Integer Set Library: Manual

Version: isl-0.17.1

Sven Verdoolaege

May 6, 2016

Contents

Chapter 1

User Manual

1.1 Introduction

isl is a thread-safe C library for manipulating sets and relations of integer points bounded by affine constraints. The descriptions of the sets and relations may involve both parameters and existentially quantified variables. All computations are performed in exact integer arithmetic using GMP or imath. The isl library offers functionality that is similar to that offered by the Omega and Omega+ libraries, but the underlying algorithms are in most cases completely different.

The library is by no means complete and some fairly basic functionality is still missing. Still, even in its current form, the library has been successfully used as a backend polyhedral library for the polyhedral scanner CLooG and as part of an equivalence checker of static affine programs. For bug reports, feature requests and questions, visit the discussion group at http://groups.google.com/group/isl-development.

1.1.1 Backward Incompatible Changes

Changes since isl-0.02

- The old printing functions have been deprecated and replaced by isl_printer functions, see Input and Output.
- Most functions related to dependence analysis have acquired an extra must argument. To obtain the old behavior, this argument should be given the value 1. See Dependence Analysis.

Changes since isl-0.03

- The function isl_pw_qpolynomial_fold_add has been renamed to isl_pw_qpolynomial_fold_fold. Similarly, isl_union_pw_qpolynomial_fold_add has been renamed to isl_union_pw_qpolynomial_fold_

Changes since isl-0.04

- All header files have been renamed from `isl_header.h` to `isl/header.h`.

Changes since isl-0.05

- The functions `isl_printer_print_basic_set` and `isl_printer_print_basic_map` no longer print a newline.
- The functions `isl_flow_get_no_source` and `isl_union_map_compute_flow` now return the accesses for which no source could be found instead of the iterations where those accesses occur.
- The functions `isl_basic_map_identity` and `isl_map_identity` now take a **map** space as input. An old call `isl_map_identity(space)` can be rewritten to `isl_map_identity(isl_space_map_from_set(space))`.
- The function `isl_map_power` no longer takes a parameter position as input. Instead, the exponent is now expressed as the domain of the resulting relation.

Changes since isl-0.06

- The format of `isl_printer_print_qpolynomial`'s `ISL_FORMAT_ISL` output has changed. Use `ISL_FORMAT_C` to obtain the old output.
- The `*_fast_*` functions have been renamed to `*_plain_*`. Some of the old names have been kept for backward compatibility, but they will be removed in the future.

Changes since isl-0.07

- The function `isl_pw_aff_max` has been renamed to `isl_pw_aff_union_max`. Similarly, the function `isl_pw_aff_add` has been renamed to `isl_pw_aff_union_add`.
- The `isl_dim` type has been renamed to `isl_space` along with the associated functions. Some of the old names have been kept for backward compatibility, but they will be removed in the future.
- Spaces of maps, sets and parameter domains are now treated differently. The distinction between map spaces and set spaces has always been made on a conceptual level, but proper use of such spaces was never checked. Furthermore, up until isl-0.07 there was no way of explicitly creating a parameter space. These can now be created directly using `isl_space_params_alloc` or from other spaces using `isl_space_params`.
- The space in which `isl_aff`, `isl_pw_aff`, `isl_qpolynomial`, `isl_pw_qpolynomial`, `isl_qpolynomial_fold` and `isl_pw_qpolynomial_fold` objects live is now a map space instead of a set space. This means, for example, that the dimensions of the domain of an `isl_aff` are now considered to be of type `isl_dim_in`

instead of isl_dim_set. Extra functions have been added to obtain the domain space. Some of the constructors still take a domain space and have therefore been renamed.

- The functions isl_equality_alloc and isl_inequality_alloc now take an isl_local_space instead of an isl_space. An isl_local_space can be created from an isl_space using isl_local_space_from_space.

- The isl_div type has been removed. Functions that used to return an isl_div now return an isl_aff. Note that the space of an isl_aff is that of relation. When replacing a call to isl_div_get_coefficient by a call to isl_aff_get_coefficient any isl_dim_set argument needs to be replaced by isl_dim_in. A call to isl_aff_from_div can be replaced by a call to isl_aff_floor. A call to isl_qpolynomial_div(div) call be replaced by the nested call

```
isl_qpolynomial_from_aff(isl_aff_floor(div))
```

  The function isl_constraint_div has also been renamed to isl_constraint_get_div.

- The nparam argument has been removed from isl_map_read_from_str and similar functions. When reading input in the original PolyLib format, the result will have no parameters. If parameters are expected, the caller may want to perform dimension manipulation on the result.

Changes since isl-0.09

- The schedule_split_parallel option has been replaced by the schedule_split_scaled option.

- The first argument of isl_pw_aff_cond is now an isl_pw_aff instead of an isl_set. A call isl_pw_aff_cond(a, b, c) can be replaced by

```
isl_pw_aff_cond(isl_set_indicator_function(a), b, c)
```

Changes since isl-0.10

- The functions isl_set_dim_has_lower_bound and isl_set_dim_has_upper_bound have been renamed to isl_set_dim_has_any_lower_bound and isl_set_dim_has_any_upper_bound. The new isl_set_dim_has_lower_bound and isl_set_dim_has_upper_bound have slightly different meanings.

Changes since isl-0.12

- isl_int has been replaced by isl_val. Some of the old functions are still available in isl/deprecated/*.h but they will be removed in the future.

- The functions isl_pw_qpolynomial_eval, isl_union_pw_qpolynomial_eval, isl_pw_qpolynomial_fold_eval and isl_union_pw_qpolynomial_fold_eval have been changed to return an isl_val instead of an isl_qpolynomial.

- The function isl_band_member_is_zero_distance has been removed. Essentially the same functionality is available through isl_band_member_is_coincident, except that it requires setting up coincidence constraints. The option schedule_outer_zero_distance has accordingly been replaced by the option schedule_outer_coincidence.
- The function isl_vertex_get_expr has been changed to return an isl_multi_aff instead of a rational isl_basic_set. The function isl_vertex_get_domain has been changed to return a regular basic set, rather than a rational basic set.

Changes since isl-0.14

- The function isl_union_pw_multi_aff_add now consistently computes the sum on the shared definition domain. The function isl_union_pw_multi_aff_union_add has been added to compute the sum on the union of definition domains. The original behavior of isl_union_pw_multi_aff_add was confused and is no longer available.
- Band forests have been replaced by schedule trees.
- The function isl_union_map_compute_flow has been replaced by the function isl_union_access_info_compute_flow. Note that the may dependence relation returned by isl_union_flow_get_may_dependence is the union of the two dependence relations returned by isl_union_map_compute_flow. Similarly for the no source relations. The function isl_union_map_compute_flow is still available for backward compatibility, but it will be removed in the future.
- The function isl_basic_set_drop_constraint has been deprecated.
- The function isl_ast_build_ast_from_schedule has been renamed to isl_ast_build_node_from_schedu The original name is still available for backward compatibility, but it will be removed in the future.
- The separation_class AST generation option has been deprecated.
- The functions isl_equality_alloc and isl_inequality_alloc have been renamed to isl_constraint_alloc_equality and isl_constraint_alloc_inequality. The original names have been kept for backward compatibility, but they will be removed in the future.
- The schedule_fuse option has been replaced by the schedule_serialize_sccs option. The effect of setting the schedule_fuse option to ISL_SCHEDULE_FUSE_MIN is now obtained by turning on the schedule_serialize_sccs option.

1.2 License

isl is released under the MIT license.

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software

without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

Note that by default `isl` requires `GMP`, which is released under the GNU Lesser General Public License (LGPL). This means that code linked against `isl` is also linked against LGPL code.

When configuring with `--with-int=imath` or `--with-int=imath-32`, `isl` will link against `imath`, a library for exact integer arithmetic released under the MIT license.

1.3 Installation

The source of `isl` can be obtained either as a tarball or from the git repository. Both are available from http://isl.gforge.inria.fr/. The installation process depends on how you obtained the source.

1.3.1 Installation from the git repository

1. Clone or update the repository

   The first time the source is obtained, you need to clone the repository.

   ```
   git clone git://repo.or.cz/isl.git
   ```

   To obtain updates, you need to pull in the latest changes

   ```
   git pull
   ```

2. Optionally get `imath` submodule

   To build `isl` with `imath`, you need to obtain the `imath` submodule by running in the git source tree of `isl`

   ```
   git submodule init
   git submodule update
   ```

This will fetch the required version of imath in a subdirectory of isl.

3. Generate configure

```
./autogen.sh
```

After performing the above steps, continue with the Common installation instructions.

1.3.2 Common installation instructions

1. Obtain GMP

   By default, building isl requires GMP, including its headers files. Your distribution may not provide these header files by default and you may need to install a package called gmp-devel or something similar. Alternatively, GMP can be built from source, available from http://gmplib.org/. GMP is not needed if you build isl with imath.

2. Configure

   isl uses the standard autoconf configure script. To run it, just type

   ```
   ./configure
   ```

   optionally followed by some configure options. A complete list of options can be obtained by running

   ```
   ./configure --help
   ```

   Below we discuss some of the more common options.

   --prefix
   : Installation prefix for isl

   --with-int=[gmp|imath|imath-32]
   : Select the integer library to be used by isl, the default is gmp. With imath-32, isl will use 32 bit integers, but fall back to imath for values out of the 32 bit range. In most applications, isl will run fastest with the imath-32 option, followed by gmp and imath, the slowest.

   --with-gmp-prefix
   : Installation prefix for GMP (architecture-independent files).

   --with-gmp-exec-prefix
   : Installation prefix for GMP (architecture-dependent files).

3. Compile

   ```
   make
   ```

4. Install (optional)

   ```
   make install
   ```

1.4 Integer Set Library

1.4.1 Memory Management

Since a high-level operation on isl objects usually involves several substeps and since the user is usually not interested in the intermediate results, most functions that return a new object will also release all the objects passed as arguments. If the user still wants to use one or more of these arguments after the function call, she should pass along a copy of the object rather than the object itself. The user is then responsible for making sure that the original object gets used somewhere else or is explicitly freed.

The arguments and return values of all documented functions are annotated to make clear which arguments are released and which arguments are preserved. In particular, the following annotations are used

`__isl_give`

> `__isl_give` means that a new object is returned. The user should make sure that the returned pointer is used exactly once as a value for an `__isl_take` argument. In between, it can be used as a value for as many `__isl_keep` arguments as the user likes. There is one exception, and that is the case where the pointer returned is `NULL`. Is this case, the user is free to use it as an `__isl_take` argument or not. When applied to a `char *`, the returned pointer needs to be freed using `free`.

`__isl_null`

> `__isl_null` means that a `NULL` value is returned.

`__isl_take`

> `__isl_take` means that the object the argument points to is taken over by the function and may no longer be used by the user as an argument to any other function. The pointer value must be one returned by a function returning an `__isl_give` pointer. If the user passes in a `NULL` value, then this will be treated as an error in the sense that the function will not perform its usual operation. However, it will still make sure that all the other `__isl_take` arguments are released.

`__isl_keep`

> `__isl_keep` means that the function will only use the object temporarily. After the function has finished, the user can still use it as an argument to other functions. A `NULL` value will be treated in the same way as a `NULL` value for an `__isl_take` argument. This annotation may also be used on return values of type `const char *`, in which case the returned pointer should not be freed by the user and is only valid until the object from which it was derived is updated or freed.

1.4.2 Initialization

All manipulations of integer sets and relations occur within the context of an `isl_ctx`. A given `isl_ctx` can only be used within a single thread. All arguments of a function

are required to have been allocated within the same context. There are currently no functions available for moving an object from one isl_ctx to another isl_ctx. This means that there is currently no way of safely moving an object from one thread to another, unless the whole isl_ctx is moved.

An isl_ctx can be allocated using isl_ctx_alloc and freed using isl_ctx_free. All objects allocated within an isl_ctx should be freed before the isl_ctx itself is freed.

```
isl_ctx *isl_ctx_alloc();
void isl_ctx_free(isl_ctx *ctx);
```

The user can impose a bound on the number of low-level *operations* that can be performed by an isl_ctx. This bound can be set and retrieved using the following functions. A bound of zero means that no bound is imposed. The number of operations performed can be reset using isl_ctx_reset_operations. Note that the number of low-level operations needed to perform a high-level computation may differ significantly across different versions of isl, but it should be the same across different platforms for the same version of isl.

Warning: This feature is experimental. isl has good support to abort and bail out during the computation, but this feature may exercise error code paths that are normally not used that much. Consequently, it is not unlikely that hidden bugs will be exposed.

```
void isl_ctx_set_max_operations(isl_ctx *ctx,
        unsigned long max_operations);
unsigned long isl_ctx_get_max_operations(isl_ctx *ctx);
void isl_ctx_reset_operations(isl_ctx *ctx);
```

In order to be able to create an object in the same context as another object, most object types (described later in this document) provide a function to obtain the context in which the object was created.

```
#include <isl/val.h>
isl_ctx *isl_val_get_ctx(__isl_keep isl_val *val);
isl_ctx *isl_multi_val_get_ctx(
        __isl_keep isl_multi_val *mv);

#include <isl/id.h>
isl_ctx *isl_id_get_ctx(__isl_keep isl_id *id);

#include <isl/local_space.h>
isl_ctx *isl_local_space_get_ctx(
        __isl_keep isl_local_space *ls);

#include <isl/set.h>
isl_ctx *isl_set_list_get_ctx(
        __isl_keep isl_set_list *list);
```

```
#include <isl/aff.h>
isl_ctx *isl_aff_get_ctx(__isl_keep isl_aff *aff);
isl_ctx *isl_multi_aff_get_ctx(
        __isl_keep isl_multi_aff *maff);
isl_ctx *isl_pw_aff_get_ctx(__isl_keep isl_pw_aff *pa);
isl_ctx *isl_pw_multi_aff_get_ctx(
        __isl_keep isl_pw_multi_aff *pma);
isl_ctx *isl_multi_pw_aff_get_ctx(
        __isl_keep isl_multi_pw_aff *mpa);
isl_ctx *isl_union_pw_aff_get_ctx(
        __isl_keep isl_union_pw_aff *upa);
isl_ctx *isl_union_pw_multi_aff_get_ctx(
        __isl_keep isl_union_pw_multi_aff *upma);
isl_ctx *isl_multi_union_pw_aff_get_ctx(
        __isl_keep isl_multi_union_pw_aff *mupa);

#include <isl/id_to_ast_expr.h>
isl_ctx *isl_id_to_ast_expr_get_ctx(
        __isl_keep isl_id_to_ast_expr *id2expr);

#include <isl/point.h>
isl_ctx *isl_point_get_ctx(__isl_keep isl_point *pnt);

#include <isl/vec.h>
isl_ctx *isl_vec_get_ctx(__isl_keep isl_vec *vec);

#include <isl/mat.h>
isl_ctx *isl_mat_get_ctx(__isl_keep isl_mat *mat);

#include <isl/vertices.h>
isl_ctx *isl_vertices_get_ctx(
        __isl_keep isl_vertices *vertices);
isl_ctx *isl_vertex_get_ctx(__isl_keep isl_vertex *vertex);
isl_ctx *isl_cell_get_ctx(__isl_keep isl_cell *cell);

#include <isl/flow.h>
isl_ctx *isl_restriction_get_ctx(
        __isl_keep isl_restriction *restr);
isl_ctx *isl_union_access_info_get_ctx(
        __isl_keep isl_union_access_info *access);
isl_ctx *isl_union_flow_get_ctx(
        __isl_keep isl_union_flow *flow);

#include <isl/schedule.h>
isl_ctx *isl_schedule_get_ctx(
        __isl_keep isl_schedule *sched);
isl_ctx *isl_schedule_constraints_get_ctx(
        __isl_keep isl_schedule_constraints *sc);
```

```
#include <isl/schedule_node.h>
isl_ctx *isl_schedule_node_get_ctx(
        __isl_keep isl_schedule_node *node);

#include <isl/band.h>
isl_ctx *isl_band_get_ctx(__isl_keep isl_band *band);

#include <isl/ast_build.h>
isl_ctx *isl_ast_build_get_ctx(
        __isl_keep isl_ast_build *build);

#include <isl/ast.h>
isl_ctx *isl_ast_expr_get_ctx(
        __isl_keep isl_ast_expr *expr);
isl_ctx *isl_ast_node_get_ctx(
        __isl_keep isl_ast_node *node);
```

1.4.3 Return Types

`isl` uses two special return types for functions that either return a boolean or that in principle do not return anything. In particular, the `isl_bool` type has three possible values: `isl_bool_true` (a positive integer value), indicating *true* or *yes*; `isl_bool_false` (the integer value zero), indicating *false* or *no*; and `isl_bool_error` (a negative integer value), indicating that something went wrong. The following function can be used to negate an `isl_bool`, where the negation of `isl_bool_error` is `isl_bool_error` again.

```
#include <isl/val.h>
isl_bool isl_bool_not(isl_bool b);
```

The `isl_stat` type has two possible values: `isl_stat_ok` (the integer value zero), indicating a successful operation; and `isl_stat_error` (a negative integer value), indicating that something went wrong. See §1.4.6 for more information on `isl_bool_error` and `isl_stat_error`.

1.4.4 Values

An `isl_val` represents an integer value, a rational value or one of three special values, infinity, negative infinity and NaN. Some predefined values can be created using the following functions.

```
#include <isl/val.h>
__isl_give isl_val *isl_val_zero(isl_ctx *ctx);
__isl_give isl_val *isl_val_one(isl_ctx *ctx);
__isl_give isl_val *isl_val_negone(isl_ctx *ctx);
__isl_give isl_val *isl_val_nan(isl_ctx *ctx);
__isl_give isl_val *isl_val_infty(isl_ctx *ctx);
__isl_give isl_val *isl_val_neginfty(isl_ctx *ctx);
```

Specific integer values can be created using the following functions.

```
#include <isl/val.h>
__isl_give isl_val *isl_val_int_from_si(isl_ctx *ctx,
        long i);
__isl_give isl_val *isl_val_int_from_ui(isl_ctx *ctx,
        unsigned long u);
__isl_give isl_val *isl_val_int_from_chunks(isl_ctx *ctx,
        size_t n, size_t size, const void *chunks);
```

The function isl_val_int_from_chunks constructs an isl_val from the n *digits*, each consisting of size bytes, stored at chunks. The least significant digit is assumed to be stored first.

Value objects can be copied and freed using the following functions.

```
#include <isl/val.h>
__isl_give isl_val *isl_val_copy(__isl_keep isl_val *v);
__isl_null isl_val *isl_val_free(__isl_take isl_val *v);
```

They can be inspected using the following functions.

```
#include <isl/val.h>
long isl_val_get_num_si(__isl_keep isl_val *v);
long isl_val_get_den_si(__isl_keep isl_val *v);
__isl_give isl_val *isl_val_get_den_val(
        __isl_keep isl_val *v);
double isl_val_get_d(__isl_keep isl_val *v);
size_t isl_val_n_abs_num_chunks(__isl_keep isl_val *v,
        size_t size);
int isl_val_get_abs_num_chunks(__isl_keep isl_val *v,
        size_t size, void *chunks);
```

isl_val_n_abs_num_chunks returns the number of *digits* of size bytes needed to store the absolute value of the numerator of v. isl_val_get_abs_num_chunks stores these digits at chunks, which is assumed to have been preallocated by the caller. The least significant digit is stored first. Note that isl_val_get_num_si, isl_val_get_den_si, isl_val_get_d, isl_val_n_abs_num_chunks and isl_val_get_abs_num_chunks can only be applied to rational values.

An isl_val can be modified using the following function.

```
#include <isl/val.h>
__isl_give isl_val *isl_val_set_si(__isl_take isl_val *v,
        long i);
```

The following unary properties are defined on isl_vals.

```
#include <isl/val.h>
int isl_val_sgn(__isl_keep isl_val *v);
isl_bool isl_val_is_zero(__isl_keep isl_val *v);
isl_bool isl_val_is_one(__isl_keep isl_val *v);
isl_bool isl_val_is_negone(__isl_keep isl_val *v);
isl_bool isl_val_is_nonneg(__isl_keep isl_val *v);
isl_bool isl_val_is_nonpos(__isl_keep isl_val *v);
isl_bool isl_val_is_pos(__isl_keep isl_val *v);
isl_bool isl_val_is_neg(__isl_keep isl_val *v);
isl_bool isl_val_is_int(__isl_keep isl_val *v);
isl_bool isl_val_is_rat(__isl_keep isl_val *v);
isl_bool isl_val_is_nan(__isl_keep isl_val *v);
isl_bool isl_val_is_infty(__isl_keep isl_val *v);
isl_bool isl_val_is_neginfty(__isl_keep isl_val *v);
```

Note that the sign of NaN is undefined.

The following binary properties are defined on pairs of `isl_val`s.

```
#include <isl/val.h>
isl_bool isl_val_lt(__isl_keep isl_val *v1,
        __isl_keep isl_val *v2);
isl_bool isl_val_le(__isl_keep isl_val *v1,
        __isl_keep isl_val *v2);
isl_bool isl_val_gt(__isl_keep isl_val *v1,
        __isl_keep isl_val *v2);
isl_bool isl_val_ge(__isl_keep isl_val *v1,
        __isl_keep isl_val *v2);
isl_bool isl_val_eq(__isl_keep isl_val *v1,
        __isl_keep isl_val *v2);
isl_bool isl_val_ne(__isl_keep isl_val *v1,
        __isl_keep isl_val *v2);
isl_bool isl_val_abs_eq(__isl_keep isl_val *v1,
        __isl_keep isl_val *v2);
```

The function `isl_val_abs_eq` checks whether its two arguments are equal in absolute value.

For integer `isl_val`s we additionally have the following binary property.

```
#include <isl/val.h>
isl_bool isl_val_is_divisible_by(__isl_keep isl_val *v1,
        __isl_keep isl_val *v2);
```

An `isl_val` can also be compared to an integer using the following function. The result is undefined for NaN.

```
#include <isl/val.h>
int isl_val_cmp_si(__isl_keep isl_val *v, long i);
```

The following unary operations are available on isl_vals.

```
#include <isl/val.h>
__isl_give isl_val *isl_val_abs(__isl_take isl_val *v);
__isl_give isl_val *isl_val_neg(__isl_take isl_val *v);
__isl_give isl_val *isl_val_floor(__isl_take isl_val *v);
__isl_give isl_val *isl_val_ceil(__isl_take isl_val *v);
__isl_give isl_val *isl_val_trunc(__isl_take isl_val *v);
__isl_give isl_val *isl_val_inv(__isl_take isl_val *v);
__isl_give isl_val *isl_val_2exp(__isl_take isl_val *v);
```

The following binary operations are available on isl_vals.

```
#include <isl/val.h>
__isl_give isl_val *isl_val_min(__isl_take isl_val *v1,
        __isl_take isl_val *v2);
__isl_give isl_val *isl_val_max(__isl_take isl_val *v1,
        __isl_take isl_val *v2);
__isl_give isl_val *isl_val_add(__isl_take isl_val *v1,
        __isl_take isl_val *v2);
__isl_give isl_val *isl_val_add_ui(__isl_take isl_val *v1,
        unsigned long v2);
__isl_give isl_val *isl_val_sub(__isl_take isl_val *v1,
        __isl_take isl_val *v2);
__isl_give isl_val *isl_val_sub_ui(__isl_take isl_val *v1,
        unsigned long v2);
__isl_give isl_val *isl_val_mul(__isl_take isl_val *v1,
        __isl_take isl_val *v2);
__isl_give isl_val *isl_val_mul_ui(__isl_take isl_val *v1,
        unsigned long v2);
__isl_give isl_val *isl_val_div(__isl_take isl_val *v1,
        __isl_take isl_val *v2);
```

On integer values, we additionally have the following operations.

```
#include <isl/val.h>
__isl_give isl_val *isl_val_2exp(__isl_take isl_val *v);
__isl_give isl_val *isl_val_mod(__isl_take isl_val *v1,
        __isl_take isl_val *v2);
__isl_give isl_val *isl_val_gcd(__isl_take isl_val *v1,
        __isl_take isl_val *v2);
__isl_give isl_val *isl_val_gcdext(__isl_take isl_val *v1,
        __isl_take isl_val *v2, __isl_give isl_val **x,
        __isl_give isl_val **y);
```

The function isl_val_gcdext returns the greatest common divisor g of v1 and v2 as well as two integers *x and *y such that *x * v1 + *y * v2 = g.

GMP specific functions

These functions are only available if isl has been compiled with GMP support.

Specific integer and rational values can be created from GMP values using the following functions.

```
#include <isl/val_gmp.h>
__isl_give isl_val *isl_val_int_from_gmp(isl_ctx *ctx,
        mpz_t z);
__isl_give isl_val *isl_val_from_gmp(isl_ctx *ctx,
        const mpz_t n, const mpz_t d);
```

The numerator and denominator of a rational value can be extracted as GMP values using the following functions.

```
#include <isl/val_gmp.h>
int isl_val_get_num_gmp(__isl_keep isl_val *v, mpz_t z);
int isl_val_get_den_gmp(__isl_keep isl_val *v, mpz_t z);
```

1.4.5 Sets and Relations

isl uses six types of objects for representing sets and relations, isl_basic_set, isl_basic_map, isl_set, isl_map, isl_union_set and isl_union_map. isl_basic_set and isl_basic_map represent sets and relations that can be described as a conjunction of affine constraints, while isl_set and isl_map represent unions of isl_basic_sets and isl_basic_maps, respectively. However, all isl_basic_sets or isl_basic_maps in the union need to live in the same space. isl_union_sets and isl_union_maps represent unions of isl_sets or isl_maps in *different* spaces, where spaces are considered different if they have a different number of dimensions and/or different names (see §1.4.8). The difference between sets and relations (maps) is that sets have one set of variables, while relations have two sets of variables, input variables and output variables.

1.4.6 Error Handling

isl supports different ways to react in case a runtime error is triggered. Runtime errors arise, e.g., if a function such as isl_map_intersect is called with two maps that have incompatible spaces. There are three possible ways to react on error: to warn, to continue or to abort.

The default behavior is to warn. In this mode, isl prints a warning, stores the last error in the corresponding isl_ctx and the function in which the error was triggered returns a value indicating that some error has occurred. In case of functions returning a pointer, this value is NULL. In case of functions returning an isl_bool or an isl_stat, this valus is isl_bool_error or isl_stat_error. An error does not corrupt internal state, such that isl can continue to be used. isl also provides functions to read the last error and to reset the memory that stores the last error. The last error is only stored for information purposes. Its presence does not change the behavior of isl. Hence, resetting an error is not required to continue to use isl, but only to observe new errors.

```
#include <isl/ctx.h>
enum isl_error isl_ctx_last_error(isl_ctx *ctx);
void isl_ctx_reset_error(isl_ctx *ctx);
```

Another option is to continue on error. This is similar to warn on error mode, except that `isl` does not print any warning. This allows a program to implement its own error reporting.

The last option is to directly abort the execution of the program from within the isl library. This makes it obviously impossible to recover from an error, but it allows to directly spot the error location. By aborting on error, debuggers break at the location the error occurred and can provide a stack trace. Other tools that automatically provide stack traces on abort or that do not want to continue execution after an error was triggered may also prefer to abort on error.

The on error behavior of isl can be specified by calling `isl_options_set_on_error` or by setting the command line option `--isl-on-error`. Valid arguments for the function call are `ISL_ON_ERROR_WARN`, `ISL_ON_ERROR_CONTINUE` and `ISL_ON_ERROR_ABORT`. The choices for the command line option are `warn`, `continue` and `abort`. It is also possible to query the current error mode.

```
#include <isl/options.h>
isl_stat isl_options_set_on_error(isl_ctx *ctx, int val);
int isl_options_get_on_error(isl_ctx *ctx);
```

1.4.7 Identifiers

Identifiers are used to identify both individual dimensions and tuples of dimensions. They consist of an optional name and an optional user pointer. The name and the user pointer cannot both be `NULL`, however. Identifiers with the same name but different pointer values are considered to be distinct. Similarly, identifiers with different names but the same pointer value are also considered to be distinct. Equal identifiers are represented using the same object. Pairs of identifiers can therefore be tested for equality using the == operator. Identifiers can be constructed, copied, freed, inspected and printed using the following functions.

```
#include <isl/id.h>
__isl_give isl_id *isl_id_alloc(isl_ctx *ctx,
        __isl_keep const char *name, void *user);
__isl_give isl_id *isl_id_set_free_user(
        __isl_take isl_id *id,
        void (*free_user)(void *user));
__isl_give isl_id *isl_id_copy(isl_id *id);
__isl_null isl_id *isl_id_free(__isl_take isl_id *id);

void *isl_id_get_user(__isl_keep isl_id *id);
__isl_keep const char *isl_id_get_name(__isl_keep isl_id *id);

__isl_give isl_printer *isl_printer_print_id(
        __isl_take isl_printer *p, __isl_keep isl_id *id);
```

The callback set by isl_id_set_free_user is called on the user pointer when the last reference to the isl_id is freed. Note that isl_id_get_name returns a pointer to some internal data structure, so the result can only be used while the corresponding isl_id is alive.

1.4.8 Spaces

Whenever a new set, relation or similar object is created from scratch, the space in which it lives needs to be specified using an isl_space. Each space involves zero or more parameters and zero, one or two tuples of set or input/output dimensions. The parameters and dimensions are identified by an isl_dim_type and a position. The type isl_dim_param refers to parameters, the type isl_dim_set refers to set dimensions (for spaces with a single tuple of dimensions) and the types isl_dim_in and isl_dim_out refer to input and output dimensions (for spaces with two tuples of dimensions). Local spaces (see §1.4.9) also contain dimensions of type isl_dim_div. Note that parameters are only identified by their position within a given object. Across different objects, parameters are (usually) identified by their names or identifiers. Only unnamed parameters are identified by their positions across objects. The use of unnamed parameters is discouraged.

```
#include <isl/space.h>
__isl_give isl_space *isl_space_alloc(isl_ctx *ctx,
        unsigned nparam, unsigned n_in, unsigned n_out);
__isl_give isl_space *isl_space_params_alloc(isl_ctx *ctx,
        unsigned nparam);
__isl_give isl_space *isl_space_set_alloc(isl_ctx *ctx,
        unsigned nparam, unsigned dim);
__isl_give isl_space *isl_space_copy(__isl_keep isl_space *space);
__isl_null isl_space *isl_space_free(__isl_take isl_space *space);
```

The space used for creating a parameter domain needs to be created using isl_space_params_alloc. For other sets, the space needs to be created using isl_space_set_alloc, while for a relation, the space needs to be created using isl_space_alloc.

To check whether a given space is that of a set or a map or whether it is a parameter space, use these functions:

```
#include <isl/space.h>
isl_bool isl_space_is_params(__isl_keep isl_space *space);
isl_bool isl_space_is_set(__isl_keep isl_space *space);
isl_bool isl_space_is_map(__isl_keep isl_space *space);
```

Spaces can be compared using the following functions:

```
#include <isl/space.h>
isl_bool isl_space_is_equal(__isl_keep isl_space *space1,
        __isl_keep isl_space *space2);
isl_bool isl_space_is_domain(__isl_keep isl_space *space1,
```

```
        __isl_keep isl_space *space2);
isl_bool isl_space_is_range(__isl_keep isl_space *space1,
        __isl_keep isl_space *space2);
isl_bool isl_space_tuple_is_equal(
        __isl_keep isl_space *space1,
        enum isl_dim_type type1,
        __isl_keep isl_space *space2,
        enum isl_dim_type type2);
```

`isl_space_is_domain` checks whether the first argument is equal to the domain of the second argument. This requires in particular that the first argument is a set space and that the second argument is a map space. `isl_space_tuple_is_equal` checks whether the given tuples (`isl_dim_in`, `isl_dim_out` or `isl_dim_set`) of the given spaces are the same. That is, it checks if they have the same identifier (if any), the same dimension and the same internal structure (if any).

It is often useful to create objects that live in the same space as some other object. This can be accomplished by creating the new objects (see §1.4.10 or §1.4.13) based on the space of the original object.

```
#include <isl/set.h>
__isl_give isl_space *isl_basic_set_get_space(
        __isl_keep isl_basic_set *bset);
__isl_give isl_space *isl_set_get_space(__isl_keep isl_set *set);

#include <isl/union_set.h>
__isl_give isl_space *isl_union_set_get_space(
        __isl_keep isl_union_set *uset);

#include <isl/map.h>
__isl_give isl_space *isl_basic_map_get_space(
        __isl_keep isl_basic_map *bmap);
__isl_give isl_space *isl_map_get_space(__isl_keep isl_map *map);

#include <isl/union_map.h>
__isl_give isl_space *isl_union_map_get_space(
        __isl_keep isl_union_map *umap);

#include <isl/constraint.h>
__isl_give isl_space *isl_constraint_get_space(
        __isl_keep isl_constraint *constraint);

#include <isl/polynomial.h>
__isl_give isl_space *isl_qpolynomial_get_domain_space(
        __isl_keep isl_qpolynomial *qp);
__isl_give isl_space *isl_qpolynomial_get_space(
        __isl_keep isl_qpolynomial *qp);
__isl_give isl_space *
```

```
isl_qpolynomial_fold_get_domain_space(
        __isl_keep isl_qpolynomial_fold *fold);
__isl_give isl_space *isl_qpolynomial_fold_get_space(
        __isl_keep isl_qpolynomial_fold *fold);
__isl_give isl_space *isl_pw_qpolynomial_get_domain_space(
        __isl_keep isl_pw_qpolynomial *pwqp);
__isl_give isl_space *isl_pw_qpolynomial_get_space(
        __isl_keep isl_pw_qpolynomial *pwqp);
__isl_give isl_space *isl_pw_qpolynomial_fold_get_domain_space(
        __isl_keep isl_pw_qpolynomial_fold *pwf);
__isl_give isl_space *isl_pw_qpolynomial_fold_get_space(
        __isl_keep isl_pw_qpolynomial_fold *pwf);
__isl_give isl_space *isl_union_pw_qpolynomial_get_space(
        __isl_keep isl_union_pw_qpolynomial *upwqp);
__isl_give isl_space *isl_union_pw_qpolynomial_fold_get_space(
        __isl_keep isl_union_pw_qpolynomial_fold *upwf);

#include <isl/val.h>
__isl_give isl_space *isl_multi_val_get_space(
        __isl_keep isl_multi_val *mv);

#include <isl/aff.h>
__isl_give isl_space *isl_aff_get_domain_space(
        __isl_keep isl_aff *aff);
__isl_give isl_space *isl_aff_get_space(
        __isl_keep isl_aff *aff);
__isl_give isl_space *isl_pw_aff_get_domain_space(
        __isl_keep isl_pw_aff *pwaff);
__isl_give isl_space *isl_pw_aff_get_space(
        __isl_keep isl_pw_aff *pwaff);
__isl_give isl_space *isl_multi_aff_get_domain_space(
        __isl_keep isl_multi_aff *maff);
__isl_give isl_space *isl_multi_aff_get_space(
        __isl_keep isl_multi_aff *maff);
__isl_give isl_space *isl_pw_multi_aff_get_domain_space(
        __isl_keep isl_pw_multi_aff *pma);
__isl_give isl_space *isl_pw_multi_aff_get_space(
        __isl_keep isl_pw_multi_aff *pma);
__isl_give isl_space *isl_union_pw_aff_get_space(
        __isl_keep isl_union_pw_aff *upa);
__isl_give isl_space *isl_union_pw_multi_aff_get_space(
        __isl_keep isl_union_pw_multi_aff *upma);
__isl_give isl_space *isl_multi_pw_aff_get_domain_space(
        __isl_keep isl_multi_pw_aff *mpa);
__isl_give isl_space *isl_multi_pw_aff_get_space(
        __isl_keep isl_multi_pw_aff *mpa);
```

```
__isl_give isl_space *
isl_multi_union_pw_aff_get_domain_space(
        __isl_keep isl_multi_union_pw_aff *mupa);
__isl_give isl_space *
isl_multi_union_pw_aff_get_space(
        __isl_keep isl_multi_union_pw_aff *mupa);

#include <isl/point.h>
__isl_give isl_space *isl_point_get_space(
        __isl_keep isl_point *pnt);
```

The number of dimensions of a given type of space may be read off from a space or an object that lives in a space using the following functions. In case of `isl_space_dim`, type may be `isl_dim_param`, `isl_dim_in` (only for relations), `isl_dim_out` (only for relations), `isl_dim_set` (only for sets) or `isl_dim_all`.

```
#include <isl/space.h>
unsigned isl_space_dim(__isl_keep isl_space *space,
        enum isl_dim_type type);

#include <isl/local_space.h>
int isl_local_space_dim(__isl_keep isl_local_space *ls,
        enum isl_dim_type type);

#include <isl/set.h>
unsigned isl_basic_set_dim(__isl_keep isl_basic_set *bset,
        enum isl_dim_type type);
unsigned isl_set_dim(__isl_keep isl_set *set,
        enum isl_dim_type type);

#include <isl/union_set.h>
unsigned isl_union_set_dim(__isl_keep isl_union_set *uset,
        enum isl_dim_type type);

#include <isl/map.h>
unsigned isl_basic_map_dim(__isl_keep isl_basic_map *bmap,
        enum isl_dim_type type);
unsigned isl_map_dim(__isl_keep isl_map *map,
        enum isl_dim_type type);

#include <isl/union_map.h>
unsigned isl_union_map_dim(__isl_keep isl_union_map *umap,
        enum isl_dim_type type);

#include <isl/val.h>
unsigned isl_multi_val_dim(__isl_keep isl_multi_val *mv,
        enum isl_dim_type type);
```

```
#include <isl/aff.h>
int isl_aff_dim(__isl_keep isl_aff *aff,
        enum isl_dim_type type);
unsigned isl_multi_aff_dim(__isl_keep isl_multi_aff *maff,
        enum isl_dim_type type);
unsigned isl_pw_aff_dim(__isl_keep isl_pw_aff *pwaff,
        enum isl_dim_type type);
unsigned isl_pw_multi_aff_dim(
        __isl_keep isl_pw_multi_aff *pma,
        enum isl_dim_type type);
unsigned isl_multi_pw_aff_dim(
        __isl_keep isl_multi_pw_aff *mpa,
        enum isl_dim_type type);
unsigned isl_union_pw_aff_dim(
        __isl_keep isl_union_pw_aff *upa,
        enum isl_dim_type type);
unsigned isl_union_pw_multi_aff_dim(
        __isl_keep isl_union_pw_multi_aff *upma,
        enum isl_dim_type type);
unsigned isl_multi_union_pw_aff_dim(
        __isl_keep isl_multi_union_pw_aff *mupa,
        enum isl_dim_type type);

#include <isl/polynomial.h>
unsigned isl_union_pw_qpolynomial_dim(
        __isl_keep isl_union_pw_qpolynomial *upwqp,
        enum isl_dim_type type);
unsigned isl_union_pw_qpolynomial_fold_dim(
        __isl_keep isl_union_pw_qpolynomial_fold *upwf,
        enum isl_dim_type type);
```

Note that an `isl_union_set`, an `isl_union_map`, an `isl_union_pw_multi_aff`, an `isl_union_pw_qpolynomial` and an `isl_union_pw_qpolynomial_fold` only have parameters.

The identifiers or names of the individual dimensions of spaces may be set or read off using the following functions on spaces or objects that live in spaces. These functions are mostly useful to obtain the identifiers, positions or names of the parameters. Identifiers of individual dimensions are essentially only useful for printing. They are ignored by all other operations and may not be preserved across those operations.

```
#include <isl/space.h>
__isl_give isl_space *isl_space_set_dim_id(
        __isl_take isl_space *space,
        enum isl_dim_type type, unsigned pos,
        __isl_take isl_id *id);
isl_bool isl_space_has_dim_id(__isl_keep isl_space *space,
        enum isl_dim_type type, unsigned pos);
```

```
__isl_give isl_id *isl_space_get_dim_id(
        __isl_keep isl_space *space,
        enum isl_dim_type type, unsigned pos);
__isl_give isl_space *isl_space_set_dim_name(
        __isl_take isl_space *space,
         enum isl_dim_type type, unsigned pos,
         __isl_keep const char *name);
isl_bool isl_space_has_dim_name(__isl_keep isl_space *space,
        enum isl_dim_type type, unsigned pos);
__isl_keep const char *isl_space_get_dim_name(
        __isl_keep isl_space *space,
        enum isl_dim_type type, unsigned pos);

#include <isl/local_space.h>
__isl_give isl_local_space *isl_local_space_set_dim_id(
        __isl_take isl_local_space *ls,
        enum isl_dim_type type, unsigned pos,
        __isl_take isl_id *id);
isl_bool isl_local_space_has_dim_id(
        __isl_keep isl_local_space *ls,
        enum isl_dim_type type, unsigned pos);
__isl_give isl_id *isl_local_space_get_dim_id(
        __isl_keep isl_local_space *ls,
        enum isl_dim_type type, unsigned pos);
__isl_give isl_local_space *isl_local_space_set_dim_name(
        __isl_take isl_local_space *ls,
        enum isl_dim_type type, unsigned pos, const char *s);
isl_bool isl_local_space_has_dim_name(
        __isl_keep isl_local_space *ls,
        enum isl_dim_type type, unsigned pos)
const char *isl_local_space_get_dim_name(
        __isl_keep isl_local_space *ls,
        enum isl_dim_type type, unsigned pos);

#include <isl/constraint.h>
const char *isl_constraint_get_dim_name(
        __isl_keep isl_constraint *constraint,
        enum isl_dim_type type, unsigned pos);

#include <isl/set.h>
__isl_give isl_id *isl_basic_set_get_dim_id(
        __isl_keep isl_basic_set *bset,
        enum isl_dim_type type, unsigned pos);
__isl_give isl_set *isl_set_set_dim_id(
        __isl_take isl_set *set, enum isl_dim_type type,
        unsigned pos, __isl_take isl_id *id);
isl_bool isl_set_has_dim_id(__isl_keep isl_set *set,
```

```
        enum isl_dim_type type, unsigned pos);
__isl_give isl_id *isl_set_get_dim_id(
        __isl_keep isl_set *set, enum isl_dim_type type,
        unsigned pos);
const char *isl_basic_set_get_dim_name(
        __isl_keep isl_basic_set *bset,
        enum isl_dim_type type, unsigned pos);
isl_bool isl_set_has_dim_name(__isl_keep isl_set *set,
        enum isl_dim_type type, unsigned pos);
const char *isl_set_get_dim_name(
        __isl_keep isl_set *set,
        enum isl_dim_type type, unsigned pos);

#include <isl/map.h>
__isl_give isl_map *isl_map_set_dim_id(
        __isl_take isl_map *map, enum isl_dim_type type,
        unsigned pos, __isl_take isl_id *id);
isl_bool isl_basic_map_has_dim_id(
        __isl_keep isl_basic_map *bmap,
        enum isl_dim_type type, unsigned pos);
isl_bool isl_map_has_dim_id(__isl_keep isl_map *map,
        enum isl_dim_type type, unsigned pos);
__isl_give isl_id *isl_map_get_dim_id(
        __isl_keep isl_map *map, enum isl_dim_type type,
        unsigned pos);
__isl_give isl_id *isl_union_map_get_dim_id(
        __isl_keep isl_union_map *umap,
        enum isl_dim_type type, unsigned pos);
const char *isl_basic_map_get_dim_name(
        __isl_keep isl_basic_map *bmap,
        enum isl_dim_type type, unsigned pos);
isl_bool isl_map_has_dim_name(__isl_keep isl_map *map,
        enum isl_dim_type type, unsigned pos);
const char *isl_map_get_dim_name(
        __isl_keep isl_map *map,
        enum isl_dim_type type, unsigned pos);

#include <isl/val.h>
__isl_give isl_multi_val *isl_multi_val_set_dim_id(
        __isl_take isl_multi_val *mv,
        enum isl_dim_type type, unsigned pos,
        __isl_take isl_id *id);
__isl_give isl_id *isl_multi_val_get_dim_id(
        __isl_keep isl_multi_val *mv,
        enum isl_dim_type type, unsigned pos);
__isl_give isl_multi_val *isl_multi_val_set_dim_name(
```

```
        __isl_take isl_multi_val *mv,
        enum isl_dim_type type, unsigned pos, const char *s);

#include <isl/aff.h>
__isl_give isl_aff *isl_aff_set_dim_id(
        __isl_take isl_aff *aff, enum isl_dim_type type,
        unsigned pos, __isl_take isl_id *id);
__isl_give isl_multi_aff *isl_multi_aff_set_dim_id(
        __isl_take isl_multi_aff *maff,
        enum isl_dim_type type, unsigned pos,
        __isl_take isl_id *id);
__isl_give isl_pw_aff *isl_pw_aff_set_dim_id(
        __isl_take isl_pw_aff *pma,
        enum isl_dim_type type, unsigned pos,
        __isl_take isl_id *id);
__isl_give isl_multi_pw_aff *
isl_multi_pw_aff_set_dim_id(
        __isl_take isl_multi_pw_aff *mpa,
        enum isl_dim_type type, unsigned pos,
        __isl_take isl_id *id);
__isl_give isl_multi_union_pw_aff *
isl_multi_union_pw_aff_set_dim_id(
        __isl_take isl_multi_union_pw_aff *mupa,
        enum isl_dim_type type, unsigned pos,
        __isl_take isl_id *id);
__isl_give isl_id *isl_multi_aff_get_dim_id(
        __isl_keep isl_multi_aff *ma,
        enum isl_dim_type type, unsigned pos);
isl_bool isl_pw_aff_has_dim_id(__isl_keep isl_pw_aff *pa,
        enum isl_dim_type type, unsigned pos);
__isl_give isl_id *isl_pw_aff_get_dim_id(
        __isl_keep isl_pw_aff *pa,
        enum isl_dim_type type, unsigned pos);
__isl_give isl_id *isl_pw_multi_aff_get_dim_id(
        __isl_keep isl_pw_multi_aff *pma,
        enum isl_dim_type type, unsigned pos);
__isl_give isl_id *isl_multi_pw_aff_get_dim_id(
        __isl_keep isl_multi_pw_aff *mpa,
        enum isl_dim_type type, unsigned pos);
__isl_give isl_id *isl_multi_union_pw_aff_get_dim_id(
        __isl_keep isl_multi_union_pw_aff *mupa,
        enum isl_dim_type type, unsigned pos);
__isl_give isl_aff *isl_aff_set_dim_name(
        __isl_take isl_aff *aff, enum isl_dim_type type,
        unsigned pos, const char *s);
__isl_give isl_multi_aff *isl_multi_aff_set_dim_name(
```

```
        __isl_take isl_multi_aff *maff,
        enum isl_dim_type type, unsigned pos, const char *s);
__isl_give isl_multi_pw_aff *
isl_multi_pw_aff_set_dim_name(
        __isl_take isl_multi_pw_aff *mpa,
        enum isl_dim_type type, unsigned pos, const char *s);
__isl_give isl_union_pw_aff *
isl_union_pw_aff_set_dim_name(
        __isl_take isl_union_pw_aff *upa,
        enum isl_dim_type type, unsigned pos,
        const char *s);
__isl_give isl_union_pw_multi_aff *
isl_union_pw_multi_aff_set_dim_name(
        __isl_take isl_union_pw_multi_aff *upma,
        enum isl_dim_type type, unsigned pos,
        const char *s);
__isl_give isl_multi_union_pw_aff *
isl_multi_union_pw_aff_set_dim_name(
        __isl_take isl_multi_union_pw_aff *mupa,
        enum isl_dim_type type, unsigned pos,
const char *isl_aff_get_dim_name(__isl_keep isl_aff *aff,
        enum isl_dim_type type, unsigned pos);
const char *isl_pw_aff_get_dim_name(
        __isl_keep isl_pw_aff *pa,
        enum isl_dim_type type, unsigned pos);
const char *isl_pw_multi_aff_get_dim_name(
        __isl_keep isl_pw_multi_aff *pma,
        enum isl_dim_type type, unsigned pos);

#include <isl/polynomial.h>
__isl_give isl_qpolynomial *isl_qpolynomial_set_dim_name(
        __isl_take isl_qpolynomial *qp,
        enum isl_dim_type type, unsigned pos,
        const char *s);
__isl_give isl_pw_qpolynomial *
isl_pw_qpolynomial_set_dim_name(
        __isl_take isl_pw_qpolynomial *pwqp,
        enum isl_dim_type type, unsigned pos,
        const char *s);
__isl_give isl_pw_qpolynomial_fold *
isl_pw_qpolynomial_fold_set_dim_name(
        __isl_take isl_pw_qpolynomial_fold *pwf,
        enum isl_dim_type type, unsigned pos,
        const char *s);
__isl_give isl_union_pw_qpolynomial *
isl_union_pw_qpolynomial_set_dim_name(
```

```
        __isl_take isl_union_pw_qpolynomial *upwqp,
        enum isl_dim_type type, unsigned pos,
        const char *s);
__isl_give isl_union_pw_qpolynomial_fold *
isl_union_pw_qpolynomial_fold_set_dim_name(
        __isl_take isl_union_pw_qpolynomial_fold *upwf,
        enum isl_dim_type type, unsigned pos,
        const char *s);
```

Note that `isl_space_get_name` returns a pointer to some internal data structure, so the result can only be used while the corresponding `isl_space` is alive. Also note that every function that operates on two sets or relations requires that both arguments have the same parameters. This also means that if one of the arguments has named parameters, then the other needs to have named parameters too and the names need to match. Pairs of `isl_set`, `isl_map`, `isl_union_set` and/or `isl_union_map` arguments may have different parameters (as long as they are named), in which case the result will have as parameters the union of the parameters of the arguments.

Given the identifier or name of a dimension (typically a parameter), its position can be obtained from the following functions.

```
#include <isl/space.h>
int isl_space_find_dim_by_id(__isl_keep isl_space *space,
        enum isl_dim_type type, __isl_keep isl_id *id);
int isl_space_find_dim_by_name(__isl_keep isl_space *space,
        enum isl_dim_type type, const char *name);

#include <isl/local_space.h>
int isl_local_space_find_dim_by_name(
        __isl_keep isl_local_space *ls,
        enum isl_dim_type type, const char *name);

#include <isl/val.h>
int isl_multi_val_find_dim_by_id(
        __isl_keep isl_multi_val *mv,
        enum isl_dim_type type, __isl_keep isl_id *id);
int isl_multi_val_find_dim_by_name(
        __isl_keep isl_multi_val *mv,
        enum isl_dim_type type, const char *name);

#include <isl/set.h>
int isl_set_find_dim_by_id(__isl_keep isl_set *set,
        enum isl_dim_type type, __isl_keep isl_id *id);
int isl_set_find_dim_by_name(__isl_keep isl_set *set,
        enum isl_dim_type type, const char *name);

#include <isl/map.h>
int isl_map_find_dim_by_id(__isl_keep isl_map *map,
```

```
        enum isl_dim_type type, __isl_keep isl_id *id);
int isl_basic_map_find_dim_by_name(
        __isl_keep isl_basic_map *bmap,
        enum isl_dim_type type, const char *name);
int isl_map_find_dim_by_name(__isl_keep isl_map *map,
        enum isl_dim_type type, const char *name);
int isl_union_map_find_dim_by_name(
        __isl_keep isl_union_map *umap,
        enum isl_dim_type type, const char *name);

#include <isl/aff.h>
int isl_multi_aff_find_dim_by_id(
        __isl_keep isl_multi_aff *ma,
        enum isl_dim_type type, __isl_keep isl_id *id);
int isl_multi_pw_aff_find_dim_by_id(
        __isl_keep isl_multi_pw_aff *mpa,
        enum isl_dim_type type, __isl_keep isl_id *id);
int isl_multi_union_pw_aff_find_dim_by_id(
        __isl_keep isl_union_multi_pw_aff *mupa,
        enum isl_dim_type type, __isl_keep isl_id *id);
int isl_aff_find_dim_by_name(__isl_keep isl_aff *aff,
        enum isl_dim_type type, const char *name);
int isl_multi_aff_find_dim_by_name(
        __isl_keep isl_multi_aff *ma,
        enum isl_dim_type type, const char *name);
int isl_pw_aff_find_dim_by_name(__isl_keep isl_pw_aff *pa,
        enum isl_dim_type type, const char *name);
int isl_multi_pw_aff_find_dim_by_name(
        __isl_keep isl_multi_pw_aff *mpa,
        enum isl_dim_type type, const char *name);
int isl_pw_multi_aff_find_dim_by_name(
        __isl_keep isl_pw_multi_aff *pma,
        enum isl_dim_type type, const char *name);
int isl_union_pw_aff_find_dim_by_name(
        __isl_keep isl_union_pw_aff *upa,
        enum isl_dim_type type, const char *name);
int isl_union_pw_multi_aff_find_dim_by_name(
        __isl_keep isl_union_pw_multi_aff *upma,
        enum isl_dim_type type, const char *name);
int isl_multi_union_pw_aff_find_dim_by_name(
        __isl_keep isl_multi_union_pw_aff *mupa,
        enum isl_dim_type type, const char *name);

#include <isl/polynomial.h>
int isl_pw_qpolynomial_find_dim_by_name(
        __isl_keep isl_pw_qpolynomial *pwqp,
```

```
        enum isl_dim_type type, const char *name);
int isl_pw_qpolynomial_fold_find_dim_by_name(
        __isl_keep isl_pw_qpolynomial_fold *pwf,
        enum isl_dim_type type, const char *name);
int isl_union_pw_qpolynomial_find_dim_by_name(
        __isl_keep isl_union_pw_qpolynomial *upwqp,
        enum isl_dim_type type, const char *name);
int isl_union_pw_qpolynomial_fold_find_dim_by_name(
        __isl_keep isl_union_pw_qpolynomial_fold *upwf,
        enum isl_dim_type type, const char *name);
```

The identifiers or names of entire spaces may be set or read off using the following functions.

```
#include <isl/space.h>
__isl_give isl_space *isl_space_set_tuple_id(
        __isl_take isl_space *space,
        enum isl_dim_type type, __isl_take isl_id *id);
__isl_give isl_space *isl_space_reset_tuple_id(
        __isl_take isl_space *space, enum isl_dim_type type);
isl_bool isl_space_has_tuple_id(
        __isl_keep isl_space *space,
        enum isl_dim_type type);
__isl_give isl_id *isl_space_get_tuple_id(
        __isl_keep isl_space *space, enum isl_dim_type type);
__isl_give isl_space *isl_space_set_tuple_name(
        __isl_take isl_space *space,
        enum isl_dim_type type, const char *s);
isl_bool isl_space_has_tuple_name(
        __isl_keep isl_space *space,
        enum isl_dim_type type);
const char *isl_space_get_tuple_name(__isl_keep isl_space *space,
        enum isl_dim_type type);

#include <isl/local_space.h>
__isl_give isl_local_space *isl_local_space_set_tuple_id(
        __isl_take isl_local_space *ls,
        enum isl_dim_type type, __isl_take isl_id *id);

#include <isl/set.h>
__isl_give isl_basic_set *isl_basic_set_set_tuple_id(
        __isl_take isl_basic_set *bset,
        __isl_take isl_id *id);
__isl_give isl_set *isl_set_set_tuple_id(
        __isl_take isl_set *set, __isl_take isl_id *id);
__isl_give isl_set *isl_set_reset_tuple_id(
        __isl_take isl_set *set);
```

```
isl_bool isl_set_has_tuple_id(__isl_keep isl_set *set);
__isl_give isl_id *isl_set_get_tuple_id(
        __isl_keep isl_set *set);
__isl_give isl_basic_set *isl_basic_set_set_tuple_name(
        __isl_take isl_basic_set *set, const char *s);
__isl_give isl_set *isl_set_set_tuple_name(
        __isl_take isl_set *set, const char *s);
const char *isl_basic_set_get_tuple_name(
        __isl_keep isl_basic_set *bset);
isl_bool isl_set_has_tuple_name(__isl_keep isl_set *set);
const char *isl_set_get_tuple_name(
        __isl_keep isl_set *set);

#include <isl/map.h>
__isl_give isl_basic_map *isl_basic_map_set_tuple_id(
        __isl_take isl_basic_map *bmap,
        enum isl_dim_type type, __isl_take isl_id *id);
__isl_give isl_map *isl_map_set_tuple_id(
        __isl_take isl_map *map, enum isl_dim_type type,
        __isl_take isl_id *id);
__isl_give isl_map *isl_map_reset_tuple_id(
        __isl_take isl_map *map, enum isl_dim_type type);
isl_bool isl_map_has_tuple_id(__isl_keep isl_map *map,
        enum isl_dim_type type);
__isl_give isl_id *isl_map_get_tuple_id(
        __isl_keep isl_map *map, enum isl_dim_type type);
__isl_give isl_map *isl_map_set_tuple_name(
        __isl_take isl_map *map,
        enum isl_dim_type type, const char *s);
const char *isl_basic_map_get_tuple_name(
        __isl_keep isl_basic_map *bmap,
        enum isl_dim_type type);
__isl_give isl_basic_map *isl_basic_map_set_tuple_name(
        __isl_take isl_basic_map *bmap,
        enum isl_dim_type type, const char *s);
isl_bool isl_map_has_tuple_name(__isl_keep isl_map *map,
        enum isl_dim_type type);
const char *isl_map_get_tuple_name(
        __isl_keep isl_map *map,
        enum isl_dim_type type);

#include <isl/val.h>
__isl_give isl_multi_val *isl_multi_val_set_tuple_id(
        __isl_take isl_multi_val *mv,
        enum isl_dim_type type, __isl_take isl_id *id);
__isl_give isl_multi_val *isl_multi_val_reset_tuple_id(
```

```
        __isl_take isl_multi_val *mv,
        enum isl_dim_type type);
isl_bool isl_multi_val_has_tuple_id(
        __isl_keep isl_multi_val *mv,
        enum isl_dim_type type);
__isl_give isl_id *isl_multi_val_get_tuple_id(
        __isl_keep isl_multi_val *mv,
        enum isl_dim_type type);
__isl_give isl_multi_val *isl_multi_val_set_tuple_name(
        __isl_take isl_multi_val *mv,
        enum isl_dim_type type, const char *s);
const char *isl_multi_val_get_tuple_name(
        __isl_keep isl_multi_val *mv,
        enum isl_dim_type type);

#include <isl/aff.h>
__isl_give isl_aff *isl_aff_set_tuple_id(
        __isl_take isl_aff *aff,
        enum isl_dim_type type, __isl_take isl_id *id);
__isl_give isl_multi_aff *isl_multi_aff_set_tuple_id(
        __isl_take isl_multi_aff *maff,
        enum isl_dim_type type, __isl_take isl_id *id);
__isl_give isl_pw_aff *isl_pw_aff_set_tuple_id(
        __isl_take isl_pw_aff *pwaff,
        enum isl_dim_type type, __isl_take isl_id *id);
__isl_give isl_pw_multi_aff *isl_pw_multi_aff_set_tuple_id(
        __isl_take isl_pw_multi_aff *pma,
        enum isl_dim_type type, __isl_take isl_id *id);
__isl_give isl_multi_union_pw_aff *
isl_multi_union_pw_aff_set_tuple_id(
        __isl_take isl_multi_union_pw_aff *mupa,
        enum isl_dim_type type, __isl_take isl_id *id);
__isl_give isl_multi_aff *isl_multi_aff_reset_tuple_id(
        __isl_take isl_multi_aff *ma,
        enum isl_dim_type type);
__isl_give isl_pw_aff *isl_pw_aff_reset_tuple_id(
        __isl_take isl_pw_aff *pa,
        enum isl_dim_type type);
__isl_give isl_multi_pw_aff *
isl_multi_pw_aff_reset_tuple_id(
        __isl_take isl_multi_pw_aff *mpa,
        enum isl_dim_type type);
__isl_give isl_pw_multi_aff *
isl_pw_multi_aff_reset_tuple_id(
        __isl_take isl_pw_multi_aff *pma,
        enum isl_dim_type type);
```

```
__isl_give isl_multi_union_pw_aff *
isl_multi_union_pw_aff_reset_tuple_id(
        __isl_take isl_multi_union_pw_aff *mupa,
        enum isl_dim_type type);
isl_bool isl_multi_aff_has_tuple_id(
        __isl_keep isl_multi_aff *ma,
        enum isl_dim_type type);
__isl_give isl_id *isl_multi_aff_get_tuple_id(
        __isl_keep isl_multi_aff *ma,
        enum isl_dim_type type);
isl_bool isl_pw_aff_has_tuple_id(__isl_keep isl_pw_aff *pa,
        enum isl_dim_type type);
__isl_give isl_id *isl_pw_aff_get_tuple_id(
        __isl_keep isl_pw_aff *pa,
        enum isl_dim_type type);
isl_bool isl_pw_multi_aff_has_tuple_id(
        __isl_keep isl_pw_multi_aff *pma,
        enum isl_dim_type type);
__isl_give isl_id *isl_pw_multi_aff_get_tuple_id(
        __isl_keep isl_pw_multi_aff *pma,
        enum isl_dim_type type);
isl_bool isl_multi_pw_aff_has_tuple_id(
        __isl_keep isl_multi_pw_aff *mpa,
        enum isl_dim_type type);
__isl_give isl_id *isl_multi_pw_aff_get_tuple_id(
        __isl_keep isl_multi_pw_aff *mpa,
        enum isl_dim_type type);
isl_bool isl_multi_union_pw_aff_has_tuple_id(
        __isl_keep isl_multi_union_pw_aff *mupa,
        enum isl_dim_type type);
__isl_give isl_id *isl_multi_union_pw_aff_get_tuple_id(
        __isl_keep isl_multi_union_pw_aff *mupa,
        enum isl_dim_type type);
__isl_give isl_multi_aff *isl_multi_aff_set_tuple_name(
        __isl_take isl_multi_aff *maff,
        enum isl_dim_type type, const char *s);
__isl_give isl_multi_pw_aff *
isl_multi_pw_aff_set_tuple_name(
        __isl_take isl_multi_pw_aff *mpa,
        enum isl_dim_type type, const char *s);
__isl_give isl_multi_union_pw_aff *
isl_multi_union_pw_aff_set_tuple_name(
        __isl_take isl_multi_union_pw_aff *mupa,
        enum isl_dim_type type, const char *s);
const char *isl_multi_aff_get_tuple_name(
        __isl_keep isl_multi_aff *multi,
```

```
        enum isl_dim_type type);
isl_bool isl_pw_multi_aff_has_tuple_name(
        __isl_keep isl_pw_multi_aff *pma,
        enum isl_dim_type type);
const char *isl_pw_multi_aff_get_tuple_name(
        __isl_keep isl_pw_multi_aff *pma,
        enum isl_dim_type type);
const char *isl_multi_union_pw_aff_get_tuple_name(
        __isl_keep isl_multi_union_pw_aff *mupa,
        enum isl_dim_type type);
```

The type argument needs to be one of isl_dim_in, isl_dim_out or isl_dim_set. As with isl_space_get_name, the isl_space_get_tuple_name function returns a pointer to some internal data structure. Binary operations require the corresponding spaces of their arguments to have the same name.

