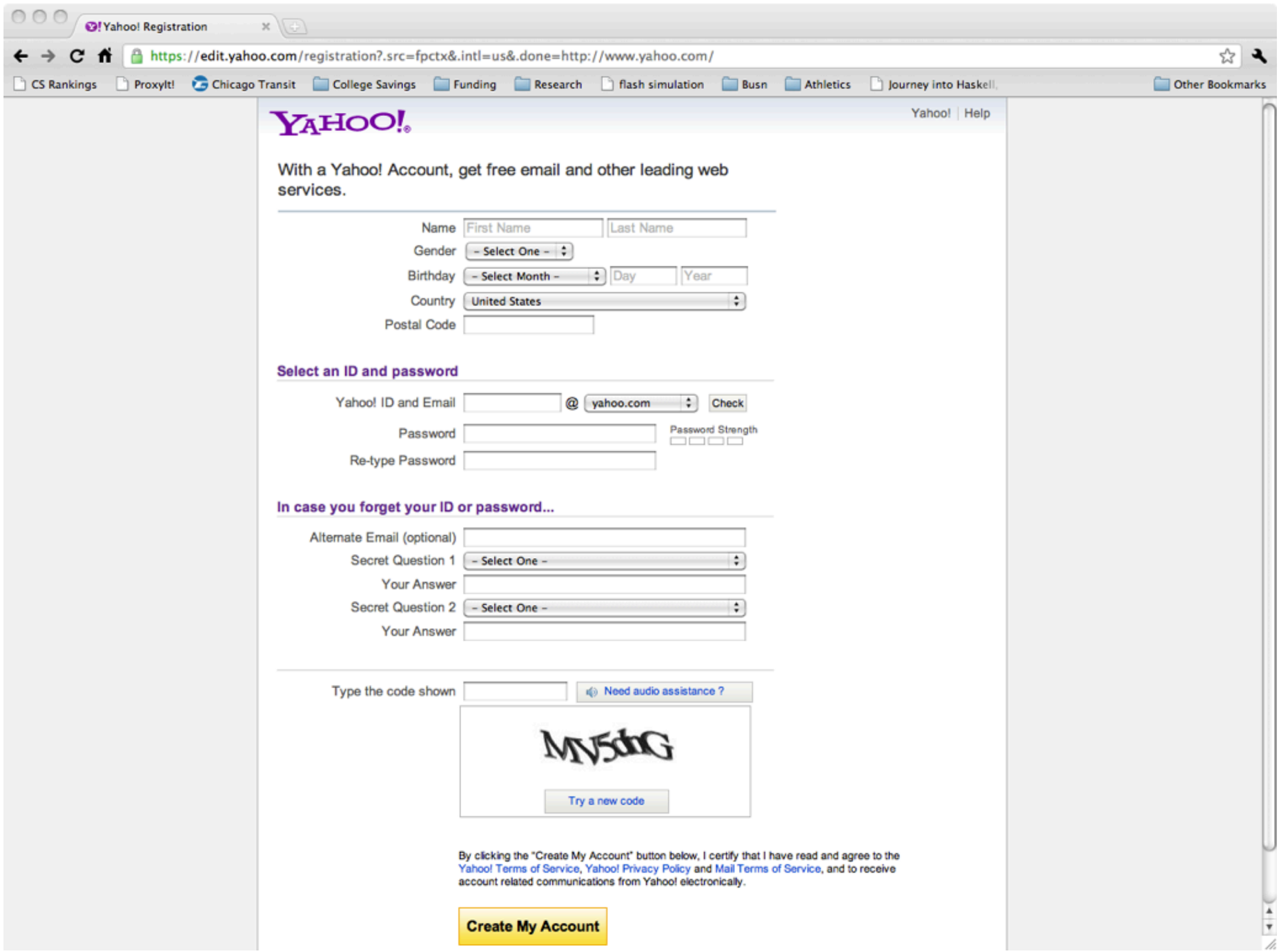


Plato: A Compiler for Web Forms

Tim Hinrichs
University of Chicago and
University of Illinois at Chicago

Interactive Web Forms are the Norm

Today's users expect web forms with two pieces of functionality:

- Implied value computation: given the user's current information, what other information is obviously intended?
- Error Detection: which user-provided information is erroneous?

Benefits of in-browser implementation:

- Rich user experience
- Reduced load on server and network

Live Demo

- Samples of interactive web forms.

Automated Reasoning and Web Forms

Building a web form can be complicated.

- HTML/CSS for layout, colors, fonts, etc. is easy.

