Instruction Selection with Bottom-Up Rewriting Systems

Miloslav Trmač

Dec 10, 2007

The Problem

- "Code generation" for expression trees:
  - Converting e.g. $+(a, +(*(b, 4), 8))$ to a sequence of machine instructions
  - Needs to be fast
  - Should generate "good" (locally optimal) code
- Automatically generated from a readable machine description
- Should support machine-independent optimizing transformations
- Limitations:
  - No register allocation
  - No shared subexpressions (only trees, not DAGs)
  - Additive operation costs

An Example Rewrite System

- Loads:
  $Reg \longrightarrow reg$
  $Const \longrightarrow amode$
- Addressing modes:
  $reg \longrightarrow amode$
  $+(Const, reg) \longrightarrow amode$
- Instructions:
  $amode \longrightarrow reg$
  $biOp(amode, amode) \longrightarrow reg$
- Transformations:
  $0 \longrightarrow Const$
  $+(X, 0) \longrightarrow X$
  $+(X, Y) \longrightarrow +(Y, X)$
  $+(X, Y) \longrightarrow biOp(X, Y)$
  $-(X, Y) \longrightarrow biOp(X, Y)$

$$
\begin{aligned}
&+(0, +(Const, Const)) \longrightarrow \\
&+(0, +(Const, amode)) \longrightarrow \\
&+(0, +(amode, amode)) \longrightarrow \\
&+(0, biOp(amode, amode)) \longrightarrow \\
&+(biOp(amode, amode), 0) \longrightarrow \\
&biOp(amode, amode) \longrightarrow reg
\end{aligned}
$$

Definitions

Rewrite rule A pair $(\alpha \to \beta)$ of tree patterns.
It may contain variables—but each only once in α and at most once in β.

Position An identification of a node within a tree.

Rewrite application An application of a rewrite rule at a particular position of a particular tree.

Rewrite sequence (τ) A sequence of rewrite applications that can be applied to at least one tree.

Loop τ loops if it has two different prefixes τ_1, τ_2, and $\exists T : \tau_1(T) = \tau_2(T)$.

Reachability problem For a fixed goal tree G and a tree T, find $\tau : \tau(T) = G$, if it exists.

C-Reachability problem Each rewrite rule has an associated cost.
For T, find a τ with minimal total cost of applied rules.

Normal Form of a Rewrite Sequence

Let τ apply to T, $\tau(T) = T'$, $T = op(T_1, \ldots, T_n)$. τ is in a normal form if all of the following is true:

- It does not loop
- $\tau = \tau_1 \ldots \tau_n \tau_0$
  $\forall i \geq 1 : \tau_i$ only affects T_i
- $\tau_0(op(\tau_1(T_1), \ldots, \tau_n(T_n))) = T'$
- No rewrite application can be moved from τ_0 to other τ_i
- $\forall i \geq 1 : \tau_i$ is in normal form

τ_0 is the *local rewrite sequence* assigned by τ to the root node of T. Local rewrite sequence assigned by τ to the root node of T_i is defined by the normal form of τ_i, etc.

Note that the choice of T is irrelevant.

BURS

k-normal τ in normal form is in k-normal form if it applies to a tree T and each local rewrite sequence assigned to a node in T by τ has length at most k.

k-BURS Let R be a set of rewrite rules, L_I and L_O sets of trees.
$< R, L_I, L_O >$ has the *k-BURS property* if
$\forall T \in L_I, T' \in L_O, \forall \tau : \tau(T) = T : \tau$ has a permutation which is in k-normal form.

BURS BURS is the set of triples $< R, L_I, L_O >$ which have the k-BURS property for some k.

Testing BURS

We can determine whether $< R, L_{Op}, L_{Op} >$ is in k-BURS, where L_{Op} is the set of all trees with operators Op:

- Each local rewrite sequence must start with an rewrite application affecting the root of the subtree.
- Each subsequent rewrite application must handle a node "touched" by previous rewrite applications in the rewrite sequence.
- So, the set of local rewrite sequences is finite and can be generated.
- $< R, L_{Op}, L_{Op} >$ is in k-BURS iff there is no local rewrite sequence (without loops) of length $k + 1$.

Extent of BURS

A rule $\alpha \to \beta$ is a:

Instruction fragment rule α is a tree without variables and β is a 0-ary symbol.

