occur, that proof demonstrates that ϕ is unsatisfiable; otherwise, place the removal of the p from clauses of ϕ first, and the result is a proof that demonstrates the unsatisfiability of ϕ. □

Proposition 10 ($\mathbf{MON}_k \models_E$ fragment hardness). *Suppose Δ is a finite set of quantifier-free, function-free, equality-free monadic first-order sentences in clausal form with one variable and no object constants. $\Delta \models_E p(a)$ is Π_1^{p}-hard.*

Proof. We know that the unsatisfiability of a propositional sentence set Γ is coNP-hard, even when Γ is in clausal form. We claim that Γ is unsatisfiable if and only if $\neg\Gamma \Rightarrow p \models p$ for a fresh p. First, notice that $\neg\Gamma \Rightarrow p$ is satisfied by the model where p is true. Second, $\neg\Gamma \Rightarrow p$ is equivalent to $\Gamma \vee p$, and since Γ is already in clausal form, the conversion of $\Gamma \vee p$ to clausal form requires only adding p to every element of Γ. Assuming the claim, this ensures that $\Gamma \models p$ is coNP-hard even if Γ is satisfiable and in clausal form.

Now we prove the claim. ($\Rightarrow$) Suppose Γ is unsatisfiable. Then $\neg\Gamma$ is true in all models, and every model satisfying the premise satisfies p; hence, p is entailed. ($\Leftarrow$) Suppose $\neg\Gamma \Rightarrow p \models p$. Then the resolution proof of the empty clause can be arranged so that p is resolved away first, leaving a proof of the empty clause from Γ.

We know $\Gamma \models p$ in propositional logic is coNP-hard, even if Γ is in clausal form and known to be satisfiable; hence, the unsatisfiability of $\Gamma \cup \{\neg p\}$ is coNP-hard. Let $\Gamma(x)$ be the clause set Γ where every propositional literal has been changed into a monadic literal using the variable x. We claim that $\Gamma(x) \cup \{\neg p(x)\}$, where $\Gamma(x)$ is satisfiable is unsatisfiable if and only if $\Gamma \cup \{\neg p\}$ is unsatisfiable. To see this, consider a resolution proof in the monadic version. Note that every resolution binds x to x, and hence we can ignore all variables during resolution. A proof of the empty clause in either version allows us to reconstruct a proof of the empty clause in the other version; moreover, the proofs are structurally identical. This suffices to conclude that the unsatisfiability of $\Gamma(x) \cup \{\neg p(x)\}$, where $\Gamma(x)$ is satisfiable and in clausal form is coNP-hard.

The clause set $\Gamma(x) \cup \{\neg p(x)\}$ is unsatisfiable if and only if $\Gamma(x) \cup \{\neg p(a)\}$ is unsatisfiable, by Herbrand's theorem. (Each set is unsatisfiable if and only if its grounding is unsatisfiable, and in the first case since there is no object constant,

we are free to choose a constant, say a. Then both sets have exactly the same grounding.) Additionally, $\Gamma(x) \cup \{\neg p(a)\}$ is unsatisfiable if and only if $\Gamma(x) \models p(a)$. Putting these two facts together allows us to embed a coNP-hard problem within our class of existential entailment queries and hence guarantees coNP-hardness (Π_1^{p}-hardness). □

Corollary 1 ($\text{MON}_*^=$ and $\text{MON}_k^= \models_E$ hardness). *Suppose Δ is a finite set of quantifier-free, function-free, monadic first-order sentences obeying the unique-names assumption. $\Delta \models_E p(a)$ is Σ_2^{p}-hard. If the number of variables is bounded by a constant k then $\Delta \models_E p(a)$ is Σ_1^{p}-hard and Π_1^{p}-hard, and if additionally Δ is restricted to clausal form, $\Delta \models_E p(a)$ is Π_1^{p}-hard.*

Proof. Trivially MON_* can be embedded in $\text{MON}_*^=$; likewise, MON_k can be embedded in $\text{MON}_k^=$. Since MON_* is Σ_2^{p}-hard, so is $\text{MON}_*^=$. Likewise, since MON_k is Σ_1^{p}-hard and Π_1^{p}-hard, so is $\text{MON}_k^=$, and since the restriction to clausal form for a fragment of MON_k is Π_1^{p}-hard, so is the restriction of $\text{MON}_k^=$ to clausal form Π_1^{p}-hard. □

Proposition 11 ($\text{MON}_*^= \models_E$ inclusion). *Suppose Δ is a finite set of quantifier-free, function-free first-order sentences obeying the unique names assumption where every relation constant has arity at most some constant k. $\Delta \models_E p(a)$ is included in Σ_3^{p}.*

Proof. Consider the following alternating time algorithm.

1. (Existential) Guess a subset Δ_0 of Δ.
   // Check satisfiability
   2. (Existential) Guess a Herbrand interpretation I
      3. (Universal) For all variable assignments v
         4. Validate that $\models_I \Delta_0[v]$
   // Check entailment
   4. (Universal) For all Herbrand interpretations I
      5. (Existential) Guess a variable assignment v.
         6. Validate that $\models_I \neg\Delta_0[v] \vee p(a)$

Here we use the fact that $\models \forall\bar{x}.\phi(\bar{x}) \Rightarrow p(a)$ if and only if every interpretation satisfies $\exists\bar{x}.\neg\phi(\bar{x}) \vee p(a)$ or equivalently that for every interpretation there is a variable assignment v such that $\neg\phi(v) \vee p(a)$ is satisfied.