To keep the names of all parameters and tuples, but reset the user pointers of all the corresponding identifiers, use the following function.

```
#include <isl/space.h>
__isl_give isl_space *isl_space_reset_user(
        __isl_take isl_space *space);

#include <isl/set.h>
__isl_give isl_set *isl_set_reset_user(
        __isl_take isl_set *set);

#include <isl/map.h>
__isl_give isl_map *isl_map_reset_user(
        __isl_take isl_map *map);

#include <isl/union_set.h>
__isl_give isl_union_set *isl_union_set_reset_user(
        __isl_take isl_union_set *uset);

#include <isl/union_map.h>
__isl_give isl_union_map *isl_union_map_reset_user(
        __isl_take isl_union_map *umap);

#include <isl/val.h>
__isl_give isl_multi_val *isl_multi_val_reset_user(
        __isl_take isl_multi_val *mv);

#include <isl/aff.h>
__isl_give isl_multi_aff *isl_multi_aff_reset_user(
        __isl_take isl_multi_aff *ma);
__isl_give isl_pw_aff *isl_pw_aff_reset_user(
        __isl_take isl_pw_aff *pa);
```

```
__isl_give isl_multi_pw_aff *isl_multi_pw_aff_reset_user(
        __isl_take isl_multi_pw_aff *mpa);
__isl_give isl_pw_multi_aff *isl_pw_multi_aff_reset_user(
        __isl_take isl_pw_multi_aff *pma);
__isl_give isl_union_pw_aff *isl_union_pw_aff_reset_user(
        __isl_take isl_union_pw_aff *upa);
__isl_give isl_multi_union_pw_aff *
isl_multi_union_pw_aff_reset_user(
        __isl_take isl_multi_union_pw_aff *mupa);
__isl_give isl_union_pw_multi_aff *
isl_union_pw_multi_aff_reset_user(
        __isl_take isl_union_pw_multi_aff *upma);

#include <isl/polynomial.h>
__isl_give isl_pw_qpolynomial *
isl_pw_qpolynomial_reset_user(
        __isl_take isl_pw_qpolynomial *pwqp);
__isl_give isl_union_pw_qpolynomial *
isl_union_pw_qpolynomial_reset_user(
        __isl_take isl_union_pw_qpolynomial *upwqp);
__isl_give isl_pw_qpolynomial_fold *
isl_pw_qpolynomial_fold_reset_user(
        __isl_take isl_pw_qpolynomial_fold *pwf);
__isl_give isl_union_pw_qpolynomial_fold *
isl_union_pw_qpolynomial_fold_reset_user(
        __isl_take isl_union_pw_qpolynomial_fold *upwf);
```

Spaces can be nested. In particular, the domain of a set or the domain or range of a relation can be a nested relation. This process is also called *wrapping*. The functions for detecting, constructing and deconstructing such nested spaces can be found in the wrapping properties of §1.4.15, the wrapping operations of §1.4.16 and the Cartesian product operations of §1.4.17.

Spaces can be created from other spaces using the functions described in §1.4.16 and §1.4.17.

1.4.9 Local Spaces

A local space is essentially a space with zero or more existentially quantified variables. The local space of various objects can be obtained using the following functions.

```
#include <isl/constraint.h>
__isl_give isl_local_space *isl_constraint_get_local_space(
        __isl_keep isl_constraint *constraint);

#include <isl/set.h>
__isl_give isl_local_space *isl_basic_set_get_local_space(
        __isl_keep isl_basic_set *bset);
```

```
#include <isl/map.h>
__isl_give isl_local_space *isl_basic_map_get_local_space(
        __isl_keep isl_basic_map *bmap);

#include <isl/aff.h>
__isl_give isl_local_space *isl_aff_get_domain_local_space(
        __isl_keep isl_aff *aff);
__isl_give isl_local_space *isl_aff_get_local_space(
        __isl_keep isl_aff *aff);
```

A new local space can be created from a space using

```
#include <isl/local_space.h>
__isl_give isl_local_space *isl_local_space_from_space(
        __isl_take isl_space *space);
```

They can be inspected, modified, copied and freed using the following functions.

```
#include <isl/local_space.h>
isl_bool isl_local_space_is_params(
        __isl_keep isl_local_space *ls);
isl_bool isl_local_space_is_set(
        __isl_keep isl_local_space *ls);
__isl_give isl_space *isl_local_space_get_space(
        __isl_keep isl_local_space *ls);
__isl_give isl_aff *isl_local_space_get_div(
        __isl_keep isl_local_space *ls, int pos);
__isl_give isl_local_space *isl_local_space_copy(
        __isl_keep isl_local_space *ls);
__isl_null isl_local_space *isl_local_space_free(
        __isl_take isl_local_space *ls);
```

Note that `isl_local_space_get_div` can only be used on local spaces of sets. Two local spaces can be compared using

```
isl_bool isl_local_space_is_equal(
        __isl_keep isl_local_space *ls1,
        __isl_keep isl_local_space *ls2);
```

Local spaces can be created from other local spaces using the functions described in §1.4.16 and §1.4.17.

1.4.10 Creating New Sets and Relations

`isl` has functions for creating some standard sets and relations.

- Empty sets and relations

```
__isl_give isl_basic_set *isl_basic_set_empty(
        __isl_take isl_space *space);
__isl_give isl_basic_map *isl_basic_map_empty(
        __isl_take isl_space *space);
__isl_give isl_set *isl_set_empty(
        __isl_take isl_space *space);
__isl_give isl_map *isl_map_empty(
        __isl_take isl_space *space);
__isl_give isl_union_set *isl_union_set_empty(
        __isl_take isl_space *space);
__isl_give isl_union_map *isl_union_map_empty(
        __isl_take isl_space *space);
```

For `isl_union_set`s and `isl_union_map`s, the space is only used to specify the parameters.

- Universe sets and relations

```
__isl_give isl_basic_set *isl_basic_set_universe(
        __isl_take isl_space *space);
__isl_give isl_basic_map *isl_basic_map_universe(
        __isl_take isl_space *space);
__isl_give isl_set *isl_set_universe(
        __isl_take isl_space *space);
__isl_give isl_map *isl_map_universe(
        __isl_take isl_space *space);
__isl_give isl_union_set *isl_union_set_universe(
        __isl_take isl_union_set *uset);
__isl_give isl_union_map *isl_union_map_universe(
        __isl_take isl_union_map *umap);
```

The sets and relations constructed by the functions above contain all integer values, while those constructed by the functions below only contain non-negative values.

```
__isl_give isl_basic_set *isl_basic_set_nat_universe(
        __isl_take isl_space *space);
__isl_give isl_basic_map *isl_basic_map_nat_universe(
        __isl_take isl_space *space);
__isl_give isl_set *isl_set_nat_universe(
        __isl_take isl_space *space);
__isl_give isl_map *isl_map_nat_universe(
        __isl_take isl_space *space);
```

- Identity relations

```
__isl_give isl_basic_map *isl_basic_map_identity(
        __isl_take isl_space *space);
__isl_give isl_map *isl_map_identity(
        __isl_take isl_space *space);
```

The number of input and output dimensions in `space` needs to be the same.

- Lexicographic order

```
__isl_give isl_map *isl_map_lex_lt(
        __isl_take isl_space *set_space);
__isl_give isl_map *isl_map_lex_le(
        __isl_take isl_space *set_space);
__isl_give isl_map *isl_map_lex_gt(
        __isl_take isl_space *set_space);
__isl_give isl_map *isl_map_lex_ge(
        __isl_take isl_space *set_space);
__isl_give isl_map *isl_map_lex_lt_first(
        __isl_take isl_space *space, unsigned n);
__isl_give isl_map *isl_map_lex_le_first(
        __isl_take isl_space *space, unsigned n);
__isl_give isl_map *isl_map_lex_gt_first(
        __isl_take isl_space *space, unsigned n);
__isl_give isl_map *isl_map_lex_ge_first(
        __isl_take isl_space *space, unsigned n);
```

The first four functions take a space for a **set** and return relations that express that the elements in the domain are lexicographically less (`isl_map_lex_lt`), less or equal (`isl_map_lex_le`), greater (`isl_map_lex_gt`) or greater or equal (`isl_map_lex_ge`) than the elements in the range. The last four functions take a space for a map and return relations that express that the first `n` dimensions in the domain are lexicographically less (`isl_map_lex_lt_first`), less or equal (`isl_map_lex_le_first`), greater (`isl_map_lex_gt_first`) or greater or equal (`isl_map_lex_ge_first`) than the first `n` dimensions in the range.

A basic set or relation can be converted to a set or relation using the following functions.

```
__isl_give isl_set *isl_set_from_basic_set(
        __isl_take isl_basic_set *bset);
__isl_give isl_map *isl_map_from_basic_map(
        __isl_take isl_basic_map *bmap);
```

Sets and relations can be converted to union sets and relations using the following functions.

```
__isl_give isl_union_set *isl_union_set_from_basic_set(
        __isl_take isl_basic_set *bset);
__isl_give isl_union_map *isl_union_map_from_basic_map(
        __isl_take isl_basic_map *bmap);
__isl_give isl_union_set *isl_union_set_from_set(
        __isl_take isl_set *set);
__isl_give isl_union_map *isl_union_map_from_map(
        __isl_take isl_map *map);
```

The inverse conversions below can only be used if the input union set or relation is known to contain elements in exactly one space.

```
__isl_give isl_set *isl_set_from_union_set(
        __isl_take isl_union_set *uset);
__isl_give isl_map *isl_map_from_union_map(
        __isl_take isl_union_map *umap);
```

Sets and relations can be copied and freed again using the following functions.

```
__isl_give isl_basic_set *isl_basic_set_copy(
        __isl_keep isl_basic_set *bset);
__isl_give isl_set *isl_set_copy(__isl_keep isl_set *set);
__isl_give isl_union_set *isl_union_set_copy(
        __isl_keep isl_union_set *uset);
__isl_give isl_basic_map *isl_basic_map_copy(
        __isl_keep isl_basic_map *bmap);
__isl_give isl_map *isl_map_copy(__isl_keep isl_map *map);
__isl_give isl_union_map *isl_union_map_copy(
        __isl_keep isl_union_map *umap);
__isl_null isl_basic_set *isl_basic_set_free(
        __isl_take isl_basic_set *bset);
__isl_null isl_set *isl_set_free(__isl_take isl_set *set);
__isl_null isl_union_set *isl_union_set_free(
        __isl_take isl_union_set *uset);
__isl_null isl_basic_map *isl_basic_map_free(
        __isl_take isl_basic_map *bmap);
__isl_null isl_map *isl_map_free(__isl_take isl_map *map);
__isl_null isl_union_map *isl_union_map_free(
        __isl_take isl_union_map *umap);
```

Other sets and relations can be constructed by starting from a universe set or relation, adding equality and/or inequality constraints and then projecting out the existentially quantified variables, if any. Constraints can be constructed, manipulated and added to (or removed from) (basic) sets and relations using the following functions.

```
#include <isl/constraint.h>
__isl_give isl_constraint *isl_constraint_alloc_equality(
```

```
        __isl_take isl_local_space *ls);
__isl_give isl_constraint *isl_constraint_alloc_inequality(
        __isl_take isl_local_space *ls);
__isl_give isl_constraint *isl_constraint_set_constant_si(
        __isl_take isl_constraint *constraint, int v);
__isl_give isl_constraint *isl_constraint_set_constant_val(
        __isl_take isl_constraint *constraint,
        __isl_take isl_val *v);
__isl_give isl_constraint *isl_constraint_set_coefficient_si(
        __isl_take isl_constraint *constraint,
        enum isl_dim_type type, int pos, int v);
__isl_give isl_constraint *
isl_constraint_set_coefficient_val(
        __isl_take isl_constraint *constraint,
        enum isl_dim_type type, int pos,
        __isl_take isl_val *v);
__isl_give isl_basic_map *isl_basic_map_add_constraint(
        __isl_take isl_basic_map *bmap,
        __isl_take isl_constraint *constraint);
__isl_give isl_basic_set *isl_basic_set_add_constraint(
        __isl_take isl_basic_set *bset,
        __isl_take isl_constraint *constraint);
__isl_give isl_map *isl_map_add_constraint(
        __isl_take isl_map *map,
        __isl_take isl_constraint *constraint);
__isl_give isl_set *isl_set_add_constraint(
        __isl_take isl_set *set,
        __isl_take isl_constraint *constraint);
```

For example, to create a set containing the even integers between 10 and 42, you would use the following code.

```
isl_space *space;
isl_local_space *ls;
isl_constraint *c;
isl_basic_set *bset;

space = isl_space_set_alloc(ctx, 0, 2);
bset = isl_basic_set_universe(isl_space_copy(space));
ls = isl_local_space_from_space(space);

c = isl_constraint_alloc_equality(isl_local_space_copy(ls));
c = isl_constraint_set_coefficient_si(c, isl_dim_set, 0, -1);
c = isl_constraint_set_coefficient_si(c, isl_dim_set, 1, 2);
bset = isl_basic_set_add_constraint(bset, c);

c = isl_constraint_alloc_inequality(isl_local_space_copy(ls));
```

```
c = isl_constraint_set_constant_si(c, -10);
c = isl_constraint_set_coefficient_si(c, isl_dim_set, 0, 1);
bset = isl_basic_set_add_constraint(bset, c);

c = isl_constraint_alloc_inequality(ls);
c = isl_constraint_set_constant_si(c, 42);
c = isl_constraint_set_coefficient_si(c, isl_dim_set, 0, -1);
bset = isl_basic_set_add_constraint(bset, c);

bset = isl_basic_set_project_out(bset, isl_dim_set, 1, 1);
```

Or, alternatively,

```
isl_basic_set *bset;
bset = isl_basic_set_read_from_str(ctx,
        "{[i] : exists (a : i = 2a and i >= 10 and i <= 42)}");
```

A basic set or relation can also be constructed from two matrices describing the equalities and the inequalities.

```
__isl_give isl_basic_set *isl_basic_set_from_constraint_matrices(
        __isl_take isl_space *space,
        __isl_take isl_mat *eq, __isl_take isl_mat *ineq,
        enum isl_dim_type c1,
        enum isl_dim_type c2, enum isl_dim_type c3,
        enum isl_dim_type c4);
__isl_give isl_basic_map *isl_basic_map_from_constraint_matrices(
        __isl_take isl_space *space,
        __isl_take isl_mat *eq, __isl_take isl_mat *ineq,
        enum isl_dim_type c1,
        enum isl_dim_type c2, enum isl_dim_type c3,
        enum isl_dim_type c4, enum isl_dim_type c5);
```

The `isl_dim_type` arguments indicate the order in which different kinds of variables appear in the input matrices and should be a permutation of `isl_dim_cst`, `isl_dim_param`, `isl_dim_set` and `isl_dim_div` for sets and of `isl_dim_cst`, `isl_dim_param`, `isl_dim_in`, `isl_dim_out` and `isl_dim_div` for relations.

A (basic or union) set or relation can also be constructed from a (union) (piecewise) (multiple) affine expression or a list of affine expressions (See §1.4.13), provided these affine expressions do not involve any NaN.

```
__isl_give isl_basic_map *isl_basic_map_from_aff(
        __isl_take isl_aff *aff);
__isl_give isl_map *isl_map_from_aff(
        __isl_take isl_aff *aff);
__isl_give isl_set *isl_set_from_pw_aff(
        __isl_take isl_pw_aff *pwaff);
```

```
__isl_give isl_map *isl_map_from_pw_aff(
        __isl_take isl_pw_aff *pwaff);
__isl_give isl_basic_map *isl_basic_map_from_aff_list(
        __isl_take isl_space *domain_space,
        __isl_take isl_aff_list *list);
__isl_give isl_basic_map *isl_basic_map_from_multi_aff(
        __isl_take isl_multi_aff *maff)
__isl_give isl_map *isl_map_from_multi_aff(
        __isl_take isl_multi_aff *maff)
__isl_give isl_set *isl_set_from_pw_multi_aff(
        __isl_take isl_pw_multi_aff *pma);
__isl_give isl_map *isl_map_from_pw_multi_aff(
        __isl_take isl_pw_multi_aff *pma);
__isl_give isl_set *isl_set_from_multi_pw_aff(
        __isl_take isl_multi_pw_aff *mpa);
__isl_give isl_map *isl_map_from_multi_pw_aff(
        __isl_take isl_multi_pw_aff *mpa);
__isl_give isl_union_map *isl_union_map_from_union_pw_aff(
        __isl_take isl_union_pw_aff *upa);
__isl_give isl_union_map *
isl_union_map_from_union_pw_multi_aff(
        __isl_take isl_union_pw_multi_aff *upma);
__isl_give isl_union_map *
isl_union_map_from_multi_union_pw_aff(
        __isl_take isl_multi_union_pw_aff *mupa);
```

The `domain_space` argument describes the domain of the resulting basic relation. It is required because the `list` may consist of zero affine expressions. The `mupa` passed to `isl_union_map_from_multi_union_pw_aff` is not allowed to be zero-dimensional. The domain of the result is the shared domain of the union piecewise affine elements.

1.4.11 Inspecting Sets and Relations

Usually, the user should not have to care about the actual constraints of the sets and maps, but should instead apply the abstract operations explained in the following sections. Occasionally, however, it may be required to inspect the individual coefficients of the constraints. This section explains how to do so. In these cases, it may also be useful to have `isl` compute an explicit representation of the existentially quantified variables.

```
__isl_give isl_set *isl_set_compute_divs(
        __isl_take isl_set *set);
__isl_give isl_map *isl_map_compute_divs(
        __isl_take isl_map *map);
__isl_give isl_union_set *isl_union_set_compute_divs(
```

```
        __isl_take isl_union_set *uset);
__isl_give isl_union_map *isl_union_map_compute_divs(
        __isl_take isl_union_map *umap);
```

This explicit representation defines the existentially quantified variables as integer divisions of the other variables, possibly including earlier existentially quantified variables. An explicitly represented existentially quantified variable therefore has a unique value when the values of the other variables are known. If, furthermore, the same existentials, i.e., existentials with the same explicit representations, should appear in the same order in each of the disjuncts of a set or map, then the user should call either of the following functions.

```
__isl_give isl_set *isl_set_align_divs(
        __isl_take isl_set *set);
__isl_give isl_map *isl_map_align_divs(
        __isl_take isl_map *map);
```

Alternatively, the existentially quantified variables can be removed using the following functions, which compute an overapproximation.

```
__isl_give isl_basic_set *isl_basic_set_remove_divs(
        __isl_take isl_basic_set *bset);
__isl_give isl_basic_map *isl_basic_map_remove_divs(
        __isl_take isl_basic_map *bmap);
__isl_give isl_set *isl_set_remove_divs(
        __isl_take isl_set *set);
__isl_give isl_map *isl_map_remove_divs(
        __isl_take isl_map *map);
```

It is also possible to only remove those divs that are defined in terms of a given range of dimensions or only those for which no explicit representation is known.

```
__isl_give isl_basic_set *
isl_basic_set_remove_divs_involving_dims(
        __isl_take isl_basic_set *bset,
        enum isl_dim_type type,
        unsigned first, unsigned n);
__isl_give isl_basic_map *
isl_basic_map_remove_divs_involving_dims(
        __isl_take isl_basic_map *bmap,
        enum isl_dim_type type,
        unsigned first, unsigned n);
__isl_give isl_set *isl_set_remove_divs_involving_dims(
        __isl_take isl_set *set, enum isl_dim_type type,
        unsigned first, unsigned n);
__isl_give isl_map *isl_map_remove_divs_involving_dims(
        __isl_take isl_map *map, enum isl_dim_type type,
        unsigned first, unsigned n);
```

```
__isl_give isl_basic_set *
isl_basic_set_remove_unknown_divs(
        __isl_take isl_basic_set *bset);
__isl_give isl_set *isl_set_remove_unknown_divs(
        __isl_take isl_set *set);
__isl_give isl_map *isl_map_remove_unknown_divs(
        __isl_take isl_map *map);
```

To iterate over all the sets or maps in a union set or map, use

```
isl_stat isl_union_set_foreach_set(
        __isl_keep isl_union_set *uset,
        isl_stat (*fn)(__isl_take isl_set *set, void *user),
        void *user);
isl_stat isl_union_map_foreach_map(
        __isl_keep isl_union_map *umap,
        isl_stat (*fn)(__isl_take isl_map *map, void *user),
        void *user);
```

These functions call the callback function once for each (pair of) space(s) for which there are elements in the input. The argument to the callback contains all elements in the input with that (pair of) space(s).

The number of sets or maps in a union set or map can be obtained from

```
int isl_union_set_n_set(__isl_keep isl_union_set *uset);
int isl_union_map_n_map(__isl_keep isl_union_map *umap);
```

To extract the set or map in a given space from a union, use

```
__isl_give isl_set *isl_union_set_extract_set(
        __isl_keep isl_union_set *uset,
        __isl_take isl_space *space);
__isl_give isl_map *isl_union_map_extract_map(
        __isl_keep isl_union_map *umap,
        __isl_take isl_space *space);
```

To iterate over all the basic sets or maps in a set or map, use

```
isl_stat isl_set_foreach_basic_set(__isl_keep isl_set *set,
        isl_stat (*fn)(__isl_take isl_basic_set *bset,
                void *user),
        void *user);
isl_stat isl_map_foreach_basic_map(__isl_keep isl_map *map,
        isl_stat (*fn)(__isl_take isl_basic_map *bmap,
                void *user),
        void *user);
```

The callback function `fn` should return 0 if successful and -1 if an error occurs. In the latter case, or if any other error occurs, the above functions will return -1.

It should be noted that `isl` does not guarantee that the basic sets or maps passed to `fn` are disjoint. If this is required, then the user should call one of the following functions first.

```
__isl_give isl_set *isl_set_make_disjoint(
        __isl_take isl_set *set);
__isl_give isl_map *isl_map_make_disjoint(
        __isl_take isl_map *map);
```

The number of basic sets in a set can be obtained or the number of basic maps in a map can be obtained from

```
#include <isl/set.h>
int isl_set_n_basic_set(__isl_keep isl_set *set);

#include <isl/map.h>
int isl_map_n_basic_map(__isl_keep isl_map *map);
```

It is also possible to obtain a list of basic sets from a set

```
#include <isl/set.h>
__isl_give isl_basic_set_list *isl_set_get_basic_set_list(
        __isl_keep isl_set *set);
```

The returned list can be manipulated using the functions in §1.4.19.

To iterate over the constraints of a basic set or map, use

```
#include <isl/constraint.h>

int isl_basic_set_n_constraint(
        __isl_keep isl_basic_set *bset);
isl_stat isl_basic_set_foreach_constraint(
        __isl_keep isl_basic_set *bset,
        isl_stat (*fn)(__isl_take isl_constraint *c,
                void *user),
        void *user);
int isl_basic_map_n_constraint(
        __isl_keep isl_basic_map *bmap);
isl_stat isl_basic_map_foreach_constraint(
        __isl_keep isl_basic_map *bmap,
        isl_stat (*fn)(__isl_take isl_constraint *c,
                void *user),
        void *user);
__isl_null isl_constraint *isl_constraint_free(
        __isl_take isl_constraint *c);
```

Again, the callback function `fn` should return 0 if successful and -1 if an error occurs. In the latter case, or if any other error occurs, the above functions will return -1. The constraint `c` represents either an equality or an inequality. Use the following function to find out whether a constraint represents an equality. If not, it represents an inequality.

```
isl_bool isl_constraint_is_equality(
        __isl_keep isl_constraint *constraint);
```

It is also possible to obtain a list of constraints from a basic map or set

```
#include <isl/constraint.h>
__isl_give isl_constraint_list *
isl_basic_map_get_constraint_list(
        __isl_keep isl_basic_map *bmap);
__isl_give isl_constraint_list *
isl_basic_set_get_constraint_list(
        __isl_keep isl_basic_set *bset);
```

These functions require that all existentially quantified variables have an explicit representation. The returned list can be manipulated using the functions in §1.4.19.

The coefficients of the constraints can be inspected using the following functions.

```
isl_bool isl_constraint_is_lower_bound(
        __isl_keep isl_constraint *constraint,
        enum isl_dim_type type, unsigned pos);
isl_bool isl_constraint_is_upper_bound(
        __isl_keep isl_constraint *constraint,
        enum isl_dim_type type, unsigned pos);
__isl_give isl_val *isl_constraint_get_constant_val(
        __isl_keep isl_constraint *constraint);
__isl_give isl_val *isl_constraint_get_coefficient_val(
        __isl_keep isl_constraint *constraint,
        enum isl_dim_type type, int pos);
```

The explicit representations of the existentially quantified variables can be inspected using the following function. Note that the user is only allowed to use this function if the inspected set or map is the result of a call to `isl_set_compute_divs` or `isl_map_compute_divs`. The existentially quantified variable is equal to the floor of the returned affine expression. The affine expression itself can be inspected using the functions in §1.4.13.

```
__isl_give isl_aff *isl_constraint_get_div(
        __isl_keep isl_constraint *constraint, int pos);
```

To obtain the constraints of a basic set or map in matrix form, use the following functions.

```
__isl_give isl_mat *isl_basic_set_equalities_matrix(
        __isl_keep isl_basic_set *bset,
        enum isl_dim_type c1, enum isl_dim_type c2,
        enum isl_dim_type c3, enum isl_dim_type c4);
__isl_give isl_mat *isl_basic_set_inequalities_matrix(
        __isl_keep isl_basic_set *bset,
        enum isl_dim_type c1, enum isl_dim_type c2,
        enum isl_dim_type c3, enum isl_dim_type c4);
__isl_give isl_mat *isl_basic_map_equalities_matrix(
        __isl_keep isl_basic_map *bmap,
        enum isl_dim_type c1,
        enum isl_dim_type c2, enum isl_dim_type c3,
        enum isl_dim_type c4, enum isl_dim_type c5);
__isl_give isl_mat *isl_basic_map_inequalities_matrix(
        __isl_keep isl_basic_map *bmap,
        enum isl_dim_type c1,
        enum isl_dim_type c2, enum isl_dim_type c3,
        enum isl_dim_type c4, enum isl_dim_type c5);
```

The `isl_dim_type` arguments dictate the order in which different kinds of variables appear in the resulting matrix. For set inputs, they should be a permutation of `isl_dim_cst`, `isl_dim_param`, `isl_dim_set` and `isl_dim_div`. For map inputs, they should be a permutation of `isl_dim_cst`, `isl_dim_param`, `isl_dim_in`, `isl_dim_out` and `isl_dim_div`.

1.4.12 Points

Points are elements of a set. They can be used to construct simple sets (boxes) or they can be used to represent the individual elements of a set. The zero point (the origin) can be created using

```
__isl_give isl_point *isl_point_zero(__isl_take isl_space *space);
```

The coordinates of a point can be inspected, set and changed using

```
__isl_give isl_val *isl_point_get_coordinate_val(
        __isl_keep isl_point *pnt,
        enum isl_dim_type type, int pos);
__isl_give isl_point *isl_point_set_coordinate_val(
        __isl_take isl_point *pnt,
        enum isl_dim_type type, int pos,
        __isl_take isl_val *v);

__isl_give isl_point *isl_point_add_ui(
        __isl_take isl_point *pnt,
        enum isl_dim_type type, int pos, unsigned val);
__isl_give isl_point *isl_point_sub_ui(
```

```
        __isl_take isl_point *pnt,
        enum isl_dim_type type, int pos, unsigned val);
```

Points can be copied or freed using

```
__isl_give isl_point *isl_point_copy(
        __isl_keep isl_point *pnt);
void isl_point_free(__isl_take isl_point *pnt);
```

A singleton set can be created from a point using

```
__isl_give isl_basic_set *isl_basic_set_from_point(
        __isl_take isl_point *pnt);
__isl_give isl_set *isl_set_from_point(
        __isl_take isl_point *pnt);
__isl_give isl_union_set *isl_union_set_from_point(
        __isl_take isl_point *pnt);
```

and a box can be created from two opposite extremal points using

```
__isl_give isl_basic_set *isl_basic_set_box_from_points(
        __isl_take isl_point *pnt1,
        __isl_take isl_point *pnt2);
__isl_give isl_set *isl_set_box_from_points(
        __isl_take isl_point *pnt1,
        __isl_take isl_point *pnt2);
```

All elements of a **bounded** (union) set can be enumerated using the following functions.

```
isl_stat isl_set_foreach_point(__isl_keep isl_set *set,
        isl_stat (*fn)(__isl_take isl_point *pnt,
                void *user),
        void *user);
isl_stat isl_union_set_foreach_point(
        __isl_keep isl_union_set *uset,
        isl_stat (*fn)(__isl_take isl_point *pnt,
                void *user),
        void *user);
```

The function `fn` is called for each integer point in `set` with as second argument the last argument of the `isl_set_foreach_point` call. The function `fn` should return `0` on success and `-1` on failure. In the latter case, `isl_set_foreach_point` will stop enumerating and return `-1` as well. If the enumeration is performed successfully and to completion, then `isl_set_foreach_point` returns `0`.

To obtain a single point of a (basic or union) set, use

```
__isl_give isl_point *isl_basic_set_sample_point(
        __isl_take isl_basic_set *bset);
__isl_give isl_point *isl_set_sample_point(
        __isl_take isl_set *set);
__isl_give isl_point *isl_union_set_sample_point(
        __isl_take isl_union_set *uset);
```

If `set` does not contain any (integer) points, then the resulting point will be "void", a property that can be tested using

```
isl_bool isl_point_is_void(__isl_keep isl_point *pnt);
```

1.4.13 Functions

Besides sets and relation, `isl` also supports various types of functions. Each of these types is derived from the value type (see §1.4.4) or from one of two primitive function types through the application of zero or more type constructors. We first describe the primitive type and then we describe the types derived from these primitive types.

Primitive Functions

`isl` support two primitive function types, quasi-affine expressions and quasipolynomials. A quasi-affine expression is defined either over a parameter space or over a set and is composed of integer constants, parameters and set variables, addition, subtraction and integer division by an integer constant. For example, the quasi-affine expression

```
[n] -> { [x] -> [2*floor((4 n + x)/9] }
```

maps `x` to `2*floor((4 n + x)/9`. A quasipolynomial is a polynomial expression in quasi-affine expression. That is, it additionally allows for multiplication. Note, though, that it is not allowed to construct an integer division of an expression involving multiplications. Here is an example of a quasipolynomial that is not quasi-affine expression

```
[n] -> { [x] -> (n*floor((4 n + x)/9) }
```

Note that the external representations of quasi-affine expressions and quasipolynomials are different. Quasi-affine expressions use a notation with square brackets just like binary relations, while quasipolynomials do not. This might change at some point.

If a primitive function is defined over a parameter space, then the space of the function itself is that of a set. If it is defined over a set, then the space of the function is that of a relation. In both cases, the set space (or the output space) is single-dimensional, anonymous and unstructured. To create functions with multiple dimensions or with other kinds of set or output spaces, use multiple expressions (see §1.4.13).

- Quasi-affine Expressions

Besides the expressions described above, a quasi-affine expression can also be set to NaN. Such expressions typically represent a failure to represent a result as a quasi-affine expression.

The zero quasi affine expression or the quasi affine expression that is equal to a given value or a specified dimension on a given domain can be created using

```
#include <isl/aff.h>
__isl_give isl_aff *isl_aff_zero_on_domain(
        __isl_take isl_local_space *ls);
__isl_give isl_aff *isl_aff_val_on_domain(
        __isl_take isl_local_space *ls,
        __isl_take isl_val *val);
__isl_give isl_aff *isl_aff_var_on_domain(
        __isl_take isl_local_space *ls,
        enum isl_dim_type type, unsigned pos);
__isl_give isl_aff *isl_aff_nan_on_domain(
        __isl_take isl_local_space *ls);
```

Quasi affine expressions can be copied and freed using

```
#include <isl/aff.h>
__isl_give isl_aff *isl_aff_copy(
        __isl_keep isl_aff *aff);
__isl_null isl_aff *isl_aff_free(
        __isl_take isl_aff *aff);
```

A (rational) bound on a dimension can be extracted from an `isl_constraint` using the following function. The constraint is required to have a non-zero coefficient for the specified dimension.

```
#include <isl/constraint.h>
__isl_give isl_aff *isl_constraint_get_bound(
        __isl_keep isl_constraint *constraint,
        enum isl_dim_type type, int pos);
```

The entire affine expression of the constraint can also be extracted using the following function.

```
#include <isl/constraint.h>
__isl_give isl_aff *isl_constraint_get_aff(
        __isl_keep isl_constraint *constraint);
```

Conversely, an equality constraint equating the affine expression to zero or an inequality constraint enforcing the affine expression to be non-negative, can be constructed using

```
__isl_give isl_constraint *isl_equality_from_aff(
        __isl_take isl_aff *aff);
__isl_give isl_constraint *isl_inequality_from_aff(
        __isl_take isl_aff *aff);
```

The coefficients and the integer divisions of an affine expression can be inspected using the following functions.

```
#include <isl/aff.h>
__isl_give isl_val *isl_aff_get_constant_val(
        __isl_keep isl_aff *aff);
__isl_give isl_val *isl_aff_get_coefficient_val(
        __isl_keep isl_aff *aff,
        enum isl_dim_type type, int pos);
int isl_aff_coefficient_sgn(__isl_keep isl_aff *aff,
        enum isl_dim_type type, int pos);
__isl_give isl_val *isl_aff_get_denominator_val(
        __isl_keep isl_aff *aff);
__isl_give isl_aff *isl_aff_get_div(
        __isl_keep isl_aff *aff, int pos);
```

They can be modified using the following functions.

```
#include <isl/aff.h>
__isl_give isl_aff *isl_aff_set_constant_si(
        __isl_take isl_aff *aff, int v);
__isl_give isl_aff *isl_aff_set_constant_val(
        __isl_take isl_aff *aff, __isl_take isl_val *v);
__isl_give isl_aff *isl_aff_set_coefficient_si(
        __isl_take isl_aff *aff,
        enum isl_dim_type type, int pos, int v);
__isl_give isl_aff *isl_aff_set_coefficient_val(
        __isl_take isl_aff *aff,
        enum isl_dim_type type, int pos,
        __isl_take isl_val *v);

__isl_give isl_aff *isl_aff_add_constant_si(
        __isl_take isl_aff *aff, int v);
__isl_give isl_aff *isl_aff_add_constant_val(
        __isl_take isl_aff *aff, __isl_take isl_val *v);
__isl_give isl_aff *isl_aff_add_constant_num_si(
        __isl_take isl_aff *aff, int v);
__isl_give isl_aff *isl_aff_add_coefficient_si(
        __isl_take isl_aff *aff,
        enum isl_dim_type type, int pos, int v);
```

```
__isl_give isl_aff *isl_aff_add_coefficient_val(
        __isl_take isl_aff *aff,
        enum isl_dim_type type, int pos,
        __isl_take isl_val *v);
```

Note that `isl_aff_set_constant_si` and `isl_aff_set_coefficient_si` set the *numerator* of the constant or coefficient, while `isl_aff_set_constant_val` and `isl_aff_set_coefficient_val` set the constant or coefficient as a whole. The `add_constant` and `add_coefficient` functions add an integer or rational value to the possibly rational constant or coefficient. The `add_constant_num` functions add an integer value to the numerator.

- Quasipolynomials

  Some simple quasipolynomials can be created using the following functions.

```
#include <isl/polynomial.h>
__isl_give isl_qpolynomial *isl_qpolynomial_zero_on_domain(
        __isl_take isl_space *domain);
__isl_give isl_qpolynomial *isl_qpolynomial_one_on_domain(
        __isl_take isl_space *domain);
__isl_give isl_qpolynomial *isl_qpolynomial_infty_on_domain(
        __isl_take isl_space *domain);
__isl_give isl_qpolynomial *isl_qpolynomial_neginfty_on_domain(
        __isl_take isl_space *domain);
__isl_give isl_qpolynomial *isl_qpolynomial_nan_on_domain(
        __isl_take isl_space *domain);
__isl_give isl_qpolynomial *isl_qpolynomial_val_on_domain(
        __isl_take isl_space *domain,
        __isl_take isl_val *val);
__isl_give isl_qpolynomial *isl_qpolynomial_var_on_domain(
        __isl_take isl_space *domain,
        enum isl_dim_type type, unsigned pos);
__isl_give isl_qpolynomial *isl_qpolynomial_from_aff(
        __isl_take isl_aff *aff);
```

  Recall that the space in which a quasipolynomial lives is a map space with a one-dimensional range. The `domain` argument in some of the functions above corresponds to the domain of this map space.

  Quasipolynomials can be copied and freed again using the following functions.

```
#include <isl/polynomial.h>
__isl_give isl_qpolynomial *isl_qpolynomial_copy(
        __isl_keep isl_qpolynomial *qp);
__isl_null isl_qpolynomial *isl_qpolynomial_free(
        __isl_take isl_qpolynomial *qp);
```

The constant term of a quasipolynomial can be extracted using

```
__isl_give isl_val *isl_qpolynomial_get_constant_val(
        __isl_keep isl_qpolynomial *qp);
```

To iterate over all terms in a quasipolynomial, use

```
isl_stat isl_qpolynomial_foreach_term(
        __isl_keep isl_qpolynomial *qp,
        isl_stat (*fn)(__isl_take isl_term *term,
                  void *user), void *user);
```

The terms themselves can be inspected and freed using these functions

```
unsigned isl_term_dim(__isl_keep isl_term *term,
        enum isl_dim_type type);
__isl_give isl_val *isl_term_get_coefficient_val(
        __isl_keep isl_term *term);
int isl_term_get_exp(__isl_keep isl_term *term,
        enum isl_dim_type type, unsigned pos);
__isl_give isl_aff *isl_term_get_div(
        __isl_keep isl_term *term, unsigned pos);
void isl_term_free(__isl_take isl_term *term);
```

Each term is a product of parameters, set variables and integer divisions. The function `isl_term_get_exp` returns the exponent of a given dimensions in the given term.

Reductions

A reduction represents a maximum or a minimum of its base expressions. The only reduction type defined by `isl` is `isl_qpolynomial_fold`.

There are currently no functions to directly create such objects, but they do appear in the piecewise quasipolynomial reductions returned by the `isl_pw_qpolynomial_bound` function. See §1.4.23.

Reductions can be copied and freed using the following functions.

```
#include <isl/polynomial.h>
__isl_give isl_qpolynomial_fold *
isl_qpolynomial_fold_copy(
        __isl_keep isl_qpolynomial_fold *fold);
void isl_qpolynomial_fold_free(
        __isl_take isl_qpolynomial_fold *fold);
```

To iterate over all quasipolynomials in a reduction, use

```
isl_stat isl_qpolynomial_fold_foreach_qpolynomial(
        __isl_keep isl_qpolynomial_fold *fold,
        isl_stat (*fn)(__isl_take isl_qpolynomial *qp,
                  void *user), void *user);
```

Multiple Expressions

A multiple expression represents a sequence of zero or more base expressions, all defined on the same domain space. The domain space of the multiple expression is the same as that of the base expressions, but the range space can be any space. In case the base expressions have a set space, the corresponding multiple expression also has a set space. Objects of the value type do not have an associated space. The space of a multiple value is therefore always a set space. Similarly, the space of a multiple union piecewise affine expression is always a set space.

The multiple expression types defined by `isl` are `isl_multi_val`, `isl_multi_aff`, `isl_multi_pw_aff`, `isl_multi_union_pw_aff`.

A multiple expression with the value zero for each output (or set) dimension can be created using the following functions.

```
#include <isl/val.h>
__isl_give isl_multi_val *isl_multi_val_zero(
        __isl_take isl_space *space);

#include <isl/aff.h>
__isl_give isl_multi_aff *isl_multi_aff_zero(
        __isl_take isl_space *space);
__isl_give isl_multi_pw_aff *isl_multi_pw_aff_zero(
        __isl_take isl_space *space);
__isl_give isl_multi_union_pw_aff *
isl_multi_union_pw_aff_zero(
        __isl_take isl_space *space);
```

Since there is no canonical way of representing a zero value of type `isl_union_pw_aff`, the space passed to `isl_multi_union_pw_aff_zero` needs to be zero-dimensional.

An identity function can be created using the following functions. The space needs to be that of a relation with the same number of input and output dimensions.

```
#include <isl/aff.h>
__isl_give isl_multi_aff *isl_multi_aff_identity(
        __isl_take isl_space *space);
__isl_give isl_multi_pw_aff *isl_multi_pw_aff_identity(
        __isl_take isl_space *space);
```

A function that performs a projection on a universe relation or set can be created using the following functions. See also the corresponding projection operations in §1.4.16.

```
#include <isl/aff.h>
__isl_give isl_multi_aff *isl_multi_aff_domain_map(
        __isl_take isl_space *space);
__isl_give isl_multi_aff *isl_multi_aff_range_map(
        __isl_take isl_space *space);
```

```
__isl_give isl_multi_aff *isl_multi_aff_project_out_map(
        __isl_take isl_space *space,
        enum isl_dim_type type,
        unsigned first, unsigned n);
```

A multiple expression can be created from a single base expression using the following functions. The space of the created multiple expression is the same as that of the base expression, except for `isl_multi_union_pw_aff_from_union_pw_aff` where the input lives in a parameter space and the output lives in a single-dimensional set space.

```
#include <isl/aff.h>
__isl_give isl_multi_aff *isl_multi_aff_from_aff(
        __isl_take isl_aff *aff);
__isl_give isl_multi_pw_aff *isl_multi_pw_aff_from_pw_aff(
        __isl_take isl_pw_aff *pa);
__isl_give isl_multi_union_pw_aff *
isl_multi_union_pw_aff_from_union_pw_aff(
        __isl_take isl_union_pw_aff *upa);
```

A multiple expression can be created from a list of base expression in a specified space. The domain of this space needs to be the same as the domains of the base expressions in the list. If the base expressions have a set space (or no associated space), then this space also needs to be a set space.

```
#include <isl/val.h>
__isl_give isl_multi_val *isl_multi_val_from_val_list(
        __isl_take isl_space *space,
        __isl_take isl_val_list *list);

#include <isl/aff.h>
__isl_give isl_multi_aff *isl_multi_aff_from_aff_list(
        __isl_take isl_space *space,
        __isl_take isl_aff_list *list);
__isl_give isl_multi_pw_aff *
isl_multi_pw_aff_from_pw_aff_list(
        __isl_take isl_space *space,
        __isl_take isl_pw_aff_list *list);
__isl_give isl_multi_union_pw_aff *
isl_multi_union_pw_aff_from_union_pw_aff_list(
        __isl_take isl_space *space,
        __isl_take isl_union_pw_aff_list *list);
```

As a convenience, a multiple piecewise expression can also be created from a multiple expression. Each piecewise expression in the result has a single universe cell.

```
#include <isl/aff.h>
```

```
__isl_give isl_multi_pw_aff *
isl_multi_pw_aff_from_multi_aff(
        __isl_take isl_multi_aff *ma);
```

Similarly, a multiple union expression can be created from a multiple expression.

```
#include <isl/aff.h>
__isl_give isl_multi_union_pw_aff *
isl_multi_union_pw_aff_from_multi_aff(
        __isl_take isl_multi_aff *ma);
__isl_give isl_multi_union_pw_aff *
isl_multi_union_pw_aff_from_multi_pw_aff(
        __isl_take isl_multi_pw_aff *mpa);
```

A multiple quasi-affine expression can be created from a multiple value with a given domain space using the following function.

```
#include <isl/aff.h>
__isl_give isl_multi_aff *
isl_multi_aff_multi_val_on_space(
        __isl_take isl_space *space,
        __isl_take isl_multi_val *mv);
```

Similarly, a multiple union piecewise affine expression can be created from a multiple value with a given domain or a multiple affine expression with a given domain using the following functions.

```
#include <isl/aff.h>
__isl_give isl_multi_union_pw_aff *
isl_multi_union_pw_aff_multi_val_on_domain(
        __isl_take isl_union_set *domain,
        __isl_take isl_multi_val *mv);
__isl_give isl_multi_union_pw_aff *
isl_multi_union_pw_aff_multi_aff_on_domain(
        __isl_take isl_union_set *domain,
        __isl_take isl_multi_aff *ma);
```

Multiple expressions can be copied and freed using the following functions.

```
#include <isl/val.h>
__isl_give isl_multi_val *isl_multi_val_copy(
        __isl_keep isl_multi_val *mv);
__isl_null isl_multi_val *isl_multi_val_free(
        __isl_take isl_multi_val *mv);

#include <isl/aff.h>
__isl_give isl_multi_aff *isl_multi_aff_copy(
        __isl_keep isl_multi_aff *maff);
```

```
__isl_null isl_multi_aff *isl_multi_aff_free(
        __isl_take isl_multi_aff *maff);
__isl_give isl_multi_pw_aff *isl_multi_pw_aff_copy(
        __isl_keep isl_multi_pw_aff *mpa);
__isl_null isl_multi_pw_aff *isl_multi_pw_aff_free(
        __isl_take isl_multi_pw_aff *mpa);
__isl_give isl_multi_union_pw_aff *
isl_multi_union_pw_aff_copy(
        __isl_keep isl_multi_union_pw_aff *mupa);
__isl_null isl_multi_union_pw_aff *
isl_multi_union_pw_aff_free(
        __isl_take isl_multi_union_pw_aff *mupa);
```

The base expression at a given position of a multiple expression can be extracted using the following functions.

```
#include <isl/val.h>
__isl_give isl_val *isl_multi_val_get_val(
        __isl_keep isl_multi_val *mv, int pos);

#include <isl/aff.h>
__isl_give isl_aff *isl_multi_aff_get_aff(
        __isl_keep isl_multi_aff *multi, int pos);
__isl_give isl_pw_aff *isl_multi_pw_aff_get_pw_aff(
        __isl_keep isl_multi_pw_aff *mpa, int pos);
__isl_give isl_union_pw_aff *
isl_multi_union_pw_aff_get_union_pw_aff(
        __isl_keep isl_multi_union_pw_aff *mupa, int pos);
```

It can be replaced using the following functions.

```
#include <isl/val.h>
__isl_give isl_multi_val *isl_multi_val_set_val(
        __isl_take isl_multi_val *mv, int pos,
        __isl_take isl_val *val);

#include <isl/aff.h>
__isl_give isl_multi_aff *isl_multi_aff_set_aff(
        __isl_take isl_multi_aff *multi, int pos,
        __isl_take isl_aff *aff);
__isl_give isl_multi_union_pw_aff *
isl_multi_union_pw_aff_set_union_pw_aff(
        __isl_take isl_multi_union_pw_aff *mupa, int pos,
        __isl_take isl_union_pw_aff *upa);
```

As a convenience, a sequence of base expressions that have their domains in a given space can be extracted from a sequence of union expressions using the following function.

```
#include <isl/aff.h>
__isl_give isl_multi_pw_aff *
isl_multi_union_pw_aff_extract_multi_pw_aff(
        __isl_keep isl_multi_union_pw_aff *mupa,
        __isl_take isl_space *space);
```

Note that there is a difference between `isl_multi_union_pw_aff` and `isl_union_pw_multi_aff` objects. The first is a sequence of unions of piecewise expressions, while the second is a union of piecewise sequences. In particular, multiple affine expressions in an `isl_union_pw_multi_aff` may live in different spaces, while there is only a single multiple expression in an `isl_multi_union_pw_aff`, which can therefore only live in a single space. This means that not every `isl_union_pw_multi_aff` can be converted to an `isl_multi_union_pw_aff`. Conversely, a zero-dimensional `isl_multi_union_pw_aff` carries no information about any possible domain and therefore cannot be converted to an `isl_union_pw_multi_aff`. Moreover, the elements of an `isl_multi_union_pw_aff` may be defined over different domains, while each multiple expression inside an `isl_union_pw_multi_aff` has a single domain. The conversion of an `isl_union_pw_multi_aff` of dimension greater than one may therefore not be exact. The following functions can be used to perform these conversions when they are possible.

```
#include <isl/aff.h>
__isl_give isl_multi_union_pw_aff *
isl_multi_union_pw_aff_from_union_pw_multi_aff(
        __isl_take isl_union_pw_multi_aff *upma);
__isl_give isl_union_pw_multi_aff *
isl_union_pw_multi_aff_from_multi_union_pw_aff(
        __isl_take isl_multi_union_pw_aff *mupa);
```

Piecewise Expressions

A piecewise expression is an expression that is described using zero or more base expression defined over the same number of cells in the domain space of the base expressions. All base expressions are defined over the same domain space and the cells are disjoint. The space of a piecewise expression is the same as that of the base expressions. If the union of the cells is a strict subset of the domain space, then the value of the piecewise expression outside this union is different for types derived from quasi-affine expressions and those derived from quasipolynomials. Piecewise expressions derived from quasi-affine expressions are considered to be undefined outside the union of their cells. Piecewise expressions derived from quasipolynomials are considered to be zero outside the union of their cells.

Piecewise quasipolynomials are mainly used by the `barvinok` library for representing the number of elements in a parametric set or map. For example, the piecewise quasipolynomial

```
[n] -> { [x] -> ((1 + n) - x) : x <= n and x >= 0 }
```

represents the number of points in the map

```
[n] -> { [x] -> [y] : x,y >= 0 and 0 <= x + y <= n }
```

The piecewise expression types defined by isl are isl_pw_aff, isl_pw_multi_aff, isl_pw_qpolynomial and isl_pw_qpolynomial_fold.

A piecewise expression with no cells can be created using the following functions.

```
#include <isl/aff.h>
__isl_give isl_pw_aff *isl_pw_aff_empty(
        __isl_take isl_space *space);
__isl_give isl_pw_multi_aff *isl_pw_multi_aff_empty(
        __isl_take isl_space *space);
```

A piecewise expression with a single universe cell can be created using the following functions.

```
#include <isl/aff.h>
__isl_give isl_pw_aff *isl_pw_aff_from_aff(
        __isl_take isl_aff *aff);
__isl_give isl_pw_multi_aff *
isl_pw_multi_aff_from_multi_aff(
        __isl_take isl_multi_aff *ma);

#include <isl/polynomial.h>
__isl_give isl_pw_qpolynomial *
isl_pw_qpolynomial_from_qpolynomial(
        __isl_take isl_qpolynomial *qp);
```

A piecewise expression with a single specified cell can be created using the following functions.

```
#include <isl/aff.h>
__isl_give isl_pw_aff *isl_pw_aff_alloc(
        __isl_take isl_set *set, __isl_take isl_aff *aff);
__isl_give isl_pw_multi_aff *isl_pw_multi_aff_alloc(
        __isl_take isl_set *set,
        __isl_take isl_multi_aff *maff);

#include <isl/polynomial.h>
__isl_give isl_pw_qpolynomial *isl_pw_qpolynomial_alloc(
        __isl_take isl_set *set,
        __isl_take isl_qpolynomial *qp);
```

The following convenience functions first create a base expression and then create a piecewise expression over a universe domain.

```
#include <isl/aff.h>
__isl_give isl_pw_aff *isl_pw_aff_zero_on_domain(
        __isl_take isl_local_space *ls);
__isl_give isl_pw_aff *isl_pw_aff_var_on_domain(
        __isl_take isl_local_space *ls,
        enum isl_dim_type type, unsigned pos);
__isl_give isl_pw_aff *isl_pw_aff_nan_on_domain(
        __isl_take isl_local_space *ls);
__isl_give isl_pw_multi_aff *isl_pw_multi_aff_zero(
        __isl_take isl_space *space);
__isl_give isl_pw_multi_aff *isl_pw_multi_aff_identity(
        __isl_take isl_space *space);
__isl_give isl_pw_multi_aff *isl_pw_multi_aff_range_map(
        __isl_take isl_space *space);
__isl_give isl_pw_multi_aff *
isl_pw_multi_aff_project_out_map(
        __isl_take isl_space *space,
        enum isl_dim_type type,
        unsigned first, unsigned n);

#include <isl/polynomial.h>
__isl_give isl_pw_qpolynomial *isl_pw_qpolynomial_zero(
        __isl_take isl_space *space);
```

The following convenience functions first create a base expression and then create a piecewise expression over a given domain.

```
#include <isl/aff.h>
__isl_give isl_pw_aff *isl_pw_aff_val_on_domain(
        __isl_take isl_set *domain,
        __isl_take isl_val *v);
__isl_give isl_pw_multi_aff *
isl_pw_multi_aff_multi_val_on_domain(
        __isl_take isl_set *domain,
        __isl_take isl_multi_val *mv);
```

As a convenience, a piecewise multiple expression can also be created from a piecewise expression. Each multiple expression in the result is derived from the corresponding base expression.

```
#include <isl/aff.h>
__isl_give isl_pw_multi_aff *isl_pw_multi_aff_from_pw_aff(
        __isl_take isl_pw_aff *pa);
```

Similarly, a piecewise quasipolynomial can be created from a piecewise quasi-affine expression using the following function.

```
#include <isl/polynomial.h>
__isl_give isl_pw_qpolynomial *
isl_pw_qpolynomial_from_pw_aff(
        __isl_take isl_pw_aff *pwaff);
```

Piecewise expressions can be copied and freed using the following functions.

```
#include <isl/aff.h>
__isl_give isl_pw_aff *isl_pw_aff_copy(
        __isl_keep isl_pw_aff *pwaff);
__isl_null isl_pw_aff *isl_pw_aff_free(
        __isl_take isl_pw_aff *pwaff);
__isl_give isl_pw_multi_aff *isl_pw_multi_aff_copy(
        __isl_keep isl_pw_multi_aff *pma);
__isl_null isl_pw_multi_aff *isl_pw_multi_aff_free(
        __isl_take isl_pw_multi_aff *pma);

#include <isl/polynomial.h>
__isl_give isl_pw_qpolynomial *isl_pw_qpolynomial_copy(
        __isl_keep isl_pw_qpolynomial *pwqp);
__isl_null isl_pw_qpolynomial *isl_pw_qpolynomial_free(
        __isl_take isl_pw_qpolynomial *pwqp);
__isl_give isl_pw_qpolynomial_fold *
isl_pw_qpolynomial_fold_copy(
        __isl_keep isl_pw_qpolynomial_fold *pwf);
__isl_null isl_pw_qpolynomial_fold *
isl_pw_qpolynomial_fold_free(
        __isl_take isl_pw_qpolynomial_fold *pwf);
```

To iterate over the different cells of a piecewise expression, use the following functions.

```
#include <isl/aff.h>
isl_bool isl_pw_aff_is_empty(__isl_keep isl_pw_aff *pwaff);
int isl_pw_aff_n_piece(__isl_keep isl_pw_aff *pwaff);
isl_stat isl_pw_aff_foreach_piece(
        __isl_keep isl_pw_aff *pwaff,
        isl_stat (*fn)(__isl_take isl_set *set,
                  __isl_take isl_aff *aff,
                  void *user), void *user);
isl_stat isl_pw_multi_aff_foreach_piece(
        __isl_keep isl_pw_multi_aff *pma,
        isl_stat (*fn)(__isl_take isl_set *set,
                    __isl_take isl_multi_aff *maff,
                    void *user), void *user);

#include <isl/polynomial.h>
```

```
isl_stat isl_pw_qpolynomial_foreach_piece(
        __isl_keep isl_pw_qpolynomial *pwqp,
        isl_stat (*fn)(__isl_take isl_set *set,
                  __isl_take isl_qpolynomial *qp,
                  void *user), void *user);
isl_stat isl_pw_qpolynomial_foreach_lifted_piece(
        __isl_keep isl_pw_qpolynomial *pwqp,
        isl_stat (*fn)(__isl_take isl_set *set,
                  __isl_take isl_qpolynomial *qp,
                  void *user), void *user);
isl_stat isl_pw_qpolynomial_fold_foreach_piece(
        __isl_keep isl_pw_qpolynomial_fold *pwf,
        isl_stat (*fn)(__isl_take isl_set *set,
                  __isl_take isl_qpolynomial_fold *fold,
                  void *user), void *user);
isl_stat isl_pw_qpolynomial_fold_foreach_lifted_piece(
        __isl_keep isl_pw_qpolynomial_fold *pwf,
        isl_stat (*fn)(__isl_take isl_set *set,
                  __isl_take isl_qpolynomial_fold *fold,
                  void *user), void *user);
```

As usual, the function `fn` should return `0` on success and `-1` on failure. The difference between `isl_pw_qpolynomial_foreach_piece` and `isl_pw_qpolynomial_foreach_lifted_piece` is that `isl_pw_qpolynomial_foreach_lifted_piece` will first compute unique representations for all existentially quantified variables and then turn these existentially quantified variables into extra set variables, adapting the associated quasipolynomial accordingly. This means that the `set` passed to `fn` will not have any existentially quantified variables, but that the dimensions of the sets may be different for different invocations of `fn`. Similarly for `isl_pw_qpolynomial_fold_foreach_piece` and `isl_pw_qpolynomial_fold_foreach_lifted_piece`.

A piecewise expression consisting of the expressions at a given position of a piecewise multiple expression can be extracted using the following function.

```
#include <isl/aff.h>
__isl_give isl_pw_aff *isl_pw_multi_aff_get_pw_aff(
        __isl_keep isl_pw_multi_aff *pma, int pos);
```

These expressions can be replaced using the following function.

```
#include <isl/aff.h>
__isl_give isl_pw_multi_aff *isl_pw_multi_aff_set_pw_aff(
        __isl_take isl_pw_multi_aff *pma, unsigned pos,
        __isl_take isl_pw_aff *pa);
```

Note that there is a difference between `isl_multi_pw_aff` and `isl_pw_multi_aff` objects. The first is a sequence of piecewise affine expressions, while the second is a piecewise sequence of affine expressions. In particular, each of the piecewise affine

expressions in an isl_multi_pw_aff may have a different domain, while all multiple expressions associated to a cell in an isl_pw_multi_aff have the same domain. It is possible to convert between the two, but when converting an isl_multi_pw_aff to an isl_pw_multi_aff, the domain of the result is the intersection of the domains of the input. The reverse conversion is exact.

```
#include <isl/aff.h>
__isl_give isl_pw_multi_aff *
isl_pw_multi_aff_from_multi_pw_aff(
        __isl_take isl_multi_pw_aff *mpa);
__isl_give isl_multi_pw_aff *
isl_multi_pw_aff_from_pw_multi_aff(
        __isl_take isl_pw_multi_aff *pma);
```

Union Expressions

A union expression collects base expressions defined over different domains. The space of a union expression is that of the shared parameter space.

The union expression types defined by isl are isl_union_pw_aff, isl_union_pw_multi_aff, isl_union_pw_qpolynomial and isl_union_pw_qpolynomial_fold. In case of isl_union_pw_aff, isl_union_pw_qpolynomial and isl_union_pw_qpolynomial_fold, there can be at most one base expression for a given domain space. In case of isl_union_pw_multi_aff, there can be multiple such expressions for a given domain space, but the domains of these expressions need to be disjoint.

