Compilation avancée et optimisation de programmes

Alain Darte

CNRS, Compsys
Laboratoire de l'Informatique du Parallélisme
École normale supérieure de Lyon

Multi-dimensional polyhedral optimizations

Outline

Outline

Affine bounds and affine array access functions

Fortran DO loops:

```
DO i=1, N
  DO j=1, N
    a(i,j) = c(i,j-1)
    c(i,j) = a(i,j) + a(i-1,N)
  ENDDO
ENDDO
```

- Nested loops, static control.
- Iteration domain and vector.
- Loop increment $= 1$.
- Affine bounds of surrounding counters & parameters.
- Multi-dimensional arrays, same restriction for access functions.

Affine bounds and affine array access functions

Fortran DO loops:

```
DO i=1, N
  DO j=1, N
    a(i,j) = c(i,j-1)
    c(i,j) = a(i,j) + a(i-1,N)
  ENDDO
ENDDO
```

- Nested loops, static control.
- Iteration domain and vector.
- Loop increment = 1.
- Affine bounds of surrounding counters & parameters.
- Multi-dimensional arrays, same restriction for access functions.

Polyhedral model: the "all-affine" world, with exact analysis

- Iteration domain = polytope.
- Sequential order $\leq_{seq}$.
- Data = images of polytopes by affine functions.

Affine bounds and affine array access functions

Fortran DO loops:

```
DO i=1, N
  DO j=1, N
    a(i,j) = c(i,j-1)
    c(i,j) = a(i,j) + a(i-1,N)
  ENDDO
ENDDO
```

- Nested loops, static control.
- Iteration domain and vector.
- Loop increment = 1.
- Affine bounds of surrounding counters & parameters.
- Multi-dimensional arrays, same restriction for access functions.

Polyhedral model: the "all-affine" world, with exact analysis

- Iteration domain = polytope.
- Sequential order $\leq_{seq}$.
- Data = images of polytopes by affine functions.

Typical criticism: such codes do not exist.

(Parametric) analysis, transformations, optimizations

Data-flow array analysis

- Array expansion.
- Single assignment.
- Liveness array analysis.
- Data reuse.

Mapping computations & data

- Systolic arrays design.
- Data distribution.
- Communication opt.

Loop transformations

- Automatic parallelization.
- Transformations framework.
- Code generation (with loops or with automaton).

Counting & Ehrhart polynomials

- Cache misses.
- Memory size computations.
- Latency computations.

And many more. . .

Many languages fit in the polyhedral model

C for loops:

```
for (i=1, i<=N, i++) {
  for (j=1, j<=N, j++) {
    a[i][j] = c[i][j-1];
    c[i][j] = a[i][j] + a[i-1][N];
  }
}
```

Uniform recurrence equations

$\forall(i,j)$ such that $1 \leq i,j \leq N$

$$\begin{cases} a(i,j) = c(i,j-1) \\ b(i,j) = a(i-1,j) + b(i,j+1) \\ c(i,j) = a(i,j) + b(i,j) \end{cases}$$

C while loops:

```
y = 0; x = 0;
while (x <= N && y <= N) {
  if (?) {
    x=x+1;
    while (y >= 0 && ?) y=y-1;
  }
  y=y+1;
}
```

FAUST: audio processing

```
random = +(12345) ~ *(1103515);
noise = random/2147483.0;
process = random/2 : @(10);
```

and more: Matlab, Fortran90, StreamIt, HPF, C for HLS, . . .

Many tools and a recent revival

PIP Parametric integer programming.

POLYLIB Polyhedra manipulations.

FADALIB Fuzzy array data-flow analysis.

CLOOG Code generation, from polytopes to loops.

EHRHART & BARVINOK Counting tools.

CL@K Critical and admissible lattices.

PIPS Automatic parallelizer & code transformation framework.

PLUTO Automatic parallelizer & locality optimizer for multicores.

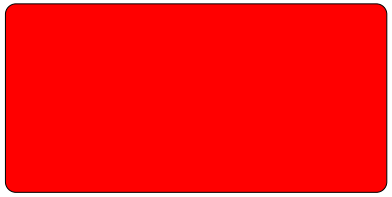
GRAPHITE High-level memory optimizations framework in GCC.

R-STREAM High-level compiler of Reservoir Labs.

. . .

But still, how to deal with non-static control programs?

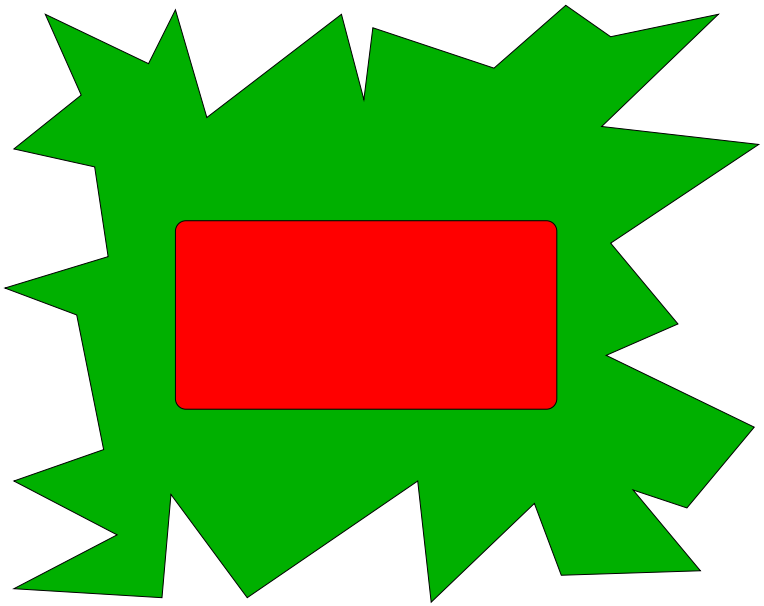
Polyhedral model.



But still, how to deal with non-static control programs?

Polyhedral model.

