Script generated by TTT

Title: Seidl: Programmoptimierung (27.01.2014)

Date: Mon Jan 27 14:15:21 CET 2014

Duration: 99:45 min

Pages: 49

SS14

Virtual Machines – the
Compiler – Axel Simon, M. Petter
Code Generation

Presburger Arithmetic == full arithmetic without multiplication

Arithmetic : highly undecidable :-(
even incomplete :-((

$\Longrightarrow$ Hilbert's 10th Problem

$\Longrightarrow$ Gödel's Theorem

Presburger Arithmetic == full arithmetic without multiplication

Arithmetic : highly undecidable :-(
even incomplete :-((

$\Longrightarrow$ Hilbert's 10th Problem

$\Longrightarrow$ Gödel's Theorem

728

Presburger Formulas over $\mathbb{N}$:

$<$

$>$

$$\begin{aligned} \phi \quad ::= \quad & x + y = z \quad | \quad x = n \quad | \\ & \phi_1 \wedge \phi_2 \quad | \quad \neg\phi \quad | \\ & \exists\, x : \ \phi \end{aligned}$$

$x \leq y \equiv \exists z : x + z = y$

Presburger Formulas over $\mathbb{N}$:

$$\begin{aligned} \phi \quad ::= \quad & x + y = z \quad | \quad x = n \quad | \\ & \phi_1 \wedge \phi_2 \quad | \quad \neg\phi \quad | \\ & \exists\, x : \ \phi \end{aligned}$$

Goal: PSAT

Find values for the free variables in $\mathbb{N}$ such that ϕ holds ...

Idea: Code the values of the variables as Words :-)

213	t	1	0	1	0	1	0	1	1	0
42	z	0	1	0	1	0	1	0	0	0
89	y	1	0	0	1	1	0	1	0	0
17	x	1	0	0	0	1	0	0	0	0

0 0 2
0 2 2

Idea: Code the values of the variables as Words :-)

213	t	1	0	1	0	1	0	1	1
42	z	0	1	0	1	0	1	0	0
89	y	1	0	0	1	1	0	1	0
17	x	1	0	0	0	1	0	0	0

Observation:

The set of satisfying variable assignments is regular :-))

$\phi_1 \wedge \phi_2$	$\Longrightarrow$	$\mathcal{L}(\phi_1) \cap \mathcal{L}(\phi_2)$	(Intersection)
$\neg\phi$	$\Longrightarrow$	$\overline{\mathcal{L}(\phi)}$	(Complement)
$\exists\, x : \phi$	$\Longrightarrow$	$\pi_x(\mathcal{L}(\phi))$	(Projection)

x, y, z, t

y, z, t

1	1	1
1	0	0
0	1	0

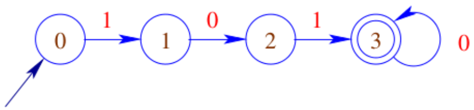
Warning:

- Our representation of numbers is not unique: 011101 should be accepted iff every word from $011101 \cdot 0^*$ is accepted!
- This property is preserved by union, intersection and complement :-)
- It is lost by projection !!!

$\Longrightarrow$ The automaton for projection must be enriched such that the property is re-established !!

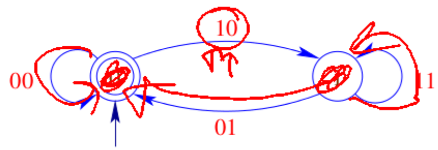
Automata for Basic Predicates:

$x = 5$



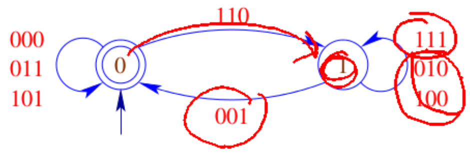
Automata for Basic Predicates:

$x + x = y$



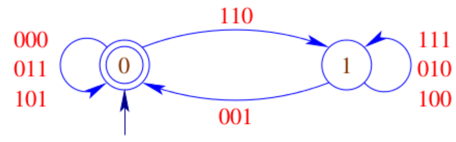
Automata for Basic Predicates:

$x+y=z$



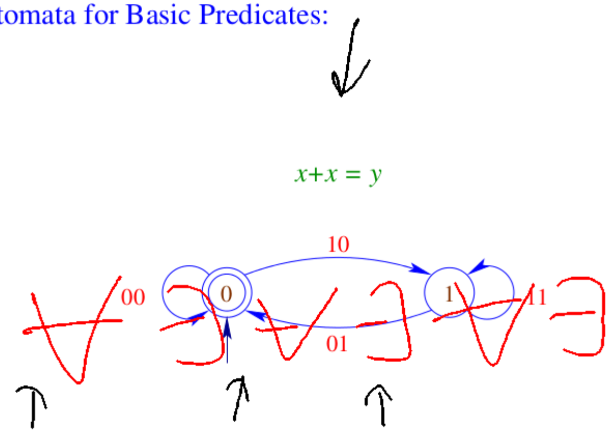
Automata for Basic Predicates:

$x+y=z$



Automata for Basic Predicates:

$x+x=y$



Projecting away the x-component:

213	t	1	0	1	0	1	0	1	1
42	z	0	1	0	1	0	1	0	0
89	y	1	0	0	1	1	0	1	0
17	x	1	0	0	0	1	0	0	0

Results:

Ferrante, Rackoff,1973 : PSAT $\leq$ DSPACE$(2^{2^{c\cdot n}})$

Fischer, Rabin,1974 : PSAT $\geq$ NTIME$(2^{2^{c\cdot n}})$

3.3 Improving the Memory Layout

Goal:

- Better utilization of caches
  $\Longrightarrow$ reduction of the number of cache misses
- Reduction of allocation/de-allocation costs
  $\Longrightarrow$ replacing heap allocation by stack allocation
  $\Longrightarrow$ support to free superfluous heap objects
- Reduction of access costs
  $\Longrightarrow$ short-circuiting indirection chains (Unboxing)

1. Cache Optimization:

Idea: local memory access

- Loading from memory fetches not just one byte but fills a complete cache line.
- Access to neighbored cells become cheaper.
- If all data of an inner loop fits into the cache, the iteration becomes maximally memory-efficient ...

1. Cache Optimization:

Idea: local memory access

- Loading from memory fetches not just one byte but fills a complete cache line.
- Access to neighbored cells become cheaper.
- If all data of an inner loop fits into the cache, the iteration becomes maximally memory-efficient ...

Possible Solutions:

→ Reorganize the data accesses !

→ Reorganize the data !

Such optimizations can be made fully automatic only for arrays :-(

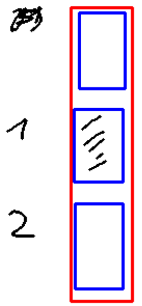
Example:

$$\begin{array}{l} \textsf{for}\ \ (j = 1; j < n; j{+}{+}) \\ \quad \textsf{for}\ \ (i = 1; i < m; i{+}{+}) \\ \qquad a[i][j] = a[i-1][j-1] + a[i][j]; \end{array}$$

Possible Solutions:

→ Reorganize the data accesses !

→ Reorganize the data !



Such optimizations can be made fully automatic only for arrays :-(

Example:

$$\begin{array}{l} \textsf{for}\ \ (j = 1; j < n; j{+}{+}) \\ \quad \textsf{for}\ \ (i = 1; i < m; i{+}{+}) \\ \qquad a[i][j] = a[i-1][j-1] + a[i][j]; \end{array}$$

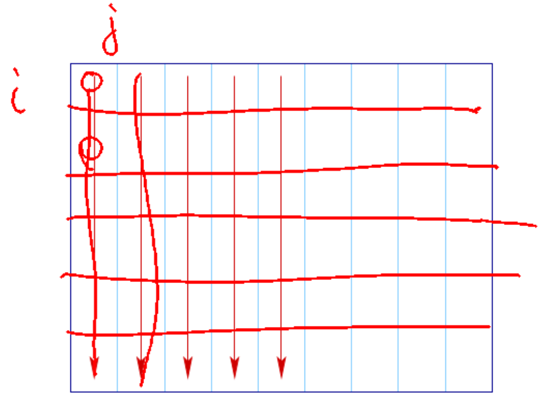
⟹ At first, always iterate over the rows!

