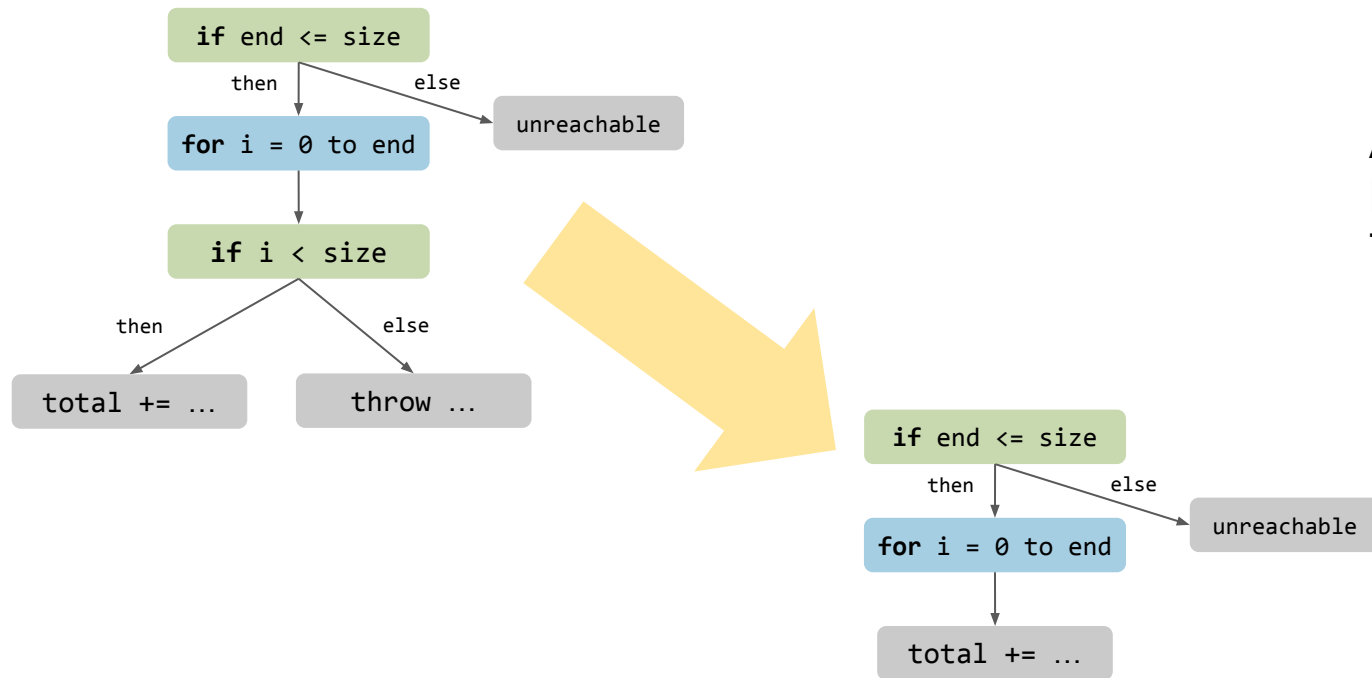


Precise Polyhedral Analyses for MLIR using the FPL Presburger Library



Arjun Pitchanathan
Kunwar Grover
Tobias Grosser

...and many more



Growing interest in MLIR's polyhedral functionality

✓ [mlir][Affine] Add support for multi-store producer fusion

This patch adds support for producer-consumer fusion scenarios with multiple producer stores to the AffineLoopFusion pass. The patch introduces some changes to the producer-consumer algorithm, including:

- * For a given consumer loop, producer-consumer fusion iterates over its producer candidates until a fixed point is reached.
- * Producer candidates are gathered beforehand for each iteration of the consumer loop and visited in reverse program order (not strictly guaranteed) to maximize the number of loops fused per iteration.

In general, these changes were needed to simplify the multi-store producer support and remove some of the workarounds that were introduced in the past to support more fusion cases under the single-store producer limitation.

This patch also preserves the existing functionality of AffineLoopFusion with one minor change in behavior. Producer-consumer fusion didn't fuse scenarios with escaping memrefs and multiple outgoing edges (from a single store). Multi-store producer scenarios will usually (always?) have multiple outgoing edges so we couldn't fuse any with escaping memrefs, which would greatly limit the applicability of this new feature. Therefore, the patch enables fusion for these scenarios. Please, see modified tests for specific details.

Reviewed By: andydavis1, bondhugula

Differential Revision: <https://reviews.llvm.org/D92876>

main
llvmorg-15-init llvmorg-12.0.0-rc1

dcaballe committed on 25 Jan 2021



✓ [Analysis] Add a DependenceAnalysis for checking memory accesses. (#1845)

This is currently a simple class that traverses pairs of Affine memory access operations and uses the upstream 'checkMemrefAccessDependence' function. The results are stored in a convenient data structure that can be queried to inform scheduling decisions. A test pass is added, which outputs the results as attributes for verification.

main (#1845)
sifive/0/7/0 pycde-0.0.1

mikeurbach committed on Sep 24, 2021 Verified

Eliminating redundant checks

```
uint64_t foo(std::vector<uint64_t> &data, size_t end) {  
    assert(end <= data.size());  
    uint64_t total = 0;  
    for (size_t i = 0; i < end; ++i) {  
        // data.at(i) internally checks if i < data.size().  
  
        total += data.at(i);  
    }  
    return total;  
}
```



2021

LLVM DEVELOPERS' MEETING



Florian Hahn

A New Approach to Removing
Range Checks in LLVM

Eliminating redundant checks

```
uint64_t foo(std::vector<uint64_t> &data, size_t end) {  
    assert(end <= data.size());  
    uint64_t total = 0;  
    for (size_t i = 0; i < end; ++i) {  
        if (i >= data.size())  
            throw std::out_of_range("");  
        total += data[i]; // operator[] has no bounds check.  
    }  
    return total;  
}
```

Eliminating redundant checks

```
uint64_t foo(std::vector<uint64_t> &data, size_t end) {  
    assert(end <= data.size());  
    uint64_t total = 0;  
    for (size_t i = 0; i < end; ++i) {  
        if (i >= data.size())  
            throw std::out_of_range("");  
        total += data[i];  
    }  
    return total;  
}
```

Eliminating redundant checks

```
uint64_t foo(std::vector<uint64_t> &data, size_t end) {  
    assert(end <= data.size());  
    uint64_t total = 0;  
    for (size_t i = 0; i < end; ++i) {  
        if (i >= data.size())  
            throw std::out_of_range("");  
        total += data[i];  
    }  
    return total;  
}
```

end <= data.size()

Eliminating redundant checks

```
uint64_t foo(std::vector<uint64_t> &data, size_t end) {  
    assert(end <= data.size());  
    uint64_t total = 0;  
    for (size_t i = 0; i < end; ++i) {  
        if (i >= data.size())  
            throw std::out_of_range("");  
        total += data[i];  
    }  
    return total;  
}
```

end <= data.size(),
i < end

implies

i < data.size()