Because Δ is quantifier-free, obeys the unique names assumption, and the query $p(a)$ is ground, Herbrand's theorem ensures that $\Delta \models p(a)$ if and only if every Herbrand model of Δ satisfies $p(a)$. Thus, the algorithm above is sound and complete. As for complexity, assuming steps (4) and (6) run in polynomial time, we have an algorithm in the polynomial hierarchy with quantifier prefix $\exists\exists\forall\exists$. That gives us Σ_3^{p} inclusion.

All that remains is showing that steps (4) and (6) run in polynomial time in the size of Δ, which requires showing that the evaluation of ground subsets of Δ over an interpretation takes polynomial time. First, note that the size of every Herbrand model is at most the number of predicates occurring in Δ times the number of object constants occurring in Δ to the kth power, which is clearly polynomial in Δ since k is a constant. Every ground instance $\Delta'[v]$ is linear in Δ, so it too is polynomial in Δ. Satisfaction of a ground instance $\Delta'[v]$ in I (including $=$ checks, which are especially trivial due to the UNA) is polynomial (at worst $|I| * |\Delta'[v]|$).

□

Proposition 12 ($\text{MON}_k^= \models_E$ inclusion). *Suppose Δ is a finite set of quantifier-free, function-free first-order sentences obeying the unique names assumption where every relation constant has arity at most some constant j, and Δ has at most some constant k variables. $\Delta \models_E p(a)$ is included in Σ_2^{p}.*

Proof. Consider the following alternating time algorithm.

1. (Existential) Guess a subset Δ_0 of Δ.
   // Check satisfiability
   2. (Existential) Guess a Herbrand interpretation I
      3. Validate that $\models_I \Delta_0[v]$
         for every variable assignment v
   // Check entailment
   4. (Universal) For all Herbrand interpretations I
      5. Validate that $\models_I \neg\Delta_0[v] \vee p(a)$
         for some variable assignment v

This algorithm differs from the previous one in that the search through variable assignments can be folded into the polynomial-time validation steps. To see why, notice that because there are at most k variables in Δ, the number of variable assignments is $|objs[\Delta]|^k$, which is polynomial in Δ. Thus, both validation steps (3) and (5) run in polynomial time. The remainder of the proof is the same as the previous one. □

Corollary 2 (MON_* and $\text{MON}_k \models_E$ inclusion). *Suppose Δ is a finite set of quantifier-free, function-free, equality-free monadic first-order sentences obeying the unique-names assumption. $\Delta \models_E p(a)$ is included in Σ_3^{p}, and if the number of variables is bounded by a constant k then $\Delta \models_E p(a)$ is included in Σ_2^{p}.*

Proof. Trivially MON_* can be embedded in $\text{MON}_*^=$; likewise, MON_k can be embedded in $\text{MON}_k^=$. Since $\text{MON}_*^=$ is included in Σ_3^{p}, so is MON_*; likewise, since $\text{MON}_k^=$ is included in Σ_2^{p}, so is MON_k. □

Strict Entailment The results for existential entailment are useful because they help us with similar bounds for strict entailment. In particular, existential entailment can be simulated by strict entailment, ensuring that strict entailment is at least as hard as existential entailment.

Proposition 13 ($\text{MON}_*^= \models_E^{\Omega}$ inclusion). *Suppose Δ is a finite set of quantifier-free, function-free first-order sentences obeying the unique names assumption where every relation constant has arity at most some constant k. Further suppose Γ is a set of ground sentences. $\Gamma \models_E^{\Delta} p(a)$ is included in Σ_3^{p}.*

Proof. Consider the following simple variation on the algorithm for existential entailment.

1. (Existential) Guess a subset Γ_0 of Γ.
   // Check satisfiability
   2. (Existential) Guess a Herbrand interpretation I
      3. (Universal) For all variable assignments v
         4. Validate that $\models_I \Gamma_0 \cup \Delta[v]$
   // Check entailment
   4. (Universal) For all Herbrand interpretations I
      5. (Existential) Guess a variable assignment v.
         6. Validate that $\models_I \neg(\Gamma_0 \cup \Delta)[v] \vee p(a)$

The soundness, completeness, and complexity arguments hold just as before, ensuring the same Σ_3^{p} upper bound. □

Proposition 14 ($\mathbf{MON}_k^{=}$ $\models_E^{\Omega}$ inclusion). *Suppose Δ is a finite set of quantifier-free, function-free first-order sentences obeying the unique names assumption where every relation constant has arity at most some constant j, and Δ has at most some constant k number of variables. Further suppose Γ is a set of ground sentences. $\Gamma \models_E^{\Delta} p(a)$ is included in Σ_2^{p}.*

Proof. The following simple variation on the previous algorithm leverages the fact that there are only a polynomial number of variable assignments and therefore only a polynomial number of instances of Δ; thus, the search through variable assignments can be folded into the validation routines, eliminating one level of alternating time splitting. See proof for $\mathrm{MON}_k^{=}$ and existential entailment.

1. (Existential) Guess a subset Γ_0 of Γ.
   // Check satisfiability
   2. (Existential) Guess a Herbrand interpretation I
      3. Validate that $\models_I \Gamma_0 \cup \Delta[v]$
         for every variable assignment v.
   // Check entailment
   4. (Universal) For all Herbrand interpretations I
      5. Validate that $\models_I \neg(\Gamma_0 \cup \Delta)[v] \vee p(a)$
         for some variable assignment v.