⟹ Exchange the ordering of the iterations:

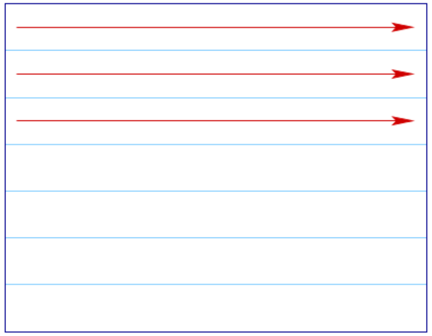
$$\begin{array}{l} \textsf{for}\ \ (i = 1; i < m; i{+}{+}) \\ \quad \textsf{for}\ \ (j = 1; j < n; j{+}{+}) \\ \qquad a[i][j] = a[i-1][j-1] + a[i][j]; \end{array}$$

When is this permitted???

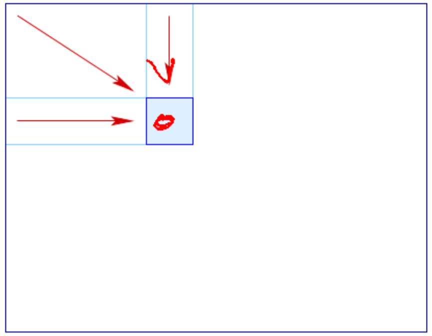
Iteration Scheme: before:



Iteration Scheme: after:



Iteration Scheme: allowed dependencies:



$\Longrightarrow$ At first, always iterate over the rows!

$\Longrightarrow$ Exchange the ordering of the iterations:

$$\begin{array}{l} \textsf{for}\ \ (i = 1; i < m; i{+}{+}) \\ \quad \textsf{for}\ \ (j = 1; j < n; j{+}{+}) \\ \quad\quad a[i][j] = a[i-1][j-1] + a[i][j]; \end{array}$$

When is this permitted???

In our case, we must check that the following equation systems have no solution:

Write		Read
(i_1, j_1)	$=$	$(i_2 - 1, j_2 - 1)$
i_1	$\leq$	i_2
j_2	$\leq$	j_1
(i_1, j_1)	$=$	$(i_2 - 1, j_2 - 1)$
i_2	$\leq$	i_1
j_1	$\leq$	j_2

The first implies: $j_2 \leq j_2 - 1$ Hurra!

The second implies: $i_2 \leq i_2 - 1$ Hurra!

$\Longrightarrow$ At first, always iterate over the rows!

$\Longrightarrow$ Exchange the ordering of the iterations:

$$\begin{array}{l} \textsf{for}\ \ (i = 1; i < m; i{+}{+}) \\ \quad \textsf{for}\ \ (j = 1; j < n; j{+}{+}) \\ \quad\quad a[i][j] = a[i-1][j-1] + a[i][j]; \end{array}$$

When is this permitted???

