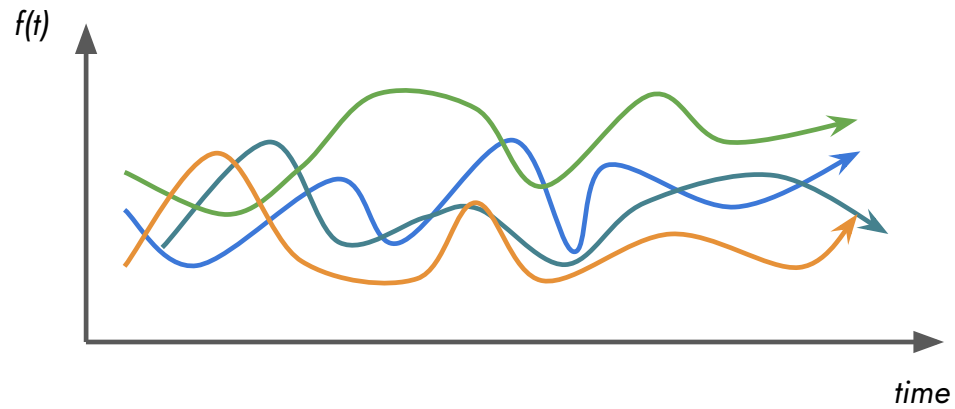


Hallucination-Resilient LLM-Driven Sound and Tunable Static Analysis

Guannan Wei, Zhuo Zhang, Caterina Urban
LMPL @ SPLASH/ICFP 2025, Singapore

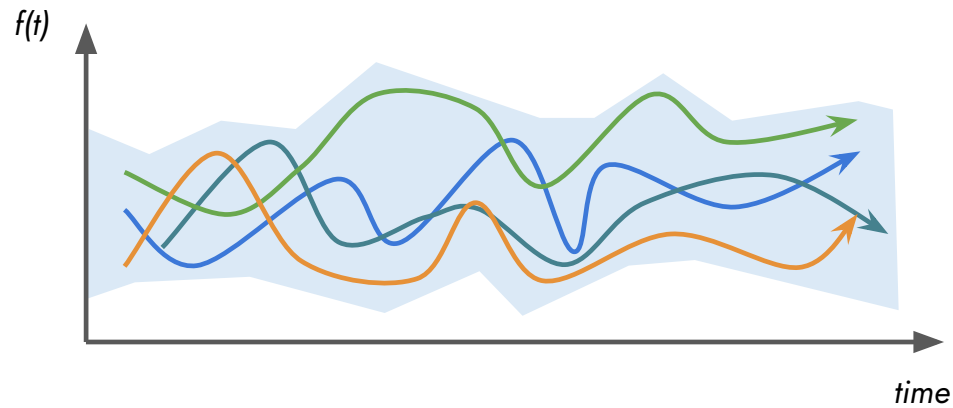


What is Static Analysis?



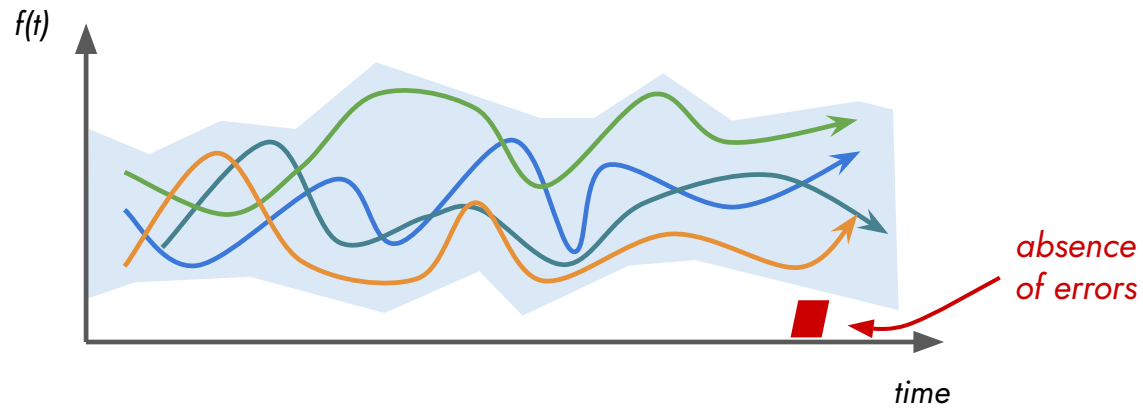
concrete traces

What is Static Analysis?



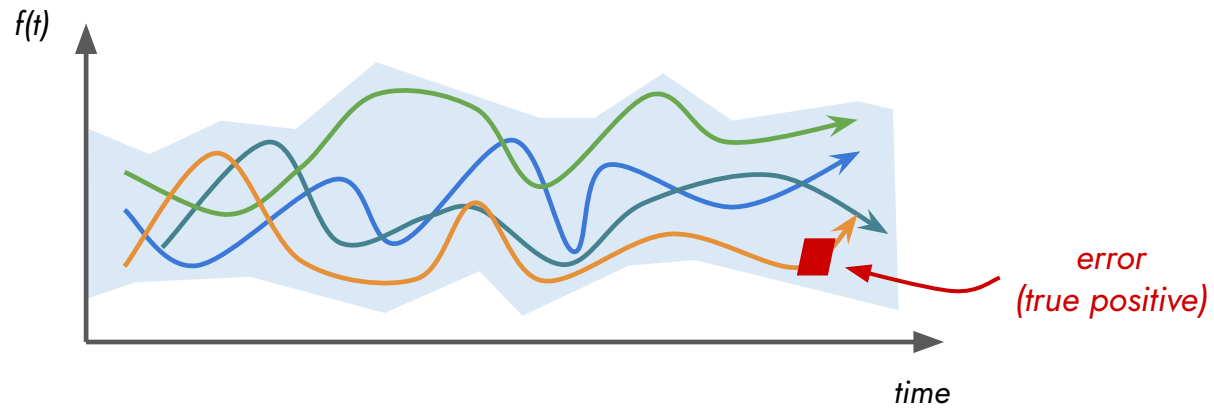
*soundness:
over-approximations*

What is Static Analysis?



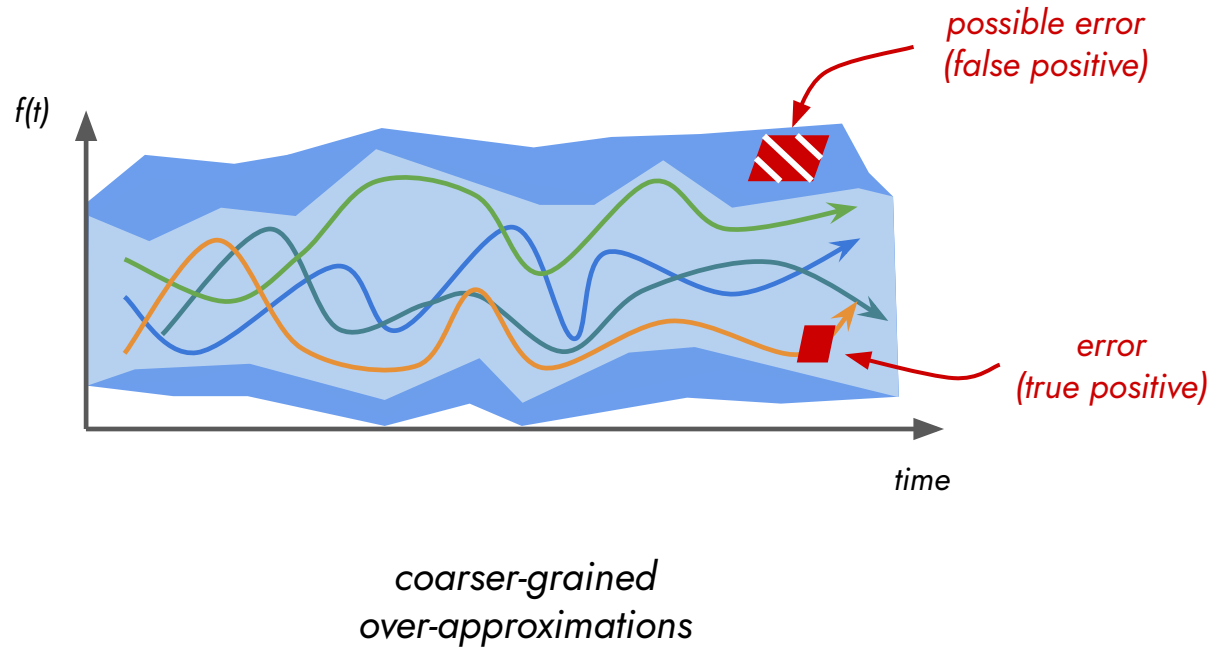
*soundness:
over-approximations*

What is Static Analysis?