□

Corollary 3 ($\mathbf{MON}_*$ and $\mathbf{MON}_k$ $\models_E^{\Omega}$ inclusion). *Suppose Δ is a finite set of quantifier-free, function-free, equality-free monadic first-order sentences obeying the unique-names assumption. Further suppose Γ is a set of ground sentences. $\Gamma \models_E^{\Delta} p(a)$ is included in Σ_3^{p}, and if the number of variables is bounded by a constant k then $\Gamma \models_E^{\Delta} p(a)$ is included in Σ_2^{p}.*

Proof. Trivially MON_* can be embedded in $\mathrm{MON}_*^{=}$; likewise, MON_k can be embedded in $\mathrm{MON}_k^{=}$. Since $\mathrm{MON}_*^{=}$ is included in Σ_3^{p}, so is MON_*; likewise, since $\mathrm{MON}_k^{=}$ is included in Σ_2^{p}, so is MON_k. □

Proposition 15 ($\models_E^{\Omega}$ is $\models_E$-hard). *Suppose Δ is a finite set of first-order sentences. Further suppose Γ is a set of ground atoms. $\Gamma \models_E^{\Delta} p(a)$ is as hard as $\Delta \models_E p(a)$.*

Proof. Consider an existential entailment problem $\Sigma \models_E p(a)$, where Σ is a finite set of first-order sentences. We show how to construct a Δ and a Γ such that $\Sigma \models_E p(a)$ if and only if $\Gamma \models_E^{\Delta} p(a)$.

Δ is constructed as follows. For each sentence σ in Σ, create the sentence $q(a) \Rightarrow \sigma$, where q is a new predicate. (Notice that any universal quantifiers implicitly applied to σ are still applied in the same way to $q(a) \Rightarrow \sigma$.) Γ is then the set of all $q(a)$ where q was introduced during the construction of Δ. Now we must show that the strict entailment query $\Gamma \models_E^{\Delta} p(a)$ exactly when $\Sigma \models_E p(a)$.

($\Leftarrow$) If $\Sigma \models_E p(a)$, then there is a satisfiable $\Sigma_0 \subseteq \Sigma$ that entails $p(a)$. Consider the sentences Σ_0' constructed from elements of Σ_0 and placed into Δ. Let Q denote the set of all $q(a)$ in the antecedents of the Σ_0' implications. Since Σ_0 is satisfiable and all the q are fresh, we can extend any model satisfying Σ_0 to additionally satisfy all of Q by forcing $q(a)$ true for exactly those fresh q occuring in Q. Since that model satisfies all of Σ_0, it satisfies all of Σ_0' because the consequent of every rule evaluates to true; moreover, since the model also satisfies Q, it satisfies $Q \cup \Sigma_0'$. Since the model fails to satisfy the additional fresh q's in Δ, it also satisfies $Q \cup \Delta$. By construction, Q is a subset of Γ, and after $|\Sigma_0|$ applications of modus ponens, $Q \cup \Delta$ produces each sentence in Σ_0, which is known to entail $p(a)$. $Q \cup \Delta$ is satisfiable and entails $p(a)$; thus, $\Gamma \models_E^{\Delta} p(a)$.

($\Rightarrow$) Suppose $\Gamma \models_E^{\Delta} p(a)$. Then there is a $Q \subseteq \Gamma$ that is consistent with Δ such that $Q \cup \Delta$ entails $p(a)$. Since all sentences in Δ are of the form $q(a) \Rightarrow \phi$, and all the q are fresh, Δ alone cannot entail $p(a)$. (The countermodel makes $p(a)$ and all the $q(a)$ false, satisfying Δ but not $p(a)$.) So Q must be nonempty. Consider just those sentences S from Δ whose antecedents belong to Q. We claim that necessarily $Q \cup S$ is both satisfiable (since $Q \cup \Delta$ was) and entails $p(a)$.

For the purpose of contradiction, suppose some of those remaining sentences R in Δ were necessary for entailing $p(a)$—that there were a model M of $Q \cup S$ that did not satisfy $p(a)$ but that all models of $Q \cup S \cup R$ satisfied $p(a)$. For M to be a model of $Q \cup S$ but not $Q \cup S \cup R$, M must not satisfy R; thus, M must satisfy all the $q(a)$ in the antecedents of R. But since all the qs are fresh, there is another model M' that is the same as M (especially in that it satisfies $p(a)$) except it fails to satisfy the $q(a)$ in R, and hence satisfies all of R. M' therefore satisfies $Q \cup S \cup R$ but fails to satisfy $p(a)$, by construction, contradicting the original premise. Hence, there can be no such R.

$Q \cup S$ is therefore satisfiable and entails $p(a)$. The models of $Q \cup S$ must all (a) satisfy $p(a)$ and (b) satisfy all the consequents S' of S (since they satisfy all of Q). Because all the q occurring in Q are fresh, the reducts of the models satisfying $Q \cup S$ to the original language is axiomatized by S'; hence, S' is satisfiable and entails $p(a)$. Since S' is a subset of Σ, $\Sigma \models_E p(a)$.

□

Corollary 4. *Suppose* MON *is one of* MON_*, MON_k, $\mathrm{MON}_*^{=}$, *or* $\mathrm{MON}_k^{=}$. *If* MON *is* C*-hard for existential entailment,* MON *is* C*-hard for strict entailment.*

Proof. The proof above introduces no additional variables and is ambivalent toward $=$; thus, the embedding of existential entailment for MON into strict entailment produces axioms in MON. Thus, if existential entailment is C-hard, then the proof above when instantiated with MON demonstrates that strict entailment is also C-hard. □

Compilation

Here we prove that our algorithms meet the definition of a paraconsistent web form compiler and discuss complexity results. First we address soundness and completeness, starting with the ontology transformation algorithm, followed by the data transformation algorithm, and then we handle complexity.