An empty union expression can be created using the following functions.

```
#include <isl/aff.h>
__isl_give isl_union_pw_aff *isl_union_pw_aff_empty(
        __isl_take isl_space *space);
__isl_give isl_union_pw_multi_aff *
isl_union_pw_multi_aff_empty(
        __isl_take isl_space *space);

#include <isl/polynomial.h>
__isl_give isl_union_pw_qpolynomial *
isl_union_pw_qpolynomial_zero(
        __isl_take isl_space *space);
```

A union expression containing a single base expression can be created using the following functions.

```
#include <isl/aff.h>
__isl_give isl_union_pw_aff *
isl_union_pw_aff_from_pw_aff(
        __isl_take isl_pw_aff *pa);
__isl_give isl_union_pw_multi_aff *
isl_union_pw_multi_aff_from_aff(
```

```
        __isl_take isl_aff *aff);
__isl_give isl_union_pw_multi_aff *
isl_union_pw_multi_aff_from_pw_multi_aff(
        __isl_take isl_pw_multi_aff *pma);

#include <isl/polynomial.h>
__isl_give isl_union_pw_qpolynomial *
isl_union_pw_qpolynomial_from_pw_qpolynomial(
        __isl_take isl_pw_qpolynomial *pwqp);
```

The following functions create a base expression on each of the sets in the union set and collect the results.

```
#include <isl/aff.h>
__isl_give isl_union_pw_multi_aff *
isl_union_pw_multi_aff_from_union_pw_aff(
        __isl_take isl_union_pw_aff *upa);
__isl_give isl_union_pw_aff *
isl_union_pw_multi_aff_get_union_pw_aff(
        __isl_keep isl_union_pw_multi_aff *upma, int pos);
__isl_give isl_union_pw_aff *
isl_union_pw_aff_val_on_domain(
        __isl_take isl_union_set *domain,
        __isl_take isl_val *v);
__isl_give isl_union_pw_multi_aff *
isl_union_pw_multi_aff_multi_val_on_domain(
        __isl_take isl_union_set *domain,
        __isl_take isl_multi_val *mv);
```

An `isl_union_pw_aff` that is equal to a (parametric) affine expression on a given domain can be created using the following function.

```
#include <isl/aff.h>
__isl_give isl_union_pw_aff *
isl_union_pw_aff_aff_on_domain(
        __isl_take isl_union_set *domain,
        __isl_take isl_aff *aff);
```

A base expression can be added to a union expression using the following functions.

```
#include <isl/aff.h>
__isl_give isl_union_pw_aff *
isl_union_pw_aff_add_pw_aff(
        __isl_take isl_union_pw_aff *upa,
        __isl_take isl_pw_aff *pa);
__isl_give isl_union_pw_multi_aff *
isl_union_pw_multi_aff_add_pw_multi_aff(
        __isl_take isl_union_pw_multi_aff *upma,
        __isl_take isl_pw_multi_aff *pma);
```

```
#include <isl/polynomial.h>
__isl_give isl_union_pw_qpolynomial *
isl_union_pw_qpolynomial_add_pw_qpolynomial(
        __isl_take isl_union_pw_qpolynomial *upwqp,
        __isl_take isl_pw_qpolynomial *pwqp);
```

Union expressions can be copied and freed using the following functions.

```
#include <isl/aff.h>
__isl_give isl_union_pw_aff *isl_union_pw_aff_copy(
        __isl_keep isl_union_pw_aff *upa);
__isl_null isl_union_pw_aff *isl_union_pw_aff_free(
        __isl_take isl_union_pw_aff *upa);
__isl_give isl_union_pw_multi_aff *
isl_union_pw_multi_aff_copy(
        __isl_keep isl_union_pw_multi_aff *upma);
__isl_null isl_union_pw_multi_aff *
isl_union_pw_multi_aff_free(
        __isl_take isl_union_pw_multi_aff *upma);

#include <isl/polynomial.h>
__isl_give isl_union_pw_qpolynomial *
isl_union_pw_qpolynomial_copy(
        __isl_keep isl_union_pw_qpolynomial *upwqp);
__isl_null isl_union_pw_qpolynomial *
isl_union_pw_qpolynomial_free(
        __isl_take isl_union_pw_qpolynomial *upwqp);
__isl_give isl_union_pw_qpolynomial_fold *
isl_union_pw_qpolynomial_fold_copy(
        __isl_keep isl_union_pw_qpolynomial_fold *upwf);
__isl_null isl_union_pw_qpolynomial_fold *
isl_union_pw_qpolynomial_fold_free(
        __isl_take isl_union_pw_qpolynomial_fold *upwf);
```

To iterate over the base expressions in a union expression, use the following functions.

```
#include <isl/aff.h>
int isl_union_pw_aff_n_pw_aff(
        __isl_keep isl_union_pw_aff *upa);
isl_stat isl_union_pw_aff_foreach_pw_aff(
        __isl_keep isl_union_pw_aff *upa,
        isl_stat (*fn)(__isl_take isl_pw_aff *pa,
                void *user), void *user);
int isl_union_pw_multi_aff_n_pw_multi_aff(
        __isl_keep isl_union_pw_multi_aff *upma);
isl_stat isl_union_pw_multi_aff_foreach_pw_multi_aff(
```

```
        __isl_keep isl_union_pw_multi_aff *upma,
        isl_stat (*fn)(__isl_take isl_pw_multi_aff *pma,
                    void *user), void *user);

#include <isl/polynomial.h>
int isl_union_pw_qpolynomial_n_pw_qpolynomial(
        __isl_keep isl_union_pw_qpolynomial *upwqp);
isl_stat isl_union_pw_qpolynomial_foreach_pw_qpolynomial(
        __isl_keep isl_union_pw_qpolynomial *upwqp,
        isl_stat (*fn)(__isl_take isl_pw_qpolynomial *pwqp,
                    void *user), void *user);
int isl_union_pw_qpolynomial_fold_n_pw_qpolynomial_fold(
        __isl_keep isl_union_pw_qpolynomial_fold *upwf);
isl_stat isl_union_pw_qpolynomial_fold_foreach_pw_qpolynomial_fold(
        __isl_keep isl_union_pw_qpolynomial_fold *upwf,
        isl_stat (*fn)(__isl_take isl_pw_qpolynomial_fold *pwf,
                    void *user), void *user);
```

To extract the base expression in a given space from a union, use the following functions.

```
#include <isl/aff.h>
__isl_give isl_pw_aff *isl_union_pw_aff_extract_pw_aff(
        __isl_keep isl_union_pw_aff *upa,
        __isl_take isl_space *space);
__isl_give isl_pw_multi_aff *
isl_union_pw_multi_aff_extract_pw_multi_aff(
        __isl_keep isl_union_pw_multi_aff *upma,
        __isl_take isl_space *space);

#include <isl/polynomial.h>
__isl_give isl_pw_qpolynomial *
isl_union_pw_qpolynomial_extract_pw_qpolynomial(
        __isl_keep isl_union_pw_qpolynomial *upwqp,
        __isl_take isl_space *space);
```

1.4.14 Input and Output

For set and relation, `isl` supports its own input/output format, which is similar to the `Omega` format, but also supports the `PolyLib` format in some cases. For other object types, typically only an `isl` format is supported.

`isl` format

The `isl` format is similar to that of `Omega`, but has a different syntax for describing the parameters and allows for the definition of an existentially quantified variable as the integer division of an affine expression. For example, the set of integers `i` between `0` and `n` such that `i % 10 <= 6` can be described as

```
[n] -> { [i] : exists (a = [i/10] : 0 <= i and i <= n and
                        i - 10 a <= 6) }
```

A set or relation can have several disjuncts, separated by the keyword `or`. Each disjunct is either a conjunction of constraints or a projection (`exists`) of a conjunction of constraints. The constraints are separated by the keyword `and`.

`PolyLib` format

If the represented set is a union, then the first line contains a single number representing the number of disjuncts. Otherwise, a line containing the number 1 is optional.

Each disjunct is represented by a matrix of constraints. The first line contains two numbers representing the number of rows and columns, where the number of rows is equal to the number of constraints and the number of columns is equal to two plus the number of variables. The following lines contain the actual rows of the constraint matrix. In each row, the first column indicates whether the constraint is an equality (`0`) or inequality (`1`). The final column corresponds to the constant term.

If the set is parametric, then the coefficients of the parameters appear in the last columns before the constant column. The coefficients of any existentially quantified variables appear between those of the set variables and those of the parameters.

Extended `PolyLib` format

The extended `PolyLib` format is nearly identical to the `PolyLib` format. The only difference is that the line containing the number of rows and columns of a constraint matrix also contains four additional numbers: the number of output dimensions, the number of input dimensions, the number of local dimensions (i.e., the number of existentially quantified variables) and the number of parameters. For sets, the number of "output" dimensions is equal to the number of set dimensions, while the number of "input" dimensions is zero.

Input

Objects can be read from input using the following functions.

```
#include <isl/val.h>
__isl_give isl_val *isl_val_read_from_str(isl_ctx *ctx,
        const char *str);
__isl_give isl_multi_val *isl_multi_val_read_from_str(
        isl_ctx *ctx, const char *str);

#include <isl/set.h>
__isl_give isl_basic_set *isl_basic_set_read_from_file(
        isl_ctx *ctx, FILE *input);
__isl_give isl_basic_set *isl_basic_set_read_from_str(
        isl_ctx *ctx, const char *str);
__isl_give isl_set *isl_set_read_from_file(isl_ctx *ctx,
```

```
        FILE *input);
__isl_give isl_set *isl_set_read_from_str(isl_ctx *ctx,
        const char *str);

#include <isl/map.h>
__isl_give isl_basic_map *isl_basic_map_read_from_file(
        isl_ctx *ctx, FILE *input);
__isl_give isl_basic_map *isl_basic_map_read_from_str(
        isl_ctx *ctx, const char *str);
__isl_give isl_map *isl_map_read_from_file(
        isl_ctx *ctx, FILE *input);
__isl_give isl_map *isl_map_read_from_str(isl_ctx *ctx,
        const char *str);

#include <isl/union_set.h>
__isl_give isl_union_set *isl_union_set_read_from_file(
        isl_ctx *ctx, FILE *input);
__isl_give isl_union_set *isl_union_set_read_from_str(
        isl_ctx *ctx, const char *str);

#include <isl/union_map.h>
__isl_give isl_union_map *isl_union_map_read_from_file(
        isl_ctx *ctx, FILE *input);
__isl_give isl_union_map *isl_union_map_read_from_str(
        isl_ctx *ctx, const char *str);

#include <isl/aff.h>
__isl_give isl_aff *isl_aff_read_from_str(
        isl_ctx *ctx, const char *str);
__isl_give isl_multi_aff *isl_multi_aff_read_from_str(
        isl_ctx *ctx, const char *str);
__isl_give isl_pw_aff *isl_pw_aff_read_from_str(
        isl_ctx *ctx, const char *str);
__isl_give isl_pw_multi_aff *isl_pw_multi_aff_read_from_str(
        isl_ctx *ctx, const char *str);
__isl_give isl_multi_pw_aff *isl_multi_pw_aff_read_from_str(
        isl_ctx *ctx, const char *str);
__isl_give isl_union_pw_aff *
isl_union_pw_aff_read_from_str(
        isl_ctx *ctx, const char *str);
__isl_give isl_union_pw_multi_aff *
isl_union_pw_multi_aff_read_from_str(
        isl_ctx *ctx, const char *str);
__isl_give isl_multi_union_pw_aff *
isl_multi_union_pw_aff_read_from_str(
        isl_ctx *ctx, const char *str);
```

```
#include <isl/polynomial.h>
__isl_give isl_union_pw_qpolynomial *
isl_union_pw_qpolynomial_read_from_str(
        isl_ctx *ctx, const char *str);
```

For sets and relations, the input format is autodetected and may be either the `PolyLib` format or the `isl` format.

Output

Before anything can be printed, an `isl_printer` needs to be created.

```
__isl_give isl_printer *isl_printer_to_file(isl_ctx *ctx,
        FILE *file);
__isl_give isl_printer *isl_printer_to_str(isl_ctx *ctx);
__isl_null isl_printer *isl_printer_free(
        __isl_take isl_printer *printer);
__isl_give char *isl_printer_get_str(
        __isl_keep isl_printer *printer);
```

The printer can be inspected using the following functions.

```
FILE *isl_printer_get_file(
        __isl_keep isl_printer *printer);
int isl_printer_get_output_format(
        __isl_keep isl_printer *p);
int isl_printer_get_yaml_style(__isl_keep isl_printer *p);
```

The behavior of the printer can be modified in various ways

```
__isl_give isl_printer *isl_printer_set_output_format(
        __isl_take isl_printer *p, int output_format);
__isl_give isl_printer *isl_printer_set_indent(
        __isl_take isl_printer *p, int indent);
__isl_give isl_printer *isl_printer_set_indent_prefix(
        __isl_take isl_printer *p, const char *prefix);
__isl_give isl_printer *isl_printer_indent(
        __isl_take isl_printer *p, int indent);
__isl_give isl_printer *isl_printer_set_prefix(
        __isl_take isl_printer *p, const char *prefix);
__isl_give isl_printer *isl_printer_set_suffix(
        __isl_take isl_printer *p, const char *suffix);
__isl_give isl_printer *isl_printer_set_yaml_style(
        __isl_take isl_printer *p, int yaml_style);
```

The `output_format` may be either `ISL_FORMAT_ISL`, `ISL_FORMAT_OMEGA`, `ISL_FORMAT_POLYLIB`, `ISL_FORMAT_EXT_POLYLIB` or `ISL_FORMAT_LATEX` and defaults to `ISL_FORMAT_ISL`.

Each line in the output is prefixed by `indent_prefix`, indented by `indent` (set by `isl_printer_set_indent`) spaces (default: 0), prefixed by `prefix` and suffixed by `suffix`. In the `PolyLib` format output, the coefficients of the existentially quantified variables appear between those of the set variables and those of the parameters. The function `isl_printer_indent` increases the indentation by the specified amount (which may be negative). The YAML style may be either `ISL_YAML_STYLE_BLOCK` or `ISL_YAML_STYLE_FLOW` and when we are printing something in YAML format.

To actually print something, use

```
#include <isl/printer.h>
__isl_give isl_printer *isl_printer_print_double(
        __isl_take isl_printer *p, double d);

#include <isl/val.h>
__isl_give isl_printer *isl_printer_print_val(
        __isl_take isl_printer *p, __isl_keep isl_val *v);

#include <isl/set.h>
__isl_give isl_printer *isl_printer_print_basic_set(
        __isl_take isl_printer *printer,
        __isl_keep isl_basic_set *bset);
__isl_give isl_printer *isl_printer_print_set(
        __isl_take isl_printer *printer,
        __isl_keep isl_set *set);

#include <isl/map.h>
__isl_give isl_printer *isl_printer_print_basic_map(
        __isl_take isl_printer *printer,
        __isl_keep isl_basic_map *bmap);
__isl_give isl_printer *isl_printer_print_map(
        __isl_take isl_printer *printer,
        __isl_keep isl_map *map);

#include <isl/union_set.h>
__isl_give isl_printer *isl_printer_print_union_set(
        __isl_take isl_printer *p,
        __isl_keep isl_union_set *uset);

#include <isl/union_map.h>
__isl_give isl_printer *isl_printer_print_union_map(
        __isl_take isl_printer *p,
        __isl_keep isl_union_map *umap);

#include <isl/val.h>
__isl_give isl_printer *isl_printer_print_multi_val(
        __isl_take isl_printer *p,
        __isl_keep isl_multi_val *mv);
```

```
#include <isl/aff.h>
__isl_give isl_printer *isl_printer_print_aff(
        __isl_take isl_printer *p, __isl_keep isl_aff *aff);
__isl_give isl_printer *isl_printer_print_multi_aff(
        __isl_take isl_printer *p,
        __isl_keep isl_multi_aff *maff);
__isl_give isl_printer *isl_printer_print_pw_aff(
        __isl_take isl_printer *p,
        __isl_keep isl_pw_aff *pwaff);
__isl_give isl_printer *isl_printer_print_pw_multi_aff(
        __isl_take isl_printer *p,
        __isl_keep isl_pw_multi_aff *pma);
__isl_give isl_printer *isl_printer_print_multi_pw_aff(
        __isl_take isl_printer *p,
        __isl_keep isl_multi_pw_aff *mpa);
__isl_give isl_printer *isl_printer_print_union_pw_aff(
        __isl_take isl_printer *p,
        __isl_keep isl_union_pw_aff *upa);
__isl_give isl_printer *isl_printer_print_union_pw_multi_aff(
        __isl_take isl_printer *p,
        __isl_keep isl_union_pw_multi_aff *upma);
__isl_give isl_printer *
isl_printer_print_multi_union_pw_aff(
        __isl_take isl_printer *p,
        __isl_keep isl_multi_union_pw_aff *mupa);

#include <isl/polynomial.h>
__isl_give isl_printer *isl_printer_print_qpolynomial(
        __isl_take isl_printer *p,
        __isl_keep isl_qpolynomial *qp);
__isl_give isl_printer *isl_printer_print_pw_qpolynomial(
        __isl_take isl_printer *p,
        __isl_keep isl_pw_qpolynomial *pwqp);
__isl_give isl_printer *isl_printer_print_union_pw_qpolynomial(
        __isl_take isl_printer *p,
        __isl_keep isl_union_pw_qpolynomial *upwqp);

__isl_give isl_printer *
isl_printer_print_pw_qpolynomial_fold(
        __isl_take isl_printer *p,
        __isl_keep isl_pw_qpolynomial_fold *pwf);
__isl_give isl_printer *
isl_printer_print_union_pw_qpolynomial_fold(
        __isl_take isl_printer *p,
        __isl_keep isl_union_pw_qpolynomial_fold *upwf);
```

For `isl_printer_print_qpolynomial`, `isl_printer_print_pw_qpolynomial`

and isl_printer_print_pw_qpolynomial_fold, the output format of the printer needs to be set to either ISL_FORMAT_ISL or ISL_FORMAT_C. For isl_printer_print_union_pw_qpolynomial and isl_printer_print_union_pw_qpolynomial_fold, only ISL_FORMAT_ISL is supported. In case of printing in ISL_FORMAT_C, the user may want to set the names of all dimensions first.

isl also provides limited support for printing YAML documents, just enough for the internal use for printing such documents.

```
#include <isl/printer.h>
__isl_give isl_printer *isl_printer_yaml_start_mapping(
        __isl_take isl_printer *p);
__isl_give isl_printer *isl_printer_yaml_end_mapping(
        __isl_take isl_printer *p);
__isl_give isl_printer *isl_printer_yaml_start_sequence(
        __isl_take isl_printer *p);
__isl_give isl_printer *isl_printer_yaml_end_sequence(
        __isl_take isl_printer *p);
__isl_give isl_printer *isl_printer_yaml_next(
        __isl_take isl_printer *p);
```

A document is started by a call to either isl_printer_yaml_start_mapping or isl_printer_yaml_start_sequence. Anything printed to the printer after such a call belong to the first key of the mapping or the first element in the sequence. The function isl_printer_yaml_next moves to the value if we are currently printing a mapping key, the next key if we are printing a value or the next element if we are printing an element in a sequence. Nested mappings and sequences are initiated by the same isl_printer_yaml_start_mapping or isl_printer_yaml_start_sequence. Each call to these functions needs to have a corresponding call to isl_printer_yaml_end_mapping or isl_printer_yaml_end_sequence.

When called on a file printer, the following function flushes the file. When called on a string printer, the buffer is cleared.

```
__isl_give isl_printer *isl_printer_flush(
        __isl_take isl_printer *p);
```

The following functions allow the user to attach notes to a printer in order to keep track of additional state.

```
#include <isl/printer.h>
isl_bool isl_printer_has_note(__isl_keep isl_printer *p,
        __isl_keep isl_id *id);
__isl_give isl_id *isl_printer_get_note(
        __isl_keep isl_printer *p, __isl_take isl_id *id);
__isl_give isl_printer *isl_printer_set_note(
        __isl_take isl_printer *p,
        __isl_take isl_id *id, __isl_take isl_id *note);
```

isl_printer_set_note associates the given note to the given identifier in the printer. isl_printer_get_note retrieves a note associated to an identifier, while isl_printer_has_note checks if there is such a note. isl_printer_get_note fails if the requested note does not exist.

Alternatively, a string representation can be obtained directly using the following functions, which always print in isl format.

```
#include <isl/space.h>
__isl_give char *isl_space_to_str(
        __isl_keep isl_space *space);

#include <isl/val.h>
__isl_give char *isl_val_to_str(__isl_keep isl_val *v);
__isl_give char *isl_multi_val_to_str(
        __isl_keep isl_multi_val *mv);

#include <isl/set.h>
__isl_give char *isl_set_to_str(
        __isl_keep isl_set *set);

#include <isl/union_set.h>
__isl_give char *isl_union_set_to_str(
        __isl_keep isl_union_set *uset);

#include <isl/map.h>
__isl_give char *isl_map_to_str(
        __isl_keep isl_map *map);

#include <isl/union_map.h>
__isl_give char *isl_union_map_to_str(
        __isl_keep isl_union_map *umap);

#include <isl/aff.h>
__isl_give char *isl_aff_to_str(__isl_keep isl_aff *aff);
__isl_give char *isl_pw_aff_to_str(
        __isl_keep isl_pw_aff *pa);
__isl_give char *isl_multi_aff_to_str(
        __isl_keep isl_multi_aff *ma);
__isl_give char *isl_pw_multi_aff_to_str(
        __isl_keep isl_pw_multi_aff *pma);
__isl_give char *isl_multi_pw_aff_to_str(
        __isl_keep isl_multi_pw_aff *mpa);
__isl_give char *isl_union_pw_aff_to_str(
        __isl_keep isl_union_pw_aff *upa);
__isl_give char *isl_union_pw_multi_aff_to_str(
        __isl_keep isl_union_pw_multi_aff *upma);
__isl_give char *isl_multi_union_pw_aff_to_str(
        __isl_keep isl_multi_union_pw_aff *mupa);
```

1.4.15 Properties

Unary Properties

- Emptiness

  The following functions test whether the given set or relation contains any integer points. The "plain" variants do not perform any computations, but simply check if the given set or relation is already known to be empty.

```
isl_bool isl_basic_set_plain_is_empty(
        __isl_keep isl_basic_set *bset);
isl_bool isl_basic_set_is_empty(
        __isl_keep isl_basic_set *bset);
isl_bool isl_set_plain_is_empty(
        __isl_keep isl_set *set);
isl_bool isl_set_is_empty(__isl_keep isl_set *set);
isl_bool isl_union_set_is_empty(
        __isl_keep isl_union_set *uset);
isl_bool isl_basic_map_plain_is_empty(
        __isl_keep isl_basic_map *bmap);
isl_bool isl_basic_map_is_empty(
        __isl_keep isl_basic_map *bmap);
isl_bool isl_map_plain_is_empty(
        __isl_keep isl_map *map);
isl_bool isl_map_is_empty(__isl_keep isl_map *map);
isl_bool isl_union_map_is_empty(
        __isl_keep isl_union_map *umap);
```

- Universality

```
isl_bool isl_basic_set_plain_is_universe(
        __isl_keep isl_basic_set *bset);
isl_bool isl_basic_set_is_universe(
        __isl_keep isl_basic_set *bset);
isl_bool isl_basic_map_plain_is_universe(
        __isl_keep isl_basic_map *bmap);
isl_bool isl_basic_map_is_universe(
        __isl_keep isl_basic_map *bmap);
isl_bool isl_set_plain_is_universe(
        __isl_keep isl_set *set);
isl_bool isl_map_plain_is_universe(
        __isl_keep isl_map *map);
```

- Single-valuedness

```
#include <isl/set.h>
isl_bool isl_set_is_singleton(__isl_keep isl_set *set);
```

```
#include <isl/map.h>
isl_bool isl_basic_map_is_single_valued(
        __isl_keep isl_basic_map *bmap);
isl_bool isl_map_plain_is_single_valued(
        __isl_keep isl_map *map);
isl_bool isl_map_is_single_valued(__isl_keep isl_map *map);

#include <isl/union_map.h>
isl_bool isl_union_map_is_single_valued(
        __isl_keep isl_union_map *umap);
```

- Injectivity

```
isl_bool isl_map_plain_is_injective(
        __isl_keep isl_map *map);
isl_bool isl_map_is_injective(
        __isl_keep isl_map *map);
isl_bool isl_union_map_plain_is_injective(
        __isl_keep isl_union_map *umap);
isl_bool isl_union_map_is_injective(
        __isl_keep isl_union_map *umap);
```

- Bijectivity

```
isl_bool isl_map_is_bijective(
        __isl_keep isl_map *map);
isl_bool isl_union_map_is_bijective(
        __isl_keep isl_union_map *umap);
```

- Identity

  The following functions test whether the given relation only maps elements to themselves.

```
#include <isl/map.h>
isl_bool isl_map_is_identity(
        __isl_keep isl_map *map);

#include <isl/union_map.h>
isl_bool isl_union_map_is_identity(
        __isl_keep isl_union_map *umap);
```

- Position

```
__isl_give isl_val *
isl_basic_map_plain_get_val_if_fixed(
        __isl_keep isl_basic_map *bmap,
        enum isl_dim_type type, unsigned pos);
__isl_give isl_val *isl_set_plain_get_val_if_fixed(
        __isl_keep isl_set *set,
        enum isl_dim_type type, unsigned pos);
__isl_give isl_val *isl_map_plain_get_val_if_fixed(
        __isl_keep isl_map *map,
        enum isl_dim_type type, unsigned pos);
```

If the set or relation obviously lies on a hyperplane where the given dimension has a fixed value, then return that value. Otherwise return NaN.

- Stride

```
isl_stat isl_set_dim_residue_class_val(
        __isl_keep isl_set *set,
        int pos, __isl_give isl_val **modulo,
        __isl_give isl_val **residue);
```

Check if the values of the given set dimension are equal to a fixed value modulo some integer value. If so, assign the modulo to `*modulo` and the fixed value to `*residue`. If the given dimension attains only a single value, then assign `0` to `*modulo` and the fixed value to `*residue`. If the dimension does not attain only a single value and if no modulo can be found then assign 1 to `*modulo` and 1 to `*residue`.

- Dependence

To check whether the description of a set, relation or function depends on one or more given dimensions, the following functions can be used.

```
#include <isl/constraint.h>
isl_bool isl_constraint_involves_dims(
        __isl_keep isl_constraint *constraint,
        enum isl_dim_type type, unsigned first, unsigned n);

#include <isl/set.h>
isl_bool isl_basic_set_involves_dims(
        __isl_keep isl_basic_set *bset,
        enum isl_dim_type type, unsigned first, unsigned n);
isl_bool isl_set_involves_dims(__isl_keep isl_set *set,
        enum isl_dim_type type, unsigned first, unsigned n);

#include <isl/map.h>
isl_bool isl_basic_map_involves_dims(
```

```
        __isl_keep isl_basic_map *bmap,
        enum isl_dim_type type, unsigned first, unsigned n);
isl_bool isl_map_involves_dims(__isl_keep isl_map *map,
        enum isl_dim_type type, unsigned first, unsigned n);

#include <isl/union_map.h>
isl_bool isl_union_map_involves_dims(
        __isl_keep isl_union_map *umap,
        enum isl_dim_type type, unsigned first, unsigned n);

#include <isl/aff.h>
isl_bool isl_aff_involves_dims(__isl_keep isl_aff *aff,
        enum isl_dim_type type, unsigned first, unsigned n);
isl_bool isl_pw_aff_involves_dims(
        __isl_keep isl_pw_aff *pwaff,
        enum isl_dim_type type, unsigned first, unsigned n);
isl_bool isl_multi_aff_involves_dims(
        __isl_keep isl_multi_aff *ma,
        enum isl_dim_type type, unsigned first, unsigned n);
isl_bool isl_multi_pw_aff_involves_dims(
        __isl_keep isl_multi_pw_aff *mpa,
        enum isl_dim_type type, unsigned first, unsigned n);

#include <isl/polynomial.h>
isl_bool isl_qpolynomial_involves_dims(
        __isl_keep isl_qpolynomial *qp,
        enum isl_dim_type type, unsigned first, unsigned n);
```

Similarly, the following functions can be used to check whether a given dimension is involved in any lower or upper bound.

```
#include <isl/set.h>
isl_bool isl_set_dim_has_any_lower_bound(
        __isl_keep isl_set *set,
        enum isl_dim_type type, unsigned pos);
isl_bool isl_set_dim_has_any_upper_bound(
        __isl_keep isl_set *set,
        enum isl_dim_type type, unsigned pos);
```

Note that these functions return true even if there is a bound on the dimension on only some of the basic sets of `set`. To check if they have a bound for all of the basic sets in `set`, use the following functions instead.

```
#include <isl/set.h>
isl_bool isl_set_dim_has_lower_bound(
```

```
        __isl_keep isl_set *set,
        enum isl_dim_type type, unsigned pos);
isl_bool isl_set_dim_has_upper_bound(
        __isl_keep isl_set *set,
        enum isl_dim_type type, unsigned pos);
```

- Space

  To check whether a set is a parameter domain, use this function:

```
isl_bool isl_set_is_params(__isl_keep isl_set *set);
isl_bool isl_union_set_is_params(
        __isl_keep isl_union_set *uset);
```

- Wrapping

  The following functions check whether the space of the given (basic) set or relation range is a wrapped relation.

```
#include <isl/space.h>
isl_bool isl_space_is_wrapping(
        __isl_keep isl_space *space);
isl_bool isl_space_domain_is_wrapping(
        __isl_keep isl_space *space);
isl_bool isl_space_range_is_wrapping(
        __isl_keep isl_space *space);

#include <isl/set.h>
isl_bool isl_basic_set_is_wrapping(
        __isl_keep isl_basic_set *bset);
isl_bool isl_set_is_wrapping(__isl_keep isl_set *set);

#include <isl/map.h>
isl_bool isl_map_domain_is_wrapping(
        __isl_keep isl_map *map);
isl_bool isl_map_range_is_wrapping(
        __isl_keep isl_map *map);

#include <isl/val.h>
isl_bool isl_multi_val_range_is_wrapping(
        __isl_keep isl_multi_val *mv);

#include <isl/aff.h>
isl_bool isl_multi_aff_range_is_wrapping(
        __isl_keep isl_multi_aff *ma);
isl_bool isl_multi_pw_aff_range_is_wrapping(
        __isl_keep isl_multi_pw_aff *mpa);
isl_bool isl_multi_union_pw_aff_range_is_wrapping(
        __isl_keep isl_multi_union_pw_aff *mupa);
```

The input to isl_space_is_wrapping should be the space of a set, while that of isl_space_domain_is_wrapping and isl_space_range_is_wrapping should be the space of a relation.

- Internal Product

```
isl_bool isl_basic_map_can_zip(
        __isl_keep isl_basic_map *bmap);
isl_bool isl_map_can_zip(__isl_keep isl_map *map);
```

Check whether the product of domain and range of the given relation can be computed, i.e., whether both domain and range are nested relations.

- Currying

```
#include <isl/space.h>
isl_bool isl_space_can_curry(
        __isl_keep isl_space *space);

#include <isl/map.h>
isl_bool isl_basic_map_can_curry(
        __isl_keep isl_basic_map *bmap);
isl_bool isl_map_can_curry(__isl_keep isl_map *map);
```

Check whether the domain of the (basic) relation is a wrapped relation.

```
#include <isl/space.h>
__isl_give isl_space *isl_space_uncurry(
        __isl_take isl_space *space);

#include <isl/map.h>
isl_bool isl_basic_map_can_uncurry(
        __isl_keep isl_basic_map *bmap);
isl_bool isl_map_can_uncurry(__isl_keep isl_map *map);
```

Check whether the range of the (basic) relation is a wrapped relation.

```
#include <isl/space.h>
isl_bool isl_space_can_range_curry(
        __isl_keep isl_space *space);

#include <isl/map.h>
isl_bool isl_map_can_range_curry(
        __isl_keep isl_map *map);
```

Check whether the domain of the relation wrapped in the range of the input is itself a wrapped relation.

- Special Values

```
#include <isl/aff.h>
isl_bool isl_aff_is_cst(__isl_keep isl_aff *aff);
isl_bool isl_pw_aff_is_cst(__isl_keep isl_pw_aff *pwaff);
isl_bool isl_multi_pw_aff_is_cst(
        __isl_keep isl_multi_pw_aff *mpa);
```

Check whether the given expression is a constant.

```
#include <isl/aff.h>
isl_bool isl_aff_is_nan(__isl_keep isl_aff *aff);
isl_bool isl_pw_aff_involves_nan(
        __isl_keep isl_pw_aff *pa);

#include <isl/polynomial.h>
isl_bool isl_qpolynomial_fold_is_nan(
        __isl_keep isl_qpolynomial_fold *fold);
```

Check whether the given expression is equal to or involves NaN.

```
#include <isl/aff.h>
isl_bool isl_aff_plain_is_zero(
        __isl_keep isl_aff *aff);
```

Check whether the affine expression is obviously zero.

Binary Properties

- Equality

  The following functions check whether two objects represent the same set, relation or function. The `plain` variants only return true if the objects are obviously the same. That is, they may return false even if the objects are the same, but they will never return true if the objects are not the same.

```
#include <isl/set.h>
isl_bool isl_basic_set_plain_is_equal(
        __isl_keep isl_basic_set *bset1,
        __isl_keep isl_basic_set *bset2);
isl_bool isl_basic_set_is_equal(
        __isl_keep isl_basic_set *bset1,
        __isl_keep isl_basic_set *bset2);
isl_bool isl_set_plain_is_equal(
        __isl_keep isl_set *set1,
        __isl_keep isl_set *set2);
isl_bool isl_set_is_equal(__isl_keep isl_set *set1,
        __isl_keep isl_set *set2);
```

```
#include <isl/map.h>
isl_bool isl_basic_map_is_equal(
        __isl_keep isl_basic_map *bmap1,
        __isl_keep isl_basic_map *bmap2);
isl_bool isl_map_is_equal(__isl_keep isl_map *map1,
        __isl_keep isl_map *map2);
isl_bool isl_map_plain_is_equal(
        __isl_keep isl_map *map1,
        __isl_keep isl_map *map2);

#include <isl/union_set.h>
isl_bool isl_union_set_is_equal(
        __isl_keep isl_union_set *uset1,
        __isl_keep isl_union_set *uset2);

#include <isl/union_map.h>
isl_bool isl_union_map_is_equal(
        __isl_keep isl_union_map *umap1,
        __isl_keep isl_union_map *umap2);

#include <isl/aff.h>
isl_bool isl_aff_plain_is_equal(
        __isl_keep isl_aff *aff1,
        __isl_keep isl_aff *aff2);
isl_bool isl_multi_aff_plain_is_equal(
        __isl_keep isl_multi_aff *maff1,
        __isl_keep isl_multi_aff *maff2);
isl_bool isl_pw_aff_plain_is_equal(
        __isl_keep isl_pw_aff *pwaff1,
        __isl_keep isl_pw_aff *pwaff2);
isl_bool isl_pw_multi_aff_plain_is_equal(
        __isl_keep isl_pw_multi_aff *pma1,
        __isl_keep isl_pw_multi_aff *pma2);
isl_bool isl_multi_pw_aff_plain_is_equal(
        __isl_keep isl_multi_pw_aff *mpa1,
        __isl_keep isl_multi_pw_aff *mpa2);
isl_bool isl_multi_pw_aff_is_equal(
        __isl_keep isl_multi_pw_aff *mpa1,
        __isl_keep isl_multi_pw_aff *mpa2);
isl_bool isl_union_pw_aff_plain_is_equal(
        __isl_keep isl_union_pw_aff *upa1,
        __isl_keep isl_union_pw_aff *upa2);
isl_bool isl_union_pw_multi_aff_plain_is_equal(
        __isl_keep isl_union_pw_multi_aff *upma1,
        __isl_keep isl_union_pw_multi_aff *upma2);
isl_bool isl_multi_union_pw_aff_plain_is_equal(
```

```
        __isl_keep isl_multi_union_pw_aff *mupa1,
        __isl_keep isl_multi_union_pw_aff *mupa2);

#include <isl/polynomial.h>
isl_bool isl_union_pw_qpolynomial_plain_is_equal(
        __isl_keep isl_union_pw_qpolynomial *upwqp1,
        __isl_keep isl_union_pw_qpolynomial *upwqp2);
isl_bool isl_union_pw_qpolynomial_fold_plain_is_equal(
        __isl_keep isl_union_pw_qpolynomial_fold *upwf1,
        __isl_keep isl_union_pw_qpolynomial_fold *upwf2);
```

- Disjointness

```
#include <isl/set.h>
isl_bool isl_basic_set_is_disjoint(
        __isl_keep isl_basic_set *bset1,
        __isl_keep isl_basic_set *bset2);
isl_bool isl_set_plain_is_disjoint(
        __isl_keep isl_set *set1,
        __isl_keep isl_set *set2);
isl_bool isl_set_is_disjoint(__isl_keep isl_set *set1,
        __isl_keep isl_set *set2);

#include <isl/map.h>
isl_bool isl_basic_map_is_disjoint(
        __isl_keep isl_basic_map *bmap1,
        __isl_keep isl_basic_map *bmap2);
isl_bool isl_map_is_disjoint(__isl_keep isl_map *map1,
        __isl_keep isl_map *map2);

#include <isl/union_set.h>
isl_bool isl_union_set_is_disjoint(
        __isl_keep isl_union_set *uset1,
        __isl_keep isl_union_set *uset2);

#include <isl/union_map.h>
isl_bool isl_union_map_is_disjoint(
        __isl_keep isl_union_map *umap1,
        __isl_keep isl_union_map *umap2);
```

- Subset

```
isl_bool isl_basic_set_is_subset(
        __isl_keep isl_basic_set *bset1,
        __isl_keep isl_basic_set *bset2);
```

```
isl_bool isl_set_is_subset(__isl_keep isl_set *set1,
        __isl_keep isl_set *set2);
isl_bool isl_set_is_strict_subset(
        __isl_keep isl_set *set1,
        __isl_keep isl_set *set2);
isl_bool isl_union_set_is_subset(
        __isl_keep isl_union_set *uset1,
        __isl_keep isl_union_set *uset2);
isl_bool isl_union_set_is_strict_subset(
        __isl_keep isl_union_set *uset1,
        __isl_keep isl_union_set *uset2);
isl_bool isl_basic_map_is_subset(
        __isl_keep isl_basic_map *bmap1,
        __isl_keep isl_basic_map *bmap2);
isl_bool isl_basic_map_is_strict_subset(
        __isl_keep isl_basic_map *bmap1,
        __isl_keep isl_basic_map *bmap2);
isl_bool isl_map_is_subset(
        __isl_keep isl_map *map1,
        __isl_keep isl_map *map2);
isl_bool isl_map_is_strict_subset(
        __isl_keep isl_map *map1,
        __isl_keep isl_map *map2);
isl_bool isl_union_map_is_subset(
        __isl_keep isl_union_map *umap1,
        __isl_keep isl_union_map *umap2);
isl_bool isl_union_map_is_strict_subset(
        __isl_keep isl_union_map *umap1,
        __isl_keep isl_union_map *umap2);
```

Check whether the first argument is a (strict) subset of the second argument.

- Order

  Every comparison function returns a negative value if the first argument is considered smaller than the second, a positive value if the first argument is considered greater and zero if the two constraints are considered the same by the comparison criterion.

  ```
  #include <isl/constraint.h>
  int isl_constraint_plain_cmp(
          __isl_keep isl_constraint *c1,
          __isl_keep isl_constraint *c2);
  ```

  This function is useful for sorting `isl_constraint`s. The order depends on the internal representation of the inputs. The order is fixed over different calls to the function (assuming the internal representation of the inputs has not changed), but may change over different versions of `isl`.

```
#include <isl/constraint.h>
int isl_constraint_cmp_last_non_zero(
        __isl_keep isl_constraint *c1,
        __isl_keep isl_constraint *c2);
```

This function can be used to sort constraints that live in the same local space. Constraints that involve "earlier" dimensions or that have a smaller coefficient for the shared latest dimension are considered smaller than other constraints. This function only defines a **partial** order.

```
#include <isl/set.h>
int isl_set_plain_cmp(__isl_keep isl_set *set1,
        __isl_keep isl_set *set2);
```

This function is useful for sorting `isl_set`s. The order depends on the internal representation of the inputs. The order is fixed over different calls to the function (assuming the internal representation of the inputs has not changed), but may change over different versions of `isl`.

```
#include <isl/aff.h>
int isl_pw_aff_plain_cmp(__isl_keep isl_pw_aff *pa1,
        __isl_keep isl_pw_aff *pa2);
```

The function `isl_pw_aff_plain_cmp` can be used to sort `isl_pw_aff`s. The order is not strictly defined. The current order sorts expressions that only involve earlier dimensions before those that involve later dimensions.

1.4.16 Unary Operations

- Complement

```
__isl_give isl_set *isl_set_complement(
        __isl_take isl_set *set);
__isl_give isl_map *isl_map_complement(
        __isl_take isl_map *map);
```

- Inverse map

```
#include <isl/space.h>
__isl_give isl_space *isl_space_reverse(
        __isl_take isl_space *space);

#include <isl/map.h>
__isl_give isl_basic_map *isl_basic_map_reverse(
        __isl_take isl_basic_map *bmap);
__isl_give isl_map *isl_map_reverse(
        __isl_take isl_map *map);
```

```
#include <isl/union_map.h>
__isl_give isl_union_map *isl_union_map_reverse(
        __isl_take isl_union_map *umap);
```

- Projection

```
#include <isl/space.h>
__isl_give isl_space *isl_space_domain(
        __isl_take isl_space *space);
__isl_give isl_space *isl_space_range(
        __isl_take isl_space *space);
__isl_give isl_space *isl_space_params(
        __isl_take isl_space *space);

#include <isl/local_space.h>
__isl_give isl_local_space *isl_local_space_domain(
        __isl_take isl_local_space *ls);
__isl_give isl_local_space *isl_local_space_range(
        __isl_take isl_local_space *ls);

#include <isl/set.h>
__isl_give isl_basic_set *isl_basic_set_project_out(
        __isl_take isl_basic_set *bset,
        enum isl_dim_type type, unsigned first, unsigned n);
__isl_give isl_set *isl_set_project_out(__isl_take isl_set *set,
        enum isl_dim_type type, unsigned first, unsigned n);
__isl_give isl_map *isl_set_project_onto_map(
        __isl_take isl_set *set,
        enum isl_dim_type type, unsigned first,
        unsigned n);
__isl_give isl_basic_set *isl_basic_set_params(
        __isl_take isl_basic_set *bset);
__isl_give isl_set *isl_set_params(__isl_take isl_set *set);
```

The function `isl_set_project_onto_map` returns a relation that projects the input set onto the given set dimensions.

```
#include <isl/map.h>
__isl_give isl_basic_map *isl_basic_map_project_out(
        __isl_take isl_basic_map *bmap,
        enum isl_dim_type type, unsigned first, unsigned n);
__isl_give isl_map *isl_map_project_out(__isl_take isl_map *map,
        enum isl_dim_type type, unsigned first, unsigned n);
__isl_give isl_basic_set *isl_basic_map_domain(
        __isl_take isl_basic_map *bmap);
```

```
__isl_give isl_basic_set *isl_basic_map_range(
        __isl_take isl_basic_map *bmap);
__isl_give isl_set *isl_map_params(__isl_take isl_map *map);
__isl_give isl_set *isl_map_domain(
        __isl_take isl_map *bmap);
__isl_give isl_set *isl_map_range(
        __isl_take isl_map *map);

#include <isl/union_set.h>
__isl_give isl_union_set *isl_union_set_project_out(
        __isl_take isl_union_set *uset,
        enum isl_dim_type type,
        unsigned first, unsigned n);
__isl_give isl_set *isl_union_set_params(
        __isl_take isl_union_set *uset);
```

The function `isl_union_set_project_out` can only project out parameters.

```
#include <isl/union_map.h>
__isl_give isl_union_map *isl_union_map_project_out(
        __isl_take isl_union_map *umap,
        enum isl_dim_type type, unsigned first, unsigned n);
__isl_give isl_set *isl_union_map_params(
        __isl_take isl_union_map *umap);
__isl_give isl_union_set *isl_union_map_domain(
        __isl_take isl_union_map *umap);
__isl_give isl_union_set *isl_union_map_range(
        __isl_take isl_union_map *umap);
```

The function `isl_union_map_project_out` can only project out parameters.

```
#include <isl/aff.h>
__isl_give isl_aff *isl_aff_project_domain_on_params(
        __isl_take isl_aff *aff);
__isl_give isl_pw_multi_aff *
isl_pw_multi_aff_project_domain_on_params(
        __isl_take isl_pw_multi_aff *pma);
__isl_give isl_set *isl_pw_aff_domain(
        __isl_take isl_pw_aff *pwaff);
__isl_give isl_set *isl_pw_multi_aff_domain(
        __isl_take isl_pw_multi_aff *pma);
__isl_give isl_set *isl_multi_pw_aff_domain(
        __isl_take isl_multi_pw_aff *mpa);
__isl_give isl_union_set *isl_union_pw_aff_domain(
        __isl_take isl_union_pw_aff *upa);
```

```
__isl_give isl_union_set *isl_union_pw_multi_aff_domain(
        __isl_take isl_union_pw_multi_aff *upma);
__isl_give isl_union_set *
isl_multi_union_pw_aff_domain(
        __isl_take isl_multi_union_pw_aff *mupa);
__isl_give isl_set *isl_pw_aff_params(
        __isl_take isl_pw_aff *pwa);
```

The function `isl_multi_union_pw_aff_domain` requires its input to have at least one set dimension.

```
#include <isl/polynomial.h>
__isl_give isl_qpolynomial *
isl_qpolynomial_project_domain_on_params(
        __isl_take isl_qpolynomial *qp);
__isl_give isl_pw_qpolynomial *
isl_pw_qpolynomial_project_domain_on_params(
        __isl_take isl_pw_qpolynomial *pwqp);
__isl_give isl_pw_qpolynomial_fold *
isl_pw_qpolynomial_fold_project_domain_on_params(
        __isl_take isl_pw_qpolynomial_fold *pwf);
__isl_give isl_set *isl_pw_qpolynomial_domain(
        __isl_take isl_pw_qpolynomial *pwqp);
__isl_give isl_union_set *isl_union_pw_qpolynomial_fold_domain(
        __isl_take isl_union_pw_qpolynomial_fold *upwf);
__isl_give isl_union_set *isl_union_pw_qpolynomial_domain(
        __isl_take isl_union_pw_qpolynomial *upwqp);

#include <isl/space.h>
__isl_give isl_space *isl_space_domain_map(
        __isl_take isl_space *space);
__isl_give isl_space *isl_space_range_map(
        __isl_take isl_space *space);

#include <isl/map.h>
__isl_give isl_map *isl_set_wrapped_domain_map(
        __isl_take isl_set *set);
__isl_give isl_basic_map *isl_basic_map_domain_map(
        __isl_take isl_basic_map *bmap);
__isl_give isl_basic_map *isl_basic_map_range_map(
        __isl_take isl_basic_map *bmap);
__isl_give isl_map *isl_map_domain_map(__isl_take isl_map *map);
__isl_give isl_map *isl_map_range_map(__isl_take isl_map *map);

#include <isl/union_map.h>
```

```
__isl_give isl_union_map *isl_union_map_domain_map(
        __isl_take isl_union_map *umap);
__isl_give isl_union_pw_multi_aff *
isl_union_map_domain_map_union_pw_multi_aff(
        __isl_take isl_union_map *umap);
__isl_give isl_union_map *isl_union_map_range_map(
        __isl_take isl_union_map *umap);
__isl_give isl_union_map *
isl_union_set_wrapped_domain_map(
        __isl_take isl_union_set *uset);
```

The functions above construct a (basic, regular or union) relation that maps (a wrapped version of) the input relation to its domain or range. `isl_set_wrapped_domain_map` maps the input set to the domain of its wrapped relation.

- Elimination

```
__isl_give isl_basic_set *isl_basic_set_eliminate(
        __isl_take isl_basic_set *bset,
        enum isl_dim_type type,
        unsigned first, unsigned n);
__isl_give isl_set *isl_set_eliminate(
        __isl_take isl_set *set, enum isl_dim_type type,
        unsigned first, unsigned n);
__isl_give isl_basic_map *isl_basic_map_eliminate(
        __isl_take isl_basic_map *bmap,
        enum isl_dim_type type,
        unsigned first, unsigned n);
__isl_give isl_map *isl_map_eliminate(
        __isl_take isl_map *map, enum isl_dim_type type,
        unsigned first, unsigned n);
```

Eliminate the coefficients for the given dimensions from the constraints, without removing the dimensions.

- Constructing a set from a parameter domain

A zero-dimensional space or (basic) set can be constructed on a given parameter domain using the following functions.

```
#include <isl/space.h>
__isl_give isl_space *isl_space_set_from_params(
        __isl_take isl_space *space);

#include <isl/set.h>
__isl_give isl_basic_set *isl_basic_set_from_params(
        __isl_take isl_basic_set *bset);
__isl_give isl_set *isl_set_from_params(
        __isl_take isl_set *set);
```

- Constructing a relation from one or two sets

  Create a relation with the given set(s) as domain and/or range. If only the domain or the range is specified, then the range or domain of the created relation is a zero-dimensional flat anonymous space.

```
#include <isl/space.h>
__isl_give isl_space *isl_space_from_domain(
        __isl_take isl_space *space);
__isl_give isl_space *isl_space_from_range(
        __isl_take isl_space *space);
__isl_give isl_space *isl_space_map_from_set(
        __isl_take isl_space *space);
__isl_give isl_space *isl_space_map_from_domain_and_range(
        __isl_take isl_space *domain,
        __isl_take isl_space *range);

#include <isl/local_space.h>
__isl_give isl_local_space *isl_local_space_from_domain(
        __isl_take isl_local_space *ls);

#include <isl/map.h>
__isl_give isl_map *isl_map_from_domain(
        __isl_take isl_set *set);
__isl_give isl_map *isl_map_from_range(
        __isl_take isl_set *set);

#include <isl/union_map.h>
__isl_give isl_union_map *
isl_union_map_from_domain_and_range(
        __isl_take isl_union_set *domain,
        __isl_take isl_union_set *range);

#include <isl/val.h>
__isl_give isl_multi_val *isl_multi_val_from_range(
        __isl_take isl_multi_val *mv);

#include <isl/aff.h>
__isl_give isl_multi_aff *isl_multi_aff_from_range(
        __isl_take isl_multi_aff *ma);
__isl_give isl_pw_aff *isl_pw_aff_from_range(
        __isl_take isl_pw_aff *pwa);
__isl_give isl_multi_pw_aff *isl_multi_pw_aff_from_range(
        __isl_take isl_multi_pw_aff *mpa);
__isl_give isl_multi_union_pw_aff *
isl_multi_union_pw_aff_from_range(
```

```
        __isl_take isl_multi_union_pw_aff *mupa);
__isl_give isl_pw_multi_aff *isl_pw_multi_aff_from_domain(
        __isl_take isl_set *set);
__isl_give isl_union_pw_multi_aff *
isl_union_pw_multi_aff_from_domain(
        __isl_take isl_union_set *uset);
```

- Slicing

```
#include <isl/set.h>
__isl_give isl_basic_set *isl_basic_set_fix_si(
        __isl_take isl_basic_set *bset,
        enum isl_dim_type type, unsigned pos, int value);
__isl_give isl_basic_set *isl_basic_set_fix_val(
        __isl_take isl_basic_set *bset,
        enum isl_dim_type type, unsigned pos,
        __isl_take isl_val *v);
__isl_give isl_set *isl_set_fix_si(__isl_take isl_set *set,
        enum isl_dim_type type, unsigned pos, int value);
__isl_give isl_set *isl_set_fix_val(
        __isl_take isl_set *set,
        enum isl_dim_type type, unsigned pos,
        __isl_take isl_val *v);

#include <isl/map.h>
__isl_give isl_basic_map *isl_basic_map_fix_si(
        __isl_take isl_basic_map *bmap,
        enum isl_dim_type type, unsigned pos, int value);
__isl_give isl_basic_map *isl_basic_map_fix_val(
        __isl_take isl_basic_map *bmap,
        enum isl_dim_type type, unsigned pos,
        __isl_take isl_val *v);
__isl_give isl_map *isl_map_fix_si(__isl_take isl_map *map,
        enum isl_dim_type type, unsigned pos, int value);
__isl_give isl_map *isl_map_fix_val(
        __isl_take isl_map *map,
        enum isl_dim_type type, unsigned pos,
        __isl_take isl_val *v);

#include <isl/aff.h>
__isl_give isl_pw_multi_aff *isl_pw_multi_aff_fix_si(
        __isl_take isl_pw_multi_aff *pma,
        enum isl_dim_type type, unsigned pos, int value);

#include <isl/polynomial.h>
```

```
__isl_give isl_pw_qpolynomial *isl_pw_qpolynomial_fix_val(
        __isl_take isl_pw_qpolynomial *pwqp,
        enum isl_dim_type type, unsigned n,
        __isl_take isl_val *v);
```

Intersect the set, relation or function domain with the hyperplane where the given dimension has the fixed given value.

```
__isl_give isl_basic_map *isl_basic_map_lower_bound_si(
        __isl_take isl_basic_map *bmap,
        enum isl_dim_type type, unsigned pos, int value);
__isl_give isl_basic_map *isl_basic_map_upper_bound_si(
        __isl_take isl_basic_map *bmap,
        enum isl_dim_type type, unsigned pos, int value);
__isl_give isl_set *isl_set_lower_bound_si(
        __isl_take isl_set *set,
        enum isl_dim_type type, unsigned pos, int value);
__isl_give isl_set *isl_set_lower_bound_val(
        __isl_take isl_set *set,
        enum isl_dim_type type, unsigned pos,
        __isl_take isl_val *value);
__isl_give isl_map *isl_map_lower_bound_si(
        __isl_take isl_map *map,
        enum isl_dim_type type, unsigned pos, int value);
__isl_give isl_set *isl_set_upper_bound_si(
        __isl_take isl_set *set,
        enum isl_dim_type type, unsigned pos, int value);
__isl_give isl_set *isl_set_upper_bound_val(
        __isl_take isl_set *set,
        enum isl_dim_type type, unsigned pos,
        __isl_take isl_val *value);
__isl_give isl_map *isl_map_upper_bound_si(
        __isl_take isl_map *map,
        enum isl_dim_type type, unsigned pos, int value);
```

Intersect the set or relation with the half-space where the given dimension has a value bounded by the fixed given integer value.

```
__isl_give isl_set *isl_set_equate(__isl_take isl_set *set,
        enum isl_dim_type type1, int pos1,
        enum isl_dim_type type2, int pos2);
__isl_give isl_basic_map *isl_basic_map_equate(
        __isl_take isl_basic_map *bmap,
        enum isl_dim_type type1, int pos1,
        enum isl_dim_type type2, int pos2);
```

```
__isl_give isl_map *isl_map_equate(__isl_take isl_map *map,
        enum isl_dim_type type1, int pos1,
        enum isl_dim_type type2, int pos2);
```

Intersect the set or relation with the hyperplane where the given dimensions are equal to each other.

```
__isl_give isl_map *isl_map_oppose(__isl_take isl_map *map,
        enum isl_dim_type type1, int pos1,
        enum isl_dim_type type2, int pos2);
```

Intersect the relation with the hyperplane where the given dimensions have opposite values.

```
__isl_give isl_map *isl_map_order_le(
        __isl_take isl_map *map,
        enum isl_dim_type type1, int pos1,
        enum isl_dim_type type2, int pos2);
__isl_give isl_basic_map *isl_basic_map_order_ge(
        __isl_take isl_basic_map *bmap,
        enum isl_dim_type type1, int pos1,
        enum isl_dim_type type2, int pos2);
__isl_give isl_map *isl_map_order_ge(
        __isl_take isl_map *map,
        enum isl_dim_type type1, int pos1,
        enum isl_dim_type type2, int pos2);
__isl_give isl_map *isl_map_order_lt(__isl_take isl_map *map,
        enum isl_dim_type type1, int pos1,
        enum isl_dim_type type2, int pos2);
__isl_give isl_basic_map *isl_basic_map_order_gt(
        __isl_take isl_basic_map *bmap,
        enum isl_dim_type type1, int pos1,
        enum isl_dim_type type2, int pos2);
__isl_give isl_map *isl_map_order_gt(__isl_take isl_map *map,
        enum isl_dim_type type1, int pos1,
        enum isl_dim_type type2, int pos2);
```

Intersect the relation with the half-space where the given dimensions satisfy the given ordering.

- Locus

```
#include <isl/aff.h>
__isl_give isl_basic_set *isl_aff_zero_basic_set(
        __isl_take isl_aff *aff);
__isl_give isl_basic_set *isl_aff_neg_basic_set(
```

```
        __isl_take isl_aff *aff);
__isl_give isl_set *isl_pw_aff_pos_set(
        __isl_take isl_pw_aff *pa);
__isl_give isl_set *isl_pw_aff_nonneg_set(
        __isl_take isl_pw_aff *pwaff);
__isl_give isl_set *isl_pw_aff_zero_set(
        __isl_take isl_pw_aff *pwaff);
__isl_give isl_set *isl_pw_aff_non_zero_set(
        __isl_take isl_pw_aff *pwaff);
__isl_give isl_union_set *
isl_union_pw_aff_zero_union_set(
        __isl_take isl_union_pw_aff *upa);
__isl_give isl_union_set *
isl_multi_union_pw_aff_zero_union_set(
        __isl_take isl_multi_union_pw_aff *mupa);
```

The function `isl_aff_neg_basic_set` returns a basic set containing those elements in the domain space of `aff` where `aff` is negative. The function `isl_pw_aff_nonneg_set` returns a set containing those elements in the domain of `pwaff` where `pwaff` is non-negative. The function `isl_multi_union_pw_aff_zero_union_set` returns a union set containing those elements in the domains of its elements where they are all zero.

- Identity

```
__isl_give isl_map *isl_set_identity(
        __isl_take isl_set *set);
__isl_give isl_union_map *isl_union_set_identity(
        __isl_take isl_union_set *uset);
__isl_give isl_union_pw_multi_aff *
isl_union_set_identity_union_pw_multi_aff(
        __isl_take isl_union_set *uset);
```

Construct an identity relation on the given (union) set.

- Function Extraction

A piecewise quasi affine expression that is equal to 1 on a set and 0 outside the set can be created using the following function.

```
#include <isl/aff.h>
__isl_give isl_pw_aff *isl_set_indicator_function(
        __isl_take isl_set *set);
```

A piecewise multiple quasi affine expression can be extracted from an `isl_set` or `isl_map`, provided the `isl_set` is a singleton and the `isl_map` is single-valued. In case of a conversion from an `isl_union_map` to an `isl_union_pw_multi_aff`,

these properties need to hold in each domain space. A conversion to a isl_multi_union_pw_aff additionally requires that the input is non-empty and involves only a single range space.

```
#include <isl/aff.h>
__isl_give isl_pw_multi_aff *isl_pw_multi_aff_from_set(
        __isl_take isl_set *set);
__isl_give isl_pw_multi_aff *isl_pw_multi_aff_from_map(
        __isl_take isl_map *map);

__isl_give isl_union_pw_multi_aff *
isl_union_pw_multi_aff_from_union_set(
        __isl_take isl_union_set *uset);
__isl_give isl_union_pw_multi_aff *
isl_union_pw_multi_aff_from_union_map(
        __isl_take isl_union_map *umap);

__isl_give isl_multi_union_pw_aff *
isl_multi_union_pw_aff_from_union_map(
        __isl_take isl_union_map *umap);
```

- Deltas

```
__isl_give isl_basic_set *isl_basic_map_deltas(
        __isl_take isl_basic_map *bmap);
__isl_give isl_set *isl_map_deltas(__isl_take isl_map *map);
__isl_give isl_union_set *isl_union_map_deltas(
        __isl_take isl_union_map *umap);
```

These functions return a (basic) set containing the differences between image elements and corresponding domain elements in the input.

```
__isl_give isl_basic_map *isl_basic_map_deltas_map(
        __isl_take isl_basic_map *bmap);
__isl_give isl_map *isl_map_deltas_map(
        __isl_take isl_map *map);
__isl_give isl_union_map *isl_union_map_deltas_map(
        __isl_take isl_union_map *umap);
```

The functions above construct a (basic, regular or union) relation that maps (a wrapped version of) the input relation to its delta set.

- Coalescing

Simplify the representation of a set, relation or functions by trying to combine pairs of basic sets or relations into a single basic set or relation.

```
#include <isl/set.h>
__isl_give isl_set *isl_set_coalesce(__isl_take isl_set *set);

#include <isl/map.h>
__isl_give isl_map *isl_map_coalesce(__isl_take isl_map *map);

#include <isl/union_set.h>
__isl_give isl_union_set *isl_union_set_coalesce(
        __isl_take isl_union_set *uset);

#include <isl/union_map.h>
__isl_give isl_union_map *isl_union_map_coalesce(
        __isl_take isl_union_map *umap);

#include <isl/aff.h>
__isl_give isl_pw_aff *isl_pw_aff_coalesce(
        __isl_take isl_pw_aff *pwqp);
__isl_give isl_pw_multi_aff *isl_pw_multi_aff_coalesce(
        __isl_take isl_pw_multi_aff *pma);
__isl_give isl_multi_pw_aff *isl_multi_pw_aff_coalesce(
        __isl_take isl_multi_pw_aff *mpa);
__isl_give isl_union_pw_aff *isl_union_pw_aff_coalesce(
        __isl_take isl_union_pw_aff *upa);
__isl_give isl_union_pw_multi_aff *
isl_union_pw_multi_aff_coalesce(
        __isl_take isl_union_pw_multi_aff *upma);
__isl_give isl_multi_union_pw_aff *
isl_multi_union_pw_aff_coalesce(
        __isl_take isl_multi_union_pw_aff *aff);

#include <isl/polynomial.h>
__isl_give isl_pw_qpolynomial_fold *
isl_pw_qpolynomial_fold_coalesce(
        __isl_take isl_pw_qpolynomial_fold *pwf);
__isl_give isl_union_pw_qpolynomial *
isl_union_pw_qpolynomial_coalesce(
        __isl_take isl_union_pw_qpolynomial *upwqp);
__isl_give isl_union_pw_qpolynomial_fold *
isl_union_pw_qpolynomial_fold_coalesce(
        __isl_take isl_union_pw_qpolynomial_fold *upwf);
```

One of the methods for combining pairs of basic sets or relations can result in coefficients that are much larger than those that appear in the constraints of the input. By default, the coefficients are not allowed to grow larger, but this can be changed by unsetting the following option.

```
isl_stat isl_options_set_coalesce_bounded_wrapping(
        isl_ctx *ctx, int val);
int isl_options_get_coalesce_bounded_wrapping(
        isl_ctx *ctx);
```

- Detecting equalities

```
__isl_give isl_basic_set *isl_basic_set_detect_equalities(
        __isl_take isl_basic_set *bset);
__isl_give isl_basic_map *isl_basic_map_detect_equalities(
        __isl_take isl_basic_map *bmap);
__isl_give isl_set *isl_set_detect_equalities(
        __isl_take isl_set *set);
__isl_give isl_map *isl_map_detect_equalities(
        __isl_take isl_map *map);
__isl_give isl_union_set *isl_union_set_detect_equalities(
        __isl_take isl_union_set *uset);
__isl_give isl_union_map *isl_union_map_detect_equalities(
        __isl_take isl_union_map *umap);
```

  Simplify the representation of a set or relation by detecting implicit equalities.

- Removing redundant constraints

```
#include <isl/set.h>
__isl_give isl_basic_set *isl_basic_set_remove_redundancies(
        __isl_take isl_basic_set *bset);
__isl_give isl_set *isl_set_remove_redundancies(
        __isl_take isl_set *set);

#include <isl/union_set.h>
__isl_give isl_union_set *
isl_union_set_remove_redundancies(
        __isl_take isl_union_set *uset);

#include <isl/map.h>
__isl_give isl_basic_map *isl_basic_map_remove_redundancies(
        __isl_take isl_basic_map *bmap);
__isl_give isl_map *isl_map_remove_redundancies(
        __isl_take isl_map *map);

#include <isl/union_map.h>
__isl_give isl_union_map *
isl_union_map_remove_redundancies(
        __isl_take isl_union_map *umap);
```

- Convex hull

```
__isl_give isl_basic_set *isl_set_convex_hull(
        __isl_take isl_set *set);
__isl_give isl_basic_map *isl_map_convex_hull(
        __isl_take isl_map *map);
```

  If the input set or relation has any existentially quantified variables, then the result of these operations is currently undefined.