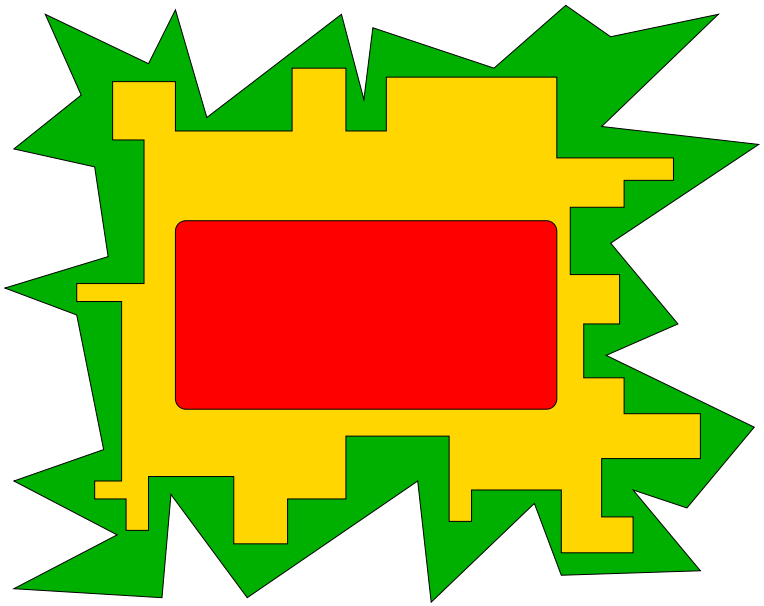
Real life.



But still, how to deal with non-static control programs?

Polyhedral model.

Real life.



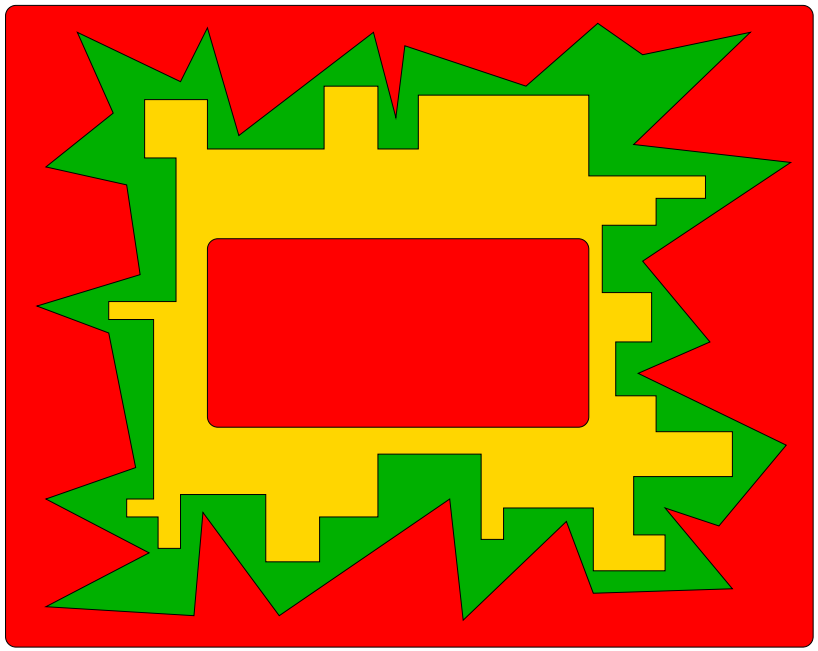
Extensions.

- Non-affine constraints.
- Handling of while loops.
- Recursive programs.
- Beyond induction variables.

But still, how to deal with non-static control programs?

Polyhedral model.

Real life.



Extensions.

- Non-affine constraints.
- Handling of while loops.
- Recursive programs.
- Beyond induction variables.

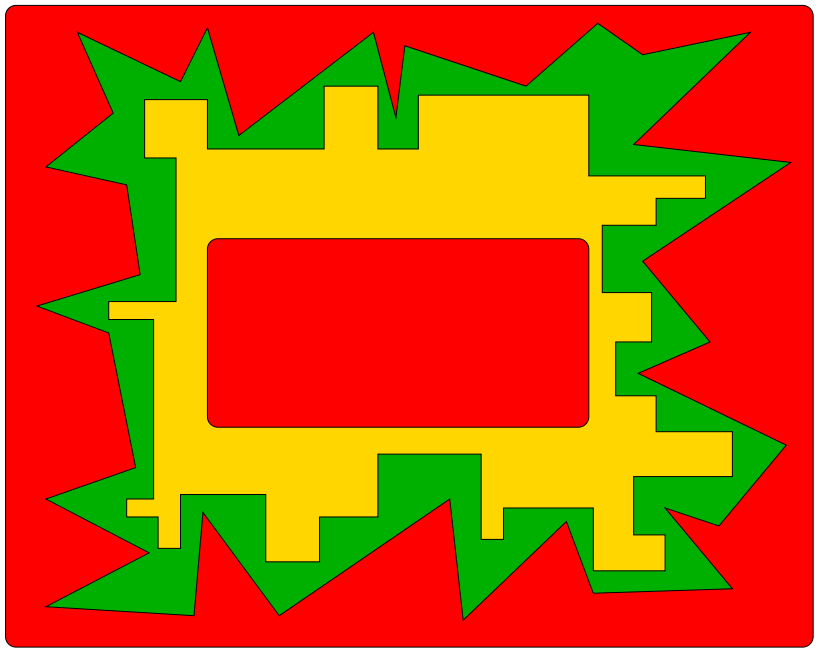
Approximations.

- Dependences, lifetime, data & iteration domains, etc.
- Do not assume exact information is available.

But still, how to deal with non-static control programs?

Polyhedral model.

Real life.



Extensions.

- Non-affine constraints.
- Handling of while loops.
- Recursive programs.
- Beyond induction variables.

Approximations.

- Dependences, lifetime, data & iteration domains, etc.
- Do not assume exact information is available.

Think conservative!

Apparent dependence graph and parallelism detection

Is there some loop parallelism (i.e., parallel loop iterations) in the following two codes? What is their degree of parallelism?