Eliminating redundant checks

```
uint64_t foo(std::vector<uint64_t> &data, size_t end) {  
    assert(end <= data.size());  
    uint64_t total = 0;  
    for (size_t i = 0; i < end; ++i) {  
        if (i >= data.size())  
        throw std::out_of_range("");  
        total += data[i];  
    }  
    return total;  
}
```

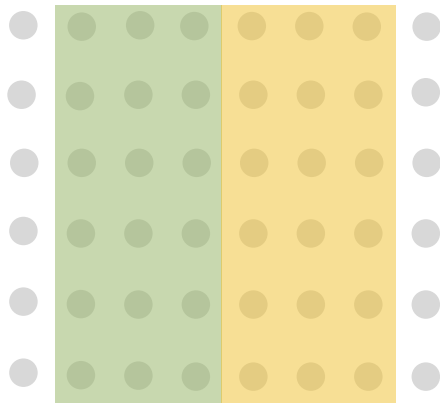
end <= data.size(),
i < end

implies

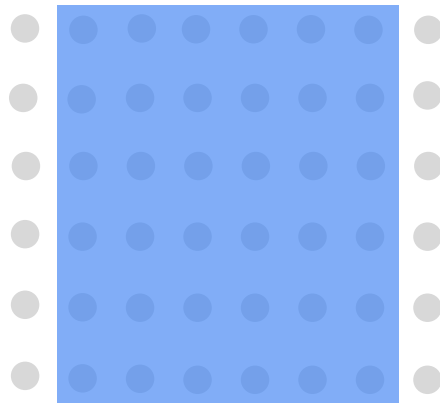
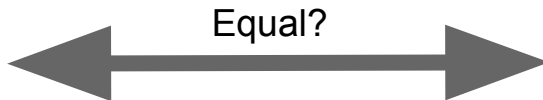
i < data.size()

Beyond Affine: Subview Fusion

```
%0 = memref.alloc() : memref<3x512xbf16, 1>
%1 = memref.subview %0[0, 0] [3, 256] [1, 1] : ...
%2 = memref.subview %0[0, 256] [3, 256] [1, 1] : ...
// write to %1
// write to %2
```



Disjoint?



Analyzing Partial Writes

MLIR



manbearian

Nov '21

I'm looking at doing some analysis to detect when a series of partial writes combines to be equal to a larger write. Is there any existing technology in MLIR i can leverage for this?

Example:

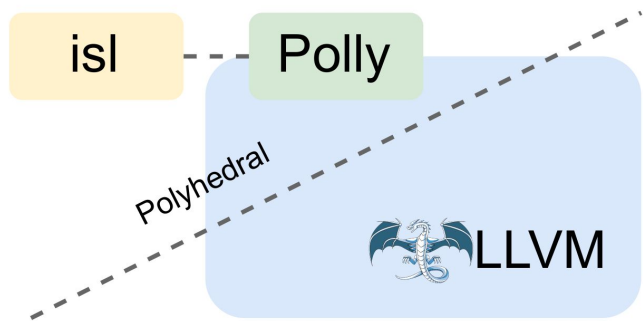
```
%0 = memref.alloc() : memref<1x3x512x256xbf16, 1>
%1 = memref.subview %0[0, 0, 0] [1, 3, 256, 256] [1, 1, 1, 1] : memref<1x
%2 = memref.subview %0[0, 0, 256, 0] [1, 3, 256, 256] [1, 1, 1, 1] : memref<
// write to %1
// write to %2
```

In this example, I'm looking to be able to detect that the two subviews combine to be totally overlapping with the original tensor.

This analysis would need to detect both a sequence of writes as well as a subview that is created dynamically within a loop.

Thanks,
ian

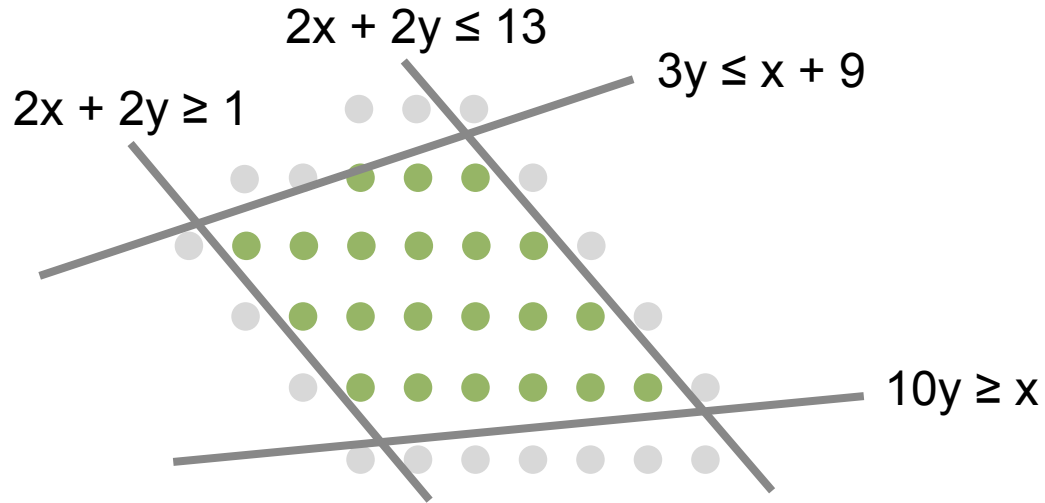
Polyhedral infrastructure: LLVM/Polly vs FPL/MLIR



Built from the ground-up for LLVM/MLIR

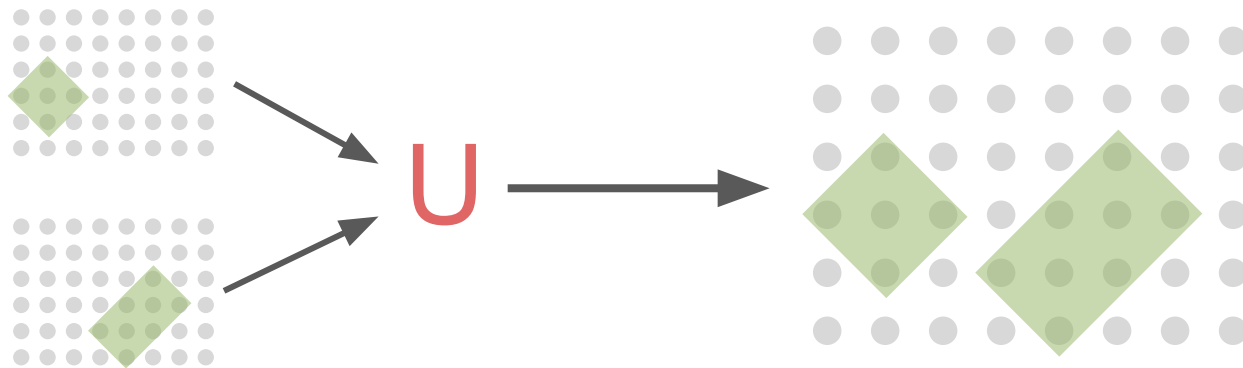
Conversion to & from MLIR IR constructs

The basic building blocks: Integer Polyhedra



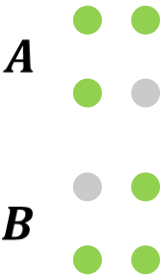
$$\{(x, y) \in \mathbf{Z}^2 : 2x + 2y \geq 1 \wedge 2x + 2y \leq 13 \wedge 3y \leq x + 9 \wedge 10y \geq x\}$$

