Scheme Project

1 Scheme Installation

The MIT/GNU Scheme development environment provides an interpreter, compiler, source-code debugger, integrated Emacs-like editor, and a large runtime library. MIT/GNU Scheme is available from http://www.gnu.org/software/mit-scheme/.

- Installation on OS X and Windows: Follow the instructions on the website.
- Installation on *nix: The MIT/GNU Scheme can be installed from the available source package using the GNU make utilities. If you do not have superuser privileges, you can specify an installation directory with the configure command (`configure --prefix=/INSTALLATIONDIR`)
- Installation on Debian based Linux distributions: MIT/GNU Scheme for x86 is also available in the software repository (package name `mit-scheme`).

2 Assignment

Compilers use algebraic simplification to improve address computations, as arithmetic overflow is not a concern in this context. The central idea is to generate sums of products. Constant expression and loop invariant computations can be combined and hoisted to a loop header [1].

In Scheme, compilers can represent arithmetic expressions in form of s-expressions. An s-expression is a recursive tree like data-structure comprised of lists. S-expressions have the form ($op\ exp_1\ exp_2$) where exp refers either to an atom (an integer constant or variable) or recursively to another s-expression. Implement algebraic simplification of s-expressions according to the rules outlined by Muchnick [1, 12.3.1]. Constants are denoted by c, other terms (i.e., variables and s-expressions) by t.

$$
\begin{aligned}
(+\ c_1\ c_2) &\rightarrow c_1 + c_2 && (1)\\
(+\ t\ c) &\rightarrow (+\ c\ t) && (2)\\
(*\ c\ _1\ c_2) &\rightarrow c_1 * c_2 && (3)\\
(*\ t\ c) &\rightarrow (*\ c\ t) && (4)\\
(-\ c_1\ c_2) &\rightarrow c_1 - c_2 && (5)\\
(-\ t\ c) &\rightarrow (+\ (-c)\ t) && (6)\\
(+\ t_1\ (+\ t_2\ t_3)) &\rightarrow (+\ (+\ t_1\ t_2)\ t_3)) && (7)\\
(+\ t_1\ (+\ t_2\ t_3)) &\rightarrow (+\ (+\ t_1\ t_2)\ t_3)) && (8)\\
(+\ (+\ c_1\ t)\ c_2) &\rightarrow (+\ (+\ c_1\ c_2)\ t) && (9)\\
(*\ (*\ c_1\ t)\ c_2) &\rightarrow (*\ (*\ c_1\ c_2)\ t) && (10)\\
(*\ (+\ c_1\ t)\ c_2) &\rightarrow (+\ (*\ c_1\ c_2)\ (*\ c_2\ t)) && (11)\\
(*\ c_1\ (+\ c_2\ t)) &\rightarrow (+\ (*\ c_1\ c_2)\ (*\ c_1\ t)) && (12)\\
(*\ (+\ t_1\ t_2)\ c) &\rightarrow (+\ (*\ c\ t_1)\ (*\ c\ t_2)) && (13)\\
(*\ c\ (+\ t_1\ t_2)) &\rightarrow (+\ (*\ c\ t_1)\ (*\ c\ t_2)) && (14)\\
(*\ (-\ t_1\ t_2)\ c) &\rightarrow (-\ (*\ c\ t_1)\ (*\ c\ t_2)) && (15)\\
(*\ c\ (-\ t_1\ t_2)) &\rightarrow (-\ (*\ c\ t_1)\ (*\ c\ t_2)) && (16)\\
(*\ (+\ t_1\ t_2)\ t_3) &\rightarrow (+\ (*\ t_1\ t_3)\ (*\ t_2\ t_3)) && (17)\\
(*\ t_1\ (+\ t_2\ t_3)) &\rightarrow (+\ (*\ t_1\ t_2)\ (*\ t_1\ t_3)) && (18)\\
(*\ (-\ t_1\ t_2)\ t_3) &\rightarrow (-\ (*\ t_1\ t_3)\ (*\ t_2\ t_3)) && (19)\\
(*\ t_1\ (-\ t_2\ t_3)) &\rightarrow (-\ (*\ t_1\ t_2)\ (*\ t_1\ t_3)) && (20)
\end{aligned}
$$

References

[1] Steven S. Muchnick. *Advanced compiler design and implementation.* Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1997.