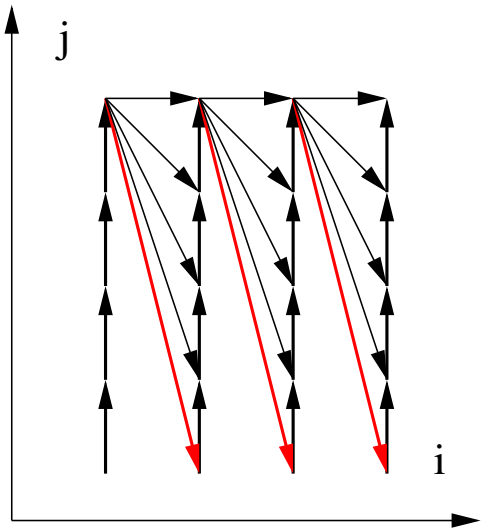
```
DO i=1, N
  DO j=1, N
    a(i,j) = c(i,j-1)
    c(i,j) = a(i,j) + a(i-1,N)
  ENDDO
ENDDO
```

```
DO i=1, N
  DO j=1, N
    a(i,j) = c(i,j-1)
    c(i,j) = a(i,j) + a(i-1,j)
  ENDDO
ENDDO
```

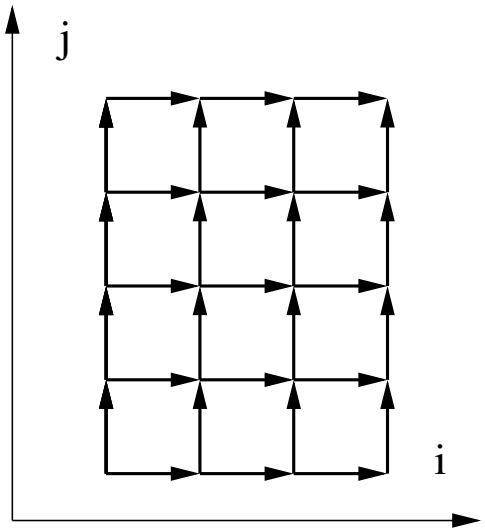
Apparent dependence graph and parallelism detection

Is there some loop parallelism (i.e., parallel loop iterations) in the following two codes? What is their degree of parallelism?

```
DO i=1, N
  DO j=1, N
    a(i,j) = c(i,j-1)
    c(i,j) = a(i,j) + a(i-1,N)
  ENDDO
ENDDO
```




```
DO i=1, N
  DO j=1, N
    a(i,j) = c(i,j-1)
    c(i,j) = a(i,j) + a(i-1,j)
  ENDDO
ENDDO
```

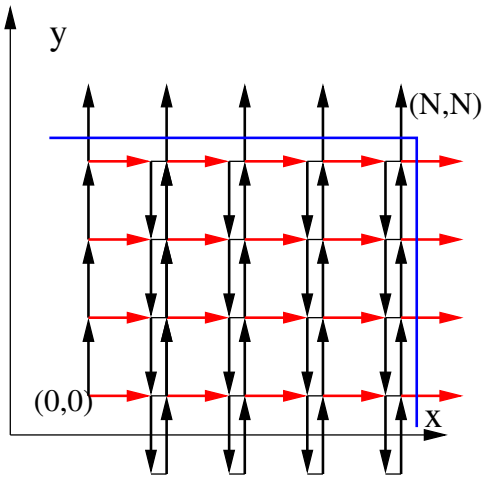


Apparent evolution of variables and program termination

Does this program terminate?

If yes, how many steps in the worst case? Useful for WCET.

```
y = 0; x = 0;
while (x <= N && y <= N) {
  if (?) {
    x=x+1;
    while (y >= 0 && ?) y=y-1;
  }
  y=y+1;
}
```

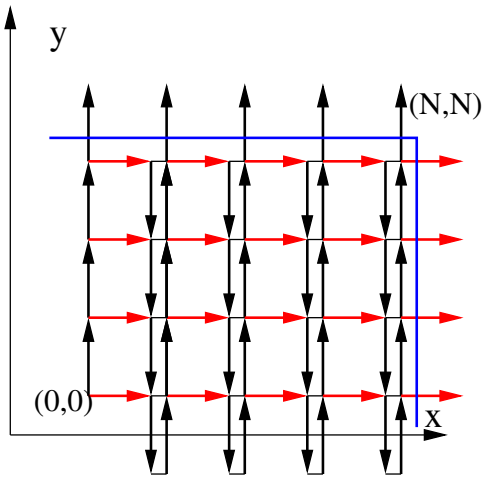


Apparent evolution of variables and program termination

Does this program terminate?

If yes, how many steps in the worst case? Useful for WCET.

```
y = 0; x = 0;
while (x <= N && y <= N) {
  if (?) {
    x=x+1;
    while (y >= 0 && ?) y=y-1;
  }
  y=y+1;
}
```



➽ Terminates in at most $N^2 + 3N + 2 = O(N^2)$ steps.

Note: a single while loop can generate quadratic (or more) WCCC.

Surprisingly, similar to parallel detection in Fortran DO loops.

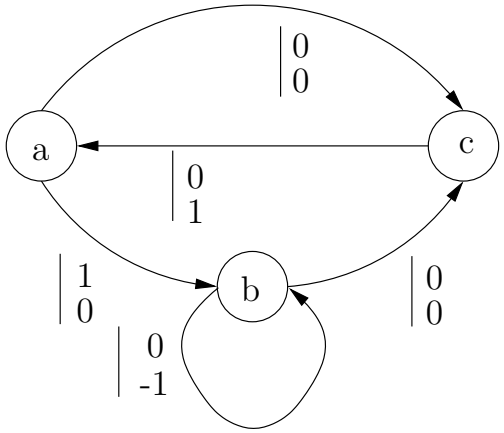
Outline

SURE: system of uniform recurrence equations (1967)

Karp, Miller, Winograd: "The organization of computations for uniform recurrence equations" (J. ACM, 14(3), pp. 563-590).

$$\forall p \in \mathcal{P} = \{p = (i,j) \mid 1 \leq i,j \leq N\}$$

$$\begin{cases} a(i,j) = c(i,j-1) \\ b(i,j) = a(i-1,j) + b(i,j+1) \\ c(i,j) = a(i,j) + b(i,j) \end{cases}$$



Semantics:

- RDG (reduced dependence graph) $G = (V, E, w)$.
- Explicit dependences & iteration domain $\mathcal{P}$, implicit schedule.
- $e = (u, v) \Leftrightarrow v(p)$ depends on $u(p - w(e))$, i.e., must be computed after. If $p - w(e) \notin \mathcal{P}$, it is an input.
- EDG (expanded dep. graph): vertices $V \times \mathcal{P}$ = unrolled RDG.