Soundness and Completeness Recall that COMPILE (Algorithm 1) is our ontology transformation operation, which transforms a logical ontology Δ to a set of first-order equivalences. Complementing COMPILE is β, our data transformation operation that constructs an intepretation $\mathcal{I}$ from a given logical widget assignment Λ. For each predicate p occurring in Λ, $\mathcal{I}(p) = \{a | p(a)$ occurs positively in $\Lambda\}$. Additionally, β maps each $consistent^{\Delta}_{P}$, where P is a set of predicates, to $true$ if and only if the widget assignment for P is consistent with Δ.

Theorem 6 (Soundness and Completeness). *The pair* $\langle$COMPILATION, $\beta\rangle$ *is a paraconsistent web form compiler.*

Proof. The soundness and the completeness theorems below guarantee the result. □

Theorem 7 (Soundness). *Consider the pair* $\langle$COMPILATION, $\beta\rangle$. *For any finite logical widget assignment* Λ *and any object constant* a*,*

$$\Lambda \models^{\Delta}_{E} p(a) \text{ if } \models_{\beta[\Lambda]} \phi^{+}_{p}(a)$$

$$\Lambda \models^{\Delta}_{E} \neg p(a) \text{ if } \models_{\beta[\Lambda]} \phi^{-}_{p}(a)$$

Proof. The positive and negative cases are analogous, so here we focus on the positive. Suppose $\models_{\beta[\Lambda]} \phi^{+}_{p}(a)$. Then there is some contrapositive $p(t) \Leftarrow \psi(t, \bar{y})$ of some clause in RES$[\Delta] \cup \{p(x) \vee \neg p(x)\}$ such that

$$\models_{\beta[\Lambda]} t = a \wedge \exists \bar{y}.\psi(t, \bar{y}) \wedge consistent^{\Delta}_{\text{PREDS}[\psi]\cup\{=\}}.$$

Since t must be syntactically equivalent to a (we make the unique names assumption for equality),

$$\models_{\beta[\Lambda]} \exists \bar{y}.\psi(a, \bar{y}) \wedge consistent^{\Delta}_{\text{PREDS}[\psi]\cup\{=\}}.$$

Since $\mathcal{I} = \beta[\Lambda]$ satisfies the conjunction above, it satisfies the two components individually. In regard to the latter, we know that the individual widget assignments for the predicates occurring in $\psi(a, \bar{y})$ (jointly denoted Λ_0) are consistent with Δ. Note that $\Lambda_0 \subseteq \Lambda$ and that $\Delta \cup \Lambda_0$ is consistent. We will now demonstrate that $\Delta \cup \Lambda_0 \models p(a)$, and hence that $\Lambda \models^{\Delta}_{E} p(a)$, which completes the proof.

To show that $\Delta \cup \Lambda_0 \models p(a)$, we demonstrate that there is a unit resolution proof of $p(a)$ from the clausal form of $p(t) \Leftarrow \psi(t, \bar{y})$ and the clausal form of Λ_0. If $p(t) \Leftarrow \psi(t, \bar{y})$ is a member of RES$[\Delta]$ then because resolution is sound, unit resolution is sound, and entailment is transitive and monotonic, such a unit resolution proof is sufficient to guarantee the required entailment. Otherwise, $p(t) \Leftarrow \psi(t, \bar{y})$ is the sentence $p(x) \Leftarrow p(x)$, in which case Λ_0 by itself entails $p(a)$. In both cases, the requisite entailment holds.

Since $\mathcal{I}$ satisfies $\exists \bar{y}.\psi(a, \bar{y})$, we know $\models_{\mathcal{I}} \psi(a, \bar{b})$ for some b-tuple in the model's universe. By construction, the universe of $\mathcal{I}$ is the set of objects occurring in $\Delta \cup \Lambda$. The truth of any ground atom $q(b)$ in interpretation $\mathcal{I}$ is exactly $b \in \mathcal{I}(q)$, and the truth of any negated ground atom $\neg q(b)$ is exactly $b \notin \mathcal{I}(q)$. Since the ground conjunction $\psi(a, \bar{b})$ is true, all of the positive atoms are true in $\mathcal{I}$, and none of the negative atoms are. Such an $\mathcal{I}$ could only have been constructed from a logical widget assignment where all the positive atoms occur positively and all of the negative atoms occur negatively, ensuring that the clausal form Λ_0 includes that set of positive and negative literals.

Now consider the clause $p(t) \vee \neg\psi(t, \bar{y})$ together with the clausal form of Λ_0 and the negated conclusion $\neg p(a)$. First resolve $\neg p(a)$ with $p(t)$ to produce $\neg\psi(a, \bar{y})$. (Recall that t and a are unifiable, as discussed above.) Now consider the instance $\psi(a, \bar{b})$ of $\psi(a, \bar{y})$ satisfied by $\mathcal{I}$. The negations of all literals in this instance are included in the clausal form of Λ_0 (since we are dealing with the negation of $\psi(a, \bar{y})$ instead of $\psi(a, \bar{y})$ itself). Thus, unit resolution will produce the empty clause, ensuring $p(a)$ is entailed. □

Theorem 8 (Completeness). *Consider the pair* $\langle$COMPILATION, $\beta\rangle$. *For any finite logical widget assignment* Λ *and any object constant* a*,*

$$\Lambda \models^{\Delta}_{E} p(a) \text{ only if } \models_{\beta[\Lambda]} \phi^{+}_{p}(a)$$

$$\Lambda \models^{\Delta}_{E} \neg p(a) \text{ only if } \models_{\beta[\Lambda]} \phi^{-}_{p}(a)$$

Proof. The positive and negative cases are analogous, so here we focus on the positive. Suppose $\Lambda \models^{\Delta}_{E} p(a)$. Then there is a $\Lambda_0 \subseteq \Lambda$ that together with Δ is consistent and entails $p(a)$. Since $\Lambda_0 \cup \Delta$ is quantifier-free, there is a resolution refutation of $p(a)$ from the clausal form of Λ_0, denoted Λ^{c}_{0}, and the clausal form of Δ, denoted Δ^{c}.

