LETTER

Extending LogicWeb via Hereditary Harrop Formulas

Keehang KWON[†a)], *Nonmember and* **Dae-Seong KANG**[††], *Member*

SUMMARY We propose HHWeb, an extension to LogicWeb with hereditary Harrop formulas. HHWeb extends the LogicWeb of Loke and Davison by allowing goals of the form $(\exists x_1 \ldots \exists x_n D) \supset G$ (or equivalently $\forall x_1 \ldots \forall x_n (D \supset G)$) where D is a web page and G is a goal. This goal is intended to be solved by instantiating $x_1, \ldots, x_n$ in D by new names and then solving the resulting goal. The existential quantifications at the head of web pages are particularly flexible in controlling the visibility of names. For example, they can provide scope to functions and constants as well as to predicates. In addition, they have such simple semantics that implementation becomes more efficient. Finally, they provide a client-side interface which is useful for customizing web pages.

key words: *prolog, semantic Web, interface*

1. Introduction

Software engineers have focused too much on the deterministic and functional paradigm (ML, C, Java) to build programs. Many programs, even when they are nondeterministic in nature, have been developed in this functional paradigm. This situation is rather unfortunate, as nondeterministic programs are everywhere from graph algorithms to AI games including Chess.

It is desirable that programs be written in a bigger paradigm, *i.e.*, a nondeterministic, relational paradigm. One successful attempt towards this direction is LogicWeb [1]. LogicWeb is especially attractive as it is a distributed, public, relation-based programming paradigm. It is also an integral part of Semantic Web [2]. Despite much attractiveness, LogicWeb have traditionally lacked elegant devices for structuring the space of names. Lacking such devices as local constants and a notion of interface, structuring the space of name in LogicWeb relies on awkward devices such as resources modules. Those devices use nonlogical constructs such as visible/1 and export/1 declarations to provide access control to predicates. One major problem with the nonlogical constructs is that the meaning of the resulting hybrid language becomes obscure and complicated. For example, it becomes difficult to define a notion of the equivalence of two LogicWeb pages.

This paper proposes HHWeb, an extension to LogicWeb with hereditary harrop formulas [3], [4]. This logic extends Horn clause goals by two major constructs: an implication goal of the form $D \supset G$ and the expression of the form $\forall x\ G$ where D is a web page and G is a goal. The former one has the following intended semantics: the rules in D are intended to be added to the current program in the course of proving G. This expression thus supports the idea of modules. The latter expression has the following intended semantics: the variable x in G is intended to be replaced with a new name before proving G. This expression thus supports the idea of local constants.

Combining these two goals leads to a goal of the form $\forall x_1 \ldots \forall x_n (D \supset G)$ (or equivalently $(\exists x_1 \ldots \exists x_n D) \supset G$, if x is not free in G) where D is a web page and G is a goal. This goal is particularly flexible in controlling the visibility of names. To be precise, they can provide scope to functions, constants, and predicates via such a simple operation as "renaming". We focus here on the goals of the form $(\exists x_1 \ldots \exists x_n D) \supset G$ because only this is cental to the later discussions of the interface notion.

In this paper we present the syntax and semantics of this extended language, show some examples of its use and study the interactions among the newly added constructs. In our presentation, we focus on first-order logic. We make this choice so as to simplify the presentation.

The remainder of this paper is structured as follows. We describe HHWeb based on first-order hereditary Harrop formulas in the next section. In Sect. 3, we present some examples of HHWeb. Section 4 concludes the paper.

2. The Language

The language we use is a slightly expanded version of hereditary Harrop formulas. It is described by G-, D- and E-formulas given by the syntax rules below:

$$G ::= A \mid G \wedge G \mid G \vee G \mid E \supset G \mid \forall x\ G \mid \exists x\ G$$

$$D ::= A \mid G \supset A \mid \forall x\ D \mid D \wedge D$$

$$E ::= D \mid \exists x E$$

In the context of HHWeb, terms are augmented with web page URLs. In the rules above, A represents an atomic formula. A D-formula is called a program clause or a first-order hereditary Harrop formula. An E-formula is called an existentially quantified program clause.

In the transition system to be considered, G-formulas will function as queries and a set of D-formulas will constitute programs. For this reason, we refer to a G-formula as

Manuscript received July 23, 2007.
Manuscript revised February 5, 2008.

[†]The author is with Computer Eng., DongA University, Korea.
[††]The author is with Electronics Eng., DongA University, Korea.
a) E-mail: khkwon@dau.ac.kr

DOI: 10.1093/ietisy/e91–d.6.1827

a goal, to a set of D-formula as a program. Our language is an extension to first-order Horn clause with the main difference that new scoping constructs are added in G-formulas and D-formulas.

We will present an operational semantics for this language. These rules in fact depend on the top-level constructor in the expression and have the effect of producing a new expression and a new program.

The rules for solving queries in our language are based on "goal-directness" in the sense that the next rule to be used depends on the top-level construct of the goal formula.

Definition 2.1. Let G be a query and let $\mathcal{P}$ be a finite set of program clauses. Then the notion of proving $\langle \mathcal{P}, G \rangle$ is defined as follows:

(1) If G is an atom and is identical to an instance of a program clause in $\mathcal{P}$, then $\langle \mathcal{P}, G \rangle$ is proved.

(2) If G is an atom and an instance of a program clause in $\mathcal{P}$ is of the form $G_1 \supset G$, prove $\langle \mathcal{P}, G_1 \rangle$.