Two main problems: computability & scheduling

Computability

Can we compute $a(p)$ in a finite number of steps?

- $a(p)$ is computable iff no infinite path in the EDG to (a, p).

Two main problems: computability & scheduling

Computability

Can we compute $a(p)$ in a finite number of steps?

- $a(p)$ is computable iff no infinite path in the EDG to (a, p).
- If $\mathcal{P} = (\mathbb{N})^n$, computable for all $p \in \mathcal{P}$ if and only the RDG has no cycle C with $w(C) \leq 0$ (component-wise).

Two main problems: computability & scheduling

Computability

Can we compute $a(p)$ in a finite number of steps?

- $a(p)$ is computable iff no infinite path in the EDG to (a, p).
- If $\mathcal{P} = (\mathbb{N})^n$, computable for all $p \in \mathcal{P}$ if and only the RDG has no cycle C with $w(C) \leq 0$ (component-wise).
- A SURE is computable for all bounded domains P if and only if the RDG has no cycle C with $w(C) = 0$.

Two main problems: computability & scheduling

Computability

Can we compute $a(p)$ in a finite number of steps?

- $a(p)$ is computable iff no infinite path in the EDG to (a, p).
- If $\mathcal{P} = (\mathbb{N})^n$, computable for all $p \in \mathcal{P}$ if and only the RDG has no cycle C with $w(C) \leq 0$ (component-wise).
- A SURE is computable for all bounded domains P if and only if the RDG has no cycle C with $w(C) = 0$.

Scheduling

For a computable SURE:

- How to compute or evaluate the minimal number of steps to compute $a(p)$ (free schedule = ASAP schedule)?
- How to evaluate the potential for parallelism?
- How to find an explicit schedule? With guaranteed latency?

Related models

Nested DO loops in imperative languages

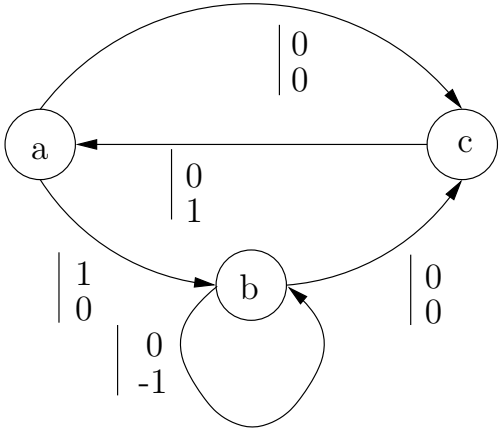
- Explicit iteration domain given by loop bounds.
- Explicit (sequential) order = lexicographic on counters + text.
- Implicit dependences: needs powerful program analysis.
- No computability problem but more general dependences.

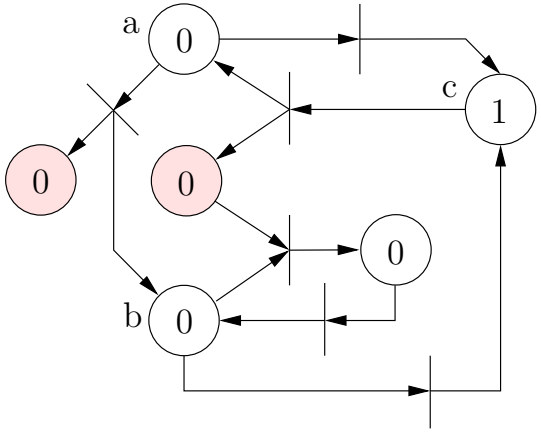
Related models

Nested DO loops in imperative languages

- Explicit iteration domain given by loop bounds.
- Explicit (sequential) order = lexicographic on counters + text.
- Implicit dependences: needs powerful program analysis.
- No computability problem but more general dependences.

Petri nets: transitions, places, accessibility problems.



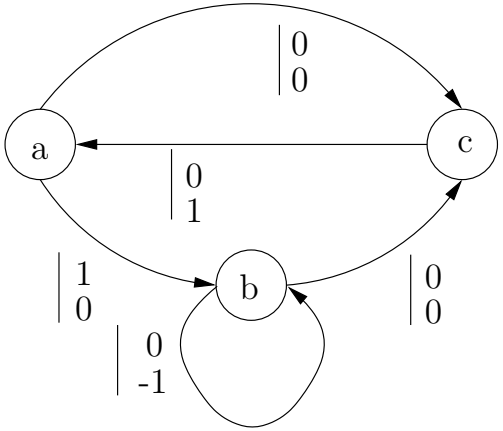


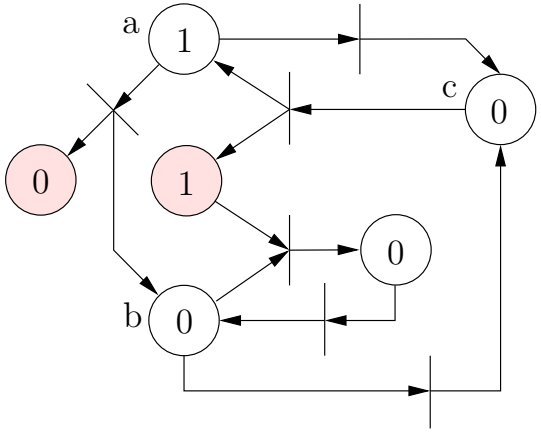
Related models

Nested DO loops in imperative languages

- Explicit iteration domain given by loop bounds.
- Explicit (sequential) order = lexicographic on counters + text.
- Implicit dependences: needs powerful program analysis.
- No computability problem but more general dependences.

Petri nets: transitions, places, accessibility problems.