Case 1. Suppose $p(a) \in \Lambda^{c}_{0}$. Consider the clause $p(x) \vee \neg p(x)$. This clause contributes the following disjunct to $\phi^{+}_{p}(x)$: $p(x) \wedge consistent^{\Delta}_{\{p\}}$. Since $\Lambda_0 \cup \Delta$ is consistent, and Λ_0 contains the individual widget assignment for p, $consistent^{\Delta}_{\{p\}}$ is satisfied by $\mathcal{I} = \beta[\Lambda]$. Since $p(a) \in \Lambda^{c}_{0}$, $p(a)$ occurs positively in Λ, $\mathcal{I}(p) \ni a$, and thus $\models_{\mathcal{I}} p(a)$. Since $\mathcal{I}$ satisfies both conjuncts, $\mathcal{I}$ satisfies $p(a) \wedge consistent^{\Delta}_{\{p\}}$. Since $\mathcal{I}$ satisfies one of the disjuncts of $\phi^{+}_{p}(a)$ and the disjuncts are independent, $\mathcal{I}$ satisfies $\phi^{+}_{p}(a)$.

Case 2. Suppose $p(a) \notin \Lambda^{c}_{0}$. Then there is a resolution refutation of $p(a)$ using clauses from Δ^{c} and Λ^{c}_{0}. Since the order of resolutions is irrelevant, move all resolutions involving just Δ^{c} and their descendents to the front. After those resolutions have been performed, the only resolutions remaining are unit resolutions between the negated conclusion ($\neg p(a)$), the ground literals of Λ^{c}_{0}, and a single clause δ in the resolution/factoring closure of Δ^{c}.

Note that δ includes a literal unifiable with $p(a)$; otherwise, there would be a proof of the empty clause without using the negated conclusion, making the premise set inconsistent and violating the choice of Λ_0. Thus, we can write δ as $p(x) \Leftarrow \psi(x, \bar{y})$. This contrapositive contributes the disjunct $\exists \bar{y}.\psi(x, \bar{y}) \wedge consistent^{\Delta}_{\text{PREDS}[\psi(x,\bar{y})]}$ to $\phi^{+}_{p}(x)$. Since the predicates occurring in $\psi(x, \bar{y})$ must all occur in

Λ_0 (or else unit resolution would fail to produce the empty clause) and $\Lambda_0 \cup \Delta$ is consistent, $consistent^{\Delta}_{\text{PREDS}[\psi(x,\bar{y})]}$ must be true in $\mathcal{I}$. We claim that $\models_{\mathcal{I}} \exists\bar{y}.\psi(a,\bar{y})$ and conclude $\models_{\mathcal{I}} \exists\bar{y}.\psi(a,\bar{y}) \wedge consistent^{\Delta}_{\text{PREDS}[\psi(x,\bar{y})]}$. This ensures that $\mathcal{I}$ satisfies one of $\phi^+_p(a)$'s independent disjuncts and hence that $\models_{\mathcal{I}} \phi^+_p(a)$.

Now to prove the claim that $\models_{\mathcal{I}} \exists\bar{y}.\psi(a,\bar{y})$. During the unit resolution proof of the empty clause from δ, written in clausal form as $p(x) \vee \neg\psi(x,\bar{y})$, suppose the variable bindings for δ are recorded as σ. (There is no need to create fresh variables since only δ is the only clause that contains variables. Moreover, all variables in δ will be bound in σ since each resolution binds all variables in the literal of δ resolved upon, and all such literals must be resolved upon to produce the empty clause.) Since all of the literals in the ground $\psi(x,\bar{y})\sigma$ are resolved away, the negation of each of those literals belongs to Λ^c_0 and appears as the appropriate truth assignment in $\mathcal{I}$ ($c \in \mathcal{I}(q)$ if $q(c) \in \Lambda^c_0$ and $c \notin \mathcal{I}(q)$ if $\neg q(c) \in \Lambda^c_0$). Thus, $\models_{\mathcal{I}} \psi(x,\bar{y})\sigma$. Without loss of generality we can assume x is bound to a and hence $\models_{\mathcal{I}} \psi(a,\bar{y})\sigma$. Therefore $\models_{\mathcal{I}} \exists\bar{y}.\psi(a,\bar{y})$, concluding the proof. □

Having proven soundness and completeness for the core of the ontology transformation algorithm, we analyze the core of the data transformation algorithm. Recall that the data transformation operation operates on a logical widget assignment Λ and is denoted $\beta[\Lambda] = \beta_3[\beta_2[\beta_1[\Lambda]]]$, where the definition for each β_i can be found below.

The theorems below utilize the following notation. Λ_P is the subset of Λ consisting of exactly the individual widget assignments for the predicates P.

$$\Lambda_P = \{i \in \Lambda \mid p \in P \text{ occurs in } i\}$$

First we show that DATACOMPILE constructs the appropriate set of database queries.