- Simple hull

```
#include <isl/set.h>
__isl_give isl_basic_set *
isl_set_unshifted_simple_hull(
        __isl_take isl_set *set);
__isl_give isl_basic_set *isl_set_simple_hull(
        __isl_take isl_set *set);
__isl_give isl_basic_set *
isl_set_unshifted_simple_hull_from_set_list(
        __isl_take isl_set *set,
        __isl_take isl_set_list *list);

#include <isl/map.h>
__isl_give isl_basic_map *
isl_map_unshifted_simple_hull(
        __isl_take isl_map *map);
__isl_give isl_basic_map *isl_map_simple_hull(
        __isl_take isl_map *map);
__isl_give isl_basic_map *
isl_map_unshifted_simple_hull_from_map_list(
        __isl_take isl_map *map,
        __isl_take isl_map_list *list);

#include <isl/union_map.h>
__isl_give isl_union_map *isl_union_map_simple_hull(
        __isl_take isl_union_map *umap);
```

  These functions compute a single basic set or relation that contains the whole input set or relation. In particular, the output is described by translates of the constraints describing the basic sets or relations in the input. In case of `isl_set_unshifted_simple_hull`, only the original constraints are used, without any translation. In case of `isl_set_unshifted_simple_hull_fr` and `isl_map_unshifted_simple_hull_from_map_list`, the constraints are taken from the elements of the second argument.

  (See Section 2.2.)

- Affine hull

```
__isl_give isl_basic_set *isl_basic_set_affine_hull(
        __isl_take isl_basic_set *bset);
__isl_give isl_basic_set *isl_set_affine_hull(
        __isl_take isl_set *set);
__isl_give isl_union_set *isl_union_set_affine_hull(
        __isl_take isl_union_set *uset);
__isl_give isl_basic_map *isl_basic_map_affine_hull(
        __isl_take isl_basic_map *bmap);
__isl_give isl_basic_map *isl_map_affine_hull(
        __isl_take isl_map *map);
__isl_give isl_union_map *isl_union_map_affine_hull(
        __isl_take isl_union_map *umap);
```

  In case of union sets and relations, the affine hull is computed per space.

- Polyhedral hull

```
__isl_give isl_basic_set *isl_set_polyhedral_hull(
        __isl_take isl_set *set);
__isl_give isl_basic_map *isl_map_polyhedral_hull(
        __isl_take isl_map *map);
__isl_give isl_union_set *isl_union_set_polyhedral_hull(
        __isl_take isl_union_set *uset);
__isl_give isl_union_map *isl_union_map_polyhedral_hull(
        __isl_take isl_union_map *umap);
```

  These functions compute a single basic set or relation not involving any existentially quantified variables that contains the whole input set or relation. In case of union sets and relations, the polyhedral hull is computed per space.

- Other approximations

```
#include <isl/set.h>
__isl_give isl_basic_set *
isl_basic_set_drop_constraints_involving_dims(
        __isl_take isl_basic_set *bset,
        enum isl_dim_type type,
        unsigned first, unsigned n);
__isl_give isl_basic_set *
isl_basic_set_drop_constraints_not_involving_dims(
        __isl_take isl_basic_set *bset,
        enum isl_dim_type type,
        unsigned first, unsigned n);
__isl_give isl_set *
isl_set_drop_constraints_involving_dims(
```

```
        __isl_take isl_set *set,
        enum isl_dim_type type,
        unsigned first, unsigned n);

#include <isl/map.h>
__isl_give isl_basic_map *
isl_basic_map_drop_constraints_involving_dims(
        __isl_take isl_basic_map *bmap,
        enum isl_dim_type type,
        unsigned first, unsigned n);
__isl_give isl_basic_map *
isl_basic_map_drop_constraints_not_involving_dims(
        __isl_take isl_basic_map *bmap,
        enum isl_dim_type type,
        unsigned first, unsigned n);
__isl_give isl_map *
isl_map_drop_constraints_involving_dims(
        __isl_take isl_map *map,
        enum isl_dim_type type,
        unsigned first, unsigned n);
```

These functions drop any constraints (not) involving the specified dimensions. Note that the result depends on the representation of the input.

```
#include <isl/polynomial.h>
__isl_give isl_pw_qpolynomial *isl_pw_qpolynomial_to_polynomial(
        __isl_take isl_pw_qpolynomial *pwqp, int sign);
__isl_give isl_union_pw_qpolynomial *
isl_union_pw_qpolynomial_to_polynomial(
        __isl_take isl_union_pw_qpolynomial *upwqp, int sign);
```

Approximate each quasipolynomial by a polynomial. If `sign` is positive, the polynomial will be an overapproximation. If `sign` is negative, it will be an underapproximation. If `sign` is zero, the approximation will lie somewhere in between.

- Feasibility

```
__isl_give isl_basic_set *isl_basic_set_sample(
        __isl_take isl_basic_set *bset);
__isl_give isl_basic_set *isl_set_sample(
        __isl_take isl_set *set);
__isl_give isl_basic_map *isl_basic_map_sample(
        __isl_take isl_basic_map *bmap);
__isl_give isl_basic_map *isl_map_sample(
        __isl_take isl_map *map);
```

If the input (basic) set or relation is non-empty, then return a singleton subset of the input. Otherwise, return an empty set.

- Optimization

```
#include <isl/ilp.h>
__isl_give isl_val *isl_basic_set_max_val(
        __isl_keep isl_basic_set *bset,
        __isl_keep isl_aff *obj);
__isl_give isl_val *isl_set_min_val(
        __isl_keep isl_set *set,
        __isl_keep isl_aff *obj);
__isl_give isl_val *isl_set_max_val(
        __isl_keep isl_set *set,
        __isl_keep isl_aff *obj);
__isl_give isl_multi_val *
isl_union_set_min_multi_union_pw_aff(
        __isl_keep isl_union_set *set,
        __isl_keep isl_multi_union_pw_aff *obj);
```

Compute the minimum or maximum of the integer affine expression `obj` over the points in `set`, returning the result in `opt`. The result is `NULL` in case of an error, the optimal value in case there is one, negative infinity or infinity if the problem is unbounded and NaN if the problem is empty.

- Parametric optimization

```
__isl_give isl_pw_aff *isl_set_dim_min(
        __isl_take isl_set *set, int pos);
__isl_give isl_pw_aff *isl_set_dim_max(
        __isl_take isl_set *set, int pos);
__isl_give isl_pw_aff *isl_map_dim_max(
        __isl_take isl_map *map, int pos);
```

Compute the minimum or maximum of the given set or output dimension as a function of the parameters (and input dimensions), but independently of the other set or output dimensions. For lexicographic optimization, see §1.4.17.

- Dual

The following functions compute either the set of (rational) coefficient values of valid constraints for the given set or the set of (rational) values satisfying the constraints with coefficients from the given set. Internally, these two sets of functions perform essentially the same operations, except that the set of coefficients is assumed to be a cone, while the set of values may be any polyhedron. The current implementation is based on the Farkas lemma and Fourier-Motzkin elimination, but this may change or be made optional in future. In particular, future implementations may use different dualization algorithms or skip the elimination step.

```
__isl_give isl_basic_set *isl_basic_set_coefficients(
        __isl_take isl_basic_set *bset);
__isl_give isl_basic_set *isl_set_coefficients(
        __isl_take isl_set *set);
__isl_give isl_union_set *isl_union_set_coefficients(
        __isl_take isl_union_set *bset);
__isl_give isl_basic_set *isl_basic_set_solutions(
        __isl_take isl_basic_set *bset);
__isl_give isl_basic_set *isl_set_solutions(
        __isl_take isl_set *set);
__isl_give isl_union_set *isl_union_set_solutions(
        __isl_take isl_union_set *bset);
```

- Power

```
__isl_give isl_map *isl_map_fixed_power_val(
        __isl_take isl_map *map,
        __isl_take isl_val *exp);
__isl_give isl_union_map *
isl_union_map_fixed_power_val(
        __isl_take isl_union_map *umap,
        __isl_take isl_val *exp);
```

Compute the given power of `map`, where `exp` is assumed to be non-zero. If the exponent `exp` is negative, then the -`exp` th power of the inverse of `map` is computed.

```
__isl_give isl_map *isl_map_power(__isl_take isl_map *map,
        int *exact);
__isl_give isl_union_map *isl_union_map_power(
        __isl_take isl_union_map *umap, int *exact);
```

Compute a parametric representation for all positive powers k of `map`. The result maps k to a nested relation corresponding to the kth power of `map`. The result may be an overapproximation. If the result is known to be exact, then `*exact` is set to 1.

- Transitive closure

```
__isl_give isl_map *isl_map_transitive_closure(
        __isl_take isl_map *map, int *exact);
__isl_give isl_union_map *isl_union_map_transitive_closure(
        __isl_take isl_union_map *umap, int *exact);
```

Compute the transitive closure of `map`. The result may be an overapproximation. If the result is known to be exact, then `*exact` is set to 1.

- Reaching path lengths

```
__isl_give isl_map *isl_map_reaching_path_lengths(
        __isl_take isl_map *map, int *exact);
```

  Compute a relation that maps each element in the range of `map` to the lengths of all paths composed of edges in `map` that end up in the given element. The result may be an overapproximation. If the result is known to be exact, then `*exact` is set to 1. To compute the *maximal* path length, the resulting relation should be postprocessed by `isl_map_lexmax`. In particular, if the input relation is a dependence relation (mapping sources to sinks), then the maximal path length corresponds to the free schedule. Note, however, that `isl_map_lexmax` expects the maximum to be finite, so if the path lengths are unbounded (possibly due to the overapproximation), then you will get an error message.

- Wrapping

```
#include <isl/space.h>
__isl_give isl_space *isl_space_wrap(
        __isl_take isl_space *space);
__isl_give isl_space *isl_space_unwrap(
        __isl_take isl_space *space);

#include <isl/local_space.h>
__isl_give isl_local_space *isl_local_space_wrap(
        __isl_take isl_local_space *ls);

#include <isl/set.h>
__isl_give isl_basic_map *isl_basic_set_unwrap(
        __isl_take isl_basic_set *bset);
__isl_give isl_map *isl_set_unwrap(
        __isl_take isl_set *set);

#include <isl/map.h>
__isl_give isl_basic_set *isl_basic_map_wrap(
        __isl_take isl_basic_map *bmap);
__isl_give isl_set *isl_map_wrap(
        __isl_take isl_map *map);

#include <isl/union_set.h>
__isl_give isl_union_map *isl_union_set_unwrap(
        __isl_take isl_union_set *uset);

#include <isl/union_map.h>
__isl_give isl_union_set *isl_union_map_wrap(
        __isl_take isl_union_map *umap);
```

The input to `isl_space_unwrap` should be the space of a set, while that of `isl_space_wrap` should be the space of a relation. Conversely, the output of `isl_space_unwrap` is the space of a relation, while that of `isl_space_wrap` is the space of a set.

- Flattening

  Remove any internal structure of domain (and range) of the given set or relation. If there is any such internal structure in the input, then the name of the space is also removed.

```
#include <isl/local_space.h>
__isl_give isl_local_space *
isl_local_space_flatten_domain(
        __isl_take isl_local_space *ls);
__isl_give isl_local_space *
isl_local_space_flatten_range(
        __isl_take isl_local_space *ls);

#include <isl/set.h>
__isl_give isl_basic_set *isl_basic_set_flatten(
        __isl_take isl_basic_set *bset);
__isl_give isl_set *isl_set_flatten(
        __isl_take isl_set *set);

#include <isl/map.h>
__isl_give isl_basic_map *isl_basic_map_flatten_domain(
        __isl_take isl_basic_map *bmap);
__isl_give isl_basic_map *isl_basic_map_flatten_range(
        __isl_take isl_basic_map *bmap);
__isl_give isl_map *isl_map_flatten_range(
        __isl_take isl_map *map);
__isl_give isl_map *isl_map_flatten_domain(
        __isl_take isl_map *map);
__isl_give isl_basic_map *isl_basic_map_flatten(
        __isl_take isl_basic_map *bmap);
__isl_give isl_map *isl_map_flatten(
        __isl_take isl_map *map);

#include <isl/val.h>
__isl_give isl_multi_val *isl_multi_val_flatten_range(
        __isl_take isl_multi_val *mv);

#include <isl/aff.h>
__isl_give isl_multi_aff *isl_multi_aff_flatten_domain(
        __isl_take isl_multi_aff *ma);
```

```
__isl_give isl_multi_aff *isl_multi_aff_flatten_range(
        __isl_take isl_multi_aff *ma);
__isl_give isl_multi_pw_aff *
isl_multi_pw_aff_flatten_range(
        __isl_take isl_multi_pw_aff *mpa);
__isl_give isl_multi_union_pw_aff *
isl_multi_union_pw_aff_flatten_range(
        __isl_take isl_multi_union_pw_aff *mupa);

#include <isl/map.h>
__isl_give isl_map *isl_set_flatten_map(
        __isl_take isl_set *set);
```

The function above constructs a relation that maps the input set to a flattened version of the set.

- Lifting

  Lift the input set to a space with extra dimensions corresponding to the existentially quantified variables in the input. In particular, the result lives in a wrapped map where the domain is the original space and the range corresponds to the original existentially quantified variables.

```
#include <isl/set.h>
__isl_give isl_basic_set *isl_basic_set_lift(
        __isl_take isl_basic_set *bset);
__isl_give isl_set *isl_set_lift(
        __isl_take isl_set *set);
__isl_give isl_union_set *isl_union_set_lift(
        __isl_take isl_union_set *uset);
```

  Given a local space that contains the existentially quantified variables of a set, a basic relation that, when applied to a basic set, has essentially the same effect as `isl_basic_set_lift`, can be constructed using the following function.

```
#include <isl/local_space.h>
__isl_give isl_basic_map *isl_local_space_lifting(
        __isl_take isl_local_space *ls);

#include <isl/aff.h>
__isl_give isl_multi_aff *isl_multi_aff_lift(
        __isl_take isl_multi_aff *maff,
        __isl_give isl_local_space **ls);
```

  If the `ls` argument of `isl_multi_aff_lift` is not `NULL`, then it is assigned the local space that lies at the basis of the lifting applied.

- Internal Product

```
#include <isl/space.h>
__isl_give isl_space *isl_space_zip(
        __isl_take isl_space *space);

#include <isl/map.h>
__isl_give isl_basic_map *isl_basic_map_zip(
        __isl_take isl_basic_map *bmap);
__isl_give isl_map *isl_map_zip(
        __isl_take isl_map *map);

#include <isl/union_map.h>
__isl_give isl_union_map *isl_union_map_zip(
        __isl_take isl_union_map *umap);
```

  Given a relation with nested relations for domain and range, interchange the range of the domain with the domain of the range.

- Currying

```
#include <isl/space.h>
__isl_give isl_space *isl_space_curry(
        __isl_take isl_space *space);
__isl_give isl_space *isl_space_uncurry(
        __isl_take isl_space *space);

#include <isl/map.h>
__isl_give isl_basic_map *isl_basic_map_curry(
        __isl_take isl_basic_map *bmap);
__isl_give isl_basic_map *isl_basic_map_uncurry(
        __isl_take isl_basic_map *bmap);
__isl_give isl_map *isl_map_curry(
        __isl_take isl_map *map);
__isl_give isl_map *isl_map_uncurry(
        __isl_take isl_map *map);

#include <isl/union_map.h>
__isl_give isl_union_map *isl_union_map_curry(
        __isl_take isl_union_map *umap);
__isl_give isl_union_map *isl_union_map_uncurry(
        __isl_take isl_union_map *umap);
```

  Given a relation with a nested relation for domain, the `curry` functions move the range of the nested relation out of the domain and use it as the domain of a nested relation in the range, with the original range as range of this nested relation. The `uncurry` functions perform the inverse operation.

```
#include <isl/space.h>
__isl_give isl_space *isl_space_range_curry(
        __isl_take isl_space *space);

#include <isl/map.h>
__isl_give isl_map *isl_map_range_curry(
        __isl_take isl_map *map);

#include <isl/union_map.h>
__isl_give isl_union_map *isl_union_map_range_curry(
        __isl_take isl_union_map *umap);
```

These functions apply the currying to the relation that is nested inside the range of the input.

- Aligning parameters

  Change the order of the parameters of the given set, relation or function such that the first parameters match those of `model`. This may involve the introduction of extra parameters. All parameters need to be named.

```
#include <isl/space.h>
__isl_give isl_space *isl_space_align_params(
        __isl_take isl_space *space1,
        __isl_take isl_space *space2)

#include <isl/set.h>
__isl_give isl_basic_set *isl_basic_set_align_params(
        __isl_take isl_basic_set *bset,
        __isl_take isl_space *model);
__isl_give isl_set *isl_set_align_params(
        __isl_take isl_set *set,
        __isl_take isl_space *model);

#include <isl/map.h>
__isl_give isl_basic_map *isl_basic_map_align_params(
        __isl_take isl_basic_map *bmap,
        __isl_take isl_space *model);
__isl_give isl_map *isl_map_align_params(
        __isl_take isl_map *map,
        __isl_take isl_space *model);

#include <isl/val.h>
__isl_give isl_multi_val *isl_multi_val_align_params(
        __isl_take isl_multi_val *mv,
        __isl_take isl_space *model);
```

```
#include <isl/aff.h>
__isl_give isl_aff *isl_aff_align_params(
        __isl_take isl_aff *aff,
        __isl_take isl_space *model);
__isl_give isl_multi_aff *isl_multi_aff_align_params(
        __isl_take isl_multi_aff *multi,
        __isl_take isl_space *model);
__isl_give isl_pw_aff *isl_pw_aff_align_params(
        __isl_take isl_pw_aff *pwaff,
        __isl_take isl_space *model);
__isl_give isl_pw_multi_aff *isl_pw_multi_aff_align_params(
        __isl_take isl_pw_multi_aff *pma,
        __isl_take isl_space *model);
__isl_give isl_union_pw_aff *
isl_union_pw_aff_align_params(
        __isl_take isl_union_pw_aff *upa,
        __isl_take isl_space *model);
__isl_give isl_union_pw_multi_aff *
isl_union_pw_multi_aff_align_params(
        __isl_take isl_union_pw_multi_aff *upma,
        __isl_take isl_space *model);
__isl_give isl_multi_union_pw_aff *
isl_multi_union_pw_aff_align_params(
        __isl_take isl_multi_union_pw_aff *mupa,
        __isl_take isl_space *model);

#include <isl/polynomial.h>
__isl_give isl_qpolynomial *isl_qpolynomial_align_params(
        __isl_take isl_qpolynomial *qp,
        __isl_take isl_space *model);
```

- Unary Arithmetic Operations

```
#include <isl/set.h>
__isl_give isl_set *isl_set_neg(
        __isl_take isl_set *set);
#include <isl/map.h>
__isl_give isl_map *isl_map_neg(
        __isl_take isl_map *map);
```

`isl_set_neg` constructs a set containing the opposites of the elements in its argument. The domain of the result of `isl_map_neg` is the same as the domain of its argument. The corresponding range elements are the opposites of the corresponding range elements in the argument.

```
#include <isl/val.h>
```

```
__isl_give isl_multi_val *isl_multi_val_neg(
        __isl_take isl_multi_val *mv);

#include <isl/aff.h>
__isl_give isl_aff *isl_aff_neg(
        __isl_take isl_aff *aff);
__isl_give isl_multi_aff *isl_multi_aff_neg(
        __isl_take isl_multi_aff *ma);
__isl_give isl_pw_aff *isl_pw_aff_neg(
        __isl_take isl_pw_aff *pwaff);
__isl_give isl_pw_multi_aff *isl_pw_multi_aff_neg(
        __isl_take isl_pw_multi_aff *pma);
__isl_give isl_multi_pw_aff *isl_multi_pw_aff_neg(
        __isl_take isl_multi_pw_aff *mpa);
__isl_give isl_union_pw_aff *isl_union_pw_aff_neg(
        __isl_take isl_union_pw_aff *upa);
__isl_give isl_union_pw_multi_aff *
isl_union_pw_multi_aff_neg(
        __isl_take isl_union_pw_multi_aff *upma);
__isl_give isl_multi_union_pw_aff *
isl_multi_union_pw_aff_neg(
        __isl_take isl_multi_union_pw_aff *mupa);
__isl_give isl_aff *isl_aff_ceil(
        __isl_take isl_aff *aff);
__isl_give isl_pw_aff *isl_pw_aff_ceil(
        __isl_take isl_pw_aff *pwaff);
__isl_give isl_aff *isl_aff_floor(
        __isl_take isl_aff *aff);
__isl_give isl_multi_aff *isl_multi_aff_floor(
        __isl_take isl_multi_aff *ma);
__isl_give isl_pw_aff *isl_pw_aff_floor(
        __isl_take isl_pw_aff *pwaff);
__isl_give isl_union_pw_aff *isl_union_pw_aff_floor(
        __isl_take isl_union_pw_aff *upa);
__isl_give isl_multi_union_pw_aff *
isl_multi_union_pw_aff_floor(
        __isl_take isl_multi_union_pw_aff *mupa);

#include <isl/aff.h>
__isl_give isl_pw_aff *isl_pw_aff_list_min(
        __isl_take isl_pw_aff_list *list);
__isl_give isl_pw_aff *isl_pw_aff_list_max(
        __isl_take isl_pw_aff_list *list);

#include <isl/polynomial.h>
__isl_give isl_qpolynomial *isl_qpolynomial_neg(
```

```
        __isl_take isl_qpolynomial *qp);
__isl_give isl_pw_qpolynomial *isl_pw_qpolynomial_neg(
        __isl_take isl_pw_qpolynomial *pwqp);
__isl_give isl_union_pw_qpolynomial *
isl_union_pw_qpolynomial_neg(
        __isl_take isl_union_pw_qpolynomial *upwqp);
__isl_give isl_qpolynomial *isl_qpolynomial_pow(
        __isl_take isl_qpolynomial *qp,
        unsigned exponent);
__isl_give isl_pw_qpolynomial *isl_pw_qpolynomial_pow(
        __isl_take isl_pw_qpolynomial *pwqp,
        unsigned exponent);
```

- Evaluation

  The following functions evaluate a function in a point.

```
#include <isl/polynomial.h>
__isl_give isl_val *isl_pw_qpolynomial_eval(
        __isl_take isl_pw_qpolynomial *pwqp,
        __isl_take isl_point *pnt);
__isl_give isl_val *isl_pw_qpolynomial_fold_eval(
        __isl_take isl_pw_qpolynomial_fold *pwf,
        __isl_take isl_point *pnt);
__isl_give isl_val *isl_union_pw_qpolynomial_eval(
        __isl_take isl_union_pw_qpolynomial *upwqp,
        __isl_take isl_point *pnt);
__isl_give isl_val *isl_union_pw_qpolynomial_fold_eval(
        __isl_take isl_union_pw_qpolynomial_fold *upwf,
        __isl_take isl_point *pnt);
```

- Dimension manipulation

  It is usually not advisable to directly change the (input or output) space of a set or a relation as this removes the name and the internal structure of the space. However, the functions below can be useful to add new parameters, assuming `isl_set_align_params` and `isl_map_align_params` are not sufficient.

```
#include <isl/space.h>
__isl_give isl_space *isl_space_add_dims(
        __isl_take isl_space *space,
        enum isl_dim_type type, unsigned n);
__isl_give isl_space *isl_space_insert_dims(
        __isl_take isl_space *space,
        enum isl_dim_type type, unsigned pos, unsigned n);
__isl_give isl_space *isl_space_drop_dims(
        __isl_take isl_space *space,
```

```
        enum isl_dim_type type, unsigned first, unsigned n);
__isl_give isl_space *isl_space_move_dims(
        __isl_take isl_space *space,
        enum isl_dim_type dst_type, unsigned dst_pos,
        enum isl_dim_type src_type, unsigned src_pos,
        unsigned n);

#include <isl/local_space.h>
__isl_give isl_local_space *isl_local_space_add_dims(
        __isl_take isl_local_space *ls,
        enum isl_dim_type type, unsigned n);
__isl_give isl_local_space *isl_local_space_insert_dims(
        __isl_take isl_local_space *ls,
        enum isl_dim_type type, unsigned first, unsigned n);
__isl_give isl_local_space *isl_local_space_drop_dims(
        __isl_take isl_local_space *ls,
        enum isl_dim_type type, unsigned first, unsigned n);

#include <isl/set.h>
__isl_give isl_basic_set *isl_basic_set_add_dims(
        __isl_take isl_basic_set *bset,
        enum isl_dim_type type, unsigned n);
__isl_give isl_set *isl_set_add_dims(
        __isl_take isl_set *set,
        enum isl_dim_type type, unsigned n);
__isl_give isl_basic_set *isl_basic_set_insert_dims(
        __isl_take isl_basic_set *bset,
        enum isl_dim_type type, unsigned pos,
        unsigned n);
__isl_give isl_set *isl_set_insert_dims(
        __isl_take isl_set *set,
        enum isl_dim_type type, unsigned pos, unsigned n);
__isl_give isl_basic_set *isl_basic_set_move_dims(
        __isl_take isl_basic_set *bset,
        enum isl_dim_type dst_type, unsigned dst_pos,
        enum isl_dim_type src_type, unsigned src_pos,
        unsigned n);
__isl_give isl_set *isl_set_move_dims(
        __isl_take isl_set *set,
        enum isl_dim_type dst_type, unsigned dst_pos,
        enum isl_dim_type src_type, unsigned src_pos,
        unsigned n);

#include <isl/map.h>
__isl_give isl_basic_map *isl_basic_map_add_dims(
        __isl_take isl_basic_map *bmap,
```

```
        enum isl_dim_type type, unsigned n);
__isl_give isl_map *isl_map_add_dims(
        __isl_take isl_map *map,
        enum isl_dim_type type, unsigned n);
__isl_give isl_basic_map *isl_basic_map_insert_dims(
        __isl_take isl_basic_map *bmap,
        enum isl_dim_type type, unsigned pos,
        unsigned n);
__isl_give isl_map *isl_map_insert_dims(
        __isl_take isl_map *map,
        enum isl_dim_type type, unsigned pos, unsigned n);
__isl_give isl_basic_map *isl_basic_map_move_dims(
        __isl_take isl_basic_map *bmap,
        enum isl_dim_type dst_type, unsigned dst_pos,
        enum isl_dim_type src_type, unsigned src_pos,
        unsigned n);
__isl_give isl_map *isl_map_move_dims(
        __isl_take isl_map *map,
        enum isl_dim_type dst_type, unsigned dst_pos,
        enum isl_dim_type src_type, unsigned src_pos,
        unsigned n);

#include <isl/val.h>
__isl_give isl_multi_val *isl_multi_val_insert_dims(
        __isl_take isl_multi_val *mv,
        enum isl_dim_type type, unsigned first, unsigned n);
__isl_give isl_multi_val *isl_multi_val_add_dims(
        __isl_take isl_multi_val *mv,
        enum isl_dim_type type, unsigned n);
__isl_give isl_multi_val *isl_multi_val_drop_dims(
        __isl_take isl_multi_val *mv,
        enum isl_dim_type type, unsigned first, unsigned n);

#include <isl/aff.h>
__isl_give isl_aff *isl_aff_insert_dims(
        __isl_take isl_aff *aff,
        enum isl_dim_type type, unsigned first, unsigned n);
__isl_give isl_multi_aff *isl_multi_aff_insert_dims(
        __isl_take isl_multi_aff *ma,
        enum isl_dim_type type, unsigned first, unsigned n);
__isl_give isl_pw_aff *isl_pw_aff_insert_dims(
        __isl_take isl_pw_aff *pwaff,
        enum isl_dim_type type, unsigned first, unsigned n);
__isl_give isl_multi_pw_aff *isl_multi_pw_aff_insert_dims(
        __isl_take isl_multi_pw_aff *mpa,
        enum isl_dim_type type, unsigned first, unsigned n);
```

```
__isl_give isl_aff *isl_aff_add_dims(
        __isl_take isl_aff *aff,
        enum isl_dim_type type, unsigned n);
__isl_give isl_multi_aff *isl_multi_aff_add_dims(
        __isl_take isl_multi_aff *ma,
        enum isl_dim_type type, unsigned n);
__isl_give isl_pw_aff *isl_pw_aff_add_dims(
        __isl_take isl_pw_aff *pwaff,
        enum isl_dim_type type, unsigned n);
__isl_give isl_multi_pw_aff *isl_multi_pw_aff_add_dims(
        __isl_take isl_multi_pw_aff *mpa,
        enum isl_dim_type type, unsigned n);
__isl_give isl_aff *isl_aff_drop_dims(
        __isl_take isl_aff *aff,
        enum isl_dim_type type, unsigned first, unsigned n);
__isl_give isl_multi_aff *isl_multi_aff_drop_dims(
        __isl_take isl_multi_aff *maff,
        enum isl_dim_type type, unsigned first, unsigned n);
__isl_give isl_pw_aff *isl_pw_aff_drop_dims(
        __isl_take isl_pw_aff *pwaff,
        enum isl_dim_type type, unsigned first, unsigned n);
__isl_give isl_pw_multi_aff *isl_pw_multi_aff_drop_dims(
        __isl_take isl_pw_multi_aff *pma,
        enum isl_dim_type type, unsigned first, unsigned n);
__isl_give isl_union_pw_aff *isl_union_pw_aff_drop_dims(
        __isl_take isl_union_pw_aff *upa,
        enum isl_dim_type type, unsigned first, unsigned n);
__isl_give isl_union_pw_multi_aff *
        isl_union_pw_multi_aff_drop_dims(
        __isl_take isl_union_pw_multi_aff *upma,
        enum isl_dim_type type,
        unsigned first, unsigned n);
__isl_give isl_multi_union_pw_aff *
isl_multi_union_pw_aff_drop_dims(
        __isl_take isl_multi_union_pw_aff *mupa,
        enum isl_dim_type type, unsigned first,
        unsigned n);
__isl_give isl_aff *isl_aff_move_dims(
        __isl_take isl_aff *aff,
        enum isl_dim_type dst_type, unsigned dst_pos,
        enum isl_dim_type src_type, unsigned src_pos,
        unsigned n);
__isl_give isl_multi_aff *isl_multi_aff_move_dims(
        __isl_take isl_multi_aff *ma,
        enum isl_dim_type dst_type, unsigned dst_pos,
        enum isl_dim_type src_type, unsigned src_pos,
```

```
        unsigned n);
__isl_give isl_pw_aff *isl_pw_aff_move_dims(
        __isl_take isl_pw_aff *pa,
        enum isl_dim_type dst_type, unsigned dst_pos,
        enum isl_dim_type src_type, unsigned src_pos,
        unsigned n);
__isl_give isl_multi_pw_aff *isl_multi_pw_aff_move_dims(
        __isl_take isl_multi_pw_aff *pma,
        enum isl_dim_type dst_type, unsigned dst_pos,
        enum isl_dim_type src_type, unsigned src_pos,
        unsigned n);

#include <isl/polynomial.h>
__isl_give isl_union_pw_qpolynomial *
isl_union_pw_qpolynomial_drop_dims(
        __isl_take isl_union_pw_qpolynomial *upwqp,
        enum isl_dim_type type,
        unsigned first, unsigned n);
__isl_give isl_union_pw_qpolynomial_fold *
        isl_union_pw_qpolynomial_fold_drop_dims(
        __isl_take isl_union_pw_qpolynomial_fold *upwf,
        enum isl_dim_type type,
        unsigned first, unsigned n);
```

The operations on union expressions can only manipulate parameters.

1.4.17 Binary Operations

The two arguments of a binary operation not only need to live in the same `isl_ctx`, they currently also need to have the same (number of) parameters.

Basic Operations

- Intersection

```
#include <isl/local_space.h>
__isl_give isl_local_space *isl_local_space_intersect(
        __isl_take isl_local_space *ls1,
        __isl_take isl_local_space *ls2);

#include <isl/set.h>
__isl_give isl_basic_set *isl_basic_set_intersect_params(
        __isl_take isl_basic_set *bset1,
        __isl_take isl_basic_set *bset2);
__isl_give isl_basic_set *isl_basic_set_intersect(
        __isl_take isl_basic_set *bset1,
```

```
        __isl_take isl_basic_set *bset2);
__isl_give isl_basic_set *isl_basic_set_list_intersect(
        __isl_take struct isl_basic_set_list *list);
__isl_give isl_set *isl_set_intersect_params(
        __isl_take isl_set *set,
        __isl_take isl_set *params);
__isl_give isl_set *isl_set_intersect(
        __isl_take isl_set *set1,
        __isl_take isl_set *set2);

#include <isl/map.h>
__isl_give isl_basic_map *isl_basic_map_intersect_domain(
        __isl_take isl_basic_map *bmap,
        __isl_take isl_basic_set *bset);
__isl_give isl_basic_map *isl_basic_map_intersect_range(
        __isl_take isl_basic_map *bmap,
        __isl_take isl_basic_set *bset);
__isl_give isl_basic_map *isl_basic_map_intersect(
        __isl_take isl_basic_map *bmap1,
        __isl_take isl_basic_map *bmap2);
__isl_give isl_basic_map *isl_basic_map_list_intersect(
        __isl_take isl_basic_map_list *list);
__isl_give isl_map *isl_map_intersect_params(
        __isl_take isl_map *map,
        __isl_take isl_set *params);
__isl_give isl_map *isl_map_intersect_domain(
        __isl_take isl_map *map,
        __isl_take isl_set *set);
__isl_give isl_map *isl_map_intersect_range(
        __isl_take isl_map *map,
        __isl_take isl_set *set);
__isl_give isl_map *isl_map_intersect(
        __isl_take isl_map *map1,
        __isl_take isl_map *map2);

#include <isl/union_set.h>
__isl_give isl_union_set *isl_union_set_intersect_params(
        __isl_take isl_union_set *uset,
        __isl_take isl_set *set);
__isl_give isl_union_set *isl_union_set_intersect(
        __isl_take isl_union_set *uset1,
        __isl_take isl_union_set *uset2);

#include <isl/union_map.h>
__isl_give isl_union_map *isl_union_map_intersect_params(
        __isl_take isl_union_map *umap,
```

```
        __isl_take isl_set *set);
__isl_give isl_union_map *isl_union_map_intersect_domain(
        __isl_take isl_union_map *umap,
        __isl_take isl_union_set *uset);
__isl_give isl_union_map *isl_union_map_intersect_range(
        __isl_take isl_union_map *umap,
        __isl_take isl_union_set *uset);
__isl_give isl_union_map *isl_union_map_intersect(
        __isl_take isl_union_map *umap1,
        __isl_take isl_union_map *umap2);

#include <isl/aff.h>
__isl_give isl_pw_aff *isl_pw_aff_intersect_domain(
        __isl_take isl_pw_aff *pa,
        __isl_take isl_set *set);
__isl_give isl_multi_pw_aff *
isl_multi_pw_aff_intersect_domain(
        __isl_take isl_multi_pw_aff *mpa,
        __isl_take isl_set *domain);
__isl_give isl_pw_multi_aff *isl_pw_multi_aff_intersect_domain(
        __isl_take isl_pw_multi_aff *pma,
        __isl_take isl_set *set);
__isl_give isl_union_pw_aff *isl_union_pw_aff_intersect_domain(
        __isl_take isl_union_pw_aff *upa,
        __isl_take isl_union_set *uset);
__isl_give isl_union_pw_multi_aff *
isl_union_pw_multi_aff_intersect_domain(
        __isl_take isl_union_pw_multi_aff *upma,
        __isl_take isl_union_set *uset);
__isl_give isl_multi_union_pw_aff *
isl_multi_union_pw_aff_intersect_domain(
        __isl_take isl_multi_union_pw_aff *mupa,
        __isl_take isl_union_set *uset);
__isl_give isl_pw_aff *isl_pw_aff_intersect_params(
        __isl_take isl_pw_aff *pa,
        __isl_take isl_set *set);
__isl_give isl_multi_pw_aff *
isl_multi_pw_aff_intersect_params(
        __isl_take isl_multi_pw_aff *mpa,
        __isl_take isl_set *set);
__isl_give isl_pw_multi_aff *isl_pw_multi_aff_intersect_params(
        __isl_take isl_pw_multi_aff *pma,
        __isl_take isl_set *set);
__isl_give isl_union_pw_aff *
isl_union_pw_aff_intersect_params(
        __isl_take isl_union_pw_aff *upa,
```

```
__isl_give isl_union_pw_multi_aff *
isl_union_pw_multi_aff_intersect_params(
        __isl_take isl_union_pw_multi_aff *upma,
        __isl_take isl_set *set);
__isl_give isl_multi_union_pw_aff *
isl_multi_union_pw_aff_intersect_params(
        __isl_take isl_multi_union_pw_aff *mupa,
        __isl_take isl_set *params);
isl_multi_union_pw_aff_intersect_range(
        __isl_take isl_multi_union_pw_aff *mupa,
        __isl_take isl_set *set);

#include <isl/polynomial.h>
__isl_give isl_pw_qpolynomial *
isl_pw_qpolynomial_intersect_domain(
        __isl_take isl_pw_qpolynomial *pwpq,
        __isl_take isl_set *set);
__isl_give isl_union_pw_qpolynomial *
isl_union_pw_qpolynomial_intersect_domain(
        __isl_take isl_union_pw_qpolynomial *upwpq,
        __isl_take isl_union_set *uset);
__isl_give isl_union_pw_qpolynomial_fold *
isl_union_pw_qpolynomial_fold_intersect_domain(
        __isl_take isl_union_pw_qpolynomial_fold *upwf,
        __isl_take isl_union_set *uset);
__isl_give isl_pw_qpolynomial *
isl_pw_qpolynomial_intersect_params(
        __isl_take isl_pw_qpolynomial *pwpq,
        __isl_take isl_set *set);
__isl_give isl_pw_qpolynomial_fold *
isl_pw_qpolynomial_fold_intersect_params(
        __isl_take isl_pw_qpolynomial_fold *pwf,
        __isl_take isl_set *set);
__isl_give isl_union_pw_qpolynomial *
isl_union_pw_qpolynomial_intersect_params(
        __isl_take isl_union_pw_qpolynomial *upwpq,
        __isl_take isl_set *set);
__isl_give isl_union_pw_qpolynomial_fold *
isl_union_pw_qpolynomial_fold_intersect_params(
        __isl_take isl_union_pw_qpolynomial_fold *upwf,
        __isl_take isl_set *set);
```

The second argument to the `_params` functions needs to be a parametric (basic) set. For the other functions, a parametric set for either argument is only allowed if the other argument is a parametric set as well. The list passed to `isl_basic_set_list_intersect` needs to have at least one element and all

elements need to live in the same space. The function `isl_multi_union_pw_aff_intersect_range` restricts the input function to those shared domain elements that map to the specified range.

- Union

```
#include <isl/set.h>
__isl_give isl_set *isl_basic_set_union(
        __isl_take isl_basic_set *bset1,
        __isl_take isl_basic_set *bset2);
__isl_give isl_set *isl_set_union(
        __isl_take isl_set *set1,
        __isl_take isl_set *set2);
__isl_give isl_set *isl_set_list_union(
        __isl_take isl_set_list *list);

#include <isl/map.h>
__isl_give isl_map *isl_basic_map_union(
        __isl_take isl_basic_map *bmap1,
        __isl_take isl_basic_map *bmap2);
__isl_give isl_map *isl_map_union(
        __isl_take isl_map *map1,
        __isl_take isl_map *map2);

#include <isl/union_set.h>
__isl_give isl_union_set *isl_union_set_union(
        __isl_take isl_union_set *uset1,
        __isl_take isl_union_set *uset2);
__isl_give isl_union_set *isl_union_set_list_union(
        __isl_take isl_union_set_list *list);

#include <isl/union_map.h>
__isl_give isl_union_map *isl_union_map_union(
        __isl_take isl_union_map *umap1,
        __isl_take isl_union_map *umap2);
```

The list passed to `isl_set_list_union` needs to have at least one element and all elements need to live in the same space.

- Set difference

```
#include <isl/set.h>
__isl_give isl_set *isl_set_subtract(
        __isl_take isl_set *set1,
        __isl_take isl_set *set2);
```

```
#include <isl/map.h>
__isl_give isl_map *isl_map_subtract(
        __isl_take isl_map *map1,
        __isl_take isl_map *map2);
__isl_give isl_map *isl_map_subtract_domain(
        __isl_take isl_map *map,
        __isl_take isl_set *dom);
__isl_give isl_map *isl_map_subtract_range(
        __isl_take isl_map *map,
        __isl_take isl_set *dom);

#include <isl/union_set.h>
__isl_give isl_union_set *isl_union_set_subtract(
        __isl_take isl_union_set *uset1,
        __isl_take isl_union_set *uset2);

#include <isl/union_map.h>
__isl_give isl_union_map *isl_union_map_subtract(
        __isl_take isl_union_map *umap1,
        __isl_take isl_union_map *umap2);
__isl_give isl_union_map *isl_union_map_subtract_domain(
        __isl_take isl_union_map *umap,
        __isl_take isl_union_set *dom);
__isl_give isl_union_map *isl_union_map_subtract_range(
        __isl_take isl_union_map *umap,
        __isl_take isl_union_set *dom);

#include <isl/aff.h>
__isl_give isl_pw_aff *isl_pw_aff_subtract_domain(
        __isl_take isl_pw_aff *pa,
        __isl_take isl_set *set);
__isl_give isl_pw_multi_aff *
isl_pw_multi_aff_subtract_domain(
        __isl_take isl_pw_multi_aff *pma,
        __isl_take isl_set *set);
__isl_give isl_union_pw_aff *
isl_union_pw_aff_subtract_domain(
        __isl_take isl_union_pw_aff *upa,
        __isl_take isl_union_set *uset);
__isl_give isl_union_pw_multi_aff *
isl_union_pw_multi_aff_subtract_domain(
        __isl_take isl_union_pw_multi_aff *upma,
        __isl_take isl_set *set);

#include <isl/polynomial.h>
```

```
__isl_give isl_pw_qpolynomial *
isl_pw_qpolynomial_subtract_domain(
        __isl_take isl_pw_qpolynomial *pwpq,
        __isl_take isl_set *set);
__isl_give isl_pw_qpolynomial_fold *
isl_pw_qpolynomial_fold_subtract_domain(
        __isl_take isl_pw_qpolynomial_fold *pwf,
        __isl_take isl_set *set);
__isl_give isl_union_pw_qpolynomial *
isl_union_pw_qpolynomial_subtract_domain(
        __isl_take isl_union_pw_qpolynomial *upwpq,
        __isl_take isl_union_set *uset);
__isl_give isl_union_pw_qpolynomial_fold *
isl_union_pw_qpolynomial_fold_subtract_domain(
        __isl_take isl_union_pw_qpolynomial_fold *upwf,
        __isl_take isl_union_set *uset);
```

- Application

```
#include <isl/space.h>
__isl_give isl_space *isl_space_join(
        __isl_take isl_space *left,
        __isl_take isl_space *right);

#include <isl/map.h>
__isl_give isl_basic_set *isl_basic_set_apply(
        __isl_take isl_basic_set *bset,
        __isl_take isl_basic_map *bmap);
__isl_give isl_set *isl_set_apply(
        __isl_take isl_set *set,
        __isl_take isl_map *map);
__isl_give isl_union_set *isl_union_set_apply(
        __isl_take isl_union_set *uset,
        __isl_take isl_union_map *umap);
__isl_give isl_basic_map *isl_basic_map_apply_domain(
        __isl_take isl_basic_map *bmap1,
        __isl_take isl_basic_map *bmap2);
__isl_give isl_basic_map *isl_basic_map_apply_range(
        __isl_take isl_basic_map *bmap1,
        __isl_take isl_basic_map *bmap2);
__isl_give isl_map *isl_map_apply_domain(
        __isl_take isl_map *map1,
        __isl_take isl_map *map2);
__isl_give isl_map *isl_map_apply_range(
        __isl_take isl_map *map1,
        __isl_take isl_map *map2);
```

```
#include <isl/union_map.h>
__isl_give isl_union_map *isl_union_map_apply_domain(
        __isl_take isl_union_map *umap1,
        __isl_take isl_union_map *umap2);
__isl_give isl_union_map *isl_union_map_apply_range(
        __isl_take isl_union_map *umap1,
        __isl_take isl_union_map *umap2);

#include <isl/aff.h>
__isl_give isl_union_pw_aff *
isl_multi_union_pw_aff_apply_aff(
        __isl_take isl_multi_union_pw_aff *mupa,
        __isl_take isl_aff *aff);
__isl_give isl_union_pw_aff *
isl_multi_union_pw_aff_apply_pw_aff(
        __isl_take isl_multi_union_pw_aff *mupa,
        __isl_take isl_pw_aff *pa);
__isl_give isl_multi_union_pw_aff *
isl_multi_union_pw_aff_apply_multi_aff(
        __isl_take isl_multi_union_pw_aff *mupa,
        __isl_take isl_multi_aff *ma);
__isl_give isl_multi_union_pw_aff *
isl_multi_union_pw_aff_apply_pw_multi_aff(
        __isl_take isl_multi_union_pw_aff *mupa,
        __isl_take isl_pw_multi_aff *pma);
```

The result of `isl_multi_union_pw_aff_apply_aff` is defined over the shared domain of the elements of the input. The dimension is required to be greater than zero. The `isl_multi_union_pw_aff` argument of `isl_multi_union_pw_aff_apply_multi_aff` is allowed to be zero-dimensional, but only if the range of the `isl_multi_aff` argument is also zero-dimensional. Similarly for `isl_multi_union_pw_aff_apply_pw_multi_aff`.

```
#include <isl/polynomial.h>
__isl_give isl_pw_qpolynomial_fold *
isl_set_apply_pw_qpolynomial_fold(
        __isl_take isl_set *set,
        __isl_take isl_pw_qpolynomial_fold *pwf,
        int *tight);
__isl_give isl_pw_qpolynomial_fold *
isl_map_apply_pw_qpolynomial_fold(
        __isl_take isl_map *map,
        __isl_take isl_pw_qpolynomial_fold *pwf,
        int *tight);
__isl_give isl_union_pw_qpolynomial_fold *
isl_union_set_apply_union_pw_qpolynomial_fold(
        __isl_take isl_union_set *uset,
```

```
        __isl_take isl_union_pw_qpolynomial_fold *upwf,
        int *tight);
__isl_give isl_union_pw_qpolynomial_fold *
isl_union_map_apply_union_pw_qpolynomial_fold(
        __isl_take isl_union_map *umap,
        __isl_take isl_union_pw_qpolynomial_fold *upwf,
        int *tight);
```

The functions taking a map compose the given map with the given piecewise quasipolynomial reduction. That is, compute a bound (of the same type as `pwf` or `upwf` itself) over all elements in the intersection of the range of the map and the domain of the piecewise quasipolynomial reduction as a function of an element in the domain of the map. The functions taking a set compute a bound over all elements in the intersection of the set and the domain of the piecewise quasipolynomial reduction.

- Preimage

```
#include <isl/set.h>
__isl_give isl_basic_set *
isl_basic_set_preimage_multi_aff(
        __isl_take isl_basic_set *bset,
        __isl_take isl_multi_aff *ma);
__isl_give isl_set *isl_set_preimage_multi_aff(
        __isl_take isl_set *set,
        __isl_take isl_multi_aff *ma);
__isl_give isl_set *isl_set_preimage_pw_multi_aff(
        __isl_take isl_set *set,
        __isl_take isl_pw_multi_aff *pma);
__isl_give isl_set *isl_set_preimage_multi_pw_aff(
        __isl_take isl_set *set,
        __isl_take isl_multi_pw_aff *mpa);

#include <isl/union_set.h>
__isl_give isl_union_set *
isl_union_set_preimage_multi_aff(
        __isl_take isl_union_set *uset,
        __isl_take isl_multi_aff *ma);
__isl_give isl_union_set *
isl_union_set_preimage_pw_multi_aff(
        __isl_take isl_union_set *uset,
        __isl_take isl_pw_multi_aff *pma);
__isl_give isl_union_set *
isl_union_set_preimage_union_pw_multi_aff(
        __isl_take isl_union_set *uset,
        __isl_take isl_union_pw_multi_aff *upma);
```

```
#include <isl/map.h>
__isl_give isl_basic_map *
isl_basic_map_preimage_domain_multi_aff(
        __isl_take isl_basic_map *bmap,
        __isl_take isl_multi_aff *ma);
__isl_give isl_map *isl_map_preimage_domain_multi_aff(
        __isl_take isl_map *map,
        __isl_take isl_multi_aff *ma);
__isl_give isl_map *isl_map_preimage_range_multi_aff(
        __isl_take isl_map *map,
        __isl_take isl_multi_aff *ma);
__isl_give isl_map *
isl_map_preimage_domain_pw_multi_aff(
        __isl_take isl_map *map,
        __isl_take isl_pw_multi_aff *pma);
__isl_give isl_map *
isl_map_preimage_range_pw_multi_aff(
        __isl_take isl_map *map,
        __isl_take isl_pw_multi_aff *pma);
__isl_give isl_map *
isl_map_preimage_domain_multi_pw_aff(
        __isl_take isl_map *map,
        __isl_take isl_multi_pw_aff *mpa);
__isl_give isl_basic_map *
isl_basic_map_preimage_range_multi_aff(
        __isl_take isl_basic_map *bmap,
        __isl_take isl_multi_aff *ma);

#include <isl/union_map.h>
__isl_give isl_union_map *
isl_union_map_preimage_domain_multi_aff(
        __isl_take isl_union_map *umap,
        __isl_take isl_multi_aff *ma);
__isl_give isl_union_map *
isl_union_map_preimage_range_multi_aff(
        __isl_take isl_union_map *umap,
        __isl_take isl_multi_aff *ma);
__isl_give isl_union_map *
isl_union_map_preimage_domain_pw_multi_aff(
        __isl_take isl_union_map *umap,
        __isl_take isl_pw_multi_aff *pma);
__isl_give isl_union_map *
isl_union_map_preimage_range_pw_multi_aff(
        __isl_take isl_union_map *umap,
        __isl_take isl_pw_multi_aff *pma);
__isl_give isl_union_map *
```

```
isl_union_map_preimage_domain_union_pw_multi_aff(
        __isl_take isl_union_map *umap,
        __isl_take isl_union_pw_multi_aff *upma);
__isl_give isl_union_map *
isl_union_map_preimage_range_union_pw_multi_aff(
        __isl_take isl_union_map *umap,
        __isl_take isl_union_pw_multi_aff *upma);
```

These functions compute the preimage of the given set or map domain/range under the given function. In other words, the expression is plugged into the set description or into the domain/range of the map.

- Pullback

```
#include <isl/aff.h>
__isl_give isl_aff *isl_aff_pullback_aff(
        __isl_take isl_aff *aff1,
        __isl_take isl_aff *aff2);
__isl_give isl_aff *isl_aff_pullback_multi_aff(
        __isl_take isl_aff *aff,
        __isl_take isl_multi_aff *ma);
__isl_give isl_pw_aff *isl_pw_aff_pullback_multi_aff(
        __isl_take isl_pw_aff *pa,
        __isl_take isl_multi_aff *ma);
__isl_give isl_pw_aff *isl_pw_aff_pullback_pw_multi_aff(
        __isl_take isl_pw_aff *pa,
        __isl_take isl_pw_multi_aff *pma);
__isl_give isl_pw_aff *isl_pw_aff_pullback_multi_pw_aff(
        __isl_take isl_pw_aff *pa,
        __isl_take isl_multi_pw_aff *mpa);
__isl_give isl_multi_aff *isl_multi_aff_pullback_multi_aff(
        __isl_take isl_multi_aff *ma1,
        __isl_take isl_multi_aff *ma2);
__isl_give isl_pw_multi_aff *
isl_pw_multi_aff_pullback_multi_aff(
        __isl_take isl_pw_multi_aff *pma,
        __isl_take isl_multi_aff *ma);
__isl_give isl_multi_pw_aff *
isl_multi_pw_aff_pullback_multi_aff(
        __isl_take isl_multi_pw_aff *mpa,
        __isl_take isl_multi_aff *ma);
__isl_give isl_pw_multi_aff *
isl_pw_multi_aff_pullback_pw_multi_aff(
        __isl_take isl_pw_multi_aff *pma1,
        __isl_take isl_pw_multi_aff *pma2);
__isl_give isl_multi_pw_aff *
```

```
isl_multi_pw_aff_pullback_pw_multi_aff(
        __isl_take isl_multi_pw_aff *mpa,
        __isl_take isl_pw_multi_aff *pma);
__isl_give isl_multi_pw_aff *
isl_multi_pw_aff_pullback_multi_pw_aff(
        __isl_take isl_multi_pw_aff *mpa1,
        __isl_take isl_multi_pw_aff *mpa2);
__isl_give isl_union_pw_aff *
isl_union_pw_aff_pullback_union_pw_multi_aff(
        __isl_take isl_union_pw_aff *upa,
        __isl_take isl_union_pw_multi_aff *upma);
__isl_give isl_union_pw_multi_aff *
isl_union_pw_multi_aff_pullback_union_pw_multi_aff(
        __isl_take isl_union_pw_multi_aff *upma1,
        __isl_take isl_union_pw_multi_aff *upma2);
__isl_give isl_multi_union_pw_aff *
isl_multi_union_pw_aff_pullback_union_pw_multi_aff(
        __isl_take isl_multi_union_pw_aff *mupa,
        __isl_take isl_union_pw_multi_aff *upma);
```

These functions precompose the first expression by the second function. In other words, the second function is plugged into the first expression.

- Locus

```
#include <isl/aff.h>
__isl_give isl_basic_set *isl_aff_eq_basic_set(
        __isl_take isl_aff *aff1,
        __isl_take isl_aff *aff2);
__isl_give isl_set *isl_aff_eq_set(
        __isl_take isl_aff *aff1,
        __isl_take isl_aff *aff2);
__isl_give isl_basic_set *isl_aff_le_basic_set(
        __isl_take isl_aff *aff1,
        __isl_take isl_aff *aff2);
__isl_give isl_set *isl_aff_le_set(
        __isl_take isl_aff *aff1,
        __isl_take isl_aff *aff2);
__isl_give isl_basic_set *isl_aff_ge_basic_set(
        __isl_take isl_aff *aff1,
        __isl_take isl_aff *aff2);
__isl_give isl_set *isl_aff_ge_set(
        __isl_take isl_aff *aff1,
        __isl_take isl_aff *aff2);
__isl_give isl_set *isl_pw_aff_eq_set(
        __isl_take isl_pw_aff *pwaff1,
```

```
        __isl_take isl_pw_aff *pwaff2);
__isl_give isl_set *isl_pw_aff_ne_set(
        __isl_take isl_pw_aff *pwaff1,
        __isl_take isl_pw_aff *pwaff2);
__isl_give isl_set *isl_pw_aff_le_set(
        __isl_take isl_pw_aff *pwaff1,
        __isl_take isl_pw_aff *pwaff2);
__isl_give isl_set *isl_pw_aff_lt_set(
        __isl_take isl_pw_aff *pwaff1,
        __isl_take isl_pw_aff *pwaff2);
__isl_give isl_set *isl_pw_aff_ge_set(
        __isl_take isl_pw_aff *pwaff1,
        __isl_take isl_pw_aff *pwaff2);
__isl_give isl_set *isl_pw_aff_gt_set(
        __isl_take isl_pw_aff *pwaff1,
        __isl_take isl_pw_aff *pwaff2);

__isl_give isl_set *isl_multi_aff_lex_le_set(
        __isl_take isl_multi_aff *ma1,
        __isl_take isl_multi_aff *ma2);
__isl_give isl_set *isl_multi_aff_lex_lt_set(
        __isl_take isl_multi_aff *ma1,
        __isl_take isl_multi_aff *ma2);
__isl_give isl_set *isl_multi_aff_lex_ge_set(
        __isl_take isl_multi_aff *ma1,
        __isl_take isl_multi_aff *ma2);
__isl_give isl_set *isl_multi_aff_lex_gt_set(
        __isl_take isl_multi_aff *ma1,
        __isl_take isl_multi_aff *ma2);

__isl_give isl_set *isl_pw_aff_list_eq_set(
        __isl_take isl_pw_aff_list *list1,
        __isl_take isl_pw_aff_list *list2);
__isl_give isl_set *isl_pw_aff_list_ne_set(
        __isl_take isl_pw_aff_list *list1,
        __isl_take isl_pw_aff_list *list2);
__isl_give isl_set *isl_pw_aff_list_le_set(
        __isl_take isl_pw_aff_list *list1,
        __isl_take isl_pw_aff_list *list2);
__isl_give isl_set *isl_pw_aff_list_lt_set(
        __isl_take isl_pw_aff_list *list1,
        __isl_take isl_pw_aff_list *list2);
__isl_give isl_set *isl_pw_aff_list_ge_set(
        __isl_take isl_pw_aff_list *list1,
        __isl_take isl_pw_aff_list *list2);
__isl_give isl_set *isl_pw_aff_list_gt_set(
```

```
        __isl_take isl_pw_aff_list *list1,
        __isl_take isl_pw_aff_list *list2);
```

The function `isl_aff_ge_basic_set` returns a basic set containing those elements in the shared space of `aff1` and `aff2` where `aff1` is greater than or equal to `aff2`. The function `isl_pw_aff_ge_set` returns a set containing those elements in the shared domain of `pwaff1` and `pwaff2` where `pwaff1` is greater than or equal to `pwaff2`. The function `isl_multi_aff_lex_le_set` returns a set containing those elements in the shared domain space where `ma1` is lexicographically smaller than or equal to `ma2`. The functions operating on `isl_pw_aff_list` apply the corresponding `isl_pw_aff` function to each pair of elements in the two lists.

```
#include <isl/aff.h>
__isl_give isl_map *isl_pw_aff_eq_map(
        __isl_take isl_pw_aff *pa1,
        __isl_take isl_pw_aff *pa2);
__isl_give isl_map *isl_pw_aff_lt_map(
        __isl_take isl_pw_aff *pa1,
        __isl_take isl_pw_aff *pa2);
__isl_give isl_map *isl_pw_aff_gt_map(
        __isl_take isl_pw_aff *pa1,
        __isl_take isl_pw_aff *pa2);

__isl_give isl_map *isl_multi_pw_aff_eq_map(
        __isl_take isl_multi_pw_aff *mpa1,
        __isl_take isl_multi_pw_aff *mpa2);
__isl_give isl_map *isl_multi_pw_aff_lex_lt_map(
        __isl_take isl_multi_pw_aff *mpa1,
        __isl_take isl_multi_pw_aff *mpa2);
__isl_give isl_map *isl_multi_pw_aff_lex_gt_map(
        __isl_take isl_multi_pw_aff *mpa1,
        __isl_take isl_multi_pw_aff *mpa2);
```

These functions return a map between domain elements of the arguments where the function values satisfy the given relation.

```
#include <isl/union_map.h>
__isl_give isl_union_map *
isl_union_map_eq_at_multi_union_pw_aff(
        __isl_take isl_union_map *umap,
        __isl_take isl_multi_union_pw_aff *mupa);
__isl_give isl_union_map *
isl_union_map_lex_lt_at_multi_union_pw_aff(
        __isl_take isl_union_map *umap,
```

```
        __isl_take isl_multi_union_pw_aff *mupa);
__isl_give isl_union_map *
isl_union_map_lex_gt_at_multi_union_pw_aff(
        __isl_take isl_union_map *umap,
        __isl_take isl_multi_union_pw_aff *mupa);
```

These functions select the subset of elements in the union map that have an equal or lexicographically smaller function value.

