A new language for Algebraic Dynamic Programming

Georg Sauthoff
gsauthof@techfak.uni-bielefeld.de

Universität Bielefeld
AG Praktische Informatik

27. Workshop der GI-Fachgruppe Programmiersprachen und Rechenkonzepte, 2010-05-04
Dynamic Programming

- Optimization method by Bellman (1953)
- Computing optimal solutions by using optimal solutions to sub-problems
- Tabulation of sub-solutions

⇒ Computation of an optimal solution in an exponential search space in polynomial time and space
Dynamic Programming

- Optimization method by Bellman (1953)
- Computing optimal solutions by using optimal solutions to sub-problems
- Tabulation of sub-solutions
  \[ \Rightarrow \text{Computation of an optimal solution in an exponential search space in polynomial time and space} \]

Applications

- Sequence comparison
- Secondary structure prediction
- ...
Algebraic Dynamic Programming

- Programming method by Robert Giegerich, 2000
- Separation of concerns
Algebraic Dynamic Programming

- Programming method by Robert Giegerich, 2000
- Separation of concerns

![Diagram of Algebraic Dynamic Programming](image)
Algebraic Dynamic Programming

- Programming method by Robert Giegerich, 2000
- Separation of concerns

- Ban on indices (→ „No subscripts no errors!“)
Example: Optimal Matrix chain multiplication

- **Input:** $A \in \mathbb{R}^{10 \times 100}$, $B \in \mathbb{R}^{100 \times 5}$, $C \in \mathbb{R}^{5,50}$, $X = A \cdot B \cdot C$
- **Goal:** Compute $X$ with the least number of elementary multiplications.
- **Search space** $T_1 = (AB)C$, $T_2 = A(BC)$
Example: Optimal Matrix chain multiplication

- **Input:** $A \in \mathbb{R}^{10\times100}$, $B \in \mathbb{R}^{100\times5}$, $C \in \mathbb{R}^{5,50}$, $X = A \cdot B \cdot C$
- **Goal:** Compute $X$ with the least number of elementary multiplications.
- **Search space** $T_1 = (AB)C$, $T_2 = A(BC)$
Example: Optimal Matrix chain multiplication

- Input: $A \in \mathbb{R}^{10 \times 100}, B \in \mathbb{R}^{100 \times 5}, C \in \mathbb{R}^{5,50}, X = A \cdot B \cdot C$
- Goal: Compute $X$ with the least number of elementary multiplications.
- Search space $T_1 = (AB)C, T_2 = A(BC)$
Example: Optimal Matrix chain multiplication

- **Input:** $A \in \mathbb{R}^{10 \times 100}$, $B \in \mathbb{R}^{100 \times 5}$, $C \in \mathbb{R}^{5 \times 50}$, $X = A \cdot B \cdot C$
- **Goal:** Compute $X$ with the least number of elementary multiplications.
- **Search space** $T_1 = (AB)C$, $T_2 = A(BC)$
Example: Optimal Matrix chain multiplication

Tree grammar

```
mult

matrices → matrices matrices

single

matrices → T
```
Example: Optimal Matrix chain multiplication

Tree grammar

```
mult
```

```
matrices → matrices matrices

single
```

```
matrices → T
```

Algebra

1. `single(a) = tuple(0, rows(a), cols(a))`
2. `mult(a, b) = tuple(ops(a)+ops(b)+rows(a) * cols(a) * cols(b), rows(a), cols(b))`
3. `h(l) = minimum(l)`
Old implementation

- as embedded domain specific language in Haskell
- a compiler for this eDSL (translation to C)

```
1 > matrixmult alg f = axiom matrices where
2   
3 > (single, mult, h) = alg
4   
5 > matrices = tabulated (  
6 >     single <<< achar |||  
7 >     mult <<< matrices +~+ matrices ... h)
```
Bellman’s GAP

- new Language for ADP programs
- C/Java-like Syntax
- new features
- tailored for a new compiler
Bellman’s GAP

- new Language for ADP programs
- C/Java-like Syntax
- new features
- tailored for a new compiler

**Signature**

```
signature Sig(alphabet, answer)
{
    answer single(int, char, int, char);
    answer mult(answer, answer);
    choice [answer] h([answer]);
}
```
Example

1 algebra minmulti implements Sig(alphabet = char, answer = tuple)
2 {
3    tuple single(int r, char a, int c, char b)
4    {
5        tuple x;
6        x.ops = 0;
7        x.rows = r;
8        x.cols = c;
9        return x;
10    }
11    tuple mult(tuple a, tuple b) { ... }  
12    choice [tuple] h([tuple] l)
13    {
14        return list(minimum(l));
15    }
16  }
17}
Example