Presburger Sets: Unions of Integer Polyhedra



$$\{(x, y) \in \mathbf{Z}^2 : (0 \leq x - y \leq 2 \wedge 2 \leq x + y \leq 4) \vee (-4 \leq x - y \leq -2 \wedge 4 \leq x + y \leq 8)\}$$

Operations on Integer Sets



A intersect B



A union B



A subtract B



complement A



is A empty false

sample A



coalesce A

Affine Dialect

```
affine.for %i = 2 * %S + 4 to 3 * %T + 8 step 2
{
  affine.if (%i >= 0, %i < %N) {

    ...

  }
}
```

Affine loop bounds

Affine conditions

Which *i* values get executed?

```
affine.for %i = 2 * %S + 4 to 3 * %T + 8 step 2
{
  affine.if (%i >= 0, %i < %N) {

    (i)[S, T, N] : (i == 2*(i floordiv 2),
                  2*S + 4 <= i <= 3*T + 8,
                  0 <= i < N)

  }
}
```

Kinds of IDs in IntegerPolyhedrons

```
affine.for %i = 2 * %S + 4 to 3 * %T + 8 step 2
{
  affine.if (%i >= 0, %i < %N) {

    (i)[S, T, N] : (i == 2*(i floordiv 2),
                  2*S + 4 <= i <= 3*T + 8,
                  0 <= i < N)

  }
}
```

Set Dimensions

Kinds of IDs in IntegerPolyhedrons

```
affine.for %i = 2 * %S + 4 to 3 * %T + 8 step 2
{
  affine.if (%i >= 0, %i < %N) {

    (i)[S, T, N] : (i == 2*(i floordiv 2),
                  2*S + 4 <= i <= 3*T + 8,
                  0 <= i < N)

  }
}
```

Set Dimensions

Symbols

Kinds of IDs in IntegerPolyhedrons

```
affine.for %i = 2 * %S + 4 to 3 * %T + 8 step 2
{
  affine.if (%i >= 0, %i < %N) {

    (i)[S, T, N] : (i == 2*(i floordiv 2),
                  2*S + 4 <= i <= 3*T + 8,
                  0 <= i < N)

  }
}
```

Set Dimensions

Symbols

Locals

PresburgerSpaces

```
affine.for %i = 2 * %S + 4 to 3 * %T + 8 step 2
{
  affine.if (%i >= 0, %i < %N) {

    (i)[S, T, N] : (i == 2*(i floordiv 2),
                  2*S + 4 <= i <= 3*T + 8,
                  0 <= i < N)

  }
}
```

PresburgerSpace

Set Dimensions

Symbols

Locals

PresburgerSpaces: mlir::Value associations

```
affine.for %i = 2 * %S + 4 to 3 * %T + 8 step 2
{
  affine.if (%i >= 0, %i < %N) {
```

```
    (i)[S, T, N] : (i == 2*(i floordiv 2),
                  2*S + 4 <= i <= 3*T + 8,
                  0 <= i < N)

  }
}
```

Set Dimensions

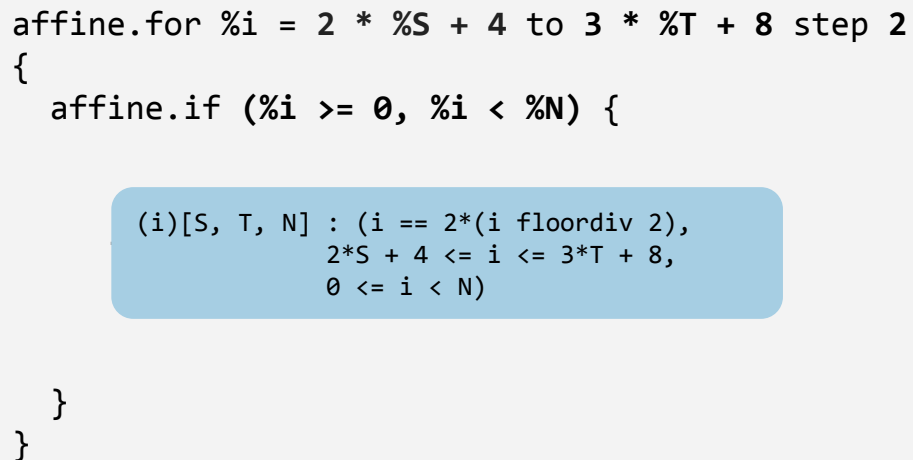
Symbols

Locals

PresburgerSpace

PresburgerSpaces: merging

```
affine.for %i = 2 * %S + 4 to 3 * %T + 8 step 2
{
  affine.if (%i >= 0, %i < %N) {
```



```
    (i)[S, T, N] : (i == 2*(i floordiv 2),
                  2*S + 4 <= i <= 3*T + 8,
                  0 <= i < N)

  }
}
```

(i)[S, T, N] : (i == 2*(i floordiv 2),
2*S + 4 <= i <= 3*T + 8,
0 <= i < N)

