La fattorizzazione cartesiana nella libreria PPLite

Luigi Zaccone

February 18, 2019

Indice

1 Introduzione

2 Fattorizzazione cartesiana

La fattorizzazione cartesiana permette di avere una complessità di esecuzione minore sulle operazioni applicate ai poliedri. L'idea è che i poliedri utilizzati nell'analisi dei programmi non mettono in relazione tutte le variabili del programma in un solo vincolo. Per esempio: un ipercubo
$H_n = \{0 \leq x_i \leq 1 \mid i = 1, ..., n\}$ richiede 2^n generatori per essere rappresentato, tramite la decomposizione invece ne sono necessari solo $2n$. La terminologia e gli esempi saranno ispirati al lavoro svolto in [1].

2.1 Blocchi e fattori

Sia $\mathcal{X} = \{x_1, x_2, ..., x_n\}$ un insieme di n variabili. Dato un poliedro, $\mathcal{X}$ può essere partizionato in un sottoinsieme $\mathcal{X}_k$ che chiamiamo *blocchi* tale che i vincoli esistono solo tra variabili presenti nello stesso *blocco*. Quindi, ogni variabile priva di vincoli risiede in un singoletto. Ci riferiamo a questo insieme come $\pi = \pi_P = \{\mathcal{X}_1, \mathcal{X}_2, ..., \mathcal{X}_t\}$.

Esempio 2.1.1. Consideriamo

$$\begin{aligned} &\mathcal{X} = \{x_1, x_2, x_3\} \ e \\ &P = \{x_1 + 2x_2 \leq 3\}. \end{aligned}$$

In questo caso $\mathcal{X}$ viene partizionato in due blocchi: $\mathcal{X}_1 = \{x_1, x_2\}$ e $\mathcal{X}_2 = \{x_3\}$.

Per ogni blocco, quindi, sarà presente un *fattore* P_k che sarà composto solo dalle variabili presenti nel suo blocco. In qualsiasi momento il poliedro originale può essere recuperato applicando l'unione dei vincoli $\mathcal{C}_{P_k}$ e il prodotto cartesiano dei generatori $\mathcal{G}_{P_k}$.

Esempio 2.1.2. Consideriamo un poliedro $\mathcal{P}$ con i seguenti vincoli e generatori:

$$\mathcal{C} = \{-x_1 \leq -1, x_1 \leq 4, -x_2 \leq -2, x_2 \leq 4\}$$
$$\mathcal{G} = \{\{(1),(2)\},\{(1),(4)\},\{(4),(2)\},\{(4),(4)\}\}$$

Il poliedro non ha vincoli tra le variabili x_1 e x_2. Quindi, $\mathcal{X} = \{x_1, x_2\}$ può essere partizionato nei blocchi: $\pi_P = \{\{x_1\},\{x_2\}\}$ con i risultanti fattori $P_1 = (\mathcal{C}_{P_1}, \mathcal{G}_{P_1})$ e $P_2 = (\mathcal{C}_{P_2}, \mathcal{G}_{P_2})$ dove:

$$\mathcal{C}_{P_1} = \{-x_1 \leq -1, x_1 \leq 4\} \qquad \mathcal{C}_{P_2} = \{-x_2 \leq -2, x_2 \leq 4\}$$
$$\mathcal{G}_{P_1} = \{\{(1),(4)\},\emptyset,\emptyset\} \qquad \mathcal{G}_{P_2} = \{\{(2),(4)\},\emptyset,\emptyset\}$$

Il poliedro originale può essere ricavato da P_1 e P_2 come P $= P_1 \bowtie P_2 = (\mathcal{C}_{P_1} \cup \mathcal{C}_{P_2}, \mathcal{G}_{P_1} \times \mathcal{G}_{P_2})$

L'insieme $\mathcal{L}$ che consiste in tutte le partizioni possibili di $\mathcal{X}$ forma un *reticolo di partizioni* $(\mathcal{L}, \sqsubseteq, \sqcup, \sqcap, \bot, \top)$. Gli elementi π del reticolo sono ordinati come segue: $\pi \sqsubseteq \pi'$, se ogni blocco di π è incluso in qualche blocco di π'. Questo reticolo contiene anche i soliti operatori di *least upper bound* ($\sqcup$) e *greatest upper bound* ($\sqcap$). È da notare che nel reticolo della partizione $\top = \{\mathcal{X}\}$ e $\bot = \{\{x_1\},\{x_2\},...,\{x_n\}\}$.

