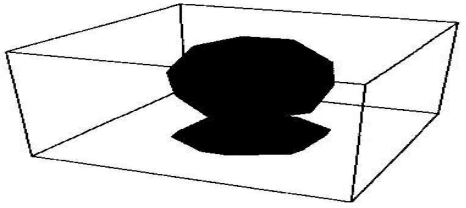
Adarsh Natarajan
Sunkari Sasidhar



A PRACTICAL ALGORITHM FOR EXACT ARRAY DEPENDENCE ANALYSIS

Adarsh Natarajan
Sunkari Sasidhar

Introduction

We describe the Omega Test which is a new method for dependence analysis.
It combines new methods from integer programming with variable elimination to produce a powerful technique.

The outline of the presentation is as follows

- Description of the steps involved in the algorithm.
- Intuitive proof of why this approach works

What is the OMEGA Test ?

- Input : An arbitrary set of linear equalities and inequalities
- Output : Whether an integer solution exists.
- Generally NP-complete.
- Worst case exponential complexity
- Usually polynomial time complexity

Outline of the Algorithm

- Reducing the problem to a simpler domain.
- Simpler domain has fewer variables.

  Done by **projecting** 'n' variables to 'n-1'.
- Solution in the simpler domain implies solution to the problem

OMEGA TEST-An overview

STEPS :

- Normalizing Constraints
- Dealing with Equality Constraints
- Dealing with Inequality Constraints

Adarsh Natarajan
Sunkari Sasidhar

Normalizing Equalities

Normalization for easy manipulation of constraints

Initial form : $\sum_{1 \le i \le n} a_i x_i = c$,

Define $x_0 = 1$ and $c = 0$

Scale rational coefficients to integers.

Let $g = \gcd(a_1, \ldots, a_n)$

For equalities $\gcd(a_1, \ldots, a_n)$ **must** divide a_0

Make $a_i = a_i / g$

Final form : $\sum_{0 \le i \le n} a_i x_i = 0$

Adarsh Natarajan
Sunkari Sasidhar

Normalising Inequalities

Initial form $\sum_{1 \le i \le m} a_i x_i \ge c$

Define $x_0 = 1$ and $c = 0$

Scale rational coefficients to integers.

Let $g = \gcd(a_1, \ldots, a_n)$

For inequalities set $a_0 = \text{floor}(a_0 / g)$

This is called *Tightening*.

Final Form $\sum_{0 \le i \le n} a_i x_i \ge 0$

An example of tightening

Consider $3x_1 + 6x_2 \geq 10$

gcd(3,6) = 3

On Normalizing, $x_1 + 2x_2 \geq 3$

Equivalent to $3x_1 + 6x_2 \geq 9$

Tighter than $3x_1 + 6x_2 \geq 10$

Adarsh Natarajan
Sunkari Sasidhar

Dealing with Equality Constraints

- Eliminate all equalities to produce a problem in inequalities.
- Each equality elimination eliminates a variable
- Selecting the variable to eliminate:
  - Select a variable x_j such that $|a_j| = 1$, if it exists
  - Otherwise, select x_j such that $|a_j|$ is the least.
- Eliminating the variable :
  - if $|aj| = 1$, solve for x_j and substitute in the system.
  - Otherwise, replace the equation with :

$$-|a_j|\sigma + \sum_{i \neq j} \left(\text{floor}(a_i/m + ½) + a_i \; mod \; m\right) x_i = 0$$

where $a \bmod b = a - b*\text{floor}(a/b + ½)$,

$m = |a_j| + 1$ and

$$m\sigma = \sum_{i=0}^{n} (a_i \bmod m) x_i$$

Adarsh Natarajan
Sunkari Sasidhar

Example of Equality Elimination

$-7x - 2y + 3z = 3$

$-24x - 7y + 11z = 10$

$1 \leq -8x - 4y - z - 1 \leq 40$

$-50 \leq y \leq 50$

---

substitute $y = x + 3\sigma$

$-3x - 2\sigma + z = 1$

$-31x - 21\sigma + 11z = 10$

$1 \leq -1 - 12x - 12\sigma - z \leq 40$

$-50 \leq x + 3\sigma \leq 50$

substitute $z = 3x + 2\sigma + 1$

$2x + \sigma = -1$

$1 \leq -2 - 15x - 14\sigma \leq 40$

$-50 \leq x + 3\sigma \leq 50$

Thus we have eliminated one equality. Similarly we eliminate the other equation to get

$1 \leq 12 + 3x \leq 40$

$-50 \leq -3 - 5x \leq 50$

Dealing with Inequalities

Do the following steps in order

Check for contradicting inequalities like

$x + y \geq 2$, $x + y \leq 1$

If present report – no solution.