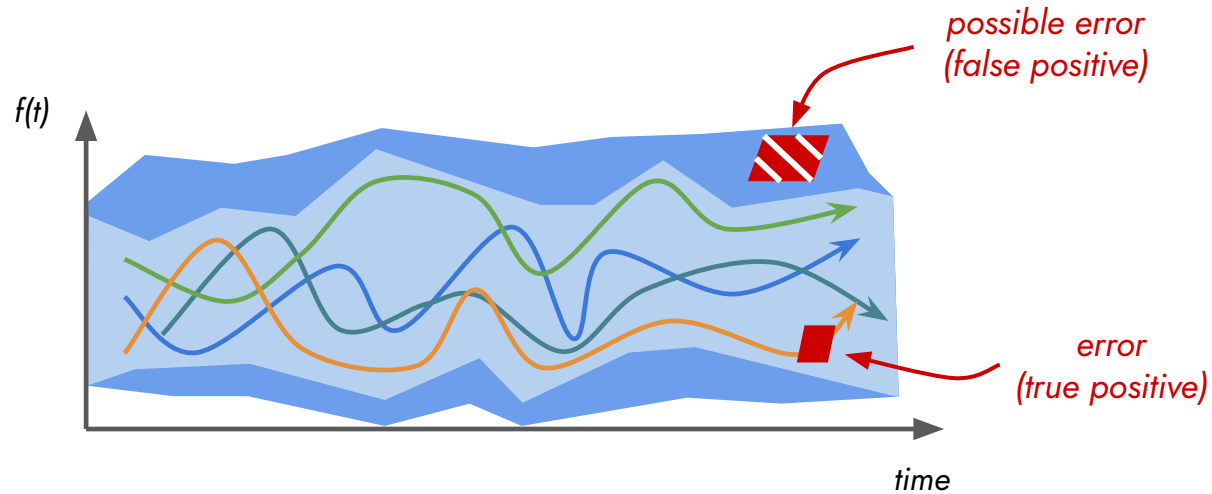


*soundness:
over-approximations*

What is Static Analysis?



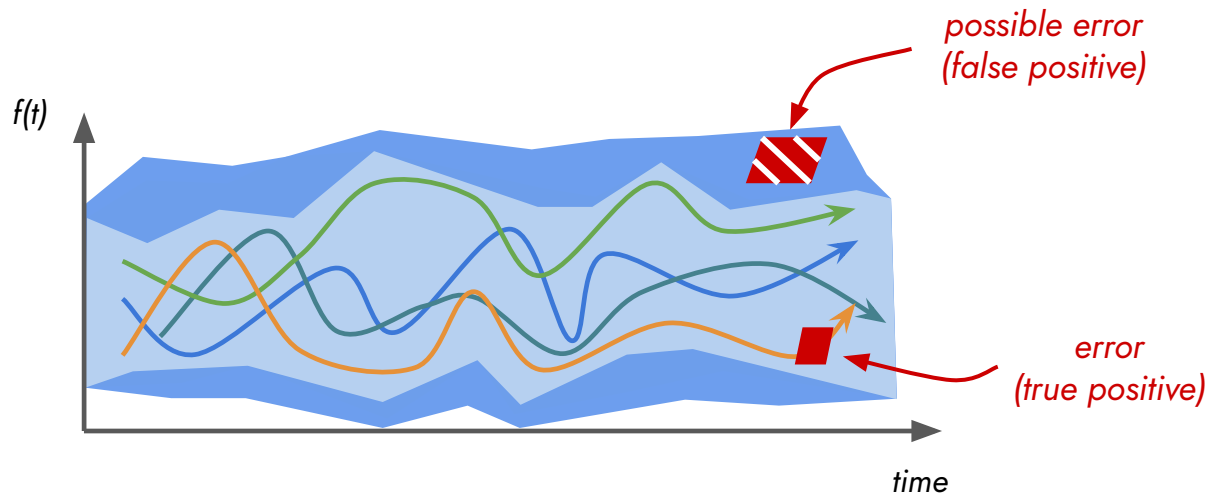
What is Static Analysis?



Goals

- Soundness:** detect all true errors
- Precision:** reduce false errors

What is Static Analysis?



So, you want to use LLMs for static analysis, but how?

Prompting LLMs

```
J demo.java 1 X
Users > xiangqian > Downloads > J demo.java > ...
1  import java.util.ArrayList;
2
3  public class Demo {
4
5      public static int foo(int a, int b) {
6          if (Math.abs(b) > 1)
7              System.out.println(a / b);
8          return b;
9      }
10
11      Run | Debug
12      public static void main(String[] args) {
13          int x = 1;
14          int y = 10 / x;
15          ArrayList<Integer> list = new ArrayList<>();
16          list.add(x);
17          list.add(y);
18          list.add(e:0);
19          y = x * 1;
20          int z = x / y;
21          z = list.get(index:2);
22          System.out.println(y / z);
23
24          x = foo(y, z);
25          System.out.println(x / x);
26      }
```

Claude-3.5-Sonnet

Prompting LLMs

```
J demo.java 1 X
Users > xiangqian > Downloads > J demo.java > ...
1  import java.util.ArrayList;
2
3  public class Demo {
4
5      public static int foo(int a, int b) {
6          if (Math.abs(b) > 1)
7              System.out.println(a / b);
8          return b;
9      }
10
11      Run | Debug
12      public static void main(String[] args) {
13          int x = 1;
14          int y = 10 / x;
15          ArrayList<Integer> list = new ArrayList<>();
16          list.add(x);
17          list.add(y);
18          list.add(e:0);
19          y = x * 1;
20          int z = x / y;
21          z = list.get(index:2);
22          System.out.println(y / z);
23
24          x = foo(y, z);
25          System.out.println(x / x);
26      }
27  }
```

CHAT

Please mimic the execution of the program and detect divide-by-zero bugs.

J demo.java:1-26

prompt

Claude-3.5-Sonnet

Prompting LLMs

```
J demo.java 1 X
Users > xiangqian > Downloads > J demo.java > ...
1  import java.util.ArrayList;
2
3  public class Demo {
4
5      public static int foo(int a, int b) {
6          if (Math.abs(b) > 1)
7              System.out.println(a / b);
8          return b;
9      }
10
11      Run | Debug
12      public static void main(String[] args) {
13          int x = 1;
14          int y = 10 / x;
15          ArrayList<Integer> list = new ArrayList<>();
16          list.add(x);
17          list.add(y);
18          list.add(e:0);
19          y = x * 1;
20          int z = x / y;
21          z = list.get(index:2);
22          System.out.println(y / z);
23
24          x = foo(y, z);
25          System.out.println(x / x);
26      }
```

CHAT

Please mimic the execution of the program and detect divide-by-zero bugs.

J demo.java:1-26

GitHub Copilot

> Used 1 reference

I'll analyze the code execution step by step and identify potential divide-by-zero bugs.

