Farkas Lemma made easy
Tool Demo

Christophe Alias

Inria, LIP/ENS-Lyon, CNRS, UCBL

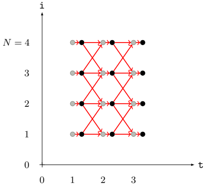
IMPACT'20 – January 22, 2020

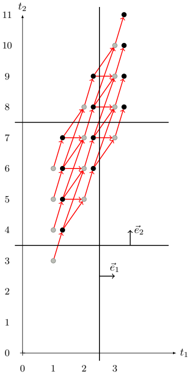
Introduction

- Many program analysis and transformations requires to handle constraints $\forall x \in \mathcal{P} : \phi(x) \geq 0$
- Examples: generation of invariants, termination analysis, loop scheduling, loop tiling
- **Trick:** Farkas lemma (affine form) eliminates universal quantification and (allows to) produce $\exists$ affine constraints
- **Challenge:** tricky algebraic manipulations, not easy to apply by hand, neither to implement.

http://foobar.ens-lyon.fr/fkcc/

```
    for t := 1 to T
      for i := 1 to N
S:      b[i] := a[i-1] + a[i]
          + a[i+1];
      for i := 1 to N
T:      a[i] := b[i];
```



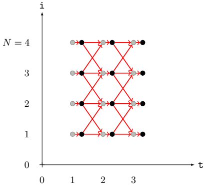


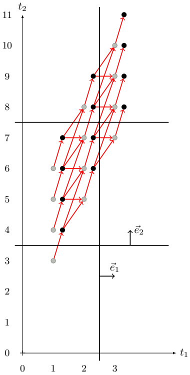
Orthogonal tiling after the affine transformation:

$$\phi_S : (t, i) \mapsto (t, 2t + i)$$
$$\phi_T : (t, i) \mapsto (t, 2t + i + 1)$$

Application: Pluto style loop tiling

```
    for t := 1 to T
      for i := 1 to N
S:      b[i] := a[i-1] + a[i]
          + a[i+1];
      for i := 1 to N
T:      a[i] := b[i];
```





Constraints

$\forall(t,i) \in D_S : \phi_S(t,i) \geq 0$ (Positivity)

$\forall(t,i,t',i') \in \Delta_{ST} : \phi_T(t',i') \geq \phi_S(t,i)$ (Causality)

$\forall(t,i,t',i') \in \Delta_{ST} : \phi_T(t',i') - \phi_S(t,i) \leq \delta(N)$ (Laziness)

Quadratic constraints & $\forall$ quantifiers $\rightsquigarrow$ **Farkas lemma**

Outline

Farkas lemma (affine form)

Lemma 1 (Farkas Lemma, affine form)

Let:

- $\mathcal{P} = \{\vec{x},\ A\vec{x} + \vec{b} \geq 0\} \subseteq \mathbb{R}^n$, $\mathcal{P} \neq \emptyset$
- $\phi : \mathbb{R}^n \to \mathbb{R}$ *an affine form*
- $\phi(x) \geq 0 \quad \forall x \in \mathcal{P}$

Then: $\exists \vec{\lambda} \geq \vec{0}, \lambda_0 \geq 0$ *such that:*

$$\phi(\vec{x}) = {}^t\vec{\lambda}(A\vec{x} + \vec{b}) + \lambda_0 \quad \forall \vec{x}$$

Notation

$$\mathfrak{F}(\lambda_0, \vec{\lambda}, A, \vec{b})(\vec{x}) = {}^t\vec{\lambda}(A\vec{x} + \vec{b}) + \lambda_0$$

Two practical corollaries

Corollary 2 (`solve`)

Consider a summation $S(\vec{x}) = \vec{u} \cdot \vec{x} + v + \sum_i \mathfrak{F}(\lambda_{i0}, \vec{\lambda}_i, A_i, \vec{b}_i)(\vec{x})$ *of affine forms, including Farkas terms. Then:*

$$\forall \vec{x} : \; S(\vec{x}) = 0 \quad \textit{iff} \quad \left\{ \begin{array}{l} \vec{u} + \sum_i \, {}^t A_i \vec{\lambda}_i = \vec{0} \, \wedge \\ v + \sum_i \left(\vec{\lambda}_i \cdot \vec{b}_i + \lambda_{0i} \right) = 0 \end{array} \right.$$

Corollary 3 (`define`)

$$\mathfrak{F}(\lambda_0, \vec{\lambda}, A, \vec{b})(\vec{x}) = (\, {}^t \vec{\lambda} A) \vec{x} + (\vec{\lambda} \cdot \vec{b} + \lambda_0)$$

Application to affine loop tiling (1/3)

