isl Basics

Ying Cao

June 20, 2020

Contents

Refer to this tutorial [1] for the first-hand knowledge.
Concepts are highlighted in violet red, and its corresponding isl implementation is highlighted in oliver green.

1 Sets of Named Integer Tuples

1.1 Named Integer Tuple

A ***Named Integer Tuple*** consists of an identifier(name) and a sequence of integer values. The identifier may be omitted and the sequence of integers may have a zero length.
Example (in *isl*'s notation):

$$A[2,8,1]$$

- Above example is a named integer tuple with identifier A and sequence of integers 2, 8 and 1.

$$[]$$

- Above example is a "named" integer tuple without identifier. Such an integer tuple is a zero-length sequence of integers.

1.2 Sets

1.2.1 Concept and Notation

A ***Set*** of named integer tuples contains zero or more named integer tuples as elements.
in isl, sets are represented by *isl_basic_set*,*isl_set*,*isl_union_set*.

1. *isl_basic_set* represents sets that can be described as a conjunction of affine constraints.
2. *isl_set* and represents unions of *isl_basic_sets*.
   all *isl_basic_set*s in the union need to live in the same space.
3. *isl_union_set*s represents unions of *isl_set*s in different spaces.
   spaces are considered different if they have a different number of dimensions and/or different names.

Example (in *isl*'s notation):

$$\{[];A[2,8,1]\}$$

- In isl, such sets are represented by an *isl_union_set*.
- No order is defined on the elements in a set.
- Elements in a set do not carry any multiplicity.

1.2.2 Basic Operations for Sets

1. intersection of sets $A \cap B$: *isl_union_set_intersect*.
2. union of sets $A \cup B$: *isl_union_set_union*.
3. set difference of two sets $A \setminus B$: *isl_union_set_subtract*.
4. compare equality $A = B$ *isl_union_set_is_equal*.
5. emptiness of a set *isl_union_set_is_empty*.
6. ...

1.3 Binary Relations

1.3.1 Concept and Notation

A ***Binary Relation*** is a set that contains pairs of named integer tuples.
in isl, relations are represented by *isl_basic_map*,*isl_map*,*isl_union_map*.

1. *isl_basic_map* represents relations that can be described as a conjunction of affine constraints.
2. *isl_map* and represents unions of *isl_basic_map*s.
   all *isl_basic_map*s in the union need to live in the same space.
3. *isl_union_map*s represents unions of *isl_map*s in different spaces.
   spaces are considered different if they have a different number of dimensions and/or different names.

Example (in *isl*'s notation):

$$\{A[2,8,1] \to B[5]; A[2,8,1] \to B[6]; B[5] \to B[5]\}$$

- In isl, the two named integer tuples in each pair in a binary relation are separated by a "->".
- A binary relation can be considered as ***representing the edges of a graph***. This graph may have loops, but no parallel edges.

1.3.2 Basic Operations for Binary Relations

1. Since a binary relation is essentially a set of pairs of tuples, the operations that apply to sets also apply to binary relations:
   - intersection: $A \cap B$, *isl_union_map_intersect*.
   - union: $A \cup B$, *isl_union_map_union*.
   - difference: $A \setminus B$, *isl_union_map_subtract*.
   - equality: $A = B$, *isl_union_map_is_equal*.
   - emptiness: *isl_union_map_is_empty*.
   - subrelation: $A \subseteq B$, *isl_union_map_is_subset*.
   - strict subrelation: $A \subsetneq B$, *isl_union_map_is_strict_subset*.
   - superrelation: $A \supseteq B$, call *isl_union_map_is_subset* with arguments reversed.
   - strict superrelation: $A \supsetneq B$, call *isl_union_map_is_strict_subset* with arguments reversed.
   - ...
2. The same comparison operators that can be applied to sets can also be applied to binary relations.
3. Binary relations additionally admit an inverse and a composition operation.
   - ***Inverse of a Binary Relation***: R^{-1}. *isl_union_map_reverse*:

     $$R^{-1} = \{\mathbf{j} \to \mathbf{i} : \mathbf{i} \to \mathbf{j} \in R\}$$

   - ***Composition of Binary Relations***: $B \circ A$:

     $$B \circ A = \{\mathbf{i} \to \mathbf{j} : \exists k : \mathbf{i} \to \mathbf{k} \in A \wedge \mathbf{k} \to \mathbf{j} \in B\}$$

   - ***Fixed Power of a Binary Relation***: R^n, *isl_union_map_fixed_power_val*
   - ...