Set Dimensions

Symbols

Locals

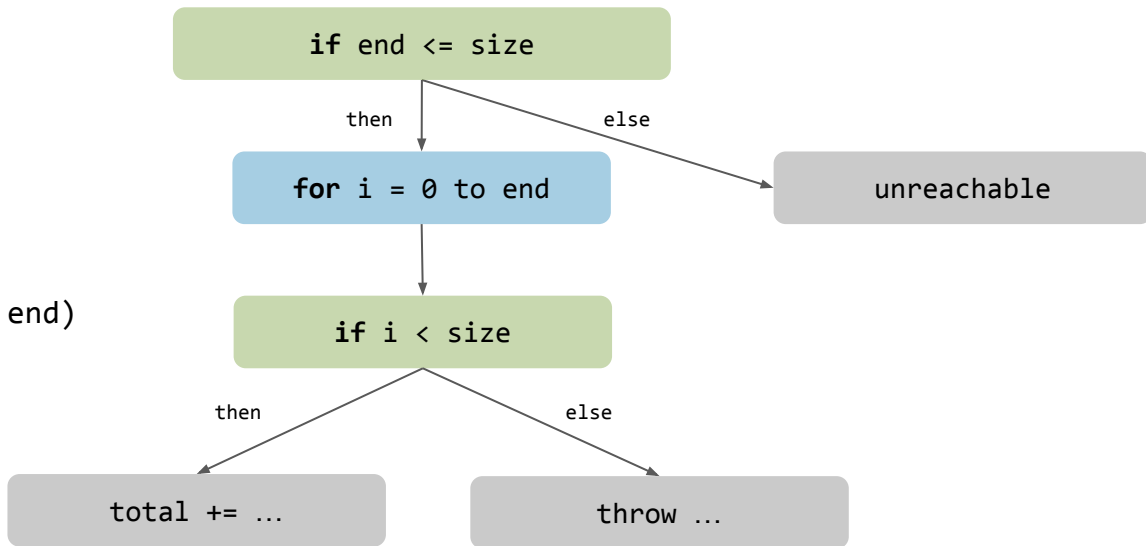
PresburgerSpace

Simplifying Affine Ifs

`()[] : ()`

`()[end, size] : (end <= size)`

`(i)[end, size] : (end <= size, 0 <= i < end)`

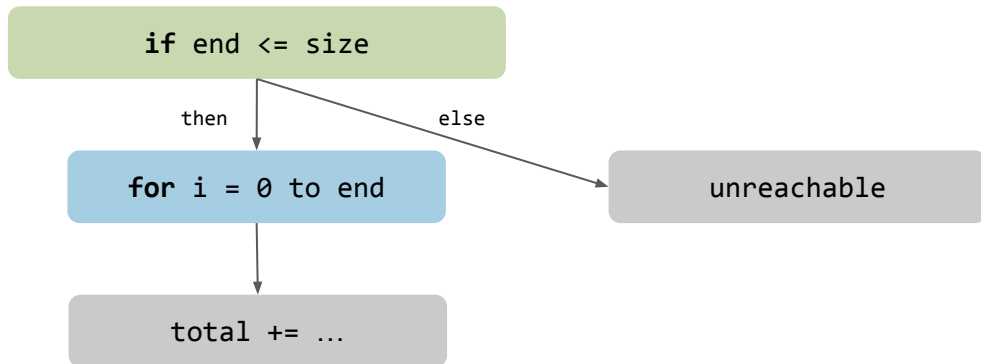


Simplifying Affine Ifs

`()[] : ()`

`()[end, size] : (end <= size)`

`(i)[end, size] : (end <= size, 0 <= i < end)`

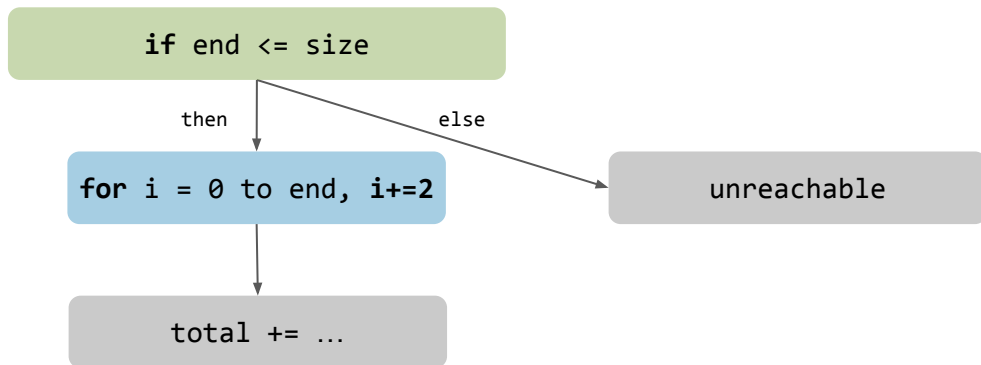


Simplifying Affine Ifs

```
()[] : ()
```

```
()[end, size] : (end <= size)
```

```
(i)[end, size] : (end <= size, 0 <= i < end  
                 i == 2*(i floordiv 2) )
```



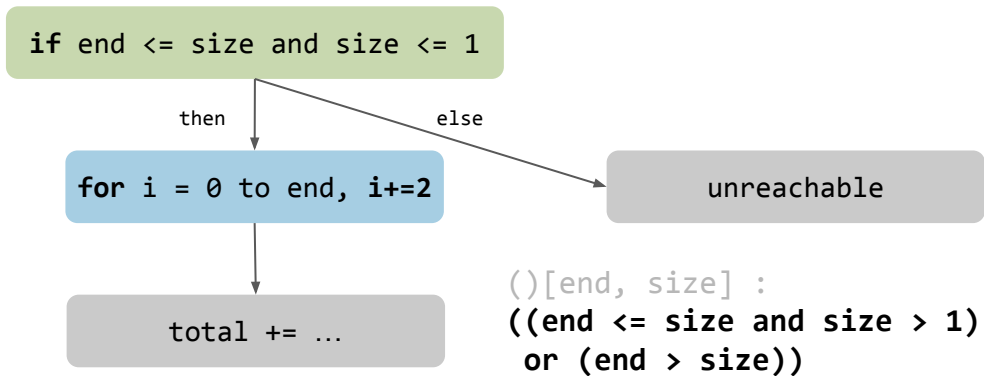
Full support for strides

Simplifying Affine Ifs

```
()[] : ()
```

```
()[end, size] : (end <= size)
```

```
(i)[end, size] : (end <= size, 0 <= i < end  
                 i == 2*(i floordiv 2) )
```



Full support for strides

Full support for analyzing the `else` branch

Code Walk: Simplifying Affine Ifs

Converting IR to Sets

```
void FlatAffineValueConstraints::addAffineIfOpDomain(AffineIfOp ifOp) {  
    // Create the base constraints from the integer set attached to ifOp.  
    FlatAffineValueConstraints cst(ifOp.getIntegerSet());  
  
    // Bind ids in the constraints to ifOp operands.  
    SmallVector<Value, 4> operands = ifOp.getOperands();  
    cst.setValues(0, cst.getNumDimAndSymbolIds(), operands);  
  
    // Merge the constraints from ifOp to the current domain. We need first merge  
    // and align the IDs from both constraints, and then append the constraints  
    // from the ifOp into the current one.  
    mergeIds(cst);  
    intersectInPlace(cst);  
}
```

simplifyGivenHolds

```
void IntegerPolyhedron::simplifyGivenHolds(const IntegerPolyhedron &cst) {  
    IntegerPolyhedron mergedCst = cst;  
    this->mergeIds(mergedCst);  
    Simplex simplex(mergedCst);  
  
    // These loop bounds change during the loop!  
    for (unsigned i = 0; i < getNumInequalities(); i++) {  
        if (simplex.isRedundantInequality(getInequality(i)))  
            removeInequality(i);  
        else  
            ++i;  
    }  
}
```

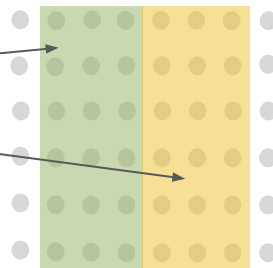
Subview Fusion in MLIR

```
%mem = memref.alloc() : memref<3x512xbf16, 1>  
%sub1 = memref.subview %mem[0, 0] [3, 256] [1, 1] : ...  
%sub2 = memref.subview %mem[0, 256] [3, 256] [1, 1] : ...
```

// Fill operation on subviews

```
linalg.fill ins(%val) outs(%sub1)
```

```
linalg.fill ins(%val) outs(%sub2)
```