Analysis

1. In `main()` :

- `x = 1`
- `y = 10 / x` → `y = 10` (safe, `x = 1`)
- Create ArrayList and add: `[1, 10, 0]`
- `y = x * 1` → `y = 1`
- `z = x / y` → `z = 1` (safe, `y = 1`)
- `z = list.get(2)` → `z = 0`
- `System.out.println(y / z)` → **DIVIDE BY ZERO ERROR!** (`y = 1, z = 0`)
- Program would crash here, but if it continued:
- `x = foo(y, z)` with `y = 1, z = 0`
- Inside `foo()` : doesn't print since `Math.abs(0) ≤ 1`
- Returns `z` which is 0
- `System.out.println(x / x)` → **DIVIDE BY ZERO ERROR!** (`x = 0`)

Divide-by-Zero Bugs Found:

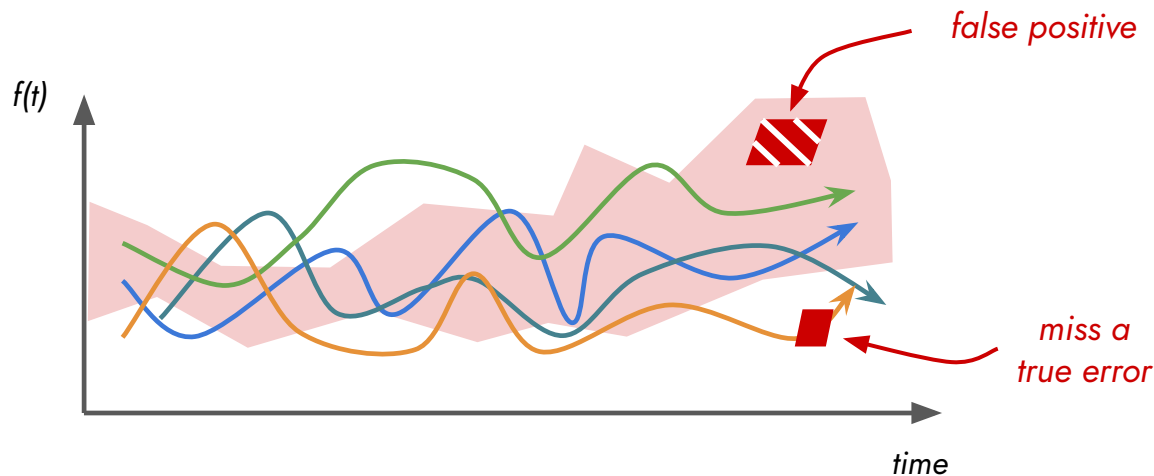
- Line 21: `System.out.println(y / z)` - when `z = 0`
- Line 24: `System.out.println(x / x)` - when `x = 0`

prompt

*reasoning &
analysis result*

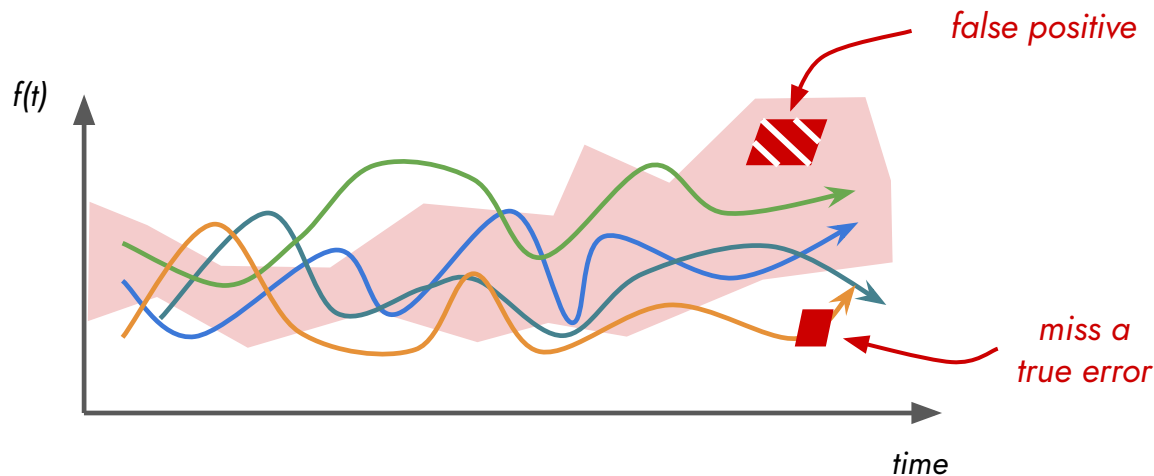
Claude-3.5-Sonnet

Can we trust LLM-reported analysis results?



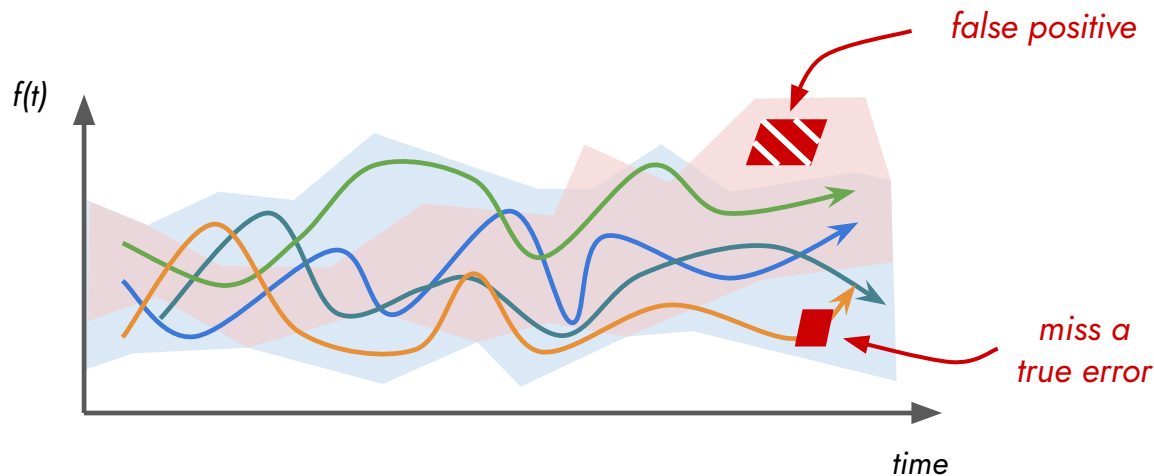
*LLMs are untrustworthy because of **hallucination**.*

Can we trust LLM-reported analysis results?



Can we *validate* the analysis results from LLMs?

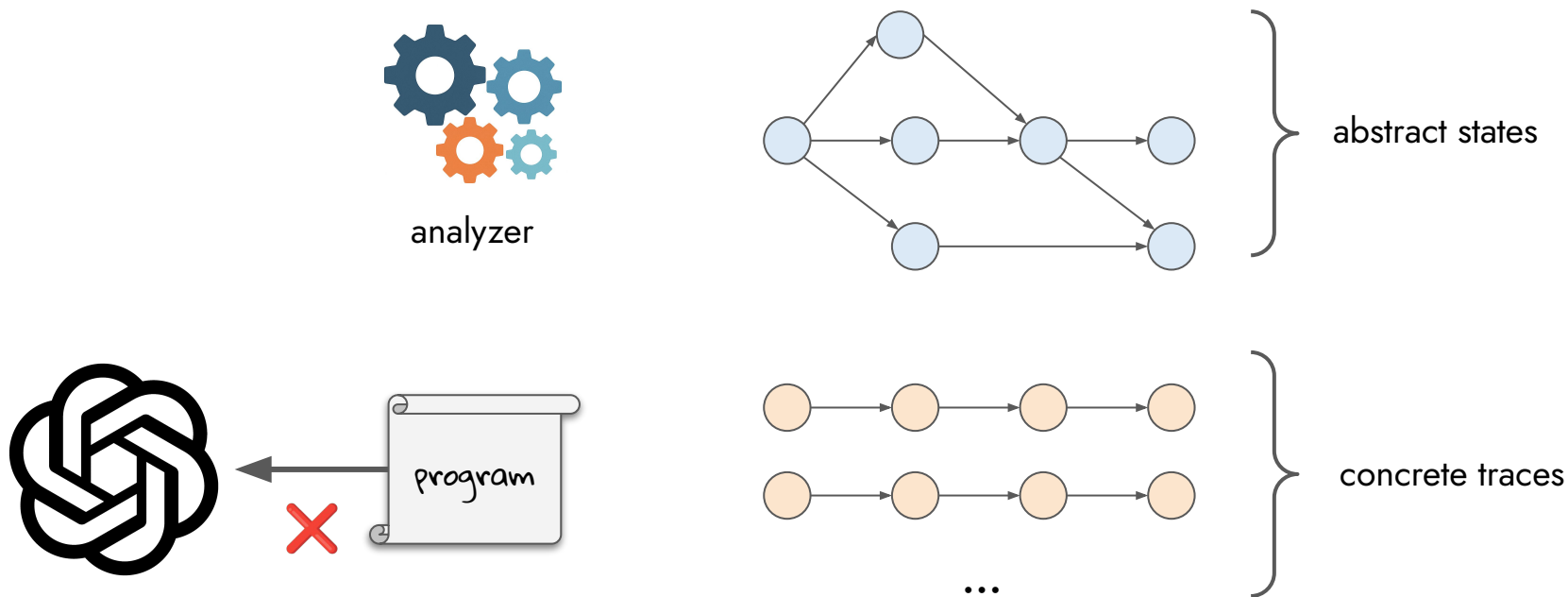
Can we trust LLM-reported analysis results?



