Lightweight compilation of (C)LP to JavaScript

Jose F. Morales[1], **Rémy Haemmerlé**[2],
Manuel Carro[1 2], **and Manuel V. Hermenegildo**[1 2]

IMDEA Software Institute, Madrid (Spain)[1]
School of Computer Science, Technical University of Madrid (UPM), (Spain)[2]
{josef.morales,manuel.carro,manuel.hermenegildo}@imdea.org
remy@clip.dia.fi.upm.es, {mcarro,herme}@fi.upm.es

Abstract: We present and evaluate a compiler from Prolog (extensions) to JavaScript which makes it possible to use (constraint) logic programming to develop the client side of web applications while being compliant with current industry standards. Targeting JavaScript makes (C)LP programs executable in virtually every modern computing device with no additional software requirements from the point of view of the user. In turn, the use of a very high-level language facilitates the development of high-quality, complex software. The compiler is a back end of the Ciao system and supports most of its features, including its module system and extension mechanism. We demonstrate the maturity of the compiler by testing it with complex code such as a CLP(FD) library written in Prolog with attributed variables. Finally, we validate our proposal by measuring the performance of some LP and CLP(FD) benchmarks running on top of major JavaScript engines.

Keywords: Prolog; Ciao; Logic Programming System; Implementation of Prolog; Modules; JavaScript; Web

The Web has evolved from a network of hypertext documents into one of the most widely used OS-neutral environments for running rich applications —the so-called Web-2.0—, where computations are carried both locally at the browser and remotely on a server. Our ambitious objective in [MHCH12] is to enable client-side execution of *full-fledged (C)LP programs* by means of their *compilation* to JavaScript, i.e., to support essentially the full language available on the server side. Our starting point is the Ciao system [HBC+12], which implements a multi-paradigm Prolog dialect with numerous extensions through a sophisticated module system and program expansion facilities. Other approaches often put emphasis on the feasibility of the translation or on performance on small programs, while ours is focused on completeness and integration:

- We share the language front end and implement the translation by redefining the (complex) last compilation phases of an existing system. In return we support a full module system, as well as the existing analysis and source-to-source program transformation tools.
- We provide a minimal but scalable runtime system (including built-ins) and a compilation scheme based on the WAM that can be progressively extended and enhanced as required.
- We offer both high-level and low-level foreign language interfaces with JavaScript to simplify the tasks of writing libraries and integrating with existing code.

M.M. Gallardo, M. Villaret, L. Iribarne (Eds.): PROLE'2012, pp. 51-52, ISBN:978-84-15487-27-2.
Jornadas SISTEDES'2012, Almería 17-19 sept. 2012, Universidad de Almería.

Main Results and Experimental Evaluation. The new back end allows us to read and compile (mostly) unmodified Prolog programs (as well as all the Ciao extensions such as different flavors of (C)LP or functional and higher-order notation, to name a few), run real benchmarks, and, in summary, be able to develop full applications, where interaction with JavaScript or HTML is performed via Prolog libraries and client-side execution in the browser does not require manual recoding. To the extent of our knowledge, ours is the first approach and full implementation which can achieve these goals. Although raw performance is currently not our main goal, we have measured experimentally the performance over a collection of unmodified, small and medium-sized benchmarks, as an initial indication of the size of problems that are amenable to client-side execution with the current implementation. On average, we got a one order of magnitude slowdown w.r.t. the Ciao virtual machine, which is sufficient for a large range of interactive, Web-bound, non computationally-intensive tasks (such as the example Queens solver shown in Fig. 1, fully coded in Prolog). [1]

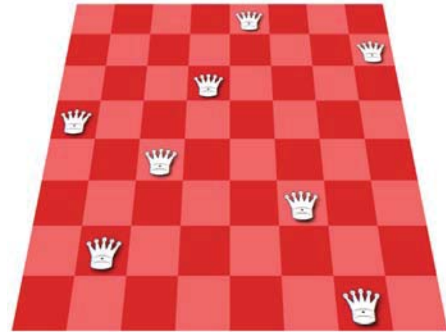

Figure 1: Graphical representation for a solution of Queens-8.