1.3.3 Conversions

There are some ways to create binary relations from sets or the other way around.

1. ***Domain of a Binary Relation***, denoted as $domR$:

$$domR = \{\mathbf{i} : \exists \mathbf{j} : \mathbf{i} \rightarrow \mathbf{j} \in R\}$$

   In isl, this operation is called *isl_union_map_domain*.
   Example (in *isl*'s notation):
   Operation: The domain of $\{A[2,8,1] \rightarrow B[5]; A[2,8,1] \rightarrow B[6]; B[5] \rightarrow B[5]\}$ is: $\{A[2,8,1]; B[5]\}$.
2. ***Range of a Binary Relation***, denoted as $ranR$:

$$domR = \{\mathbf{j} : \exists \mathbf{i} : \mathbf{i} \rightarrow \mathbf{j} \in R\}$$

   In isl, this operation is called *isl_union_map_range*.
   Example (in *isl*'s notation):
   The domain of $\{A[2,8,1] \rightarrow B[5]; A[2,8,1] \rightarrow B[6]; B[5] \rightarrow B[5]\}$ is: $\{B[5]; B[6]\}$.
3. ***Universal Binary Relation between Sets***:

$$\mathbf{A} \rightarrow \mathbf{B} = \{\mathbf{i} \rightarrow \mathbf{j} : \mathbf{i} \in A \wedge \mathbf{j} \in B\}$$

   In isl, this operation is called *isl_union_map_from_domain_and_range*.
4. ***Identity Relation on a Set***, denoted as $\mathbf{1}_S$:

$$\mathbf{1}_S = \{\mathbf{i} \rightarrow \mathbf{i} : \mathbf{i} \in S\}$$

1.3.4 Mixed Operations

There are some mixed operations that combine a binary relation and a set.

1. ***Domain restriction*** $R \cap_{dom} S$ of a binary relation R with respect to a set S is:

$$R \cap_{dom} S = R \cap (S \rightarrow (ranR))$$

   in isl, this operation is *isl_union_map_intersect_domain*.
2. ***Range restriction*** $R \cap_{ran} S$ of a binary relation R with respect to a set S is:

$$R \cap_{ran} S = R \cap ((domR) \rightarrow S$$

   in isl, this operation is *isl_union_map_intersect_range*.
3. ***Domain subtraction*** $R \setminus_{dom} S$ of a binary relation R with respect to a set S is:

$$R \setminus_{dom} S = R \setminus (S \rightarrow (ranR))$$

   .
   in isl, this operation is *isl_union_map_subtract_domain*.
4. ***Range subtraction*** $R \setminus_{ran} S$ of a binary relation R with respect to a set S is:

$$R \setminus_{ran} S = R \setminus (dom(R) \rightarrow (S))$$

   in isl, this operation is *isl_union_map_subtract_range*.
5. ***Application*** of a binary relation R to a set S is a set (*isl_union_set*) that:

$$R(S) = ran(R \cap_{dom} S) = \{\mathbf{j} : \exists \mathbf{i} \in S : \mathbf{i} \rightarrow \mathbf{j} \in R\}$$

   in isl, this operation is *isl_union_set_apply*.

6. ...

1.4 Wrapped Relations

While sets keep track of named integer tuples and binary relations keep track of pairs of such tuples, it can sometimes be convenient to keep track of relations between more than two such tuples. isl currently does not support ternary or general n-ary relations. However isl allows a pair of tuples to be combined into a single tuple, which can then again appear as the first or second tuple in a pair of tuples. This process is called ***"wrapping"***. The result of wrapping a pair of named integer tuples is called a structured named integer tuple.

1.4.1 Structured Named Integer Tuples

A structured named integer tuple is either:

- *a named integer tuple, or*
- *a named pair of structured named integer tuples.*

wrap and unwrap

- The wrap WR of a binary relation R is a set: wrap a pairs of tuple in R into anonymous tuples.
- The unwrap $W^{-1}S$ of a set S is a binary relation.

1.4.2 Products

A product of two sets (or binary relations) is a set (or binary relation) that **combines** the tuples in its arguments **into wrapped relations**.