- Cartesian Product

```
#include <isl/space.h>
__isl_give isl_space *isl_space_product(
        __isl_take isl_space *space1,
        __isl_take isl_space *space2);
__isl_give isl_space *isl_space_domain_product(
        __isl_take isl_space *space1,
        __isl_take isl_space *space2);
__isl_give isl_space *isl_space_range_product(
        __isl_take isl_space *space1,
        __isl_take isl_space *space2);
```

The functions `isl_space_product`, `isl_space_domain_product` and `isl_space_range_product` take pairs or relation spaces and produce a single relations space, where either the domain, the range or both domain and range are wrapped spaces of relations between the domains and/or ranges of the input spaces. If the product is only constructed over the domain or the range then the ranges or the domains of the inputs should be the same. The function `isl_space_product` also accepts a pair of set spaces, in which case it returns a wrapped space of a relation between the two input spaces.

```
#include <isl/set.h>
__isl_give isl_set *isl_set_product(
        __isl_take isl_set *set1,
        __isl_take isl_set *set2);

#include <isl/map.h>
__isl_give isl_basic_map *isl_basic_map_domain_product(
        __isl_take isl_basic_map *bmap1,
        __isl_take isl_basic_map *bmap2);
__isl_give isl_basic_map *isl_basic_map_range_product(
        __isl_take isl_basic_map *bmap1,
        __isl_take isl_basic_map *bmap2);
__isl_give isl_basic_map *isl_basic_map_product(
        __isl_take isl_basic_map *bmap1,
        __isl_take isl_basic_map *bmap2);
```

```
__isl_give isl_map *isl_map_domain_product(
        __isl_take isl_map *map1,
        __isl_take isl_map *map2);
__isl_give isl_map *isl_map_range_product(
        __isl_take isl_map *map1,
        __isl_take isl_map *map2);
__isl_give isl_map *isl_map_product(
        __isl_take isl_map *map1,
        __isl_take isl_map *map2);

#include <isl/union_set.h>
__isl_give isl_union_set *isl_union_set_product(
        __isl_take isl_union_set *uset1,
        __isl_take isl_union_set *uset2);

#include <isl/union_map.h>
__isl_give isl_union_map *isl_union_map_domain_product(
        __isl_take isl_union_map *umap1,
        __isl_take isl_union_map *umap2);
__isl_give isl_union_map *isl_union_map_range_product(
        __isl_take isl_union_map *umap1,
        __isl_take isl_union_map *umap2);
__isl_give isl_union_map *isl_union_map_product(
        __isl_take isl_union_map *umap1,
        __isl_take isl_union_map *umap2);

#include <isl/val.h>
__isl_give isl_multi_val *isl_multi_val_range_product(
        __isl_take isl_multi_val *mv1,
        __isl_take isl_multi_val *mv2);
__isl_give isl_multi_val *isl_multi_val_product(
        __isl_take isl_multi_val *mv1,
        __isl_take isl_multi_val *mv2);

#include <isl/aff.h>
__isl_give isl_multi_aff *isl_multi_aff_range_product(
        __isl_take isl_multi_aff *ma1,
        __isl_take isl_multi_aff *ma2);
__isl_give isl_multi_aff *isl_multi_aff_product(
        __isl_take isl_multi_aff *ma1,
        __isl_take isl_multi_aff *ma2);
__isl_give isl_multi_pw_aff *
isl_multi_pw_aff_range_product(
        __isl_take isl_multi_pw_aff *mpa1,
        __isl_take isl_multi_pw_aff *mpa2);
```

```
__isl_give isl_multi_pw_aff *isl_multi_pw_aff_product(
        __isl_take isl_multi_pw_aff *mpa1,
        __isl_take isl_multi_pw_aff *mpa2);
__isl_give isl_pw_multi_aff *
isl_pw_multi_aff_range_product(
        __isl_take isl_pw_multi_aff *pma1,
        __isl_take isl_pw_multi_aff *pma2);
__isl_give isl_pw_multi_aff *isl_pw_multi_aff_product(
        __isl_take isl_pw_multi_aff *pma1,
        __isl_take isl_pw_multi_aff *pma2);
__isl_give isl_multi_union_pw_aff *
isl_multi_union_pw_aff_range_product(
        __isl_take isl_multi_union_pw_aff *mupa1,
        __isl_take isl_multi_union_pw_aff *mupa2);
```

The above functions compute the cross product of the given sets, relations or functions. The domains and ranges of the results are wrapped maps between domains and ranges of the inputs. To obtain a "flat" product, use the following functions instead.

```
#include <isl/set.h>
__isl_give isl_basic_set *isl_basic_set_flat_product(
        __isl_take isl_basic_set *bset1,
        __isl_take isl_basic_set *bset2);
__isl_give isl_set *isl_set_flat_product(
        __isl_take isl_set *set1,
        __isl_take isl_set *set2);

#include <isl/map.h>
__isl_give isl_basic_map *isl_basic_map_flat_range_product(
        __isl_take isl_basic_map *bmap1,
        __isl_take isl_basic_map *bmap2);
__isl_give isl_map *isl_map_flat_domain_product(
        __isl_take isl_map *map1,
        __isl_take isl_map *map2);
__isl_give isl_map *isl_map_flat_range_product(
        __isl_take isl_map *map1,
        __isl_take isl_map *map2);
__isl_give isl_basic_map *isl_basic_map_flat_product(
        __isl_take isl_basic_map *bmap1,
        __isl_take isl_basic_map *bmap2);
__isl_give isl_map *isl_map_flat_product(
        __isl_take isl_map *map1,
        __isl_take isl_map *map2);

#include <isl/union_map.h>
```

```
__isl_give isl_union_map *
isl_union_map_flat_domain_product(
        __isl_take isl_union_map *umap1,
        __isl_take isl_union_map *umap2);
__isl_give isl_union_map *
isl_union_map_flat_range_product(
        __isl_take isl_union_map *umap1,
        __isl_take isl_union_map *umap2);

#include <isl/val.h>
__isl_give isl_multi_val *isl_multi_val_flat_range_product(
        __isl_take isl_multi_val *mv1,
        __isl_take isl_multi_aff *mv2);

#include <isl/aff.h>
__isl_give isl_multi_aff *isl_multi_aff_flat_range_product(
        __isl_take isl_multi_aff *ma1,
        __isl_take isl_multi_aff *ma2);
__isl_give isl_pw_multi_aff *
isl_pw_multi_aff_flat_range_product(
        __isl_take isl_pw_multi_aff *pma1,
        __isl_take isl_pw_multi_aff *pma2);
__isl_give isl_multi_pw_aff *
isl_multi_pw_aff_flat_range_product(
        __isl_take isl_multi_pw_aff *mpa1,
        __isl_take isl_multi_pw_aff *mpa2);
__isl_give isl_union_pw_multi_aff *
isl_union_pw_multi_aff_flat_range_product(
        __isl_take isl_union_pw_multi_aff *upma1,
        __isl_take isl_union_pw_multi_aff *upma2);
__isl_give isl_multi_union_pw_aff *
isl_multi_union_pw_aff_flat_range_product(
        __isl_take isl_multi_union_pw_aff *mupa1,
        __isl_take isl_multi_union_pw_aff *mupa2);

#include <isl/space.h>
__isl_give isl_space *isl_space_factor_domain(
        __isl_take isl_space *space);
__isl_give isl_space *isl_space_factor_range(
        __isl_take isl_space *space);
__isl_give isl_space *isl_space_domain_factor_domain(
        __isl_take isl_space *space);
__isl_give isl_space *isl_space_domain_factor_range(
        __isl_take isl_space *space);
__isl_give isl_space *isl_space_range_factor_domain(
        __isl_take isl_space *space);
```

```
__isl_give isl_space *isl_space_range_factor_range(
        __isl_take isl_space *space);
```

The functions `isl_space_range_factor_domain` and `isl_space_range_factor_range` extract the two arguments from the result of a call to `isl_space_range_product`.

The arguments of a call to a product can be extracted from the result using the following functions.

```
#include <isl/map.h>
__isl_give isl_map *isl_map_factor_domain(
        __isl_take isl_map *map);
__isl_give isl_map *isl_map_factor_range(
        __isl_take isl_map *map);
__isl_give isl_map *isl_map_domain_factor_domain(
        __isl_take isl_map *map);
__isl_give isl_map *isl_map_domain_factor_range(
        __isl_take isl_map *map);
__isl_give isl_map *isl_map_range_factor_domain(
        __isl_take isl_map *map);
__isl_give isl_map *isl_map_range_factor_range(
        __isl_take isl_map *map);

#include <isl/union_map.h>
__isl_give isl_union_map *isl_union_map_factor_domain(
        __isl_take isl_union_map *umap);
__isl_give isl_union_map *isl_union_map_factor_range(
        __isl_take isl_union_map *umap);
__isl_give isl_union_map *
isl_union_map_domain_factor_domain(
        __isl_take isl_union_map *umap);
__isl_give isl_union_map *
isl_union_map_domain_factor_range(
        __isl_take isl_union_map *umap);
__isl_give isl_union_map *
isl_union_map_range_factor_domain(
        __isl_take isl_union_map *umap);
__isl_give isl_union_map *
isl_union_map_range_factor_range(
        __isl_take isl_union_map *umap);

#include <isl/val.h>
__isl_give isl_multi_val *isl_multi_val_factor_range(
        __isl_take isl_multi_val *mv);
__isl_give isl_multi_val *
isl_multi_val_range_factor_domain(
```

```
        __isl_take isl_multi_val *mv);
__isl_give isl_multi_val *
isl_multi_val_range_factor_range(
        __isl_take isl_multi_val *mv);

#include <isl/aff.h>
__isl_give isl_multi_aff *isl_multi_aff_factor_range(
        __isl_take isl_multi_aff *ma);
__isl_give isl_multi_aff *
isl_multi_aff_range_factor_domain(
        __isl_take isl_multi_aff *ma);
__isl_give isl_multi_aff *
isl_multi_aff_range_factor_range(
        __isl_take isl_multi_aff *ma);
__isl_give isl_multi_pw_aff *
isl_multi_pw_aff_factor_range(
        __isl_take isl_multi_pw_aff *mpa);
__isl_give isl_multi_pw_aff *
isl_multi_pw_aff_range_factor_domain(
        __isl_take isl_multi_pw_aff *mpa);
__isl_give isl_multi_pw_aff *
isl_multi_pw_aff_range_factor_range(
        __isl_take isl_multi_pw_aff *mpa);
__isl_give isl_multi_union_pw_aff *
isl_multi_union_pw_aff_factor_range(
        __isl_take isl_multi_union_pw_aff *mupa);
__isl_give isl_multi_union_pw_aff *
isl_multi_union_pw_aff_range_factor_domain(
        __isl_take isl_multi_union_pw_aff *mupa);
__isl_give isl_multi_union_pw_aff *
isl_multi_union_pw_aff_range_factor_range(
        __isl_take isl_multi_union_pw_aff *mupa);
```

The splice functions are a generalization of the flat product functions, where the second argument may be inserted at any position inside the first argument rather than being placed at the end. The functions `isl_multi_val_factor_range`, `isl_multi_aff_factor_range`, `isl_multi_pw_aff_factor_range` and `isl_multi_union_pw_aff_facto` take functions that live in a set space.

```
#include <isl/val.h>
__isl_give isl_multi_val *isl_multi_val_range_splice(
        __isl_take isl_multi_val *mv1, unsigned pos,
        __isl_take isl_multi_val *mv2);

#include <isl/aff.h>
```

```
__isl_give isl_multi_aff *isl_multi_aff_range_splice(
        __isl_take isl_multi_aff *ma1, unsigned pos,
        __isl_take isl_multi_aff *ma2);
__isl_give isl_multi_aff *isl_multi_aff_splice(
        __isl_take isl_multi_aff *ma1,
        unsigned in_pos, unsigned out_pos,
        __isl_take isl_multi_aff *ma2);
__isl_give isl_multi_pw_aff *
isl_multi_pw_aff_range_splice(
        __isl_take isl_multi_pw_aff *mpa1, unsigned pos,
        __isl_take isl_multi_pw_aff *mpa2);
__isl_give isl_multi_pw_aff *isl_multi_pw_aff_splice(
        __isl_take isl_multi_pw_aff *mpa1,
        unsigned in_pos, unsigned out_pos,
        __isl_take isl_multi_pw_aff *mpa2);
__isl_give isl_multi_union_pw_aff *
isl_multi_union_pw_aff_range_splice(
        __isl_take isl_multi_union_pw_aff *mupa1,
        unsigned pos,
        __isl_take isl_multi_union_pw_aff *mupa2);
```

- Simplification

  When applied to a set or relation, the gist operation returns a set or relation that has the same intersection with the context as the input set or relation. Any implicit equality in the intersection is made explicit in the result, while all inequalities that are redundant with respect to the intersection are removed. In case of union sets and relations, the gist operation is performed per space.

  When applied to a function, the gist operation applies the set gist operation to each of the cells in the domain of the input piecewise expression. The context is also exploited to simplify the expression associated to each cell.

```
#include <isl/set.h>
__isl_give isl_basic_set *isl_basic_set_gist(
        __isl_take isl_basic_set *bset,
        __isl_take isl_basic_set *context);
__isl_give isl_set *isl_set_gist(__isl_take isl_set *set,
        __isl_take isl_set *context);
__isl_give isl_set *isl_set_gist_params(
        __isl_take isl_set *set,
        __isl_take isl_set *context);

#include <isl/map.h>
__isl_give isl_basic_map *isl_basic_map_gist(
        __isl_take isl_basic_map *bmap,
        __isl_take isl_basic_map *context);
```

```
__isl_give isl_basic_map *isl_basic_map_gist_domain(
        __isl_take isl_basic_map *bmap,
        __isl_take isl_basic_set *context);
__isl_give isl_map *isl_map_gist(__isl_take isl_map *map,
        __isl_take isl_map *context);
__isl_give isl_map *isl_map_gist_params(
        __isl_take isl_map *map,
        __isl_take isl_set *context);
__isl_give isl_map *isl_map_gist_domain(
        __isl_take isl_map *map,
        __isl_take isl_set *context);
__isl_give isl_map *isl_map_gist_range(
        __isl_take isl_map *map,
        __isl_take isl_set *context);

#include <isl/union_set.h>
__isl_give isl_union_set *isl_union_set_gist(
        __isl_take isl_union_set *uset,
        __isl_take isl_union_set *context);
__isl_give isl_union_set *isl_union_set_gist_params(
        __isl_take isl_union_set *uset,
        __isl_take isl_set *set);

#include <isl/union_map.h>
__isl_give isl_union_map *isl_union_map_gist(
        __isl_take isl_union_map *umap,
        __isl_take isl_union_map *context);
__isl_give isl_union_map *isl_union_map_gist_params(
        __isl_take isl_union_map *umap,
        __isl_take isl_set *set);
__isl_give isl_union_map *isl_union_map_gist_domain(
        __isl_take isl_union_map *umap,
        __isl_take isl_union_set *uset);
__isl_give isl_union_map *isl_union_map_gist_range(
        __isl_take isl_union_map *umap,
        __isl_take isl_union_set *uset);

#include <isl/aff.h>
__isl_give isl_aff *isl_aff_gist_params(
        __isl_take isl_aff *aff,
        __isl_take isl_set *context);
__isl_give isl_aff *isl_aff_gist(__isl_take isl_aff *aff,
        __isl_take isl_set *context);
__isl_give isl_multi_aff *isl_multi_aff_gist_params(
        __isl_take isl_multi_aff *maff,
        __isl_take isl_set *context);
```

```
__isl_give isl_multi_aff *isl_multi_aff_gist(
        __isl_take isl_multi_aff *maff,
        __isl_take isl_set *context);
__isl_give isl_pw_aff *isl_pw_aff_gist_params(
        __isl_take isl_pw_aff *pwaff,
        __isl_take isl_set *context);
__isl_give isl_pw_aff *isl_pw_aff_gist(
        __isl_take isl_pw_aff *pwaff,
        __isl_take isl_set *context);
__isl_give isl_pw_multi_aff *isl_pw_multi_aff_gist_params(
        __isl_take isl_pw_multi_aff *pma,
        __isl_take isl_set *set);
__isl_give isl_pw_multi_aff *isl_pw_multi_aff_gist(
        __isl_take isl_pw_multi_aff *pma,
        __isl_take isl_set *set);
__isl_give isl_multi_pw_aff *isl_multi_pw_aff_gist_params(
        __isl_take isl_multi_pw_aff *mpa,
        __isl_take isl_set *set);
__isl_give isl_multi_pw_aff *isl_multi_pw_aff_gist(
        __isl_take isl_multi_pw_aff *mpa,
        __isl_take isl_set *set);
__isl_give isl_union_pw_aff *isl_union_pw_aff_gist(
        __isl_take isl_union_pw_aff *upa,
        __isl_take isl_union_set *context);
__isl_give isl_union_pw_aff *isl_union_pw_aff_gist_params(
        __isl_take isl_union_pw_aff *upa,
        __isl_take isl_set *context);
__isl_give isl_union_pw_multi_aff *
isl_union_pw_multi_aff_gist_params(
        __isl_take isl_union_pw_multi_aff *upma,
        __isl_take isl_set *context);
__isl_give isl_union_pw_multi_aff *
isl_union_pw_multi_aff_gist(
        __isl_take isl_union_pw_multi_aff *upma,
        __isl_take isl_union_set *context);
__isl_give isl_multi_union_pw_aff *
isl_multi_union_pw_aff_gist_params(
        __isl_take isl_multi_union_pw_aff *aff,
        __isl_take isl_set *context);
__isl_give isl_multi_union_pw_aff *
isl_multi_union_pw_aff_gist(
        __isl_take isl_multi_union_pw_aff *aff,
        __isl_take isl_union_set *context);

#include <isl/polynomial.h>
__isl_give isl_qpolynomial *isl_qpolynomial_gist_params(
```

```
        __isl_take isl_qpolynomial *qp,
        __isl_take isl_set *context);
__isl_give isl_qpolynomial *isl_qpolynomial_gist(
        __isl_take isl_qpolynomial *qp,
        __isl_take isl_set *context);
__isl_give isl_qpolynomial_fold *
isl_qpolynomial_fold_gist_params(
        __isl_take isl_qpolynomial_fold *fold,
        __isl_take isl_set *context);
__isl_give isl_qpolynomial_fold *isl_qpolynomial_fold_gist(
        __isl_take isl_qpolynomial_fold *fold,
        __isl_take isl_set *context);
__isl_give isl_pw_qpolynomial *isl_pw_qpolynomial_gist_params(
        __isl_take isl_pw_qpolynomial *pwqp,
        __isl_take isl_set *context);
__isl_give isl_pw_qpolynomial *isl_pw_qpolynomial_gist(
        __isl_take isl_pw_qpolynomial *pwqp,
        __isl_take isl_set *context);
__isl_give isl_pw_qpolynomial_fold *
isl_pw_qpolynomial_fold_gist(
        __isl_take isl_pw_qpolynomial_fold *pwf,
        __isl_take isl_set *context);
__isl_give isl_pw_qpolynomial_fold *
isl_pw_qpolynomial_fold_gist_params(
        __isl_take isl_pw_qpolynomial_fold *pwf,
        __isl_take isl_set *context);
__isl_give isl_union_pw_qpolynomial *
isl_union_pw_qpolynomial_gist_params(
        __isl_take isl_union_pw_qpolynomial *upwqp,
        __isl_take isl_set *context);
__isl_give isl_union_pw_qpolynomial *isl_union_pw_qpolynomial_gist(
        __isl_take isl_union_pw_qpolynomial *upwqp,
        __isl_take isl_union_set *context);
__isl_give isl_union_pw_qpolynomial_fold *
isl_union_pw_qpolynomial_fold_gist(
        __isl_take isl_union_pw_qpolynomial_fold *upwf,
        __isl_take isl_union_set *context);
__isl_give isl_union_pw_qpolynomial_fold *
isl_union_pw_qpolynomial_fold_gist_params(
        __isl_take isl_union_pw_qpolynomial_fold *upwf,
        __isl_take isl_set *context);
```

- Binary Arithmetic Operations

```
#include <isl/set.h>
__isl_give isl_set *isl_set_sum(
```

```
        __isl_take isl_set *set1,
        __isl_take isl_set *set2);
#include <isl/map.h>
__isl_give isl_map *isl_map_sum(
        __isl_take isl_map *map1,
        __isl_take isl_map *map2);
```

isl_set_sum computes the Minkowski sum of its two arguments, i.e., the set containing the sums of pairs of elements from set1 and set2. The domain of the result of isl_map_sum is the intersection of the domains of its two arguments. The corresponding range elements are the sums of the corresponding range elements in the two arguments.

```
#include <isl/val.h>
__isl_give isl_multi_val *isl_multi_val_add(
        __isl_take isl_multi_val *mv1,
        __isl_take isl_multi_val *mv2);
__isl_give isl_multi_val *isl_multi_val_sub(
        __isl_take isl_multi_val *mv1,
        __isl_take isl_multi_val *mv2);

#include <isl/aff.h>
__isl_give isl_aff *isl_aff_add(
        __isl_take isl_aff *aff1,
        __isl_take isl_aff *aff2);
__isl_give isl_multi_aff *isl_multi_aff_add(
        __isl_take isl_multi_aff *maff1,
        __isl_take isl_multi_aff *maff2);
__isl_give isl_pw_aff *isl_pw_aff_add(
        __isl_take isl_pw_aff *pwaff1,
        __isl_take isl_pw_aff *pwaff2);
__isl_give isl_multi_pw_aff *isl_multi_pw_aff_add(
        __isl_take isl_multi_pw_aff *mpa1,
        __isl_take isl_multi_pw_aff *mpa2);
__isl_give isl_pw_multi_aff *isl_pw_multi_aff_add(
        __isl_take isl_pw_multi_aff *pma1,
        __isl_take isl_pw_multi_aff *pma2);
__isl_give isl_union_pw_aff *isl_union_pw_aff_add(
        __isl_take isl_union_pw_aff *upa1,
        __isl_take isl_union_pw_aff *upa2);
__isl_give isl_union_pw_multi_aff *isl_union_pw_multi_aff_add(
        __isl_take isl_union_pw_multi_aff *upma1,
        __isl_take isl_union_pw_multi_aff *upma2);
__isl_give isl_multi_union_pw_aff *
isl_multi_union_pw_aff_add(
        __isl_take isl_multi_union_pw_aff *mupa1,
```

```
        __isl_take isl_multi_union_pw_aff *mupa2);
__isl_give isl_pw_aff *isl_pw_aff_min(
        __isl_take isl_pw_aff *pwaff1,
        __isl_take isl_pw_aff *pwaff2);
__isl_give isl_pw_aff *isl_pw_aff_max(
        __isl_take isl_pw_aff *pwaff1,
        __isl_take isl_pw_aff *pwaff2);
__isl_give isl_aff *isl_aff_sub(
        __isl_take isl_aff *aff1,
        __isl_take isl_aff *aff2);
__isl_give isl_multi_aff *isl_multi_aff_sub(
        __isl_take isl_multi_aff *ma1,
        __isl_take isl_multi_aff *ma2);
__isl_give isl_pw_aff *isl_pw_aff_sub(
        __isl_take isl_pw_aff *pwaff1,
        __isl_take isl_pw_aff *pwaff2);
__isl_give isl_multi_pw_aff *isl_multi_pw_aff_sub(
        __isl_take isl_multi_pw_aff *mpa1,
        __isl_take isl_multi_pw_aff *mpa2);
__isl_give isl_pw_multi_aff *isl_pw_multi_aff_sub(
        __isl_take isl_pw_multi_aff *pma1,
        __isl_take isl_pw_multi_aff *pma2);
__isl_give isl_union_pw_aff *isl_union_pw_aff_sub(
        __isl_take isl_union_pw_aff *upa1,
        __isl_take isl_union_pw_aff *upa2);
__isl_give isl_union_pw_multi_aff *isl_union_pw_multi_aff_sub(
        __isl_take isl_union_pw_multi_aff *upma1,
        __isl_take isl_union_pw_multi_aff *upma2);
__isl_give isl_multi_union_pw_aff *
isl_multi_union_pw_aff_sub(
        __isl_take isl_multi_union_pw_aff *mupa1,
        __isl_take isl_multi_union_pw_aff *mupa2);
```

`isl_aff_sub` subtracts the second argument from the first.

```
#include <isl/polynomial.h>
__isl_give isl_qpolynomial *isl_qpolynomial_add(
        __isl_take isl_qpolynomial *qp1,
        __isl_take isl_qpolynomial *qp2);
__isl_give isl_pw_qpolynomial *isl_pw_qpolynomial_add(
        __isl_take isl_pw_qpolynomial *pwqp1,
        __isl_take isl_pw_qpolynomial *pwqp2);
__isl_give isl_pw_qpolynomial *isl_pw_qpolynomial_add_disjoint(
        __isl_take isl_pw_qpolynomial *pwqp1,
        __isl_take isl_pw_qpolynomial *pwqp2);
__isl_give isl_pw_qpolynomial_fold *isl_pw_qpolynomial_fold_add(
```

```
        __isl_take isl_pw_qpolynomial_fold *pwf1,
        __isl_take isl_pw_qpolynomial_fold *pwf2);
__isl_give isl_union_pw_qpolynomial *isl_union_pw_qpolynomial_add(
        __isl_take isl_union_pw_qpolynomial *upwqp1,
        __isl_take isl_union_pw_qpolynomial *upwqp2);
__isl_give isl_qpolynomial *isl_qpolynomial_sub(
        __isl_take isl_qpolynomial *qp1,
        __isl_take isl_qpolynomial *qp2);
__isl_give isl_pw_qpolynomial *isl_pw_qpolynomial_sub(
        __isl_take isl_pw_qpolynomial *pwqp1,
        __isl_take isl_pw_qpolynomial *pwqp2);
__isl_give isl_union_pw_qpolynomial *isl_union_pw_qpolynomial_sub(
        __isl_take isl_union_pw_qpolynomial *upwqp1,
        __isl_take isl_union_pw_qpolynomial *upwqp2);
__isl_give isl_pw_qpolynomial_fold *isl_pw_qpolynomial_fold_fold(
        __isl_take isl_pw_qpolynomial_fold *pwf1,
        __isl_take isl_pw_qpolynomial_fold *pwf2);
__isl_give isl_union_pw_qpolynomial_fold *
isl_union_pw_qpolynomial_fold_fold(
        __isl_take isl_union_pw_qpolynomial_fold *upwf1,
        __isl_take isl_union_pw_qpolynomial_fold *upwf2);

#include <isl/aff.h>
__isl_give isl_pw_aff *isl_pw_aff_union_add(
        __isl_take isl_pw_aff *pwaff1,
        __isl_take isl_pw_aff *pwaff2);
__isl_give isl_pw_multi_aff *isl_pw_multi_aff_union_add(
        __isl_take isl_pw_multi_aff *pma1,
        __isl_take isl_pw_multi_aff *pma2);
__isl_give isl_union_pw_aff *isl_union_pw_aff_union_add(
        __isl_take isl_union_pw_aff *upa1,
        __isl_take isl_union_pw_aff *upa2);
__isl_give isl_union_pw_multi_aff *
isl_union_pw_multi_aff_union_add(
        __isl_take isl_union_pw_multi_aff *upma1,
        __isl_take isl_union_pw_multi_aff *upma2);
__isl_give isl_multi_union_pw_aff *
isl_multi_union_pw_aff_union_add(
        __isl_take isl_multi_union_pw_aff *mupa1,
        __isl_take isl_multi_union_pw_aff *mupa2);
__isl_give isl_pw_aff *isl_pw_aff_union_min(
        __isl_take isl_pw_aff *pwaff1,
        __isl_take isl_pw_aff *pwaff2);
__isl_give isl_pw_aff *isl_pw_aff_union_max(
        __isl_take isl_pw_aff *pwaff1,
        __isl_take isl_pw_aff *pwaff2);
```

The function isl_pw_aff_union_max computes a piecewise quasi-affine expression with a domain that is the union of those of pwaff1 and pwaff2 and such that on each cell, the quasi-affine expression is the maximum of those of pwaff1 and pwaff2. If only one of pwaff1 or pwaff2 is defined on a given cell, then the associated expression is the defined one. This in contrast to the isl_pw_aff_max function, which is only defined on the shared definition domain of the arguments.

```
#include <isl/val.h>
__isl_give isl_multi_val *isl_multi_val_add_val(
        __isl_take isl_multi_val *mv,
        __isl_take isl_val *v);
__isl_give isl_multi_val *isl_multi_val_mod_val(
        __isl_take isl_multi_val *mv,
        __isl_take isl_val *v);
__isl_give isl_multi_val *isl_multi_val_scale_val(
        __isl_take isl_multi_val *mv,
        __isl_take isl_val *v);
__isl_give isl_multi_val *isl_multi_val_scale_down_val(
        __isl_take isl_multi_val *mv,
        __isl_take isl_val *v);

#include <isl/aff.h>
__isl_give isl_aff *isl_aff_mod_val(__isl_take isl_aff *aff,
        __isl_take isl_val *mod);
__isl_give isl_pw_aff *isl_pw_aff_mod_val(
        __isl_take isl_pw_aff *pa,
        __isl_take isl_val *mod);
__isl_give isl_union_pw_aff *isl_union_pw_aff_mod_val(
        __isl_take isl_union_pw_aff *upa,
        __isl_take isl_val *f);
__isl_give isl_aff *isl_aff_scale_val(__isl_take isl_aff *aff,
        __isl_take isl_val *v);
__isl_give isl_multi_aff *isl_multi_aff_scale_val(
        __isl_take isl_multi_aff *ma,
        __isl_take isl_val *v);
__isl_give isl_pw_aff *isl_pw_aff_scale_val(
        __isl_take isl_pw_aff *pa, __isl_take isl_val *v);
__isl_give isl_multi_pw_aff *isl_multi_pw_aff_scale_val(
        __isl_take isl_multi_pw_aff *mpa,
        __isl_take isl_val *v);
__isl_give isl_pw_multi_aff *isl_pw_multi_aff_scale_val(
        __isl_take isl_pw_multi_aff *pma,
        __isl_take isl_val *v);
__isl_give isl_union_pw_multi_aff *
__isl_give isl_union_pw_aff *isl_union_pw_aff_scale_val(
```

```
        __isl_take isl_union_pw_aff *upa,
        __isl_take isl_val *f);
isl_union_pw_multi_aff_scale_val(
        __isl_take isl_union_pw_multi_aff *upma,
        __isl_take isl_val *val);
__isl_give isl_multi_union_pw_aff *
isl_multi_union_pw_aff_scale_val(
        __isl_take isl_multi_union_pw_aff *mupa,
        __isl_take isl_val *v);
__isl_give isl_aff *isl_aff_scale_down_ui(
        __isl_take isl_aff *aff, unsigned f);
__isl_give isl_aff *isl_aff_scale_down_val(
        __isl_take isl_aff *aff, __isl_take isl_val *v);
__isl_give isl_multi_aff *isl_multi_aff_scale_down_val(
        __isl_take isl_multi_aff *ma,
        __isl_take isl_val *v);
__isl_give isl_pw_aff *isl_pw_aff_scale_down_val(
        __isl_take isl_pw_aff *pa,
        __isl_take isl_val *f);
__isl_give isl_multi_pw_aff *isl_multi_pw_aff_scale_down_val(
        __isl_take isl_multi_pw_aff *mpa,
        __isl_take isl_val *v);
__isl_give isl_pw_multi_aff *isl_pw_multi_aff_scale_down_val(
        __isl_take isl_pw_multi_aff *pma,
        __isl_take isl_val *v);
__isl_give isl_union_pw_aff *isl_union_pw_aff_scale_down_val(
        __isl_take isl_union_pw_aff *upa,
        __isl_take isl_val *v);
__isl_give isl_union_pw_multi_aff *
isl_union_pw_multi_aff_scale_down_val(
        __isl_take isl_union_pw_multi_aff *upma,
        __isl_take isl_val *val);
__isl_give isl_multi_union_pw_aff *
isl_multi_union_pw_aff_scale_down_val(
        __isl_take isl_multi_union_pw_aff *mupa,
        __isl_take isl_val *v);

#include <isl/polynomial.h>
__isl_give isl_qpolynomial *isl_qpolynomial_scale_val(
        __isl_take isl_qpolynomial *qp,
        __isl_take isl_val *v);
__isl_give isl_qpolynomial_fold *
isl_qpolynomial_fold_scale_val(
        __isl_take isl_qpolynomial_fold *fold,
        __isl_take isl_val *v);
__isl_give isl_pw_qpolynomial *
```

```
isl_pw_qpolynomial_scale_val(
        __isl_take isl_pw_qpolynomial *pwqp,
        __isl_take isl_val *v);
__isl_give isl_pw_qpolynomial_fold *
isl_pw_qpolynomial_fold_scale_val(
        __isl_take isl_pw_qpolynomial_fold *pwf,
        __isl_take isl_val *v);
__isl_give isl_union_pw_qpolynomial *
isl_union_pw_qpolynomial_scale_val(
        __isl_take isl_union_pw_qpolynomial *upwqp,
        __isl_take isl_val *v);
__isl_give isl_union_pw_qpolynomial_fold *
isl_union_pw_qpolynomial_fold_scale_val(
        __isl_take isl_union_pw_qpolynomial_fold *upwf,
        __isl_take isl_val *v);
__isl_give isl_qpolynomial *
isl_qpolynomial_scale_down_val(
        __isl_take isl_qpolynomial *qp,
        __isl_take isl_val *v);
__isl_give isl_qpolynomial_fold *
isl_qpolynomial_fold_scale_down_val(
        __isl_take isl_qpolynomial_fold *fold,
        __isl_take isl_val *v);
__isl_give isl_pw_qpolynomial *
isl_pw_qpolynomial_scale_down_val(
        __isl_take isl_pw_qpolynomial *pwqp,
        __isl_take isl_val *v);
__isl_give isl_pw_qpolynomial_fold *
isl_pw_qpolynomial_fold_scale_down_val(
        __isl_take isl_pw_qpolynomial_fold *pwf,
        __isl_take isl_val *v);
__isl_give isl_union_pw_qpolynomial *
isl_union_pw_qpolynomial_scale_down_val(
        __isl_take isl_union_pw_qpolynomial *upwqp,
        __isl_take isl_val *v);
__isl_give isl_union_pw_qpolynomial_fold *
isl_union_pw_qpolynomial_fold_scale_down_val(
        __isl_take isl_union_pw_qpolynomial_fold *upwf,
        __isl_take isl_val *v);

#include <isl/val.h>
__isl_give isl_multi_val *isl_multi_val_mod_multi_val(
        __isl_take isl_multi_val *mv1,
        __isl_take isl_multi_val *mv2);
__isl_give isl_multi_val *isl_multi_val_scale_multi_val(
        __isl_take isl_multi_val *mv1,
```

```
        __isl_take isl_multi_val *mv2);
__isl_give isl_multi_val *
isl_multi_val_scale_down_multi_val(
        __isl_take isl_multi_val *mv1,
        __isl_take isl_multi_val *mv2);

#include <isl/aff.h>
__isl_give isl_multi_aff *isl_multi_aff_mod_multi_val(
        __isl_take isl_multi_aff *ma,
        __isl_take isl_multi_val *mv);
__isl_give isl_multi_union_pw_aff *
isl_multi_union_pw_aff_mod_multi_val(
        __isl_take isl_multi_union_pw_aff *upma,
        __isl_take isl_multi_val *mv);
__isl_give isl_multi_pw_aff *
isl_multi_pw_aff_mod_multi_val(
        __isl_take isl_multi_pw_aff *mpa,
        __isl_take isl_multi_val *mv);
__isl_give isl_multi_aff *isl_multi_aff_scale_multi_val(
        __isl_take isl_multi_aff *ma,
        __isl_take isl_multi_val *mv);
__isl_give isl_pw_multi_aff *
isl_pw_multi_aff_scale_multi_val(
        __isl_take isl_pw_multi_aff *pma,
        __isl_take isl_multi_val *mv);
__isl_give isl_multi_pw_aff *
isl_multi_pw_aff_scale_multi_val(
        __isl_take isl_multi_pw_aff *mpa,
        __isl_take isl_multi_val *mv);
__isl_give isl_multi_union_pw_aff *
isl_multi_union_pw_aff_scale_multi_val(
        __isl_take isl_multi_union_pw_aff *mupa,
        __isl_take isl_multi_val *mv);
__isl_give isl_union_pw_multi_aff *
isl_union_pw_multi_aff_scale_multi_val(
        __isl_take isl_union_pw_multi_aff *upma,
        __isl_take isl_multi_val *mv);
__isl_give isl_multi_aff *
isl_multi_aff_scale_down_multi_val(
        __isl_take isl_multi_aff *ma,
        __isl_take isl_multi_val *mv);
__isl_give isl_multi_pw_aff *
isl_multi_pw_aff_scale_down_multi_val(
        __isl_take isl_multi_pw_aff *mpa,
        __isl_take isl_multi_val *mv);
__isl_give isl_multi_union_pw_aff *
```

```
isl_multi_union_pw_aff_scale_down_multi_val(
        __isl_take isl_multi_union_pw_aff *mupa,
        __isl_take isl_multi_val *mv);
```

isl_multi_aff_scale_multi_val scales the elements of ma by the corresponding elements of mv.

```
#include <isl/aff.h>
__isl_give isl_aff *isl_aff_mul(
        __isl_take isl_aff *aff1,
        __isl_take isl_aff *aff2);
__isl_give isl_aff *isl_aff_div(
        __isl_take isl_aff *aff1,
        __isl_take isl_aff *aff2);
__isl_give isl_pw_aff *isl_pw_aff_mul(
        __isl_take isl_pw_aff *pwaff1,
        __isl_take isl_pw_aff *pwaff2);
__isl_give isl_pw_aff *isl_pw_aff_div(
        __isl_take isl_pw_aff *pa1,
        __isl_take isl_pw_aff *pa2);
__isl_give isl_pw_aff *isl_pw_aff_tdiv_q(
        __isl_take isl_pw_aff *pa1,
        __isl_take isl_pw_aff *pa2);
__isl_give isl_pw_aff *isl_pw_aff_tdiv_r(
        __isl_take isl_pw_aff *pa1,
        __isl_take isl_pw_aff *pa2);
```

When multiplying two affine expressions, at least one of the two needs to be a constant. Similarly, when dividing an affine expression by another, the second expression needs to be a constant. isl_pw_aff_tdiv_q computes the quotient of an integer division with rounding towards zero. isl_pw_aff_tdiv_r computes the corresponding remainder.

```
#include <isl/polynomial.h>
__isl_give isl_qpolynomial *isl_qpolynomial_mul(
        __isl_take isl_qpolynomial *qp1,
        __isl_take isl_qpolynomial *qp2);
__isl_give isl_pw_qpolynomial *isl_pw_qpolynomial_mul(
        __isl_take isl_pw_qpolynomial *pwqp1,
        __isl_take isl_pw_qpolynomial *pwqp2);
__isl_give isl_union_pw_qpolynomial *isl_union_pw_qpolynomial_mul(
        __isl_take isl_union_pw_qpolynomial *upwqp1,
        __isl_take isl_union_pw_qpolynomial *upwqp2);
```

Lexicographic Optimization

Given a (basic) set set (or bset) and a zero-dimensional domain dom, the following functions compute a set that contains the lexicographic minimum or maximum of the elements in set (or bset) for those values of the parameters that satisfy dom. If empty is not NULL, then *empty is assigned a set that contains the parameter values in dom for which set (or bset) has no elements. In other words, the union of the parameter values for which the result is non-empty and of *empty is equal to dom.

```
#include <isl/set.h>
__isl_give isl_set *isl_basic_set_partial_lexmin(
        __isl_take isl_basic_set *bset,
        __isl_take isl_basic_set *dom,
        __isl_give isl_set **empty);
__isl_give isl_set *isl_basic_set_partial_lexmax(
        __isl_take isl_basic_set *bset,
        __isl_take isl_basic_set *dom,
        __isl_give isl_set **empty);
__isl_give isl_set *isl_set_partial_lexmin(
        __isl_take isl_set *set, __isl_take isl_set *dom,
        __isl_give isl_set **empty);
__isl_give isl_set *isl_set_partial_lexmax(
        __isl_take isl_set *set, __isl_take isl_set *dom,
        __isl_give isl_set **empty);
```

Given a (basic) set set (or bset), the following functions simply return a set containing the lexicographic minimum or maximum of the elements in set (or bset). In case of union sets, the optimum is computed per space.

```
#include <isl/set.h>
__isl_give isl_set *isl_basic_set_lexmin(
        __isl_take isl_basic_set *bset);
__isl_give isl_set *isl_basic_set_lexmax(
        __isl_take isl_basic_set *bset);
__isl_give isl_set *isl_set_lexmin(
        __isl_take isl_set *set);
__isl_give isl_set *isl_set_lexmax(
        __isl_take isl_set *set);
__isl_give isl_union_set *isl_union_set_lexmin(
        __isl_take isl_union_set *uset);
__isl_give isl_union_set *isl_union_set_lexmax(
        __isl_take isl_union_set *uset);
```

Given a (basic) relation map (or bmap) and a domain dom, the following functions compute a relation that maps each element of dom to the single lexicographic minimum or maximum of the elements that are associated to that same element in map (or bmap). If empty is not NULL, then *empty is assigned a set that contains the elements in dom

that do not map to any elements in map (or bmap). In other words, the union of the domain of the result and of *empty is equal to dom.

```
#include <isl/map.h>
__isl_give isl_map *isl_basic_map_partial_lexmax(
        __isl_take isl_basic_map *bmap,
        __isl_take isl_basic_set *dom,
        __isl_give isl_set **empty);
__isl_give isl_map *isl_basic_map_partial_lexmin(
        __isl_take isl_basic_map *bmap,
        __isl_take isl_basic_set *dom,
        __isl_give isl_set **empty);
__isl_give isl_map *isl_map_partial_lexmax(
        __isl_take isl_map *map, __isl_take isl_set *dom,
        __isl_give isl_set **empty);
__isl_give isl_map *isl_map_partial_lexmin(
        __isl_take isl_map *map, __isl_take isl_set *dom,
        __isl_give isl_set **empty);
```

Given a (basic) map map (or bmap), the following functions simply return a map mapping each element in the domain of map (or bmap) to the lexicographic minimum or maximum of all elements associated to that element. In case of union relations, the optimum is computed per space.

```
#include <isl/map.h>
__isl_give isl_map *isl_basic_map_lexmin(
        __isl_take isl_basic_map *bmap);
__isl_give isl_map *isl_basic_map_lexmax(
        __isl_take isl_basic_map *bmap);
__isl_give isl_map *isl_map_lexmin(
        __isl_take isl_map *map);
__isl_give isl_map *isl_map_lexmax(
        __isl_take isl_map *map);
__isl_give isl_union_map *isl_union_map_lexmin(
        __isl_take isl_union_map *umap);
__isl_give isl_union_map *isl_union_map_lexmax(
        __isl_take isl_union_map *umap);
```

The following functions return their result in the form of a piecewise multi-affine expression, but are otherwise equivalent to the corresponding functions returning a basic set or relation.

```
#include <isl/set.h>
__isl_give isl_pw_multi_aff *
isl_basic_set_partial_lexmin_pw_multi_aff(
        __isl_take isl_basic_set *bset,
        __isl_take isl_basic_set *dom,
```

```
        __isl_give isl_set **empty);
__isl_give isl_pw_multi_aff *
isl_basic_set_partial_lexmax_pw_multi_aff(
        __isl_take isl_basic_set *bset,
        __isl_take isl_basic_set *dom,
        __isl_give isl_set **empty);
__isl_give isl_pw_multi_aff *isl_set_lexmin_pw_multi_aff(
        __isl_take isl_set *set);
__isl_give isl_pw_multi_aff *isl_set_lexmax_pw_multi_aff(
        __isl_take isl_set *set);

#include <isl/map.h>
__isl_give isl_pw_multi_aff *
isl_basic_map_lexmin_pw_multi_aff(
        __isl_take isl_basic_map *bmap);
__isl_give isl_pw_multi_aff *
isl_basic_map_partial_lexmin_pw_multi_aff(
        __isl_take isl_basic_map *bmap,
        __isl_take isl_basic_set *dom,
        __isl_give isl_set **empty);
__isl_give isl_pw_multi_aff *
isl_basic_map_partial_lexmax_pw_multi_aff(
        __isl_take isl_basic_map *bmap,
        __isl_take isl_basic_set *dom,
        __isl_give isl_set **empty);
__isl_give isl_pw_multi_aff *isl_map_lexmin_pw_multi_aff(
        __isl_take isl_map *map);
__isl_give isl_pw_multi_aff *isl_map_lexmax_pw_multi_aff(
        __isl_take isl_map *map);
```

The following functions return the lexicographic minimum or maximum on the shared domain of the inputs and the single defined function on those parts of the domain where only a single function is defined.

```
#include <isl/aff.h>
__isl_give isl_pw_multi_aff *isl_pw_multi_aff_union_lexmin(
        __isl_take isl_pw_multi_aff *pma1,
        __isl_take isl_pw_multi_aff *pma2);
__isl_give isl_pw_multi_aff *isl_pw_multi_aff_union_lexmax(
        __isl_take isl_pw_multi_aff *pma1,
        __isl_take isl_pw_multi_aff *pma2);
```

1.4.18 Ternary Operations

```
#include <isl/aff.h>
__isl_give isl_pw_aff *isl_pw_aff_cond(
```

```
        __isl_take isl_pw_aff *cond,
        __isl_take isl_pw_aff *pwaff_true,
        __isl_take isl_pw_aff *pwaff_false);
```

The function isl_pw_aff_cond performs a conditional operator and returns an expression that is equal to pwaff_true for elements where cond is non-zero and equal to pwaff_false for elements where cond is zero.

1.4.19 Lists

Lists are defined over several element types, including isl_val, isl_id, isl_aff, isl_pw_aff, isl_union_pw_aff, isl_union_pw_multi_aff, isl_constraint, isl_basic_set, isl_set, isl_basic_map, isl_map, isl_union_set, isl_union_map, isl_ast_expr and isl_ast_node. Here we take lists of isl_sets as an example. Lists can be created, copied, modified and freed using the following functions.

```
#include <isl/set.h>
__isl_give isl_set_list *isl_set_list_from_set(
        __isl_take isl_set *el);
__isl_give isl_set_list *isl_set_list_alloc(
        isl_ctx *ctx, int n);
__isl_give isl_set_list *isl_set_list_copy(
        __isl_keep isl_set_list *list);
__isl_give isl_set_list *isl_set_list_insert(
        __isl_take isl_set_list *list, unsigned pos,
        __isl_take isl_set *el);
__isl_give isl_set_list *isl_set_list_add(
        __isl_take isl_set_list *list,
        __isl_take isl_set *el);
__isl_give isl_set_list *isl_set_list_drop(
        __isl_take isl_set_list *list,
        unsigned first, unsigned n);
__isl_give isl_set_list *isl_set_list_set_set(
        __isl_take isl_set_list *list, int index,
        __isl_take isl_set *set);
__isl_give isl_set_list *isl_set_list_concat(
        __isl_take isl_set_list *list1,
        __isl_take isl_set_list *list2);
__isl_give isl_set_list *isl_set_list_sort(
        __isl_take isl_set_list *list,
        int (*cmp)(__isl_keep isl_set *a,
                __isl_keep isl_set *b, void *user),
        void *user);
__isl_null isl_set_list *isl_set_list_free(
        __isl_take isl_set_list *list);
```

isl_set_list_alloc creates an empty list with an initial capacity for n elements. isl_set_list_insert and isl_set_list_add add elements to a list, increasing its capacity as needed. isl_set_list_from_set creates a list with a single element.

Lists can be inspected using the following functions.

```
#include <isl/set.h>
int isl_set_list_n_set(__isl_keep isl_set_list *list);
__isl_give isl_set *isl_set_list_get_set(
        __isl_keep isl_set_list *list, int index);
isl_stat isl_set_list_foreach(__isl_keep isl_set_list *list,
        isl_stat (*fn)(__isl_take isl_set *el, void *user),
        void *user);
isl_stat isl_set_list_foreach_scc(
        __isl_keep isl_set_list *list,
        isl_bool (*follows)(__isl_keep isl_set *a,
                __isl_keep isl_set *b, void *user),
        void *follows_user,
        isl_stat (*fn)(__isl_take isl_set *el, void *user),
        void *fn_user);
```

The function isl_set_list_foreach_scc calls fn on each of the strongly connected components of the graph with as vertices the elements of list and a directed edge from vertex b to vertex a iff follows(a, b) returns isl_bool_true. The callbacks follows and fn should return isl_bool_error or isl_stat_error on error.

Lists can be printed using

```
#include <isl/set.h>
__isl_give isl_printer *isl_printer_print_set_list(
        __isl_take isl_printer *p,
        __isl_keep isl_set_list *list);
```

1.4.20 Associative arrays

Associative arrays map isl objects of a specific type to isl objects of some (other) specific type. They are defined for several pairs of types, including (isl_map, isl_basic_set), (isl_id, isl_ast_expr), (isl_id, isl_id) and (isl_id, isl_pw_aff). Here, we take associative arrays that map isl_ids to isl_ast_exprs as an example.

Associative arrays can be created, copied and freed using the following functions.

```
#include <isl/id_to_ast_expr.h>
__isl_give isl_id_to_ast_expr *isl_id_to_ast_expr_alloc(
        isl_ctx *ctx, int min_size);
__isl_give isl_id_to_ast_expr *isl_id_to_ast_expr_copy(
        __isl_keep isl_id_to_ast_expr *id2expr);
__isl_null isl_id_to_ast_expr *isl_id_to_ast_expr_free(
        __isl_take isl_id_to_ast_expr *id2expr);
```

The min_size argument to isl_id_to_ast_expr_alloc can be used to specify the expected size of the associative array. The associative array will be grown automatically as needed.

Associative arrays can be inspected using the following functions.

```
#include <isl/id_to_ast_expr.h>
__isl_give isl_maybe_isl_ast_expr
isl_id_to_ast_expr_try_get(
        __isl_keep isl_id_to_ast_expr *id2expr,
        __isl_keep isl_id *key);
isl_bool isl_id_to_ast_expr_has(
        __isl_keep isl_id_to_ast_expr *id2expr,
        __isl_keep isl_id *key);
__isl_give isl_ast_expr *isl_id_to_ast_expr_get(
        __isl_keep isl_id_to_ast_expr *id2expr,
        __isl_take isl_id *key);
isl_stat isl_id_to_ast_expr_foreach(
        __isl_keep isl_id_to_ast_expr *id2expr,
        isl_stat (*fn)(__isl_take isl_id *key,
                __isl_take isl_ast_expr *val, void *user),
        void *user);
```

The function isl_id_to_ast_expr_try_get returns a structure containing two elements, valid and value. If there is a value associated to the key, then valid is set to isl_bool_true and value contains a copy of the associated value. Otherwise value is NULL and valid may be isl_bool_error or isl_bool_false depending on whether some error has occurred or there simply is no associated value. The function isl_id_to_ast_expr_has returns the valid field in the structure and the function isl_id_to_ast_expr_get returns the value field.

Associative arrays can be modified using the following functions.

```
#include <isl/id_to_ast_expr.h>
__isl_give isl_id_to_ast_expr *isl_id_to_ast_expr_set(
        __isl_take isl_id_to_ast_expr *id2expr,
        __isl_take isl_id *key,
        __isl_take isl_ast_expr *val);
__isl_give isl_id_to_ast_expr *isl_id_to_ast_expr_drop(
        __isl_take isl_id_to_ast_expr *id2expr,
        __isl_take isl_id *key);
```

Associative arrays can be printed using the following function.

```
#include <isl/id_to_ast_expr.h>
__isl_give isl_printer *isl_printer_print_id_to_ast_expr(
        __isl_take isl_printer *p,
        __isl_keep isl_id_to_ast_expr *id2expr);
```

1.4.21 Vectors

Vectors can be created, copied and freed using the following functions.

```
#include <isl/vec.h>
__isl_give isl_vec *isl_vec_alloc(isl_ctx *ctx,
        unsigned size);
__isl_give isl_vec *isl_vec_copy(__isl_keep isl_vec *vec);
__isl_null isl_vec *isl_vec_free(__isl_take isl_vec *vec);
```

Note that the elements of a newly created vector may have arbitrary values. The elements can be changed and inspected using the following functions.

```
int isl_vec_size(__isl_keep isl_vec *vec);
__isl_give isl_val *isl_vec_get_element_val(
        __isl_keep isl_vec *vec, int pos);
__isl_give isl_vec *isl_vec_set_element_si(
        __isl_take isl_vec *vec, int pos, int v);
__isl_give isl_vec *isl_vec_set_element_val(
        __isl_take isl_vec *vec, int pos,
        __isl_take isl_val *v);
__isl_give isl_vec *isl_vec_set_si(__isl_take isl_vec *vec,
        int v);
__isl_give isl_vec *isl_vec_set_val(
        __isl_take isl_vec *vec, __isl_take isl_val *v);
int isl_vec_cmp_element(__isl_keep isl_vec *vec1,
        __isl_keep isl_vec *vec2, int pos);
```

`isl_vec_get_element` will return a negative value if anything went wrong. In that case, the value of `*v` is undefined.

The following function can be used to concatenate two vectors.

```
__isl_give isl_vec *isl_vec_concat(__isl_take isl_vec *vec1,
        __isl_take isl_vec *vec2);
```

1.4.22 Matrices

Matrices can be created, copied and freed using the following functions.

```
#include <isl/mat.h>
__isl_give isl_mat *isl_mat_alloc(isl_ctx *ctx,
        unsigned n_row, unsigned n_col);
__isl_give isl_mat *isl_mat_copy(__isl_keep isl_mat *mat);
__isl_null isl_mat *isl_mat_free(__isl_take isl_mat *mat);
```

Note that the elements of a newly created matrix may have arbitrary values. The elements can be changed and inspected using the following functions.

```
int isl_mat_rows(__isl_keep isl_mat *mat);
int isl_mat_cols(__isl_keep isl_mat *mat);
__isl_give isl_val *isl_mat_get_element_val(
        __isl_keep isl_mat *mat, int row, int col);
__isl_give isl_mat *isl_mat_set_element_si(__isl_take isl_mat *mat,
        int row, int col, int v);
__isl_give isl_mat *isl_mat_set_element_val(
        __isl_take isl_mat *mat, int row, int col,
        __isl_take isl_val *v);
```

`isl_mat_get_element` will return a negative value if anything went wrong. In that case, the value of `*v` is undefined.

The following function can be used to compute the (right) inverse of a matrix, i.e., a matrix such that the product of the original and the inverse (in that order) is a multiple of the identity matrix. The input matrix is assumed to be of full row-rank.

```
__isl_give isl_mat *isl_mat_right_inverse(__isl_take isl_mat *mat);
```

The following function can be used to compute the (right) kernel (or null space) of a matrix, i.e., a matrix such that the product of the original and the kernel (in that order) is the zero matrix.

```
__isl_give isl_mat *isl_mat_right_kernel(__isl_take isl_mat *mat);
```

1.4.23 Bounds on Piecewise Quasipolynomials and Piecewise Quasipolynomial Reductions

The following functions determine an upper or lower bound on a quasipolynomial over its domain.

```
__isl_give isl_pw_qpolynomial_fold *
isl_pw_qpolynomial_bound(
        __isl_take isl_pw_qpolynomial *pwqp,
        enum isl_fold type, int *tight);

__isl_give isl_union_pw_qpolynomial_fold *
isl_union_pw_qpolynomial_bound(
        __isl_take isl_union_pw_qpolynomial *upwqp,
        enum isl_fold type, int *tight);
```

The `type` argument may be either `isl_fold_min` or `isl_fold_max`. If `tight` is not `NULL`, then `*tight` is set to 1 is the returned bound is known be tight, i.e., for each value of the parameters there is at least one element in the domain that reaches the bound. If the domain of `pwqp` is not wrapping, then the bound is computed over all elements in that domain and the result has a purely parametric domain. If the domain of `pwqp` is wrapping, then the bound is computed over the range of the wrapped relation. The domain of the wrapped relation becomes the domain of the result.

1.4.24 Parametric Vertex Enumeration

The parametric vertex enumeration described in this section is mainly intended to be used internally and by the barvinok library.

```
#include <isl/vertices.h>
__isl_give isl_vertices *isl_basic_set_compute_vertices(
        __isl_keep isl_basic_set *bset);
```

The function isl_basic_set_compute_vertices performs the actual computation of the parametric vertices and the chamber decomposition and store the result in an isl_vertices object. This information can be queried by either iterating over all the vertices or iterating over all the chambers or cells and then iterating over all vertices that are active on the chamber.

```
isl_stat isl_vertices_foreach_vertex(
        __isl_keep isl_vertices *vertices,
        isl_stat (*fn)(__isl_take isl_vertex *vertex,
                void *user), void *user);

isl_stat isl_vertices_foreach_cell(
        __isl_keep isl_vertices *vertices,
        isl_stat (*fn)(__isl_take isl_cell *cell,
                void *user), void *user);
isl_stat isl_cell_foreach_vertex(__isl_keep isl_cell *cell,
        isl_stat (*fn)(__isl_take isl_vertex *vertex,
                void *user), void *user);
```

Other operations that can be performed on an isl_vertices object are the following.

```
int isl_vertices_get_n_vertices(
        __isl_keep isl_vertices *vertices);
void isl_vertices_free(__isl_take isl_vertices *vertices);
```

Vertices can be inspected and destroyed using the following functions.

```
int isl_vertex_get_id(__isl_keep isl_vertex *vertex);
__isl_give isl_basic_set *isl_vertex_get_domain(
        __isl_keep isl_vertex *vertex);
__isl_give isl_multi_aff *isl_vertex_get_expr(
        __isl_keep isl_vertex *vertex);
void isl_vertex_free(__isl_take isl_vertex *vertex);
```

isl_vertex_get_expr returns a multiple quasi-affine expression describing the vertex in terms of the parameters, while isl_vertex_get_domain returns the activity domain of the vertex.

Chambers can be inspected and destroyed using the following functions.

```
__isl_give isl_basic_set *isl_cell_get_domain(
        __isl_keep isl_cell *cell);
void isl_cell_free(__isl_take isl_cell *cell);
```

1.5 Polyhedral Compilation Library

This section collects functionality in `isl` that has been specifically designed for use during polyhedral compilation.

1.5.1 Schedule Trees

A schedule tree is a structured representation of a schedule, assigning a relative order to a set of domain elements. The relative order expressed by the schedule tree is defined recursively. In particular, the order between two domain elements is determined by the node that is closest to the root that refers to both elements and that orders them apart. Each node in the tree is of one of several types. The root node is always of type `isl_schedule_node_domain` (or `isl_schedule_node_extension`) and it describes the (extra) domain elements to which the schedule applies. The other types of nodes are as follows.

`isl_schedule_node_band`

A band of schedule dimensions. Each schedule dimension is represented by a union piecewise quasi-affine expression. If this expression assigns a different value to two domain elements, while all previous schedule dimensions in the same band assign them the same value, then the two domain elements are ordered according to these two different values. Each expression is required to be total in the domain elements that reach the band node.

`isl_schedule_node_expansion`

An expansion node maps each of the domain elements that reach the node to one or more domain elements. The image of this mapping forms the set of domain elements that reach the child of the expansion node. The function that maps each of the expanded domain elements to the original domain element from which it was expanded is called the contraction.

`isl_schedule_node_filter`

A filter node does not impose any ordering, but rather intersects the set of domain elements that the current subtree refers to with a given union set. The subtree of the filter node only refers to domain elements in the intersection. A filter node is typically only used a child of a sequence or set node.

`isl_schedule_node_leaf`

A leaf of the schedule tree. Leaf nodes do not impose any ordering.

`isl_schedule_node_mark`

A mark node can be used to attach any kind of information to a subtree of the schedule tree.

`isl_schedule_node_sequence`

A sequence node has one or more children, each of which is a filter node. The filters on these filter nodes form a partition of the domain elements that the current subtree refers to. If two domain elements appear in distinct filters then the sequence node orders them according to the child positions of the corresponding filter nodes.

`isl_schedule_node_set`

A set node is similar to a sequence node, except that it expresses that domain elements appearing in distinct filters may have any order. The order of the children of a set node is therefore also immaterial.

The following node types are only supported by the AST generator.

`isl_schedule_node_context`

The context describes constraints on the parameters and the schedule dimensions of outer bands that the AST generator may assume to hold. It is also the only kind of node that may introduce additional parameters. The space of the context is that of the flat product of the outer band nodes. In particular, if there are no outer band nodes, then this space is the unnamed zero-dimensional space. Since a context node references the outer band nodes, any tree containing a context node is considered to be anchored.

`isl_schedule_node_extension`

An extension node instructs the AST generator to add additional domain elements that need to be scheduled. The additional domain elements are described by the range of the extension map in terms of the outer schedule dimensions, i.e., the flat product of the outer band nodes. Note that domain elements are added whenever the AST generator reaches the extension node, meaning that there are still some active domain elements for which an AST needs to be generated. The conditions under which some domain elements are still active may however not be completely described by the outer AST nodes generated at that point.

An extension node may also appear as the root of a schedule tree, when it is intended to be inserted into another tree using `isl_schedule_node_graft_before` or `isl_schedule_node_graft_after`. In this case, the domain of the extension node should correspond to the flat product of the outer band nodes in this other schedule tree at the point where the extension tree will be inserted.

`isl_schedule_node_guard`

The guard describes constraints on the parameters and the schedule dimensions of outer bands that need to be enforced by the outer nodes in the generated AST. The space of the guard is that of the flat product of the outer band nodes. In particular, if there are no outer band nodes, then this space is the unnamed zero-dimensional space. Since a guard node references the outer band nodes, any tree containing a guard node is considered to be anchored.

Except for the isl_schedule_node_context nodes, none of the nodes may introduce any parameters that were not already present in the root domain node.

A schedule tree is encapsulated in an isl_schedule object. The simplest such objects, those with a tree consisting of single domain node, can be created using the following functions with either an empty domain or a given domain.

```
#include <isl/schedule.h>
__isl_give isl_schedule *isl_schedule_empty(
        __isl_take isl_space *space);
__isl_give isl_schedule *isl_schedule_from_domain(
        __isl_take isl_union_set *domain);
```

The function isl_schedule_constraints_compute_schedule described in §1.5.3 can also be used to construct schedules.

isl_schedule objects may be copied and freed using the following functions.

```
#include <isl/schedule.h>
__isl_give isl_schedule *isl_schedule_copy(
        __isl_keep isl_schedule *sched);
__isl_null isl_schedule *isl_schedule_free(
        __isl_take isl_schedule *sched);
```

The following functions checks whether two isl_schedule objects are obviously the same.

```
#include <isl/schedule.h>
isl_bool isl_schedule_plain_is_equal(
        __isl_keep isl_schedule *schedule1,
        __isl_keep isl_schedule *schedule2);
```

The domain of the schedule, i.e., the domain described by the root node, can be obtained using the following function.

```
#include <isl/schedule.h>
__isl_give isl_union_set *isl_schedule_get_domain(
        __isl_keep isl_schedule *schedule);
```

An extra top-level band node (right underneath the domain node) can be introduced into the schedule using the following function. The schedule tree is assumed not to have any anchored nodes.

```
#include <isl/schedule.h>
__isl_give isl_schedule *
isl_schedule_insert_partial_schedule(
        __isl_take isl_schedule *schedule,
        __isl_take isl_multi_union_pw_aff *partial);
```

A top-level context node (right underneath the domain node) can be introduced into the schedule using the following function.

```
#include <isl/schedule.h>
__isl_give isl_schedule *isl_schedule_insert_context(
        __isl_take isl_schedule *schedule,
        __isl_take isl_set *context)
```

A top-level guard node (right underneath the domain node) can be introduced into the schedule using the following function.

```
#include <isl/schedule.h>
__isl_give isl_schedule *isl_schedule_insert_guard(
        __isl_take isl_schedule *schedule,
        __isl_take isl_set *guard)
```

A schedule that combines two schedules either in the given order or in an arbitrary order, i.e., with an `isl_schedule_node_sequence` or an `isl_schedule_node_set` node, can be created using the following functions.

```
#include <isl/schedule.h>
__isl_give isl_schedule *isl_schedule_sequence(
        __isl_take isl_schedule *schedule1,
        __isl_take isl_schedule *schedule2);
__isl_give isl_schedule *isl_schedule_set(
        __isl_take isl_schedule *schedule1,
        __isl_take isl_schedule *schedule2);
```

The domains of the two input schedules need to be disjoint.