Theorem 9 (Soundness and Completeness of DATACOMPILE). *Suppose Δ is a logical ontology, Λ is a logical widget assignment, and P is a predicate set. $\Lambda_P \cup \Delta$ is inconsistent if and only if there is some $Q \subseteq P$ such that $\sigma_Q \in$ DATACOMPILE$[\Delta]$ and $\models_{\beta_1[\Lambda]} \sigma_Q$.*

Proof. ($\Leftarrow$) Use $\mathcal{I}$ to denote $\beta_1[\Lambda]$. Suppose there is some $Q \subseteq P$ such that $\sigma_Q \in$ DATACOMPILE$[\Delta]$ and $\models_{\beta_1[\Lambda]} \sigma_Q$. Then by the construction of σ_Q there is some clause $c \in$ RES$[\Delta]$ such that PREDS$[c] = Q$ and $\models_{\mathcal{I}} \exists^*.\neg c$. Thus, there is some variable binding γ from variables in c to the elements of the universe of $\mathcal{I}$ such that $\models_{\mathcal{I}} \neg c\gamma$. By construction, the universe of $\mathcal{I}$ is the set of object constants occurring in $\Delta \cup \Lambda$; thus, there is some ground instance of $\neg c$ satisfied by $\mathcal{I}$. Suppose we write $\neg c$ so that $\neg$ is only applied to atoms and call the result ρ. Then every positive literal appearing in ρ is satisfied by $\mathcal{I}$ and thus occurs positively in Λ, and every atom from a negative literal appearing in ρ is falsified by $\mathcal{I}$ ensuring that the atom appears negatively in Λ.

Consider the clausal form of $\Delta \cup \Lambda_Q$. Since c belongs to the resolution closure of Δ, it belongs to the resolution closure of $\Delta \cup \Lambda_Q$. Additionally, the clausal form of Λ_Q includes the literals described above. The variable binding γ above witnesses a resolution proof of the empty clause from c and the literals so described. Since the predicates occurring in c are exactly Q, the necessary literals are drawn from Λ_Q. Thus, the resolution proof that $\Delta \cup \Lambda_Q$ is inconsistent begins with the resolution proof of c from Δ and continues with a sequence of unit resolutions on c using literals from Λ_Q. Since every superset of an inconsistent set is inconsistent, and $P \supseteq Q$, we know $\Lambda_P \supseteq \Lambda_Q$ and therefore that $\Delta \cup \Lambda_P$ is a superset of the inconsistent $\Delta \cup \Lambda_Q$. Ergo $\Delta \cup \Lambda_P$ is inconsistent.

($\Rightarrow$) Suppose $\Lambda_P \cup \Delta$ is inconsistent. Then there is a resolution proof of the empty clause from the clausal form of $\Lambda_P \cup \Delta$. Reorder the proof so that all resolutions between clauses from Δ and their resolvents come first, thus leaving a sequence of unit resolutions between some clause c in the resolution closure of Δ and the literals appearing in Λ_P at the end of the proof. Since the proof produces the empty clause, all of the predicates occurring in c must belong to P or else they could not be resolved away. Therefore, PREDS$[c] = Q \subseteq P$. Consequently, when DATACOMPILE constructs σ_Q, one of the disjuncts is $\exists^*.\neg c$. Similar to the soundness proof, the existence of literals from Λ_P participating in the proof of the empty clause guarantees that $\beta_1[\Lambda]$ satisfies $\exists^*.\neg c$; thus, $\models_{\beta_1[\Lambda]} \sigma_Q$. Since $Q \subseteq P$, this concludes the proof. □

Next we prove that $\beta = \beta_3 \circ \beta_2 \circ \beta_1$ is sound and complete.

$$\begin{aligned}
\beta_1[\Lambda] &= \{p \rightarrow \{a \mid p(a) \text{ is a positive conjunct in } \Lambda\} \\
&\quad \mid p \text{ is a widget}\} \\
\beta_2[\mathcal{I}] &= \{inconsistent_P \rightarrow true \mid \models_{\mathcal{I}} \sigma_P \text{ and} \\
&\quad \sigma_P \in \text{DATACOMPILE}[\Delta]\} \cup \mathcal{I} \\
\beta_3[\mathcal{I}] &= \{consistent^{\Delta}_P \rightarrow true \mid \\
&\quad consistent^{\Delta}_P \text{ occurs in COMPILE}[\Delta] \text{ and} \\
&\quad \text{there is no } Q \text{ such that } Q \subseteq P \text{ and} \\
&\quad \models_{\mathcal{I}} inconsistent^{\Delta}_Q\} \cup \mathcal{I}
\end{aligned}$$

Theorem 10 (Soundness and Completeness of β). *Suppose Δ is a logical ontology, and Λ is a logical widget assignment. If $consistent^{\Delta}_P$ occurs in* COMPILE*$[\Delta]$ then*

$$\models_{\beta_3[\beta_2[\beta_1[\Lambda]]]} consistent^{\Delta}_P \textit{ iff } \Delta \cup \Lambda_P \textit{ is consistent}$$

Proof. Use Σ to denote the results of DATACOMPILE$[\Delta]$.

($\Rightarrow$) Suppose $\models_{\beta_3[\beta_2[\beta_1[\Lambda]]]} consistent^{\Delta}_{P}$.