$(x, y) : (0 \leq x < 3, 0 \leq y < 256)$

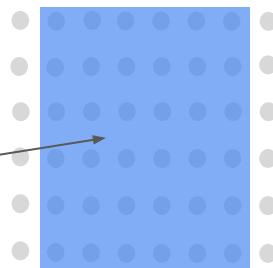
\cup

$(0 \leq x < 3, 256 \leq y < 512)$



// Fill operation on main memref

```
linalg.fill ins(%val) outs(%mem)
```



$(x, y) : (0 \leq x < 3, 0 \leq y < 512)$

Code Walk: Subview Fusion

Simplified Dependence Analysis with FPL

```

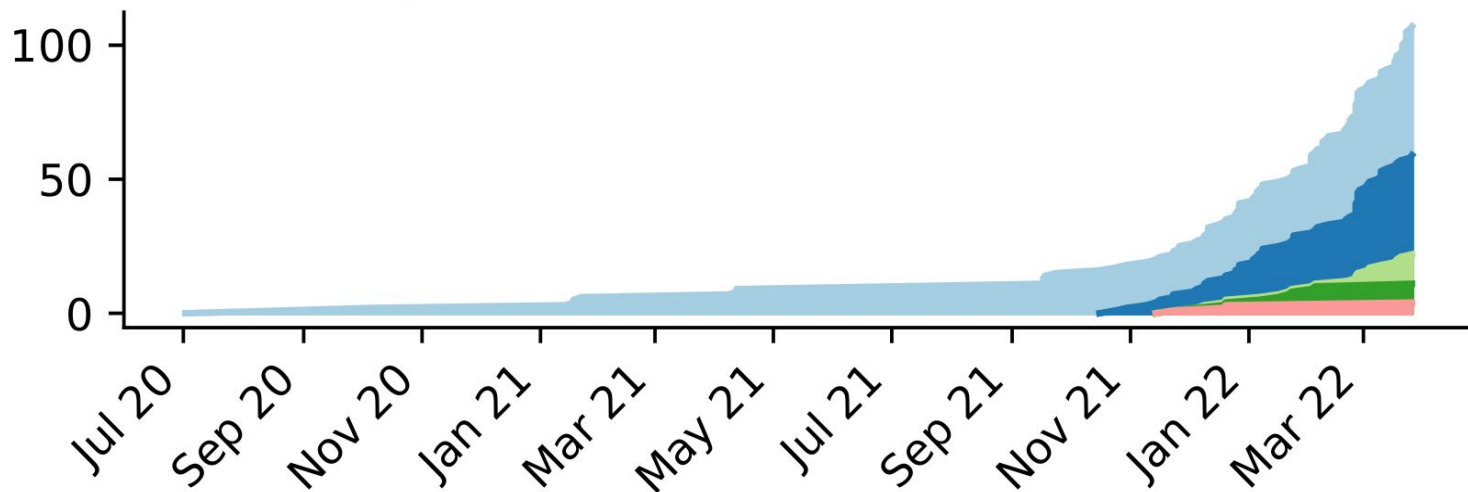
325 // Builds a map from value to identifier position in a new merged identifier
326 // list, which is the result of merging dis/symbol lists from src/dst
327 // 10 378 auto value = values[i];
328 // 379 if (isDma == FALSE || (isDma == UNKNOWN && !isForInductionVar(value)))
329 // 380
330 // 381
331 // 382 484 for (unsigned i = 0, e = dependenceConstraints->getNumDimAndSymbolIds());
332 // 383 485 {
333 // 384 486   addDomain("/isSrc=true, /isEq=true, /localOffset=0);
334 // 385 487   }
335 // 386 488   addDomain("/isSrc=true, /isEq=false, /localOffset=0);
336 // 387 489   std::vector<SmallVector<int4_t, 8>> srcFlatExprs;
337 // 388 490   std::vector<SmallVector<int4_t, 8>> dstFlatExprs;
338 // 389 491   // Uses 540 587 FLA 626 // Has equality constraints for any operands that are defined by constant ops.
339 // 390 492 // src 541 588 // 637 auto addDefForConstOperands = [](ArrayRef<Value> operands) {
340 // 391 493 static 542 589 if 638 for (auto op : operands) {
341 // 392 494 543 590 639 998 // Get composed access function for 'srcAccess'.
342 // 393 495 544 591 640 999 AffineValueMap srcAccessMap;
343 // 394 496 545 592 641 1000 srcAccess->getAccessMap(srcAccessMap);
344 // 395 497 546 // 593 642 1001
345 // 396 498 547 // 594 643 1002 // Get composed access function for 'dstAccess'.
346 // 397 499 548 // 595 644 1003 AffineValueMap dstAccessMap;
347 // 398 500 549 // 596 645 1004 dstAccess->getAccessMap(dstAccessMap);
348 // 399 501 550 // 597 646 1005
349 // 400 502 551 // 598 647 1006 // Get iteration domain for the 'srcAccess' operation.
350 // 401 503 552 // 599 648 1007 FlatAffineValueConstraints srcDomain;
351 // 402 504 553 // 600 649 1008 if (failed(getOpIndexSet(srcAccess, opInst, &srcDomain)))
352 // 403 505 554 // 601 650 1009 return DependenceResult::Failure;
353 // 404 506 555 // 602 651 1010 // Get iteration domain for 'dstAccess' operation.
354 // 405 507 556 // 603 652 // 1011 FlatAffineValueConstraints dstDomain;
355 // 406 508 557 // 604 653 1012 // 653 1012 addEq 1012
356 // 407 509 558 // 605 654 1013 if (failed(getOpIndexSet(dstAccess, opInst, &dstDomain)))
357 // 408 510 559 // 606 655 // 1014 return DependenceResult::Failure;
358 // 409 511 560 // 607 656 1015
359 // 410 512 561 // 608 657 1016
360 // 411 513 562 // 609 658 1017
361 // 412 514 563 // 610 659 1018
362 // 413 515 564 // 611 660 1019 // Return 'NoDependence' if loopDepth > numCommonLoops and if the ancestor
363 // 414 516 565 // 612 661 1020 // operation of 'srcAccess' does not properly dominate the ancestor
364 // 415 517 566 // 613 662 1021 // operation of 'dstAccess' in the same common operation block.
365 // 416 518 567 // 614 663 1022 // Note: this check is skipped if 'allowRAR' is true, because because RAR
366 // 417 519 568 // 615 664 1023 // deps can exist irrespective of lexicographic ordering b/w src and dst.
367 // 418 520 569 665 1024 unsigned numCommonLoops = getNumCommonLoops(srcDomain, dstDomain);
368 // 419 521 570 666 1025 assert(loopDepth <= numCommonLoops + 1);
369 // 420 522 571 667 1026 if (!allowRAR && loopDepth > numCommonLoops &&
370 // 421 523 572 668 1027 !srcAppearsBeforeDstInAncestralBlock(srcAccess, dstAccess, srcDomain,
371 // 422 524 573 669 1028 dstDomain, numCommonLoops)) {
372 // 423 525 574 670 1029 return DependenceResult::NoDependence;
373 // 424 526 575 671 1030 }
374 // 425 527 576 672 1031 // Build dim and symbol position maps for each access from access operand
375 // 426 528 577 673 1032 // Value to position in merged constraint system.
376 // 427 529 578 674 1033 ValuePositionMap valuePosMap;
377 // 428 530 579 675 1034 buildDimAndSymbolPositionMaps(srcDomain, dstDomain, srcAccessMap,
378 // 429 531 580 676 1035 dstAccessMap, valuePosMap,
379 // 430 532 581 677 1036 dependenceConstraints);
380 // 431 533 582 678 1037 for (int32_t i = 0; i < dependenceConstraints->getNumDimAndSymbolIds(); ++i)
381 // 432 534 583 679 1038 initDependenceConstraints(srcDomain, dstDomain, srcAccessMap, dstAccessMap,
382 // 433 535 584 680 1039 valuePosMap, dependenceConstraints);
383 // 434 536 585 681 1040
384 // 435 537 586 682 1041 assert(valuePosMap.getNumDims() ==
385 // 436 538 587 683 1042 srcDomain.getNumDims() + dstDomain.getNumDims());
386 // 437 539 588 684 1043 // Create memref access constraint by equating src/dst access functions.
387 // 438 540 589 685 1044 // Note that this check is conservative, and will fail in the future when
388 // 439 541 590 686 1045 // local variables for mod/div exprs are supported.
389 // 440 542 591 687 1046 if (failed(addMemRefAccessConstraints(srcAccessMap, dstAccessMap, valuePosMap,
390 // 441 543 592 688 1047 dependenceConstraints)))
391 // 442 544 593 689 1048 return DependenceResult::Failure;
392 // 443 545 594 690 1049
393 // 444 546 595 691 1050
394 // 445 547 596 692 1051
395 // 446 548 597 693 1052