- JavaScript for detecting errors and computing implied values is hard, *e.g.* feature-oriented programming, configuration management.

Error detection: SAT solving specialized to the web form.

Implied value computation: theorem proving (TP) specialized to the web form.

A version of TP that tolerates inconsistency: paraconsistent TP.

Plato: A Declarative Approach to Web Forms

Declarative approach

- Web programmer uses formal logic to describe acceptable web form data: *web form ontology*.
- Web programmer invokes a compiler (Plato) on the ontology to construct error-detection and implied-value computation code automatically.

Benefits

- Writing/maintaining an ontology is easier than JavaScript.
- Error-prone specialization of SAT/TP is performed automatically.
- Choice of paraconsistent TP can be left up to experts (compiler writers).

Live Demo 2

- Ontologies for earlier samples.

Outline

- **Logical Foundations of Web Forms**
- Plato's Architecture
- Plato's Compilation Algorithms
- Evaluation
- Related Work and Conclusions

Logical Foundations of Web Forms

Notation:

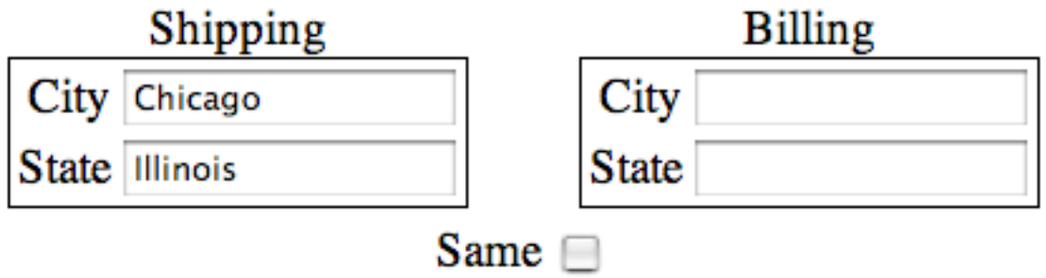
- F is a set of web form fields, e.g. shipCity, shipState, shipZip, billCity, billState, billZip.
- ∑ is some character set, e.g. Latin-1 or UTF-8.

Definition (Payload): A web form payload is a subset of F x $\sum^*$. Logically, a payload is a set of ground atoms f(v) where f in F and v in $\sum^*$.

Definition (Ontology): A web form ontology is a logical axiomatization of a set of payloads -- the acceptable payloads.

Today, we consider ontologies written in a fragment of first-order logic.

Shipping and Billing Addresses



Payload: $\{shipCity(Chicago), shipState(Illinois)\}$

Ontology: $$same \Leftrightarrow \forall x. \left(\bigwedge \begin{array}{l} shipCity(x) \Leftrightarrow billCity(x) \\ shipState(x) \Leftrightarrow billState(x) \end{array} \right)$$

Logical Foundations of Web Forms 2

- **Definition (Consistent)**: A payload is *consistent* if it is a subset of an acceptable payload. All other payloads are *inconsistent*. A payload is *minimally inconsistent* if it is inconsistent and no subset is inconsistent.

- **Definition (Error)**: There is one *error* in a payload for every minimally inconsistent payload contained within it.

- **Definition (Traditional Implication)**: Suppose P is a consistent payload for ontology Δ. P implies value v for field f if every consistent superset of P includes f(v).

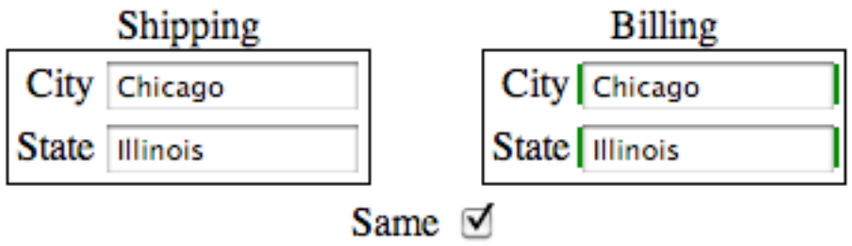
$$P \models^{\Delta} f(v)$$

- **Definition (Paraconsistent Implication)**: Suppose P is an inconsistent payload for ontology Δ. P implies value v for field f if there is some $P_0 \subset P$ that is consistent and implies f(v).