1. ***Set Product*** of two sets A and B is a set (*isl_union_set*):

$$A \times B = W(A \to B) = \{[\mathbf{i} \to \mathbf{j}] : \mathbf{i} \in A \wedge \mathbf{j} \in B\}$$

   in isl, this is *isl_union_set_product*.
2. ***Binary Relation Product*** of two binary relations A and B is a binary relation (*isl_union_map*):

$$A \times B = \{[\mathbf{i} \to \mathbf{m}] \to [\mathbf{j} \to \mathbf{n}] : \mathbf{i} \to \mathbf{j} \in A \wedge \mathbf{m} \to \mathbf{n} \in B\}$$

   in isl, this is *isl_union_map_product*.
3. ***Zip of a Binary Relation*** R is a binary relation (*isl_union_map*) that:

$$zipR = \{[\mathbf{i} \to \mathbf{m}] \to [\mathbf{j} \to \mathbf{n}] : [\mathbf{i} \to \mathbf{j}] \to [\mathbf{m} \to \mathbf{n}] \in R\}$$

   in isl, this is called *isl_union_map_zip*.
4. ***Domain Product*** $A \rtimes B$ of two binary relations A and B is a binary relation (*isl_union_map*) that:

$$A \rtimes B = \{[\mathbf{i} \to \mathbf{j}] \to \mathbf{k} : \mathbf{i} \to \mathbf{k} \in A \wedge [\mathbf{j} \to \mathbf{k}] \in B\}$$

   in isl, this is called *isl_union_map_domain_product*.
5. ***Range Product*** $A \rtimes B$ of two binary relations A and B is a binary relation (*isl_union_map*) that:

$$A \rtimes B = \{\mathbf{i} \to [\mathbf{j} \to \mathbf{k}] : \mathbf{i} \to \mathbf{j} \in A \wedge \mathbf{i} \to \mathbf{k}] \in B\}$$

   in isl, this is called *isl_union_map_range_product*.
6. ***Domain Factor of Product*** of a binary relation R is a binary relation (*isl_union_map*) that:

$$\{\mathbf{i} \to \mathbf{m} : \exists \mathbf{j}, \mathbf{n} : [\mathbf{i} \to \mathbf{j}] \to [\mathbf{m} \to \mathbf{n}] \in R\}$$

in isl, this is called *isl_union_map_factor_domain*.

7. ***Range Factor of Product*** of a binary relation R is a binary relation (union map) that:

$$\left\{\mathbf{j} \to \mathbf{n} : \exists \mathbf{j}, \mathbf{m} : [\mathbf{i} \to \mathbf{j}] \to [\mathbf{m} \to \mathbf{n}] \in R\right\}$$

   in isl, this is called *isl_union_map_factor_range*.

8. ***Domain Factor of Domain Product*** of a binary relation R is a binary relation (*isl_union_map*) that:

$$\{\mathbf{i} \to \mathbf{k} : \exists \mathbf{j} : [\mathbf{i} \to \mathbf{j}] \to \mathbf{k} \in R\}$$

   in isl, this is called *isl_union_map_domain_factor_domain*.

9. ***Range Factor of Domain Product*** of a binary relation R is a binary relation (*isl_union_map*) that:

$$\{\mathbf{j} \to \mathbf{k} : \exists \mathbf{j} : [\mathbf{i} \to \mathbf{j}] \to \mathbf{k} \in R\}$$

   in isl, this is called *isl_union_map_domain_factor_range*.

10. ***Domain Factor of Range Product*** of a binary relation R is a binary relation (*isl_union_map*) that:

$$\{\mathbf{i} \to \mathbf{j} : \exists \mathbf{k} : \mathbf{i} \to [\mathbf{j} \to \mathbf{k}] \in R\}$$

    in isl, this is called *isl_union_map_range_factor_domain*

11. ***Range Factor of Range Product*** of a binary relation R is a binary relation (*isl_union_map*) that:

$$\{\mathbf{i} \to \mathbf{k} : \exists \mathbf{j} : \mathbf{i} \to [\mathbf{j} \to \mathbf{k}] \in R\}$$

    in isl, this is called *isl_union_map_range_factor_range*

1.4.3 Domain and Range Projection

These operations take a binary relation as input and produce a binary relation that projects a wrapped copy of the input onto its domain or range.