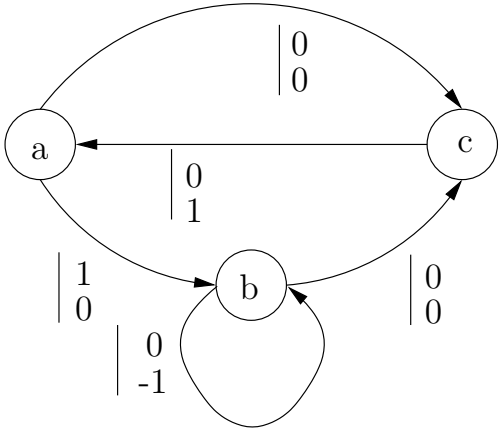


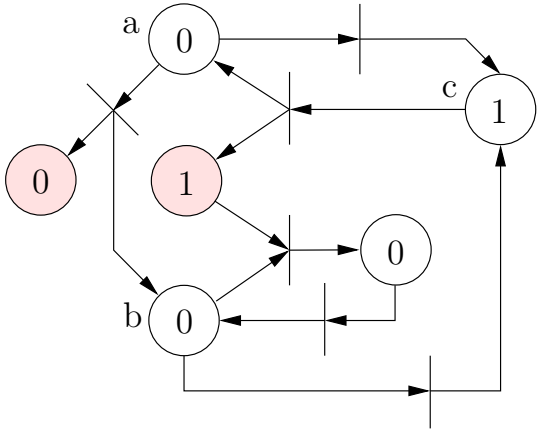
Related models

Nested DO loops in imperative languages

- Explicit iteration domain given by loop bounds.
- Explicit (sequential) order = lexicographic on counters + text.
- Implicit dependences: needs powerful program analysis.
- No computability problem but more general dependences.

Petri nets: transitions, places, accessibility problems.



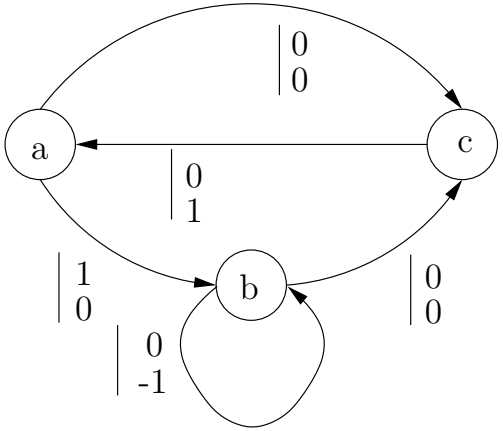


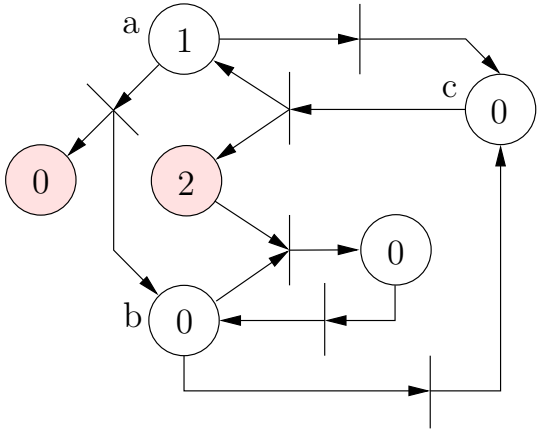
Related models

Nested DO loops in imperative languages

- Explicit iteration domain given by loop bounds.
- Explicit (sequential) order = lexicographic on counters + text.
- Implicit dependences: needs powerful program analysis.
- No computability problem but more general dependences.

Petri nets: transitions, places, accessibility problems.



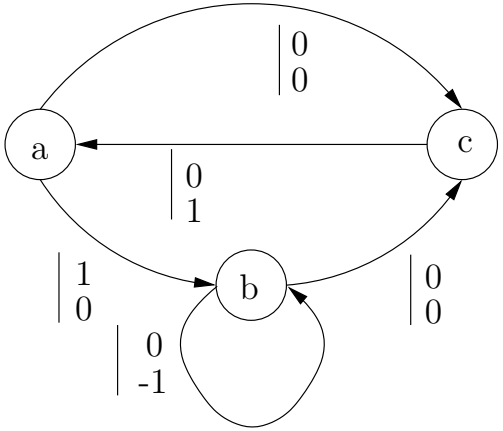


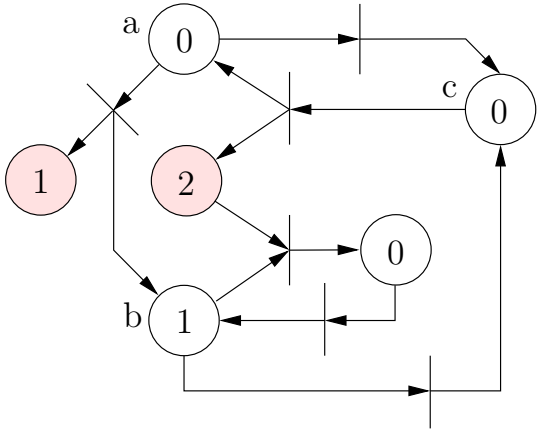
Related models

Nested DO loops in imperative languages

- Explicit iteration domain given by loop bounds.
- Explicit (sequential) order = lexicographic on counters + text.
- Implicit dependences: needs powerful program analysis.
- No computability problem but more general dependences.

Petri nets: transitions, places, accessibility problems.



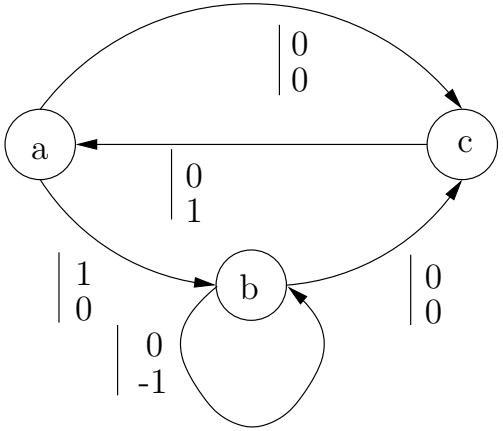


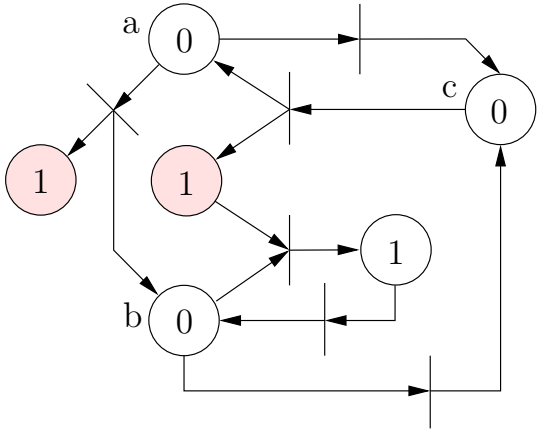
Related models

Nested DO loops in imperative languages

- Explicit iteration domain given by loop bounds.
- Explicit (sequential) order = lexicographic on counters + text.
- Implicit dependences: needs powerful program analysis.
- No computability problem but more general dependences.