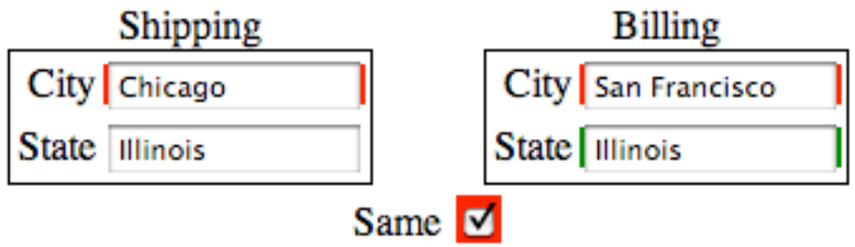
$$P \models^{\Delta}_{E} f(v)$$

- **Definition (Implied value)**: Suppose P is the payload comprised of the user-supplied key-value pairs on a web form. The form implies f(v) exactly when P paraconsistently implies f(v).

Shipping and Billing Addresses 2



- Payload: {shipCity(Chicago), shipState(Illinois),same}
- Error: none
- Implied values: billCity(Chicago) billState(Illinois)
- Consistent/inconsistent



- Add to payload: billCity(San Francisco)
- Error: {shipCity(Chicago), same, billCity(San Francisco)}
- Implied values under $\models^{\Delta}$ *<everything>*
- Implied values under $\models^{\Delta}_{E}$ billState(Illinois)

Monadic First-order Logic

A web form payload is a set of statements in monadic first-order logic.
(All relations have at most 1 argument.)

{billCity(Chicago), billState(Illinois)}

Ontology language for today:
monadic, function-free, quantifier-free, equality-free, first-order logic (MON for short)

Syntax

- V: a countable set of variables
  O: a set of object constants (that includes $\sum^*$)
  R: a set of propositions
  P: a set of relation constants
- term T ::= V | O
- sentence S ::= R | p(T) | S $\wedge$ S | S $\vee$ S | $\neg$ S | S $\Rightarrow$ S | S $\Leftarrow$ S | S $\Leftrightarrow$ S
- free variables are implicitly universally quantified

Semantics: standard first-order semantics.

Computational Complexity of Implication

Theorem: Suppose Δ is in MON and P is a finite set of ground atoms. The complexity of $\models^{\Delta}_{E}$ is as follows.

- $\prod_2$-hard and included in $\prod_3$.

- $\prod_1$-hard and $\sum_1$-hard and included in $\prod_2$ if the number of variables in Δ is bounded by a constant.

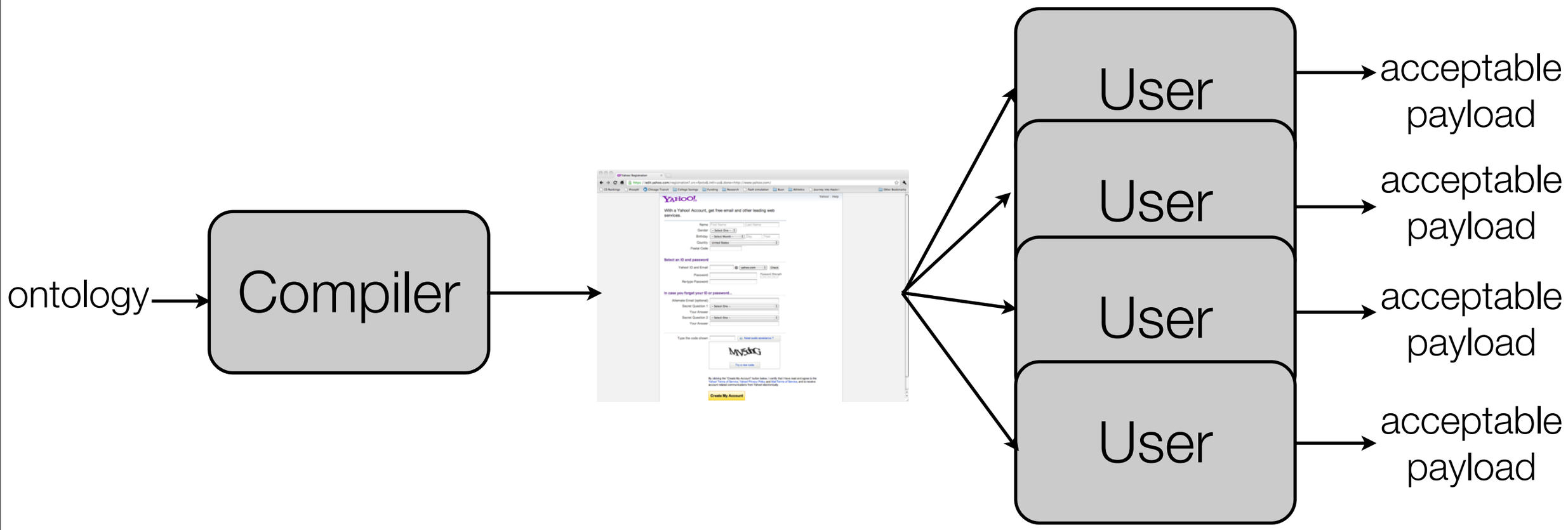
- $\prod_1$-hard if Δ is in clausal form and contains 1 variable.