1. ***Domain Projection*** of a binary relation R is a binary relation (*isl_union_map*) that:

$$\xrightarrow{dom} R = \{[\mathbf{i} \to \mathbf{j}] \to \mathbf{i} : \mathbf{i} \to \mathbf{j} \in R\}$$

   in isl, this is called *isl_union_map_domain_map*.

2. ***Range Projection*** of a binary relation R is a binary relation (*isl_union_map*) that:

$$\xrightarrow{ran} R = \{[\mathbf{i} \to \mathbf{j}] \to \mathbf{j} : \mathbf{i} \to \mathbf{j} \in R\}$$

   in isl, this is called *isl_union_map_range_map*.

1.4.4 Difference Set Projection

The difference set of a binary relation contains the difference of the pairs of elements in the relation. The difference set projection maps the original pairs to their difference.

1. ***Difference Set of a Binary Relation*** is a set (*isl_union_set*) a set containing the differences between pairs of elements in R that ***have the same space***, where the difference between a pair of elements ***has the same space*** as those two elements and has values that are the difference between the values of the second and those of the first element:

$$\triangle R = \{\mathbf{d} : \exists \mathbf{x} \to \mathbf{y} \in R : S\mathbf{d} = S\mathbf{x} = S\mathbf{y} \wedge V\mathbf{d} = V\mathbf{y} - V\mathbf{x}\}\}$$

in isl, this is called *isl_union_map_deltas*.

2. ***Difference Set Project of a Binary Relation*** is a binary relation (*isl_union_map*) that:

$$\underrightarrow{\triangle} R = \{[\mathbf{x} \to \mathbf{y}] \to \mathbf{d} : \mathbf{x} \to \mathbf{y} \in R \wedge S\mathbf{d} = S\mathbf{x} = S\mathbf{y} \wedge V\mathbf{d} = V\mathbf{y} - V\mathbf{x}\}$$

in isl, this is called *isl_union_map_deltas_map*.

2 Presburger Sets and Relations

Presburger formula provides a way to describe set and binary relation through properties that need to be satisfied instead of listing the elements contained in the set of binary relation.

The elements of a Presburger set are descriped in terms of ***Structured Named Integer Tuple Templates*** which are essentially the same as structured named integer tuples. except that **the integers have been replaced by variables**.

2.1 Presburger Sets and Relations

Refer to pp.43 to 45 (section 3.2) of [1] for the concept of Presburger formula.

Example 1 (not in *isl*'s notation):

$$\{B[i] : 5 \le i \le 6; C[] :\}$$

- the set above is equal to $\{B[5]; B[6]; C[]\}$ in the notaion of sets of named integer tuples.

Example 2 (not in *isl*'s notation):

$$\{[i] : 0 \ge i \wedge i \le 10 \wedge \exists \alpha : i = \alpha + \alpha]\}$$

- the set above is equal to $\{[0]; [2]; [4]; [6]; [8]; [10];\}$ in the notaion of sets of named integer tuples.

Example 3 (not in *isl*'s notation):

$$\{S[i] : 0 \le i \wedge i \le n\}$$

- The isl notation for above set above is: $[n] \to \{S[i] : 0 \le i \quad \text{and} \quad i \le n\}$
- In isl, ***a constant symbol is called a parameter***.
  - A parameter has to be declared in front of the set or binary relation description.
  - ***All parameters need to be placed in a comma separated list enclosed in brackets and followed by a "-> in front of the set or binary relation description***.
  - The order of the parameters inside the list is immaterial.

Example 4 (in *isl*'s notation):

$$\{: n \ge 0\}$$

- The set above is called a ***Unit Set***.
- In isl, unit sets are called *parameter sets* and they are represented by an *isl_set*.

2.2 Operations for Presburger Sets and Relations

Most of the operations defined in section[1] are not affected by the presence of constant symbols. The operation is simply applied uniformly for all possible values of those constant symbols. Some operations, in particular the comparison operations, are affected, however. See pp. 48 53 of [1] for details.