2.2 Partizione ammissibile

Una partizione *ammissibile* $\overline{\pi}$ è tale se, dato un poliedro P, non esistono variabili x_i e x_j in diversi blocchi di π relazionati da un vincolo di P, ovvero $\pi \sqsupseteq \pi_P$. Le partizioni ammissibili sono un sottoinsieme chiuso superiormente del reticolo, chiameremo questo sottoinsieme $\mathcal{B}$. Se $\overline{\pi} \in \mathcal{B}$ allora lo saranno anche tutte le $\overline{\pi}'$ tali che $\overline{\pi}, \overline{\pi}' \in \mathcal{L}$ e $\overline{\pi} \sqsubseteq \overline{\pi}'$. Le partizioni ammissibili sono dotate di elemento minimo ovvero la più fine partizione appartenente a $\mathcal{B}$, partizione che verrà anche considerata come la migliore. Questa partizione minima viene calcolata come spiegato nell'**esempio 2.1.1**.

Il nostro obiettivo è quello di cercare di utilizzare sempre la partizione ammissibile minima.

2.3 Operazioni

In questo paragrafo verrà spiegato l'effetto che alcune operazioni dei poliedri generano sui fattori. Innanzitutto è fondamentalche che, nelle operazioni binarie, i poliedri abbiano lo stesso numero di dimensioni. Per descrivere meglio queste operazioni, è necessario prima dare una specifica di alcune funzioni utilizzate all'interno delle prime.

Convert Constraint. Funzione che mappa le dimensioni di un vincolo o generatore all'interno di un fattore. La PPLite utilizza dimensioni nel range $[0, ..., n]$ per i poliedri. Tramite la fattorizzazione è quindi necessario tradurre gli indici esterni a quelli interni dei vari blocchi.

Least Upper Bound. Questa funzione estrae il *lub* di due partizioni. Dall'Algoritmo 1 si può notare come sia necessario generare l'insieme delle unioni dei due blocchi $\pi_{\mathcal{P}}$ e $\pi_{\mathcal{Q}}$ in $\overline{\pi}$. Successivamente si uniscono tra loro tutti i blocchi di $\overline{\pi}$ che hanno un'intersezione non vuota, calcolando successivamente la partizione corretta.

Algoritmo 1 Least-Upper-Bound

function Least-Upper-Bound($\pi_{\mathcal{P}}, \pi_{\mathcal{Q}}$)
    $\overline{\pi} := \emptyset$
    for each $\mathcal{X}_{\mathcal{P}i}$ **in** $\pi_{\mathcal{P}}$ **do**
        $\mathcal{A} := \emptyset$
        for each $\mathcal{X}_{\mathcal{Q}k} in \pi_{\mathcal{Q}}$ **do**
            if $\mathcal{X}_{\mathcal{P}i} \cap \mathcal{X}_{\mathcal{Q}k} \neq \emptyset$ **then**
                $\mathcal{A} := \mathcal{B} \cup \mathcal{X}_{\mathcal{Q}k}$
        $\overline{\pi}$.add($\mathcal{A}$)
    while $\exists$ coppie $(\overline{\pi}_i, \overline{\pi}_j)$, $i \neq j$ **t.c.** $\overline{\pi}_i \cap \overline{\pi}_j \neq \emptyset$ **do**
        $\overline{\pi} := \_\{\overline{\pi}_i, \overline{\pi}_j\} \cup \{\overline{\pi}_i \cup \overline{\pi}_j\}\}$
    return $\overline{\pi}$

Merge. Definito come $\uparrow$, quando si ha l'operazione $\pi \uparrow \mathcal{A}$, dove π è una partizione di un poliedro e $\mathcal{A}$ è un sottoindieme delle sue variabili (non per forza connesse da vincoli), genera una partizione π_{merge} tale che:

- $\exists\ \mathcal{X}_i \in \pi_{merge} : \mathcal{A} \subseteq \mathcal{X}_i$
- $\pi \sqsubseteq \pi_{merge}$

Viene quindi generata una nuova fattorizzazione unendo i blocchi del poliedro che hanno variabili in comune con $\mathcal{A}$. Questa unione genera un blocco unico $\mathcal{D}$, tale che $\mathcal{A} \subseteq \mathcal{B}$.

Algoritmo 2 Merge

function MERGE($\pi, \mathcal{A}$)
    $\pi_{\mathcal{O}} := \emptyset$
    $\mathcal{D} := \emptyset$
    for each $\mathcal{X}_i$ **in** π **do**
        if $\mathcal{X}_i \cap \mathcal{A} \neq \emptyset$ **then**
            $\mathcal{D} := \mathcal{D} \cup \mathcal{X}_i$
        else
            $\pi_{\mathcal{O}}$.add($\mathcal{X}_i$)
    $\pi_{\mathcal{O}}$.add($\mathcal{D}$)
    return $\pi_{\mathcal{O}}$

Refactor. Questa operazione ha come argomenti un poliedro fattorizzato e una fattorizzazione ammissibile $\overline{\pi}$ per questo poliedro. Non fa altro che convertire i fattori rispetto alla seconda fattorizzazione. È importante notare che la partizione ammissibile non sarà più fine di $\pi_{\mathcal{P}}$, visto che i blocchi non vengono divisi bensì solo uniti.