- AC^0 (included in P) if Δ is of constant size.

Outline

- Logical Foundations of Web Forms
- **Plato's Architecture**
- Plato's Compilation Algorithms
- Evaluation
- Related Work and Conclusions

Compile Time vs. Run Time



Compile Time

- Compiler used by web programmer once.
- Web programmer will wait minutes.

Run Time

- Form used by many people.
- Each person generates many SAT/TP queries.
- People expect real-time results.

Web Form Skeleton

OnChange(field f)

```
1:    F := dependentFields(f)

// Compute errors       (some bookkeeping not shown)
2:    for each field g in F
3:        if hasvalue(g) then
4:            if error_g() then paintred(g)

// Compute implied values
5:    for each field g in F
6:        if !(hasvalue(g)) then
7:            V := impliedValues_g()
8:            setValues(g,V)
9:            if V ≠ ∅ then paintgreen(g)
```

Relies on two functions for each form field: $error_g$ and $impliedValues_g$

Compiler Architecture

Traditional approach: write a SAT solver/a paraconsistent TP in JavaScript.

Drawback: the same ontology analysis is performed repeatedly but with different data.

Plato: construct JavaScript SAT/TP specialized to the ontology.

Benefits: (i) database compilation techniques well-known, (ii) (non-recursive) database queries are sentences in first-order logic.

Web Form Implication Compiler

Definition (Web Form Implication Compiler):

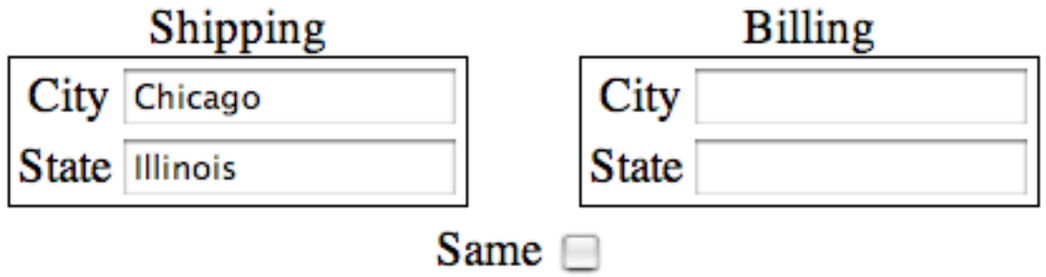
A web form ontology compiler is a function α that maps an ontology and a set of predicates to a set of first-order formulae.

α is a compiler if for any ontology Δ, predicate set F, and predicate f ∈ F, there is a sentence φ_f (impliedValues$_f$) in α[Δ,F] such that for any payload P and any object v

$$P \models_E^{\Delta} f(v) \text{ if and only if } \models_P \phi_f(v)$$

On the left, P is treated with the open world assumption.
On the right, P is treated with the closed world assumption.

Shipping and Billing Addresses 3



Ontology:

$$same \Leftrightarrow \left(\bigwedge \begin{array}{l} shipCity(x) \Leftrightarrow billCity(x) \\ shipState(x) \Leftrightarrow billState(x) \end{array} \right)$$

Compilation for fields {shipCity,shipState,same}:

$$\phi_{shipCity}(x) \equiv same \wedge billCity(x)$$

$$\phi_{shipState}(x) \equiv same \wedge billState(x)$$

$$\phi_{same} \equiv \forall x. \left(\bigwedge \begin{array}{l} shipCity(x) \Leftrightarrow billCity(x) \\ shipState(x) \Leftrightarrow billState(x) \end{array} \right)$$

Outline

- Logical Foundations of Web Forms
- Plato's Architecture
- **Plato's Compilation Algorithms**
- Evaluation
- Related Work and Conclusions

Overview

implCompile: Algorithm for constructing impliedValues_f for each form field f.

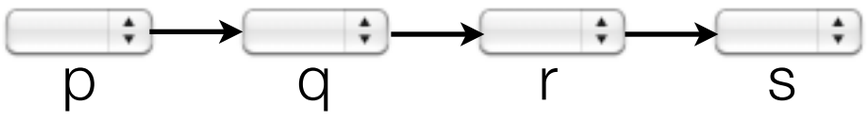
errCompile: Algorithm for constructing error_f for each form field f.

compression: Algorithm to compress ontology before compilation.