2.2.1 Lexicographic Order in Presburger Relations

A ***Lexicographic Order*** expressed in Presburger Formula is:
Given two vector **a** and **b** of equal length, **a** is said to be Lexicographically smaller than **b** if:

$$\mathbf{a} \prec \mathbf{b} = \bigvee_{i:1\le i\le n} \left(\left(\bigwedge_{j:1\le j\le i} a_j = b_j \wedge a_i < b_i \right) \right)$$

notations:

1. $\prec /2$: *lexicographically smaller-than*
2. $\preccurlyeq /2$: *lexicographically smaller-than-or-equal*
3. $\succ /2$: *lexicographically greater-than*
4. $\succcurlyeq /2$: *lexicographically greater-than-or-equal*

1. ***Lexicographically-smaller-than Relation on Sets***. The lexicographically-smaller-than relation $A \prec B$ on two sets A and B is a binary relation (*isl_union_map*) that contains pairs of elements, one from A and one from B such that the two elements ***have the same space*** and the first is lexicographically smaller than the second. That is:

   $$A \prec B = \{\mathbf{a} \to \mathbf{b} : \mathbf{a} \in A \wedge \mathbf{b} \in B \wedge Sa = Sb \wedge V\mathbf{a} \prec V\mathbf{b}\}$$

   in isl, this operation is called *isl_union_set_lex_lt_union_set*.
2. ***Lexicographically-smaller-than-or-equal Relation on Sets***:

   $$A \prec B = \{\mathbf{a} \to \mathbf{b} : \mathbf{a} \in A \wedge \mathbf{b} \in B \wedge Sa = Sb \wedge V\mathbf{a} \preccurlyeq V\mathbf{b}\}$$

   in isl, this operation is called *isl_union_set_lex_le_union_set*.
3. ***Lexicographically-greater-than Relation on Sets***:

   $$A \prec B = \{\mathbf{a} \to \mathbf{b} : \mathbf{a} \in A \wedge \mathbf{b} \in B \wedge Sa = Sb \wedge V\mathbf{a} \succ V\mathbf{b}\}$$

   in isl, this operation is called *isl_union_set_lex_gt_union_set*.
4. ***Lexicographically-greater-than-or-equal Relation on Sets***:

   $$A \prec B = \{\mathbf{a} \to \mathbf{b} : \mathbf{a} \in A \wedge \mathbf{b} \in B \wedge Sa = Sb \wedge V\mathbf{a} \succcurlyeq V\mathbf{b}\}$$

   in isl, this operation is called *isl_union_set_lex_ge_union_set*.

The same operations are also available on binary relations, but in this case the comparison is ***performed on the range elements*** of the input relations and the ***result collects the corresponding domain elements***.

1. ***Lexicographically-smaller-than Relation on Binary Relations***. The lexicographically-smaller-than relation $A \prec B$ on two binary relations A and B is a binary relation (*isl_union_map*) that contains pairs of elements, one from ***the domain of*** A and one from ***the domain of*** B that have corresponding ***range elements*** such that the first is lexicographically smaller than the second. That is:

$$A \prec B = \{\mathbf{a} \to \mathbf{b} : \exists \mathbf{c}, \mathbf{d} : \mathbf{a} \to \mathbf{c} \in A \wedge \mathbf{b} \to \mathbf{d} \in B \wedge S\mathbf{c} = S\mathbf{d} \wedge V\mathbf{c} \prec V\mathbf{d}\}$$

in isl, this operation is called *isl_union_map_lex_le_union_map*.

2. ***Lexicographically-smaller-than-or-equal Relation on Binary Relations*** is a binary relation (*isl_union_map*) that:

$$A \prec B = \{\mathbf{a} \to \mathbf{b} : \exists \mathbf{c}, \mathbf{d} : \mathbf{a} \to \mathbf{c} \in A \wedge \mathbf{b} \to \mathbf{d} \in B \wedge S\mathbf{c} = S\mathbf{d} \wedge V\mathbf{c} \preccurlyeq V\mathbf{d}\}$$

in isl, this operation is called *isl_union_map_lex_le_union_map*.

3. ***Lexicographically-greater-than Relation on Binary Relations*** is a binary relation (*isl_union_map*) that:

$$A \prec B = \{\mathbf{a} \to \mathbf{b} : \exists \mathbf{c}, \mathbf{d} : \mathbf{a} \to \mathbf{c} \in A \wedge \mathbf{b} \to \mathbf{d} \in B \wedge S\mathbf{c} = S\mathbf{d} \wedge V\mathbf{c} \succ V\mathbf{d}\}$$

in isl, this operation is called *isl_union_map_lex_gt_union_map*.