Petri nets: transitions, places, accessibility problems.



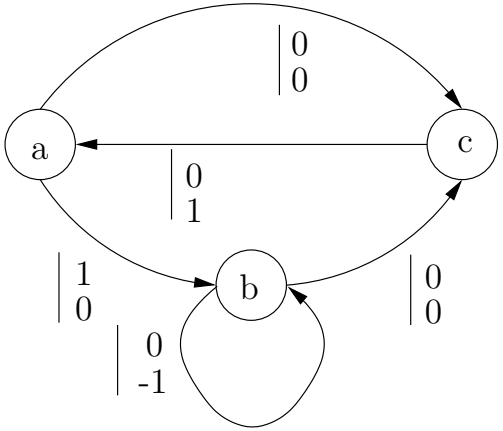


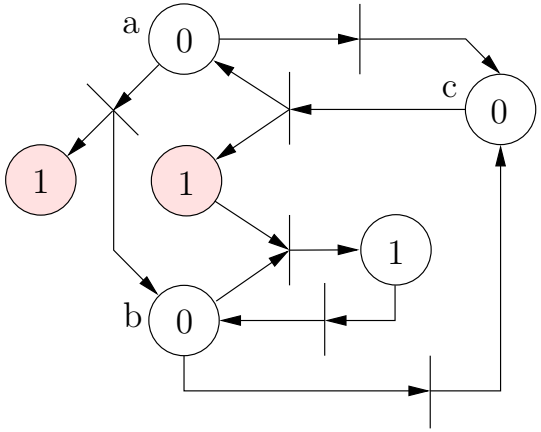
Related models

Nested DO loops in imperative languages

- Explicit iteration domain given by loop bounds.
- Explicit (sequential) order = lexicographic on counters + text.
- Implicit dependences: needs powerful program analysis.
- No computability problem but more general dependences.

Petri nets: transitions, places, accessibility problems.





Circuits: retiming, cyclic scheduling, clock and registers.

General programs: flowchart automaton, evolution of variables.

Looking for zero-weight cycles

Computability: Can we compute $a(p)$ in a finite number of steps?
Scheduling: If yes, how to find an explicit and "good" schedule?

Lemma 1

A SURE is computable for all bounded domains $\mathcal{P}$ if and only if the RDG has no cycle C with $w(C) = 0$.

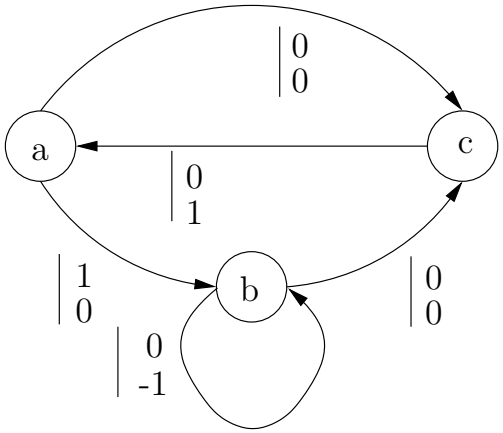
Looking for zero-weight cycles

Computability: Can we compute $a(p)$ in a finite number of steps?
Scheduling: If yes, how to find an explicit and "good" schedule?

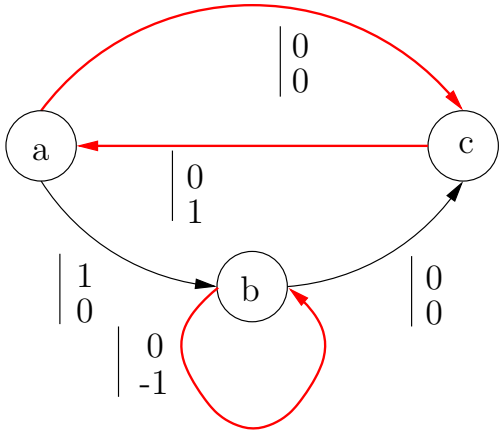
Lemma 1

A SURE is computable for all bounded domains $\mathcal{P}$ if and only if the RDG has no cycle C with $w(C) = 0$.

Key structure: the subgraph G' induced by all edges that belong to a multi-cycle (i.e., union of cycles) of zero weight.



Graph G



Graphs G and G'

Key properties

Three elementary key lemmas.

Lemma 2

A zero-weight cycle is a zero-weight multi-cycle.

➽ *Look in G' only.*

Key properties

Three elementary key lemmas.

Lemma 2

A zero-weight cycle is a zero-weight multi-cycle.

➽ *Look in G' only.*

Lemma 3

A zero-weight cycle belongs to a strongly connected component.

➽ *Look in each strongly connected component (SCC) separately.*

Key properties

Three elementary key lemmas.

Lemma 2

A zero-weight cycle is a zero-weight multi-cycle.

➽ *Look in G' only.*

Lemma 3

A zero-weight cycle belongs to a strongly connected component.