$\models_{\beta_3[\beta_2[\beta_1[\Lambda]]]} consistent^{\Delta}_{P}$
(by defn of β_3)
$\Rightarrow \nexists Q.Q \subseteq P$ and $\models_{\beta_2[\beta_1[\Lambda]]} inconsistent^{\Delta}_{Q}$
(by logical equivalence)
$\Rightarrow \forall Q.$ if $\models_{\beta_2[\beta_1[\Lambda]]} inconsistent^{\Delta}_{Q}$ then $\neg(Q \subseteq P)$
(by defn of β_2)
$\Rightarrow \forall Q.$ if ($\sigma_Q \in \Sigma$ and $\models_{\beta_1[\Lambda]} \sigma_Q$) then $\neg(Q \subseteq P)$
(by above theorem)
$\Rightarrow \forall Q.$ if ($\sigma_Q \in \Sigma$ and $\Lambda_Q \cup \Delta$ is inconsistent) then $\neg(Q \subseteq P)$
(by defn of DATACOMPILE)
$\Rightarrow \forall Q.$ if ($\exists c \in \text{RES}[\Delta] \wedge \text{PREDS}[c] = Q$ and $\Lambda_Q \cup \Delta$ is inconsistent) then $\neg(Q \subseteq P)$
(by logical equivalence)
$\Rightarrow \forall Q.$ if ($\exists c \in \text{RES}[\Delta] \wedge \text{PREDS}[c] = Q$ and $Q \subseteq P$ then $\Lambda_Q \cup \Delta$ is consistent)

Suppose that substituting P for Q satisfies the antecedent of the above implication. Then clearly $\Lambda_P \cup \Delta$ is consistent.

Now suppose that P does not satisfy the antecedent (because there is no $c \in \text{RES}[\Delta]$ such that $\text{PREDS}[c] = P$) and thus for all $Q \subset P$ for which there is a $c \in \text{RES}[\Delta]$ such that $\text{PREDS}[c] = Q$, we know $\Lambda_Q \cup \Delta$ is consistent. Suppose for the purpose of contradiction that $\Lambda_P \cup \Delta$ were inconsistent. Then there would be a resolution proof of the empty clause from $\Lambda_P \cup \Delta$, and we can reorder that proof so all resolutions between clauses in Δ come first, followed by a series of unit resolutions between some clause $d \in \text{RES}[\Delta]$ and the literals of Λ_P. Since d belongs to the closure and no clause in the closure includes exactly the predicates P, d must contain a subset R of the predicates P. Thus, the proof of the empty clause from d guarantees that $\Lambda_R \cup \Delta$ is inconsistent, but that contradicts the fact that for all $Q \subseteq P$ where there is some clause in the closure with predicates Q, $\Lambda_Q \cup \Delta$ is consistent.

($\Leftarrow$) Suppose $\Delta \cup \Lambda_P$ is consistent.

$\Delta \cup \Lambda_P$ is consistent
(by defn of consistency)
$\Rightarrow$ for all $Q \subseteq P.\Delta \cup \Lambda_Q$ is consistent
(by above theorem)
$\Rightarrow$ for all $Q \subseteq P.\ \nexists R \subseteq Q.(\sigma_R \in \Sigma$ and $\models_{\beta_1[\Lambda]} \sigma_R)$
(by logical equivalence)
$\Rightarrow$ for all $Q \subseteq P.\forall R \subseteq Q.\sigma_R \in \Sigma$ implies $\not\models_{\beta_1[\Lambda]} \sigma_R)$
(since every R is a Q and vice versa)
$\Rightarrow$ for all $Q \subseteq P.\sigma_Q \in \Sigma$ implies $\not\models_{\beta_1[\Lambda]} \sigma_Q)$
(by defn of β_2)
$\Rightarrow$ (1) for all $Q \subseteq P.\sigma_Q \in \Sigma$ implies $\not\models_{\beta_2[\beta_1[\Lambda]]} incons^{\Delta}_{Q}$ and
(2) for all $Q \subseteq P.\sigma_Q \notin \Sigma$ implies $\not\models_{\beta_2[\beta_1[\Lambda]]} incons^{\Delta}_{Q}$
(by (1) and (2))
$\Rightarrow$ for all $Q \subseteq P.\ \not\models_{\beta_2[\beta_1[\Lambda]]} incons^{\Delta}_{Q}$
(by defn of β_3, and theorem's assumption)
$\Rightarrow \models_{\beta_3[\beta_2[\beta_1[\Lambda]]]} consistent^{\Delta}_{P}$

□

Complexity First we describe results on the complexity of the resolution closure, in particular the size of the resolution closure. For quantifier-free, function-free, equality-free monadic logic in clausal form, the size of the resolution closure in the worst case is bounded from below by a single exponential and is bounded from above by a double exponential.

Proposition 16. *The output complexity of the resolution closure with subsumption and tautology elimination is* EXPSPACE*-hard and belongs to* 2EXPSPACE.

Proof. The single exponential result comes from the fact that in propositional clausal logic, proofs can be exponentially long. All steps in such proofs belong to the resolution closure and hence the size of the resolution closure in propositional logic (and hence clausal monadic logic) is singly exponential, *i.e.*, is EXPSPACE-hard.

The doubly exponential upper bound arises by counting the number of logically distinct monadic clauses. Each monadic clause is a disjunction of separable components, where each component includes all those literals with the same argument, *e.g.*, $p(x) \vee q(x) \vee p(y) \vee q(a)$ contains three components. The number of distinct possible components is the same as the number of propositional clauses: 3^p, where p is the number of predicates. Each monadic clause can include any subset of components built using variables, giving 2^{3^p} distinct object-free monadic clauses. Each such clause can then be augmented with any one component for each of the o distinct object constants, giving a total of $2^{3^p}(3^p)^o = 2^{3^p}3^{po}$ logically distinct monadic clauses. This ensures that the computation of the resolution closure belongs to 2EXPSPACE. □

In the case of one variable and no object constants, the resolution closure is bounded from above by a single exponential, and hence the output complexity is exactly EXPSPACE.

Proposition 17. *If the number of variables appearing in the given clause set is one and there are no object constants, the output complexity of resolution belongs to* EXPSPACE. *Moreover, the length of each clause in that closure is linear in the number of predicates.*

Proof. By straightforward induction we show below that the number of variables in any clause in the closure is exactly one. The number of clauses with a single variable is the same as the number of propositional clauses: 3^p, where p is the number of predicates. The length of each clause is linear in p. Thus, the maximal size of the closure is linear in $p3^p$, ensuring the EXPSPACE output complexity.