La refactor è un'operazione necessaria in quanto, avendo due poliedri $\mathcal{P}$ e $\mathcal{Q}$ definiti sullo stesso insieme di variabili $X = \{x_1, x_2, ..., x_n\}$ e presi $\overline{\pi}_{\mathcal{P}} = \{X_{\mathcal{P}_1}, X_{\mathcal{P}_2}, ..., X_{\mathcal{P}_r}\}$, $\overline{\pi}_{\mathcal{Q}} = \{X_{\mathcal{Q}_1}, X_{\mathcal{Q}_2}, ..., X_{\mathcal{Q}_s}\}$ partizioni corrette di $\mathcal{P}$ e $\mathcal{Q}$ in genere abbiamo che $\overline{\pi}_{\mathcal{P}} \neq \overline{\pi}_{\mathcal{Q}}$. È necessario che, come in molte operazioni comuni nell'analisi e manipolazione dei poliedri, le dimensioni delle partizioni utilizzate siano uguali in quantità e valori. Se si applica la fattorizzazione su poliedri aventi gli stessi blocchi non si fa altro che calcolare il *lub* ($\overline{\pi} = \overline{\pi}_{\mathcal{P}} \sqcup \overline{\pi}_{\mathcal{Q}}$).

Algoritmo 3 Refactor

function REFACTOR($\mathcal{P}, \pi_{\mathcal{P}}, \overline{\pi}$)
    $\mathcal{O}, \pi_{\mathcal{O}} := \emptyset$
    for each i **in** $\{1, ..., r\}$ **do** ▷ r = numero blocchi di $\pi_{\mathcal{P}}$
        $\mathcal{O}_i, \pi_{\mathcal{O}i} := \emptyset$
        for each j **in** $\{1, ..., m\}$ **do** ▷ m = numero blocchi di $\overline{\pi}$
            if $\pi_{\mathcal{P}i} \cap \overline{\pi}_i \neq \emptyset$ **then**
                $\mathcal{O}_i := \mathcal{O}_j \bowtie \mathcal{P}_i$
                $\pi_{\mathcal{O}i} := \pi_{\mathcal{O}i} \bowtie \pi_{\mathcal{P}i}$
        $\mathcal{O}$.add($\mathcal{O}_i$)
        $\pi_{\mathcal{O}}$.add($\pi_{\mathcal{O}i}$)
    REMAP-DIMENSIONS($\mathcal{O}, \pi_{\mathcal{O}}, \overline{\pi}$)
    return $\mathcal{O}, \pi_{\mathcal{O}}$

Come spiegato precedentemente, la refactor prende in input un poliedro fattorizzato $\mathcal{P}$ e la sua partizione $\pi_{\mathcal{P}}$ per rifattorizzarlo in base al terzo parametro $\overline{\pi}$. Non fa altro che unire i fattori che hanno intersezione dei relativi blocchi non vuota tra $\pi_{\mathcal{P}}$ e $\overline{\pi}$ e generare il blocco finale. È importante ri-mappare le dimensioni dei fattori di output in base a quelle descritte nel blocco di $\overline{\pi}$.

2.3.1 Inclusion Test ($\sqsubseteq$)

Operazione necessaria per controllare se un poliedro è incluso in un altro, disponendo di una doppia rappresentazione è possibile controllarlo se, dati due poliedri $\mathcal{P}$ e $\mathcal{Q}$ tutti i generatori in $\mathcal{G}_{\mathcal{P}}$ soddisfano tutti i vincoli in $\mathcal{C}_{\mathcal{Q}}$.

Nel nostro caso e, come mostrato nell'Algoritmo 4, è stata implementata tramite wrapper. In particolare vengono rifattorizzati i poliedri con il *lub* $\pi_{\mathcal{P}} \sqcup \pi_{\mathcal{Q}}$ e successivamente viene applicata l'*inclusion* in ordine su ogni coppia di fattori: il test avrà successo se tutti i test ritornano *True*.

Algoritmo 4 Inclusion Test

```
function INCLUSION(𝒫, π_𝒫, 𝒬, π_𝒬)
    π̄ := LEAS-UPPER-BOUND(π_𝒫, π_𝒬)
    𝒫' := REFACTOR(𝒫, π_𝒫, π̄)
    𝒬' := REFACTOR(𝒬, π_𝒬, π̄)
    for each k in {1, ..., r} do                ▷ r = numero di blocchi in π̄
        if 𝒫'_k non include 𝒬'_k then
            return False
    return True
```

2.3.2 Join ($\sqcup$)

Tramite la doppia reppresentazione, i generatori $\mathcal{G}_{\mathcal{O}}$ dove $\mathcal{O}$ è il risultato del *join*, sono semplicemente l'unione dei generatori dei poliedri presi in input dalla funzione, ovvero $\mathcal{G}_{\mathcal{O}} = \mathcal{G}_{\mathcal{P}} \cup \mathcal{G}_{\mathcal{Q}}$. I vincoli $\mathcal{C}_{\mathcal{O}}$ sono ottenuti aggiungendo incrementalmente i generatori di $\mathcal{G}_{\mathcal{Q}}$ al poliedro definito da $\mathcal{C}_{\mathcal{P}}$.