Can we *validate* the analysis results from LLMs?
Yes, but this becomes a static analysis problem...

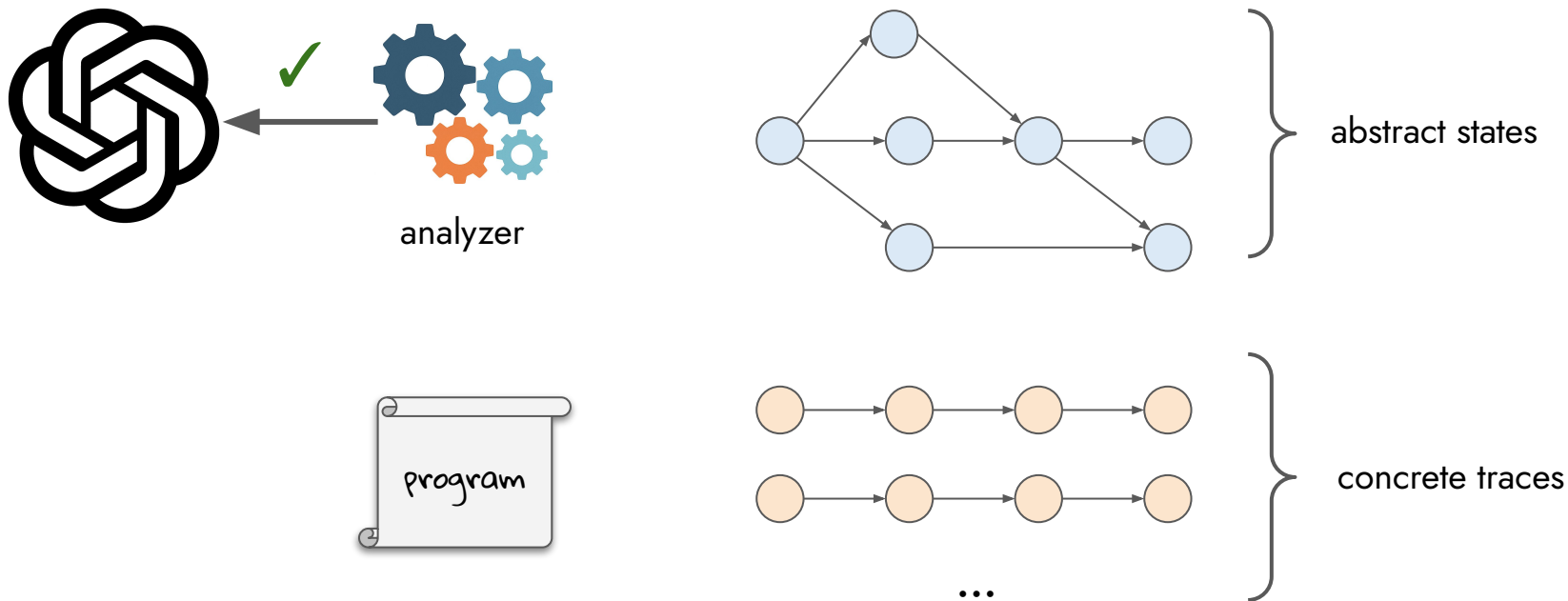
Our Idea: LLMs as Meta-Analysis

Instead of analyzing target programs, use LLMs to “analyze” the analysis!



Our Idea: LLMs as Meta-Analysis

Instead of analyzing target programs, use LLMs to “analyze” the analysis!



Our Idea: LLMs as Meta-Analysis

Recipe:

- **Identify precision-critical decisions in the static analysis**
 - Any decision does not hinder soundness,
 - but only impacts precision
- **During the analysis, use LLMs as the decision maker**
 - Prompt describes the process/implementation of static analysis
 - Prompt contains current abstract state and other (meta-) information
 - Prompt instructs to decide for better precision

Our Idea: LLMs as Meta-Analysis

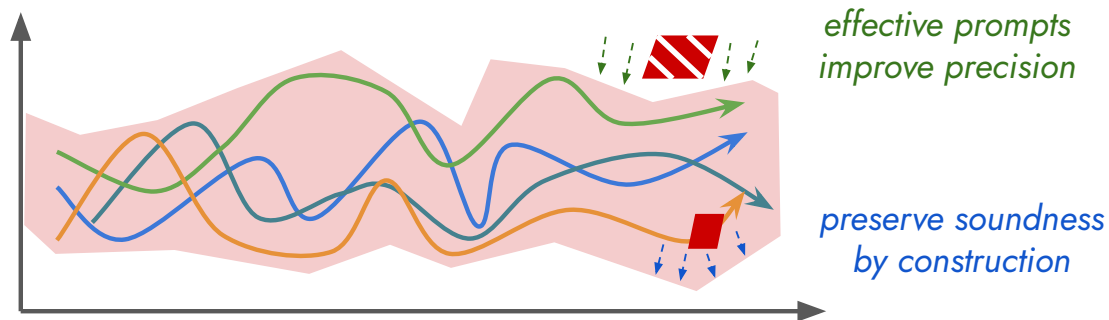
Recipe:

- Identify precision-critical decisions in the static analysis
 - Any decision does not hinder soundness,
 - but only impacts precision
- **During the analysis, use LLMs as the decision maker**
 - Prompt describes the process/implementation of static analysis
 - Prompt contains current *abstract* state and other (meta-) information
 - Prompt instructs to decide for better precision

Our Idea: LLMs as Meta-Analysis

Recipe:

- **Identify precision-critical decisions in the static analysis**
 - Any outcome does not hinder soundness,
 - but only impacts on precision
- **During the analysis, use LLMs as the decision maker**
 - Prompt describes the process/implementation of static analysis
 - Prompt contains current abstract state and other (meta-) information
 - Prompt instructs to decide for precision



A Case of Higher-Order Control-Flow Analysis

- Control flow analysis:
 - determines the function to be called at each call-site
 - undecidable problem once data- and control-flow are entangled
(e.g. when with first-class functions in Scheme, dynamic dispatch in Java, indirect call with pointers in C, etc.)
- Abstracting abstract machine (AAM) framework [ICFP '10]
 - analysis is defined as an abstraction of a concrete abstract machine (e.g. CEK machine)
$$\Sigma \in \widehat{\text{State}} \triangleq (\text{Expr} + \widehat{\text{Value}}) \times \widehat{\text{Env}} \times \widehat{\text{BStore}} \times \widehat{\text{KStore}} \times \widehat{\text{Kont}} \times \widehat{\text{Time}}$$
 - result is an over-approximated transition graph of abstract states

A Case of Higher-Order Control-Flow Analysis

- **[Recipe step 1]** Identify precision-critical decisions in the static analysis

A Case of Higher-Order Control-Flow Analysis

- [Recipe step 1] Identify precision-critical decisions in the static analysis

$tick : \widehat{\text{State}} \rightarrow \widehat{\text{Time}}$

$alloc_v : \widehat{\text{State}} \times \text{Var} \times \widehat{\text{Time}} \rightarrow \widehat{\text{BAddr}}$

$alloc_\kappa : \widehat{\text{State}} \times \text{Expr} \times \widehat{\text{Env}} \times \widehat{\text{BStore}} \times \widehat{\text{Time}} \rightarrow \widehat{\text{KAddr}}$

LLM as abstract address allocator

A Case of Higher-Order Control-Flow Analysis

- [Recipe step 1] Identify precision-critical decisions in the static analysis

$$tick : \widehat{\text{State}} \rightarrow \widehat{\text{Time}}$$

$$alloc_v : \widehat{\text{State}} \times \text{Var} \times \widehat{\text{Time}} \rightarrow \widehat{\text{BAddr}}$$

$$alloc_\kappa : \widehat{\text{State}} \times \text{Expr} \times \widehat{\text{Env}} \times \widehat{\text{BStore}} \times \widehat{\text{Time}} \rightarrow \widehat{\text{KAddr}}$$

LLM as abstract address allocator

Soundness-safe:

Any allocation strategy from a finite set is sound (a Posteriori Soundness Theorem)

Precision-critical:

Addresses decide the degree of approximation (e.g. context-sensitivity, ...)