Conclusions. We believe that our system makes a significant contribution towards the practical feasibility of client-side Web applications based (fully or partially) on (constraint) logic programming, while relying exclusively on Web standards. This reliance makes it possible to execute code on a variety of devices without any need to install additional plug-ins or proprietary code. We believe this is an important advantage, specially since a good number of the currently popular portable devices make such installation hard or impossible. The system is integrated in the Ciao repository and will be included in upcoming Ciao distributions.

Acknowledgements: The research leading to these results has received funding from the Madrid Regional Government under CM project P2009/TIC/1465 (PROMETIDOS), and from the Spanish Ministry of Economy and Competitiveness under project TIN-2008-05624 *DOVES*. The research by Rémy Haemmerlé has also been supported by PICD, the Programme for Attracting Talent / young PHDs of the Montegancedo Campus of International Excellence.

Bibliography

[HBC+12] M. V. Hermenegildo, F. Bueno, M. Carro, P. López, E. Mera, J. Morales, G. Puebla. An Overview of Ciao and its Design Philosophy. *TPLP* 12(1–2):219–252, 2012. http://arxiv.org/abs/1102.5497.

[MHCH12] J. F. Morales, R. Haemmerlé, M. Carro, M. V. Hermenegildo. Lightweight compilation of (C)LP to JavaScript. *Theory and Practice of Logic Programming, 28th Int'l. Conference on Logic Programming (ICLP'12) Special Issue*, 2012. To appear.

[1] This program, as well as more examples, and benchmarks, are publicly available from http://cliplab.org/~jfran/ptojs.

Mejora del rendimiento de la depuración declarativa mediante expansión y compresión de bucles*

David Insa[1], **Josep Silva**[2], **César Tomás**[3]

[1] dinsa@dsic.upv.es
[2] jsilva@dsic.upv.es
[3] ctomas@dsic.upv.es
Universitat Politècnica de València
Camino de Vera s/n, E-46022 Valencia, Spain.

Abstract: Uno de los principales objetivos en la depuración es reducir al máximo el tiempo necesario para encontrar los errores. En la depuración declarativa este tiempo depende en gran medida del número de preguntas realizadas al usuario por el depurador y, por tanto, reducir el número de preguntas generadas es un objetivo prioritario. En este trabajo demostramos que transformar los bucles del programa a depurar puede tener una influencia importante sobre el rendimiento del depurador. Concretamente, introducimos dos algoritmos que expanden y comprimen la representación interna utilizada por los depuradores declarativos para representar bucles. El resultado es una serie de transformaciones que pueden realizarse automáticamente antes de que el usuario intervenga en la depuración y que producen una mejora considerable a un coste muy bajo.

Keywords: Depuración declarativa, Árbol de ejecución, *Tree Compression*, *Loop Expansion*

1. Introducción

La depuración es una de las tareas más difíciles y menos automatizadas de la ingeniería del software. Esto se debe al hecho de que los errores están generalmente ocultos tras complejas condiciones que únicamente ocurren en interacciones particulares de componentes software. Normalmente, los programadores no pueden considerar todas las ejecuciones posibles de su software, y precisamente esas ejecuciones no consideradas son las que producen un error. Esta dificultad inherente a la depuración es explicada por Brian Kernighan así:

> *“Everyone knows that debugging is twice as hard as writing a program in the first place. So if you’re as clever as you can be when you write it, how will you ever debug it?”*
>
> The Elements of Programming Style, 2nd edition

* Este trabajo ha sido parcialmente financiado por el *Ministerio de Economía y Competitividad (Secretaría de Estado de Investigación, Desarrollo e Innovación)* con referencia TIN2008-06622-C03-02 y por la *Generalitat Valenciana* con referencia PROMETEO/2011/052. David Insa ha sido parcialmente financiado por el Ministerio de Educación con beca FPU AP2010-4415.

M.M. Gallardo, M. Villaret, L. Iribarne (Eds.): PROLE’2012, pp. 53-67, ISBN:978-84-15487-27-2.
Jornadas SISTEDES’2012, Almería 17-19 sept. 2012, Universidad de Almería.