The following function can be used to restrict the domain of a schedule with a domain node as root to be a subset of the given union set. This operation may remove nodes in the tree that have become redundant.

```
#include <isl/schedule.h>
__isl_give isl_schedule *isl_schedule_intersect_domain(
        __isl_take isl_schedule *schedule,
        __isl_take isl_union_set *domain);
```

The following function can be used to simplify the domain of a schedule with a domain node as root with respect to the given parameter domain.

```
#include <isl/schedule.h>
__isl_give isl_schedule *isl_schedule_gist_domain_params(
        __isl_take isl_schedule *schedule,
        __isl_take isl_set *context);
```

The following function resets the user pointers on all parameter and tuple identifiers referenced by the nodes of the given schedule.

```
#include <isl/schedule.h>
__isl_give isl_schedule *isl_schedule_reset_user(
        __isl_take isl_schedule *schedule);
```

The following function aligns the parameters of all nodes in the given schedule to the given space.

```
#include <isl/schedule.h>
__isl_give isl_schedule *isl_schedule_align_params(
        __isl_take isl_schedule *schedule,
        __isl_take isl_space *space);
```

The following function allows the user to plug in a given function in the iteration domains. The input schedule is not allowed to contain any expansion nodes.

```
#include <isl/schedule.h>
__isl_give isl_schedule *
isl_schedule_pullback_union_pw_multi_aff(
        __isl_take isl_schedule *schedule,
        __isl_take isl_union_pw_multi_aff *upma);
```

The following function can be used to plug in the schedule `expansion` in the leaves of `schedule`, where `contraction` describes how the domain elements of `expansion` map to the domain elements at the original leaves of `schedule`. The resulting schedule will contain expansion nodes, unless `contraction` is an identity function.

```
#include <isl/schedule.h>
__isl_give isl_schedule *isl_schedule_expand(
        __isl_take isl_schedule *schedule,
        __isl_take isl_union_pw_multi_aff *contraction,
        __isl_take isl_schedule *expansion);
```

An `isl_union_map` representation of the schedule can be obtained from an `isl_schedule` using the following function.

```
#include <isl/schedule.h>
__isl_give isl_union_map *isl_schedule_get_map(
        __isl_keep isl_schedule *sched);
```

The resulting relation encodes the same relative ordering as the schedule by mapping the domain elements to a common schedule space. If the schedule_separate_components option is set, then the order of the children of a set node is explicitly encoded in the result. If the tree contains any expansion nodes, then the relation is formulated in terms of the expanded domain elements.

Schedules can be read from input using the following functions.

```
#include <isl/schedule.h>
__isl_give isl_schedule *isl_schedule_read_from_file(
        isl_ctx *ctx, FILE *input);
__isl_give isl_schedule *isl_schedule_read_from_str(
        isl_ctx *ctx, const char *str);
```

A representation of the schedule can be printed using

```
#include <isl/schedule.h>
__isl_give isl_printer *isl_printer_print_schedule(
        __isl_take isl_printer *p,
        __isl_keep isl_schedule *schedule);
__isl_give char *isl_schedule_to_str(
        __isl_keep isl_schedule *schedule);
```

`isl_schedule_to_str` prints the schedule in flow format.

The schedule tree can be traversed through the use of `isl_schedule_node` objects that point to a particular position in the schedule tree. Whenever a `isl_schedule_node` is use to modify a node in the schedule tree, the original schedule tree is left untouched and the modifications are performed to a copy of the tree. The returned `isl_schedule_node` then points to this modified copy of the tree.

The root of the schedule tree can be obtained using the following function.

```
#include <isl/schedule.h>
__isl_give isl_schedule_node *isl_schedule_get_root(
        __isl_keep isl_schedule *schedule);
```

A pointer to a newly created schedule tree with a single domain node can be created using the following functions.

```
#include <isl/schedule_node.h>
__isl_give isl_schedule_node *
isl_schedule_node_from_domain(
        __isl_take isl_union_set *domain);
__isl_give isl_schedule_node *
isl_schedule_node_from_extension(
        __isl_take isl_union_map *extension);
```

`isl_schedule_node_from_extension` creates a tree with an extension node as root.

Schedule nodes can be copied and freed using the following functions.

```
#include <isl/schedule_node.h>
__isl_give isl_schedule_node *isl_schedule_node_copy(
        __isl_keep isl_schedule_node *node);
__isl_null isl_schedule_node *isl_schedule_node_free(
        __isl_take isl_schedule_node *node);
```

The following functions can be used to check if two schedule nodes point to the same position in the same schedule.

```
#include <isl/schedule_node.h>
isl_bool isl_schedule_node_is_equal(
        __isl_keep isl_schedule_node *node1,
        __isl_keep isl_schedule_node *node2);
```

The following properties can be obtained from a schedule node.

```
#include <isl/schedule_node.h>
enum isl_schedule_node_type isl_schedule_node_get_type(
        __isl_keep isl_schedule_node *node);
enum isl_schedule_node_type
isl_schedule_node_get_parent_type(
        __isl_keep isl_schedule_node *node);
__isl_give isl_schedule *isl_schedule_node_get_schedule(
        __isl_keep isl_schedule_node *node);
```

The function `isl_schedule_node_get_type` returns the type of the node, while `isl_schedule_node_get_parent_type` returns type of the parent of the node, which is required to exist. The function `isl_schedule_node_get_schedule` returns a copy to the schedule to which the node belongs.

The following functions can be used to move the schedule node to a different position in the tree or to check if such a position exists.

```
#include <isl/schedule_node.h>
isl_bool isl_schedule_node_has_parent(
        __isl_keep isl_schedule_node *node);
__isl_give isl_schedule_node *isl_schedule_node_parent(
        __isl_take isl_schedule_node *node);
__isl_give isl_schedule_node *isl_schedule_node_root(
        __isl_take isl_schedule_node *node);
__isl_give isl_schedule_node *isl_schedule_node_ancestor(
        __isl_take isl_schedule_node *node,
        int generation);
int isl_schedule_node_n_children(
        __isl_keep isl_schedule_node *node);
__isl_give isl_schedule_node *isl_schedule_node_child(
        __isl_take isl_schedule_node *node, int pos);
isl_bool isl_schedule_node_has_children(
        __isl_keep isl_schedule_node *node);
__isl_give isl_schedule_node *isl_schedule_node_first_child(
        __isl_take isl_schedule_node *node);
isl_bool isl_schedule_node_has_previous_sibling(
        __isl_keep isl_schedule_node *node);
__isl_give isl_schedule_node *
isl_schedule_node_previous_sibling(
        __isl_take isl_schedule_node *node);
isl_bool isl_schedule_node_has_next_sibling(
        __isl_keep isl_schedule_node *node);
__isl_give isl_schedule_node *
isl_schedule_node_next_sibling(
        __isl_take isl_schedule_node *node);
```

For isl_schedule_node_ancestor, the ancestor of generation 0 is the node itself, the ancestor of generation 1 is its parent and so on.

It is also possible to query the number of ancestors of a node, the position of the current node within the children of its parent, the position of the subtree containing a node within the children of an ancestor or to obtain a copy of a given child without destroying the current node. Given two nodes that point to the same schedule, their closest shared ancestor can be obtained using isl_schedule_node_get_shared_ancestor.

```
#include <isl/schedule_node.h>
int isl_schedule_node_get_tree_depth(
        __isl_keep isl_schedule_node *node);
int isl_schedule_node_get_child_position(
        __isl_keep isl_schedule_node *node);
int isl_schedule_node_get_ancestor_child_position(
        __isl_keep isl_schedule_node *node,
        __isl_keep isl_schedule_node *ancestor);
__isl_give isl_schedule_node *isl_schedule_node_get_child(
        __isl_keep isl_schedule_node *node, int pos);
__isl_give isl_schedule_node *
isl_schedule_node_get_shared_ancestor(
        __isl_keep isl_schedule_node *node1,
        __isl_keep isl_schedule_node *node2);
```

All nodes in a schedule tree or all descendants of a specific node (including the node) can be visited in depth-first pre-order using the following functions.

```
#include <isl/schedule.h>
isl_stat isl_schedule_foreach_schedule_node_top_down(
        __isl_keep isl_schedule *sched,
        isl_bool (*fn)(__isl_keep isl_schedule_node *node,
                void *user), void *user);

#include <isl/schedule_node.h>
isl_stat isl_schedule_node_foreach_descendant_top_down(
        __isl_keep isl_schedule_node *node,
        isl_bool (*fn)(__isl_keep isl_schedule_node *node,
                void *user), void *user);
```

The callback function is slightly different from the usual callbacks in that it not only indicates success (non-negative result) or failure (negative result), but also indicates whether the children of the given node should be visited. In particular, if the callback returns a positive value, then the children are visited, but if the callback returns zero, then the children are not visited.

The ancestors of a node in a schedule tree can be visited from the root down to and including the parent of the node using the following function.

```
#include <isl/schedule_node.h>
```

```
isl_stat isl_schedule_node_foreach_ancestor_top_down(
        __isl_keep isl_schedule_node *node,
        isl_stat (*fn)(__isl_keep isl_schedule_node *node,
                void *user), void *user);
```

The following functions allows for a depth-first post-order traversal of the nodes in a schedule tree or of the descendants of a specific node (including the node itself), where the user callback is allowed to modify the visited node.

```
#include <isl/schedule.h>
__isl_give isl_schedule *
isl_schedule_map_schedule_node_bottom_up(
        __isl_take isl_schedule *schedule,
        __isl_give isl_schedule_node *(*fn)(
                __isl_take isl_schedule_node *node,
                void *user), void *user);

#include <isl/schedule_node.h>
__isl_give isl_schedule_node *
isl_schedule_node_map_descendant_bottom_up(
        __isl_take isl_schedule_node *node,
        __isl_give isl_schedule_node *(*fn)(
                __isl_take isl_schedule_node *node,
                void *user), void *user);
```

The traversal continues from the node returned by the callback function. It is the responsibility of the user to ensure that this does not lead to an infinite loop. It is safest to always return a pointer to the same position (same ancestors and child positions) as the input node.

The following function removes a node (along with its descendants) from a schedule tree and returns a pointer to the leaf at the same position in the updated tree. It is not allowed to remove the root of a schedule tree or a child of a set or sequence node.

```
#include <isl/schedule_node.h>
__isl_give isl_schedule_node *isl_schedule_node_cut(
        __isl_take isl_schedule_node *node);
```

The following function removes a single node from a schedule tree and returns a pointer to the child of the node, now located at the position of the original node or to a leaf node at that position if there was no child. It is not allowed to remove the root of a schedule tree, a set or sequence node, a child of a set or sequence node or a band node with an anchored subtree.

```
#include <isl/schedule_node.h>
__isl_give isl_schedule_node *isl_schedule_node_delete(
        __isl_take isl_schedule_node *node);
```

Most nodes in a schedule tree only contain local information. In some cases, however, a node may also refer to outer band nodes. This means that the position of the node within the tree should not be changed, or at least that no changes are performed to the outer band nodes. The following function can be used to test whether the subtree rooted at a given node contains any such nodes.

```
#include <isl/schedule_node.h>
isl_bool isl_schedule_node_is_subtree_anchored(
        __isl_keep isl_schedule_node *node);
```

The following function resets the user pointers on all parameter and tuple identifiers referenced by the given schedule node.

```
#include <isl/schedule_node.h>
__isl_give isl_schedule_node *isl_schedule_node_reset_user(
        __isl_take isl_schedule_node *node);
```

The following function aligns the parameters of the given schedule node to the given space.

```
#include <isl/schedule_node.h>
__isl_give isl_schedule_node *
isl_schedule_node_align_params(
        __isl_take isl_schedule_node *node,
        __isl_take isl_space *space);
```

Several node types have their own functions for querying (and in some cases setting) some node type specific properties.

```
#include <isl/schedule_node.h>
__isl_give isl_space *isl_schedule_node_band_get_space(
        __isl_keep isl_schedule_node *node);
__isl_give isl_multi_union_pw_aff *
isl_schedule_node_band_get_partial_schedule(
        __isl_keep isl_schedule_node *node);
__isl_give isl_union_map *
isl_schedule_node_band_get_partial_schedule_union_map(
        __isl_keep isl_schedule_node *node);
unsigned isl_schedule_node_band_n_member(
        __isl_keep isl_schedule_node *node);
isl_bool isl_schedule_node_band_member_get_coincident(
        __isl_keep isl_schedule_node *node, int pos);
__isl_give isl_schedule_node *
isl_schedule_node_band_member_set_coincident(
        __isl_take isl_schedule_node *node, int pos,
        int coincident);
isl_bool isl_schedule_node_band_get_permutable(
```

```
        __isl_keep isl_schedule_node *node);
__isl_give isl_schedule_node *
isl_schedule_node_band_set_permutable(
        __isl_take isl_schedule_node *node, int permutable);
enum isl_ast_loop_type
isl_schedule_node_band_member_get_ast_loop_type(
        __isl_keep isl_schedule_node *node, int pos);
__isl_give isl_schedule_node *
isl_schedule_node_band_member_set_ast_loop_type(
        __isl_take isl_schedule_node *node, int pos,
        enum isl_ast_loop_type type);
__isl_give isl_union_set *
enum isl_ast_loop_type
isl_schedule_node_band_member_get_isolate_ast_loop_type(
        __isl_keep isl_schedule_node *node, int pos);
__isl_give isl_schedule_node *
isl_schedule_node_band_member_set_isolate_ast_loop_type(
        __isl_take isl_schedule_node *node, int pos,
        enum isl_ast_loop_type type);
isl_schedule_node_band_get_ast_build_options(
        __isl_keep isl_schedule_node *node);
__isl_give isl_schedule_node *
isl_schedule_node_band_set_ast_build_options(
        __isl_take isl_schedule_node *node,
        __isl_take isl_union_set *options);
```

The function `isl_schedule_node_band_get_space` returns the space of the partial schedule of the band. The function `isl_schedule_node_band_get_partial_schedule_union_map` returns a representation of the partial schedule of the band node in the form of an `isl_union_map`. The coincident and permutable properties are set by `isl_schedule_constraints_compute_schedu` on the schedule tree it produces. A scheduling dimension is considered to be "coincident" if it satisfies the coincidence constraints within its band. That is, if the dependence distances of the coincidence constraints are all zero in that direction (for fixed iterations of outer bands). A band is marked permutable if it was produced using the Pluto-like scheduler. Note that the scheduler may have to resort to a Feautrier style scheduling step even if the default scheduler is used. An `isl_ast_loop_type` is one of `isl_ast_loop_default`, `isl_ast_loop_atomic`, `isl_ast_loop_unroll` or `isl_ast_loop_separate`. For the meaning of these loop AST generation types and the difference between the regular loop AST generation type and the isolate loop AST generation type, see §1.5.4. The functions `isl_schedule_node_band_member_get_ast_loop_type` and `isl_schedule_node_band_member_get_isolate_ast_loop_type` may return `isl_ast_loop_error` if an error occurs. The AST build options govern how an AST is generated for the individual schedule dimensions during AST generation. See §1.5.4.

```
#include <isl/schedule_node.h>
__isl_give isl_set *
```

```
isl_schedule_node_context_get_context(
        __isl_keep isl_schedule_node *node);

#include <isl/schedule_node.h>
__isl_give isl_union_set *
isl_schedule_node_domain_get_domain(
        __isl_keep isl_schedule_node *node);

#include <isl/schedule_node.h>
__isl_give isl_union_map *
isl_schedule_node_expansion_get_expansion(
        __isl_keep isl_schedule_node *node);
__isl_give isl_union_pw_multi_aff *
isl_schedule_node_expansion_get_contraction(
        __isl_keep isl_schedule_node *node);

#include <isl/schedule_node.h>
__isl_give isl_union_map *
isl_schedule_node_extension_get_extension(
        __isl_keep isl_schedule_node *node);

#include <isl/schedule_node.h>
__isl_give isl_union_set *
isl_schedule_node_filter_get_filter(
        __isl_keep isl_schedule_node *node);

#include <isl/schedule_node.h>
__isl_give isl_set *isl_schedule_node_guard_get_guard(
        __isl_keep isl_schedule_node *node);

#include <isl/schedule_node.h>
__isl_give isl_id *isl_schedule_node_mark_get_id(
        __isl_keep isl_schedule_node *node);
```

The following functions can be used to obtain an `isl_multi_union_pw_aff`, an `isl_union_pw_multi_aff` or `isl_union_map` representation of partial schedules related to the node.

```
#include <isl/schedule_node.h>
__isl_give isl_multi_union_pw_aff *
isl_schedule_node_get_prefix_schedule_multi_union_pw_aff(
        __isl_keep isl_schedule_node *node);
__isl_give isl_union_pw_multi_aff *
isl_schedule_node_get_prefix_schedule_union_pw_multi_aff(
        __isl_keep isl_schedule_node *node);
__isl_give isl_union_map *
isl_schedule_node_get_prefix_schedule_union_map(
```

```
        __isl_keep isl_schedule_node *node);
__isl_give isl_union_map *
isl_schedule_node_get_prefix_schedule_relation(
        __isl_keep isl_schedule_node *node);
__isl_give isl_union_map *
isl_schedule_node_get_subtree_schedule_union_map(
        __isl_keep isl_schedule_node *node);
```

In particular, the functions `isl_schedule_node_get_prefix_schedule_multi_union_pw_aff`, `isl_schedule_node_get_prefix_schedule_union_pw_multi_aff` and `isl_schedule_node_get_prefix_sch` return a relative ordering on the domain elements that reach the given node determined by its ancestors. The function `isl_schedule_node_get_prefix_schedule_relation` additionally includes the domain constraints in the result. The function `isl_schedule_node_get_subtree_schedule` returns a representation of the partial schedule defined by the subtree rooted at the given node. If the tree contains any expansion nodes, then the subtree schedule is formulated in terms of the expanded domain elements. The tree passed to functions returning a prefix schedule may only contain extension nodes if these would not affect the result of these functions. That is, if one of the ancestors is an extension node, then all of the domain elements that were added by the extension node need to have been filtered out by filter nodes between the extension node and the input node. The tree passed to `isl_schedule_node_get_subtree_schedule_union_map` may not contain in extension nodes in the selected subtree.

The expansion/contraction defined by an entire subtree, combining the expansions/-contractions on the expansion nodes in the subtree, can be obtained using the following functions.

```
#include <isl/schedule_node.h>
__isl_give isl_union_map *
isl_schedule_node_get_subtree_expansion(
        __isl_keep isl_schedule_node *node);
__isl_give isl_union_pw_multi_aff *
isl_schedule_node_get_subtree_contraction(
        __isl_keep isl_schedule_node *node);
```

The total number of outer band members of given node, i.e., the shared output dimension of the maps in the result of `isl_schedule_node_get_prefix_schedule_union_map` can be obtained using the following function.

```
#include <isl/schedule_node.h>
int isl_schedule_node_get_schedule_depth(
        __isl_keep isl_schedule_node *node);
```

The following functions return the elements that reach the given node or the union of universes in the spaces that contain these elements.

```
#include <isl/schedule_node.h>
__isl_give isl_union_set *
```

```
isl_schedule_node_get_domain(
        __isl_keep isl_schedule_node *node);
__isl_give isl_union_set *
isl_schedule_node_get_universe_domain(
        __isl_keep isl_schedule_node *node);
```

The input tree of `isl_schedule_node_get_domain` may only contain extension nodes if these would not affect the result of this function. That is, if one of the ancestors is an extension node, then all of the domain elements that were added by the extension node need to have been filtered out by filter nodes between the extension node and the input node.

The following functions can be used to introduce additional nodes in the schedule tree. The new node is introduced at the point in the tree where the `isl_schedule_node` points to and the results points to the new node.

```
#include <isl/schedule_node.h>
__isl_give isl_schedule_node *
isl_schedule_node_insert_partial_schedule(
        __isl_take isl_schedule_node *node,
        __isl_take isl_multi_union_pw_aff *schedule);
```

This function inserts a new band node with (the greatest integer part of) the given partial schedule. The subtree rooted at the given node is assumed not to have any anchored nodes.

```
#include <isl/schedule_node.h>
__isl_give isl_schedule_node *
isl_schedule_node_insert_context(
        __isl_take isl_schedule_node *node,
        __isl_take isl_set *context);
```

This function inserts a new context node with the given context constraints.

```
#include <isl/schedule_node.h>
__isl_give isl_schedule_node *
isl_schedule_node_insert_filter(
        __isl_take isl_schedule_node *node,
        __isl_take isl_union_set *filter);
```

This function inserts a new filter node with the given filter. If the original node already pointed to a filter node, then the two filter nodes are merged into one.

```
#include <isl/schedule_node.h>
__isl_give isl_schedule_node *
isl_schedule_node_insert_guard(
        __isl_take isl_schedule_node *node,
        __isl_take isl_set *guard);
```

This function inserts a new guard node with the given guard constraints.

```
#include <isl/schedule_node.h>
__isl_give isl_schedule_node *
isl_schedule_node_insert_mark(
        __isl_take isl_schedule_node *node,
        __isl_take isl_id *mark);
```

This function inserts a new mark node with the give mark identifier.

```
#include <isl/schedule_node.h>
__isl_give isl_schedule_node *
isl_schedule_node_insert_sequence(
        __isl_take isl_schedule_node *node,
        __isl_take isl_union_set_list *filters);
__isl_give isl_schedule_node *
isl_schedule_node_insert_set(
        __isl_take isl_schedule_node *node,
        __isl_take isl_union_set_list *filters);
```

These functions insert a new sequence or set node with the given filters as children.

```
#include <isl/schedule_node.h>
__isl_give isl_schedule_node *isl_schedule_node_group(
        __isl_take isl_schedule_node *node,
        __isl_take isl_id *group_id);
```

This function introduces an expansion node in between the current node and its parent that expands instances of a space with tuple identifier `group_id` to the original domain elements that reach the node. The group instances are identified by the prefix schedule of those domain elements. The ancestors of the node are adjusted to refer to the group instances instead of the original domain elements. The return value points to the same node in the updated schedule tree as the input node, i.e., to the child of the newly introduced expansion node. Grouping instances of different statements ensures that they will be treated as a single statement by the AST generator up to the point of the expansion node.

The following function can be used to flatten a nested sequence.

```
#include <isl/schedule_node.h>
__isl_give isl_schedule_node *
isl_schedule_node_sequence_splice_child(
        __isl_take isl_schedule_node *node, int pos);
```

That is, given a sequence node `node` that has another sequence node in its child at position `pos` (in particular, the child of that filter node is a sequence node), attach the children of that other sequence node as children of `node`, replacing the original child at position `pos`.

The partial schedule of a band node can be scaled (down) or reduced using the following functions.

```
#include <isl/schedule_node.h>
__isl_give isl_schedule_node *
isl_schedule_node_band_scale(
        __isl_take isl_schedule_node *node,
        __isl_take isl_multi_val *mv);
__isl_give isl_schedule_node *
isl_schedule_node_band_scale_down(
        __isl_take isl_schedule_node *node,
        __isl_take isl_multi_val *mv);
__isl_give isl_schedule_node *
isl_schedule_node_band_mod(
        __isl_take isl_schedule_node *node,
        __isl_take isl_multi_val *mv);
```

The spaces of the two arguments need to match. After scaling, the partial schedule is replaced by its greatest integer part to ensure that the schedule remains integral.

The partial schedule of a band node can be shifted by an `isl_multi_union_pw_aff` with a domain that is a superset of the domain of the partial schedule using the following function.

```
#include <isl/schedule_node.h>
__isl_give isl_schedule_node *
isl_schedule_node_band_shift(
        __isl_take isl_schedule_node *node,
        __isl_take isl_multi_union_pw_aff *shift);
```

A band node can be tiled using the following function.

```
#include <isl/schedule_node.h>
__isl_give isl_schedule_node *isl_schedule_node_band_tile(
        __isl_take isl_schedule_node *node,
        __isl_take isl_multi_val *sizes);

isl_stat isl_options_set_tile_scale_tile_loops(isl_ctx *ctx,
        int val);
int isl_options_get_tile_scale_tile_loops(isl_ctx *ctx);
isl_stat isl_options_set_tile_shift_point_loops(isl_ctx *ctx,
        int val);
int isl_options_get_tile_shift_point_loops(isl_ctx *ctx);
```

The `isl_schedule_node_band_tile` function tiles the band using the given tile sizes inside its schedule. A new child band node is created to represent the point loops and it is inserted between the modified band and its children. The subtree rooted at the given node is assumed not to have any anchored nodes. The `tile_scale_tile_loops` option specifies whether the tile loops iterators should be scaled by the tile sizes. If the `tile_shift_point_loops` option is set, then the point loops are shifted to start at zero.

A band node can be split into two nested band nodes using the following function.

```
#include <isl/schedule_node.h>
__isl_give isl_schedule_node *isl_schedule_node_band_split(
        __isl_take isl_schedule_node *node, int pos);
```

The resulting outer band node contains the first `pos` dimensions of the schedule of `node` while the inner band contains the remaining dimensions. The schedules of the two band nodes live in anonymous spaces.

A band node can be moved down to the leaves of the subtree rooted at the band node using the following function.

```
#include <isl/schedule_node.h>
__isl_give isl_schedule_node *isl_schedule_node_band_sink(
        __isl_take isl_schedule_node *node);
```

The subtree rooted at the given node is assumed not to have any anchored nodes. The result points to the node in the resulting tree that is in the same position as the node pointed to by `node` in the original tree.

```
#include <isl/schedule_node.h>
__isl_give isl_schedule_node *
isl_schedule_node_order_before(
        __isl_take isl_schedule_node *node,
        __isl_take isl_union_set *filter);
__isl_give isl_schedule_node *
isl_schedule_node_order_after(
        __isl_take isl_schedule_node *node,
        __isl_take isl_union_set *filter);
```

These functions split the domain elements that reach `node` into those that satisfy `filter` and those that do not and arranges for the elements that do satisfy the filter to be executed before (in case of `isl_schedule_node_order_before`) or after (in case of `isl_schedule_node_order_after`) those that do not. The order is imposed by a sequence node, possibly reusing the grandparent of `node` on two copies of the subtree attached to the original `node`. Both copies are simplified with respect to their filter.

Return a pointer to the copy of the subtree that does not satisfy `filter`. If there is no such copy (because all reaching domain elements satisfy the filter), then return the original pointer.

```
#include <isl/schedule_node.h>
__isl_give isl_schedule_node *
isl_schedule_node_graft_before(
        __isl_take isl_schedule_node *node,
        __isl_take isl_schedule_node *graft);
__isl_give isl_schedule_node *
isl_schedule_node_graft_after(
        __isl_take isl_schedule_node *node,
        __isl_take isl_schedule_node *graft);
```

This function inserts the graft tree into the tree containing node such that it is executed before (in case of isl_schedule_node_graft_before) or after (in case of isl_schedule_node_graft_after) node. The root node of graft should be an extension node where the domain of the extension is the flat product of all outer band nodes of node. The root node may also be a domain node. The elements of the domain or the range of the extension may not intersect with the domain elements that reach "node". The schedule tree of graft may not be anchored.

The schedule tree of node is modified to include an extension node corresponding to the root node of graft as a child of the original parent of node. The original node that node points to and the child of the root node of graft are attached to this extension node through a sequence, with appropriate filters and with the child of graft appearing before or after the original node.

If node already appears inside a sequence that is the child of an extension node and if the spaces of the new domain elements do not overlap with those of the original domain elements, then that extension node is extended with the new extension rather than introducing a new segment of extension and sequence nodes.

Return a pointer to the same node in the modified tree that node pointed to in the original tree.

A representation of the schedule node can be printed using

```
#include <isl/schedule_node.h>
__isl_give isl_printer *isl_printer_print_schedule_node(
        __isl_take isl_printer *p,
        __isl_keep isl_schedule_node *node);
__isl_give char *isl_schedule_node_to_str(
        __isl_keep isl_schedule_node *node);
```

isl_schedule_node_to_str prints the schedule node in block format.

1.5.2 Dependence Analysis

isl contains specialized functionality for performing array dataflow analysis. That is, given a *sink* access relation and a collection of possible *source* access relations, isl can compute relations that describe for each iteration of the sink access, which iteration of which of the source access relations was the last to access the same data element before the given iteration of the sink access. The resulting dependence relations map source iterations to either the corresponding sink iterations or pairs of corresponding sink iterations and accessed data elements. To compute standard flow dependences, the sink should be a read, while the sources should be writes. If any of the source accesses are marked as being *may* accesses, then there will be a dependence from the last *must* access **and** from any *may* access that follows this last *must* access. In particular, if *all* sources are *may* accesses, then memory based dependence analysis is performed. If, on the other hand, all sources are *must* accesses, then value based dependence analysis is performed.

High-level Interface

A high-level interface to dependence analysis is provided by the following function.

```
#include <isl/flow.h>
__isl_give isl_union_flow *
isl_union_access_info_compute_flow(
        __isl_take isl_union_access_info *access);
```

The input `isl_union_access_info` object describes the sink access relations, the source access relations and a schedule, while the output `isl_union_flow` object describes the resulting dependence relations and the subsets of the sink relations for which no source was found.

An `isl_union_access_info` is created, modified, copied and freed using the following functions.

```
#include <isl/flow.h>
__isl_give isl_union_access_info *
isl_union_access_info_from_sink(
        __isl_take isl_union_map *sink);
__isl_give isl_union_access_info *
isl_union_access_info_set_must_source(
        __isl_take isl_union_access_info *access,
        __isl_take isl_union_map *must_source);
__isl_give isl_union_access_info *
isl_union_access_info_set_may_source(
        __isl_take isl_union_access_info *access,
        __isl_take isl_union_map *may_source);
__isl_give isl_union_access_info *
isl_union_access_info_set_schedule(
        __isl_take isl_union_access_info *access,
        __isl_take isl_schedule *schedule);
__isl_give isl_union_access_info *
isl_union_access_info_set_schedule_map(
        __isl_take isl_union_access_info *access,
        __isl_take isl_union_map *schedule_map);
__isl_give isl_union_access_info *
isl_union_access_info_copy(
        __isl_keep isl_union_access_info *access);
__isl_null isl_union_access_info *
isl_union_access_info_free(
        __isl_take isl_union_access_info *access);
```

The may sources set by `isl_union_access_info_set_may_source` do not need to include the must sources set by `isl_union_access_info_set_must_source` as a subset. The user is free not to call one (or both) of these functions, in which case the corresponding set is kept to its empty default. Similarly, the default schedule initialized by `isl_union_access_info_from_sink` is empty. The current schedule

is determined by the last call to either isl_union_access_info_set_schedule or isl_union_access_info_set_schedule_map. The domain of the schedule corresponds to the domains of the access relations. In particular, the domains of the access relations are effectively intersected with the domain of the schedule and only the resulting accesses are considered by the dependence analysis.

A representation of the information contained in an object of type isl_union_access_info can be obtained using

```
#include <isl/flow.h>
__isl_give isl_printer *
isl_printer_print_union_access_info(
        __isl_take isl_printer *p,
        __isl_keep isl_union_access_info *access);
__isl_give char *isl_union_access_info_to_str(
        __isl_keep isl_union_access_info *access);
```

isl_union_access_info_to_str prints the information in flow format.

The output of isl_union_access_info_compute_flow can be examined and freed using the following functions.

```
#include <isl/flow.h>
__isl_give isl_union_map *isl_union_flow_get_must_dependence(
        __isl_keep isl_union_flow *flow);
__isl_give isl_union_map *isl_union_flow_get_may_dependence(
        __isl_keep isl_union_flow *flow);
__isl_give isl_union_map *
isl_union_flow_get_full_must_dependence(
        __isl_keep isl_union_flow *flow);
__isl_give isl_union_map *
isl_union_flow_get_full_may_dependence(
        __isl_keep isl_union_flow *flow);
__isl_give isl_union_map *isl_union_flow_get_must_no_source(
        __isl_keep isl_union_flow *flow);
__isl_give isl_union_map *isl_union_flow_get_may_no_source(
        __isl_keep isl_union_flow *flow);
__isl_null isl_union_flow *isl_union_flow_free(
        __isl_take isl_union_flow *flow);
```

The relation returned by isl_union_flow_get_must_dependence relates domain elements of must sources to domain elements of the sink. The relation returned by isl_union_flow_get_may_dependence relates domain elements of must or may sources to domain elements of the sink and includes the previous relation as a subset. The relation returned by isl_union_flow_get_full_must_dependence relates domain elements of must sources to pairs of domain elements of the sink and accessed data elements. The relation returned by isl_union_flow_get_full_may_dependence relates domain elements of must or may sources to pairs of domain elements of the

sink and accessed data elements. This relation includes the previous relation as a subset. The relation returned by isl_union_flow_get_must_no_source is the subset of the sink relation for which no dependences have been found. The relation returned by isl_union_flow_get_may_no_source is the subset of the sink relation for which no definite dependences have been found. That is, it contains those sink access that do not contribute to any of the elements in the relation returned by isl_union_flow_get_must_dependence.

A representation of the information contained in an object of type isl_union_flow can be obtained using

```
#include <isl/flow.h>
__isl_give isl_printer *isl_printer_print_union_flow(
        __isl_take isl_printer *p,
        __isl_keep isl_union_flow *flow);
__isl_give char *isl_union_flow_to_str(
        __isl_keep isl_union_flow *flow);
```

isl_union_flow_to_str prints the information in flow format.

Low-level Interface

A lower-level interface is provided by the following functions.

```
#include <isl/flow.h>

typedef int (*isl_access_level_before)(void *first, void *second);

__isl_give isl_access_info *isl_access_info_alloc(
        __isl_take isl_map *sink,
        void *sink_user, isl_access_level_before fn,
        int max_source);
__isl_give isl_access_info *isl_access_info_add_source(
        __isl_take isl_access_info *acc,
        __isl_take isl_map *source, int must,
        void *source_user);
__isl_null isl_access_info *isl_access_info_free(
        __isl_take isl_access_info *acc);

__isl_give isl_flow *isl_access_info_compute_flow(
        __isl_take isl_access_info *acc);

isl_stat isl_flow_foreach(__isl_keep isl_flow *deps,
        isl_stat (*fn)(__isl_take isl_map *dep, int must,
                  void *dep_user, void *user),
        void *user);
__isl_give isl_map *isl_flow_get_no_source(
        __isl_keep isl_flow *deps, int must);
void isl_flow_free(__isl_take isl_flow *deps);
```

The function `isl_access_info_compute_flow` performs the actual dependence analysis. The other functions are used to construct the input for this function or to read off the output.

The input is collected in an `isl_access_info`, which can be created through a call to `isl_access_info_alloc`. The arguments to this functions are the sink access relation `sink`, a token `sink_user` used to identify the sink access to the user, a callback function for specifying the relative order of source and sink accesses, and the number of source access relations that will be added. The callback function has type `int (*)(void *first, void *second)`. The function is called with two user supplied tokens identifying either a source or the sink and it should return the shared nesting level and the relative order of the two accesses. In particular, let *n* be the number of loops shared by the two accesses. If `first` precedes `second` textually, then the function should return *2 * n + 1*; otherwise, it should return *2 * n*. The sources can be added to the `isl_access_info` by performing (at most) `max_source` calls to `isl_access_info_add_source`. `must` indicates whether the source is a *must* access or a *may* access. Note that a multi-valued access relation should only be marked *must* if every iteration in the domain of the relation accesses *all* elements in its image. The `source_user` token is again used to identify the source access. The range of the source access relation `source` should have the same dimension as the range of the sink access relation. The `isl_access_info_free` function should usually not be called explicitly, because it is called implicitly by `isl_access_info_compute_flow`.

The result of the dependence analysis is collected in an `isl_flow`. There may be elements of the sink access for which no preceding source access could be found or for which all preceding sources are *may* accesses. The relations containing these elements can be obtained through calls to `isl_flow_get_no_source`, the first with `must` set and the second with `must` unset. In the case of standard flow dependence analysis, with the sink a read and the sources *must* writes, the first relation corresponds to the reads from uninitialized array elements and the second relation is empty. The actual flow dependences can be extracted using `isl_flow_foreach`. This function will call the user-specified callback function `fn` for each **non-empty** dependence between a source and the sink. The callback function is called with four arguments, the actual flow dependence relation mapping source iterations to sink iterations, a boolean that indicates whether it is a *must* or *may* dependence, a token identifying the source and an additional `void *` with value equal to the third argument of the `isl_flow_foreach` call. A dependence is marked *must* if it originates from a *must* source and if it is not followed by any *may* sources.

After finishing with an `isl_flow`, the user should call `isl_flow_free` to free all associated memory.

Interaction with the Low-level Interface

During the dependence analysis, we frequently need to perform the following operation. Given a relation between sink iterations and potential source iterations from a particular source domain, what is the last potential source iteration corresponding to each sink iteration. It can sometimes be convenient to adjust the set of potential source iterations before or after each such operation. The prototypical example is fuzzy array

dataflow analysis, where we need to analyze if, based on data-dependent constraints, the sink iteration can ever be executed without one or more of the corresponding potential source iterations being executed. If so, we can introduce extra parameters and select an unknown but fixed source iteration from the potential source iterations. To be able to perform such manipulations, `isl` provides the following function.

```
#include <isl/flow.h>

typedef __isl_give isl_restriction *(*isl_access_restrict)(
        __isl_keep isl_map *source_map,
        __isl_keep isl_set *sink, void *source_user,
        void *user);
__isl_give isl_access_info *isl_access_info_set_restrict(
        __isl_take isl_access_info *acc,
        isl_access_restrict fn, void *user);
```

The function `isl_access_info_set_restrict` should be called before calling `isl_access_info_compute_flow` and registers a callback function that will be called any time `isl` is about to compute the last potential source. The first argument is the (reverse) proto-dependence, mapping sink iterations to potential source iterations. The second argument represents the sink iterations for which we want to compute the last source iteration. The third argument is the token corresponding to the source and the final argument is the token passed to `isl_access_info_set_restrict`. The callback is expected to return a restriction on either the input or the output of the operation computing the last potential source. If the input needs to be restricted then restrictions are needed for both the source and the sink iterations. The sink iterations and the potential source iterations will be intersected with these sets. If the output needs to be restricted then only a restriction on the source iterations is required. If any error occurs, the callback should return `NULL`. An `isl_restriction` object can be created, freed and inspected using the following functions.

```
#include <isl/flow.h>

__isl_give isl_restriction *isl_restriction_input(
        __isl_take isl_set *source_restr,
        __isl_take isl_set *sink_restr);
__isl_give isl_restriction *isl_restriction_output(
        __isl_take isl_set *source_restr);
__isl_give isl_restriction *isl_restriction_none(
        __isl_take isl_map *source_map);
__isl_give isl_restriction *isl_restriction_empty(
        __isl_take isl_map *source_map);
__isl_null isl_restriction *isl_restriction_free(
        __isl_take isl_restriction *restr);
```

`isl_restriction_none` and `isl_restriction_empty` are special cases of `isl_restriction_input`. `isl_restriction_none` is essentially equivalent to

```
isl_restriction_input(isl_set_universe(
    isl_space_range(isl_map_get_space(source_map))),
                    isl_set_universe(
    isl_space_domain(isl_map_get_space(source_map))));
```

whereas `isl_restriction_empty` is essentially equivalent to

```
isl_restriction_input(isl_set_empty(
    isl_space_range(isl_map_get_space(source_map))),
                    isl_set_universe(
    isl_space_domain(isl_map_get_space(source_map))));
```

1.5.3 Scheduling

```
#include <isl/schedule.h>
__isl_give isl_schedule *
isl_schedule_constraints_compute_schedule(
        __isl_take isl_schedule_constraints *sc);
```

The function `isl_schedule_constraints_compute_schedule` can be used to compute a schedule that satisfies the given schedule constraints. These schedule constraints include the iteration domain for which a schedule should be computed and dependences between pairs of iterations. In particular, these dependences include *validity* dependences and *proximity* dependences. By default, the algorithm used to construct the schedule is similar to that of `Pluto`. Alternatively, Feautrier's multi-dimensional scheduling algorithm can be selected. The generated schedule respects all validity dependences. That is, all dependence distances over these dependences in the scheduled space are lexicographically positive.

The default algorithm tries to ensure that the dependence distances over coincidence constraints are zero and to minimize the dependence distances over proximity dependences. Moreover, it tries to obtain sequences (bands) of schedule dimensions for groups of domains where the dependence distances over validity dependences have only non-negative values. Note that when minimizing the maximal dependence distance over proximity dependences, a single affine expression in the parameters is constructed that bounds all dependence distances. If no such expression exists, then the algorithm will fail and resort to an alternative scheduling algorithm. In particular, this means that adding proximity dependences may eliminate valid solutions. A typical example where this phenomenon may occur is when some subset of the proximity dependences has no restriction on some parameter, forcing the coefficient of that parameter to be zero, while some other subset forces the dependence distance to depend on that parameter, requiring the same coefficient to be non-zero. When using Feautrier's algorithm, the coincidence and proximity constraints are only taken into account during the extension to a full-dimensional schedule.

An `isl_schedule_constraints` object can be constructed and manipulated using the following functions.

```
#include <isl/schedule.h>
```

```
__isl_give isl_schedule_constraints *
isl_schedule_constraints_copy(
        __isl_keep isl_schedule_constraints *sc);
__isl_give isl_schedule_constraints *
isl_schedule_constraints_on_domain(
        __isl_take isl_union_set *domain);
__isl_give isl_schedule_constraints *
isl_schedule_constraints_set_context(
        __isl_take isl_schedule_constraints *sc,
        __isl_take isl_set *context);
__isl_give isl_schedule_constraints *
isl_schedule_constraints_set_validity(
        __isl_take isl_schedule_constraints *sc,
        __isl_take isl_union_map *validity);
__isl_give isl_schedule_constraints *
isl_schedule_constraints_set_coincidence(
        __isl_take isl_schedule_constraints *sc,
        __isl_take isl_union_map *coincidence);
__isl_give isl_schedule_constraints *
isl_schedule_constraints_set_proximity(
        __isl_take isl_schedule_constraints *sc,
        __isl_take isl_union_map *proximity);
__isl_give isl_schedule_constraints *
isl_schedule_constraints_set_conditional_validity(
        __isl_take isl_schedule_constraints *sc,
        __isl_take isl_union_map *condition,
        __isl_take isl_union_map *validity);
__isl_give isl_schedule_constraints *
isl_schedule_constraints_apply(
        __isl_take isl_schedule_constraints *sc,
        __isl_take isl_union_map *umap);
__isl_null isl_schedule_constraints *
isl_schedule_constraints_free(
        __isl_take isl_schedule_constraints *sc);
```

The initial `isl_schedule_constraints` object created by `isl_schedule_constraints_on_domain` does not impose any constraints. That is, it has an empty set of dependences. The function `isl_schedule_constraints_set_context` allows the user to specify additional constraints on the parameters that may be assumed to hold during the construction of the schedule. The function `isl_schedule_constraints_set_validity` replaces the validity dependences, mapping domain elements i to domain elements that should be scheduled after i. The function `isl_schedule_constraints_set_coincidence` replaces the coincidence dependences, mapping domain elements i to domain elements that should be scheduled together with I, if possible. The function `isl_schedule_constraints_set_proximity` replaces the proximity dependences, mapping domain elements i to domain elements that should be scheduled either before I or as early as possible after i.

The function isl_schedule_constraints_set_conditional_validity replaces the conditional validity constraints. A conditional validity constraint is only imposed when any of the corresponding conditions is satisfied, i.e., when any of them is non-zero. That is, the scheduler ensures that within each band if the dependence distances over the condition constraints are not all zero then all corresponding conditional validity constraints are respected. A conditional validity constraint corresponds to a condition if the two are adjacent, i.e., if the domain of one relation intersect the range of the other relation. The typical use case of conditional validity constraints is to allow order constraints between live ranges to be violated as long as the live ranges themselves are local to the band. To allow more fine-grained control over which conditions correspond to which conditional validity constraints, the domains and ranges of these relations may include *tags*. That is, the domains and ranges of those relation may themselves be wrapped relations where the iteration domain appears in the domain of those wrapped relations and the range of the wrapped relations can be arbitrarily chosen by the user. Conditions and conditional validity constraints are only considered adjacent to each other if the entire wrapped relation matches. In particular, a relation with a tag will never be considered adjacent to a relation without a tag.

The function isl_schedule_constraints_compute_schedule takes schedule constraints that are defined on some set of domain elements and transforms them to schedule constraints on the elements to which these domain elements are mapped by the given transformation.

An isl_schedule_constraints object can be inspected using the following functions.

```
#include <isl/schedule.h>
__isl_give isl_union_set *
isl_schedule_constraints_get_domain(
        __isl_keep isl_schedule_constraints *sc);
__isl_give isl_union_map *
isl_schedule_constraints_get_validity(
        __isl_keep isl_schedule_constraints *sc);
__isl_give isl_union_map *
isl_schedule_constraints_get_coincidence(
        __isl_keep isl_schedule_constraints *sc);
__isl_give isl_union_map *
isl_schedule_constraints_get_proximity(
        __isl_keep isl_schedule_constraints *sc);
__isl_give isl_union_map *
isl_schedule_constraints_get_conditional_validity(
        __isl_keep isl_schedule_constraints *sc);
__isl_give isl_union_map *
isl_schedule_constraints_get_conditional_validity_condition(
        __isl_keep isl_schedule_constraints *sc);
```

The following function computes a schedule directly from an iteration domain and validity and proximity dependences and is implemented in terms of the functions described above. The use of isl_union_set_compute_schedule is discouraged.

```
#include <isl/schedule.h>
__isl_give isl_schedule *isl_union_set_compute_schedule(
        __isl_take isl_union_set *domain,
        __isl_take isl_union_map *validity,
        __isl_take isl_union_map *proximity);
```

The generated schedule represents a schedule tree. For more information on schedule trees, see §1.5.1.

Options

```
#include <isl/schedule.h>
isl_stat isl_options_set_schedule_max_coefficient(
        isl_ctx *ctx, int val);
int isl_options_get_schedule_max_coefficient(
        isl_ctx *ctx);
isl_stat isl_options_set_schedule_max_constant_term(
        isl_ctx *ctx, int val);
int isl_options_get_schedule_max_constant_term(
        isl_ctx *ctx);
isl_stat isl_options_set_schedule_serialize_sccs(
        isl_ctx *ctx, int val);
int isl_options_get_schedule_serialize_sccs(isl_ctx *ctx);
isl_stat isl_options_set_schedule_whole_component(
        isl_ctx *ctx, int val);
int isl_options_get_schedule_whole_component(
        isl_ctx *ctx);
isl_stat isl_options_set_schedule_maximize_band_depth(
        isl_ctx *ctx, int val);
int isl_options_get_schedule_maximize_band_depth(
        isl_ctx *ctx);
isl_stat isl_options_set_schedule_maximize_coincidence(
        isl_ctx *ctx, int val);
int isl_options_get_schedule_maximize_coincidence(
        isl_ctx *ctx);
isl_stat isl_options_set_schedule_outer_coincidence(
        isl_ctx *ctx, int val);
int isl_options_get_schedule_outer_coincidence(
        isl_ctx *ctx);
isl_stat isl_options_set_schedule_split_scaled(
        isl_ctx *ctx, int val);
int isl_options_get_schedule_split_scaled(
        isl_ctx *ctx);
isl_stat isl_options_set_schedule_treat_coalescing(
        isl_ctx *ctx, int val);
int isl_options_get_schedule_treat_coalescing(
```

```
        isl_ctx *ctx);
isl_stat isl_options_set_schedule_algorithm(
        isl_ctx *ctx, int val);
int isl_options_get_schedule_algorithm(
        isl_ctx *ctx);
isl_stat isl_options_set_schedule_separate_components(
        isl_ctx *ctx, int val);
int isl_options_get_schedule_separate_components(
        isl_ctx *ctx);
```

- schedule_max_coefficient

  This option enforces that the coefficients for variable and parameter dimensions in the calculated schedule are not larger than the specified value. This option can significantly increase the speed of the scheduling calculation and may also prevent fusing of unrelated dimensions. A value of -1 means that this option does not introduce bounds on the variable or parameter coefficients.

- schedule_max_constant_term

  This option enforces that the constant coefficients in the calculated schedule are not larger than the maximal constant term. This option can significantly increase the speed of the scheduling calculation and may also prevent fusing of unrelated dimensions. A value of -1 means that this option does not introduce bounds on the constant coefficients.

- schedule_serialize_sccs

  If this option is set, then all strongly connected components in the dependence graph are serialized as soon as they are detected. This means in particular that instances of statements will only appear in the same band node if these statements belong to the same strongly connected component at the point where the band node is constructed.

- schedule_whole_component

  If this option is set, then entire (weakly) connected components in the dependence graph are scheduled together as a whole. Otherwise, each strongly connected component within such a weakly connected component is first scheduled separately and then combined with other strongly connected components. This option has no effect if `schedule_serialize_sccs` is set.

- schedule_maximize_band_depth

  If this option is set, then the scheduler tries to maximize the width of the bands. Wider bands give more possibilities for tiling. In particular, if the `schedule_whole_component` option is set, then bands are split if this might result in wider bands. Otherwise, the effect of this option is to only allow strongly connected components to be combined if this does not reduce the width of the bands. Note that if the `schedule_serialize_sccs` options is set, then the `schedule_maximize_band_depth` option therefore has no effect.

- schedule_maximize_coincidence

  This option is only effective if the `schedule_whole_component` option is turned off. If the `schedule_maximize_coincidence` option is set, then (clusters of) strongly connected components are only combined with each other if this does not reduce the number of coincident band members.

- schedule_outer_coincidence

  If this option is set, then we try to construct schedules where the outermost scheduling dimension in each band satisfies the coincidence constraints.

- schedule_algorithm

  Selects the scheduling algorithm to be used. Available scheduling algorithms are `ISL_SCHEDULE_ALGORITHM_ISL` and `ISL_SCHEDULE_ALGORITHM_FEAUTRIER`.

- schedule_split_scaled

  If this option is set, then we try to construct schedules in which the constant term is split off from the linear part if the linear parts of the scheduling rows for all nodes in the graphs have a common non-trivial divisor. The constant term is then placed in a separate band and the linear part is reduced. This option is only effective when the Feautrier style scheduler is being used, either as the main scheduler or as a fallback for the Pluto-like scheduler.

- schedule_treat_coalescing

  If this option is set, then the scheduler will try and avoid producing schedules that perform loop coalescing. In particular, for the Pluto-like scheduler, this option places bounds on the schedule coefficients based on the sizes of the instance sets. For the Feautrier style scheduler, this option detects potentially coalescing schedules and then tries to adjust the schedule to avoid the coalescing.

- schedule_separate_components

  If this option is set then the function `isl_schedule_get_map` will treat set nodes in the same way as sequence nodes.

1.5.4 AST Generation

This section describes the `isl` functionality for generating ASTs that visit all the elements in a domain in an order specified by a schedule tree or a schedule map. In case the schedule given as a `isl_union_map`, an AST is generated that visits all the elements in the domain of the `isl_union_map` according to the lexicographic order of the corresponding image element(s). If the range of the `isl_union_map` consists of elements in more than one space, then each of these spaces is handled separately in an arbitrary order. It should be noted that the schedule tree or the image elements in a schedule map only specify the *order* in which the corresponding domain elements should be visited. No direct relation between the partial schedule values or the image elements on the one hand and the loop iterators in the generated AST on the other hand should be assumed.

Each AST is generated within a build. The initial build simply specifies the constraints on the parameters (if any) and can be created, inspected, copied and freed using the following functions.

```
#include <isl/ast_build.h>
__isl_give isl_ast_build *isl_ast_build_alloc(
        isl_ctx *ctx);
__isl_give isl_ast_build *isl_ast_build_from_context(
        __isl_take isl_set *set);
__isl_give isl_ast_build *isl_ast_build_copy(
        __isl_keep isl_ast_build *build);
__isl_null isl_ast_build *isl_ast_build_free(
        __isl_take isl_ast_build *build);
```

The `set` argument is usually a parameter set with zero or more parameters. In fact, when creating an AST using `isl_ast_build_node_from_schedule`, this set is required to be a parameter set. An `isl_ast_build` created using `isl_ast_build_alloc` does not specify any parameter constraints. More `isl_ast_build` functions are described in §1.5.4 and §1.5.4. Finally, the AST itself can be constructed using one of the following functions.

```
#include <isl/ast_build.h>
__isl_give isl_ast_node *isl_ast_build_node_from_schedule(
        __isl_keep isl_ast_build *build,
        __isl_take isl_schedule *schedule);
__isl_give isl_ast_node *
isl_ast_build_node_from_schedule_map(
        __isl_keep isl_ast_build *build,
        __isl_take isl_union_map *schedule);
```

Inspecting the AST

The basic properties of an AST node can be obtained as follows.

```
#include <isl/ast.h>
enum isl_ast_node_type isl_ast_node_get_type(
        __isl_keep isl_ast_node *node);
```

The type of an AST node is one of `isl_ast_node_for`, `isl_ast_node_if`, `isl_ast_node_block`, `isl_ast_node_mark` or `isl_ast_node_user`. An `isl_ast_node_for` represents a for node. An `isl_ast_node_if` represents an if node. An `isl_ast_node_block` represents a compound node. An `isl_ast_node_mark` introduces a mark in the AST. An `isl_ast_node_user` represents an expression statement. An expression statement typically corresponds to a domain element, i.e., one of the elements that is visited by the AST.

Each type of node has its own additional properties.

```
#include <isl/ast.h>
__isl_give isl_ast_expr *isl_ast_node_for_get_iterator(
        __isl_keep isl_ast_node *node);
__isl_give isl_ast_expr *isl_ast_node_for_get_init(
        __isl_keep isl_ast_node *node);
__isl_give isl_ast_expr *isl_ast_node_for_get_cond(
        __isl_keep isl_ast_node *node);
__isl_give isl_ast_expr *isl_ast_node_for_get_inc(
        __isl_keep isl_ast_node *node);
__isl_give isl_ast_node *isl_ast_node_for_get_body(
        __isl_keep isl_ast_node *node);
isl_bool isl_ast_node_for_is_degenerate(
        __isl_keep isl_ast_node *node);
```

An `isl_ast_for` is considered degenerate if it is known to execute exactly once.

```
#include <isl/ast.h>
__isl_give isl_ast_expr *isl_ast_node_if_get_cond(
        __isl_keep isl_ast_node *node);
__isl_give isl_ast_node *isl_ast_node_if_get_then(
        __isl_keep isl_ast_node *node);
isl_bool isl_ast_node_if_has_else(
        __isl_keep isl_ast_node *node);
__isl_give isl_ast_node *isl_ast_node_if_get_else(
        __isl_keep isl_ast_node *node);

__isl_give isl_ast_node_list *
isl_ast_node_block_get_children(
        __isl_keep isl_ast_node *node);

__isl_give isl_id *isl_ast_node_mark_get_id(
        __isl_keep isl_ast_node *node);
__isl_give isl_ast_node *isl_ast_node_mark_get_node(
        __isl_keep isl_ast_node *node);
```

`isl_ast_node_mark_get_id` returns the identifier of the mark. `isl_ast_node_mark_get_node` returns the child node that is being marked.

```
#include <isl/ast.h>
__isl_give isl_ast_expr *isl_ast_node_user_get_expr(
        __isl_keep isl_ast_node *node);
```

All descendants of a specific node in the AST (including the node itself) can be visited in depth-first pre-order using the following function.

```
#include <isl/ast.h>
isl_stat isl_ast_node_foreach_descendant_top_down(
        __isl_keep isl_ast_node *node,
        isl_bool (*fn)(__isl_keep isl_ast_node *node,
                void *user), void *user);
```

The callback function should return isl_bool_true if the children of the given node should be visited and isl_bool_false if they should not. It should return isl_bool_error in case of failure, in which case the entire traversal is aborted.

Each of the returned isl_ast_exprs can in turn be inspected using the following functions.

```
#include <isl/ast.h>
enum isl_ast_expr_type isl_ast_expr_get_type(
        __isl_keep isl_ast_expr *expr);
```

The type of an AST expression is one of isl_ast_expr_op, isl_ast_expr_id or isl_ast_expr_int. An isl_ast_expr_op represents the result of an operation. An isl_ast_expr_id represents an identifier. An isl_ast_expr_int represents an integer value.

Each type of expression has its own additional properties.

```
#include <isl/ast.h>
enum isl_ast_op_type isl_ast_expr_get_op_type(
        __isl_keep isl_ast_expr *expr);
int isl_ast_expr_get_op_n_arg(__isl_keep isl_ast_expr *expr);
__isl_give isl_ast_expr *isl_ast_expr_get_op_arg(
        __isl_keep isl_ast_expr *expr, int pos);
isl_stat isl_ast_expr_foreach_ast_op_type(
        __isl_keep isl_ast_expr *expr,
        isl_stat (*fn)(enum isl_ast_op_type type,
                void *user), void *user);
isl_stat isl_ast_node_foreach_ast_op_type(
        __isl_keep isl_ast_node *node,
        isl_stat (*fn)(enum isl_ast_op_type type,
                void *user), void *user);
```

isl_ast_expr_get_op_type returns the type of the operation performed. isl_ast_expr_get_op_n_arg returns the number of arguments. isl_ast_expr_get_op_arg returns the specified argument. isl_ast_expr_foreach_ast_op_type calls fn for each distinct isl_ast_op_type that appears in expr. isl_ast_node_foreach_ast_op_type does the same for each distinct isl_ast_op_type that appears in node. The operation type is one of the following.

isl_ast_op_and

Logical *and* of two arguments. Both arguments can be evaluated.

isl_ast_op_and_then

Logical *and* of two arguments. The second argument can only be evaluated if the first evaluates to true.

isl_ast_op_or

Logical *or* of two arguments. Both arguments can be evaluated.

isl_ast_op_or_else

Logical *or* of two arguments. The second argument can only be evaluated if the first evaluates to false.

isl_ast_op_max

Maximum of two or more arguments.

isl_ast_op_min

Minimum of two or more arguments.

isl_ast_op_minus

Change sign.

isl_ast_op_add

Sum of two arguments.

isl_ast_op_sub

Difference of two arguments.

isl_ast_op_mul

Product of two arguments.

isl_ast_op_div

Exact division. That is, the result is known to be an integer.

isl_ast_op_fdiv_q

Result of integer division, rounded towards negative infinity.

isl_ast_op_pdiv_q

Result of integer division, where dividend is known to be non-negative.

isl_ast_op_pdiv_r

Remainder of integer division, where dividend is known to be non-negative.

isl_ast_op_zdiv_r

Equal to zero iff the remainder on integer division is zero.

isl_ast_op_cond

Conditional operator defined on three arguments. If the first argument evaluates to true, then the result is equal to the second argument. Otherwise, the result is equal to the third argument. The second and third argument may only be evaluated if the first argument evaluates to true and false, respectively. Corresponds to `a ? b : c` in C.

isl_ast_op_select

Conditional operator defined on three arguments. If the first argument evaluates to true, then the result is equal to the second argument. Otherwise, the result is equal to the third argument. The second and third argument may be evaluated independently of the value of the first argument. Corresponds to `a * b + (1 - a) * c` in C.

isl_ast_op_eq

Equality relation.

isl_ast_op_le

Less than or equal relation.

isl_ast_op_lt

Less than relation.

isl_ast_op_ge

Greater than or equal relation.

isl_ast_op_gt

Greater than relation.

isl_ast_op_call

A function call. The number of arguments of the `isl_ast_expr` is one more than the number of arguments in the function call, the first argument representing the function being called.

isl_ast_op_access

An array access. The number of arguments of the `isl_ast_expr` is one more than the number of index expressions in the array access, the first argument representing the array being accessed.

isl_ast_op_member

A member access. This operation has two arguments, a structure and the name of the member of the structure being accessed.

```
#include <isl/ast.h>
__isl_give isl_id *isl_ast_expr_get_id(
        __isl_keep isl_ast_expr *expr);
```

Return the identifier represented by the AST expression.

```
#include <isl/ast.h>
__isl_give isl_val *isl_ast_expr_get_val(
        __isl_keep isl_ast_expr *expr);
```

Return the integer represented by the AST expression.

Properties of ASTs

```
#include <isl/ast.h>
isl_bool isl_ast_expr_is_equal(
        __isl_keep isl_ast_expr *expr1,
        __isl_keep isl_ast_expr *expr2);
```

Check if two `isl_ast_expr`s are equal to each other.

Manipulating and printing the AST

AST nodes can be copied and freed using the following functions.

```
#include <isl/ast.h>
__isl_give isl_ast_node *isl_ast_node_copy(
        __isl_keep isl_ast_node *node);
__isl_null isl_ast_node *isl_ast_node_free(
        __isl_take isl_ast_node *node);
```

AST expressions can be copied and freed using the following functions.

```
#include <isl/ast.h>
__isl_give isl_ast_expr *isl_ast_expr_copy(
        __isl_keep isl_ast_expr *expr);
__isl_null isl_ast_expr *isl_ast_expr_free(
        __isl_take isl_ast_expr *expr);
```

New AST expressions can be created either directly or within the context of an `isl_ast_build`.

```
#include <isl/ast.h>
__isl_give isl_ast_expr *isl_ast_expr_from_val(
        __isl_take isl_val *v);
__isl_give isl_ast_expr *isl_ast_expr_from_id(
        __isl_take isl_id *id);
__isl_give isl_ast_expr *isl_ast_expr_neg(
        __isl_take isl_ast_expr *expr);
__isl_give isl_ast_expr *isl_ast_expr_address_of(
        __isl_take isl_ast_expr *expr);
__isl_give isl_ast_expr *isl_ast_expr_add(
        __isl_take isl_ast_expr *expr1,
        __isl_take isl_ast_expr *expr2);
__isl_give isl_ast_expr *isl_ast_expr_sub(
        __isl_take isl_ast_expr *expr1,
        __isl_take isl_ast_expr *expr2);
__isl_give isl_ast_expr *isl_ast_expr_mul(
        __isl_take isl_ast_expr *expr1,
        __isl_take isl_ast_expr *expr2);
```

```
__isl_give isl_ast_expr *isl_ast_expr_div(
        __isl_take isl_ast_expr *expr1,
        __isl_take isl_ast_expr *expr2);
__isl_give isl_ast_expr *isl_ast_expr_pdiv_q(
        __isl_take isl_ast_expr *expr1,
        __isl_take isl_ast_expr *expr2);
__isl_give isl_ast_expr *isl_ast_expr_pdiv_r(
        __isl_take isl_ast_expr *expr1,
        __isl_take isl_ast_expr *expr2);
__isl_give isl_ast_expr *isl_ast_expr_and(
        __isl_take isl_ast_expr *expr1,
        __isl_take isl_ast_expr *expr2)
__isl_give isl_ast_expr *isl_ast_expr_and_then(
        __isl_take isl_ast_expr *expr1,
        __isl_take isl_ast_expr *expr2)
__isl_give isl_ast_expr *isl_ast_expr_or(
        __isl_take isl_ast_expr *expr1,
        __isl_take isl_ast_expr *expr2)
__isl_give isl_ast_expr *isl_ast_expr_or_else(
        __isl_take isl_ast_expr *expr1,
        __isl_take isl_ast_expr *expr2)
__isl_give isl_ast_expr *isl_ast_expr_eq(
        __isl_take isl_ast_expr *expr1,
        __isl_take isl_ast_expr *expr2);
__isl_give isl_ast_expr *isl_ast_expr_le(
        __isl_take isl_ast_expr *expr1,
        __isl_take isl_ast_expr *expr2);
__isl_give isl_ast_expr *isl_ast_expr_lt(
        __isl_take isl_ast_expr *expr1,
        __isl_take isl_ast_expr *expr2);
__isl_give isl_ast_expr *isl_ast_expr_ge(
        __isl_take isl_ast_expr *expr1,
        __isl_take isl_ast_expr *expr2);
__isl_give isl_ast_expr *isl_ast_expr_gt(
        __isl_take isl_ast_expr *expr1,
        __isl_take isl_ast_expr *expr2);
__isl_give isl_ast_expr *isl_ast_expr_access(
        __isl_take isl_ast_expr *array,
        __isl_take isl_ast_expr_list *indices);
__isl_give isl_ast_expr *isl_ast_expr_call(
        __isl_take isl_ast_expr *function,
        __isl_take isl_ast_expr_list *arguments);
```

The function `isl_ast_expr_address_of` can be applied to an `isl_ast_expr` of type `isl_ast_op_access` only. It is meant to represent the address of the `isl_ast_expr_access`. The function `isl_ast_expr_and_then` as well as `isl_ast_expr_or_else` are short-

circuit versions of isl_ast_expr_and and isl_ast_expr_or, respectively.

```
#include <isl/ast_build.h>
__isl_give isl_ast_expr *isl_ast_build_expr_from_set(
        __isl_keep isl_ast_build *build,
        __isl_take isl_set *set);
__isl_give isl_ast_expr *isl_ast_build_expr_from_pw_aff(
        __isl_keep isl_ast_build *build,
        __isl_take isl_pw_aff *pa);
__isl_give isl_ast_expr *
isl_ast_build_access_from_pw_multi_aff(
        __isl_keep isl_ast_build *build,
        __isl_take isl_pw_multi_aff *pma);
__isl_give isl_ast_expr *
isl_ast_build_access_from_multi_pw_aff(
        __isl_keep isl_ast_build *build,
        __isl_take isl_multi_pw_aff *mpa);
__isl_give isl_ast_expr *
isl_ast_build_call_from_pw_multi_aff(
        __isl_keep isl_ast_build *build,
        __isl_take isl_pw_multi_aff *pma);
__isl_give isl_ast_expr *
isl_ast_build_call_from_multi_pw_aff(
        __isl_keep isl_ast_build *build,
        __isl_take isl_multi_pw_aff *mpa);
```

The set <set> and the domains of pa, mpa and pma should correspond to the schedule space of build. The tuple id of mpa or pma is used as the array being accessed or the function being called. If the accessed space is a nested relation, then it is taken to represent an access of the member specified by the range of this nested relation of the structure specified by the domain of the nested relation.