A Case of Higher-Order Control-Flow Analysis

- **[Recipe step 2]** During the analysis, use LLMs as the decision maker.

[illegible]

A Case of Higher-Order Control-Flow Analysis

- **[Recipe step 2]** During the analysis, use LLMs as the decision maker.

The "State" field is an abstract state, which can be one of the following:

|enum State:

```
| case EState(e: Expr, ρ: Env, σv: BStore, σk: KStore, k: Kont, t: Time)
```

```
case VState(v: Value, ρ: Env, σv: BStore, σk: KStore, k: Kont, t: Time)
```

```
| case ErrState()
```

Describe data structures used in the analyzer

[illegible][illegible]

A Case of Higher-Order Control-Flow Analysis

- [Recipe step 2] During the analysis, use LLMs as the decision maker.

Your input is a JSON object with the following schema:

```
{
  "state": State,
  "query-type": "BAddr" or "KAddr" or "Tick",
  "variable": String, // for BAddr only
  "time": Time, // for BAddr, KAddr, and "Tick"
  "source-expression": Expr, // for KAddr only
  "target-expression": Expr, // for KAddr only
  "target-environment": Env, // for KAddr only
  "target-binding-store": BStore, // for KAddr only
}
```

Describe input JSON format for each query.

```
NOT (empty) STRING = ""
// This is an expert in static analysis and abstract interpretation, specifically in the "abstracting abstract machine" approach.
// Your task is to analyze the program of a small-time abstract interpreter that transforms an abstract machine state into possible concrete states.
// It will provide you the data structures used in the abstract interpretation framework.
// Then give your the current analysis state, and you to analyze it, and you to decide the context-sensitivity or
// create an abstract address used in the abstraction.
0
// The "State" field is an abstract state, which can be one of the following:
// enum State {
//   CASE STATE0: STATE, // 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50, 51, 52, 53, 54, 55, 56, 57, 58, 59, 60, 61, 62, 63, 64, 65, 66, 67, 68, 69, 70, 71, 72, 73, 74, 75, 76, 77, 78, 79, 80, 81, 82, 83, 84, 85, 86, 87, 88, 89, 90, 91, 92, 93, 94, 95, 96, 97, 98, 99, 100, 101, 102, 103, 104, 105, 106, 107, 108, 109, 110, 111, 112, 113, 114, 115, 116, 117, 118, 119, 120, 121, 122, 123, 124, 125, 126, 127, 128, 129, 130, 131, 132, 133, 134, 135, 136, 137, 138, 139, 140, 141, 142, 143, 144, 145, 146, 147, 148, 149, 150, 151, 152, 153, 154, 155, 156, 157, 158, 159, 160, 161, 162, 163, 164, 165, 166, 167, 168, 169, 170, 171, 172, 173, 174, 175, 176, 177, 178, 179, 180, 181, 182, 183, 184, 185, 186, 187, 188, 189, 190, 191, 192, 193, 194, 195, 196, 197, 198, 199, 200, 201, 202, 203, 204, 205, 206, 207, 208, 209, 210, 211, 212, 213, 214, 215, 216, 217, 218, 219, 220, 221, 222, 223, 224, 225, 226, 227, 228, 229, 230, 231, 232, 233, 234, 235, 236, 237, 238, 239, 240, 241, 242, 243, 244, 245, 246, 247, 248, 249, 250, 251, 252, 253, 254, 255, 256, 257, 258, 259, 260, 261, 262, 263, 264, 265, 266, 267, 268, 269, 270, 271, 272, 273, 274, 275, 276, 277, 278, 279, 280, 281, 282, 283, 284, 285, 286, 287, 288, 289, 290, 291, 292, 293, 294, 295, 296, 297, 298, 299, 300, 301, 302, 303, 304, 305, 306, 307, 308, 309, 310, 311, 312, 313, 314, 315, 316, 317, 318, 319, 320, 321, 322, 323, 324, 325, 326, 327, 328, 329, 330, 331, 332, 333, 334, 335, 336, 337, 338, 339, 340, 341, 342, 343, 344, 345, 346, 347, 348, 349, 350, 351, 352, 353, 354, 355, 356, 357, 358, 359, 360, 361, 362, 363, 364, 365, 366, 367, 368, 369, 370, 371, 372, 373, 374, 375, 376, 377, 378, 379, 380, 381, 382, 383, 384, 385, 386, 387, 388, 389, 390, 391, 392, 393, 394, 395, 396, 397, 398, 399, 400, 401, 402, 403, 404, 405, 406, 407, 408, 409, 410, 411, 412, 413, 414, 415, 416, 417, 418, 419, 420, 421, 422, 423, 424, 425, 426, 427, 428, 429, 430, 431, 432, 433, 434, 435, 436, 437, 438, 439, 440, 441, 442, 443, 444, 445, 446, 447, 448, 449, 450, 451, 452, 453, 454, 455, 456, 457, 458, 459, 460, 461, 462, 463, 464, 465, 466, 467, 468, 469, 470, 471, 472, 473, 474, 475, 476, 477, 478, 479, 480, 481, 482, 483, 484, 485, 486, 487, 488, 489, 490, 491, 492, 493, 494, 495, 496, 497, 498, 499, 500, 501, 502, 503, 504, 505, 506, 507, 508, 509, 510, 511, 512, 513, 514, 515, 516, 517, 518, 519, 520, 521, 522, 523, 524, 525, 526, 527, 528, 529, 530, 531, 532, 533, 534, 535, 536, 537, 538, 539, 540, 541, 542, 543, 544, 545, 546, 547, 548, 549, 550, 551, 552, 553, 554, 555, 556, 557, 558, 559, 560, 561, 562, 563, 564, 565, 566, 567, 568, 569, 570, 571, 572, 573, 574, 575, 576, 577, 578, 579, 580, 581, 582, 