·Remove inequalities expressed by tighter ones

· $x + y \geq 3$, $x + y \leq 3$ replaced by $x + y = 3$

·Remove redundant constraints

· $x + y \geq 3$, $x + y \geq 2$, $x + y \geq 2$ can be removed.

·If problem has only a single variable – then we report a solution to the set of constraints.

·Else *eliminate* one variable and recurse.

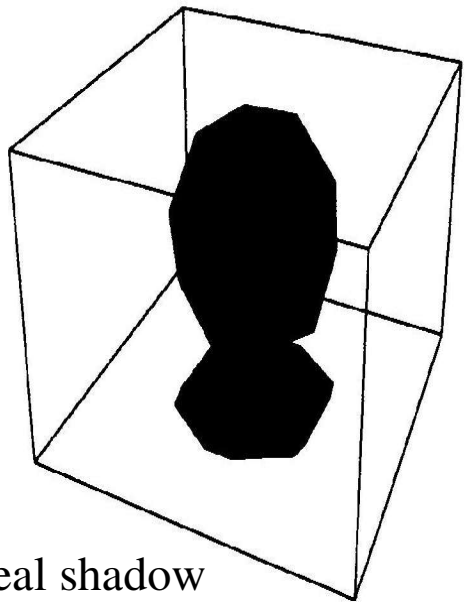
Variable Elimination

Using the *Fourier-Motzkin* method.

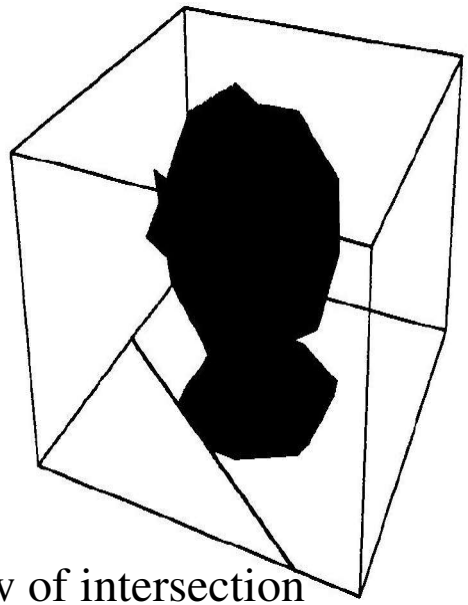
Intuition

- Think of each variable as representing a dimension in n-dimensional space.
- The set of constraints represent an object in this space
- Elimination proceeds by casting the n-1 dimensional shadow of the object.

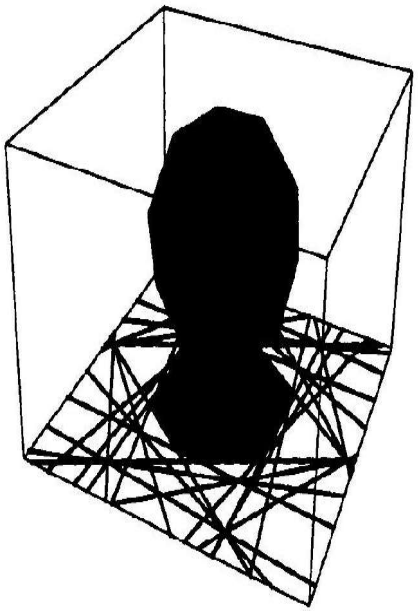
Shadows



real shadow



shadow of intersection
of 2 constraints



shadow of all constaints

Formalisation

Consider two constraints on z

$$\beta \leq bz \text{ , } az \leq \alpha \implies a\beta \leq abz \leq b\alpha$$