Algoritmo 5 Join

```
function JOIN(𝒫, π_𝒫, 𝒪, π_𝒪)
    if IS_EMPTY(𝒫) then
        𝒪, π_𝒪 := 𝒬, π_𝒬
    else if IS_EMPTY(𝒬) then
        𝒪, π_𝒪 := 𝒫, π_𝒫
    else
        𝒪, π_𝒪 := JOIN-POLY(𝒫, π_𝒫, 𝒪, π_𝒪)
    return 𝒪, π_𝒪
```

Avendo un sistema di poliedri fattorizzati, è necessario per prima cosa rifattorizzare $\mathcal{P}$ e $\mathcal{Q}$ utilizzando il loro *lub*. Per ogni coppia di fattori $(\mathcal{P}'_i, \mathcal{Q}'_i)$ uguali, se ne aggiunge uno con il rispettivo blocco $\overline{\pi}_i$ al risultato. Se invece i due fattori risultano diversi, ognuno viene unito a un fattore comune $(\mathcal{P}'_i, \mathcal{Q}'_i)$ e successivamente si calcola la *join* di questi due fattori e si inserisce insieme al rispettivo blocco nel poliedro di output.

Un caso speciale è stato assegnato ai poliedri *vuoti*. Dati $\mathcal{P}$ come poliedro *vuoto* e $\mathcal{Q}$ un poliedro generico allora $\mathcal{Q} \cup \mathcal{P} = \mathcal{Q}$ e anche $\mathcal{P} \cup \mathcal{Q} = \mathcal{Q}$.

Se invece $\overline{\pi} = \overline{\pi}_{\mathcal{P}} \sqcup \overline{\pi}_{\mathcal{Q}}$ e $\mathcal{U} = \{\mathcal{X}_k \mid \mathcal{P} = \mathcal{Q}, \mathcal{X}_k \in \overline{\pi}\}$ allora la partizione ammissibile, in questo caso, sarà uguale a:

$$\overline{\pi}_{\mathcal{P} \sqcup \mathcal{Q}} = \mathcal{U} \cup \bigcup_{\mathcal{T} \in \overline{\pi} \setminus \mathcal{U}} \mathcal{T}$$

Algoritmo 6 Join-Poly

function Join-Poly($\mathcal{P}, \pi_{\mathcal{P}}, \mathcal{O}, \pi_{\mathcal{O}}$)
$\overline{\pi} := \pi_{\mathcal{P}} \sqcup \pi_{\mathcal{Q}}$
$\mathcal{P}' :=$ Refactor($\mathcal{P}, \pi_{\mathcal{P}}, \overline{\pi}$)
$\mathcal{Q}' :=$ Refactor($\mathcal{Q}, \pi_{\mathcal{Q}}, \overline{\pi}$)
for each i **in** $\{0, ..., |\mathcal{P}'|\}$ **do**
if $\mathcal{P}'_i = \mathcal{Q}'_i$ **then**
$\mathcal{O}$.add($\mathcal{P}'_i$)
$\pi_{\mathcal{O}}$.add($\mathcal{X}'_i$)
else
$\mathcal{X}_{\mathcal{T}} := \mathcal{X}_{\mathcal{T}} \cup \overline{\pi}_i$
$\mathcal{P}_{\mathcal{T}} := \mathcal{P}_{\mathcal{T}} \bowtie \mathcal{P}'_i$
$\mathcal{Q}_{\mathcal{T}} := \mathcal{Q}_{\mathcal{T}} \bowtie \mathcal{Q}'_i$
$\mathcal{P}_{\mathcal{T}} := \mathcal{P}_{\mathcal{T}} \sqcap \mathcal{Q}_{\mathcal{T}}$
$\pi_{\mathcal{O}}$.add($\mathcal{X}_{\mathcal{T}}$)
$\mathcal{O}$.add($\mathcal{P}_{\mathcal{T}}$)
return $\mathcal{O}, \pi_{\mathcal{O}}$

2.3.3 Meet ($\sqcap$)

Per la doppia rappresentazione, $\mathcal{P} \sqcap \mathcal{Q}$ genera un poliedro i cui vincoli $\mathcal{C}_{\mathcal{P} \sqcap \mathcal{Q}}$ sono risultati dall'unione di $\mathcal{C}_{\mathcal{P}}$ e $\mathcal{C}_{\mathcal{Q}}$, mentre $\mathcal{G}_{\mathcal{P} \sqcap \mathcal{Q}}$ si ottiene aggiungendo incrementalmente i vincoli di $\mathcal{C}_{\mathcal{Q}}$ al poliedro $\mathcal{P}$. Se $\mathcal{P} \sqcap \mathcal{Q}$ risulta non soddisfacibile allora $\mathcal{P} \sqcap \mathcal{Q} = \perp$ e $\mathcal{G}_{\mathcal{P} \sqcap \mathcal{Q}} = \emptyset$.