Example: Matrix-Matrix Multiplication

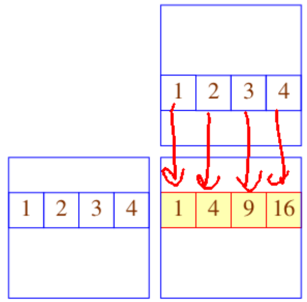
$$\begin{array}{l} \textsf{for}\ \ (i = 0; i < N; i{+}{+}) \\ \quad \textsf{for}\ \ (j = 0; j < M; j{+}{+}) \\ \quad\quad \textsf{for}\ \ (k = 0; k < K; k{+}{+}) \\ \quad\quad\quad c[i][j] = c[i][j] + a[i][k] \cdot b[k][j]; \end{array}$$

Over $b[][]$ the iteration is columnwise :-(

Exchange the two inner loops:

$$\begin{array}{l} \textsf{for}\ \ (i = 0; i < N; i{+}{+}) \\ \quad \textsf{for}\ \ (k = 0; k < K; k{+}{+}) \\ \quad\quad \textsf{for}\ \ (j = 0; j < M; j{+}{+}) \\ \quad\quad\quad c[i][j] = c[i][j] + a[i][k] \cdot b[k][j]; \end{array}$$

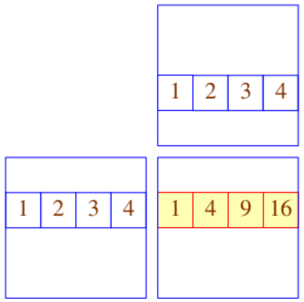
Is this permitted ???



Exchange the two inner loops:

$$
\begin{array}{l}
\text{for } (i = 0; i < N; i{+}{+}) \\
\quad \text{for } (k = 0; k < K; k{+}{+}) \\
\quad\quad \text{for } (j = 0; j < M; j{+}{+}) \\
\quad\quad\quad c[i][j] = c[i][j] + a[i][k] \cdot b[k][j];
\end{array}
$$

Is this permitted ???



Discussion:

- Correctness follows as before :-)
- A similar idea can also be used for the implementation of multiplication for row compressed matrices :-))
- Sometimes, the program must be massaged such that the transformation becomes applicable :-(
- Matrix-matrix multiplication perhaps requires initialization of the result matrix first ...

Discussion:

- Correctness follows as before :-)
- A similar idea can also be used for the implementation of multiplication for row compressed matrices :-))
- Sometimes, the program must be massaged such that the transformation becomes applicable :-(
- Matrix-matrix multiplication perhaps requires initialization of the result matrix first ...

763

```
for (i = 0; i < N; i++)
    for (j = 0; j < M; j++) {
        c[i][j] = 0;
        for (k = 0; k < K; k++)
            c[i][j] = c[i][j] + a[i][k] · b[k][j];
    }
```

- Now, the two iterations can no longer be exchanged :-(
- The iteration over j, however, can be duplicated ...


```
for (i = 0; i < N; i++) {
    for (j = 0; j < M; j++) c[i][j] = 0;
    for (j = 0; j < M; j++)
        for (k = 0; k < K; k++)
            c[i][j] = c[i][j] + a[i][k] · b[k][j];
}
```

Correctness:

⟹ The read entries (here: no) may not be modified in the remaining body of the loop !!!

⟹ The ordering of the write accesses to a memory cell may not be changed :-)

We obtain:

```
for (i = 0; i < N; i++) {
    for (j = 0; j < M; j++) c[i][j] = 0;
    for (k = 0; k < K; k++)
        for (j = 0; j < M; j++)
            c[i][j] = c[i][j] + a[i][k] · b[k][j];
}
```

Discussion:

- Instead of fusing several loops, we now have distributed the loops :-)
- Accordingly, conditionals may be moved out of the loop ⟹ if-distribution ...

Warning:

Instead of using this transformation, the inner loop could also be optimized as follows:

```
for (i = 0; i < N; i++)
    for (j = 0; j < M; j++) {
        t = 0;
        for (k = 0; k < K; k++)
            t = t + a[i][k] · b[k][j];
        c[i][j] = t;
    }
```

We obtain:

$$
\begin{array}{l}
\text{for } (i = 0; i < N; i{+}{+}) \ \{ \\
\quad \text{for } (j = 0; j < M; j{+}{+}) \ c[i][j] = 0; \\
\quad \text{for } (k = 0; k < K; k{+}{+}) \\
\qquad \text{for } (j = 0; j < M; j{+}{+}) \\
\qquad\quad c[i][j] = c[i][j] + a[i][k] \cdot b[k][j]; \\
\}
\end{array}
$$

Discussion:

- Instead of fusing several loops, we now have distributed the loops :-)
- Accordingly, conditionals may be moved out of the loop $\Longrightarrow$ if-distribution ...