The following functions can be used to modify an isl_ast_expr.

```
#include <isl/ast.h>
__isl_give isl_ast_expr *isl_ast_expr_set_op_arg(
        __isl_take isl_ast_expr *expr, int pos,
        __isl_take isl_ast_expr *arg);
```

Replace the argument of expr at position pos by arg.

```
#include <isl/ast.h>
__isl_give isl_ast_expr *isl_ast_expr_substitute_ids(
        __isl_take isl_ast_expr *expr,
        __isl_take isl_id_to_ast_expr *id2expr);
```

The function isl_ast_expr_substitute_ids replaces the subexpressions of expr of type isl_ast_expr_id by the corresponding expression in id2expr, if there is any.

User specified data can be attached to an isl_ast_node and obtained from the same isl_ast_node using the following functions.

```
#include <isl/ast.h>
__isl_give isl_ast_node *isl_ast_node_set_annotation(
        __isl_take isl_ast_node *node,
        __isl_take isl_id *annotation);
__isl_give isl_id *isl_ast_node_get_annotation(
        __isl_keep isl_ast_node *node);
```

Basic printing can be performed using the following functions.

```
#include <isl/ast.h>
__isl_give isl_printer *isl_printer_print_ast_expr(
        __isl_take isl_printer *p,
        __isl_keep isl_ast_expr *expr);
__isl_give isl_printer *isl_printer_print_ast_node(
        __isl_take isl_printer *p,
        __isl_keep isl_ast_node *node);
__isl_give char *isl_ast_expr_to_str(
        __isl_keep isl_ast_expr *expr);
```

More advanced printing can be performed using the following functions.

```
#include <isl/ast.h>
__isl_give isl_printer *isl_ast_op_type_set_print_name(
        __isl_take isl_printer *p,
        enum isl_ast_op_type type,
        __isl_keep const char *name);
isl_stat isl_options_set_ast_print_macro_once(
        isl_ctx *ctx, int val);
int isl_options_get_ast_print_macro_once(isl_ctx *ctx);
__isl_give isl_printer *isl_ast_op_type_print_macro(
        enum isl_ast_op_type type,
        __isl_take isl_printer *p);
__isl_give isl_printer *isl_ast_expr_print_macros(
        __isl_keep isl_ast_expr *expr,
        __isl_take isl_printer *p);
__isl_give isl_printer *isl_ast_node_print_macros(
        __isl_keep isl_ast_node *node,
        __isl_take isl_printer *p);
__isl_give isl_printer *isl_ast_node_print(
        __isl_keep isl_ast_node *node,
        __isl_take isl_printer *p,
        __isl_take isl_ast_print_options *options);
__isl_give isl_printer *isl_ast_node_for_print(
        __isl_keep isl_ast_node *node,
```

```
        __isl_take isl_printer *p,
        __isl_take isl_ast_print_options *options);
__isl_give isl_printer *isl_ast_node_if_print(
        __isl_keep isl_ast_node *node,
        __isl_take isl_printer *p,
        __isl_take isl_ast_print_options *options);
```

While printing an isl_ast_node in ISL_FORMAT_C, isl may print out an AST that makes use of macros such as floord, min and max. The names of these macros may be modified by a call to isl_ast_op_type_set_print_name. The user-specified names are associated to the printer object. isl_ast_op_type_print_macro prints out the macro corresponding to a specific isl_ast_op_type. If the print-macro-once option is set, then a given macro definition is only printed once to any given printer object. isl_ast_expr_print_macros scans the isl_ast_expr for subexpressions where these macros would be used and prints out the required macro definitions. Essentially, isl_ast_expr_print_macros calls isl_ast_expr_foreach_ast_op_type with isl_ast_op_type_print_macro as function argument. isl_ast_node_print_macros does the same for expressions in its isl_ast_node argument. isl_ast_node_print, isl_ast_node_for_print and isl_ast_node_if_print print an isl_ast_node in ISL_FORMAT_C, but allow for some extra control through an isl_ast_print_options object. This object can be created using the following functions.

```
#include <isl/ast.h>
__isl_give isl_ast_print_options *
isl_ast_print_options_alloc(isl_ctx *ctx);
__isl_give isl_ast_print_options *
isl_ast_print_options_copy(
        __isl_keep isl_ast_print_options *options);
__isl_null isl_ast_print_options *
isl_ast_print_options_free(
        __isl_take isl_ast_print_options *options);

__isl_give isl_ast_print_options *
isl_ast_print_options_set_print_user(
        __isl_take isl_ast_print_options *options,
        __isl_give isl_printer *(*print_user)(
                __isl_take isl_printer *p,
                __isl_take isl_ast_print_options *options,
                __isl_keep isl_ast_node *node, void *user),
        void *user);
__isl_give isl_ast_print_options *
isl_ast_print_options_set_print_for(
        __isl_take isl_ast_print_options *options,
        __isl_give isl_printer *(*print_for)(
                __isl_take isl_printer *p,
                __isl_take isl_ast_print_options *options,
                __isl_keep isl_ast_node *node, void *user),
```

```
        void *user);
```

The callback set by `isl_ast_print_options_set_print_user` is called whenever a node of type `isl_ast_node_user` needs to be printed. The callback set by `isl_ast_print_options_set_print_for` is called whenever a node of type `isl_ast_node_for` needs to be printed. Note that `isl_ast_node_for_print` will *not* call the callback set by `isl_ast_print_options_set_print_for` on the node on which `isl_ast_node_for_print` is called, but only on nested nodes of type `isl_ast_node_for`. It is therefore safe to call `isl_ast_node_for_print` from within the callback set by `isl_ast_print_options_set_print_for`.

The following option determines the type to be used for iterators while printing the AST.

```
isl_stat isl_options_set_ast_iterator_type(
        isl_ctx *ctx, const char *val);
const char *isl_options_get_ast_iterator_type(
        isl_ctx *ctx);
```

The AST printer only prints body nodes as blocks if these blocks cannot be safely omitted. For example, a `for` node with one body node will not be surrounded with braces in `ISL_FORMAT_C`. A block will always be printed by setting the following option.

```
isl_stat isl_options_set_ast_always_print_block(isl_ctx *ctx,
        int val);
int isl_options_get_ast_always_print_block(isl_ctx *ctx);
```

Options

```
#include <isl/ast_build.h>
isl_stat isl_options_set_ast_build_atomic_upper_bound(
        isl_ctx *ctx, int val);
int isl_options_get_ast_build_atomic_upper_bound(
        isl_ctx *ctx);
isl_stat isl_options_set_ast_build_prefer_pdiv(isl_ctx *ctx,
        int val);
int isl_options_get_ast_build_prefer_pdiv(isl_ctx *ctx);
isl_stat isl_options_set_ast_build_detect_min_max(
        isl_ctx *ctx, int val);
int isl_options_get_ast_build_detect_min_max(
        isl_ctx *ctx);
isl_stat isl_options_set_ast_build_exploit_nested_bounds(
        isl_ctx *ctx, int val);
int isl_options_get_ast_build_exploit_nested_bounds(
        isl_ctx *ctx);
isl_stat isl_options_set_ast_build_group_coscheduled(
        isl_ctx *ctx, int val);
int isl_options_get_ast_build_group_coscheduled(
```

```
        isl_ctx *ctx);
isl_stat isl_options_set_ast_build_scale_strides(
        isl_ctx *ctx, int val);
int isl_options_get_ast_build_scale_strides(
        isl_ctx *ctx);
isl_stat isl_options_set_ast_build_allow_else(isl_ctx *ctx,
        int val);
int isl_options_get_ast_build_allow_else(isl_ctx *ctx);
isl_stat isl_options_set_ast_build_allow_or(isl_ctx *ctx,
        int val);
int isl_options_get_ast_build_allow_or(isl_ctx *ctx);
```

- ast_build_atomic_upper_bound

  Generate loop upper bounds that consist of the current loop iterator, an operator and an expression not involving the iterator. If this option is not set, then the current loop iterator may appear several times in the upper bound. For example, when this option is turned off, AST generation for the schedule

  ```
  [n] -> { A[i] -> [i] : 0 <= i <= 100, n }
  ```

  produces

  ```
  for (int c0 = 0; c0 <= 100 && n >= c0; c0 += 1)
    A(c0);
  ```

  When the option is turned on, the following AST is generated

  ```
  for (int c0 = 0; c0 <= min(100, n); c0 += 1)
    A(c0);
  ```

- ast_build_prefer_pdiv

  If this option is turned off, then the AST generation will produce ASTs that may only contain `isl_ast_op_fdiv_q` operators, but no `isl_ast_op_pdiv_q` or `isl_ast_op_pdiv_r` operators. If this option is turned on, then `isl` will try to convert some of the `isl_ast_op_fdiv_q` operators to (expressions containing) `isl_ast_op_pdiv_q` or `isl_ast_op_pdiv_r` operators.

- ast_build_detect_min_max

  If this option is turned on, then `isl` will try and detect min or max-expressions when building AST expressions from piecewise affine expressions.

- ast_build_exploit_nested_bounds

  Simplify conditions based on bounds of nested for loops. In particular, remove conditions that are implied by the fact that one or more nested loops have at least one iteration, meaning that the upper bound is at least as large as the lower bound. For example, when this option is turned off, AST generation for the schedule

```
[N,M] -> { A[i,j] -> [i,j] : 0 <= i <= N and
                                 0 <= j <= M }
```

produces

```
if (M >= 0)
  for (int c0 = 0; c0 <= N; c0 += 1)
    for (int c1 = 0; c1 <= M; c1 += 1)
      A(c0, c1);
```

When the option is turned on, the following AST is generated

```
for (int c0 = 0; c0 <= N; c0 += 1)
  for (int c1 = 0; c1 <= M; c1 += 1)
    A(c0, c1);
```

- ast_build_group_coscheduled

  If two domain elements are assigned the same schedule point, then they may be executed in any order and they may even appear in different loops. If this options is set, then the AST generator will make sure that coscheduled domain elements do not appear in separate parts of the AST. This is useful in case of nested AST generation if the outer AST generation is given only part of a schedule and the inner AST generation should handle the domains that are coscheduled by this initial part of the schedule together. For example if an AST is generated for a schedule

  ```
  { A[i] -> [0]; B[i] -> [0] }
  ```

  then the `isl_ast_build_set_create_leaf` callback described below may get called twice, once for each domain. Setting this option ensures that the callback is only called once on both domains together.

- ast_build_separation_bounds

  This option specifies which bounds to use during separation. If this option is set to `ISL_AST_BUILD_SEPARATION_BOUNDS_IMPLICIT` then all (possibly implicit) bounds on the current dimension will be used during separation. If this option is set to `ISL_AST_BUILD_SEPARATION_BOUNDS_EXPLICIT` then only those bounds that are explicitly available will be used during separation.

- ast_build_scale_strides

  This option specifies whether the AST generator is allowed to scale down iterators of strided loops.

- ast_build_allow_else

  This option specifies whether the AST generator is allowed to construct if statements with else branches.

- ast_build_allow_or

  This option specifies whether the AST generator is allowed to construct if conditions with disjunctions.

AST Generation Options (Schedule Tree)

In case of AST construction from a schedule tree, the options that control how an AST is created from the individual schedule dimensions are stored in the band nodes of the tree (see §1.5.1).

In particular, a schedule dimension can be handled in four different ways, atomic, separate, unroll or the default. This loop AST generation type can be set using `isl_schedule_node_band_member_set` Alternatively, the first three can be selected by including a one-dimensional element with as value the position of the schedule dimension within the band and as name one of `atomic`, `separate` or `unroll` in the options set by `isl_schedule_node_band_set_ast_build_options`. Only one of these three may be specified for any given schedule dimension within a band node. If none of these is specified, then the default is used. The meaning of the options is as follows.

`atomic`

When this option is specified, the AST generator will make sure that a given domains space only appears in a single loop at the specified level.

For example, for the schedule tree

```
domain: "{ a[i] : 0 <= i < 10; b[i] : 0 <= i < 10 }"
child:
  schedule: "[{ a[i] -> [i]; b[i] -> [i+1] }]"
  options: "{ atomic[x] }"
```

the following AST will be generated

```
for (int c0 = 0; c0 <= 10; c0 += 1) {
  if (c0 >= 1)
    b(c0 - 1);
  if (c0 <= 9)
    a(c0);
}
```

On the other hand, for the schedule tree

```
domain: "{ a[i] : 0 <= i < 10; b[i] : 0 <= i < 10 }"
child:
  schedule: "[{ a[i] -> [i]; b[i] -> [i+1] }]"
  options: "{ separate[x] }"
```

the following AST will be generated

```
{
  a(0);
  for (int c0 = 1; c0 <= 9; c0 += 1) {
    b(c0 - 1);
    a(c0);
  }
  b(9);
}
```

If neither `atomic` nor `separate` is specified, then the AST generator may produce either of these two results or some intermediate form.

`separate`

When this option is specified, the AST generator will split the domain of the specified schedule dimension into pieces with a fixed set of statements for which instances need to be executed by the iterations in the schedule domain part. This option tends to avoid the generation of guards inside the corresponding loops. See also the `atomic` option.

`unroll`

When this option is specified, the AST generator will *completely* unroll the corresponding schedule dimension. It is the responsibility of the user to ensure that such unrolling is possible. To obtain a partial unrolling, the user should apply an additional strip-mining to the schedule and fully unroll the inner schedule dimension.

The `isolate` option is a bit more involved. It allows the user to isolate a range of schedule dimension values from smaller and greater values. Additionally, the user may specify a different atomic/separate/unroll choice for the isolated part and the remaining parts. The typical use case of the `isolate` option is to isolate full tiles from partial tiles. The part that needs to be isolated may depend on outer schedule dimensions. The option therefore needs to be able to reference those outer schedule dimensions. In particular, the space of the `isolate` option is that of a wrapped map with as domain the flat product of all outer band nodes and as range the space of the current band node. The atomic/separate/unroll choice for the isolated part is determined by an option that lives in an unnamed wrapped space with as domain a zero-dimensional `isolate` space and as range the regular `atomic`, `separate` or `unroll` space. This option may also be set directly using `isl_schedule_node_band_member_set_isolate_ast_loop_type`. The atomic/separate/unroll choice for the remaining part is determined by the regular `atomic`, `separate` or `unroll` option. The use of the `isolate` option causes any tree containing the node to be considered anchored.

As an example, consider the isolation of full tiles from partial tiles in a tiling of a triangular domain. The original schedule is as follows.

```
domain: "{ A[i,j] : 0 <= i,j and i + j <= 100 }"
child:
```

```
  schedule: "[{ A[i,j] -> [floor(i/10)] }, \
        { A[i,j] -> [floor(j/10)] }, \
        { A[i,j] -> [i] }, { A[i,j] -> [j] }]"
```

The output is

```
for (int c0 = 0; c0 <= 10; c0 += 1)
  for (int c1 = 0; c1 <= -c0 + 10; c1 += 1)
    for (int c2 = 10 * c0;
         c2 <= min(10 * c0 + 9, -10 * c1 + 100); c2 += 1)
      for (int c3 = 10 * c1;
           c3 <= min(10 * c1 + 9, -c2 + 100); c3 += 1)
        A(c2, c3);
```

Isolating the full tiles, we have the following input

```
domain: "{ A[i,j] : 0 <= i,j and i + j <= 100 }"
child:
  schedule: "[{ A[i,j] -> [floor(i/10)] }, \
        { A[i,j] -> [floor(j/10)] }, \
        { A[i,j] -> [i] }, { A[i,j] -> [j] }]"
  options: "{ isolate[[] -> [a,b,c,d]] : 0 <= 10a,10b and \
        10a+9+10b+9 <= 100 }"
```

and output

```
{
  for (int c0 = 0; c0 <= 8; c0 += 1) {
    for (int c1 = 0; c1 <= -c0 + 8; c1 += 1)
      for (int c2 = 10 * c0;
           c2 <= 10 * c0 + 9; c2 += 1)
        for (int c3 = 10 * c1;
             c3 <= 10 * c1 + 9; c3 += 1)
          A(c2, c3);
    for (int c1 = -c0 + 9; c1 <= -c0 + 10; c1 += 1)
      for (int c2 = 10 * c0;
           c2 <= min(10 * c0 + 9, -10 * c1 + 100); c2 += 1)
        for (int c3 = 10 * c1;
             c3 <= min(10 * c1 + 9, -c2 + 100); c3 += 1)
          A(c2, c3);
  }
  for (int c0 = 9; c0 <= 10; c0 += 1)
    for (int c1 = 0; c1 <= -c0 + 10; c1 += 1)
      for (int c2 = 10 * c0;
           c2 <= min(10 * c0 + 9, -10 * c1 + 100); c2 += 1)
        for (int c3 = 10 * c1;
             c3 <= min(10 * c1 + 9, -c2 + 100); c3 += 1)
          A(c2, c3);
}
```

We may then additionally unroll the innermost loop of the isolated part

```
domain: "{ A[i,j] : 0 <= i,j and i + j <= 100 }"
child:
  schedule: "[{ A[i,j] -> [floor(i/10)] }, \
        { A[i,j] -> [floor(j/10)] }, \
        { A[i,j] -> [i] }, { A[i,j] -> [j] }]"
  options: "{ isolate[[] -> [a,b,c,d]] : 0 <= 10a,10b and \
        10a+9+10b+9 <= 100; [isolate[] -> unroll[3]] }"
```

to obtain

```
{
  for (int c0 = 0; c0 <= 8; c0 += 1) {
    for (int c1 = 0; c1 <= -c0 + 8; c1 += 1)
      for (int c2 = 10 * c0; c2 <= 10 * c0 + 9; c2 += 1) {
        A(c2, 10 * c1);
        A(c2, 10 * c1 + 1);
        A(c2, 10 * c1 + 2);
        A(c2, 10 * c1 + 3);
        A(c2, 10 * c1 + 4);
        A(c2, 10 * c1 + 5);
        A(c2, 10 * c1 + 6);
        A(c2, 10 * c1 + 7);
        A(c2, 10 * c1 + 8);
        A(c2, 10 * c1 + 9);
      }
    for (int c1 = -c0 + 9; c1 <= -c0 + 10; c1 += 1)
      for (int c2 = 10 * c0;
           c2 <= min(10 * c0 + 9, -10 * c1 + 100); c2 += 1)
        for (int c3 = 10 * c1;
             c3 <= min(10 * c1 + 9, -c2 + 100); c3 += 1)
          A(c2, c3);
  }
  for (int c0 = 9; c0 <= 10; c0 += 1)
    for (int c1 = 0; c1 <= -c0 + 10; c1 += 1)
      for (int c2 = 10 * c0;
           c2 <= min(10 * c0 + 9, -10 * c1 + 100); c2 += 1)
        for (int c3 = 10 * c1;
             c3 <= min(10 * c1 + 9, -c2 + 100); c3 += 1)
          A(c2, c3);
}
```

AST Generation Options (Schedule Map)

In case of AST construction using `isl_ast_build_node_from_schedule_map`, the options that control how an AST is created from the individual schedule dimensions are stored in the `isl_ast_build`. They can be set using the following function.

```
#include <isl/ast_build.h>
__isl_give isl_ast_build *
isl_ast_build_set_options(
        __isl_take isl_ast_build *control,
        __isl_take isl_union_map *options);
```

The options are encoded in an `isl_union_map`. The domain of this union relation refers to the schedule domain, i.e., the range of the schedule passed to `isl_ast_build_node_from_schedule_map`. In the case of nested AST generation (see §1.5.4), the domain of `options` should refer to the extra piece of the schedule. That is, it should be equal to the range of the wrapped relation in the range of the schedule. The range of the options can consist of elements in one or more spaces, the names of which determine the effect of the option. The values of the range typically also refer to the schedule dimension to which the option applies. In case of nested AST generation (see §1.5.4), these values refer to the position of the schedule dimension within the innermost AST generation. The constraints on the domain elements of the option should only refer to this dimension and earlier dimensions. We consider the following spaces.

`separation_class`

This option has been deprecated. Use the isolate option on schedule trees instead.

This space is a wrapped relation between two one dimensional spaces. The input space represents the schedule dimension to which the option applies and the output space represents the separation class. While constructing a loop corresponding to the specified schedule dimension(s), the AST generator will try to generate separate loops for domain elements that are assigned different classes. If only some of the elements are assigned a class, then those elements that are not assigned any class will be treated as belonging to a class that is separate from the explicitly assigned classes. The typical use case for this option is to separate full tiles from partial tiles. The other options, described below, are applied after the separation into classes.

As an example, consider the separation into full and partial tiles of a tiling of a triangular domain. Take, for example, the domain

```
{ A[i,j] : 0 <= i,j and i + j <= 100 }
```

and a tiling into tiles of 10 by 10. The input to the AST generator is then the schedule

```
{ A[i,j] -> [([i/10]),[j/10],i,j] : 0 <= i,j and
                                    i + j <= 100 }
```

Without any options, the following AST is generated

```
for (int c0 = 0; c0 <= 10; c0 += 1)
  for (int c1 = 0; c1 <= -c0 + 10; c1 += 1)
    for (int c2 = 10 * c0;
         c2 <= min(-10 * c1 + 100, 10 * c0 + 9);
         c2 += 1)
      for (int c3 = 10 * c1;
           c3 <= min(10 * c1 + 9, -c2 + 100);
           c3 += 1)
        A(c2, c3);
```

Separation into full and partial tiles can be obtained by assigning a class, say `0`, to the full tiles. The full tiles are represented by those values of the first and second schedule dimensions for which there are values of the third and fourth dimensions to cover an entire tile. That is, we need to specify the following option

```
{ [a,b,c,d] -> separation_class[[0]->[0]] :
       exists b': 0 <= 10a,10b' and
                  10a+9+10b'+9 <= 100;
  [a,b,c,d] -> separation_class[[1]->[0]] :
       0 <= 10a,10b and 10a+9+10b+9 <= 100 }
```

which simplifies to

```
{ [a, b, c, d] -> separation_class[[1] -> [0]] :
       a >= 0 and b >= 0 and b <= 8 - a;
  [a, b, c, d] -> separation_class[[0] -> [0]] :
       a >= 0 and a <= 8 }
```

With this option, the generated AST is as follows

```
{
  for (int c0 = 0; c0 <= 8; c0 += 1) {
    for (int c1 = 0; c1 <= -c0 + 8; c1 += 1)
      for (int c2 = 10 * c0;
           c2 <= 10 * c0 + 9; c2 += 1)
        for (int c3 = 10 * c1;
             c3 <= 10 * c1 + 9; c3 += 1)
          A(c2, c3);
    for (int c1 = -c0 + 9; c1 <= -c0 + 10; c1 += 1)
      for (int c2 = 10 * c0;
           c2 <= min(-10 * c1 + 100, 10 * c0 + 9);
           c2 += 1)
        for (int c3 = 10 * c1;
             c3 <= min(-c2 + 100, 10 * c1 + 9);
```

```
            c3 += 1)
          A(c2, c3);
  }
  for (int c0 = 9; c0 <= 10; c0 += 1)
    for (int c1 = 0; c1 <= -c0 + 10; c1 += 1)
      for (int c2 = 10 * c0;
           c2 <= min(-10 * c1 + 100, 10 * c0 + 9);
           c2 += 1)
        for (int c3 = 10 * c1;
             c3 <= min(10 * c1 + 9, -c2 + 100);
             c3 += 1)
          A(c2, c3);
}
```

separate

This is a single-dimensional space representing the schedule dimension(s) to which "separation" should be applied. Separation tries to split a loop into several pieces if this can avoid the generation of guards inside the loop. See also the `atomic` option.

atomic

This is a single-dimensional space representing the schedule dimension(s) for which the domains should be considered "atomic". That is, the AST generator will make sure that any given domain space will only appear in a single loop at the specified level.

Consider the following schedule

```
{ a[i] -> [i] : 0 <= i < 10;
  b[i] -> [i+1] : 0 <= i < 10 }
```

If the following option is specified

```
{ [i] -> separate[x] }
```

then the following AST will be generated

```
{
  a(0);
  for (int c0 = 1; c0 <= 9; c0 += 1) {
    a(c0);
    b(c0 - 1);
  }
  b(9);
}
```

If, on the other hand, the following option is specified

```
{ [i] -> atomic[x] }
```

then the following AST will be generated

```
for (int c0 = 0; c0 <= 10; c0 += 1) {
  if (c0 <= 9)
    a(c0);
  if (c0 >= 1)
    b(c0 - 1);
}
```

If neither `atomic` nor `separate` is specified, then the AST generator may produce either of these two results or some intermediate form.

`unroll`

This is a single-dimensional space representing the schedule dimension(s) that should be *completely* unrolled. To obtain a partial unrolling, the user should apply an additional strip-mining to the schedule and fully unroll the inner loop.

Fine-grained Control over AST Generation

Besides specifying the constraints on the parameters, an `isl_ast_build` object can be used to control various aspects of the AST generation process. In case of AST construction using `isl_ast_build_node_from_schedule_map`, the most prominent way of control is through "options", as explained above.

Additional control is available through the following functions.

```
#include <isl/ast_build.h>
__isl_give isl_ast_build *
isl_ast_build_set_iterators(
        __isl_take isl_ast_build *control,
        __isl_take isl_id_list *iterators);
```

The function `isl_ast_build_set_iterators` allows the user to specify a list of iterator `isl_id`s to be used as iterators. If the input schedule is injective, then the number of elements in this list should be as large as the dimension of the schedule space, but no direct correspondence should be assumed between dimensions and elements. If the input schedule is not injective, then an additional number of `isl_id`s equal to the largest dimension of the input domains may be required. If the number of provided `isl_id`s is insufficient, then additional names are automatically generated.

```
#include <isl/ast_build.h>
__isl_give isl_ast_build *
isl_ast_build_set_create_leaf(
```

```
        __isl_take isl_ast_build *control,
        __isl_give isl_ast_node *(*fn)(
                __isl_take isl_ast_build *build,
                void *user), void *user);
```

The `isl_ast_build_set_create_leaf` function allows for the specification of a callback that should be called whenever the AST generator arrives at an element of the schedule domain. The callback should return an AST node that should be inserted at the corresponding position of the AST. The default action (when the callback is not set) is to continue generating parts of the AST to scan all the domain elements associated to the schedule domain element and to insert user nodes, "calling" the domain element, for each of them. The `build` argument contains the current state of the `isl_ast_build`. To ease nested AST generation (see §1.5.4), all control information that is specific to the current AST generation such as the options and the callbacks has been removed from this `isl_ast_build`. The callback would typically return the result of a nested AST generation or a user defined node created using the following function.

```
#include <isl/ast.h>
__isl_give isl_ast_node *isl_ast_node_alloc_user(
        __isl_take isl_ast_expr *expr);

#include <isl/ast_build.h>
__isl_give isl_ast_build *
isl_ast_build_set_at_each_domain(
        __isl_take isl_ast_build *build,
        __isl_give isl_ast_node *(*fn)(
                __isl_take isl_ast_node *node,
                __isl_keep isl_ast_build *build,
                void *user), void *user);
__isl_give isl_ast_build *
isl_ast_build_set_before_each_for(
        __isl_take isl_ast_build *build,
        __isl_give isl_id *(*fn)(
                __isl_keep isl_ast_build *build,
                void *user), void *user);
__isl_give isl_ast_build *
isl_ast_build_set_after_each_for(
        __isl_take isl_ast_build *build,
        __isl_give isl_ast_node *(*fn)(
                __isl_take isl_ast_node *node,
                __isl_keep isl_ast_build *build,
                void *user), void *user);
__isl_give isl_ast_build *
isl_ast_build_set_before_each_mark(
        __isl_take isl_ast_build *build,
        isl_stat (*fn)(__isl_keep isl_id *mark,
                __isl_keep isl_ast_build *build,
```

```
                void *user), void *user);
__isl_give isl_ast_build *
isl_ast_build_set_after_each_mark(
        __isl_take isl_ast_build *build,
        __isl_give isl_ast_node *(*fn)(
                __isl_take isl_ast_node *node,
                __isl_keep isl_ast_build *build,
                void *user), void *user);
```

The callback set by `isl_ast_build_set_at_each_domain` will be called for each domain AST node. The callbacks set by `isl_ast_build_set_before_each_for` and `isl_ast_build_set_after_each_for` will be called for each for AST node. The first will be called in depth-first pre-order, while the second will be called in depth-first post-order. Since `isl_ast_build_set_before_each_for` is called before the for node is actually constructed, it is only passed an `isl_ast_build`. The returned `isl_id` will be added as an annotation (using `isl_ast_node_set_annotation`) to the constructed for node. In particular, if the user has also specified an `after_each_for` callback, then the annotation can be retrieved from the node passed to that callback using `isl_ast_node_get_annotation`. The callbacks set by `isl_ast_build_set_before_each_mark` and `isl_ast_build_set_after_each_mark` will be called for each mark AST node that is created, i.e., for each mark schedule node in the input schedule tree. The first will be called in depth-first pre-order, while the second will be called in depth-first post-order. Since the callback set by `isl_ast_build_set_before_each_mark` is called before the mark AST node is actually constructed, it is passed the identifier of the mark node. All callbacks should `NULL` (or -1) on failure. The given `isl_ast_build` can be used to create new `isl_ast_expr` objects using `isl_ast_build_expr_from_pw_aff` or `isl_ast_build_call_from_pw_multi_aff`.

Nested AST Generation

`isl` allows the user to create an AST within the context of another AST. These nested ASTs are created using the same `isl_ast_build_node_from_schedule_map` function that is used to create the outer AST. The `build` argument should be an `isl_ast_build` passed to a callback set by `isl_ast_build_set_create_leaf`. The space of the range of the `schedule` argument should refer to this build. In particular, the space should be a wrapped relation and the domain of this wrapped relation should be the same as that of the range of the schedule returned by `isl_ast_build_get_schedule` below. In practice, the new schedule is typically created by calling `isl_union_map_range_product` on the old schedule and some extra piece of the schedule. The space of the schedule domain is also available from the `isl_ast_build`.

```
#include <isl/ast_build.h>
__isl_give isl_union_map *isl_ast_build_get_schedule(
        __isl_keep isl_ast_build *build);
__isl_give isl_space *isl_ast_build_get_schedule_space(
        __isl_keep isl_ast_build *build);
```

```
__isl_give isl_ast_build *isl_ast_build_restrict(
        __isl_take isl_ast_build *build,
        __isl_take isl_set *set);
```

The isl_ast_build_get_schedule function returns a (partial) schedule for the domains elements for which part of the AST still needs to be generated in the current build. In particular, the domain elements are mapped to those iterations of the loops enclosing the current point of the AST generation inside which the domain elements are executed. No direct correspondence between the input schedule and this schedule should be assumed. The space obtained from isl_ast_build_get_schedule_space can be used to create a set for isl_ast_build_restrict to intersect with the current build. In particular, the set passed to isl_ast_build_restrict can have additional parameters. The ids of the set dimensions in the space returned by isl_ast_build_get_schedule_space correspond to the iterators of the already generated loops. The user should not rely on the ids of the output dimensions of the relations in the union relation returned by isl_ast_build_get_schedule having any particular value.

1.6 Applications

Although isl is mainly meant to be used as a library, it also contains some basic applications that use some of the functionality of isl. The input may be specified in either the isl format or the PolyLib format.

1.6.1 isl_polyhedron_sample

isl_polyhedron_sample takes a polyhedron as input and prints an integer element of the polyhedron, if there is any. The first column in the output is the denominator and is always equal to 1. If the polyhedron contains no integer points, then a vector of length zero is printed.

1.6.2 isl_pip

isl_pip takes the same input as the example program from the piplib distribution, i.e., a set of constraints on the parameters, a line containing only -1 and finally a set of constraints on a parametric polyhedron. The coefficients of the parameters appear in the last columns (but before the final constant column). The output is the lexicographic minimum of the parametric polyhedron. As isl currently does not have its own output format, the output is just a dump of the internal state.

1.6.3 isl_polyhedron_minimize

isl_polyhedron_minimize computes the minimum of some linear or affine objective function over the integer points in a polyhedron. If an affine objective function is given, then the constant should appear in the last column.

1.6.4 isl_polytope_scan

Given a polytope, `isl_polytope_scan` prints all integer points in the polytope.

1.6.5 isl_codegen

Given a schedule, a context set and an options relation, `isl_codegen` prints out an AST that scans the domain elements of the schedule in the order of their image(s) taking into account the constraints in the context set.

Chapter 2

Implementation Details

2.1 Sets and Relations

Definition 2.1.1 (Polyhedral Set) *A* polyhedral set *S is a finite union of basic sets $S = \bigcup_i S_i$, each of which can be represented using affine constraints*

$$S_i : \mathbb{Z}^n \to 2^{\mathbb{Z}^d} : \mathbf{s} \mapsto S_i(\mathbf{s}) = \{\, \mathbf{x} \in \mathbb{Z}^d \mid \exists \mathbf{z} \in \mathbb{Z}^e : A\mathbf{x} + B\mathbf{s} + D\mathbf{z} + \mathbf{c} \geq \mathbf{0} \,\},$$

with $A \in \mathbb{Z}^{m \times d}$, $B \in \mathbb{Z}^{m \times n}$, $D \in \mathbb{Z}^{m \times e}$ and $\mathbf{c} \in \mathbb{Z}^m$.

Definition 2.1.2 (Parameter Domain of a Set) *Let $S \in \mathbb{Z}^n \to 2^{\mathbb{Z}^d}$ be a set. The* parameter domain *of S is the set*

$$\operatorname{pdom} S \coloneqq \{\, \mathbf{s} \in \mathbb{Z}^n \mid S(\mathbf{s}) \neq \emptyset \,\}.$$

Definition 2.1.3 (Polyhedral Relation) *A* polyhedral relation *R is a finite union of basic relations $R = \bigcup_i R_i$ of type $\mathbb{Z}^n \to 2^{\mathbb{Z}^{d_1 + d_2}}$, each of which can be represented using affine constraints*

$$R_i = \mathbf{s} \mapsto R_i(\mathbf{s}) = \{\, \mathbf{x}_1 \to \mathbf{x}_2 \in \mathbb{Z}^{d_1} \times \mathbb{Z}^{d_2} \mid \exists \mathbf{z} \in \mathbb{Z}^e : A_1\mathbf{x}_1 + A_2\mathbf{x}_2 + B\mathbf{s} + D\mathbf{z} + \mathbf{c} \geq \mathbf{0} \,\},$$

with $A_i \in \mathbb{Z}^{m \times d_i}$, $B \in \mathbb{Z}^{m \times n}$, $D \in \mathbb{Z}^{m \times e}$ and $\mathbf{c} \in \mathbb{Z}^m$.

Definition 2.1.4 (Parameter Domain of a Relation) *Let $R \in \mathbb{Z}^n \to 2^{\mathbb{Z}^{d+d}}$ be a relation. The* parameter domain *of R is the set*

$$\operatorname{pdom} R \coloneqq \{\, \mathbf{s} \in \mathbb{Z}^n \mid R(\mathbf{s}) \neq \emptyset \,\}.$$

Definition 2.1.5 (Domain of a Relation) *Let $R \in \mathbb{Z}^n \to 2^{\mathbb{Z}^{d+d}}$ be a relation. The* domain *of R is the polyhedral set*

$$\operatorname{dom} R \coloneqq \mathbf{s} \mapsto \{\, \mathbf{x}_1 \in \mathbb{Z}^{d_1} \mid \exists \mathbf{x}_2 \in \mathbb{Z}^{d_2} : (\mathbf{x}_1, \mathbf{x}_2) \in R(\mathbf{s}) \,\}.$$

Definition 2.1.6 (Range of a Relation) *Let $R \in \mathbb{Z}^n \to 2^{\mathbb{Z}^{d+d}}$ be a relation. The* range *of R is the polyhedral set*

$$\operatorname{ran} R \coloneqq \mathbf{s} \mapsto \{\, \mathbf{x}_2 \in \mathbb{Z}^{d_2} \mid \exists \mathbf{x}_1 \in \mathbb{Z}^{d_1} : (\mathbf{x}_1, \mathbf{x}_2) \in R(\mathbf{s}) \,\}.$$

Definition 2.1.7 (Composition of Relations) *Let $R \in \mathbb{Z}^n \to 2^{\mathbb{Z}^{d_1+d_2}}$ and $S \in \mathbb{Z}^n \to 2^{\mathbb{Z}^{d_2+d_3}}$ be two relations, then the composition of R and S is defined as*

$$S \circ R \coloneqq \mathbf{s} \mapsto \{\, \mathbf{x}_1 \to \mathbf{x}_3 \in \mathbb{Z}^{d_1} \times \mathbb{Z}^{d_3} \mid \exists \mathbf{x}_2 \in \mathbb{Z}^{d_2} : \mathbf{x}_1 \to \mathbf{x}_2 \in R(\mathbf{s}) \wedge \mathbf{x}_2 \to \mathbf{x}_3 \in S(\mathbf{s}) \,\}.$$

Definition 2.1.8 (Difference Set of a Relation) *Let $R \in \mathbb{Z}^n \to 2^{\mathbb{Z}^{d+d}}$ be a relation. The difference set (ΔR) of R is the set of differences between image elements and the corresponding domain elements,*

$$\Delta R \coloneqq \mathbf{s} \mapsto \{\, \boldsymbol{\delta} \in \mathbb{Z}^d \mid \exists \mathbf{x} \to \mathbf{y} \in R : \boldsymbol{\delta} = \mathbf{y} - \mathbf{x} \,\}$$

2.2 Simple Hull

It is sometimes useful to have a single basic set or basic relation that contains a given set or relation. For rational sets, the obvious choice would be to compute the (rational) convex hull. For integer sets, the obvious choice would be the integer hull. However, `isl` currently does not support an integer hull operation and even if it did, it would be fairly expensive to compute. The convex hull operation is supported, but it is also fairly expensive to compute given only an implicit representation.

Usually, it is not required to compute the exact integer hull, and an overapproximation of this hull is sufficient. The "simple hull" of a set is such an overapproximation and it is defined as the (inclusion-wise) smallest basic set that is described by constraints that are translates of the constraints in the input set. This means that the simple hull is relatively cheap to compute and that the number of constraints in the simple hull is no larger than the number of constraints in the input.

Definition 2.2.1 (Simple Hull of a Set) *The* simple hull *of a set $S = \bigcup_{1 \le i \le v} S_i$, with*

$$S : \mathbb{Z}^n \to 2^{\mathbb{Z}^d} : \mathbf{s} \mapsto S(\mathbf{s}) = \left\{ \mathbf{x} \in \mathbb{Z}^d \mid \exists \mathbf{z} \in \mathbb{Z}^e : \bigvee_{1 \le i \le v} A_i\mathbf{x} + B_i\mathbf{s} + D_i\mathbf{z} + \mathbf{c}_i \ge \mathbf{0} \right\}$$

is the set

$$H : \mathbb{Z}^n \to 2^{\mathbb{Z}^d} : \mathbf{s} \mapsto S(\mathbf{s}) = \left\{ \mathbf{x} \in \mathbb{Z}^d \mid \exists \mathbf{z} \in \mathbb{Z}^e : \bigwedge_{1 \le i \le v} A_i\mathbf{x} + B_i\mathbf{s} + D_i\mathbf{z} + \mathbf{c}_i + \mathbf{K}_i \ge \mathbf{0} \right\},$$

with $\mathbf{K}_i$ the (component-wise) smallest non-negative integer vectors such that $S \subseteq H$.

The $\mathbf{K}_i$ can be obtained by solving a number of LP problems, one for each element of each $\mathbf{K}_i$. If any LP problem is unbounded, then the corresponding constraint is dropped.

2.3 Parametric Integer Programming

2.3.1 Introduction

Parametric integer programming (P. Feautrier 1988) is used to solve many problems within the context of the polyhedral model. Here, we are mainly interested in dependence analysis (P. Feautrier 1991) and in computing a unique representation for

existentially quantified variables. The latter operation has been used for counting elements in sets involving such variables (Boulet and Redon 1998; Verdoolaege, Beyls, et al. 2005) and lies at the core of the internal representation of `isl`.

Parametric integer programming was first implemented in `PipLib`. An alternative method for parametric integer programming was later implemented in `barvinok` Verdoolaege 2006. This method is not based on Feautrier's algorithm, but on rational generating functions Barvinok and Woods 2003 and was inspired by the "digging" technique of De Loera, Haws, et al. (2004) for solving non-parametric integer programming problems.

In the following sections, we briefly recall the dual simplex method combined with Gomory cuts and describe some extensions and optimizations. The main algorithm is applied to a matrix data structure known as a tableau. In case of parametric problems, there are two tableaus, one for the main problem and one for the constraints on the parameters, known as the context tableau. The handling of the context tableau is described in Section 2.3.7.

2.3.2 The Dual Simplex Method

Tableaus can be represented in several slightly different ways. In `isl`, the dual simplex method uses the same representation as that used by its incremental LP solver based on the *primal* simplex method. The implementation of this LP solver is based on that of `Simplify` (Detlefs, G. Nelson, et al. 2005), which, in turn, was derived from the work of C. G. Nelson (1980). In the original (C. G. Nelson 1980), the tableau was implemented as a sparse matrix, but neither `Simplify` nor the current implementation of `isl` does so.

Given some affine constraints on the variables, $A\mathbf{x} + \mathbf{b} \geq \mathbf{0}$, the tableau represents the relationship between the variables $\mathbf{x}$ and non-negative variables $\mathbf{y} = A\mathbf{x} + \mathbf{b}$ corresponding to the constraints. The initial tableau contains $\begin{pmatrix}\mathbf{b} & A\end{pmatrix}$ and expresses the constraints $\mathbf{y}$ in the rows in terms of the variables $\mathbf{x}$ in the columns. The main operation defined on a tableau exchanges a column and a row variable and is called a pivot. During this process, some coefficients may become rational. As in the `PipLib` implementation, `isl` maintains a shared denominator per row. The sample value of a tableau is one where each column variable is assigned zero and each row variable is assigned the constant term of the row. This sample value represents a valid solution if each constraint variable is assigned a non-negative value, i.e., if the constant terms of rows corresponding to constraints are all non-negative.

The dual simplex method starts from an initial sample value that may be invalid, but that is known to be (lexicographically) no greater than any solution, and gradually increments this sample value through pivoting until a valid solution is obtained. In particular, each pivot exchanges a row variable $r = -n + \sum_i a_i c_i$ with negative sample value $-n$ with a column variable c_j such that $a_j > 0$. Since $c_j = (n + r - \sum_{i \neq j} a_i c_i)/a_j$, the new row variable will have a positive sample value n. If no such column can be found, then the problem is infeasible. By always choosing the column that leads to the (lexicographically) smallest increment in the variables $\mathbf{x}$, the first solution found is guaranteed to be the (lexicographically) minimal solution P. Feautrier 1988. In order to be able to determine the smallest increment, the tableau is (implicitly) extended

with extra rows defining the original variables in terms of the column variables. If we assume that all variables are non-negative, then we know that the zero vector is no greater than the minimal solution and then the initial extended tableau looks as follows.

$$\begin{array}{c} \\ \mathbf{x} \\ \mathbf{r} \end{array} \begin{array}{c} \begin{array}{cc} 1 & \mathbf{c} \end{array} \\ \left(\begin{array}{cc} \mathbf{0} & I \\ \mathbf{b} & A \end{array} \right) \end{array}$$

Each column in this extended tableau is lexicographically positive and will remain so because of the column choice explained above. It is then clear that the value of $\mathbf{x}$ will increase in each step. Note that there is no need to store the extra rows explicitly. If a given x_i is a column variable, then the corresponding row is the unit vector e_i. If, on the other hand, it is a row variable, then the row already appears somewhere else in the tableau.

In case of parametric problems, the sign of the constant term may depend on the parameters. Each time the constant term of a constraint row changes, we therefore need to check whether the new term can attain negative and/or positive values over the current set of possible parameter values, i.e., the context. If all these terms can only attain non-negative values, the current state of the tableau represents a solution. If one of the terms can only attain non-positive values and is not identically zero, the corresponding row can be pivoted. Otherwise, we pick one of the terms that can attain both positive and negative values and split the context into a part where it only attains non-negative values and a part where it only attains negative values.

2.3.3 Gomory Cuts

The solution found by the dual simplex method may have non-integral coordinates. If so, some rational solutions (including the current sample value), can be cut off by applying a (parametric) Gomory cut. Let $r = b(\mathbf{p}) + \langle \mathbf{a}, \mathbf{c} \rangle$ be the row corresponding to the first non-integral coordinate of $\mathbf{x}$, with $b(\mathbf{p})$ the constant term, an affine expression in the parameters $\mathbf{p}$, i.e., $b(\mathbf{p}) = \langle \mathbf{f}, \mathbf{p} \rangle + g$. Note that only row variables can attain non-integral values as the sample value of the column variables is zero. Consider the expression $b(\mathbf{p}) - \lceil b(\mathbf{p}) \rceil + \langle \{\mathbf{a}\}, \mathbf{c} \rangle$, with $\lceil \cdot \rceil$ the ceiling function and $\{\cdot\}$ the fractional part. This expression is negative at the sample value since $\mathbf{c} = \mathbf{0}$ and $r = b(\mathbf{p})$ is fractional, i.e., $\lceil b(\mathbf{p}) \rceil > b(\mathbf{p})$. On the other hand, for each integral value of r and $\mathbf{c} \geq 0$, the expression is non-negative because $b(\mathbf{p}) - \lceil b(\mathbf{p}) \rceil > -1$. Imposing this expression to be non-negative therefore does not invalidate any integral solutions, while it does cut away the current fractional sample value. To be able to formulate this constraint, a new variable $q = \lfloor -b(\mathbf{p}) \rfloor = -\lceil b(\mathbf{p}) \rceil$ is added to the context. This integral variable is uniquely defined by the constraints $0 \leq -d\, b(\mathbf{p}) - d\, q \leq d - 1$, with d the common denominator of $\mathbf{f}$ and g. In practice, the variable $q' = \lfloor \langle \{-f\}, \mathbf{p} \rangle + \{-g\} \rfloor$ is used instead and the coefficients of the new constraint are adjusted accordingly. The sign of the constant term of this new constraint need not be determined as it is non-positive by construction. When several of these extra context variables are added, it is important to avoid adding duplicates. Recent versions of `PipLib` also check for such duplicates.

2.3.4 Negative Unknowns and Maximization

There are two places in the above algorithm where the unknowns $\mathbf{x}$ are assumed to be non-negative: the initial tableau starts from sample value $\mathbf{x} = \mathbf{0}$ and $\mathbf{c}$ is assumed to be non-negative during the construction of Gomory cuts. To deal with negative unknowns, P. Feautrier (1991, Appendix A.2) proposed to use a "big parameter", say M, that is taken to be an arbitrarily large positive number. Instead of looking for the lexicographically minimal value of $\mathbf{x}$, we search instead for the lexicographically minimal value of $\mathbf{x}' = \mathbf{M} + \mathbf{x}$. The sample value $\mathbf{x}' = \mathbf{0}$ of the initial tableau then corresponds to $\mathbf{x} = -\mathbf{M}$, which is clearly not greater than any potential solution. The sign of the constant term of a row is determined lexicographically, with the coefficient of M considered first. That is, if the coefficient of M is not zero, then its sign is the sign of the entire term. Otherwise, the sign is determined by the remaining affine expression in the parameters. If the original problem has a bounded optimum, then the final sample value will be of the form $\mathbf{M} + \mathbf{v}$ and the optimal value of the original problem is then $\mathbf{v}$. Maximization problems can be handled in a similar way by computing the minimum of $\mathbf{M} - \mathbf{x}$.

When the optimum is unbounded, the optimal value computed for the original problem will involve the big parameter. In the original implementation of `PipLib`, the big parameter could even appear in some of the extra variables $\mathbf{q}$ created during the application of a Gomory cut. The final result could then contain implicit conditions on the big parameter through conditions on such $\mathbf{q}$ variables. This problem was resolved in later versions of `PipLib` by taking M to be divisible by any positive number. The big parameter can then never appear in any $\mathbf{q}$ because $\{\alpha M\} = 0$. It should be noted, though, that an unbounded problem usually (but not always) indicates an incorrect formulation of the problem.

The original version of `PipLib` required the user to "manually" add a big parameter, perform the reformulation and interpret the result (P. Feautrier, Collard, et al. 2002). Recent versions allow the user to simply specify that the unknowns may be negative or that the maximum should be computed and then these transformations are performed internally. Although there are some application, e.g., that of Paul Feautrier (1992), where it is useful to have explicit control over the big parameter, negative unknowns and maximization are by far the most common applications of the big parameter and we believe that the user should not be bothered with such implementation issues. The current version of `isl` therefore does not provide any interface for specifying big parameters. Instead, the user can specify whether a maximum needs to be computed and no assumptions are made on the sign of the unknowns. Instead, the sign of the unknowns is checked internally and a big parameter is automatically introduced when needed. For compatibility with `PipLib`, the `isl_pip` tool does explicitly add non-negativity constraints on the unknowns unless the `Urs_unknowns` option is specified. Currently, there is also no way in `isl` of expressing a big parameter in the output. Even though `isl` makes the same divisibility assumption on the big parameter as recent versions of `PipLib`, it will therefore eventually produce an error if the problem turns out to be unbounded.

2.3.5 Preprocessing

In this section, we describe some transformations that are or can be applied in advance to reduce the running time of the actual dual simplex method with Gomory cuts.

Feasibility Check and Detection of Equalities

Experience with the original `PipLib` has shown that Gomory cuts do not perform very well on problems that are (non-obviously) empty, i.e., problems with rational solutions, but no integer solutions. In `isl`, we therefore first perform a feasibility check on the original problem considered as a non-parametric problem over the combined space of unknowns and parameters. In fact, we do not simply check the feasibility, but we also check for implicit equalities among the integer points by computing the integer affine hull. The algorithm used is the same as that described in Section 2.3.7 below. Computing the affine hull is fairly expensive, but it can bring huge benefits if any equalities can be found or if the problem turns out to be empty.

Constraint Simplification

If the coefficients of the unknown and parameters in a constraint have a common factor, then this factor should be removed, possibly rounding down the constant term. For example, the constraint $2x - 5 \geq 0$ should be simplified to $x - 3 \geq 0$. `isl` performs such simplifications on all sets and relations. Recent versions of `PipLib` also perform this simplification on the input.

Exploiting Equalities

If there are any (explicit) equalities in the input description, `PipLib` converts each into a pair of inequalities. It is also possible to write r equalities as $r + 1$ inequalities (P. Feautrier, Collard, et al. 2002), but it is even better to *exploit* the equalities to reduce the dimensionality of the problem. Given an equality involving at least one unknown, we pivot the row corresponding to the equality with the column corresponding to the last unknown with non-zero coefficient. The new column variable can then be removed completely because it is identically zero, thereby reducing the dimensionality of the problem by one. The last unknown is chosen to ensure that the columns of the initial tableau remain lexicographically positive. In particular, if the equality is of the form $b + \sum_{i \leq j} a_i x_i = 0$ with $a_j \neq 0$, then the (implicit) top rows of the initial tableau are changed as follows

$$\begin{array}{c} \\ j \end{array} \begin{array}{c} \begin{array}{c} \text{j} \end{array} \\ \left(\begin{array}{ccc} 0 & I_1 & \\ 0 & 1 & \\ 0 & & I_2 \end{array} \right) \end{array} \rightsquigarrow \begin{array}{c} \\ j \end{array} \begin{array}{c} \\ \left(\begin{array}{ccc} 0 & I_1 & \\ -b/a_j & -a_i/a_j & \\ 0 & & I_2 \end{array} \right) \end{array}$$

Currently, `isl` also eliminates equalities involving only parameters in a similar way, provided at least one of the coefficients is equal to one. The application of parameter compression (see below) would obviate the need for removing parametric equalities.

Offline Symmetry Detection

Some problems, notably those of Bygde (2010), have a collection of constraints, say $b_i(\mathbf{p}) + \langle \mathbf{a}, \mathbf{x} \rangle \geq 0$, that only differ in their (parametric) constant terms. These constant terms will be non-negative on different parts of the context and this context may have to be split for each of the constraints. In the worst case, the basic algorithm may have to consider all possible orderings of the constant terms. Instead, `isl` introduces a new parameter, say u, and replaces the collection of constraints by the single constraint $u + \langle \mathbf{a}, \mathbf{x} \rangle \geq 0$ along with context constraints $u \leq b_i(\mathbf{p})$. Any solution to the new system is also a solution to the original system since $\langle \mathbf{a}, \mathbf{x} \rangle \geq -u \geq -b_i(\mathbf{p})$. Conversely, $m = \min_i b_i(\mathbf{p})$ satisfies the constraints on u and therefore extends a solution to the new system. It can also be plugged into a new solution. See Section 2.3.6 for how this substitution is currently performed in `isl`. The method described in this section can only detect symmetries that are explicitly available in the input. See Section 2.3.9 for the detection and exploitation of symmetries that appear during the course of the dual simplex method.

Parameter Compression

It may in some cases be apparent from the equalities in the problem description that there can only be a solution for a sublattice of the parameters. In such cases "parameter compression" (Meister 2004; Meister and Verdoolaege 2008) can be used to replace the parameters by alternative "dense" parameters. For example, if there is a constraint $2x = n$, then the system will only have solutions for even values of n and n can be replaced by $2n'$. Similarly, the parameters n and m in a system with the constraint $2n = 3m$ can be replaced by a single parameter n' with $n = 3n'$ and $m = 2n'$. It is also possible to perform a similar compression on the unknowns, but it would be more complicated as the compression would have to preserve the lexicographical order. Moreover, due to our handling of equalities described above there should be no need for such variable compression. Although parameter compression has been implemented in `isl`, it is currently not yet used during parametric integer programming.

2.3.6 Postprocessing

The output of `PipLib` is a quast (quasi-affine selection tree). Each internal node in this tree corresponds to a split of the context based on a parametric constant term in the main tableau with indeterminate sign. Each of these nodes may introduce extra variables in the context corresponding to integer divisions. Each leaf of the tree prescribes the solution in that part of the context that satisfies all the conditions on the path leading to the leaf. Such a quast is a very economical way of representing the solution, but it would not be suitable as the (only) internal representation of sets and relations in `isl`. Instead, `isl` represents the constraints of a set or relation in disjunctive normal form. The result of a parametric integer programming problem is then also converted to this internal representation. Unfortunately, the conversion to disjunctive normal form can lead to an explosion of the size of the representation. In some cases, this overhead would have to be paid anyway in subsequent operations, but in other cases, especially

for outside users that just want to solve parametric integer programming problems, we would like to avoid this overhead in future. That is, we are planning on introducing quasts or a related representation as one of several possible internal representations and on allowing the output of `isl_pip` to optionally be printed as a quast.

Currently, `isl` also does not have an internal representation for expressions such as $\min_i b_i(\mathbf{p})$ from the offline symmetry detection of Section 2.3.5. Assume that one of these expressions has n bounds $b_i(\mathbf{p})$. If the expression does not appear in the affine expression describing the solution, but only in the constraints, and if moreover, the expression only appears with a positive coefficient, i.e., $\min_i b_i(\mathbf{p}) \geq f_j(\mathbf{p})$, then each of these constraints can simply be reduplicated n times, once for each of the bounds. Otherwise, a conversion to disjunctive normal form leads to n cases, each described as $u = b_i(\mathbf{p})$ with constraints $b_i(\mathbf{p}) \leq b_j(\mathbf{p})$ for $j > i$ and $b_i(\mathbf{p}) < b_j(\mathbf{p})$ for $j < i$. Note that even though this conversion leads to a size increase by a factor of n, not detecting the symmetry could lead to an increase by a factor of $n!$ if all possible orderings end up being considered.

2.3.7 Context Tableau

The main operation that a context tableau needs to provide is a test on the sign of an affine expression over the elements of the context. This sign can be determined by solving two integer linear feasibility problems, one with a constraint added to the context that enforces the expression to be non-negative and one where the expression is negative. As already mentioned by P. Feautrier (1988), any integer linear feasibility solver could be used, but the `PipLib` implementation uses a recursive call to the dual simplex with Gomory cuts algorithm to determine the feasibility of a context. In `isl`, two ways of handling the context have been implemented, one that performs the recursive call and one, used by default, that uses generalized basis reduction. We start with some optimizations that are shared between the two implementations and then discuss additional details of each of them.

Maintaining Witnesses

A common feature of both integer linear feasibility solvers is that they will not only say whether a set is empty or not, but if the set is non-empty, they will also provide a *witness* for this result, i.e., a point that belongs to the set. By maintaining a list of such witnesses, we can avoid many feasibility tests during the determination of the signs of affine expressions. In particular, if the expression evaluates to a positive number on some of these points and to a negative number on some others, then no feasibility test needs to be performed. If all the evaluations are non-negative, we only need to check for the possibility of a negative value and similarly in case of all non-positive evaluations. Finally, in the rare case that all points evaluate to zero or at the start, when no points have been collected yet, one or two feasibility tests need to be performed depending on the result of the first test.

When a new constraint is added to the context, the points that violate the constraint are temporarily removed. They are reconsidered when we backtrack over the addition of the constraint, as they will satisfy the negation of the constraint. It is only when

we backtrack over the addition of the points that they are finally removed completely. When an extra integer division is added to the context, the new coordinates of the witnesses can easily be computed by evaluating the integer division. The idea of keeping track of witnesses was first used in `barvinok`.

Choice of Constant Term on which to Split

Recall that if there are no rows with a non-positive constant term, but there are rows with an indeterminate sign, then the context needs to be split along the constant term of one of these rows. If there is more than one such row, then we need to choose which row to split on first. `PipLib` uses a heuristic based on the (absolute) sizes of the coefficients. In particular, it takes the largest coefficient of each row and then selects the row where this largest coefficient is smaller than those of the other rows.

In `isl`, we take that row for which non-negativity of its constant term implies non-negativity of as many of the constant terms of the other rows as possible. The intuition behind this heuristic is that on the positive side, we will have fewer negative and indeterminate signs, while on the negative side, we need to perform a pivot, which may affect any number of rows meaning that the effect on the signs is difficult to predict. This heuristic is of course much more expensive to evaluate than the heuristic used by `PipLib`. More extensive tests are needed to evaluate whether the heuristic is worthwhile.

Dual Simplex + Gomory Cuts

When a new constraint is added to the context, the first steps of the dual simplex method applied to this new context will be the same or at least very similar to those taken on the original context, i.e., before the constraint was added. In `isl`, we therefore apply the dual simplex method incrementally on the context and backtrack to a previous state when a constraint is removed again. An initial implementation that was never made public would also keep the Gomory cuts, but the current implementation backtracks to before the point where Gomory cuts are added before adding an extra constraint to the context. Keeping the Gomory cuts has the advantage that the sample value is always an integer point and that this point may also satisfy the new constraint. However, due to the technique of maintaining witnesses explained above, we would not perform a feasibility test in such cases and then the previously added cuts may be redundant, possibly resulting in an accumulation of a large number of cuts.

If the parameters may be negative, then the same big parameter trick used in the main tableau is applied to the context. This big parameter is of course unrelated to the big parameter from the main tableau. Note that it is not a requirement for this parameter to be "big", but it does allow for some code reuse in `isl`. In `PipLib`, the extra parameter is not "big", but this may be because the big parameter of the main tableau also appears in the context tableau.

Finally, it was reported by Galea (2009), who worked on a parametric integer programming implementation in `PPL` (Bagnara, Hill, et al. n.d.), that it is beneficial to add cuts for *all* rational coordinates in the context tableau. Based on this report, the initial `isl` implementation was adapted accordingly.

Generalized Basis Reduction

The default algorithm used in `isl` for feasibility checking is generalized basis reduction (Cook, Rutherford, et al. 1991). This algorithm is also used in the `barvinok` implementation. The algorithm is fairly robust, but it has some overhead. We therefore try to avoid calling the algorithm in easy cases. In particular, we incrementally keep track of points for which the entire unit hypercube positioned at that point lies in the context. This set is described by translates of the constraints of the context and if (rationally) non-empty, any rational point in the set can be rounded up to yield an integer point in the context.