396 // 447 549 598 694 1053
397 // 448 550 599 695 1054
398 // 449 551 600 696 1055
399 // 450 552 601 697 1056
400 // 451 553 602 698 1057
401 // 452 554 603 699 1058
402 // 453 555 604 700 1059
403 // 454 556 605 701 1060
404 // 455 557 606 702 1061
405 // 456 558 607 703 1062
406 // 457 559 608 704 1063
407 // 458 560 609 705 1064
408 // 459 561 610 706 1065
409 // 460 562 611 707 1066
410 // 461 563 612 708 1067
411 // 462 564 613 709 1068
412 // 463 565 614 710 1069
413 // 464 566 615 711 1070
414 // 465 567 616 712 1071
415 // 466 568 617 713 1072
416 // 467 569 618 714 1073
417 // 468 570 619 715 1074
418 // 469 571 620 716 1075
419 // 470 572 621 717 1076
420 // 471 573 622 718 1077
421 // 472 574 623 719 1078
422 // 473 575 624 720 1079
423 // 474 576 625 721 1080
424 // 475 577 626 722 1081
425 // 476 578 627 723 1082
426 // 477 579 628 724 1083
427 // 478 580 629 725 1084
428 // 479 581 630 726 1085
429 // 480 582 631 727 1086
430 // 481 583 632 728 1087
431 // 482 584 633 729 1088
432 // 483 585 634 730 1089
433 // 484 586 635 731 1090
434 // 485 587 636 732 1091
435 // 486 588 637 733 1092
436 // 487 589 638 734 1093
437 // 488 590 639 735 1094
438 // 489 591 640 736 1095
439 // 490 592 641 737 1096
440 // 491 593 642 738 1097
441 // 492 594 643 739 1098
442 // 493 595 644 740 1099
443 // 494 596 645 741 1100
444 // 495 597 646 742 1101
445 // 496 598 647 743 1102
446 // 497 599 648 744 1103
447 // 498 600 649 745 1104
448 // 499 601 650 746 1105
449 // 500 602 651 747 1106
450 // 501 603 652 748 1107
451 // 502 604 653 749 1108
452 // 503 605 654 750 1109
453 // 504 606 655 751 1110
454 // 505 607 656 752 1111
455 // 506 608 657 753 1112
456 // 507 609 658 754 1113
457 // 508 610 659 755 1114
458 // 509 611 660 756 1115
459 // 510 612 661 757 1116
460 // 511 613 662 758 1117
461 // 512 614 663 759 1118
462 // 513 615 664 760 1119
463 // 514 616 665 761 1120
464 // 515 617 666 762 1121
465 // 516 618 667 763 1122
466 // 517 619 668 764 1123
467 // 518 620 669 765 1124
468 // 519 621 670 766 1125
469 // 520 622 671 767 1126
470 // 521 623 672 768 1127
471 // 522 624 673 769 1128
472 // 523 625 674 770 1129
473 // 524 626 675 771 1130
474 // 525 627 676 772 1131
475 // 526 628 677 773 1132
476 // 527 629 678 774 1133
477 // 528 630 679 775 1134
478 // 529 631 680 776 1135
479 // 530 632 681 777 1136
480 // 531 633 682 778 1137
481 // 532 634 683 779 1138
482 // 533 635 684 780 1139
483 // 534 636 685 781 1140
484 // 535 637 686 782 1141
485 // 536 638 687 783 1142
486 // 537 639 688 784 1143
487 // 538 640 689 785 1144
488 // 539 641 690 786 1145
489 // 540 642 691 787 1146
490 // 541 643 692 788 1147
491 // 542 644 693 789 1148
492 // 543 645 694 790 1149
493 // 544 646 695 791 1150
494 // 545 647 696 792 1151
495 // 546 648 697 793 1152
496 // 547 649 698 794 1153
497 // 548 650 699 795 1154
498 // 549 651 700 796 1155
499 // 550 652 701 797 1156
500 // 551 653 702 798 1157
501 // 552 654 703 799 1158
502 // 553 655 704 800 1159
503 // 554 656 705 801 1160
504 // 555 657 706 802 1161
505 // 556 658 707 803 1162
506 // 557 659 708 804 1163
507 // 558 660 709 805 1164
508 // 559 661 710 806 1165
509 // 560 662 711 807 1166
510 // 561 663 712 808 1167
511 // 562 664 713 809 1168
512 // 563 665 714 810 1169
513 // 564 666 715 811 1170
514 // 565 667 716 812 1171
515 // 566 668 717 813 1172
516 // 567 669 718 814 1173
517 // 568 670 719 815 1174
518 // 569 671 720 816 1175
519 // 570 672 721 817 1176
520 // 571 673 722 818 1177
521 // 572 674 723 819 1178
522 // 573 675 724 820 1179
523 // 574 676 725 821 1180
524 // 575 677 726 822 1181
525 // 576 678 727 823 1182
526 // 577 679 728 824 1183
527 // 578 680 729 825 1184
528 // 579 681 730 826 1185
529 // 580 682 731 827 1186
530 // 581 683 732 828 1187
531 // 582 684 733 829 1188
532 // 583 685 734 830 1189
533 // 584 686 735 831 1190
534 // 585 687 736 832 1191
535 // 586 688 737 833 1192
536 // 587 689 738 834 1193
537 // 588 690 739 835 1194
538 // 589 691 740 836 1195
539 // 590 692 741 837 1196
540 // 591 693 742 838 1197
541 // 592 694 743 839 1198
542 // 593 695 744 840 1199
543 // 594 696 745 841 1200
544 // 595 697 746 842 1201
545 // 596 698 747 843 1202
546 // 597 699 748 844 1203
547 // 598 700 749 845 1204
548 // 599 701 750 846 1205
549 // 600 702 751 847 1206
550 // 601 703 752 848 1207
551 // 602 704 753 849 1208
552 // 603 705 754 850 1209
553 // 604 706 755 851 1210
554 // 605 707 756 852 1211
555 // 606 708 757 853 1212
556 // 607 709 758 854 1213
557 // 608 710 759 855 1214
558 // 609 711 760 856 1215
559 // 610 712 761 857 1216
560 // 611 713 762 858 1217
561 // 612 714 763 859 1218
562 // 613 715 764 860 1219
563 // 614 716 765 861 1220
564 // 615 717 766 862 1221
565 // 616 718 767 863 1222
566 // 617 719 768 864 1223
567 // 618 720 769 865 1224
568 // 619 721 770 866 1225
569 // 620 722 771 867 1226
570 // 621 723 772 868 1227
571 // 622 724 773 869 1228
572 // 623 725 774 870 1229
573 // 624 726 775 871 1230
574 // 625 727 776 872 1231
575 // 626 728 777 873 1232
576 // 627 729 778 874 1233
577 // 628 730 779 875 1234
578 // 629 731 780 876 1235
579 // 630 732 781 877 1236
580 // 631 733 782 878 1237
581 // 632 734 783 879 1238
582 // 633 735 784 880 1239