➽ *Look in each strongly connected component (SCC) separately.*

Lemma 4

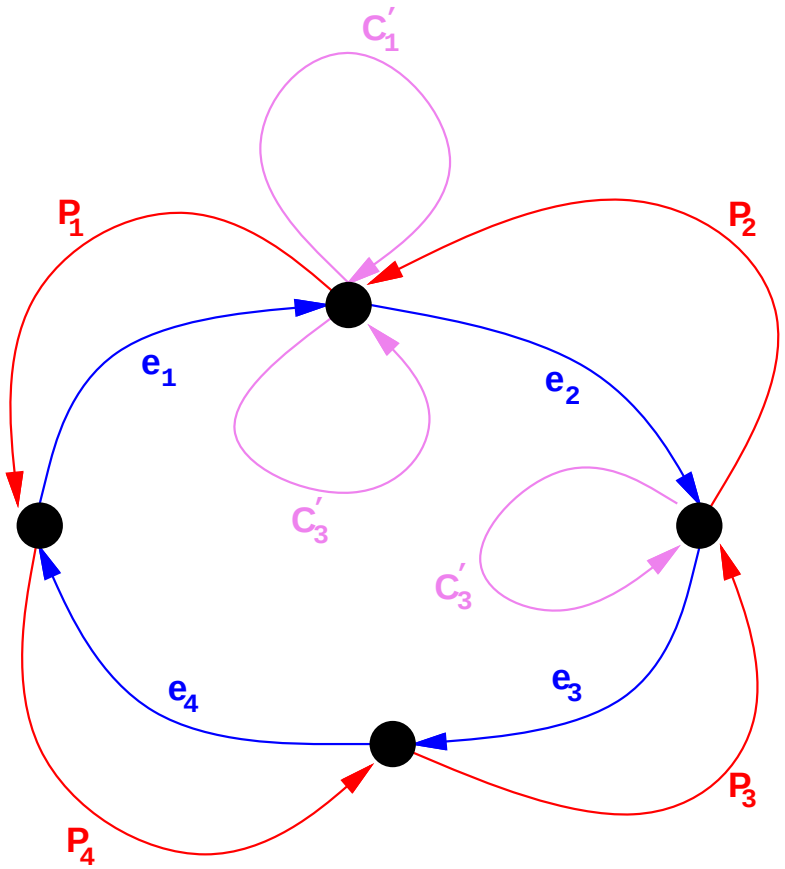
If G' is strongly connected, there is a zero-weight cycle.

➽ *Terminating case.*

Key properties

Lemma 4

If G' is strongly connected, there is a zero-weight cycle.



- $\sum_i e_i$ cycle that visits all vertices.
- e_i in multi-cycle C_i, with $w(C_i) = 0$.
- $C_i = e_i + P_i + C'_i$.
- Follow the e_i, then the P_i and, on the way, plug the C'_i.

Karp, Miller, and Winograd's decomposition

Boolean KMW(G):

- Build G' the subgraph of zero-weight multicycles of G.
- Compute $G'_1, \ldots, G'_s$, the s SCCs of G'.
  - If $s = 0$, G' is empty, return TRUE.
  - If $s = 1$, G' is strongly connected, return FALSE.
  - Otherwise return $\wedge_i$KMW(G'_i) (logical AND).

Then, G is computable iff KMW(G) returns TRUE.

Karp, Miller, and Winograd's decomposition

Boolean KMW(G):

- Build G' the subgraph of zero-weight multicycles of G.
- Compute $G'_1, \ldots, G'_s$, the s SCCs of G'.
  - If $s = 0$, G' is empty, return TRUE.
  - If $s = 1$, G' is strongly connected, return FALSE.
  - Otherwise return $\wedge_i$KMW(G'_i) (logical AND).

Then, G is computable iff KMW(G) returns TRUE.

Depth d of the decomposition

$d = 0$ if G is acyclic, $d = 1$ if all SCCs have an empty G', etc.

Theorem 1 (Depth of the decomposition)

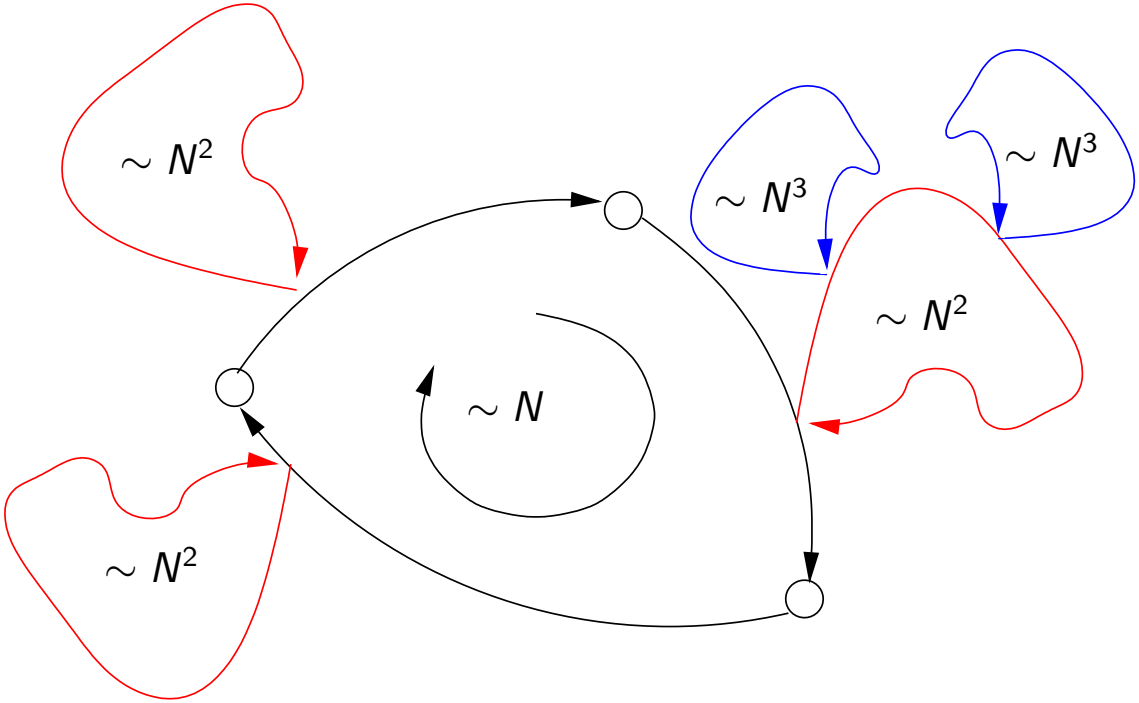
If G is computable, $d \leq n$, otherwise, $d \leq n + 1$.