Automatic algebras

1 \texttt{algebra co auto count ;}
2 \texttt{algebra en auto enum ;}
Example

Automatic algebras

1. `algebra co auto count ;`
2. `algebra en auto enum ;`

Algebra extension

1. `algebra maxmulti extends minmulti`
2. `{`
3. `choice [tuple] h([tuple] l)`
4. `{`
5. `return list(maximum(l));`
6. `}`
7. `}`
Example

Grammar

```plaintext
grammar mopt uses Sig(axiom = matrix) {
  matrix = single(INT, CHAR(',,'), INT, CHAR(',,')) |
    mult(matrix, matrix) # h ;
}
```
Bellman’s GAP Compiler

- ADP-aware type checking
- Semantic Analyses/Optimizations
  - Runtime
  - Table Design
  - Yield Size
  - Index elimination
  - List elimination
  - ...

A new language for Algebraic Dynamic Programming

Georg Sauthoff

Introduction

Bellman’s GAP

Conclusion
Bellman’s GAP Compiler

- ADP-aware type checking
- Semantic Analyses/Optimizations
  - Runtime
  - Table Design
  - Yield Size
  - Index elimination
  - List elimination
  - ...

- Codegeneration
  - Automatic parallelization for OpenMP
  - Top-Down vs. Bottom-Up evaluators
  - Different Backtracing schemes
  - Sampling
  - Window-Mode
  - ...
Parallelization
Parallelization
Results, ADPfold, mfe, $|s| = 4000$
A new language for Algebraic Dynamic Programming

Georg Sauthoff

Introduction

Bellman’s GAP

Conclusion

Results, ADPfold, mfe, $|s| = 4000$

![Graph showing parallel efficiency vs. number of CPUs for different processors.](image)
Multitrack

6 alignment = nil( < SEQ, SEQ> ) | 
7 del( < CHAR, EMPTY >, alignment) | 
8 ins( < EMPTY, CHAR > , alignment) | 
9 match( < CHAR, CHAR >, alignment) ≠ h ;

A new language for Algebraic Dynamic Programming

Georg Sauthoff

Introduction

Bellman’s GAP

Conclusion
A new language for Algebraic Dynamic Programming

Georg Sauthoff

Introduction

Bellman's GAP

Conclusion

Multitrack

6 7 \texttt{alignment} = \texttt{nil}(\langle\texttt{SEQ, SEQ}\rangle) \mid
8 \quad \texttt{del}(\langle\texttt{CHAR, EMPTY}\rangle, \texttt{xDel}) \mid
9 \quad \texttt{ins}(\langle\texttt{EMPTY, CHAR}\rangle, \texttt{xIns}) \mid
10 \quad \texttt{match}(\langle\texttt{CHAR, CHAR}\rangle, \texttt{alignment}) \# h ;
12
13 \texttt{xDel} = \texttt{alignment} \mid
14 \quad \texttt{delx}(\langle\texttt{CHAR, EMPTY}\rangle, \texttt{xDel}) \# h ;
15
16 \texttt{xIns} = \texttt{alignment} \mid
17 \quad \texttt{insx}(\langle\texttt{EMPTY, CHAR}\rangle, \texttt{xIns}) \# h ;
Multitrack

1. \( \text{skipR} = \text{sr}(\text{<EMPTY, CHAR>}, \text{skipR}) \mid \) \\
   \( \text{skipL} \neq \text{h} ; \)

2. \( \text{skipL} = \text{sl}(\text{<CHAR, EMPTY>}, \text{skipL}) \mid \) \\
   \( \text{alignment} \neq \text{h} ; \)

3. \( \text{alignment} = \text{nil}(\text{<SEQ, SEQ>}) \mid \) \\
   \( \text{del}(\text{<CHAR, EMPTY>}, \text{xDel}) \mid \) \\
   \( \text{ins}(\text{<EMPTY, CHAR>}, \text{xIns}) \mid \) \\
   \( \text{match}(\text{<CHAR, CHAR>}, \text{alignment}) \neq \text{h} ; \)

4. \( \text{xDel} = \text{alignment} \mid \) \\
   \( \text{delx}(\text{<CHAR, EMPTY>}, \text{xDel}) \neq \text{h} ; \)