583 // 634 736 785 881 1240
584 // 635 737 786 882 1241
585 // 636 738 787 883 1242
586 // 637 739 788 884 1243
587 // 638 740 789 885 1244
588 // 639 741 790 886 1245
589 // 640 742 791 887 1246
590 // 641 743 792 888 1247
591 // 642 744 793 889 1248
592 // 643 745 794 890 1249
593 // 644 746 795 891 1250
594 // 645 747 796 892 1251
595 // 646 748 797 893 1252
596 // 647 749 798 894 1253
597 // 648 750 799 895 1254
598 // 649 751 800 896 1255
599 // 650 752 801 897 1256
600 // 651 753 802 898 1257
601 // 652 754 803 899 1258
602 // 653 755 804 900 1259
603 // 654 756 805 901 1260
604 // 655 757 806 902 1261
605 // 656 758 807 903 1262
606 // 657 759 808 904 1263
607 // 658 760 809 905 1264
608 // 659 761 810 906 1265
609 // 660 762 811 907 1266
610 // 661 763 812 908 1267
611 // 662 764 813 909 1268
612 // 663 765 814 910 1269
613 // 664 766 815 911 1270
614 // 665 767 816 912 1271
615 // 666 768 817 913 1272
616 // 667 769 818 914 1273
617 // 668 770 819 915 1274
618 // 669 771 820 916 1275
619 // 670 772 821 917 1276
620 // 671 773 822 918 1277
621 // 672 774 823 919 1278
622 // 673 775 824 920 1279
623 // 674 776 825 921 1280
624 // 675 777 826 922 1281
625 // 676 778 827 923 1282
626 // 677 779 828 924 1283
627 // 678 780 829 925 1284
628 // 679 781 830 926 1285
629 // 680 782 831 927 1286
630 // 681 783 832 928 1287
631 // 682 784 833 929 1288
632 // 683 785 834 930 1289
633 // 684 786 835 931 1290
634 // 685 787 836 932 1291
635 // 686 788 837 933 1292
636 // 687 789 838 934 1293
637 // 688 790 839 935 1294
638 // 689 791 840 936 1295
639 // 690 792 841 937 1296
640 // 691 793 842 938 1297
641 // 692 794 843 939 1298
642 // 693 795 844 940 1299
643 // 694 796 845 941 1300
644 // 695 797 846 942 1301
645 // 696 798 847 943 1302
646 // 697 799 848 944 1303
647 // 698 800 849 945 1304
648 // 699 801 850 946 1305
649 // 700 802 851 947 1306
650 // 701 803 852 948 1307
651 // 702 804 853 949 1308
652 // 703 805 854 950 1309
653 // 704 806 855 951 1310
654 // 705 807 856 952 1311
655 // 706 808 857 953 1312
656 // 707 809 858 954 1313
657 // 708 810 859 955 1314
658 // 709 811 860 956 1315
659 // 710 812 861 957 1316
660 // 711 813 862 958 1317
661 // 712 814 863 959 1318
662 // 713 815 864 960 1319
663 // 714 816 865 961 1320
664 // 715 817 866 962 1321
665 // 716 818 867 963 1322
666 // 717 819 868 964 1323
667 // 718 820 869 965 1324
668 // 719 821 870 966 1325
669 // 720 822 871 967 1326
670 // 721 823 872 968 1327
671 // 722 824 873 969 1328
672 // 723 825 874 970 1329
673 // 724 826 875 971 1330
674 // 725 827 876 972 1331
675 // 726 828 877 973 1332
676 // 727 829 878 974 1333
677 // 728 830 879 975 1334
678 // 729 831 880 976 1335
679 // 730 832 881 977 1336
680 // 731 833 882 978 1337
681 // 732 834 883 979 1338
682 // 733 835 884 980 1339
683 // 734 836 885 981 1340
684 // 735 837 886 982 1341
685 // 736 838 887 983 1342
686 // 737 839 888 984 1343
687 // 738 840 889 985 1344
688 // 739 841 890 986 1345
689 // 740 842 891 987 1346
690 // 741 843 892 988 1347
691 // 742 844 893 989 1348
692 // 743 845 894 990 1349
693 // 744 846 895 991 1350
694 // 745 847 896 992 1351
695 // 746 848 897 993 1352
696 // 747 849 898 994 1353
697 // 748 850 899 995 1354
698 // 749 851 900 996 1355
699 // 750 852 901 997 1356
700 // 751 853 902 998 1357
701 // 752 854 903 999 1358
702 // 753 855 904 1000 1359
703 // 754 856 905 1001 1360
704 // 755 857 906 1002 1361
705 // 756 858 907 1003 1362
706 // 757 859 908 1004 1363
707 // 758 860 909 1005 1364
708 // 759 861 910 1006 1365
709 // 760 862 911 1007 1366
710 // 761 863 912 1008 1367
711 // 762 864 913 1009 1368
712 // 763 865 914 1010 1369
713 // 764 866 915 1011 1370
714 // 765 867 916 1012 1371
715 // 766 868 917 1013 1372
716 // 767 869 918 1014 1373
717 // 768 870 919 1015 1374
718 // 769 871 920 1016 1375
719 // 770 872 921 1017 1376
720 // 771 873 922 1018 1377
721 // 772 874 923 1019 1378
722 // 773 875 924 1020 1379
723 // 774 876 925 1021 1380
724 // 775 877 926 1022 1381
725 // 776 878 927 1023 1382
726 // 777 879 928 1024 1383
727 // 778 880 929 1025 1384
728 // 779 881 930 1026 1385
729 // 780 882 931 1027 1386
730 // 781 883 932 1028 1387
731 // 782 884 933 1029 1388
732 // 783 885 934 1030 1389
733 // 784 886 935 1031 1390
734 // 785 887 936 1032 1391
735 // 786 888 937 1033 1392
736 // 787 889 938 1034 1393
737 // 788 890 939 1035 1394
738 // 789 891 940 1036 1395
739 // 790 892 941 1037 1396
740 // 791 893 942 1038 1397
741 // 792 894 943 1039 1398
742 // 793 895 944 1040 1399
743 // 794 896 945 1041 1400
744 // 795 897 946 1042 1401
745 // 796 898 947 1043 1402
746 // 797 899 948 1044 1403
747 // 798 900 949 1045 1404
748 // 799 901 950 1046 1405
749 // 800 902 951 1047 1406
750 // 801 903 952 1048 1407
751 // 802 904 953 1049 1408
752 // 803 905 954 1050 1409
753 // 804 906 955 1051 1410
754 // 805 907 956 1052 1411
755 // 806 908 957 1053 1412
756 // 807 909 958 1054 1413
757 // 808 910 959 1055 1414
758 // 809 911 960 1056 1415
759 // 810 912 961 1057 1416
760 // 811 913 962 1058 1417
761 // 812 914 963 1059 1418
762 // 813 915 964 1060 1419
763 // 814 916 965 1061 1420
764 // 815 917 966 1062 1421
765 // 816 918 967 1063 1422
766 // 817 919 968 1064 1423
767 // 818 9
```