Traditional Implication via Database queries

Example 1.

$$p(x) \Rightarrow q(x)$$
$$q(x) \Rightarrow r(x)$$
$$r(x) \Rightarrow s(x)$$

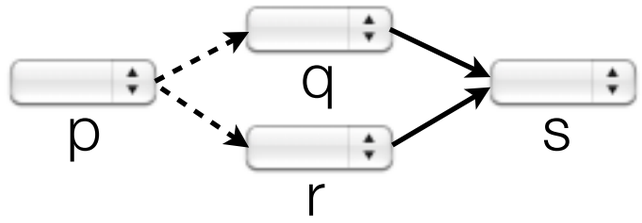


Database queries for implication.

$$\phi_p(x) \equiv \bot$$
$$\phi_q(x) \equiv p(x)$$
$$\phi_r(x) \equiv p(x) \vee q(x)$$
$$\phi_s(x) \equiv p(x) \vee q(x) \vee r(x)$$

Example 2.

$$p(x) \Rightarrow q(x) \vee r(x)$$
$$q(x) \Rightarrow s(x)$$
$$r(x) \Rightarrow s(x)$$



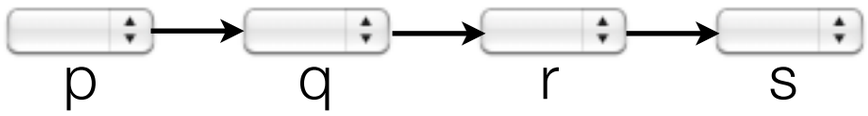
Database queries for implication.

$$\phi_p(x) \equiv \bot$$
$$\phi_q(x) \equiv \bot$$
$$\phi_r(x) \equiv \bot$$
$$\phi_s(x) \equiv p(x) \vee q(x) \vee r(x)$$

Paraconsistent Implication via Database Queries

Example 3.

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



Database queries for paraconsistent implication.

$$\phi_p(x) \equiv \bot$$
$$\phi_q(x) \equiv p(x) \wedge consistent_{p(x)}(x)$$
$$\phi_r(x) \equiv \left(\bigvee \begin{array}{l} p(x) \wedge consistent_{p(x)}(x) \\ q(x) \wedge consistent_{q(x)}(x) \end{array} \right)$$
$$\phi_s(x) \equiv \left(\bigvee \begin{array}{l} p(x) \wedge consistent_{p(x)}(x) \\ q(x) \wedge consistent_{q(x)}(x) \\ r(x) \wedge consistent_{r(x)}(x) \end{array} \right)$$

Compilation Algorithm for Implied Values

Name: implCompile
Inputs: Δ, a web form ontology in MON.
F, a set of web form fields (predicates).
Outputs: a set of first-order equivalences for computing paraconsistent implications.
Summary: (i) Compute definite Horn consequences, (ii) add consistency checks,
(iii) apply predicate completion

1: $\Delta := \text{resolutionClosure}[\Delta]$
2: $\Gamma_p := \varnothing$ **for each** predicate p
3: **for each** clause c in Δ
4: **for each** contrapositive d of c
5: write d as $p(x) \Leftarrow \psi(x,\mathbf{y})$
6: **if** $p \in F$ **and** ¬ does not occur in $\psi(x,\mathbf{y})$
7: $\Gamma_p := \Gamma_p \cup \{\psi(x,\mathbf{y}) \wedge \text{consistent}_{\psi(x,\mathbf{y})}(x,\mathbf{y})\}$
8: return $\varphi_p \equiv \bigvee \Gamma_p$ **for each** predicate p

Example

1. Resolution Closure

$$\neg p(x) \vee q(x) \vee r(x)$$
$$\neg p(x) \vee s(x) \vee r(x)$$
$$\neg p(x) \vee q(x) \vee s(x)$$
$$\neg p(x) \vee s(x)$$
$$\neg q(x) \vee s(x)$$
$$\neg r(x) \vee s(x)$$

2. Definite Horn Rules

$$s(x) \Leftarrow p(x)$$
$$s(x) \Leftarrow q(x)$$
$$s(x) \Leftarrow r(x)$$

3. Consistency checks and Predicate completion

$$\phi_p(x) \equiv \bot$$
$$\phi_q(x) \equiv \bot$$
$$\phi_r(x) \equiv \bot$$
$$\phi_s(x) \equiv \left(\bigvee \begin{array}{l} p(x) \wedge consistent_{p(x)}(x) \\ q(x) \wedge consistent_{q(x)}(x) \\ r(x) \wedge consistent_{r(x)}(x) \end{array} \right)$$

