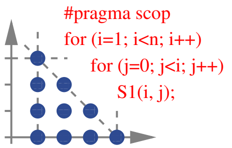


CLAN REFERENCE CARD

Version 1.0 for Clan 0.7.0

About Clan

Clan is a translator from C-like code parts to polyhedral representation. It opens the gates of powerful polyhedral compilation techniques provided by, e.g., PoCC or Pluto. Programmers should ensure their computation-intensive code parts are compatible with Clan's input to benefit from state-of-the-art automatic optimization and parallelization.

Basic Concepts

Static Control Parts

Clan is capable to translate program parts easily amenable to the polyhedral model. We call them *Static Control Parts* (SCoP for short). They are basically loop-based codes where loop bounds, `if` conditions and array subscripts are made of affine expressions involving only outer loop iterators, integer constants (a.k.a. parameters) and integer literals.

SCoP Pragmas

Clan translates code parts delimited by specific pragmas and ignores what is outside those regions:

- between `#pragma scop` and `#pragma endscop` for C/C++,
- between `/*@ scop */` and `/*@ end scop */` for JAVA.

In addition to the syntactic restrictions imposed by Clan, inserting SCoP pragmas in a code also implicitely specifies that:

- all functions called within the SCoP are pure (no side-effects),
- no aliasing of array names is possible within the SCoP,
- pointer references behave like variables or arrays.

Affine Expressions

Affine expressions are additive forms of loop iterators (e.g., i), parameters (e.g., N) and integers, with integer coefficients, e.g., $7*i+13*N+42$.

- Expressions simplifying to affine forms are OK, e.g., $3*(i*2+N)$.

Specific Operators

Four particular operators may be used in Clan's input. Let us suppose that `a` and `b` are affine expressions and `n` an integer:

Maximum of `a` and `b` (`a` and `b` may be `max` expressions)	`max(a, b)`
Minimum of `a` and `b` (`a` and `b` may be `min` expressions)	`min(a, b)`
Ceil of `a` divided by `n` (considered as a `max` expression)	`ceild(a, n)`
Floor of `a` divided by `n` (considered as a `min` expression)	`floord(a, n)`

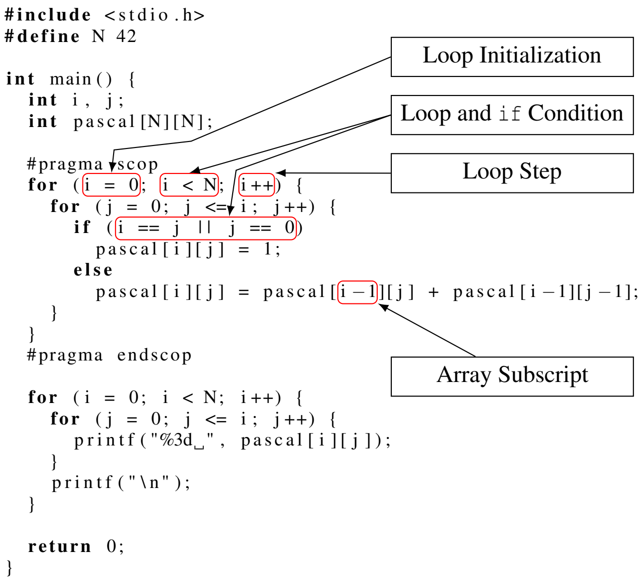
Please send comments and corrections to Cédric Bastoul, `cedric.bastoul@u-psud.fr`

General Restrictions

Codes between SCoP pragmas must comply to the following restrictions:

- The only allowed control keywords are `for`, `while`, `if` and `else`, with restrictions as described below.
- Declarations are not allowed.
- Any C instruction without control keywords is accepted, with restrictions for array subscripts as described below.

Identification of Constrained Elements



```
#include <stdio.h>
#define N 42

int main() {
  int i, j;
  int pascal[N][N];

  #pragma scop
  for (i = 0; i < N; i++) {
    for (j = 0; j <= i; j++) {
      if (i == j || j == 0)
        pascal[i][j] = 1;
      else
        pascal[i][j] = pascal[i-1][j] + pascal[i-1][j-1];
    }
  }
  #pragma endscop

  for (i = 0; i < N; i++) {
    for (j = 0; j <= i; j++) {
      printf("%3d␣", pascal[i][j]);
    }
    printf("\n");
  }

  return 0;
}
```

Loop Initialization

Loop and `if` Condition

Loop Step

Array Subscript

Loop Initialization

Each loop initialization must be an assignment of the loop counter such that the right hand side is one or several affine expressions aggregated with `max` (resp. `min`) operators if the loop step is positive (resp. negative).

Example of Loop Initialization	Diagnostic
`j = 3*i + 2*N`	Correct
`j = ceild(i + N, 10)`	Correct if `j` step is positive
`j = max(i, ceild(N, 3))`	Correct if `j` step is positive
`j = min(min(N,10), 7*i)`	Correct if `j` step is negative
`j = min(max(i, 1), N)`	Incorrect: mixed `min` and `max`

Tip: if the initialization form is too restrictive for a given program, it may be possible to move the troublesome constraints to the loop condition or to an external or internal `if` condition.

Loop and `if` Condition

Each loop or `if` condition must be a (composition of) constraint(s) on affine expressions, ~~and function calls~~.

- Supported C operators are `>`, `>=`, `<`, `<=`, `==`, `!=`, `!`, `&&` and `||`.
- `min` and `max` operators can be used to aggregate expressions in `>`, `>=`, `<` and `<=` constraints. `min` (resp. `max`) expressions must be in the greater (resp. lower) side of the constraints.
- Constraints involving the modulo operator are possible in the following form: let `a` be an affine expression and `x` and `y` two positive integers, then the condition `(a % x == y)` is accepted.
- ~~Function calls alone can be used as valid `if` conditions~~.

Example of Condition	Diagnostic
`i + 2*j < N`	Correct
`max(i, j) < floord(N, 7)`	Correct
`N>i && !(j>0 || N!=1)`	Correct
`((2*i+1)%3 == 1) && i>j`	Correct
`func(A[i], b)`	Correct in a `if` condition
`min(2*i, N) < 0`	Incorrect: `min` on the lower side
`i + 2`	Incorrect: use `(i + 2) != 0`
`i<N && g(a)`	Incorrect: function call not alone

~~*Tip: to include data-dependent conditions, e.g.,* `if (A[i] == 0)`*, create a preprocessor macro containing the condition and replace it in the SCoP by the macro-function call, e.g.,* `if (my_condition(A[i]))`~~.

Loop Step

Updating the loop iterator is only allowed in the loop step part. It must be done by adding an integer to the previous iterator value. Let `i` be a loop iterator and `x` an integer, the following forms are accepted for the loop step part: `i++`, `++i`, `i--`, `--i`, `i += x`, `i -= x`, `i = i+x` and `i = i-x`.

Array Subscript

Array subscripts must be either affine expressions ~~or function calls~~.

~~*Tip: to include indirections, e.g.,* `A[B[i]]`*, create a preprocessor macro containing the subscript and replace it in the SCoP by the macro-function call, e.g.,* `A[my_subscript(B[i])]`~~.

Infinite and `while` Loops

Infinite `for` loops in the form `for (;;)` are supported. `while` loops are supported when the condition is ~~either~~ `1` (infinite loop) ~~or a function call~~.