Per quanto riguarda i poliedri fattorizzati è necessario generare $\overline{\pi} = \overline{\pi}_{\mathcal{P}} \sqcap \overline{\pi}_{\mathcal{Q}}$ e rifattorizzare i due poliedri $\mathcal{P}$ e $\mathcal{Q}$ tramite essa, generando quindi $\mathcal{P}'$ e $\mathcal{Q}'$. Per tutte le r coppie di fattori $\mathcal{P}'_i$ e $\mathcal{Q}'_i$, se sono uguali aggiungo $\mathcal{P}'_i$ con il rispettivo blocco al poliedro di output $\mathcal{O}$, altrimenti creo un fattore $\mathcal{F} = \mathcal{P}'_i \sqcap \mathcal{Q}'_i$. Se $\mathcal{F}$ dovesse risultare vuoto, allora il risultato dell'intera operazione di *meet* sarebbe un poliedro $\perp$. Se al contrario non fosse vuoto, aggiungo $\mathcal{F}$ a $\mathcal{O}$ per poi riprender il ciclo fino all'esaurimento dei fattori e ritornando infine $\mathcal{O}$ e il suo blocco (rappresentato dal *lub* di $\pi_{\mathcal{P}}$ e $\pi_{\mathcal{Q}}$) come risultato. $\overline{\pi}_{\mathcal{P}\sqcap\mathcal{Q}} = \pi_{\mathcal{P}} \sqcap \pi_{\mathcal{Q}}$ è una partizione ammissibile se $\mathcal{P} \sqcap \mathcal{Q} \neq \perp$, altrimenti $\perp$ è ammissibile.

Algoritmo 7 Meet

function $\textsc{Meet}(\mathcal{P}, \pi_{\mathcal{P}}, \mathcal{Q}, \pi_{\mathcal{Q}})$
  $\overline{\pi} := \pi_{\mathcal{P}} \sqcup \pi_{\mathcal{Q}}$
  $\mathcal{P}' := \textsc{Refactor}(\mathcal{P}, \pi_{\mathcal{P}}, \overline{\pi})$
  $\mathcal{Q}' := \textsc{Refactor}(\mathcal{Q}, \pi_{\mathcal{Q}}, \overline{\pi})$
  $\mathcal{O} := \emptyset$
  for each i **in** $\{1, ..., r\}$ **do** $\triangleright$ r = numero di blocchi in $\overline{\pi}$
    if $\mathcal{P}'_i = \mathcal{Q}'_i$ **then**
      $\mathcal{O}$.add($\mathcal{P}'_i$)
    else
      $\mathcal{F} := \mathcal{P}'_i \bowtie \mathcal{Q}'_i$
      if $\textsc{is\_empty}(\mathcal{F})$ **then**
        return $\perp, \perp$
      $\mathcal{O}$.add($\mathcal{F}$)
  $\pi_{\mathcal{O}} := \overline{\pi}$
  return $\mathcal{O}, \pi_{\mathcal{O}}$

2.3.4 Conditional

Questa operazione viene utilizzata per aggiungere nuovi vincoli tra le variabili di un poliedro. Tramite la doppia rappresentazione è possibile aggiungendo un vincolo arbitrario c all'insieme $\mathcal{C}_{\mathcal{P}}$. Se dopo l'inserimento il sistema di vincoli risulta insoddisfacibile allora il poliedro diventa vuoto. Il sistema di generatori è, come sempre, ricavato dall'aggiunta incrementale del vincolo c nel poliedro attraverso la conversione.

Utilizzando poliedri fattorizzati è necessare ricavare il blocco $\mathcal{B}$ che contiene le variabili utilizzate nel vincolo, rifattorizzando $\mathcal{P}$ con $\overline{\pi}_\mathcal{P} \uparrow \mathcal{B}$ si genera $\mathcal{P}'$. Successivamente bisogna prendere il fattore relativo contenente le variabili di $\mathcal{B}$, convertire il vincolo con le dimensioni *interne* del fattore e aggiungerlo a quest'ultimo utilizzando un'operazione per normali poliedri. Si controlla, infine, che il sistema di vincoli sia ancora soddisfacibile e, in caso negativo i blocchi e i fattori in $\bot$ vengono modificati e il poliedro si rende *vuoto*. È importante far notare che, dato $\mathcal{O}$ un poliedro risultante dall'aggiunta di un vincolo, $\overline{\pi}_\mathcal{O} = \overline{\pi}_\mathcal{P} \uparrow \mathcal{B}$ è ammissibile se $\mathcal{O} \neq \bot$, altrimenti $\overline{\pi}_\mathcal{O} = \bot$.