Soundness and Completeness

Theorem: implCompile is a web form implication compiler, *i.e*. for any ontology Δ, predicate set F, and predicate $f \in F$, φ_f in ImplCompile[Δ,F] ensures that for any payload P and any object v

$$P \models_E^{\Delta} f(v) \text{ if and only if } \models_P \phi_f(v)$$

Consistency checks

Definition (consistent$_{\psi(x)}$): Suppose Δ is the web form ontology. consistent$_{\psi(x)}$ (**v**) is true if and only if {ψ(**v**)} U Δ is consistent.

All of the consistency checks are of the form

$$consistent_{p_1(x_1)\wedge\cdots\wedge p_n(x_n)}(x_1,\ldots,x_n)$$

where the p_is are web form fields.

The consistency check for $p_1(v_1) \wedge \ldots \wedge p_n(v_n)$ is true exactly when the payload $\{p_1(v_1), \ldots, p_n(v_n)\}$ is error-free.

Since the web form computes errors (before computing implied values), the consistency checks amount to simple subset checks of the errors.

Compilation Algorithm for Errors

Name: errCompile
Inputs: Δ, a web form ontology in MON.
F, a set of web form fields (predicates).
Outputs: a set of first-order equivalences for computing (non-minimal) errors
Summary: (i) Compute anti-definite Horn consequences, (ii) apply predicate completion

1: Δ := resolutionClosure[Δ]
2: $\Gamma_p := \varnothing$ **for each** predicate p
3: **for each** clause c in Δ
4: **for each** contrapositive d of c
5: write d as $\neg p(x) \Leftarrow \psi(x,\mathbf{y})$ // Negative literal
6: **if** $p \in F$ **and** ¬ does not occur in $\psi(x,\mathbf{y})$
7: $\Gamma_p := \Gamma_p \cup \{\psi(x,\mathbf{y})\}$ // Dropped consistency check
8: return $\varphi_p \equiv \bigvee \Gamma_p$ **for each** predicate p

Ontology Compression

Before compiling, Plato sometimes compresses the ontology with the hope of reducing the cost of the resolution closure.

Original Ontology:

$$\bigvee \begin{array}{l} p(a) \wedge q(b) \wedge r(c) \\ p(b) \wedge q(d) \wedge r(e) \\ p(d) \wedge q(c) \wedge r(a) \end{array}$$

Compressed Ontology:

$$p(x) \wedge q(y) \wedge r(z) \Rightarrow t(x, y, z)$$

where *t* is a new database table:

a	b	c
b	d	e
d	c	a

Outline

- Logical Foundations of Web Forms
- Plato's Architecture
- Plato's Compilation Algorithms
- **Evaluation**
- Related Work and Conclusions

Evaluation of Ontology Language

Successes

- Built a handful of web form ontologies by hand.
- Wrote a translator from a language for configuration management problems (Configit) into MON.

Limitations

- String-level constraints, *e.g*. date field has the format mm/dd/yyyy, or password field must contain at least 6 characters.
- Structural constraints, *e.g*. each paper author has a first name, last name, and email; there can be arbitrarily many authors.

Evaluation of Compiler

Analytical: size of resolution closure (without compression)

Compiler

- Run-time is polynomial in the size of the resolution closure of the ontology.
- Output complexity is polynomial in the size of the resolution closure.

Code the compiler generates

- Size is polynomial in size of the resolution closure.
- Run-time is a single-exponential factor of the size of the resolution closure.