Positivity

$$\phi_S(\vec{x}) \geq 0 \qquad \forall \vec{x} \in D_S$$

with: $D_S = \{\vec{x},\ A_S\vec{x} + \vec{b}_S \geq 0\}$

Apply Farkas

$\exists \lambda_0^S \geq 0, \vec{\lambda}^S \geq \vec{0}$:

$$\phi_S(\vec{x}) = \mathfrak{F}(\lambda_0^S, \vec{\lambda}^S, A_S, \vec{b}_S)(\vec{x})$$

Application to affine loop tiling (2/3)

Causality

$$\phi_T(\vec{y}) - \phi_S(\vec{x}) \geq 0 \qquad \forall(\vec{x}, \vec{y}) \in \Delta_{ST}$$

with: $\Delta_{ST} = \{(\vec{x}, \vec{y}),\ A_{ST}(\vec{x}, \vec{y}) + \vec{b}_{ST} \geq 0\}$

Apply Farkas

$\exists \lambda_0^{ST} \geq 0, \vec{\lambda}^{ST} \geq \vec{0}$:

$$\phi_T(\vec{y}) - \phi_S(\vec{x}) = \mathfrak{F}(\lambda_0^{ST}, \vec{\lambda}^{ST}, A_{ST}, \vec{b}_{ST})(\vec{x}, \vec{y})$$

Application to affine loop tiling (2/3)

Causality

$$\phi_T(\vec{y}) - \phi_S(\vec{x}) \geq 0 \qquad \forall(\vec{x},\vec{y}) \in \Delta_{ST}$$

with: $\Delta_{ST} = \{(\vec{x},\vec{y}),\ A_{ST}(\vec{x},\vec{y}) + \vec{b}_{ST} \geq 0\}$

Apply Farkas

$\exists \lambda_0^{ST} \geq 0, \vec{\lambda}^{ST} \geq \vec{0}$:

$$\phi_T(\vec{y}) - \phi_S(\vec{x}) = \mathfrak{F}(\lambda_0^{ST}, \vec{\lambda}^{ST}, A_{ST}, \vec{b}_{ST})(\vec{x},\vec{y})$$

Putting it all together

$$\mathfrak{F}(\lambda_0^{T}, \vec{\lambda}^{T}, [0\ A_T], \vec{b}_T) - \mathfrak{F}(\lambda_0^{S}, \vec{\lambda}^{S}, [A_S\ 0], \vec{b}_S) = \mathfrak{F}(\lambda_0^{ST}, \vec{\lambda}^{ST}, A_{ST}, \vec{b}_{ST})$$

$\rightarrow$ By Corollary 2, we obtain $\exists$ affine constraints!

Application to affine loop tiling (3/3)

Laziness: $\forall(\vec{x},\vec{y}) \in \Delta_{ST} : \phi_T(\vec{y}) - \phi_S(\vec{x}) \leq \delta(N)$

$$\delta(\vec{N}) \geq 0 \qquad \forall \vec{N} \in \mathcal{C} = \{\vec{N},\ A_{\mathcal{C}}\vec{N} + \vec{b}_{\mathcal{C}} \geq 0\}$$

Apply Farkas

$\exists \mu_0 \geq 0, \vec{\mu} \geq \vec{0}$:

$$\delta(\vec{N}) = \mathfrak{F}(\mu_0, \vec{\mu}, A_{\mathcal{C}}, \vec{b}_{\mathcal{C}})(\vec{N})$$

Application to affine loop tiling (3/3)

Laziness: $\forall(\vec{x},\vec{y}) \in \Delta_{ST} : \phi_T(\vec{y}) - \phi_S(\vec{x}) \leq \delta(N)$

$$\delta(\vec{N}) \geq 0 \qquad \forall \vec{N} \in \mathcal{C} = \{\vec{N},\ A_{\mathcal{C}}\vec{N} + \vec{b}_{\mathcal{C}} \geq 0\}$$

Apply Farkas

$\exists \mu_0 \geq 0, \vec{\mu} \geq \vec{0}$:

$$\delta(\vec{N}) = \mathfrak{F}(\mu_0, \vec{\mu}, A_{\mathcal{C}}, \vec{b}_{\mathcal{C}})(\vec{N})$$

Putting it all together

$\forall(\vec{x},\vec{y}) \in \Delta_{ST} : \delta(\vec{N}) - \phi_T(\vec{y}) + \phi_S(\vec{x}) \geq 0$ gives:

$$\mathfrak{F}(\lambda_0^T, \vec{\lambda}^T, [0\ A_{\mathcal{C}}], \vec{b}_{\mathcal{C}}) - \mathfrak{F}(\lambda_0^T, \vec{\lambda}^T, [0\ A_T], \vec{b}_T) + \mathfrak{F}(\lambda_0^S, \vec{\lambda}^S, [A_S\ 0], \vec{b}_S) = \mathfrak{F}(\lambda_0^{ST}, \vec{\lambda}^{ST}, A_{ST}, \vec{b}_{ST})$$

$\rightarrow$ By Corollary 2, we obtain $\exists$ affine constraints!