Generic operator rule $\alpha = op(X_1, \ldots, X_n), \beta = op'(X_1, \ldots, X_n)$

Commutativity rule $\alpha = op(X_1, \ldots, X_n), \beta = op(X_{\pi(1)}, \ldots, X_{\pi(n)})$

Identity rule $\alpha = op(X, T), \beta = X$

Any rewrite system containing only the above types of rules is in BURS.

Some rewrite systems are not in BURS:
$a(b(X)) \to a(bb(X))$
$bb(b(X)) \to bb(bb(X))$
$bb(c) \to c$
$a(c) \to d$
Consider $a(b(b(\cdots b(c) \cdots)))$ with goal d.

Local Rewrite Graphs

For a tree T, we construct a graph:

- $\forall T_0$, such that $\exists \tau$ in normal form, $\tau = \tau_1 \dots \tau_n \tau_0$, $T_0 = \tau_n(\cdots \tau_1(T)\cdots)$, consider all local rewrite sequences: $\tau' : T_0 \to T'$.
- Let $pre(\tau', 1), \dots, pre(\tau', m) = \tau'$. Add a directed path $T_0 \Rightarrow pre(\tau', 1)(T_0) \Rightarrow \cdots \Rightarrow pre(\tau', m-1)(T_0) \Rightarrow \tau'(T_0) = T'$ to the graph. T_0 is called an *input node*.

The graph summarizes the trees reachable from T before starting a local rewrite sequence, and all trees reachable during the local rewrite sequence.

Example: $T = +(0, +(Const, Const))$
$In_1 = +(Const, reg) \Rightarrow Out_1$
$In_2 = +(0, reg) \Rightarrow +(reg, 0) \Rightarrow Out_2$
$In_3 = +(amode, amode) \Rightarrow biOp(amode, amode) \Rightarrow Out_2$
$Out_1 = amode \Rightarrow Out_2$
$Out_2 = reg \Rightarrow Out_1$

Solving "Reachability"

Given a fixed rewrite system R, a fixed goal G, and a tree T:

- Compute the LR graphs of all subtrees of T.
  (If the number of LR graphs for R and G is finite, this can be precomputed and the LR graphs can be assigned by a bottom-up tree automaton.)
- If G does not appear in the LR graph of T, fail. Otherwise, call $sub(T, G)$.
  (Checking for G can be precomputed.)

$sub(T_{in}, T_{out})$

- In the LR graph of T_{in}, select any input node $op(T'_1, \ldots, T_n)$ from which T_{out} is reachable in the LR graph, and let τ_0 be the corresponding local rewrite sequence.
  (Both can be precomputed.)
- Let $T_{in} = op(T_1, \ldots, T_n)$. $\forall i$: call $sub(T_i, T'_i)$.
- Output the rewrites specified by τ_0.

"If the number of LR graphs is finite"

Rewrite systems with finite number of LR graphs are called *finite BURS*.

For R and G, define sets of tree patterns $I_{R,G}$ and $O_{R,G}$:

- $G \in O_{R,G}$
- $\forall \beta \in O_{R,G}, \exists T$, T can be rewritten to β, and the local rewrite sequence assigned to T is $\alpha \rightarrow \beta$: Add α to $I_{R,G}$, and add all proper subtrees of α to $O_{R,G}$.

$I_{R,G}$ are input nodes in all "relevant" LR graphs. If both $I_{R,G}$ and $O_{R,G}$ is finite, $< R, G >$ is a finite BURS.

Any rewrite system containing only instruction fragment rules, generic operator rules, commutativity rules and identity rules is a finite BURS.

Implementation Considerations

- From the LR graphs we can check whether there are trees from which G is unreachable.
- We can drop unused parts of each LR graph (e.g. choose to keep only the minimum input nodes to "cover" all output nodes).
- At run-time, we don't need to store the LR graphs. For each tree ID and output tree ID, only IDs of subtree outputs and the rewrite sequence needs to be stored.
- This can, in turn, allow replacing equivalent "LR graph extracts" by a single one—but that depends on which parts nodes in LR graphs were dropped. Minimization of number of "LR graph extracts" is NP-complete.
- "C-Reachability": Storing rewrite cost for each LR node leads to infinite number of LR graphs; costs "delta-adjusted" by subtracting the minimum cost from all costs in the graph.

For More Information

- S. Graham, E. Pelegrí-Llopart: Optimal Code Genration for Expression Trees: An Application of BURS Theory
- T. Proebsting: BURS Automata Generation
- T. Proebsting, B. Whaley: One-Pass, Optimal Tree Parsing—With or Without Trees

Thank You

Any questions?