Empirical: impact of compression on resolution closure

Analytical Evaluation of Compiler

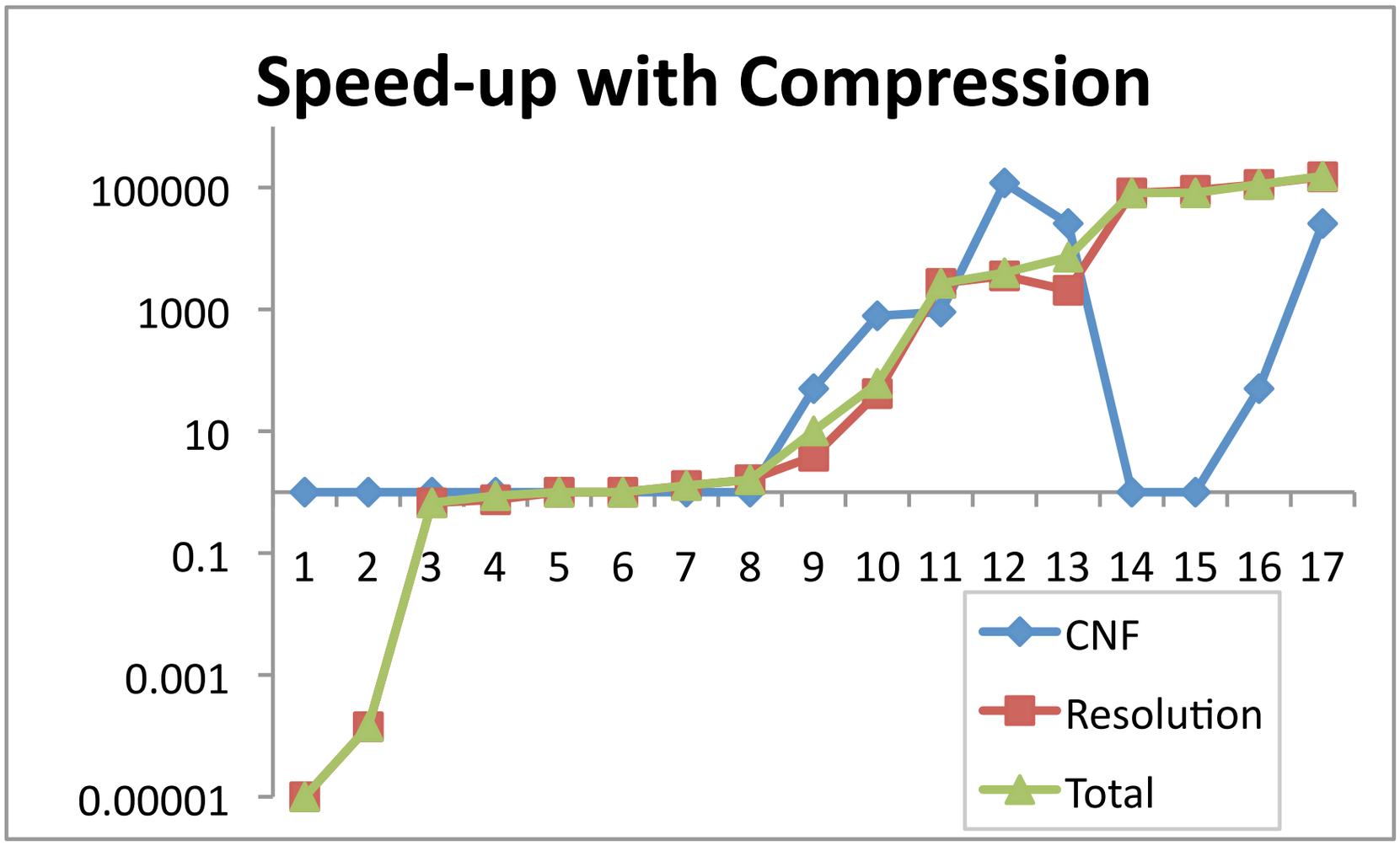
Proposition: The output complexity of the resolution closure for MON is EXPSPACE-hard and included in 2EXPSPACE. When the premises are in clausal form, contain exactly one variable, and include no object constants, the output complexity if EXPSPACE-complete.

Proposition: For any class of MON for which resolution's output complexity is included in EXPSPACE, Plato produces time-optimal implementations of implied-value computation and error detection, unless P=NP.

Corollary: Plato produces time-optimal implementations for ontologies written in clausal form with one variable and no object constants, unless P=NP.