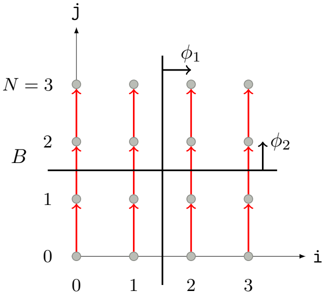
Back to the example...

Demo

Outline

fkcc: positive_on, solve

for i := 0 **to** N
  for j := 0 **to** N
B: a[i] := a[i] + 1;




```
D := [] -> { [i,j,N]: 0 <= i and i < N ...};
phi := positive_on D;

Delta := [] -> { [i,j,i',j',N]: ...};
to_target := {[i,j,i',j',N] -> [i',j',N]};
to_source := {[i,j,i',j',N] -> [i,j,N]};

solve (phi . to_target) - (phi . to_source)
                        - positive_on Delta = 0;
```

fkcc: define, keep

```
...
phi_correct :=
  (solve (phi . to_target) - (phi . to_source)
                           - positive_on Delta = 0) *
  (define phi with phi);
phi_correct;
keep phi_0,phi_1,phi_2,phi_3 in phi_correct;
```

console

```
$ fkcc < test.fk
[] -> {[lambda_0,lambda_1,lambda_2,lambda_3,lambda_4,lambda_5,lambda_6,lambda_7,lambda_8,
lambda_9,lambda_10,lambda_11,lambda_12,lambda_13,phi_0,phi_1,phi_2,phi_3] :
(((((-1*lambda_0)+lambda_1)+lambda_5)+(-1*lambda_6))+(-1*lambda_9))+lambda_10 >= 0 and
((((lambda_0+(-1*lambda_1))+(-1*lambda_5))+lambda_6)+lambda_9)+(-1*lambda_10) >= 0 and
[...]
((-1*lambda_0)+lambda_1)+phi_0 >= 0 and (lambda_0+(-1*lambda_1))+(-1*phi_0) >= 0 and
((-1*lambda_2)+lambda_3)+phi_1 >= 0 and (lambda_2+(-1*lambda_3))+(-1*phi_1) >= 0 and
[...]};

[] -> {[phi_0,phi_1,phi_2,phi_3] : phi_2+phi_3 >= 0 and phi_0+phi_2 >= 0 and phi_1 >= 0 and
phi_2 >= 0 and 1 >= 0};
```

fkcc: find, lexmin

find

`find v1, ..., vn s.t. <farkas> = 0` is a macro for:

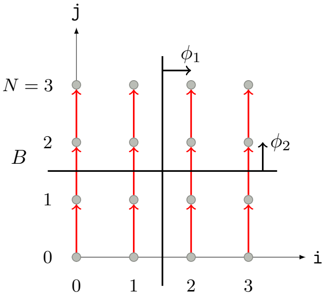
```
keep v1, ..., vn in
  (solve <farkas> = 0) * (define v1 with v1) * ...
```

```
...
phi_correct :=
  find phi s.t. (phi . to_target) - (phi . to_source)
                                   - positive_on Delta = 0
phi_correct;
lexmin phi_correct;
```

Outline

Demo: complete example

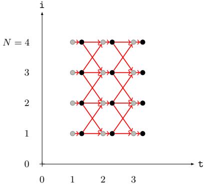
$$\textbf{for } i := 0 \textbf{ to } N$$
$$\quad \textbf{for } j := 0 \textbf{ to } N$$
$$\text{B:} \qquad \text{a[i]} := \text{a[i]} + 1;$$

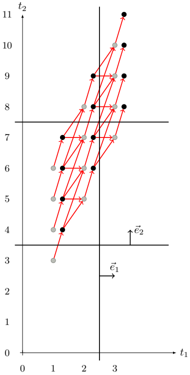


Expected:

$$\phi(i,j) = (i,j) \qquad \delta(N) = (0,1)$$

```
   for t := 1 to T
     for i := 1 to N
S:     b[i] := a[i-1] + a[i]
         + a[i+1];
     for i := 1 to N
T:     a[i] := b[i];
```





Expected:

$$\phi_S : (t, i) \mapsto (t, 2t + i)$$
$$\phi_T : (t, i) \mapsto (t, 2t + i + 1)$$
$$\delta(T, N) = (T, 2T)$$

Outline

Conclusion

- `fkcc`, a scripting tool to prototype program analysis and transformations using the affine form of Farkas lemma
- `fkcc` is powerful enough to write in a few lines tricky scheduling algorithms and termination analysis
- Object representation (polyhedron, affine functions) is compatible with `iscc`

http://foobar.ens-lyon.fr/fkcc/