Algoritmo 8 Conditional

function CONDITIONAL(con)
  $\mathcal{B}$:= EXTRACT_BLOCK($\mathcal{P}$, $\pi_\mathcal{P}$, con)
  $\overline{\pi} := \pi_\mathcal{P} \uparrow \mathcal{B}$
  $\mathcal{P}'$:= REFACTOR($\mathcal{P}, \pi_\mathcal{P}, \overline{\pi}$)
  for each i **in** $\{1, ..., r\}$ **do** ▷ r = numero di blocchi in $\overline{\pi}$
    if $\mathcal{B} \cap \overline{\pi}_i \neq \emptyset$ **then**
      con_{int} := CONVERT_CON($\overline{\pi}_i$, con)
      $\mathcal{F}$:= ADD_CON($\mathcal{P}'_i, con_{int}$)
      $\mathcal{O}$.add($\mathcal{F}$)
      if IS_EMPTY($\mathcal{O}_\rangle$) **then**
        return $\bot, \bot$
      $\mathcal{O}$.add($\mathcal{P}'_\rangle$)
  $\pi_\mathcal{O} := \overline{\pi}$
  return $\mathcal{O}, \pi_\mathcal{O}$

2.3.5 Assignment

Per quanto riguarda il dominio dei poliedri fattorizzati, l'operazione di *assignment* è molto simile alla *conditional*. Dopo aver estratto il blocco $\mathcal{B}$ che indica le variabili utilizzate nell'espressione lineare, si fattorizza il poliedro $\mathcal{P}$ con $\overline{\pi}_\mathcal{P} = \pi_\mathcal{P} \uparrow \mathcal{B}$ ottenendo $\mathcal{P}'$, successivamente si prende il blocco con le variabili di $\mathcal{B}$ e si applica al relativo fattore un *assignment* come per i poliedri non fattorizzati, rimappando correttamente le variabili esterne rispetto a quelle interne al fattore. La partizione ammissibile per un poliedro risultante da questa operazione è $\overline{\pi}_{\mathcal{P}} \uparrow \mathcal{B}$.

Algoritmo 9 Assignment

function $\textsc{Assignment}(\mathcal{P}, \pi_{\mathcal{P}}, stmt)$
    let $stmt = (x_i = ax + \epsilon)$
    $\mathcal{B} := \textsc{extract\_block}(stmt)$
    $\overline{\pi} := \pi_{\mathcal{P}} \uparrow \mathcal{B}$
    $\mathcal{P}' := \textsc{refactor}(\mathcal{P}, \pi_{\mathcal{P}}, \overline{\pi})$
    for each i **in** $\{1, ..., r\}$ **do**   ▷ $r = |\mathcal{P}|$
        if $\overline{\pi}_i \subseteq \mathcal{B}$ **then**
            $stmt_{int} := \textsc{convert}(\overline{\pi}, stmt)$
            $\mathcal{F} := \textsc{assignment}(\mathcal{P}'_i, stmt_{int})$ ▷ Poliedri non fattorizzati
            $\mathcal{O}$.add($\mathcal{F}$)
            if $\mathcal{P}'_i := \emptyset$ **then**
                return $\bot, \bot$
        else
            $\mathcal{O}$.add($\mathcal{P}'_i$)
    return $\mathcal{O}, \overline{\pi}$

2.3.6 Widening (∇)

Per la doppia rappresentazione, l'operatore di *widening* necessità come parametri i generatori e i vincoli di $\mathcal{P}$ e i vincolo di $\mathcal{Q}$. Il risultato dell'operazione $\mathcal{P}\nabla\mathcal{Q}$ conterrà i vincoli $\mathcal{C}_{\mathcal{Q}}$ che sono presenti in $\mathcal{C}_{\mathcal{P}}$ o che possono sostituirne un vincolo senza cambiare $\mathcal{P}$.

Per implementarlo è necessario effettuare uan rifattorizzazione $\mathcal{P}'$ del primo argomento $\mathcal{P}$ con $\overline{\pi} = \pi_{\mathcal{P}} \sqcup \pi_{\mathcal{Q}}$, mentre non è necessario rifattorizzare il secondo argomento. Successivamente, per ogni fattore $\mathcal{Q}_i$ del secondo argomento si individua il fattore corrispondente $\mathcal{P}'_k$ in $\mathcal{P}'$ e si procede a individuare l'insieme $\mathcal{C}_{\mathcal{O}_i}$ di vincoli di $\mathcal{Q}_i$ che sono *stabili* rispetto a $\mathcal{P}'_k$: questi vincoli costituiscono il risultato del widening rispetto a quel fattore. È possibile, come caso speciale, che se $\pi_{\mathcal{P}'_k} = \pi_{\mathcal{Q}_i}$, allora è possibile applicare direttamente l'operatore di widening per poliedri non fattorizzati.