removing z , we get

$$a\beta \leq b\alpha$$

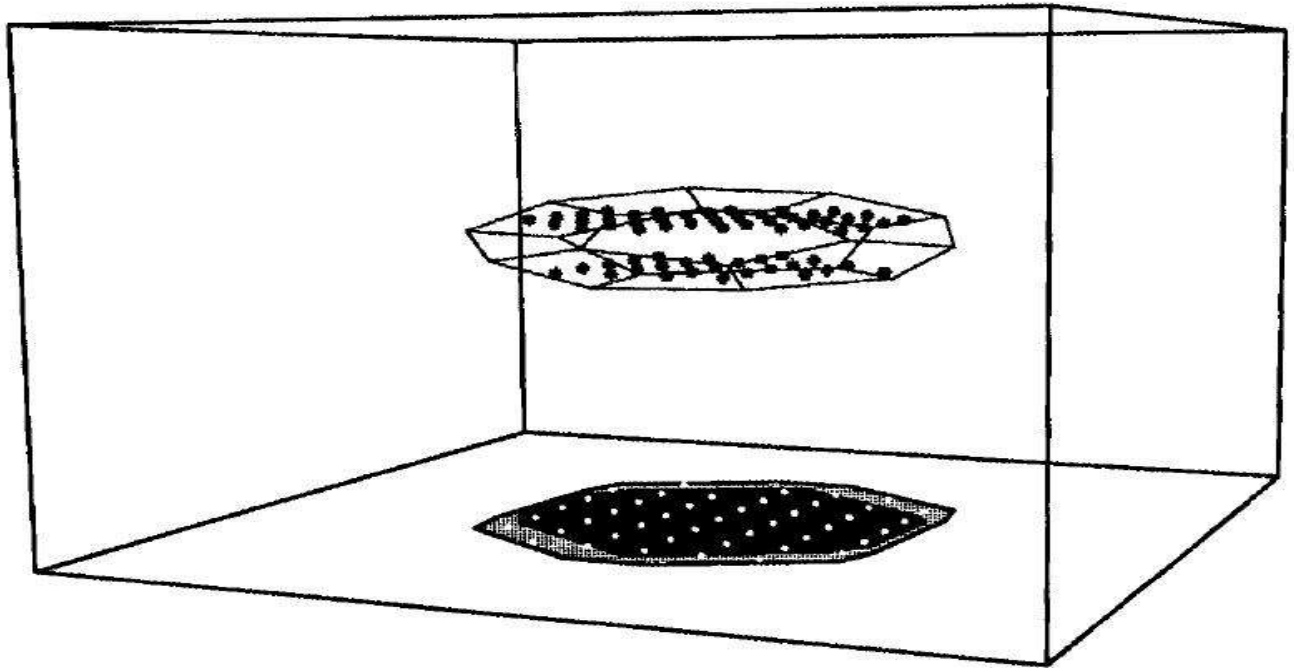
This defines the real shadow of the original object.

The *integer shadow* is a shadow such that for every integer point in the shadow there is alteast one corresponding integer point in the object.

Calculation of the integer shadow cannot always be done, we therefore define the *dark shadow* of the object which may be thought to be translucent.

Dark Shadow

For every integer point in the dark shadow, there is an integer point in the obiect.



Determining the dark shadow

The dark shadow differs from the real shadow when

- $a\beta \leq b\alpha$ has an integer solution
- $a\beta \leq abz \leq b\alpha$ has no integer solution

The dark shadow of $\alpha \geq az$ and $bz \geq \beta$

is $b\alpha - a\beta > (a - 1) * (b - 1)$

if a = 1 or b = 1 , then the dark shadow is identical to the real shadow – this is called an exact projection.

For most problems in dependence analysis exact projection is almost always possible.

Summary of the Omega Test

- Eliminate the variable which ensures an exact projection or one that has coeffcients as close to zero as possible
- Calculate the real and dark shadows
- If we have an exact projection, then
  integer points in the shadow => integer solutions to the constraints.
- If the projection is non-exact
- no integer points in the real shadow => no solution.
- integer points in the dark shadow => integer solutions.
- Otherwise check for solutions very close to the dark shadow by exhaustively checking each constraint.

Exhaustive checking however rarely occurs in practice.

Adarsh Natarajan
Sunkari Sasidhar

Additional features of Omega Test

- Allows handling of symbolic constants by integer programming methods by adding them as additional variables.
- Allows dependence analysis when no information is available about n

```
for i = 1 to n do
  a[i+n] = a[i]
```

$1 \leq i\_w , i\_r \leq n, \quad i\_w + n = i\_r$

- Allows non-linear specifications like max,min etc.
- Accomodates integer division and remainder, e = a div m

$0 \leq a - m\sigma \leq m - 1$, using σ as the value for e.