583, 584, 585, 586, 587, 588, 589, 590, 591, 592, 593, 594, 595, 596, 597, 598, 599, 600, 601, 602, 603, 604, 605, 606, 607, 608, 609, 610, 611, 612, 613, 614, 615, 616, 617, 618, 619, 620, 621, 622, 623, 624, 625, 626, 627, 628, 629, 630, 631, 632, 633, 634, 635, 636, 637, 638, 639, 640, 641, 642, 643, 644, 645, 646, 647, 648, 649, 650, 651, 652, 653, 654, 655, 656, 657, 658, 659, 660, 661, 662, 663, 664, 665, 666, 667, 668, 669, 670, 671, 672, 673, 674, 675, 676, 677, 678, 679, 680, 681, 682, 683, 684, 685, 686, 687, 688, 689, 690, 691, 692, 693, 694, 695, 696, 697, 698, 699, 700, 701, 702, 703, 704, 705, 706, 707, 708, 709, 710, 711, 712, 713, 714, 715, 716, 717, 718, 719, 720, 721, 722, 723, 724, 725, 726, 727, 728, 729, 730, 731, 732, 733, 734, 735, 736, 737, 738, 739, 740, 741, 742, 743, 744, 745, 746, 747, 748, 749, 750, 751, 752, 753, 754, 755, 756, 757, 758, 759, 760, 761, 762, 763, 764, 765, 766, 767, 768, 769, 770, 771, 772, 773, 774, 775, 776, 777, 778, 779, 780, 781, 782, 783, 784, 785, 786, 787, 788, 789, 790, 791, 792, 793, 794, 795, 796, 797, 798, 799, 800, 801, 802, 803, 804, 805, 806, 807, 808, 809, 810, 811, 812, 813, 814, 815, 816, 817, 818, 819, 820, 821, 822, 823, 824, 825, 826, 827, 828, 829, 830, 831, 832, 833, 834, 835, 836, 837, 838, 839, 840, 841, 842, 843, 844, 845, 846, 847, 848, 849, 850, 851, 852, 853, 854, 855, 856, 857, 858, 859, 860, 861, 862, 863, 864, 865, 866, 867, 868, 869, 870, 871, 872, 873, 874, 875, 876, 877, 878, 879, 880, 881, 882, 883, 884, 885, 886, 887, 888, 889, 890, 891, 892, 893, 894, 895, 896, 897, 898, 899, 900, 901, 902, 903, 904, 905, 906, 907, 908, 909, 910, 911, 912, 913, 914, 915, 916, 917, 918, 919, 920, 921, 922, 923, 924, 925, 926, 927, 928, 929, 930, 931, 932, 933, 934, 935, 936, 937, 938, 939, 940, 941, 942, 943, 944, 945, 946, 947, 948, 949, 950, 951, 952, 953, 954, 955, 956, 957, 958, 959, 960, 961, 962, 963, 964, 965, 966, 967, 968, 969, 970, 971, 972, 973, 974, 975, 976, 977, 978, 979, 980, 981, 982, 983, 984, 985, 986, 987, 988, 989, 990, 991, 992, 993, 994, 995, 996, 997, 998, 999, 1000, 1001, 1002, 1003, 1004, 1005, 1006, 1007, 1008, 1009, 1010, 1011, 1012, 1013, 1014, 1015, 1016, 1017, 1018, 1019, 1020, 1021, 1022, 1023, 1024, 1025, 1026, 1027, 1028, 1029, 1030, 1031, 1032, 1033, 1034, 1035, 1036, 1037, 1038, 1039, 1040, 1041, 1042, 1043, 1044, 1045, 1046, 1047, 1048, 1049, 1050, 1051, 1052, 1053, 1054, 1055, 1056, 1057, 1058, 1059, 1060, 1061, 1062, 1063, 1064, 1065, 1066, 1067, 1068, 1069, 1070, 1071, 1072, 1073, 1074, 1075, 1076, 1077, 1078, 1079, 1080, 1081, 1082, 1083, 1084, 1085, 1086, 1087, 1088, 1089, 1090, 1091, 1092, 1093, 1094, 1095, 1096, 1097, 1098, 1099, 1100, 1101, 1102, 1103, 1104, 1105, 1106, 1107, 1108, 1109, 1110, 1111, 1112, 1113, 1114, 1115, 1116, 1117, 1118, 1119, 1120, 1121, 1122, 1123, 1124, 1125, 1126, 1127, 1128, 1129, 1130, 1131, 1132, 1133, 1134, 1135, 1136, 1137, 1138, 1139, 1140, 1141, 1142, 1143, 1144, 1145, 1146, 1147, 1148, 1149, 1150, 1151, 1152, 1153, 1154, 1155, 1156, 1157, 1158, 1159, 1160, 1161, 1162, 1163, 1164, 1165, 1166, 1167, 1168, 1169, 1170, 1171, 1172, 1173, 1174, 1175, 1176, 1177, 1178, 1179, 1180, 1181, 1182, 1183, 1184, 1185, 1186, 1187, 1188, 1189, 1190, 1191, 1192, 1193, 1194, 1195, 1196, 1197, 1198, 1199, 1200, 1201, 1202, 1203, 1204, 1205, 1206, 1207, 1208, 1209, 1210, 1211, 1212, 1213, 1214, 1215, 1216, 1217, 1218, 1219, 1220, 1221, 1222, 1223, 1224, 1225, 1226, 1227, 1228, 1229, 1230, 1231, 1232, 1233, 1234, 1235, 1236, 1237, 1238, 1239, 1240, 1241, 1242, 1243, 1244, 1245, 1246, 1247, 1248, 1249, 1250, 1251, 1252, 1253, 1254, 1255, 1256, 1257, 1258, 1259, 1260, 1261, 1262, 1263, 1264, 1265, 1266, 1267, 1268, 1269, 1270, 1271, 1272, 1273, 1274, 1275, 1276, 1277, 1278, 1279, 1280, 1281, 1282, 1283, 1284, 1285, 1286, 1287, 1288, 1289, 1290, 1291, 1292, 1293, 1294, 1295, 1296, 1297, 1298, 1299, 1300, 1301, 1302, 1303, 1304, 1305, 1306, 1307, 1308, 1309, 1310, 1311, 1312, 1313, 1314, 1315, 1316, 1317, 1318, 1319, 1320, 1321, 1322, 1323, 1324, 1325, 1326, 1327, 1328, 1329, 1330, 1331, 1332, 1333, 1334, 1335, 1336, 1337, 1338, 1339, 1340, 1341, 1342, 1343, 1344, 1345, 1346, 1347, 1348, 1349, 1350, 1351, 1352, 1353, 1354, 1355, 1356, 1357, 1358, 1359, 1360, 1361, 1362, 1363, 1364, 1365, 1366, 1367, 1368, 1369, 1370, 1371, 1372, 1373, 1374, 1375, 1376, 1377, 1378, 1379, 1380, 1381, 1382, 1383, 1384, 1385, 1386, 1387, 1388, 1389, 1390, 1391, 1392, 1393, 1394, 1395, 1396, 1397, 1398, 1399, 1400, 1401, 1402, 1403, 1404, 1405, 1406, 1407, 1408, 