Algoritmo 10 Meet

function $\textsc{Widenind}(\mathcal{P}, \pi_{\mathcal{P}}, \mathcal{Q}, \pi_{\mathcal{Q}})$
    $\overline{\pi} := \pi_{\mathcal{P}} \sqcup \pi_{\mathcal{Q}}$
    $\mathcal{P}' := \textsc{Refactor}(\mathcal{P}, \pi_{\mathcal{P}}, \overline{\pi})$
    $\mathcal{O} := \emptyset$
    for each i **in** $\{1, ..., r\}$ **do** ▷ q = numero di blocchi in $\pi_{\mathcal{Q}}$
        $\mathcal{C}_{\mathcal{O}_i} := \emptyset$
        $k := j$, t.c. $\mathcal{X}_{\mathcal{Q}_i} \subseteq \mathcal{X}_j, \mathcal{X}_j \in \overline{\pi}$
        if $\overline{\pi}_k = \overline{\pi}_{\mathcal{Q}_i}$ **then**
            $\mathcal{Q}_i := \textsc{Widening}(\mathcal{P}'_k, \mathcal{Q}_i)$
        else
            $\mathcal{C}_{\mathcal{O}_i} := \textsc{select\_stable\_con}(\mathcal{C}_{\mathcal{Q}_i}, \mathcal{P}'_k)$
            $\mathcal{Q}_i := \textsc{new\_factor}(\mathcal{C}_{\mathcal{O}_i})$
        $\mathcal{O}$.add$(\mathcal{O}_i)$
    return $\mathcal{O}, \pi_{\mathcal{O}}$

3 Implementazione

Questo capitolo tratterà dell'implementazione effettiva dei poliedri fattorizzati e di alcune loro operazioni. La struttura è stata basata su di un *wrapper* preesistente nella *PPlite* mentre alcune delle implementazioni sono state attuate effettuando un porting dalla libreria *PPL*. La maggiorparte delle operazioni sono state sviluppate come un wrapper per la *PPLite*, difatti le operazioni sui singoli poliedri sono gestite da essa. Lo scopo principale è quello di inserirsi all'interno dell'analisi statica e utilizzare i poliedri fattorizzati se e quando vengono generati poliedri che implicano un alto overhead per essere gestiti, in questo caso i suddetti poliedri vengono fattorizzati in maniera tale da diminuire il carico computazionale. L'implementazione è stata sviluppata interamente in *C++11* con l'obiettivo di utilizzare i metodi della *Standard Library* così da incrementare la leggibilità del codice. L'efficienza non è stata presa come obiettivo principale durante lo sviluppo, dando importanza alla correttezza e alla leggibilità del codice.

3.1 `F_Poly`

La classe dove sono compresi gran parte dei medoti è `F_Poly`. È stata definita nel file `F_Poly.hh` dove sono presenti 92 metodi pubblici e 14 metodi privati.

Per la costruzione/distruzione degli oggetti è stata seguita la *Rule of Five*, quindi abbiamo:

- Un costruttore esplicito, un costruttore per copia, un costruttore per spostamento
- Operazioni di assegnamento per copia e per spostamento;
- Un distruttore;

Abbiamo anche:

- Un metodo `check_inv()` per controllare che le invarianti di classe non siano state violate;
- Le operazioni principali descritte nel Capitolo 2.3;
- Funzioni di appoggio per gestire il poliedro;

Sono inoltre presenti delle stutture dati per la gestione dei fattori:

- `Block`: un `std::vector`[1] di `dim_type` che rappresenta un blocco $\mathcal{X}_i$ che contiene alcune dimensioni del poliedro;
- `Blocks`: un `std::vector` di `Block` che esprime le partizioni del poliedro;
- `Factor`: un fattore è un poliedro, difatti `Factor` è un alias per oggetti di tipo `Poly`;
- `Factors`: un `std::vector` di `Factor`;

Sono inoltre presenti tre variabili:

- `dim_type dim`: indicante le dimensioni del poliedro
- `Topol topol`: indicante la topologia del poliedro
- `bool is_empty`: indicante se il poliedro è vuoto o no

3.1.1 CHECK_INV()

Il metodo menzionato contiene tutti i controlli necessari per verificare che l'invariante di classe non sia stata violata, e che quindi il poliedro su cui si sta lavorando è in condizioni di correttezza. Le invarianti di classe che devono essere rispettate sono le seguenti:

- La dimensione minima del poliedro deve essere ≥ 0

[1] `template<class T, class Allocator = std::allocator<T>> class vector;`

```
if (dim < 0) {
  reason = "F_Poly broken: invalid space dimension";
  maybe_dump();
  return false;
}
```

- Se il poliedro è vuoto, i fattori e i blocchi devono essere anch'essi vuoti

```
if (is_empty()) {
  if (!(factors.empty() && blocks.empty())) {
    reason = "F_Poly broken: empty polyhedron has factors";
    maybe_dump();
    return false;
  }
  // No other check for an empty polyhedron.
  return true;
}
```

- Il numero di blocchi deve essere uguale al numero di fattori

```
if (factors.size() != blocks.size()) {
  reason = "F_Poly broken: #factors != #blocks";
  maybe_dump();
  return false;
}
```

- La cardinalità dei blocchi deve essere uguale a `dim`

```
dim_type dim_ = 0;
for (const auto& block : blocks)
  dim_ += block.size();
if (dim != dim_) {
  reason = "F_Poly broken: space dimension mismatch";
  maybe_dump();
  return false;
}
```

- Le dimensioni di un fattore devono essere uguali alle dimensioni del blocco che lo rappresenta

```
// Each factor space dim should match its block.
for (dim_type i = 0; i < num_rows(factors); ++i)
  if (factors[i].space_dim() != num_rows(blocks[i])) {
    reason = "F_Poly broken: factor vs block space dim mismatch";
    maybe_dump();
    return false;
  }
```

- Nessun fattore deve essere vuoto

```
if (std::any_of(factors.begin(), factors.end(),
                std::mem_fn(&Poly::is_empty))) {
  reason = "F_Poly broken: empty factor";
  maybe_dump();
  return false;
}
```

- Ogni dimensione dello spazio deve apparire una e una volta soltanto nei blocchi e non devono esserci blocchi vuoti

```
std::vector<bool> dims(dim, false);
for (const auto& block : blocks) {
  if (block.size() == 0) {
    reason = "F_Poly broken: found empty block";
    maybe_dump();
    return false;
  }
  for (const auto d : block) {
    if (d < 0 || d >= dim) {
      reason = "F_Poly broken: block contains an illegal
                space dim";
      maybe_dump();
      return false;
    }
    if (dims[d]) {
      reason = "F_Poly broken: repeated space dim in blocks";
      maybe_dump();
      return false;
```

```
        }
        dims[d] = true;
      }
    }
```

- Ogni dimensione dello spazio deve essere presente in un blocco

```
if (std::find(dims.begin(), dims.end(), false) != dims.end()) {
  reason = "F_Poly broken: a space dim is missing from blocks";
  maybe_dump();
  return false;
}
```

3.2 Operazione di base

Durante la lettura di questo paragrafo, sarà palese la diversa notazione utilizzata rispetto a [1]: questo perché, per motivi di compatibilità e trasparenza si è deciso di implementare le varie operazioni utilizzando i nomi con il loro corrispettivo della *PPLite*.

Inizialmente il lavoro si è basato sul attuare un porting dai poliedri fattorizzati implementati nella *PPL*, successivamente il focus è stato nell'adattarli al meglio alle strutture utilizzate nella PPLite, permettendo una semplificazione nella codifica e nel testing. Il principio base sotto all'implementazione di alcune operazioni è stato quello di preparare strutturalmente i poliedri fattorizzati in maniera tale da rendere possibile richiamare le varie operazioni.

Diverse sono le funzioni più significative definite all'interno di `F_Poly.hh`:

```
static Factors refactor(const Factors& f,
                        const Blocks& b1, const Blocks& b2);
static Blocks least_upper_bound(const Blocks& b1, const Blocks& b2);
static Blocks merge(const Blocks& b, const Block& added);
void join(const Factors& f, const Blocks& b);
void join(const F_Poly& f);
void affine_image(Var var, const Linear_Expr& expr,
                  const Integer& inhomo = Integer::zero(),
                  const Integer& den = Integer::one());
```

```
void affine_preimage(Var var, const Linear_Expr& expr,
                     const Integer& inhomo = Integer::zero(),
                     const Integer& den = Integer::one());
void add_con(const Con& c);
void add_gen(const Gen& c);
bool inclusion(const Factors& f, const Blocks& b) const;
```

3.2.1 refactor

La semantica rimane invariata invariata rispetto a quanto descritto nell'Algoritmo 3. Per la *join* ($\bowtie$) tra i fattori è stata utilizzata la `concatenate_assign()` della PPlite, una funzione omonima, per quanto riguarda il join dei blocchi, è stata implementata utilizzando `std::insert`[2]

[2] `template <class InputIt> iterator insert(const_iterator pos, InputIt first, InputIt last);`

4 Conclusione

Bibliografia

[1] Martin Vechev Gagandeep Singh, Markus Püschel. Fast polyhedra abstract domain. Technical report, 2017.