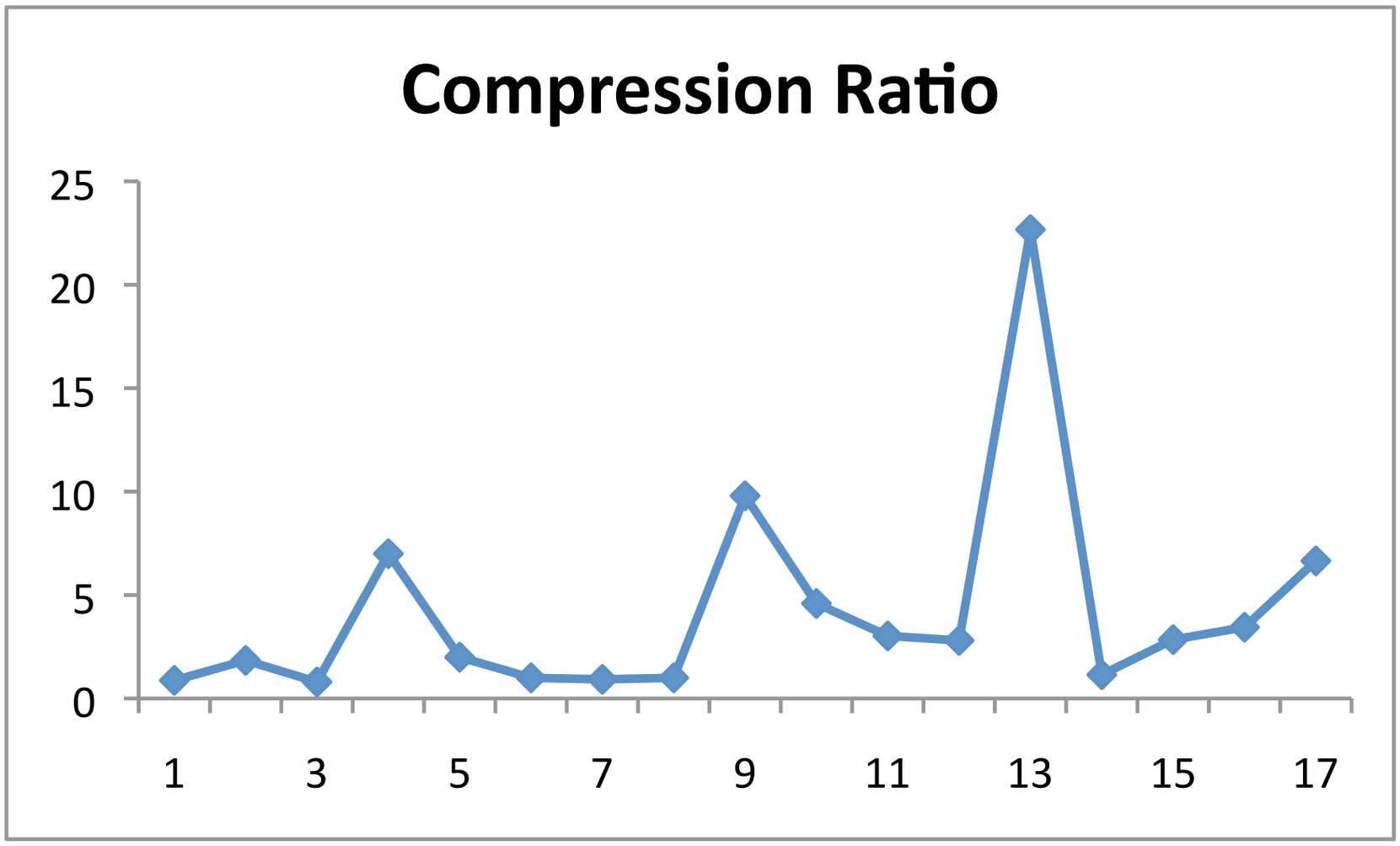
Empirical Evaluation of Compiler

- Ran resolution on compressed and uncompressed ontologies; 17 finished.
- Ontologies drawn from CLib configuration management problems.



Empirical Evaluation of Compiler

- Ran resolution on compressed and uncompressed ontologies; 17 finished.
- Ontologies drawn from CLib configuration management problems.



Outline

- Logical Foundations of Web Forms
- Plato's Architecture
- Plato's Compilation Algorithms
- Evaluation
- **Related Work and Conclusions**

Related Work

Related work found in: web engineering, computer security, formal methods, programming languages, databases, artificial intelligence, configuration management.

Most work either disallows errors, does not support implication, or utilizes a version of paraconsistent implication that does not reduce to traditional implication without errors (values are only propagated in certain directions).

Exceptions:

- [Kassoff-Genesereth, 2007] used our verison of paraconsistent implication but did not introduce compilation algorithms, complexity results, or compression.
- [Vlaeminck-Vennekens-Denecker, 2009] utilized omni-directional paraconsistent implication, but focused on approximate inference instead of compilation.
- [Hinrichs-Kao-Genesereth, 2009] focused on databases but did not leverage web form environment for consistency checks, include complexity results, or discuss compression.

Future Work

- Expanded ontology language and accompanying compilation.
- Expand to multiple ontologies for describing web forms, *e.g*. conflict resolution, layout, dynamic behavior.
- For web form front-ends to relational databases, extract web form ontology from database integrity constraints automatically.
- AJAX for checking constraints dependent on data in database that cannot be shipped to client.
- Expand from single web form to series of forms and their interaction with server.