1409, 1410, 1411, 1412, 1413, 1414, 1415, 1416, 1417, 1418, 1419, 1420, 1421, 1422, 1423, 1424, 1425, 1426, 1427, 1428, 1429, 1430, 1431, 1432, 1433, 1434, 1435, 1436, 1437, 1438, 1439, 1440, 1441, 1442, 1443, 1444, 1445, 1446, 1447, 1448, 1449, 1450, 1451, 1452, 1453, 1454, 1455, 1456, 1457, 1458, 1459, 1460, 1461, 1462, 1463, 1464, 1465, 1466, 1467, 1468, 1469, 1470, 1471, 1472, 1473, 1474, 1475, 1476, 1477, 1478, 1479, 1480, 1481, 1482, 1483, 1484, 1485, 1486, 1487, 1488, 1489, 1490, 1491, 1492, 1493, 1494, 1495, 1496, 1497, 1498, 1499, 1500, 1501, 1502, 1503, 1504, 1505, 1506, 1507, 1508, 1509, 1510, 1511, 1512, 1513, 1514, 1515, 1516, 1517, 1518, 1519, 1520, 1521, 1522, 1523, 1524, 1525, 1526, 1527, 1528, 1529, 1530, 1531, 1532, 1533, 1534, 1535, 1536, 1537, 1538, 1539, 1540, 1541, 1542, 1543, 1544, 1545, 1546, 1547, 1548, 1549, 1550, 1551, 1552, 1553, 1554, 1555, 1556, 1557, 1558, 1559, 1560, 1561, 1562, 1563, 1564, 1565, 1566, 1567, 1568, 1569, 1570, 1571, 1572, 1573, 1574, 1575, 1576, 1577, 1578, 1579, 1580, 1581, 1582, 1583, 1584, 1585, 1586, 1587, 1588, 1589, 1590, 1591, 1592, 1593, 1594, 1595, 1596, 1597, 1598, 1599, 1600, 1601, 1602, 1603, 1604, 1605, 1606, 1607, 1608, 1609, 1610, 1611, 1612, 1613, 1614, 1615, 1616, 1617, 1618, 1619, 1620, 1621, 1622, 1623, 1624, 1625, 1626, 1627, 1628, 1629, 1630, 1631, 1632, 1633, 1634, 1635, 1636, 1637, 1638, 1639, 1640, 1641, 1642, 1643, 1644, 1645, 1646, 1647, 1648, 1649, 1650, 1651, 1652, 1653, 1654, 1655, 1656, 1657, 1658, 1659, 1660, 1661, 1662, 1663, 1664, 1665, 1666, 1667, 1668, 1669, 1670, 1671, 1672, 1673, 1674, 1675, 1676, 1677, 1678, 1679, 1680, 1681, 1682, 1683, 1684, 1685, 1686, 1687, 1688, 1689, 1690, 1691, 1692, 1693, 1694, 1695, 1696, 1697, 1698, 1699, 1700, 1701, 1702, 1703, 1704, 1705, 1706, 1707, 1708, 1709, 1710, 1711, 1712, 1713, 1714, 1715, 1716, 1717, 1718, 1719, 1720, 1721, 1722, 1723, 1724, 1725, 1726, 1727, 1728, 1729, 1730, 1731, 1732, 1733, 1734, 1735, 1736, 1737, 1738, 1739, 1740, 1741, 1742, 1743, 1744, 1745, 1746, 1747, 1748, 1749, 1750, 1751, 1752, 1753, 1754, 1755, 1756, 1757, 1758, 1759, 1760, 1761, 1762, 1763, 1764, 1765, 1766, 1767, 1768, 1769, 1770, 1771, 1772, 1773, 1774, 1775, 1776, 1777, 1778, 1779, 1780, 1781, 1782, 1783, 1784, 1785, 1786, 1787, 1788, 1789, 1790, 1791, 1792, 1793, 1794, 1795, 1796, 1797, 1798, 1799, 1800, 1801, 1802, 1803, 1804, 1805, 1806, 1807, 1808, 1809, 1810, 1811, 1812, 1813, 1814, 1815, 1816, 1817, 1818, 1819, 1820, 1821, 1822, 1823, 1824, 1825, 1826, 1827, 1828, 1829, 1830, 1831, 1832, 1833, 1834, 1835, 1836, 1837, 1838, 1839, 1840, 1841, 1842, 1843, 1844, 1845, 1846, 1847, 1848, 1849, 1850, 1851, 1852, 1853, 1854, 1855, 1856, 1857, 1858, 1859, 1860, 1861, 1862, 1863, 1864, 1865, 1866, 1867, 1868, 1869, 1870, 1871, 1872, 1873, 1874, 1875, 1876, 1877, 1878, 1879, 1880, 1881, 1882, 1883, 1884, 1885, 1886, 1887, 1888, 1889, 1890, 1891, 1892, 1893, 1894, 1895, 1896, 1897, 1898, 1899, 1900, 1901, 1902, 1903, 1904, 1905, 1906, 1907, 1908, 1909, 1910, 1911, 1912, 1913, 1914, 1915, 1916, 1917, 1918, 1919, 1920, 1921, 1922, 1923, 1924, 1925, 1926, 1927, 1928, 1929, 1930, 1931, 1932, 1933, 1934, 1935, 1936, 1937, 1938, 1939, 1940, 1941, 1942, 1943, 1944, 1945, 1946, 1947, 1948, 1949, 1950, 1951, 1952, 1953, 1954, 1955, 1956, 1957, 1958, 1959, 1960, 1961, 1962, 1963, 1964, 1965, 1966, 1967, 1968, 1969, 1970, 1971, 1972, 1973, 1974, 1975, 1976, 1977, 1978, 1979, 1980, 1981, 1982, 1983, 1984, 1985, 1986, 1987, 1988, 1989, 1990, 1991, 1992, 1993, 1994, 1995, 1996, 1997, 1998, 1999, 2000, 2001, 2002, 2003, 2004, 2005, 2006, 2007, 2008, 2009, 2010, 2011, 2012, 2013, 2014, 2015, 2016, 2017, 2018, 2019, 2020, 2021, 2022, 2023, 2024, 2025, 2026, 2027, 2028, 2029, 2030, 2031, 2032, 2033, 2034, 2035, 2036, 2037, 2038, 2039, 2040, 2041, 2042, 2043, 2044, 2045, 2046, 2047, 2048, 2049, 2050, 2051, 2052, 2053, 2054, 2055, 2056, 2057, 2058, 2059, 2060, 2061, 2062, 2063, 2064, 2065, 2066, 2067, 2068, 2069, 2070, 2071, 2072, 2073, 2074, 2075, 2076, 2077, 2078, 2079, 2080, 2081, 2082, 2083, 2084, 2085, 2086, 2087, 2088, 2089, 2090, 2091, 2092, 2093, 2094, 2095, 2096, 2097, 2098, 2099, 2100, 2101, 2102, 2103, 2104, 2105, 2106, 2107, 2108, 2109, 2110, 2111, 2112, 2113, 2114, 2115, 2116, 2117, 2118, 2119, 2120, 2121, 2122, 2123, 2124, 2125, 2126, 2127, 2128, 2129, 2130, 2131, 2132, 2133, 2134, 2135, 2136, 2137, 2138, 2139, 2140, 2141, 2142, 2143, 2144, 2145, 2146, 2147, 2148, 2149, 2150, 2151, 2152, 2153, 2154, 2155, 2156, 2157, 2158, 2159, 2160, 2161, 2162, 2163, 21
```