4. ***Lexicographically-greater-than-or-equal Relation on Binary Relations*** is a binary relation (*isl_union_map*):

$$A \prec B = \{\mathbf{a} \to \mathbf{b} : \exists \mathbf{c}, \mathbf{d} : \mathbf{a} \to \mathbf{c} \in A \wedge \mathbf{b} \to \mathbf{d} \in B \wedge S\mathbf{c} = S\mathbf{d} \wedge V\mathbf{c} \succcurlyeq V\mathbf{d}\}$$

in isl, this operation is called *isl_union_map_lex_ge_union_map*.

2.2.2 Space-Local Operations

1. ***Space Decomposition of a Set*** denoted as DS is:

$$S_i := \{\mathbf{x} : \mathbf{x} \in S \wedge S\mathbf{x} = U_i\}, \text{ then: } DS = \{S_i\}_i.$$

in isl, this operation is called *isl_union_set_foreach_set*.

2. ***Space Decomposition of a Binary Relation*** denoted as DR is:

$$R_i := \{\mathbf{x} \to \mathbf{y} : \mathbf{x} \to \mathbf{y} \in S \wedge S\mathbf{x} = U_i \wedge S\mathbf{y} = V_i\}, \text{ then: } DR = \{R_i\}_i.$$

in isl, this operation is called *isl_union_set_foreach_map*.

Lexicographic optimizatioin can be defined in terms of the space decomposition.

1. ***Lexicographic Maximum of a Set*** is a subset of S that contains the lexicographically maximal element of each of the spaces with elements in S. ***If there is any such space with no lexicographically maximal element, then the operation is undefined***. That is, let $DS =: \{S_i\}_i$, Define:

$$M_i := \{\mathbf{x} : \mathbf{x} \in S_i \wedge \forall \mathbf{y} \in S_i : V\mathbf{x} \succcurlyeq V\mathbf{y}\},$$

Then:

$$\text{lexmax} S = \bigcup_i M_i$$

in isl, this operation is called *isl_union_set_lexmax*.

2. ***Lexicographic Minimum of a Set*** is a subset of S that contains the lexicographically minimal element of each of the spaces with elements in S. ***If there is any such space with no lexicographically minimal element, then the operation is undefined***. That is, let $DS =: \{S_i\}_i$, Define:

$$M_i := \{\mathbf{x} : \mathbf{x} \in S_i \wedge \forall \mathbf{y} \in S_i : V\mathbf{x} \preccurlyeq V\mathbf{y}\},$$

Then:

$$\text{lexmin} S = \bigcup_i M_i$$

in isl, this operation is called *isl_union_set_lexmin*.

3. ***Lexicographic Maximum of a Binary Relation*** is a subset of R that for each first element in the pairs of elements in R and for each of the spaces of the corresponding second elements, the lexicographically maximal of those corresponding elements. ***If there is any such first element and space with no corresponding lexicographically maximal second element, then the operation is undefined.*** That is, let $DR =: \{R_i\}_i$, Define:

$$M_i := \{\mathbf{x} \to \mathbf{y} : \mathbf{x} \to \mathbf{y} \in R_i \wedge \forall \mathbf{x}' \to \mathbf{z} \in R_i : \mathbf{x} = \mathbf{x}' \Rightarrow V\mathbf{y} \succcurlyeq V\mathbf{z}\},$$

Then:

$$\text{lexmax} R = \bigcup_i M_i.$$

$a \Rightarrow b$ (implication) is equivalent to $\neg a \vee b$.

in isl, this operation is called *isl_union_map_lexmax*.

4. ***Lexicographic Minimum of a Binary Relation*** is a subset of R that for each first element in the pairs of elements in R and for each of the spaces of the corresponding second elements, the lexicographically mimimal of those corresponding elements. ***If there is any such first element and space with no corresponding lexicographically minimal second element, then the operation is undefined.*** That is, let $DR =: \{R_i\}_i$, Define:

$$M_i := \{\mathbf{x} \to \mathbf{y} : \mathbf{x} \to \mathbf{y} \in R_i \wedge \forall \mathbf{x}' \to \mathbf{z} \in R_i : \mathbf{x} = \mathbf{x}' \Rightarrow V\mathbf{y} \preccurlyeq V\mathbf{z}\},$$

Then:

$$\text{lexmin} R = \bigcup_i M_i.$$

in isl, this operation is called *isl_union_map_lexmin*.