(3) If G is $G_1 \wedge G_2$, then prove $\langle \mathcal{P}, G_1 \rangle$ and $\langle \mathcal{P}, G_2 \rangle$.

(4) If G is $G_1 \vee G_2$, then prove either $\langle \mathcal{P}, G_1 \rangle$ or $\langle \mathcal{P}, G_2 \rangle$.

(5) If G is $\exists x G_1$, then prove $\langle \mathcal{P}, [t/x]G_1 \rangle$ where t is a term.

(6) If G is $\forall x G_1$, then prove $\langle \mathcal{P}, [a/x]G_1 \rangle$ where a is a new constant.

(7) If G is $D \supset G_1$, then prove $\langle \{D\} \cup \mathcal{P}, G_1 \rangle$.

(8) If G is $(\exists x E) \supset G_1$, then prove $\langle \mathcal{P}, ([a/x]E) \supset G_1 \rangle$, where a is a new constant.

In the above rules, the symbols $\supset$ and $\forall x$ provide scoping mechanisms: they allow, respectively, for the augmentation of the program and the introduction of new names in the course of proving a goal. The $\exists$ construct in E-formulas provides a means for information hiding.

3. HHWeb

In our context, a HHWeb page corresponds simply to a E-formula with a URL. The module construct *mod* allows a URL to be associated to a E-formula. An example of the use of this construct is provided by the following "graph" module which contains some basic graph-handling rules.

mod(*www.krx.com/graph*).
htext(*www.krx.com/graphdoc.html*). % web page
path(X, Y) : – *edge*(X, Y).
path(X, Y) : – *edge*(X, Z), *path*(Z, Y).
spantree$(Tree)$: – ....

Our language in Sect. 2 permits constant names to be made local to a D-formula using the $\exists$ construct. This allows for a *server-side interface*, which leads to the hiding of a data structure in a page. The names of the constants listed then become unavailable outside the module. An example of the use of this construct is provided by the modified page which made *spantree* local:

mod(*www.krx.com/graph*1).
htext(*www.krx.com/graph*1*doc.html*). % web page
$\exists$ *spantree www.krx.com/graph*.

Now let us consider the *fly* page which contains the flight information and the *tour* page which uses the *graph*1 and *fly* pages.

mod(*www.kair.com/fly*).
htext(*www.kair.com/flydoc.html*). % web page
% pilot(flight no, name)
% flight(flight no, origin, destination)
pilot(31, *tom*).
flight(31, *paris*, *nice*).
flight(20, *nice*, *tokyo*).
flight(21, *seoul*, *rome*).
edge(X, Y) : – *flight*$(\_, X, Y)$.

mod(*www.htour.com/tour*).
htext(*www.htour.com/tourdoc.html*). % web page
% tourguide(age, name)
% tour(origin, destination)
tourguide(31, *tom*).
tour(X, Y) : – *www.kair.com/fly* $\supset$
*www.krx.com/graph*1 $\supset$
path(X, Y).

These pages can be made available in specific contexts by explicitly mentioning the URL via a hyperlink. For example, consider a goal *www.htour.com/tour* $\supset$ *tour*(*paris*, *tokyo*). Solving this goal has the effect of adding the rules in *tour*, *fly* and *graph*1 — after replacing *spantree* with a new name in *graph*1 — to the program before evaluating *path*(*paris*, *tokyo*). Thus, the name *spantree* is not visible in the program.

Unfortunately the above program is actually problematic: there is a problem of name clash. For example, *tom* appears both in the *fly* and *tour* pages. One novel way of handling name clashes is the *client-side interface*. For example, the client-side interface makes it possible for a client to customize the server page. The following is an improvement of the *tour* page based on this idea.

mod(*www.htour.com/tour*1).
htext(*www.htour.com/tourdoc.html*). % web page
% tourguide(age, name)
% tour(origin, destination)
tourguide(31, *tom*).
tour(X, Y) : – ($\exists$*tom www.kair.com/fly*) $\supset$
*www.krx.com/graph*1 $\supset$
path(X, Y).

In the above, it is interesting to note that the *tour*1 page limits the scope of *tom* via an expression $\exists$*tom www.kair.com/fly* so that its scope is local to the *fly* page.

4. Conclusion

In this paper, we have considered an extension to LogicWeb with hereditary Harrop formulas. This extension allows goals of the form $(\exists x_1 \ldots \exists x_n D) \supset G$ where D is a web page and G is a goal. The existential quantifications at the head of web pages are particularly flexible in controlling the visibility of names. For example, they can provide scope to functions and constants as well as to predicates.

Logic programming paradigm does not support imperative features such as sequential computation, global state and memory addressing. These features are certainly desirable in terms of efficiency. Our ultimate interest is in a language which extends hereditary Harrop formulas with imperative features.

Acknowledgements

This paper was supported by Dong-A University Research Fund in 2008.

References

[1] S.W. Lok and A. Davison, "Logic programming with the WWW," Proc. 7th ACM Conference on Hypertext, ACM Press, 1996.
[2] J. Davies, D. Fensel, and F.V. Harmelen, Towards the Semantic Web, John Wiley, 2003.
[3] D. Miller, "A logical analysis of modules in logic programming," J. Log. Program., vol.6, pp.79–108, 1989.
[4] D. Miller, G. Nadathur, F. Pfenning, and A. Scedrov, "Uniform proofs as a foundation for logic programming," Annals of Pure and Applied Logic, vol.51, pp.125–157, 1991.