A Case of Higher-Order Control-Flow Analysis

- [Recipe step 2] During the analysis, use LLMs as the decision maker.

Your output should be only a JSON object with the following schema:

```
{
  "reason": string
  "query-type": "BAddr" or "KAddr" or "Tick",
  "variable": String,           // for BAddr only
  "time": Bool,                // for both BAddr and KAddr
  "source-expression": Bool,   // for KAddr only
  "target-expression": Bool,   // for KAddr only
  "target-environment": Bool,   // for KAddr only
  "target-binding-store": Bool, // for KAddr only
  "k": Int represented as String, // for Tick only
}
```

Describe output JSON format for each query.

```
NOT (empty) string = ""
// This is an expert in static analysis and abstract interpretation, specifically in the "abstracting abstract machine" approach.
// Your task is to analyze the program of a multi-addr abstract interpreter that transforms an abstract machine state into possible concrete states.
// It will provide you the data structures used in the abstract interpretation task.
// You give your the current analysis state, and you to analyze it, and you are to decide the content-semanticity or
// return an abstract address used in the allocation.
0
// The "State" field is an abstract state, which can be one of the following:
// enum State {
//   0: None
//   1: BAddr
//   2: KAddr
//   3: Tick
// }
// The "BAddr" field is the current program address being evaluated, which is defined by the following algebraic data type:
// enum BAddr {
//   0: None
//   1: BAddr
//   2: BAddr
//   3: BAddr
//   4: BAddr
//   5: BAddr
//   6: BAddr
//   7: BAddr
//   8: BAddr
//   9: BAddr
//   10: BAddr
//   11: BAddr
//   12: BAddr
//   13: BAddr
//   14: BAddr
//   15: BAddr
//   16: BAddr
//   17: BAddr
//   18: BAddr
//   19: BAddr
//   20: BAddr
//   21: BAddr
//   22: BAddr
//   23: BAddr
//   24: BAddr
//   25: BAddr
//   26: BAddr
//   27: BAddr
//   28: BAddr
//   29: BAddr
//   30: BAddr
//   31: BAddr
//   32: BAddr
//   33: BAddr
//   34: BAddr
//   35: BAddr
//   36: BAddr
//   37: BAddr
//   38: BAddr
//   39: BAddr
//   40: BAddr
//   41: BAddr
//   42: BAddr
//   43: BAddr
//   44: BAddr
//   45: BAddr
//   46: BAddr
//   47: BAddr
//   48: BAddr
//   49: BAddr
//   50: BAddr
//   51: BAddr
//   52: BAddr
//   53: BAddr
//   54: BAddr
//   55: BAddr
//   56: BAddr
//   57: BAddr
//   58: BAddr
//   59: BAddr
//   60: BAddr
//   61: BAddr
//   62: BAddr
//   63: BAddr
//   64: BAddr
//   65: BAddr
//   66: BAddr
//   67: BAddr
//   68: BAddr
//   69: BAddr
//   70: BAddr
//   71: BAddr
//   72: BAddr
//   73: BAddr
//   74: BAddr
//   75: BAddr
//   76: BAddr
//   77: BAddr
//   78: BAddr
//   79: BAddr
//   80: BAddr
//   81: BAddr
//   82: BAddr
//   83: BAddr
//   84: BAddr
//   85: BAddr
//   86: BAddr
//   87: BAddr
//   88: BAddr
//   89: BAddr
//   90: BAddr
//   91: BAddr
//   92: BAddr
//   93: BAddr
//   94: BAddr
//   95: BAddr
//   96: BAddr
//   97: BAddr
//   98: BAddr
//   99: BAddr
// }
// The "KAddr" field is the current continuation address, which is defined by the following algebraic data type:
// enum KAddr {
//   0: None
//   1: KAddr
//   2: KAddr
//   3: KAddr
//   4: KAddr
//   5: KAddr
//   6: KAddr
//   7: KAddr
//   8: KAddr
//   9: KAddr
//   10: KAddr
//   11: KAddr
//   12: KAddr
//   13: KAddr
//   14: KAddr
//   15: KAddr
//   16: KAddr
//   17: KAddr
//   18: KAddr
//   19: KAddr
//   20: KAddr
//   21: KAddr
//   22: KAddr
//   23: KAddr
//   24: KAddr
//   25: KAddr
//   26: KAddr
//   27: KAddr
//   28: KAddr
//   29: KAddr
//   30: KAddr
//   31: KAddr
//   32: KAddr
//   33: KAddr
//   34: KAddr
//   35: KAddr
//   36: KAddr
//   37: KAddr
//   38: KAddr
//   39: KAddr
//   40: KAddr
//   41: KAddr
//   42: KAddr
//   43: KAddr
//   44: KAddr
//   45: KAddr
//   46: KAddr
//   47: KAddr
//   48: KAddr
//   49: KAddr
//   50: KAddr
//   51: KAddr
//   52: KAddr
//   53: KAddr
//   54: KAddr
//   55: KAddr
//   56: KAddr
//   57: KAddr
//   58: KAddr
//   59: KAddr
//   60: KAddr
//   61: KAddr
//   62: KAddr
//   63: KAddr
//   64: KAddr
//   65: KAddr
//   66: KAddr
//   67: KAddr
//   68: KAddr
//   69: KAddr
//   70: KAddr
//   71: KAddr
//   72: KAddr
//   73: KAddr
//   74: KAddr
//   75: KAddr
//   76: KAddr
//   77: KAddr
//   78: KAddr
//   79: KAddr
//   80: KAddr
//   81: KAddr
//   82: KAddr
//   83: KAddr
//   84: KAddr
//   85: KAddr
//   86: KAddr
//   87: KAddr
//   88: KAddr
//   89: KAddr
//   90: KAddr
//   91: KAddr
//   92: KAddr
//   93: KAddr
//   94: KAddr
//   95: KAddr
//   96: KAddr
//   97: KAddr
//   98: KAddr
//   99: KAddr
// }
// The "Tick" field is a list of program call sites expression that represent the history of
// (call) leading to the current state (current state).
// enum Tick {
//   0: None
//   1: Tick
//   2: Tick
//   3: Tick
//   4: Tick
//   5: Tick
//   6: Tick
//   7: Tick
//   8: Tick
//   9: Tick
//   10: Tick
//   11: Tick
//   12: Tick
//   13: Tick
//   14: Tick
//   15: Tick
//   16: Tick
//   17: Tick
//   18: Tick
//   19: Tick
//   20: Tick
//   21: Tick
//   22: Tick
//   23: Tick
//   24: Tick
//   25: Tick
//   26: Tick
//   27: Tick
//   28: Tick
//   29: Tick
//   30: Tick
//   31: Tick
//   32: Tick
//   33: Tick
//   34: Tick
//   35: Tick
//   36: Tick
//   37: Tick
//   38: Tick
//   39: Tick
//   40: Tick
//   41: Tick
//   42: Tick
//   43: Tick
//   44: Tick
//   45: Tick
//   46: Tick
//   47: Tick
//   48: Tick
//   49: Tick
//   50: Tick
//   51: Tick
//   52: Tick
//   53: Tick
//   54: Tick
//   55: Tick
//   56: Tick
//   57: Tick
//   58: Tick
//   59: Tick
//   60: Tick
//   61: Tick
//   62: Tick
//   63: Tick
//   64: Tick
//   65: Tick
//   66: Tick
//   67: Tick
//   68: Tick
//   69: Tick
//   70: Tick
//   71: Tick
//   72: Tick
//   73: Tick
//   74: Tick
//   75: Tick
//   76: Tick
//   77: Tick
//   78: Tick
//   79: Tick
//   80: Tick
//   81: Tick
//   82: Tick
//   83: Tick
//   84: Tick
//   85: Tick
//   86: Tick
//   87: Tick
//   88: Tick
//   89: Tick
//   90: Tick
//   91: Tick
//   92: Tick
//   93: Tick
//   94: Tick
//   95: Tick
//   96: Tick
//   97: Tick
//   98: Tick
//   99: Tick
// }
// Your input is a JSON object with the following schema:
// {
//   "reason": string,
//   "query-type": "BAddr" or "KAddr" or "Tick",
//   "variable": String,
//   "time": Bool,
//   "source-expression": Bool,
//   "target-expression": Bool,
//   "target-environment": Bool,
//   "target-binding-store": Bool,
//   "k": Int represented as String,
// }
// You should analyze the current "State" and return an abstract address for better analysis (BAddr).
// You should look at the address of the current binding store and continuation store.
// You should look at the binding address (BAddr) and return the analysis of the address.
// Your output should be only a JSON object with the following schema:
// {
//   "reason": string,
//   "query-type": "BAddr" or "KAddr" or "Tick",
//   "variable": String,
//   "time": Bool,
//   "source-expression": Bool,
//   "target-expression": Bool,
//   "target-environment": Bool,
//   "target-binding-store": Bool,
//   "k": Int represented as String,
// }
// The reason you give should be specific to the current state and should not be generic.
// Field "query-type" should be either "BAddr", "KAddr", or "Tick" according to the "query-type" in the input query.
// This may not be true if you think it is not needed for the analysis.
// Field "variable" should be the same variable name as in the input query for binding address.
// Only if "query-type" is "BAddr":
// Field "time" should be true if the binding address should be instrumented with the time (tick) history.
// This may not be true if you think it is not needed for the analysis.
// Only if "query-type" is "KAddr":
// Field "source-expression" should be true if the continuation address should be instrumented with the source expression, otherwise false.
// Field "target-expression" should be true if the continuation address should be instrumented with the target expression, otherwise false.
// Field "target-environment" should be true if the continuation address should be instrumented with the target environment, otherwise false.
// Field "target-binding-store" should be true if the continuation address should be instrumented with the target binding store, otherwise false.
// Only if "query-type" is "Tick":
// Field "k" should be the number represented as a string of call-site-continuity levels to use in the analysis.
// (The returned value "k" is the analysis will be in call-site-continuity).
// (It is to use the current call expression in progress in the history, and then the take first a call site of the whole history).
// (Typically, field "k" is a small number, like 0 or 1, if you think k = 2 is needed, please justify in the "reason" field.)
// --analysis--
```

- **[Recipe step 2]** During the analysis, use LLMs as the decision maker.

[illegible][illegible]

A Case of Higher-Order Control-Flow Analysis

- A prototype “LLMAAM” implemented in Scala
<https://github.com/Kraks/llmaam>
- Preliminary evaluation of micro-benchmarks with GPT-4o-mini

	idid		kcfa2		mj09		loop2	
	#N	#E	#N	#E	#N	#E	#N	#E
0CFA+P4F	42	41	307	306	303	302	110	108
0CFA+AAC	33	32	203	202	179	175	110	108
0CFA+SRC	42	41	246	245	647	644	149	146
1CFA+P4F	33	32	TO	TO	1437	1407	108	106
1CFA+AAC	33	32	336	324	331	323	108	106
1CFA+SRC	42	41	12189	12117	6849	6834	108	106
llmaam	33	32	133	131	199	202	108	106
	33	32	153	150	268	259	108	106
	33	32	272	265	281	273	108	106

#edges and #states as approximation of precision – smaller is better

Use LLMs to “Analyze” the Analysis!

Recipe for sound, precise, tunable static analysis driven by LLMs:

- Identify precision-critical decisions in static analysis
- During the analysis, use LLMs as meta-analysis and the decision maker

Future opportunities – Analyses:

- Control-flow sensitivity
- Numeric analysis
- Termination analysis
- ...

Future opportunities – Optimizations:

- Prompt optimizations
- Fine-tuning
- Reinforcement learning
- ...