5. \( \text{xIns} = \text{alignment} \mid \) \\
   \( \text{insx}(\text{<EMPTY, CHAR>}, \text{xIns}) \neq \text{h} ; \)
Index Translation Example

alignment_{i,j,k,l} = h \begin{cases} 
\text{nil}(), \text{ if } j - i = 0 \land l - k = 0 \\
\text{del}(u_i, x\text{Del}_{i+1,j,k,l}) \\
\text{ins}(v_k, x\text{Ins}_{i,j,k+1,l}) \\
\text{match}(u_i, v_k, \text{alignment}_{i+1,j,k+1,l}) 
\end{cases}
Index Translation Example

\[
\text{alignment}_{i,k} = h \begin{cases} 
\text{nil()}, & \text{if } |u| - i = 0 \land |v| - k = 0 \\
\text{del}(u_i, x\text{Del}_{i+1,k}) \\
\text{ins}(v_k, x\text{Ins}_{i,k+1}) \\
\text{match}(u_i, v_k, \text{alignment}_{i+1,k+1}) 
\end{cases}
\]
## Terminal parsers

<table>
<thead>
<tr>
<th>Parser</th>
<th>Yield Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>CHAR, CHAR(arg)</td>
<td>1</td>
</tr>
<tr>
<td>STRING</td>
<td>1</td>
</tr>
<tr>
<td>BASE</td>
<td>1</td>
</tr>
<tr>
<td>REGION</td>
<td>1</td>
</tr>
<tr>
<td>REGION0</td>
<td>0</td>
</tr>
<tr>
<td>FLOAT</td>
<td>1</td>
</tr>
<tr>
<td>INT</td>
<td>1</td>
</tr>
<tr>
<td>SEQ</td>
<td>1</td>
</tr>
<tr>
<td>EMPTY</td>
<td>0</td>
</tr>
<tr>
<td>LOC</td>
<td>0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>min</th>
<th>max</th>
</tr>
</thead>
<tbody>
<tr>
<td>CHAR, CHAR(arg)</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>STRING</td>
<td>1</td>
<td>$n$</td>
</tr>
<tr>
<td>BASE</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>REGION</td>
<td>1</td>
<td>$n$</td>
</tr>
<tr>
<td>REGION0</td>
<td>0</td>
<td>$n$</td>
</tr>
<tr>
<td>FLOAT</td>
<td>1</td>
<td>$n$</td>
</tr>
<tr>
<td>INT</td>
<td>1</td>
<td>$n$</td>
</tr>
<tr>
<td>SEQ</td>
<td>1</td>
<td>$n$</td>
</tr>
<tr>
<td>EMPTY</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>LOC</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

...
Grammar filtering

closed = \{ \text{stack} \mid \text{hairpin} \mid \text{leftB} \mid \text{rightB} \mid \text{iloop} \mid \text{multiloop} \}

with stackpairing \# h ;

iloop = il( \text{BASE, BASE, REGION with maxsize(30), closed, REGION with maxsize(30), BASE, BASE} ) \# h ;
Grammar filtering

closed = { stack | hairpin | leftB | rightB |
           iloop | multiloop }
with stackpairing # h ;

iloo p = il( BASE, BASE, REGION with maxsize(30),
closed ,
REGION with maxsize(30), BASE, BASE) # h ;

- syntactic and semantic filters

- with—overlay, e.g.
  f(A, B) with—overlay samesize
  - access on argument-subwords of a function symbol

- suchthat, suchthat—overlay
  - filter evaluated candidates
Combination of Algebras

Example

1. \texttt{minmem * minmult * pretty}
2. \texttt{shape * mfe * pretty}
Combination of Algebras

Example

```
minmem * minmult * pretty
shape * mfe * pretty
(p_func | p_func_id) * (shape5 * pretty)
suchthat sample_filter
shape5 * (mfe % p_func) * pretty
(shape5 / mfe) * pretty
```
Bellman’s GAP is used for hand-written grammars with up to 30 – 40 non-terminals. There are grammar generators that generate Bellman’s GAP code, resulting in grammars with 200 – 300 non-terminals that are not uncommon in this use case.
Bellman’s GAP is a domain specific language for specification of DP algorithms on sequences, that simplifies the development of large DP algorithms.

Bellman’s GAP compiler contains multiple semantic analyses and codegeneration options:
- The generated C++-Code is competitive with hand-optimized code.
- Reduces development time.
Thanks! & Do you have questions?
Photo images in this document are from http://commons.wikimedia.org/. I.e. they are licensed under the GNU Free Documentation License (GFDL) and/or a Creative Commons license (e.g. CC-by-sa). If you need more information about the specific license or URL of a single photo image, don’t hesitate to write an e-mail to gsauthof@techfak.uni-bielefeld.de.