Warning:

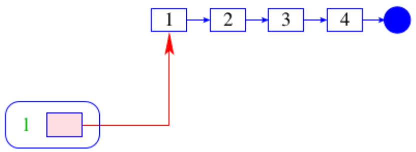
Instead of using this transformation, the inner loop could also be optimized as follows:

$$
\begin{array}{l}
\text{for } (i = 0; i < N; i{+}{+}) \\
\quad \text{for } (j = 0; j < M; j{+}{+}) \ \{ \\
\qquad t = 0; \\
\qquad \text{for } (k = 0; k < K; k{+}{+}) \\
\qquad\quad t = t + a[i][k] \cdot b[k][j]; \\
\qquad c[i][j] = t; \\
\quad \}
\end{array}
$$

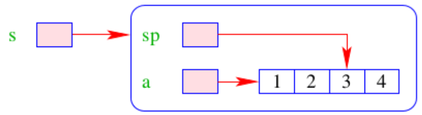
Discussion:

- so far, the optimizations are concerned with iterations over arrays.
- Cache-aware organization of other data-structures is possible, but in general not fully automatic ...

Example: Stacks



Alternative:



Advantage:

\+ The implementation is also simple :-)

\+ The operations push / pop still require constant time :-)

\+ The data are consequtively allocated; stack oscillations are typically small

$\Longrightarrow$ better Cache behavior !!!

2. Stack Allocation instead of Heap Allocation

Problem:

- Programming languages such as Java allocate all data-structures in the heap — even if they are only used within the current method :-(
- If no reference to these data survives the call, we want to allocate these on the stack :-)

$\Longrightarrow$ Escape Analysis

Idea:

Determine points-to information.

Determine if a created object is possibly reachable from the out side ...

Example: Our Pointer Language

$$x = \mathsf{new}();$$
$$y = \mathsf{new}();$$
$$x[A] = y;$$
$$z = y;$$
$$\mathsf{ret} = z;$$

... could be a possible method body ;-)

Accessible from the outside world are memory blocks which:

- are assigned to a global variable such as ret; or
- are reachable from global variables.

... in the Example:

$$x = \mathsf{new}();$$
$$y = \mathsf{new}();$$
$$x[A] = y;$$
$$z = y;$$
$$\mathsf{ret} = \boxed{z};$$

Accessible from the outside world are memory blocks which:

- are assigned to a global variable such as ret; or
- are reachable from global variables.

... in the Example:

$$x = \mathsf{new}();$$
$$y = \mathsf{new}();$$
$$x[A] = y;$$
$$z = \boxed{y};$$
$$\mathsf{ret} = \boxed{z};$$

Extension: Procedures

- We require an interprocedural points-to analysis :-)
- We know the whole program, we can, e.g., merge the control-flow graphs of all procedures into one and compute the points-to information for this.
- Warning: If we always use the same global variables $y_1, y_2, \ldots$ for (the simulation of) parameter passing, the computed information is necessarily imprecise :-(
- If the whole program is not known, we must assume that each reference which is known to a procedure escapes :-((

Extension: Procedures

- We require an interprocedural points-to analysis :-)
- We know the whole program, we can, e.g., merge the control-flow graphs of all procedures into one and compute the points-to information for this.
- Warning: If we always use the same global variables $y_1, y_2, \ldots$ for (the simulation of) parameter passing, the computed information is necessarily imprecise :-(
- If the whole program is not known, we must assume that each reference which is known to a procedure escapes :-((

781