A restriction of the algorithm is that it only works on bounded sets. The affine hull of the recession cone therefore needs to be projected out first. As soon as the algorithm is invoked, we then also incrementally keep track of this recession cone. The reduced basis found by one call of the algorithm is also reused as initial basis for the next call.

Some problems lead to the introduction of many integer divisions. Within a given context, some of these integer divisions may be equal to each other, even if the expressions are not identical, or they may be equal to some affine combination of other variables. To detect such cases, we compute the affine hull of the context each time a new integer division is added. The algorithm used for computing this affine hull is that of Karr (1976), while the points used in this algorithm are obtained by performing integer feasibility checks on that part of the context outside the current approximation of the affine hull. The list of witnesses is used to construct an initial approximation of the hull, while any extra points found during the construction of the hull is added to this list. Any equality found in this way that expresses an integer division as an *integer* affine combination of other variables is propagated to the main tableau, where it is used to eliminate that integer division.

2.3.8 Experiments

Table 2.1 compares the execution times of `isl` (with both types of context tableau) on some more difficult instances to those of other tools, run on an Intel Xeon W3520 @ 2.66GHz. These instances are available in the `testsets/pip` directory of the `isl` distribution. Easier problems such as the test cases distributed with `PipLib` can be solved so quickly that we would only be measuring overhead such as input/output and conversions and not the running time of the actual algorithm. We compare the following versions: `piplib-1.4.0-5-g0132fd9`, `barvinok-0.32.1-73-gc5d7751`, `isl-0.05.1-82-g3a37260` and `PPL` version 0.11.2.

The first test case is the following dependence analysis problem originating from the Phideo project (Verhaegh 1995) that was communicated to us by Bart Kienhuis:

```
lexmax { [j1,j2] -> [i1,i2,i3,i4,i5,i6,i7,i8,i9,i10] : 1 <= i1,j1
     <= 8 and 1 <= i2,i3,i4,i5,i6,i7,i8,i9,i10 <= 2 and 1 <= j2
    <= 128 and i1-1 = j1-1 and i2-1+2*i3-2+4*i4-4+8*i5-8+16*i6
    -16+32*i7-32+64*i8-64+128*i9-128+256*i10-256=3*j2-3+66 };
```

This problem was the main inspiration for some of the optimizations in Section 2.3.7. The second group of test cases are projections used during counting. The first nine

	PipLib	barvinok	isl cut	isl gbr	PPL
Phideo	TC	793m	>999m	2.7s	372m
e1	0.33s	3.5s	0.08s	0.11s	0.18s
e3	0.14s	0.13s	0.10s	0.10s	0.17s
e4	0.24s	9.1s	0.09s	0.11s	0.70s
e5	0.12s	6.0s	0.06s	0.14s	0.17s
e6	0.10s	6.8s	0.17s	0.08s	0.21s
e7	0.03s	0.27s	0.04s	0.04s	0.03s
e8	0.03s	0.18s	0.03s	0.04s	0.01s
e9	OOM	70m	2.6s	0.94s	22s
vd	0.04s	0.10s	0.03s	0.03s	0.03s
bouleti	0.25s	line	0.06s	0.06s	0.15s
difficult	OOM	1.3s	1.7s	0.33s	1.4s
cnt/sum	TC	max	2.2s	2.2s	OOM
jcomplex	TC	max	3.7s	3.9s	OOM

Table 2.1: Comparison of Execution Times

of these come from Seghir and Loechner (2006). The remaining two come from Verdoolaege, Beyls, et al. (2005) and were used to drive the first, Gomory cuts based, implementation in `isl`. The third and final group of test cases are borrowed from Bygde (2010) and inspired the offline symmetry detection of Section 2.3.5. Without symmetry detection, the running times are 11s and 5.9s. All running times of `barvinok` and `isl` include a conversion to disjunctive normal form. Without this conversion, the final two cases can be solved in 0.07s and 0.21s. The `PipLib` implementation has some fixed limits and will sometimes report the problem to be too complex (TC), while on some other problems it will run out of memory (OOM). The `barvinok` implementation does not support problems with a non-trivial lineality space (line) nor maximization problems (max). The Gomory cuts based `isl` implementation was terminated after 1000 minutes on the first problem. The gbr version introduces some overhead on some of the easier problems, but is overall the clear winner.

2.3.9 Online Symmetry Detection

Manual experiments on small instances of the problems of Bygde (2010) and an analysis of the results by the approximate MPA method developed by Bygde (2010) have revealed that these problems contain many more symmetries than can be detected using the offline method of Section 2.3.5. In this section, we present an online detection mechanism that has not been implemented yet, but that has shown promising results in manual applications.

Let us first consider what happens when we do not perform offline symmetry detection. At some point, one of the $b_i(\mathbf{p}) + \langle \mathbf{a}, \mathbf{x} \rangle \geq 0$ constraints, say the jth constraint, appears as a column variable, say c_1, while the other constraints are represented as rows of the form $b_i(\mathbf{p}) - b_j(\mathbf{p}) + c$. The context is then split according to the relative order

of $b_j(\mathbf{p})$ and one of the remaining $b_i(\mathbf{p})$. The offline method avoids this split by replacing all $b_i(\mathbf{p})$ by a single newly introduced parameter that represents the minimum of these $b_i(\mathbf{p})$. In the online method the split is similarly avoided by the introduction of a new parameter. In particular, a new parameter is introduced that represents $\left|b_j(\mathbf{p}) - b_i(\mathbf{p})\right|_+ = \max(b_j(\mathbf{p}) - b_i(\mathbf{p}), 0)$.

In general, let $r = b(\mathbf{p}) + \langle \mathbf{a}, \mathbf{c} \rangle$ be a row of the tableau such that the sign of $b(\mathbf{p})$ is indeterminate and such that exactly one of the elements of $\mathbf{a}$ is a 1, while all remaining elements are non-positive. That is, $r = b(\mathbf{p}) + c_j - f$ with $f = -\sum_{i \neq j} a_i c_i \geq 0$. We introduce a new parameter t with context constraints $t \geq -b(\mathbf{p})$ and $t \geq 0$ and replace the column variable c_j by $c' + t$. The row r is now equal to $b(\mathbf{p}) + t + c' - f$. The constant term of this row is always non-negative because any negative value of $b(\mathbf{p})$ is compensated by $t \geq -b(\mathbf{p})$ while and non-negative value remains non-negative because $t \geq 0$.

We need to show that this transformation does not eliminate any valid solutions and that it does not introduce any spurious solutions. Given a valid solution for the original problem, we need to find a non-negative value of c' satisfying the constraints. If $b(\mathbf{p}) \geq 0$, we can take $t = 0$ so that $c' = c_j - t = c_j \geq 0$. If $b(\mathbf{p}) < 0$, we can take $t = -b(\mathbf{p})$. Since $r = b(\mathbf{p}) + c_j - f \geq 0$ and $f \geq 0$, we have $c' = c_j + b(\mathbf{p}) \geq 0$. Note that these choices amount to plugging in $t = |-b(\mathbf{p})|_+ = \max(-b(\mathbf{p}), 0)$. Conversely, given a solution to the new problem, we need to find a non-negative value of c_j, but this is easy since $c_j = c' + t$ and both of these are non-negative.

Plugging in $t = \max(-b(\mathbf{p}), 0)$ can be performed as in Section 2.3.6, but, as in the case of offline symmetry detection, it may be better to provide a direct representation for such expressions in the internal representation of sets and relations or at least in a quast-like output format.

2.4 Coalescing

See Verdoolaege (2015) for details on integer set coalescing.

2.5 Transitive Closure

2.5.1 Introduction

Definition 2.5.1 (Power of a Relation) *Let $R \in \mathbb{Z}^n \to 2^{\mathbb{Z}^{d+d}}$ be a relation and $k \in \mathbb{Z}_{\geq 1}$ a positive number, then power k of relation R is defined as*

$$R^k := \begin{cases} R & \text{if } k = 1 \\ R \circ R^{k-1} & \text{if } k \geq 2. \end{cases} \tag{2.1}$$

Definition 2.5.2 (Transitive Closure of a Relation) *Let $R \in \mathbb{Z}^n \to 2^{\mathbb{Z}^{d+d}}$ be a relation, then the transitive closure R^+ of R is the union of all positive powers of R,*

$$R^+ := \bigcup_{k \geq 1} R^k.$$

Alternatively, the transitive closure may be defined inductively as

$$R^+ := R \cup (R \circ R^+)\,. \tag{2.2}$$

Since the transitive closure of a polyhedral relation may no longer be a polyhedral relation (Kelly, Pugh, et al. 1996), we can, in the general case, only compute an approximation of the transitive closure. Whereas Kelly, Pugh, et al. (1996) compute underapproximations, we, like Beletska, Barthou, et al. (2009), compute overapproximations. That is, given a relation R, we will compute a relation T such that $R^+ \subseteq T$. Of course, we want this approximation to be as close as possible to the actual transitive closure R^+ and we want to detect the cases where the approximation is exact, i.e., where $T = R^+$.

For computing an approximation of the transitive closure of R, we follow the same general strategy as Beletska, Barthou, et al. (2009) and first compute an approximation of R^k for $k \geq 1$ and then project out the parameter k from the resulting relation.

Example 2.5.3 *As a trivial example, consider the relation* $R = \{\, x \to x + 1 \,\}$. *The kth power of this map for arbitrary k is*

$$R^k = k \mapsto \{\, x \to x + k \mid k \geq 1 \,\}.$$

The transitive closure is then

$$\begin{aligned} R^+ &= \{\, x \to y \mid \exists k \in \mathbb{Z}_{\geq 1} : y = x + k \,\} \\ &= \{\, x \to y \mid y \geq x + 1 \,\}. \end{aligned}$$

2.5.2 Computing an Approximation of R^k

There are some special cases where the computation of R^k is very easy. One such case is that where R does not compose with itself, i.e., $R \circ R = \emptyset$ or $\operatorname{dom} R \cap \operatorname{ran} R = \emptyset$. In this case, R^k is only non-empty for $k = 1$ where it is equal to R itself.

In general, it is impossible to construct a closed form of R^k as a polyhedral relation. We will therefore need to make some approximations. As a first approximations, we will consider each of the basic relations in R as simply adding one or more offsets to a domain element to arrive at an image element and ignore the fact that some of these offsets may only be applied to some of the domain elements. That is, we will only consider the difference set ΔR of the relation. In particular, we will first construct a collection P of paths that move through a total of k offsets and then intersect domain and range of this collection with those of R. That is,

$$K = P \cap (\operatorname{dom} R \to \operatorname{ran} R)\,, \tag{2.3}$$

with

$$P = \mathbf{s} \mapsto \{\, \mathbf{x} \to \mathbf{y} \mid \exists k_i \in \mathbb{Z}_{\geq 0}, \boldsymbol{\delta}_i \in k_i\, \Delta_i(\mathbf{s}) : \mathbf{y} = \mathbf{x} + \sum_i \boldsymbol{\delta}_i \wedge \sum_i k_i = k > 0 \,\} \tag{2.4}$$

and with Δ_i the basic sets that compose the difference set ΔR. Note that the number of basic sets Δ_i need not be the same as the number of basic relations in R. Also note that

since addition is commutative, it does not matter in which order we add the offsets and so we are allowed to group them as we did in (2.4).

If all the Δ_is are singleton sets $\Delta_i = \{ \boldsymbol{\delta}_i \}$ with $\boldsymbol{\delta}_i \in \mathbb{Z}^d$, then (2.4) simplifies to

$$P = \{ \mathbf{x} \to \mathbf{y} \mid \exists k_i \in \mathbb{Z}_{\geq 0} : \mathbf{y} = \mathbf{x} + \sum_i k_i \boldsymbol{\delta}_i \wedge \sum_i k_i = k > 0 \} \tag{2.5}$$

and then the approximation computed in (2.3) is essentially the same as that of Beletska, Barthou, et al. (2009). If some of the Δ_is are not singleton sets or if some of $\boldsymbol{\delta}_i$s are parametric, then we need to resort to further approximations.

To ease both the exposition and the implementation, we will for the remainder of this section work with extended offsets $\Delta'_i = \Delta_i \times \{ 1 \}$. That is, each offset is extended with an extra coordinate that is set equal to one. The paths constructed by summing such extended offsets have the length encoded as the difference of their final coordinates. The path P' can then be decomposed into paths P'_i, one for each Δ_i,

$$P' = ((P'_m \cup \mathrm{Id}) \circ \cdots \circ (P'_2 \cup \mathrm{Id}) \circ (P'_1 \cup \mathrm{Id})) \cap \{ \mathbf{x}' \to \mathbf{y}' \mid y_{d+1} - x_{d+1} = k > 0 \}, \tag{2.6}$$

with

$$P'_i = \mathbf{s} \mapsto \{ \mathbf{x}' \to \mathbf{y}' \mid \exists k \in \mathbb{Z}_{\geq 1}, \boldsymbol{\delta} \in k \, \Delta'_i(\mathbf{s}) : \mathbf{y}' = \mathbf{x}' + \boldsymbol{\delta} \}.$$

Note that each P'_i contains paths of length at least one. We therefore need to take the union with the identity relation when composing the P'_is to allow for paths that do not contain any offsets from one or more Δ'_i. The path that consists of only identity relations is removed by imposing the constraint $y_{d+1} - x_{d+1} > 0$. Taking the union with the identity relation means that that the relations we compose in (2.6) each consist of two basic relations. If there are m disjuncts in the input relation, then a direct application of the composition operation may therefore result in a relation with 2^m disjuncts, which is prohibitively expensive. It is therefore crucial to apply coalescing (Section 2.4) after each composition.

Let us now consider how to compute an overapproximation of P'_i. Those that correspond to singleton Δ_is are grouped together and handled as in (2.5). Note that this is just an optimization. The procedure described below would produce results that are at least as accurate. For simplicity, we first assume that no constraint in Δ'_i involves any existentially quantified variables. We will return to existentially quantified variables at the end of this section. Without existentially quantified variables, we can classify the constraints of Δ'_i as follows

1. non-parametric constraints
$$A_1 \mathbf{x} + \mathbf{c}_1 \geq \mathbf{0} \tag{2.7}$$
2. purely parametric constraints
$$B_2 \mathbf{s} + \mathbf{c}_2 \geq \mathbf{0} \tag{2.8}$$
3. negative mixed constraints
$$A_3 \mathbf{x} + B_3 \mathbf{s} + \mathbf{c}_3 \geq \mathbf{0} \tag{2.9}$$
such that for each row j and for all $\mathbf{s}$,
$$\Delta'_i(\mathbf{s}) \cap \{ \boldsymbol{\delta}' \mid B_{3,j} \mathbf{s} + c_{3,j} > 0 \} = \emptyset$$

4. positive mixed constraints

$$A_4\mathbf{x} + B_4\mathbf{s} + \mathbf{c}_4 \geq \mathbf{0}$$

such that for each row j, there is at least one $\mathbf{s}$ such that

$$\Delta'_i(\mathbf{s}) \cap \{\, \boldsymbol{\delta}' \mid B_{4,j}\mathbf{s} + c_{4,j} > 0 \,\} \neq \emptyset$$

We will use the following approximation Q_i for P'_i:

$$\begin{aligned} Q_i = \mathbf{s} \mapsto \{\, \mathbf{x}' \to \mathbf{y}' \mid \exists k \in \mathbb{Z}_{\geq 1}, \mathbf{f} \in \mathbb{Z}^d : \mathbf{y}' = \mathbf{x}' + (\mathbf{f}, k) \wedge \\ A_1\mathbf{f} + k\mathbf{c}_1 \geq \mathbf{0} \wedge B_2\mathbf{s} + \mathbf{c}_2 \geq \mathbf{0} \wedge A_3\mathbf{f} + B_3\mathbf{s} + \mathbf{c}_3 \geq \mathbf{0} \,\}. \end{aligned} \tag{2.10}$$

To prove that Q_i is indeed an overapproximation of P'_i, we need to show that for every $\mathbf{s} \in \mathbb{Z}^n$, for every $k \in \mathbb{Z}_{\geq 1}$ and for every $\mathbf{f} \in k\,\Delta_i(\mathbf{s})$ we have that $(\mathbf{f}, k)$ satisfies the constraints in (2.10). If $\Delta_i(\mathbf{s})$ is non-empty, then $\mathbf{s}$ must satisfy the constraints in (2.8). Each element $(\mathbf{f}, k) \in k\,\Delta'_i(\mathbf{s})$ is a sum of k elements $(\mathbf{f}_j, 1)$ in $\Delta'_i(\mathbf{s})$. Each of these elements satisfies the constraints in (2.7), i.e.,

$$\begin{bmatrix} A_1 & \mathbf{c}_1 \end{bmatrix} \begin{bmatrix} \mathbf{f}_j \\ 1 \end{bmatrix} \geq \mathbf{0}.$$

The sum of these elements therefore satisfies the same set of inequalities, i.e., $A_1\mathbf{f} + k\mathbf{c}_1 \geq \mathbf{0}$. Finally, the constraints in (2.9) are such that for any $\mathbf{s}$ in the parameter domain of Δ, we have $-\mathbf{r}(\mathbf{s}) \coloneqq B_3\mathbf{s} + \mathbf{c}_3 \leq \mathbf{0}$, i.e., $A_3\mathbf{f}_j \geq \mathbf{r}(\mathbf{s}) \geq \mathbf{0}$ and therefore also $A_3\mathbf{f} \geq \mathbf{r}(\mathbf{s})$. Note that if there are no mixed constraints and if the rational relaxation of $\Delta_i(\mathbf{s})$, i.e., $\{\, \mathbf{x} \in \mathbb{Q}^d \mid A_1\mathbf{x} + \mathbf{c}_1 \geq \mathbf{0} \,\}$, has integer vertices, then the approximation is exact, i.e., $Q_i = P'_i$. In this case, the vertices of $\Delta'_i(\mathbf{s})$ generate the rational cone $\{\, \mathbf{x}' \in \mathbb{Q}^{d+1} \mid \begin{bmatrix} A_1 & \mathbf{c}_1 \end{bmatrix} \mathbf{x}' \,\}$ and therefore $\Delta'_i(\mathbf{s})$ is a Hilbert basis of this cone (Schrijver 1986, Theorem 16.4).

Note however that, as pointed out by De Smet (2010), if there *are* any mixed constraints, then the above procedure may not compute the most accurate affine approximation of $k\,\Delta_i(\mathbf{s})$ with $k \geq 1$. In particular, we only consider the negative mixed constraints that happen to appear in the description of $\Delta_i(\mathbf{s})$, while we should instead consider *all* valid such constraints. It is also sufficient to consider those constraints because any constraint that is valid for $k\,\Delta_i(\mathbf{s})$ is also valid for $1\,\Delta_i(\mathbf{s}) = \Delta_i(\mathbf{s})$. Take therefore any constraint $\langle \mathbf{a}, \mathbf{x} \rangle + \langle \mathbf{b}, \mathbf{s} \rangle + c \geq 0$ valid for $\Delta_i(\mathbf{s})$. This constraint is also valid for $k\,\Delta_i(\mathbf{s})$ iff $k\,\langle \mathbf{a}, \mathbf{x} \rangle + \langle \mathbf{b}, \mathbf{s} \rangle + c \geq 0$. If $\langle \mathbf{b}, \mathbf{s} \rangle + c$ can attain any positive value, then $\langle \mathbf{a}, \mathbf{x} \rangle$ may be negative for some elements of $\Delta_i(\mathbf{s})$. We then have $k\,\langle \mathbf{a}, \mathbf{x} \rangle < \langle \mathbf{a}, \mathbf{x} \rangle$ for $k > 1$ and so the constraint is not valid for $k\,\Delta_i(\mathbf{s})$. We therefore need to impose $\langle \mathbf{b}, \mathbf{s} \rangle + c \leq 0$ for all values of $\mathbf{s}$ such that $\Delta_i(\mathbf{s})$ is non-empty, i.e., $\mathbf{b}$ and c need to be such that $-\langle \mathbf{b}, \mathbf{s} \rangle - c \geq 0$ is a valid constraint of $\Delta_i(\mathbf{s})$. That is, $(\mathbf{b}, c)$ are the opposites of the coefficients of a valid constraint of $\Delta_i(\mathbf{s})$. The approximation of $k\,\Delta_i(\mathbf{s})$ can therefore be obtained using three applications of Farkas' lemma. The first obtains the coefficients of constraints valid for $\Delta_i(\mathbf{s})$. The second obtains the coefficients of constraints valid for the projection of $\Delta_i(\mathbf{s})$ onto the parameters. The opposite of the second set is then computed and intersected with the first set. The result is the set of coefficients of constraints valid for $k\,\Delta_i(\mathbf{s})$. A final application of Farkas' lemma is needed to obtain the approximation of $k\,\Delta_i(\mathbf{s})$ itself.

Example 2.5.4 *Consider the relation*

$$n \to \{ (x, y) \to (1 + x, 1 - n + y) \mid n \geq 2 \}.$$

The difference set of this relation is

$$\Delta = n \to \{ (1, 1 - n) \mid n \geq 2 \}.$$

Using our approach, we would only consider the mixed constraint $y - 1 + n \geq 0$*, leading to the following approximation of the transitive closure:*

$$n \to \{ (x, y) \to (o_0, o_1) \mid n \geq 2 \wedge o_1 \leq 1 - n + y \wedge o_0 \geq 1 + x \}.$$

If, instead, we apply Farkas's lemma to Δ*, i.e.,*

```
D := [n] -> { [1, 1 - n] : n >= 2 };
CD := coefficients D;
CD;
```

we obtain

```
{ rat: coefficients[[c_cst, c_n] -> [i2, i3]] : i3 <= c_n and
  i3 <= c_cst + 2c_n + i2 }
```

The pure-parametric constraints valid for Δ*,*

```
P := { [a,b] -> [] }(D);
CP := coefficients P;
CP;
```

are

```
{ rat: coefficients[[c_cst, c_n] -> []] : c_n >= 0 and 2c_n >= -c_cst }
```

Negating these coefficients and intersecting with `CD`*,*

```
NCP := { rat: coefficients[[a,b] -> []]
              -> coefficients[[-a,-b] -> []] }(CP);
CK := wrap((unwrap CD) * (dom (unwrap NCP)));
CK;
```

we obtain

```
{ rat: [[c_cst, c_n] -> [i2, i3]] : i3 <= c_n and
  i3 <= c_cst + 2c_n + i2 and c_n <= 0 and 2c_n <= -c_cst }
```

The approximation for $k\,\Delta$*,*

```
K := solutions CK;
K;
```

is then

```
[n] -> { rat: [i0, i1] : i1 <= -i0 and i0 >= 1 and i1 <= 2 - n - i0 }
```

Finally, the computed approximation for R^+,

```
T := unwrap({ [dx,dy] -> [[x,y] -> [x+dx,y+dy]] }(K));
R := [n] -> { [x,y] -> [x+1,y+1-n] : n >= 2 };
T := T * ((dom R) -> (ran R));
T;
```

is

```
[n] -> { [x, y] -> [o0, o1] : o1 <= x + y - o0 and
         o0 >= 1 + x and o1 <= 2 - n + x + y - o0 and n >= 2 }
```

Existentially quantified variables can be handled by classifying them into variables that are uniquely determined by the parameters, variables that are independent of the parameters and others. The first set can be treated as parameters and the second as variables. Constraints involving the other existentially quantified variables are removed.

Example 2.5.5 *Consider the relation*

$$R = n \to \{\, x \to y \mid \exists\, \alpha_0, \alpha_1 : 7\alpha_0 = -2 + n \wedge 5\alpha_1 = -1 - x + y \wedge y \geq 6 + x \,\}.$$

The difference set of this relation is

$$\Delta = \Delta R = n \to \{\, x \mid \exists\, \alpha_0, \alpha_1 : 7\alpha_0 = -2 + n \wedge 5\alpha_1 = -1 + x \wedge x \geq 6 \,\}.$$

The existentially quantified variables can be defined in terms of the parameters and variables as

$$\alpha_0 = \left\lfloor \frac{-2+n}{7} \right\rfloor \qquad \textit{and} \qquad \alpha_1 = \left\lfloor \frac{-1+x}{5} \right\rfloor.$$

α_0 can therefore be treated as a parameter, while α_1 can be treated as a variable. This in turn means that $7\alpha_0 = -2+n$ can be treated as a purely parametric constraint, while the other two constraints are non-parametric. The corresponding Q (2.10) is therefore

$$\begin{aligned} n \to \{\, (x,z) \to (y,w) \mid \exists\, \alpha_0, \alpha_1, k, f : k \geq 1 \wedge y = x + f \wedge w = z + k \wedge \\ 7\alpha_0 = -2 + n \wedge 5\alpha_1 = -k + x \wedge x \geq 6k \,\}. \end{aligned}$$

Projecting out the final coordinates encoding the length of the paths, results in the exact transitive closure

$$R^+ = n \to \{\, x \to y \mid \exists\, \alpha_0, \alpha_1 : 7\alpha_1 = -2 + n \wedge 6\alpha_0 \geq -x + y \wedge 5\alpha_0 \leq -1 - x + y \,\}.$$

The fact that we ignore some impure constraints clearly leads to a loss of accuracy. In some cases, some of this loss can be recovered by not considering the parameters in a special way. That is, instead of considering the set

$$\Delta = \Delta R = \mathbf{s} \mapsto \{\, \boldsymbol{\delta} \in \mathbb{Z}^d \mid \exists \mathbf{x} \to \mathbf{y} \in R : \boldsymbol{\delta} = \mathbf{y} - \mathbf{x} \,\}$$

we consider the set

$$\Delta' = \Delta R' = \{ \boldsymbol{\delta} \in \mathbb{Z}^{n+d} \mid \exists (\mathbf{s}, \mathbf{x}) \to (\mathbf{s}, \mathbf{y}) \in R' : \boldsymbol{\delta} = (\mathbf{s} - \mathbf{s}, \mathbf{y} - \mathbf{x}) \}.$$

The first n coordinates of every element in Δ' are zero. Projecting out these zero coordinates from Δ' is equivalent to projecting out the parameters in Δ. The result is obviously a superset of Δ, but all its constraints are of type (2.7) and they can therefore all be used in the construction of Q_i.

Example 2.5.6 *Consider the relation*

$$R = n \to \{ (x, y) \to (1 + x, 1 - n + y) \mid n \geq 2 \}.$$

We have

$$\Delta R = n \to \{ (1, 1 - n) \mid n \geq 2 \}$$

and so, by treating the parameters in a special way, we obtain the following approximation for R^+:

$$n \to \{ (x, y) \to (x', y') \mid n \geq 2 \wedge y' \leq 1 - n + y \wedge x' \geq 1 + x \}.$$

If we consider instead

$$R' = \{ (n, x, y) \to (n, 1 + x, 1 - n + y) \mid n \geq 2 \}$$

then

$$\Delta R' = \{ (0, 1, y) \mid y \leq -1 \}$$

and we obtain the approximation

$$n \to \{ (x, y) \to (x', y') \mid n \geq 2 \wedge x' \geq 1 + x \wedge y' \leq x + y - x' \}.$$

If we consider both ΔR and $\Delta R'$, then we obtain

$$n \to \{ (x, y) \to (x', y') \mid n \geq 2 \wedge y' \leq 1 - n + y \wedge x' \geq 1 + x \wedge y' \leq x + y - x' \}.$$

Note, however, that this is not the most accurate affine approximation that can be obtained. That would be

$$n \to \{ (x, y) \to (x', y') \mid y' \leq 2 - n + x + y - x' \wedge n \geq 2 \wedge x' \geq 1 + x \}.$$

2.5.3 Checking Exactness

The approximation T for the transitive closure R^+ can be obtained by projecting out the parameter k from the approximation K (2.3) of the power R^k. Since K is an overapproximation of R^k, T will also be an overapproximation of R^+. To check whether the results are exact, we need to consider two cases depending on whether R is *cyclic*, where R is defined to be cyclic if R^+ maps any element to itself, i.e., $R^+ \cap \mathrm{Id} \neq \emptyset$. If R is acyclic, then the inductive definition of (2.2) is equivalent to its completion, i.e.,

$$R^+ = R \cup (R \circ R^+)$$

is a defining property. Since T is known to be an overapproximation, we only need to check whether

$$T \subseteq R \cup (R \circ T)\,.$$

This is essentially Theorem 5 of Kelly, Pugh, et al. (1996). The only difference is that they only consider lexicographically forward relations, a special case of acyclic relations.

If, on the other hand, R is cyclic, then we have to resort to checking whether the approximation K of the power is exact. Note that T may be exact even if K is not exact, so the check is sound, but incomplete. To check exactness of the power, we simply need to check (2.1). Since again K is known to be an overapproximation, we only need to check whether

$$\begin{aligned} K'|_{y_{d+1}-x_{d+1}=1} &\subseteq R' \\ K'|_{y_{d+1}-x_{d+1}\geq 2} &\subseteq R' \circ K'|_{y_{d+1}-x_{d+1}\geq 1}, \end{aligned}$$

where $R' = \{\, \mathbf{x}' \to \mathbf{y}' \mid \mathbf{x} \to \mathbf{y} \in R \wedge y_{d+1} - x_{d+1} = 1 \,\}$, i.e., R extended with path lengths equal to 1.

All that remains is to explain how to check the cyclicity of R. Note that the exactness on the power is always sound, even in the acyclic case, so we only need to be careful that we find all cyclic cases. Now, if R is cyclic, i.e., $R^+ \cap \mathrm{Id} \neq \emptyset$, then, since T is an overapproximation of R^+, also $T \cap \mathrm{Id} \neq \emptyset$. This in turn means that $\Delta K'$ contains a point whose first d coordinates are zero and whose final coordinate is positive. In the implementation we currently perform this test on P' instead of K'. Note that if R^+ is acyclic and T is not, then the approximation is clearly not exact and the approximation of the power K will not be exact either.

2.5.4 Decomposing R into strongly connected components

If the input relation R is a union of several basic relations that can be partially ordered then the accuracy of the approximation may be improved by computing an approximation of each strongly connected components separately. For example, if $R = R_1 \cup R_2$ and $R_1 \circ R_2 = \emptyset$, then we know that any path that passes through R_2 cannot later pass through R_1, i.e.,

$$R^+ = R_1^+ \cup R_2^+ \cup (R_2^+ \circ R_1^+)\,. \tag{2.11}$$

We can therefore compute (approximations of) transitive closures of R_1 and R_2 separately. Note, however, that the condition $R_1 \circ R_2 = \emptyset$ is actually too strong. If $R_1 \circ R_2$ is a subset of $R_2 \circ R_1$ then we can reorder the segments in any path that moves through both R_1 and R_2 to first move through R_1 and then through R_2.

This idea can be generalized to relations that are unions of more than two basic relations by constructing the strongly connected components in the graph with as vertices the basic relations and an edge between two basic relations R_i and R_j if R_i needs to follow R_j in some paths. That is, there is an edge from R_i to R_j iff

$$R_i \circ R_j \not\subseteq R_j \circ R_i. \tag{2.12}$$

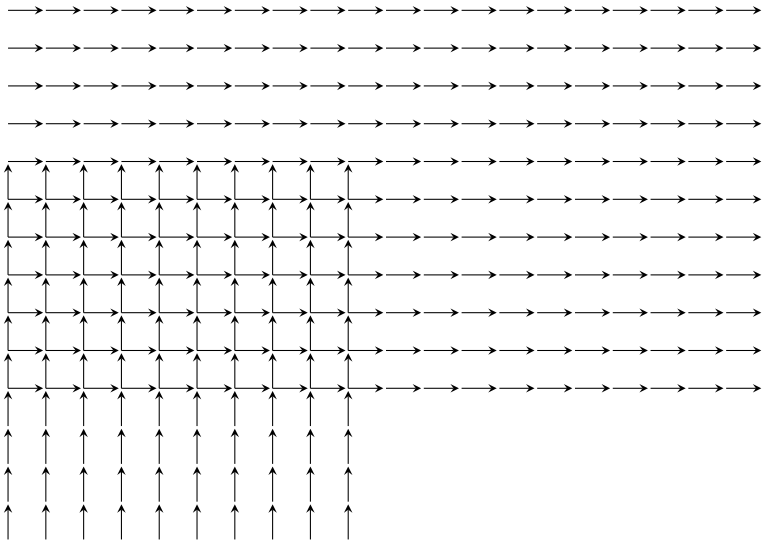

Figure 2.1: The relation from Example 2.5.7

The components can be obtained from the graph by applying Tarjan's algorithm (Tarjan 1972).

In practice, we compute the (extended) powers K_i' of each component separately and then compose them as in (2.6). Note, however, that in this case the order in which we apply them is important and should correspond to a topological ordering of the strongly connected components. Simply applying Tarjan's algorithm will produce topologically sorted strongly connected components. The graph on which Tarjan's algorithm is applied is constructed on-the-fly. That is, whenever the algorithm checks if there is an edge between two vertices, we evaluate (2.12). The exactness check is performed on each component separately. If the approximation turns out to be inexact for any of the components, then the entire result is marked inexact and the exactness check is skipped on the components that still need to be handled.

It should be noted that (2.11) is only valid for exact transitive closures. If overapproximations are computed in the right hand side, then the result will still be an overapproximation of the left hand side, but this result may not be transitively closed. If we only separate components based on the condition $R_i \circ R_j = \emptyset$, then there is no problem, as this condition will still hold on the computed approximations of the transitive closures. If, however, we have exploited (2.12) during the decomposition and if the result turns out not to be exact, then we check whether the result is transitively closed. If not, we recompute the transitive closure, skipping the decomposition. Note that testing for transitive closedness on the result may be fairly expensive, so we may want to make this check configurable.

Example 2.5.7 *Consider the relation in example* `closure4` *that comes with the Omega*

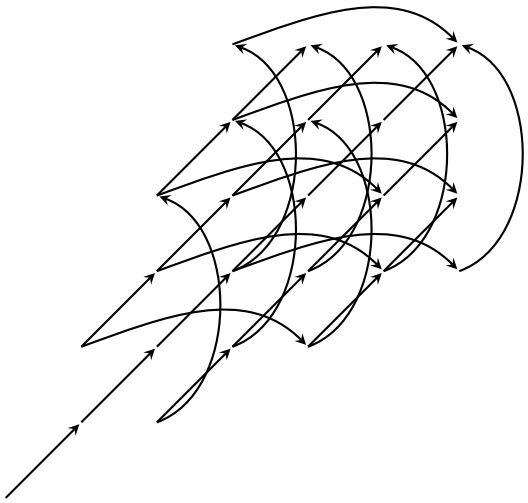

Figure 2.2: The relation from Example 2.5.8

calculator (Kelly, Maslov, et al. 1996a), $R = R_1 \cup R_2$, with

$$R_1 = \{\,(x,y) \to (x,y+1) \mid 1 \le x,y \le 10\,\}$$
$$R_2 = \{\,(x,y) \to (x+1,y) \mid 1 \le x \le 20 \wedge 5 \le y \le 15\,\}.$$

This relation is shown graphically in Figure 2.1. We have

$$R_1 \circ R_2 = \{\,(x,y) \to (x+1,y+1) \mid 1 \le x \le 9 \wedge 5 \le y \le 10\,\}$$
$$R_2 \circ R_1 = \{\,(x,y) \to (x+1,y+1) \mid 1 \le x \le 10 \wedge 4 \le y \le 10\,\}.$$

Clearly, $R_1 \circ R_2 \subseteq R_2 \circ R_1$ and so

$$(R_1 \cup R_2)^+ = \left(R_2^+ \circ R_1^+\right) \cup R_1^+ \cup R_2^+.$$

Example 2.5.8 *Consider the relation on the right of Beletska, Barthou, et al. (2009, Figure 2), reproduced in Figure 2.2. The relation can be described as $R = R_1 \cup R_2 \cup R_3$, with*

$$R_1 = n \mapsto \{\,(i,j) \to (i+3,j) \mid i \le 2j-4 \wedge i \le n-3 \wedge j \le 2i-1 \wedge j \le n\,\}$$
$$R_2 = n \mapsto \{\,(i,j) \to (i,j+3) \mid i \le 2j-1 \wedge i \le n \wedge j \le 2i-4 \wedge j \le n-3\,\}$$
$$R_3 = n \mapsto \{\,(i,j) \to (i+1,j+1) \mid i \le 2j-1 \wedge i \le n-1 \wedge j \le 2i-1 \wedge j \le n-1\,\}.$$

The figure shows this relation for $n = 7$. Both $R_3 \circ R_1 \subseteq R_1 \circ R_3$ and $R_3 \circ R_2 \subseteq R_2 \circ R_3$, which the reader can verify using the `iscc` *calculator:*

```
R1 := [n] -> { [i,j] -> [i+3,j] : i <= 2 j - 4 and i <= n - 3 and
                                  j <= 2 i - 1 and j <= n };
R2 := [n] -> { [i,j] -> [i,j+3] : i <= 2 j - 1 and i <= n and
                                  j <= 2 i - 4 and j <= n - 3 };
R3 := [n] -> { [i,j] -> [i+1,j+1] : i <= 2 j - 1 and i <= n - 1 and
                                    j <= 2 i - 1 and j <= n - 1 };
(R1 . R3) - (R3 . R1);
(R2 . R3) - (R3 . R2);
```

R_3 can therefore be moved forward in any path. For the other two basic relations, we have both $R_2 \circ R_1 \not\subseteq R_1 \circ R_2$ and $R_1 \circ R_2 \not\subseteq R_2 \circ R_1$ and so R_1 and R_2 form a strongly connected component. By computing the power of R_3 and $R_1 \cup R_2$ separately and composing the results, the power of R can be computed exactly using (2.5)*. As explained by Beletska, Barthou, et al. (2009), applying the same formula to R directly, without a decomposition, would result in an overapproximation of the power.*

2.5.5 Partitioning the domains and ranges of R

The algorithm of Section 2.5.2 assumes that the input relation R can be treated as a union of translations. This is a reasonable assumption if R maps elements of a given abstract domain to the same domain. However, if R is a union of relations that map between different domains, then this assumption no longer holds. In particular, when an entire dependence graph is encoded in a single relation, as is done by, e.g., Barthou, Cohen, et al. (2000, Section 6.1), then it does not make sense to look at differences between iterations of different domains. Now, arguably, a modified Floyd-Warshall algorithm should be applied to the dependence graph, as advocated by Kelly, Pugh, et al. (1996), with the transitive closure operation only being applied to relations from a given domain to itself. However, it is also possible to detect disjoint domains and ranges and to apply Floyd-Warshall internally.

Algorithm 1: The modified Floyd-Warshall algorithm of Kelly, Pugh, et al. (1996)

Input: Relations R_{pq}, $0 \leq p, q < n$
Output: Updated relations R_{pq} such that each relation R_{pq} contains all indirect paths from p to q in the input graph

```
1 for r ∈ [0, n − 1] do
2     R_rr := R_rr^+
3     for p ∈ [0, n − 1] do
4         for q ∈ [0, n − 1] do
5             if p ≠ r or q ≠ r then
6                 R_pq := R_pq ∪ (R_rq ∘ R_pr) ∪ (R_rq ∘ R_rr ∘ R_pr)
```

Let the input relation R be a union of m basic relations R_i. Let D_{2i} be the domains of R_i and D_{2i+1} the ranges of R_i. The first step is to group overlapping D_j until a partition

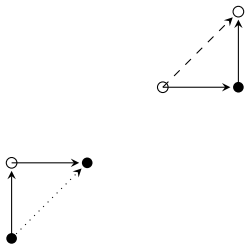

Figure 2.3: The relation (solid arrows) on the right of Figure 1 of Beletska, Barthou, et al. (2009) and its transitive closure

is obtained. If the resulting partition consists of a single part, then we continue with the algorithm of Section 2.5.2. Otherwise, we apply Floyd-Warshall on the graph with as vertices the parts of the partition and as edges the R_i attached to the appropriate pairs of vertices. In particular, let there be n parts P_k in the partition. We construct n^2 relations

$$R_{pq} \coloneqq \bigcup_{i \text{ s.t. } \operatorname{dom} R_i \subseteq P_p \wedge \operatorname{ran} R_i \subseteq P_q} R_i,$$

apply Algorithm 1 and return the union of all resulting R_{pq} as the transitive closure of R. Each iteration of the r-loop in Algorithm 1 updates all relations R_{pq} to include paths that go from p to r, possibly stay there for a while, and then go from r to q. Note that paths that "stay in r" include all paths that pass through earlier vertices since R_{rr} itself has been updated accordingly in previous iterations of the outer loop. In principle, it would be sufficient to use the R_{pr} and R_{rq} computed in the previous iteration of the r-loop in Line 6. However, from an implementation perspective, it is easier to allow either or both of these to have been updated in the same iteration of the r-loop. This may result in duplicate paths, but these can usually be removed by coalescing (Section 2.4) the result of the union in Line 6, which should be done in any case. The transitive closure in Line 2 is performed using a recursive call. This recursive call includes the partitioning step, but the resulting partition will usually be a singleton. The result of the recursive call will either be exact or an overapproximation. The final result of Floyd-Warshall is therefore also exact or an overapproximation.

Example 2.5.9 *Consider the relation on the right of Figure 1 of Beletska, Barthou, et al. (2009), reproduced in Figure 2.3. This relation can be described as*

$$\{(x, y) \to (x_2, y_2) \mid (3y = 2x \wedge x_2 = x \wedge 3y_2 = 3 + 2x \wedge x \geq 0 \wedge x \leq 3) \vee (x_2 = 1 + x \wedge y_2 = y \wedge x \geq 0 \wedge 3y \geq 2 + 2x \wedge x \leq 2 \wedge 3y \leq 3 + 2x)\}.$$

Note that the domain of the upward relation overlaps with the range of the rightward relation and vice versa, but that the domain of neither relation overlaps with its own range or the domain of the other relation. The domains and ranges can therefore be partitioned into two parts, P_0 and P_1, shown as the white and black dots in Figure 2.3,

respectively. Initially, we have

$$
\begin{aligned}
R_{00} &= \emptyset \\
R_{01} &= \{ (x, y) \to (x + 1, y) \mid (x \geq 0 \wedge 3y \geq 2 + 2x \wedge x \leq 2 \wedge 3y \leq 3 + 2x) \} \\
R_{10} &= \{ (x, y) \to (x_2, y_2) \mid (3y = 2x \wedge x_2 = x \wedge 3y_2 = 3 + 2x \wedge x \geq 0 \wedge x \leq 3) \} \\
R_{11} &= \emptyset.
\end{aligned}
$$

In the first iteration, R_{00} remains the same ($\emptyset^+ = \emptyset$). R_{01} and R_{10} are therefore also unaffected, but R_{11} is updated to include $R_{01} \circ R_{10}$, i.e., the dashed arrow in the figure. This new R_{11} is obviously transitively closed, so it is not changed in the second iteration and it does not have an effect on R_{01} and R_{10}. However, R_{00} is updated to include $R_{10} \circ R_{01}$, i.e., the dotted arrow in the figure. The transitive closure of the original relation is then equal to $R_{00} \cup R_{01} \cup R_{10} \cup R_{11}$.

2.5.6 Incremental Computation

In some cases it is possible and useful to compute the transitive closure of union of basic relations incrementally. In particular, if R is a union of m basic maps,

$$R = \bigcup_j R_j,$$

then we can pick some R_i and compute the transitive closure of R as

$$R^+ = R_i^+ \cup \left(\bigcup_{j \neq i} R_i^* \circ R_j \circ R_i^* \right)^+ . \tag{2.13}$$

For this approach to be successful, it is crucial that each of the disjuncts in the argument of the second transitive closure in (2.13) be representable as a single basic relation, i.e., without a union. If this condition holds, then by using (2.13), the number of disjuncts in the argument of the transitive closure can be reduced by one. Now, $R_i^* = R_i^+ \cup \mathrm{Id}$, but in some cases it is possible to relax the constraints of R_i^+ to include part of the identity relation, say on domain D. We will use the notation $C(R_i, D) = R_i^+ \cup \mathrm{Id}_D$ to represent this relaxed version of R^+. Kelly, Pugh, et al. (1996) use the notation $R_i^?$. $C(R_i, D)$ can be computed by allowing k to attain the value 0 in (2.10) and by using

$$P \cap (D \to D)$$

instead of (2.3). Typically, D will be a strict superset of both $\operatorname{dom} R_i$ and $\operatorname{ran} R_i$. We therefore need to check that domain and range of the transitive closure are part of $C(R_i, D)$, i.e., the part that results from the paths of positive length ($k \geq 1$), are equal to the domain and range of R_i. If not, then the incremental approach cannot be applied for the given choice of R_i and D.

In order to be able to replace R^* by $C(R_i, D)$ in (2.13), D should be chosen to include both $\operatorname{dom} R$ and $\operatorname{ran} R$, i.e., such that $\mathrm{Id}_D \circ R_j \circ \mathrm{Id}_D = R_j$ for all $j \neq i$. Kelly, Pugh, et al. (1996) say that they use $D = \operatorname{dom} R_i \cup \operatorname{ran} R_i$, but presumably they mean

that they use $D = \operatorname{dom} R \cup \operatorname{ran} R$. Now, this expression of D contains a union, so it not directly usable. Kelly, Pugh, et al. (1996) do not explain how they avoid this union. Apparently, in their implementation, they are using the convex hull of $\operatorname{dom} R \cup \operatorname{ran} R$ or at least an approximation of this convex hull. We use the simple hull (Section 2.2) of $\operatorname{dom} R \cup \operatorname{ran} R$.

It is also possible to use a domain D that does *not* include $\operatorname{dom} R \cup \operatorname{ran} R$, but then we have to compose with $C(R_i, D)$ more selectively. In particular, if we have

$$\text{for each } j \neq i \text{ either } \operatorname{dom} R_j \subseteq D \text{ or } \operatorname{dom} R_j \cap \operatorname{ran} R_i = \emptyset \tag{2.14}$$

and, similarly,

$$\text{for each } j \neq i \text{ either } \operatorname{ran} R_j \subseteq D \text{ or } \operatorname{ran} R_j \cap \operatorname{dom} R_i = \emptyset \tag{2.15}$$

then we can refine (2.13) to

$$R_i^+ \cup \left(\left(\bigcup_{\substack{\operatorname{dom} R_j \subseteq D \\ \operatorname{ran} R_j \subseteq D}} C \circ R_j \circ C \right) \cup \left(\bigcup_{\substack{\operatorname{dom} R_j \cap \operatorname{ran} R_i = \emptyset \\ \operatorname{ran} R_j \subseteq D}} C \circ R_j \right) \cup \left(\bigcup_{\substack{\operatorname{dom} R_j \subseteq D \\ \operatorname{ran} R_j \cap \operatorname{dom} R_i = \emptyset}} R_j \circ C \right) \cup \left(\bigcup_{\substack{\operatorname{dom} R_j \cap \operatorname{ran} R_i = \emptyset \\ \operatorname{ran} R_j \cap \operatorname{dom} R_i = \emptyset}} R_j \right) \right)^+ .$$

If only property (2.14) holds, we can use

$$R_i^+ \cup \left((R_i^+ \cup \mathrm{Id}) \circ \left(\left(\bigcup_{\operatorname{dom} R_j \subseteq D} R_j \circ C \right) \cup \left(\bigcup_{\operatorname{dom} R_j \cap \operatorname{ran} R_i = \emptyset} R_j \right) \right)^+ \right),$$

while if only property (2.15) holds, we can use

$$R_i^+ \cup \left(\left(\left(\bigcup_{\operatorname{ran} R_j \subseteq D} C \circ R_j \right) \cup \left(\bigcup_{\operatorname{ran} R_j \cap \operatorname{dom} R_i = \emptyset} R_j \right) \right)^+ \circ (R_i^+ \cup \mathrm{Id}) \right).$$

It should be noted that if we want the result of the incremental approach to be transitively closed, then we can only apply it if all of the transitive closure operations involved are exact. If, say, the second transitive closure in (2.13) contains extra elements, then the result does not necessarily contain the composition of these extra elements with powers of R_i.

2.5.7 An `Omega`-like implementation

While the main algorithm of Kelly, Pugh, et al. (1996) is designed to compute and underapproximation of the transitive closure, the authors mention that they could also compute overapproximations. In this section, we describe our implementation of an algorithm that is based on their ideas. Note that the `Omega` library computes underapproximations (Kelly, Maslov, et al. 1996b, Section 6.4).

The main tool is Equation (2) of Kelly, Pugh, et al. (1996). The input relation R is first overapproximated by a "d-form" relation

$$\{ \mathbf{i} \to \mathbf{j} \mid \exists \boldsymbol{\alpha} : \mathbf{L} \leq \mathbf{j} - \mathbf{i} \leq \mathbf{U} \wedge (\forall p : j_p - i_p = M_p \alpha_p) \},$$

where p ranges over the dimensions and $\mathbf{L}$, $\mathbf{U}$ and $\mathbf{M}$ are constant integer vectors. The elements of $\mathbf{U}$ may be ∞, meaning that there is no upper bound corresponding to that element, and similarly for $\mathbf{L}$. Such an overapproximation can be obtained by computing strides, lower and upper bounds on the difference set ΔR. The transitive closure of such a "d-form" relation is

$$\{\,\mathbf{i} \to \mathbf{j} \mid \exists \boldsymbol{\alpha}, k : k \geq 1 \wedge k\,\mathbf{L} \leq \mathbf{j} - \mathbf{i} \leq k\,\mathbf{U} \wedge (\forall p : j_p - i_p = M_p \alpha_p)\,\}. \tag{2.16}$$

The domain and range of this transitive closure are then intersected with those of the input relation. This is a special case of the algorithm in Section 2.5.2.

In their algorithm for computing lower bounds, the authors use the above algorithm as a substep on the disjuncts in the relation. At the end, they say

> If an upper bound is required, it can be calculated in a manner similar to that of a single conjunct [sic] relation.

Presumably, the authors mean that a "d-form" approximation of the whole input relation should be used. However, the accuracy can be improved by also trying to apply the incremental technique from the same paper, which is explained in more detail in Section 2.5.6. In this case, $C(R_i, D)$ can be obtained by allowing the value zero for k in (2.16), i.e., by computing

$$\{\,\mathbf{i} \to \mathbf{j} \mid \exists \boldsymbol{\alpha}, k : k \geq 0 \wedge k\,\mathbf{L} \leq \mathbf{j} - \mathbf{i} \leq k\,\mathbf{U} \wedge (\forall p : j_p - i_p = M_p \alpha_p)\,\}.$$

In our implementation we take as D the simple hull (Section 2.2) of $\operatorname{dom} R \cup \operatorname{ran} R$. To determine whether it is safe to use $C(R_i, D)$, we check the following conditions, as proposed by Kelly, Pugh, et al. (1996): $C(R_i, D) - R_i^+$ is not a union and for each $j \neq i$ the condition

$$\left(C(R_i, D) - R_i^+\right) \circ R_j \circ \left(C(R_i, D) - R_i^+\right) = R_j$$

holds.

Chapter 3

Further Reading

Verdoolaege (2016) describes the concepts behind `isl` in some detail, mainly focusing on Presburger formulas, but also including some information on polyhedral compilation, especially on dependence analysis. Individual aspects of `isl` are described in the following publications.

- Verdoolaege, Janssens, et al. (2009) introduce `isl` as a library for manipulating sets of integers defined by linear inequalities and integer divisions that is used in their equivalence checker.
- Verdoolaege (2010a) provides a more detailed description of `isl` at the time and still stands as the official reference for `isl`. However, many features were only added later on and one or more of the publications below may be more appropriate as a reference to these features.
- Verdoolaege (2010b, Section 5.1) provides some details on the dataflow analysis step, but also see Verdoolaege (2016, Chapter 6) and Verdoolaege and Cohen (2016) for a more recent treatment.
- The concepts of structured and named spaces and the manipulation of sets containing elements in different spaces were introduced by Verdoolaege (2011).
- The transitive closure operation is described by Verdoolaege, Cohen, and Beletska (2011).
- The scheduler is briefly described by Verdoolaege, Juega, et al. (2013, Section 6.2) and Verdoolaege and Cohen (2016, Section 2.4).
- Schedule trees started out as "trees of bands" (Verdoolaege, Juega, et al. 2013, Section 6.2), were formally introduced by Verdoolaege, Guelton, et al. (2014), and were slightly refined by Grosser, Verdoolaege, et al. (2015).
- The coalescing operation is described by Verdoolaege (2015).
- The AST generator is described by Grosser, Verdoolaege, et al. (2015).

Bibliography

Bagnara, R., P. M. Hill, and E. Zaffanella. *The Parma Polyhedra Library*. http://www.cs.unipr.it/ppl/. [215]

Barthou, Denis, Albert Cohen, and Jean-François Collard (2000). "Maximal Static Expansion". In: *Int. J. Parallel Program.* 28.3, pp. 213–243. DOI: 10.1023/A:1007500431910. [228]

Barvinok, Alexander and Kevin Woods (Apr. 2003). "Short rational generating functions for lattice point problems". In: *J. Amer. Math. Soc.* 16, pp. 957–979. DOI: 10.1090/S0894-0347-03-00428-4. [209]

Beletska, Anna, Denis Barthou, Wlodzimierz Bielecki, and Albert Cohen (2009). "Computing the Transitive Closure of a Union of Affine Integer Tuple Relations". In: *COCOA '09: Proceedings of the 3rd International Conference on Combinatorial Optimization and Applications*. Huangshan, China: Springer-Verlag, pp. 98–109. DOI: 10.1007/978-3-642-02026-1_9. [219, 220, 227–229]

Boulet, Pierre and Xavier Redon (1998). "Communication Pre-evaluation in HPF". In: *EUROPAR'98*. Vol. 1470. Lecture Notes in Computer Science. Springer-Verlag, Berlin, pp. 263–272. DOI: 10.1007/BFb0057861. [209]

Bygde, Stefan (Mar. 2010). *Static WCET Analysis based on Abstract Interpretation and Counting of Elements*. Licentiate thesis. [213, 217]

Cook, William, Thomas Rutherford, Herbert E. Scarf, and David F. Shallcross (Aug. 1991). *An Implementation of the Generalized Basis Reduction Algorithm for Integer Programming*. Cowles Foundation Discussion Papers 990. available at http://ideas.repec.org/p/cwl/cwldpp/990.html. Cowles Foundation, Yale University. [216]

De Loera, J. A., D. Haws, R. Hemmecke, P. Huggins, and R. Yoshida (Jan. 2004). "Three Kinds of Integer Programming Algorithms based on Barvinok's Rational Functions". In: *Integer Programming and Combinatorial Optimization: 10th International IPCO Conference*. Vol. 3064. Lecture Notes in Computer Science, pp. 244–255. DOI: 10.1007/978-3-540-25960-2_19. [209]

De Smet, Sven (Apr. 2010). *personal communication*. [221]

Detlefs, David, Greg Nelson, and James B. Saxe (2005). "Simplify: a theorem prover for program checking". In: *J. ACM* 52.3, pp. 365–473. DOI: 10.1145/1066100.1066102. [209]

Feautrier, P. (1988). "Parametric Integer Programming". In: *RAIRO Recherche Opérationnelle* 22.3, pp. 243–268. [208, 209, 214]

Feautrier, P. (1991). "Dataflow analysis of array and scalar references". In: *International Journal of Parallel Programming* 20.1, pp. 23–53. DOI: 10.1007/BF01407931. [208, 211]

Feautrier, P., J. Collard, and C. Bastoul (2002). *Solving systems of affine (in)equalities*. Tech. rep. PRiSM, Versailles University. [211, 212]

Feautrier, Paul (Dec. 1992). "Some Efficient Solutions to the Affine Scheduling Problem. Part II. Multidimensional Time". In: *International Journal of Parallel Programming* 21.6, pp. 389–420. DOI: 10.1007/BF01379404. [211]

Galea, François (Nov. 2009). *personal communication*. [215]

Grosser, Tobias, Sven Verdoolaege, and Albert Cohen (July 2015). "Polyhedral AST generation is more than scanning polyhedra". In: *ACM Transactions on Programming Languages and Systems* 37.4, 12:1–12:50. DOI: 10.1145/2743016. [233]

Karr, Michael (1976). "Affine Relationships Among Variables of a Program". In: *Acta Informatica* 6, pp. 133–151. DOI: 10.1007/BF00268497. [216]

Kelly, Wayne, Vadim Maslov, William Pugh, Evan Rosser, Tatiana Shpeisman, and Dave Wonnacott (Nov. 1996a). *The Omega Calculator and Library*. Tech. rep. University of Maryland. [227]

Kelly, Wayne, Vadim Maslov, William Pugh, Evan Rosser, Tatiana Shpeisman, and Dave Wonnacott (Nov. 1996b). *The Omega Library*. Tech. rep. University of Maryland. [231]

Kelly, Wayne, William Pugh, Evan Rosser, and Tatiana Shpeisman (1996). "Transitive Closure of Infinite Graphs and Its Applications". In: *Languages and Compilers for Parallel Computing, 8th International Workshop, LCPC'95, Columbus, Ohio, USA, August 10-12, 1995, Proceedings*. Ed. by Chua-Huang Huang, P. Sadayappan, Utpal Banerjee, David Gelernter, Alexandru Nicolau, and David A. Padua. Vol. 1033. Lecture Notes in Computer Science. Springer, pp. 126–140. DOI: 10.1007/BFb0014196. [219, 225, 228, 230–232]

Meister, Benoît (Dec. 2004). "Stating and Manipulating Periodicity in the Polytope Model. Applications to Program Analysis and Optimization". PhD thesis. Université Louis Pasteur. [213]

Meister, Benoît and Sven Verdoolaege (Apr. 2008). "Polynomial Approximations in the Polytope Model: Bringing the Power of Quasi-Polynomials to the Masses". In: *Digest of the 6th Workshop on Optimization for DSP and Embedded Systems, ODES-6*. Ed. by Jagadeesh Sankaran and Tom Vander Aa. [213]

Nelson, Charles Gregory (1980). "Techniques for program verification". PhD thesis. Stanford, CA, USA: Stanford University. [209]

Schrijver, Alexander (1986). *Theory of Linear and Integer Programming*. John Wiley & Sons. [221]

Seghir, Rachid and Vincent Loechner (Oct. 2006). "Memory Optimization by Counting Points in Integer Transformations of Parametric Polytopes". In: *Proceedings of the International Conference on Compilers, Architectures, and Synthesis for Embedded Systems, CASES 2006, Seoul, Korea*. DOI: 10.1145/1176760.1176771. [217]

Tarjan, Robert (1972). "Depth-First Search and Linear Graph Algorithms". In: *SIAM Journal on Computing* 1.2, pp. 146–160. DOI: 10.1137/0201010. [226]

Verdoolaege, Sven (2006). *barvinok, version 0.22*. Available from http://barvinok.gforge.inria.fr/. [209]

Verdoolaege, Sven (2010a). “isl: An Integer Set Library for the Polyhedral Model”. In: *Mathematical Software - ICMS 2010*. Ed. by Komei Fukuda, Joris Hoeven, Michael Joswig, and Nobuki Takayama. Vol. 6327. Lecture Notes in Computer Science. Springer, pp. 299–302. DOI: 10.1007/978-3-642-15582-6_49. [233]

Verdoolaege, Sven (2010b). “Polyhedral process networks”. In: *Handbook of Signal Processing Systems*. Ed. by Shuvra Bhattacharrya, Ed Deprettere, Rainer Leupers, and Jarmo Takala. Springer, pp. 931–965. DOI: 10.1007/978-1-4419-6345-1_33. [233]

Verdoolaege, Sven (Apr. 2011). “Counting Affine Calculator and Applications”. In: *First International Workshop on Polyhedral Compilation Techniques (IMPACT'11)*. Chamonix, France. DOI: 10.13140/RG.2.1.2959.5601. [233]

Verdoolaege, Sven (Jan. 2015). “Integer Set Coalescing”. In: *Proceedings of the 5th International Workshop on Polyhedral Compilation Techniques*. Amsterdam, The Netherlands. DOI: 10.13140/2.1.1313.6968. [218, 233]

Verdoolaege, Sven (2016). *Presburger Formulas and Polyhedral Compilation*. DOI: 10.13140/RG.2.1.1174.6323. [233]

Verdoolaege, Sven, Kristof Beyls, Maurice Bruynooghe, and Francky Catthoor (2005). “Experiences with enumeration of integer projections of parametric polytopes”. In: *Proceedings of 14th International Conference on Compiler Construction, Edinburgh, Scotland*. Ed. by R. Bodik. Vol. 3443. Lecture Notes in Computer Science. Berlin: Springer-Verlag, pp. 91–105. DOI: 10.1007/b107108. [209, 217]

Verdoolaege, Sven and Albert Cohen (Jan. 2016). “Live-Range Reordering”. In: *Proceedings of the sixth International Workshop on Polyhedral Compilation Techniques*. Prague, Czech Republic. DOI: 10.13140/RG.2.1.3272.9680. [233]

Verdoolaege, Sven, Albert Cohen, and Anna Beletska (2011). “Transitive Closures of Affine Integer Tuple Relations and Their Overapproximations”. In: *Proceedings of the 18th International Conference on Static Analysis*. SAS'11. Venice, Italy: Springer-Verlag, pp. 216–232. DOI: 10.1007/978-3-642-23702-7_18. [233]

Verdoolaege, Sven, Serge Guelton, Tobias Grosser, and Albert Cohen (Jan. 2014). “Schedule Trees”. In: *Proceedings of the 4th International Workshop on Polyhedral Compilation Techniques*. Vienna, Austria. DOI: 10.13140/RG.2.1.4475.6001. [233]

Verdoolaege, Sven, Gerda Janssens, and Maurice Bruynooghe (June 2009). “Equivalence checking of static affine programs using widening to handle recurrences”. In: *Computer Aided Verification 21*. Springer, pp. 599–613. DOI: 10.1007/978-3-642-02658-4_44. [233]

Verdoolaege, Sven, Juan Carlos Juega, Albert Cohen, José Ignacio Gómez, Christian Tenllado, and Francky Catthoor (2013). “Polyhedral parallel code generation for CUDA”. In: *ACM Trans. Archit. Code Optim.* 9.4, p. 54. DOI: 10.1145/2400682.2400713. [233]

Verhaegh, Wim F. J. (1995). “Multidimensional Periodic Scheduling”. PhD thesis. Technische Universiteit Eindhoven. [216]