2.2.3 Simplification

In isl, sets and binary relations are represented internally in ***disjunctive normal form***: all disjunctions are moved to the outermost positions in the formula, while all conjunctions are moved innermost:

$$\bigvee_i \left(\exists \alpha_i : \left(\bigwedge_j t_{i,j}(\mathbf{x}_i, \alpha_x) = 0 \wedge \bigwedge_k u_{i,k}(\mathbf{x}, \alpha_\mathbf{i}) \geq 0 \right) \right)$$

Coalescing takes a formula in disjunctive normal form and rewrites it using fewer or the same number of disjuncts. In isl, this operation is called *isl_union_set_coalesce* for sets and *isl_union_map_coalesce* for binary relations.

3 Piecewise Quasi-Affine Expressions

Concept	Notes	*isl*'s representation for quasi-affine
Base Expression	maps integer tuples to a rational value	*isl_aff*
Tuple of Expressions	combines $n \geq 0$ base expressions of the same type and with the same domain space into ***a multi-dimensional expression*** that shares this domain space.	*isl_multi_aff*
Piecewise Expression	combines $n \geq 0$ pairs of **fixed-space** sets S_i and base quasi-affine expressions E_i into **a single quasi-affine expression**.	*isl_pw_aff*
Piecewise Tuple of Expression	apply the definition of piecewise expression to tuples of expressions. The results is a piecewise expression.	*isl_pw_multi_aff*
Tuple of Piecewise Expressions	is the results of applying the definition of "tuple of expressions" to piecewise expressions. The result is a tuple.	*isl_multi_pw_aff*
Multi-Space Expression	combines piecewise expressions with different domain and/or range spaces, but with pair-wise disjoint domains into **a single expression**.	*isl_union_pw_aff*

3.1 Concepts and Definitions

3.1.1 Quasi-Affine Expression

A ***Quasi-Affine Expression*** f is a function that maps a named integer tuple with a given space S to a rational value, where the function is specified as a Presburger term in the variables of the tuple, optionally divided by an integer constant (integer division is not affine, so the name "quasi-" comes).

In isl, a quasi-affine expression is represented by an *isl_aff*.

- ***The domain space of a quasi-affine expression*** is the space of the input integer tuple and is written $S^{\text{dom}} f$.
- ***The range space of a quasi-affine expression*** is fixed to the anonymous single-dimensional space.

The quasi-affine expression may also be a (symbolic) constant expression, in which case there is no domain space, written $S^{\text{dom}} f = \bot$, and the domain is a unit set.

Example (in *isl*'s notation):

$$\{[x, y] \to [x + 2 + y - 6]\}$$

3.1.2 Tuple of Quasi-Affine Expressions

A ***Tuple of Quasi-Affine Expressions*** combines zero or more base affine expressions of ***the same type and with the same domain space*** (or no domain space) into a multi-dimensional expression that ***shares this domain space*** and that ***has a prescribed range space***.

In particular, it is either:

- an identifier n along with $d \geq 0$ base expressions e_j for $0 \leq j < d$, written $n[e_0, e_1, ..., e_{d-1}]$, or
- an identifier n along with two tuples of expressions **e** and **f** written $n[\mathbf{e} \to \mathbf{f}]$.

Info:

1. The ***domain of a Tuple of Quasi-Affine Expressions*** is the intersection of the domains of the quasi-affine expressions.
2. The domain of a tuple of zero expressions is undefined.

In *isl*:

1. a tuple of quasi-affine expressions is represented by an *isl_multi_aff*.
2. the space of a tuple of quasi-affine expressions is called the *range space* (?? confusing to me) of its *isl_multi_aff* representation.

3.1.3 Piecewise Quasi-Affine Expression

A ***Piecewise Quasi-Affine Expression*** combines $n \geq 0$ pairs of **fixed-space** sets S_i (domain spaces) and base quasi-affine expressions E_i into **a single quasi-affine expression**.

- The spaces of the S_i and the domain and range spaces of the E_i **all need to be the same**.
- **The S_i need to be pairwise disjoint**.

