Presburger Arithmetic $\equiv$ full arithmetic without multiplication

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

$\Longrightarrow$ Hilbert’s 10th Problem
$\Longrightarrow$ Gödel’s Theorem
Presburger Formulas over $\mathbb{N}$:

$\phi ::= x + y = z \mid x = n \mid \phi_1 \land \phi_2 \mid \neg \phi \mid \exists x : \phi$

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

Goal: PSAT

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

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

The set of satisfying variable assignments is regular \( \Rightarrow \)

\[ \phi_1 \land \phi_2 \implies \mathcal{L}(\phi_1) \cap \mathcal{L}(\phi_2) \] (Intersection)

\[ \neg \phi \implies \overline{\mathcal{L}(\phi)} \] (Complement)

\[ \exists x : \mathcal{L} \implies \pi_x(\mathcal{L}(\phi)) \] (Projection)

Warning:

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

\( \implies \) The automaton for projection must be enriched such that the property is re-established \(!!\)

Automata for Basic Predicates:

\[ x = 5 \]

![Automaton for x = 5](image)

Automata for Basic Predicates:

\[ x + x = y \]

![Automaton for x + x = y](image)
Automata for Basic Predicates:

$$x + y = z$$

Automata for Basic Predicates:

$$x + x = y$$

Projecting away the $x$-component:

<table>
<thead>
<tr>
<th></th>
<th>t</th>
<th>1 0 1 0 1 0 1 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>213</td>
<td>z</td>
<td>0 1 0 1 0 1 0 0</td>
</tr>
<tr>
<td>42</td>
<td>y</td>
<td>1 0 0 1 1 0 1 0</td>
</tr>
<tr>
<td>89</td>
<td>x</td>
<td>1 0 0 0 1 0 0 0</td>
</tr>
</tbody>
</table>
1. Cache Optimization:

**Idea:**
- Local memory access

- If all data of an inner loop fits into the cache, the iteration becomes maximally memory-efficient...

**Local Memory Optimization:**

- Leading from memory fetches not just one byte but fills a complete cache line.

2. Improving the Memory Layout

**Goal:**
- Better utilization of caches

- Reduction of the number of cache misses

- Replacing heap allocation by stack allocation

- Support to free superfluous heap objects

- Short-circuiting induction chains (unboxed)
Possible Solutions:

→ Reorganize the data accesses!
→ Reorganize the data!

Such optimizations can be made fully automatic only for arrays.

Example:

\[
\begin{align*}
\text{for} \ (j = 1; j < n; j++) & \\
\quad \text{for} \ (i = 1; i < m; i++) & \\
\quad \quad \quad a[i][j] = a[i - 1][j - 1] + a[i][j];
\end{align*}
\]

⇒ At first, always iterate over the rows!
⇒ Exchange the ordering of the iterations:

\[
\begin{align*}
\text{for} \ (i = 1; i < m; i++) & \\
\quad \text{for} \ (j = 1; j < n; j++) & \\
\quad \quad \quad a[i][j] = a[i - 1][j - 1] + a[i][j];
\end{align*}
\]

When is this permitted???
Iteration Scheme: after:

\[ a[i][j] = a[i-1][j-1] + a[i][j] \]

\[ \text{for } (i = 1; i < n; i++) \]
\[ \text{for } (j = 1; j < n; j++) \]

When is this permitted???

\[ \Rightarrow \] At first, always iterate over the rows!
\[ \Rightarrow \] Exchange the ordering of the iterations:

\[ (i_1, j_1) = (i_2 - 1, j_2 - 1) \]
\[ i_1 \leq i_2 \]
\[ j_2 \leq j_1 \]

\[ (i_1, j_1) = (i_2 - 1, j_2 - 1) \]
\[ i_2 \leq i_1 \]
\[ j_1 \leq j_2 \]

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

<table>
<thead>
<tr>
<th>Write</th>
<th>Read</th>
</tr>
</thead>
<tbody>
<tr>
<td>( (i_1, j_1) ) = ( (i_2 - 1, j_2 - 1) )</td>
<td>( i_1 \leq i_2 )</td>
</tr>
<tr>
<td>( j_2 \leq j_1 )</td>
<td>( (i_1, j_1) ) = ( (i_2 - 1, j_2 - 1) )</td>
</tr>
<tr>
<td>( i_2 \leq i_1 )</td>
<td>( j_1 \leq j_2 )</td>
</tr>
</tbody>
</table>

The first implies: \( j_2 \leq j_2 - 1 \) Hurra!
The second implies: \( i_2 \leq i_2 - 1 \) Hurra!
At first, always iterate over the rows!

Exchange the ordering of the iterations:

\[
\text{for } (i = 1; i < n; i++)
\text{ for } (j = 1; j < n; j++)
\quad a[i][j] = a[i-1][j-1] + a[i][j];
\]

**When is this permitted??**

---

**Example:** Matrix-Matrix Multiplication

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

Over \(b[[]]\) the iteration is columnwise :-(

---

Exchange the two inner loops:

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

**Is this permitted??**
Exchange the two inner loops:

\[
\begin{align*}
\text{for } & (i = 0; i < N; i++) \\
\text{for } & (k = 0; k < K; k++) \\
\text{for } & (j = 0; j < M; j++) \\
& c[i][j] = c[i][j] + a[i][k] \cdot b[k][j];
\end{align*}
\]

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 ...
for \((i = 0; i < N; i++)\)
for \((j = 0; j < M; j++)\) { 
\(c[i][j] = 0;\) 
\(c[i][j] = c[i][j] + a[i][k] \cdot b[k][j];\)
} 

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

---

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 \((j = 0; j < M; j++)\) 
\(c[i][j] = c[i][j] + a[i][k] \cdot b[k][j];\) 
} 

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

---

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

Correctness:

\(\Rightarrow\) The read entries (here: no) may not be modified in the remaining body of the loop !!!
\(\Rightarrow\) The ordering of the write accesses to a memory cell may not be changed :-)

---

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 \((k = 0; k < K; k++)\) 
\(t = t + a[i][k] \cdot b[k][j];\) 
\(c[i][j] = t;\) 
}
We obtain:

```plaintext
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:

```plaintext
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;
```

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 consecutively allocated; stack oscillations are typically small ➞ 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 :)

  \[ \implies \text{Escape Analysis} \]

   Idea:
   Determine points-to information.
   Determine if a created object is possibly reachable from the outside ...

   Example: Our Pointer Language

   \[
   \begin{align*}
   x &= \text{new}(); \\
   y &= \text{new}(); \\
   x[A] &= y; \\
   z &= y; \\
   \ret &= z;
   \end{align*}
   \]

   ... 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:

   \[
   \begin{align*}
   x &= \text{new}(); \\
   y &= \text{new}(); \\
   x[A] &= y; \\
   z &= y; \\
   \ret &= z;
   \end{align*}
   \]

   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:

   \[
   \begin{align*}
   x &= \text{new}(); \\
   y &= \text{new}(); \\
   x[A] &= y; \\
   z &= y; \\
   \ret &= z;
   \end{align*}
   \]
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.