FPL: A Complete Set of Operations

	union	intersect	subtract	equality	emptiness	lexmin
MLIR (before)	no	no local ID support	no	no	inexact heuristic	no
MLIR's FPL	yes	yes	yes	yes	yes	yes

FPL's Growing Developer Community

Number of landed patches

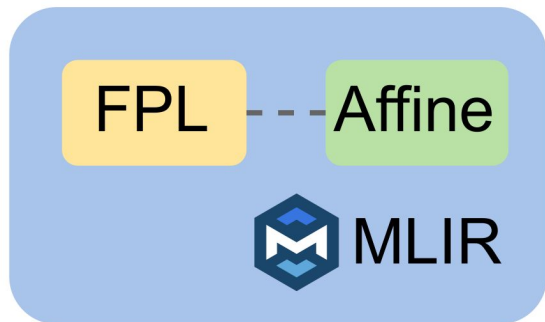
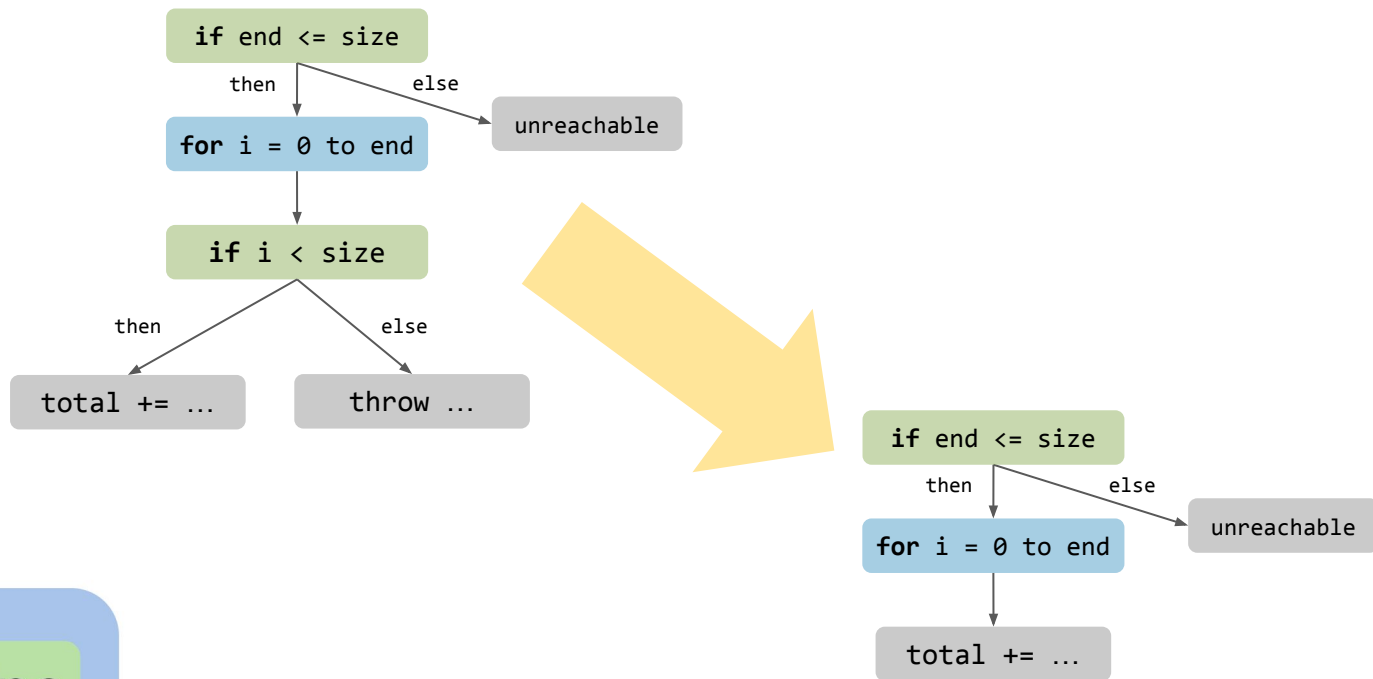


Available in the monorepo today!

The screenshot shows the LLVM monorepo GitHub repository page. The repository is named `llvm-project` and is public. It has 545 watchers. The navigation bar includes links for Code, Issues (5k+), Pull requests, Actions, Security, and Insights. The breadcrumb trail is `main > llvm-project / mlir / include / mlir / Analysis / Presburger /`. The main content area shows a commit by `Groverkss` titled `[MLIR][Presburger] Remove inheritance in MultiAffineFunction`. Below the commit, there is a list of files in the `Presburger` namespace, each with a description of the change.

File	Change Description
<code>Fraction.h</code>	<code>[MLIR][Presburger]</code> Move Presburger/ files to presburger namespace
<code>IntegerRelation.h</code>	<code>[MLIR][Presburger]</code> Remove inheritance in MultiAffineFunction
<code>LinearTransform.h</code>	<code>[MLIR][Presburger]</code> Move functionality from IntegerPolyhedron to Integer...
<code>Matrix.h</code>	<code>[MLIR][Presburger][Simplex]</code> symbolic lexmin: add some normalization h...
<code>PWMAFunction.h</code>	<code>[MLIR][Presburger]</code> Remove inheritance in MultiAffineFunction
<code>PresburgerRelation.h</code>	<code>[MLIR][Presburger]</code> Remove inheritance from PresburgerSpace in Integer...
<code>PresburgerSpace.h</code>	<code>[MLIR][Presburger]</code> Remove inheritance from PresburgerSpace in Integer...
<code>Simplex.h</code>	<code>[MLIR][Presburger][Simplex]</code> moveRowUnknownToColumn: support the row s...
<code>Utils.h</code>	<code>[MLIR][Presburger]</code> Remove inheritance in MultiAffineFunction

Conclusion



grosser.science/FPL