The induction to show that each clause in the closure contains one variable is done on the number of resolutions used to produce a clause. The base case is obvious: that each clause in the original clause set contains one variable. For the inductive step, consider any resolution: $\dfrac{p(x) \vee \Phi(x) \qquad \neg p(x) \vee \Psi(x)}{\Phi(x) \vee \Psi(x)}$ Because there are no object constants, the literals resolved upon must both contain variables, and those variables must be unified together. By the inductive hypothesis there is a single variable in each

clause, and because those variables are unified together, the resulting clause has a single variable. □

We know that in order to produce doubly exponential resolution closures, there must clauses with many variables in that closure. For if all clauses have no more than k variables, the number of distinct clauses is singly exponential, each of which is at most singly exponential long, resulting in a singly exponential output size.

Proposition 18. *If there is some polynomial $poly(o, p)$ dependent on the number of predicates p and number of object constants o, and a clause set S has no more than $poly(o, p)$ variables, then the size of S is singly exponential in $poly(o, p)$.*

Proof. Once the number of predicates is fixed, $poly(p)$ is simply a constant k. For k variables, the number of clauses without object constants is $(3^p)^k$. Each such clause can be augmented with one of $(3^p)^o$ clauses with just object constants, where o is the number of object constants. Thus the total number of clauses is $3^{pk}3^{po}$, which is singly exponential in $poly(o, p)$. □

The complexity of Algorithms 1 and 2 are both polynomial in the size of the resolution closure of the logical ontology. These are the compilation algorithms run once to generate a web form.

Proposition 19. *The complexity and output complexity of Algorithm 1 is the same as the complexity of converting the input to clausal form and then computing the resolution closure.*

Proof. After converting to clausal form and computing the resolution closure, Algorithm 1 computes all contrapositives, which produces a sentence set no larger than quadratic in the size of the closure. Adjoining consistency checks increases the sentence set size by at most a linear factor. Predicate completion does not appreciably affect the size. Thus, the complexity and output-complexity of Algorithm 1 is a polynomial over the complexity of the resolution closure. □

Proposition 20. *The complexity and output complexity of Algorithm 2 is the same as the complexity of converting the input to clausal form and then computing the resolution closure.*

Proof. Algorithm 2 effectively performs predicate completion on the result of resolution, ensuring the same complexity as the clausal form and closure computation. □

More importantly, the output of Algorithms 1 and 2 is used to compute strict entailment on the browser. That computation is in the worst case doubly exponential. To make the bound tight, we need a tight bound on the output complexity of Algorithm 1.

Proposition 21. *Assuming the logical ontology was written in clausal form, strict entailment computation on the browser runs in at worse doubly exponential time in the size of the logical ontology and the size of the logical widget assignment.*

Proof. Computing strict entailment amounts to evaluating the database queries produced by Algorithm 1 over the database constructed by Algorithm 2. The number of database queries constructed by Algorithm 1 is either singly or doubly exponential, and the length of each query is at most singly exponential in the number of predicates p and object constants o. The evaluation of each database query requires evaluating over the web form data plus a single consistency check. The consistency check can be indexed so that lookup is linear in the number of predicates; thus, the cost of evaluating a single query is at worst an exponential where the base depends on the size of the current widget assignment, and the exponent depends on the length of the query. Since the length of the query is at most singly exponential, evaluating a single query is at most doubly exponential. Full evaluation then requires evaluating at most doubly exponentially many such queries; hence, database evaluation and strict entailment computation is no worse than doubly exponential time. □

Proposition 22. *Assuming the logical ontology was written in clausal form, includes one variable and contains no object constants, strict entailment computation on the browser runs in at worst singly exponential time in the size of the logical ontology and the size of the logical widget assignment.*

Proof. The run-time is the cost of evaluating the database queries produced by Algorithm 1 over the database constructed by Algorithm 2. The resolution closure in this case produces a singly-exponential number of clauses of length linear in p, the number of predicates in the ontology. Just as in the previous proof, the cost of looking up the consistency check is negligible; thus, the cost of evaluating a single database query is at worst an exponential where the base depends on the size of the current widget assignment, and the exponent depends on the length of the query, p. Thus, evaluating a single query is at most singly exponential, and full evaluation requires evaluating at most singly exponentially many such queries; hence, database evaluation and strict entailment computation is no worse than single exponential time. □

These run-time results ensure that for the case of a single variable and no objects, the browser-side computation of strict entailment runs in optimal time, unless $\mathrm{P} = \mathrm{coNP}$.

Corollary 5. *When the logical ontology is in clausal form, contains one variable, and includes no object constants, browser-side strict entailment computation runs in optimal time, as long as* $\mathrm{P} \neq \mathrm{coNP}$.

Proof. The complexity of strict entailment is coNP-hard when the ontology is in clausal form and includes a single variable but no object constants. Thus, as long as $\mathrm{P} \neq \mathrm{coNP}$, the optimal algorithm for computing strict entailment is singly exponential. The proposition above ensures that our strict entailment computation runs in singly exponential time; hence, it runs in optimal time. □

Proposition 23 (Polynomial Parameterized Complexity). *The parameterized complexity of computing strict entailment is polynomial when the ontology is fixed.*

Proof. By fixing the ontology, the size of both the database queries and the data manipulation statements for extending a logical widget assignment are fixed; thus, the only variable size parameter is the web form data, and the polynomial data complexity of database evaluation is well-known. □