Info:

1. ***The domain of the piecewise quasi-affine expression*** is the union of the S_i.
2. ***The value of the piecewise quasi-affine expression*** at an integer tuple $\mathbf{x}$ is: $E_i(\mathbf{x})$ if $x \in S_i$ for some i. Otherwise, the value is undefined.

In isl, a piecewise quasi-affine expression is represented by an *isl_pw_aff*.

3.1.4 Piecewise Tuple of Quasi-Affine Expressions

A ***Piecewise Tuple of Quasi-Affine Expressions*** is to apply the concept and definition of piecewise expression to tuples of quasi-affine expressions.
In isl, a piecewise quasi-affine expression is represented by an *isl_pw_multi_aff*.

3.1.5 Multi-Space Piecewise Quasi-Affine Expression

Info: Notes about multi-space expression

1. A multi-space expression does not have a specific domain or range space.
2. The domain of a multi-space expression is the union of the domains of the combined piecewise expressions.
3. The value of a multi-space expression at an integer tuple $\mathbf{x}$ is the value of the piecewise expression at $\mathbf{x}$ that contains $\mathbf{x}$ in its domain, if any.

A ***Multi-Space Piecewise Quasi-Affine Expression*** is the result of applying the concept and definition of *multi-space expressionn* to piecewise quasi-affine expressions.
In isl, a piecewise quasi-affine expression is represented by an *isl_union_pw_aff*.

3.1.6 Multi-Space Piecewise Tuple of Quasi-Affine Expressioins

A ***Multi-Space Piecewise Tuple of Quasi-Affine Expressioins*** is the result of applying the concept and definition of *multi-space expression* to piecewise tuples of quasi-affine expressions.

Table 1: piecewise tuples of quasi-affine **vs**. tuples of piecewise quasi-affine

piecewise tuples of quasi-affine	**tuples of piecewise quasi-affine**
a piecewise expressioin the entire tuple is either defined or undefined at any particular point in the domain space.	a tuple each element of the tuple is a piecewise expression that may be undefined in different parts of the domain.

In isl, a piecewise quasi-affine expression is represented by an *isl_union_pw_multi_aff*.

3.1.7 Tuple of Piecewise Quasi-Affine Expressions

A ***Tuple of Piecewise Quasi-Affine Expressions*** is the result of applying the concept and definition of *tuple of expressions* to piecewise quasi-affine expressions.
In isl, a piecewise quasi-affine expression is represented by an *isl_multi_pw_aff*.

Info:

1. piecewise tuples of quasi-affine is a piecewise expression.
2. tuples of piecewise quasi-affine is a tuple.

Example 1
a tuple of piecewise quasi-affine expressions:

$$\{[i] \to [(i : i \geq 0), (i-1 : i \geq 1)]\}$$

In particular, the first piecewise quasi-affine expression has domain $\{[i] : i \geq 0\}$ while the second has domain $\{[i] : i \geq 1\}$.
Example 2 (in *isl*'s notation):
a piecewise tuple of quasi-affine expresions:

$$\{[i] \to [(i), (-1+i)] : i > 0\}$$

a tuple of piecewise quasi-affine expression:

$$\{[i] \to [((i) : i > 0), ((-1+i) : i > 0)]\}$$

3.1.8 Tuple of Multi-Space Piecewise Quasi-Affine Expressions

A ***Tuple of Multi-Space Piecewise Quasi-Affine Expressions*** is the result of applying the concept and definition of tuple of expressions to multi=space piecewise quasi-affine expressions. A tuple of multi-space piecewise quasi-affine expressions does not have a domain space.
In isl, a tuple of multi-space piecewise quasi-affine expression is represented by an *isl_multi_union_pw_aff*.
Example (in *isl*'s notation):

$$[n] \to A[\{S2[i,j] \to [(i)]; S1[] \to [(n)]\}, \{S2[i,j] \to [(j)]; S1[] \to [0]\}]$$

3.2 Operations

1. ***Sum***: The sum $f+g$ of two quasi-affine expressions f and g with the same domain space is a function with the same domain space and as value the sum of the values of f and g.

2. ***Union***: The union of two expressions with disjoint domains combines them into a single expression defined over the union of the domains.

References

[1] S. Verdoolaege, "Presburger formulas and polyhedral compilation," 2016.