(n is the dimension of the problem, i.e., the dimension of $\mathcal{P}$.)

Length of longest dependence path in the EDG

Theorem 2 (Longest dependence path)

If $\mathcal{P}$ contains a n-dimensional cube of size $\Omega(N)$, there exists a dependence path of length $\Omega(N^d)$.



Subtlety: needs to make sure that the path stays in the EDG.

But how to compute G'? Primal and dual programs.

$e \in G'$ iff $v_e = 0$ in any optimal solution of the linear program:

$$\min \left\{ \sum_e v_e \;\middle|\; q \geq 0,\ v \geq 0,\ q + v \geq 1,\ Cq = 0,\ Wq = 0 \right\}$$

☛ A single (rational) linear program.

But how to compute G'? Primal and dual programs.

$e \in G'$ iff $v_e = 0$ in any optimal solution of the linear program:

$$\min \left\{ \sum_e v_e \;\mid\; q \geq 0,\; v \geq 0,\; q + v \geq 1,\; Cq = 0,\; Wq = 0 \right\}$$

☛ A single (rational) linear program.

Always interesting to take a look at the dual program:

$$\max \left\{ \sum_e z_e \;\mid\; 0 \leq z \leq 1,\; X.w(e) + \rho_v - \rho_u \geq z_e,\; \forall e = (u, v) \in E \right\}$$

Additional property, for any optimal solution:

- $e \in G' \Leftrightarrow X.w(e) + \rho_v - \rho_u = 0$.
- $e \notin G' \Leftrightarrow X.w(e) + \rho_v - \rho_u \geq 1$.

But how to compute G'? Primal and dual programs.

$e \in G'$ iff $v_e = 0$ in any optimal solution of the linear program:

$$\min \left\{ \sum_e v_e \;\middle|\; q \geq 0,\ v \geq 0,\ q + v \geq 1,\ Cq = 0,\ Wq = 0 \right\}$$

☛ A single (rational) linear program.

Always interesting to take a look at the dual program:

$$\max \left\{ \sum_e z_e \;\middle|\; 0 \leq z \leq 1,\ X.w(e) + \rho_v - \rho_u \geq z_e,\ \forall e = (u, v) \in E \right\}$$

Additional property, for any optimal solution:

- $e \in G' \Leftrightarrow X.w(e) + \rho_v - \rho_u = 0$.
- $e \notin G' \Leftrightarrow X.w(e) + \rho_v - \rho_u \geq 1$.

Schedule $\sigma : V \times \mathcal{P} \to \mathbb{N}$, with $\sigma(u, p) = X.p + \rho_u$, is valid if:

$$\begin{aligned} &\sigma(v, p) \geq \sigma(u, p - w(e)) + 1 \\ &\Leftrightarrow X.p + \rho_v \geq X.(p - w(e)) + \rho_u + 1 \\ &\Leftrightarrow X.w(e) + \rho_v - \rho_u \geq 1 \end{aligned}$$

Scheduling: dual of computability.

- $e \in G' \Leftrightarrow X.w(e) + \rho_v - \rho_u = 0$.
- $e \notin G' \Leftrightarrow X.w(e) + \rho_v - \rho_u \geq 1$.

Multi-dimensional scheduling: hours, minutes, seconds, etc.

$e \notin G'$: u & v computed at different hours.
Different iterations of the outer loop = loop-carried.

$e \in G'$: u & v same hour, constraints pushed to inner dimensions.
Same iteration of outer loop = loop-independent.

Special form of schedule: affine, same linear part in a SCC of G'.

Scheduling: dual of computability.

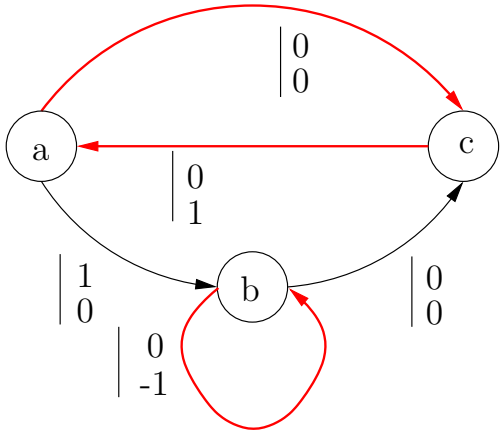
- $e \in G' \Leftrightarrow X.w(e) + \rho_v - \rho_u = 0.$
- $e \notin G' \Leftrightarrow X.w(e) + \rho_v - \rho_u \geq 1.$

Multi-dimensional scheduling: hours, minutes, seconds, etc.

$e \notin G'$: u & v computed at different hours.
Different iterations of the outer loop = loop-carried.

$e \in G'$: u & v same hour, constraints pushed to inner dimensions.
Same iteration of outer loop = loop-independent.

Special form of schedule: affine, same linear part in a SCC of G'.



$$\left.\begin{array}{l} X_1.(0,1) = 0 \\ X_1.(1,1) \geq 2 \end{array}\right\} \Rightarrow \left\{\begin{array}{l} X_1 = (2,0),\ \rho_a = 1 \\ \rho_b = 0,\ \rho_c = 1 \end{array}\right.$$

$$\text{Final schedule} \left\{\begin{array}{l} \sigma_a(i,j) = (2i+1, 2j) \\ \sigma_b(i,j) = (2i, -j) \\ \sigma_c(i,j) = (2i+1, 2j+1) \end{array}\right.$$

Performances of schedules for computable equations

Theorem 3 (Optimality of multi-dimensional schedules)

If P contains a n-dim. cube of size $\theta(N)$, there is a dependence path of length $\Omega(N^d)$ and a schedule of latency $O(N^d)$.

Theorem 4 (Case of one-dimensional schedules)

If $d = 1$, the best affine schedule is $\sim \lambda N$, for some $\lambda > 0$, and so is the maximal dependence length.

Theorem 5 (Case of a single equation)

For one equation, $d = 0$ or $d = 1$. Moreover, if $d = 1$, the best linear schedule is optimal up to a constant.

Theorem 6 (Link with tiling)

The maximal number of permutable loops is linked to the dimension of the vector space $Vect(\{w(C) \mid C \text{ cycle of } G'\})$.