Adarsh Natarajan
Sunkari Sasidhar

Example

Consider the inequalities

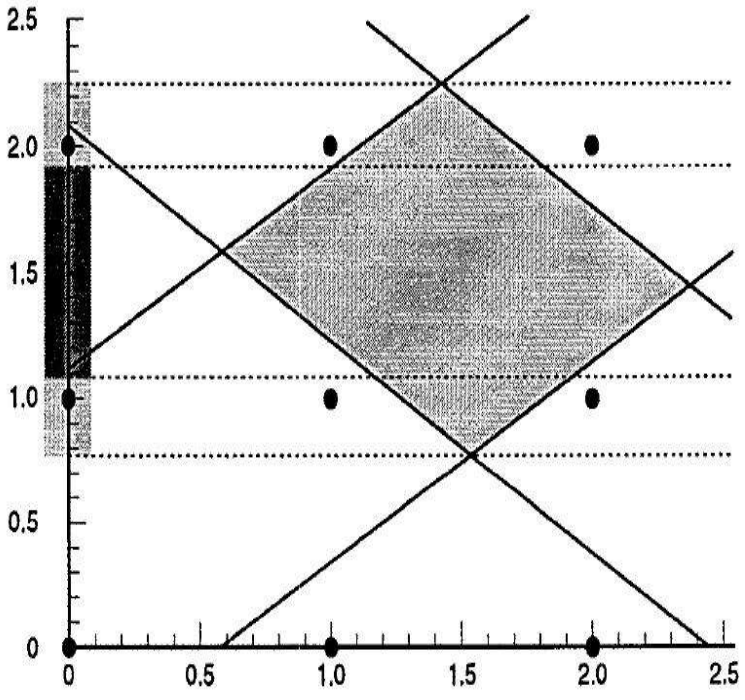
$$27 \leq 11x + 13y \leq 45$$
$$-10 \leq 7x - 9y \leq 4$$

No exact projection $b\alpha - a\beta > (a - 1) * (b - 1)$ for both inequalities, project with respect to x, since it has a slightly lesser coefficient.
The real shadow contains integer solutions , but these are spurious and are eliminated by checking through the shadow region

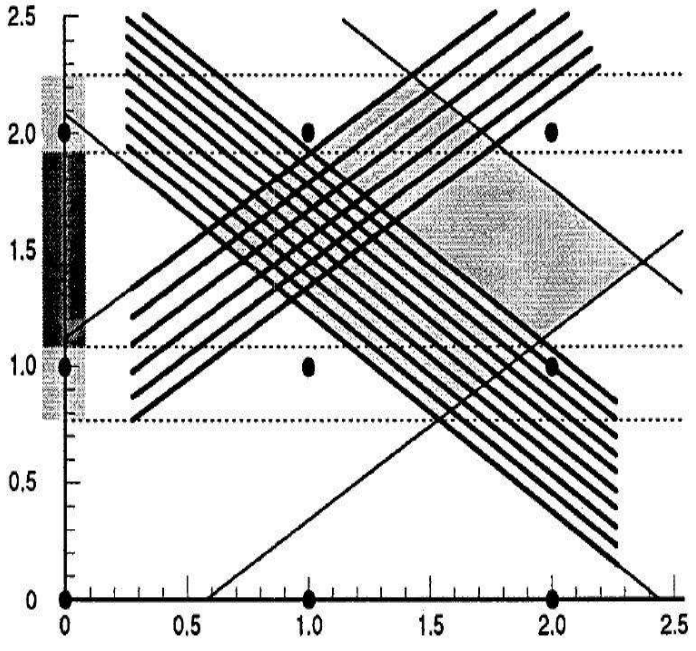
$7x = 9y - 10 + j,\ \ j = \text{floor}(7 - 1 - (7/11)) = 5$
$11x = 27 - 13y + j = \text{floor}(11 - 1 -(11/11)) = 11$

Adarsh Natarajan
Sunkari Sasidhar

Shape of Constraint space



Projection



Checking for spurious solutions

Advantages

- Gives more information on the dependence.
- Can project the the programming problem into a set of variables that describe all possible values of the other variables.
- Gives methods for calculation of the values of variables projected out.

eg) $\{a = 10b + 25c ; a \geq 13 \} \Rightarrow \{\sigma \geq 3 ; a = 5\sigma \}$

- Gives an accurate summary of the locations of an array affected by a single assignment.
- Can be used to determine loop bounds when interchanging non-rectangular loops.

Application to dependence testing

- Uses integer programming to determine if a dependence exists.
- Introduces a new variable for the dependence distance in a shared loop.
- Projects the system into a the dependence distance variables.
- Gives more information than dependence vectors.
- Dependence direction and distance vectors can also be efficiently determined.

Limitations

- In worst cases it can go propotional to absolute value of coefficients.
- It doesnot produce significant savings for typical simple cases.
- Cost of building dependence problem may be as large as the time spent in analysis.

Conclusion

- Omega test is a faster and practical method for performing data dependence analysis.
- It is adequate for problems encountered with sophisticated program transformation.
- Conventional analysis can also be done easily.

References

W.Pugh ,A practical algorithm for exact array dependence analysis
CACM, 3.5(8) Aug 1992