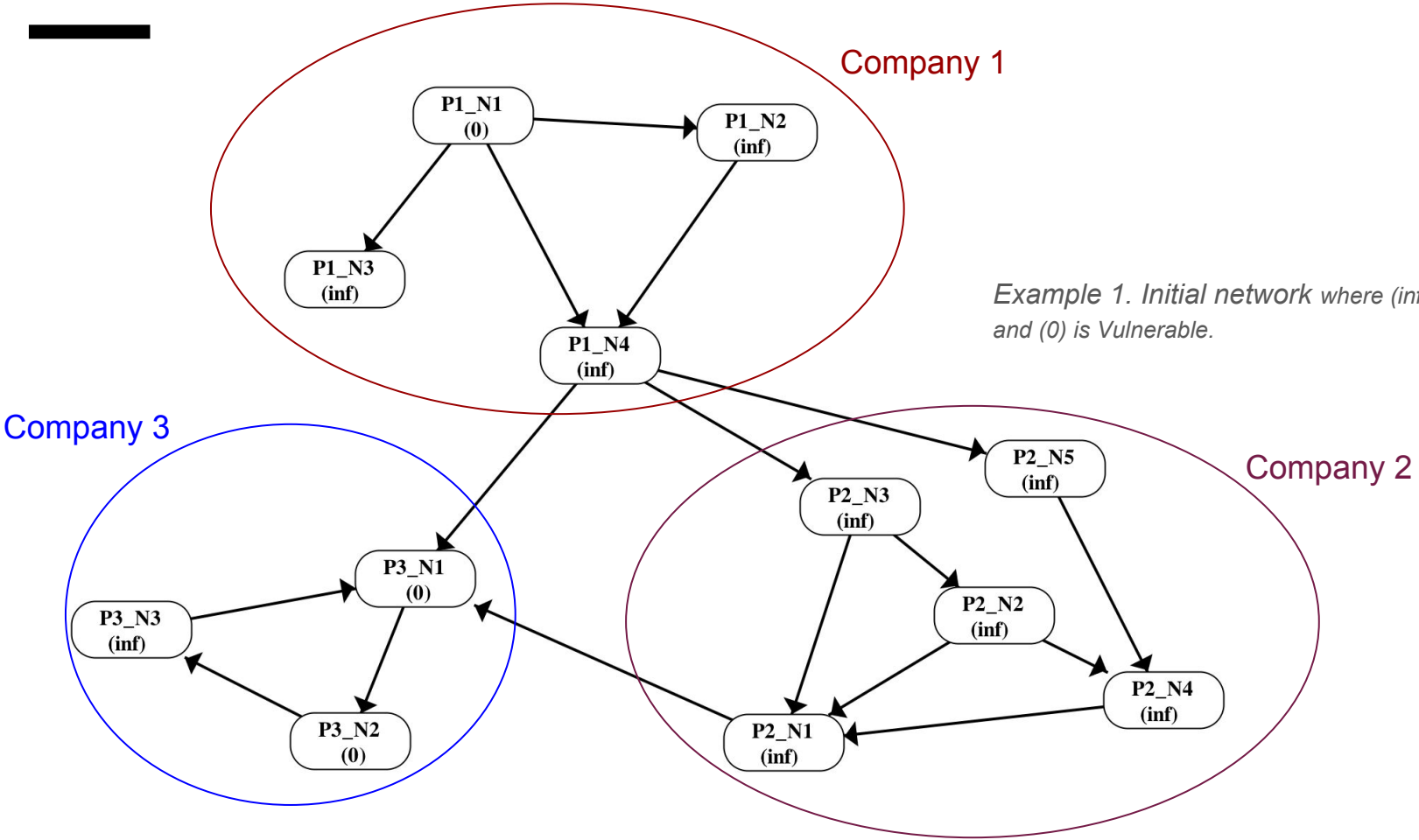
Network Distance

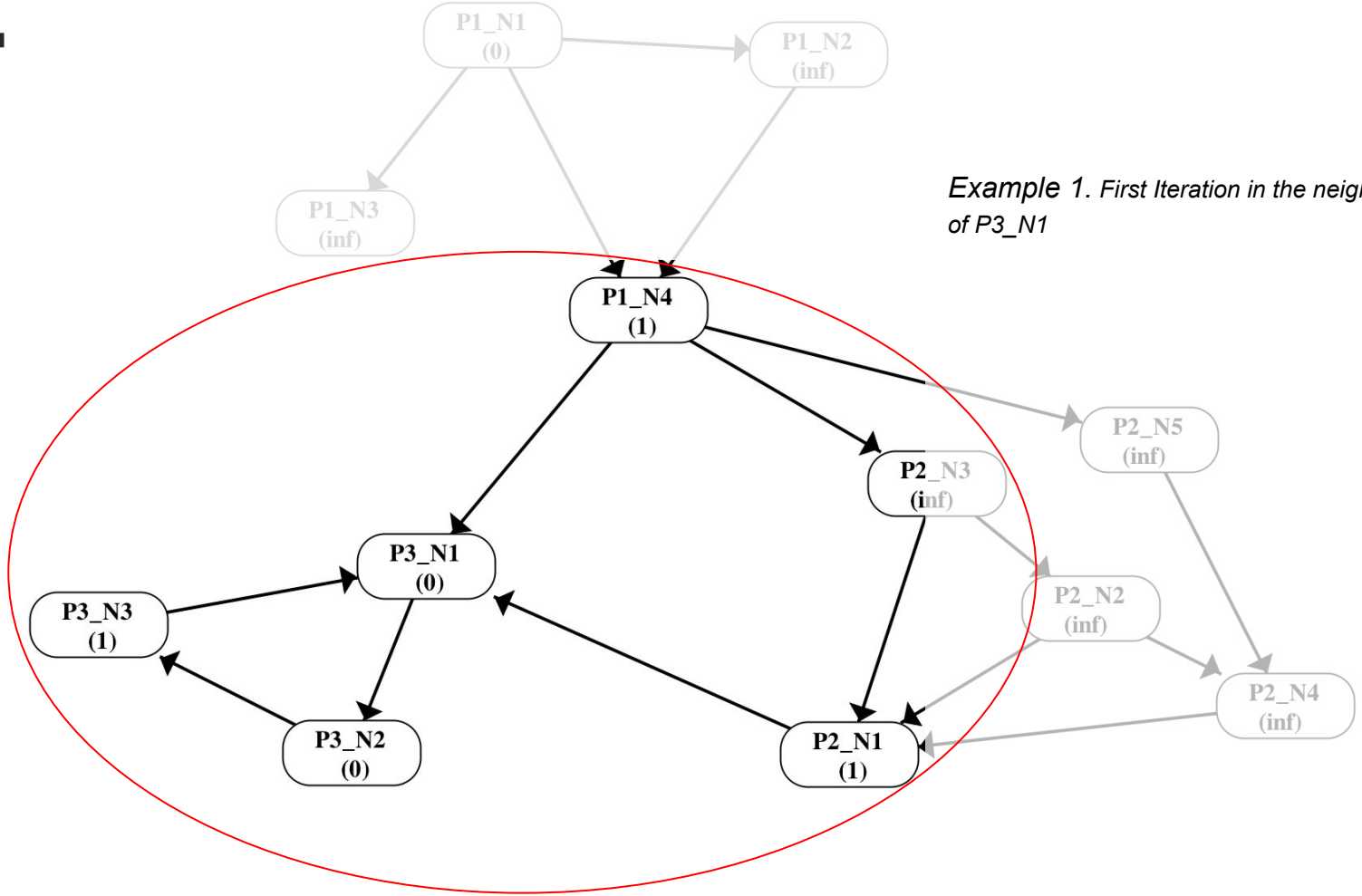
And Other iterative problems in MPC

Network Distance

- Large network with nodes belonging to different companies.
- Two kinds of nodes: Vulnerable, and Safe.
- Compute minimum distance to a vulnerable node for all nodes.
- Vulnerable nodes have distance 0
- Safe nodes have distance infinity.
- Computation consists of alternating rounds of addition and min.



Example 1. Initial network where (inf) is safe and (0) is Vulnerable.



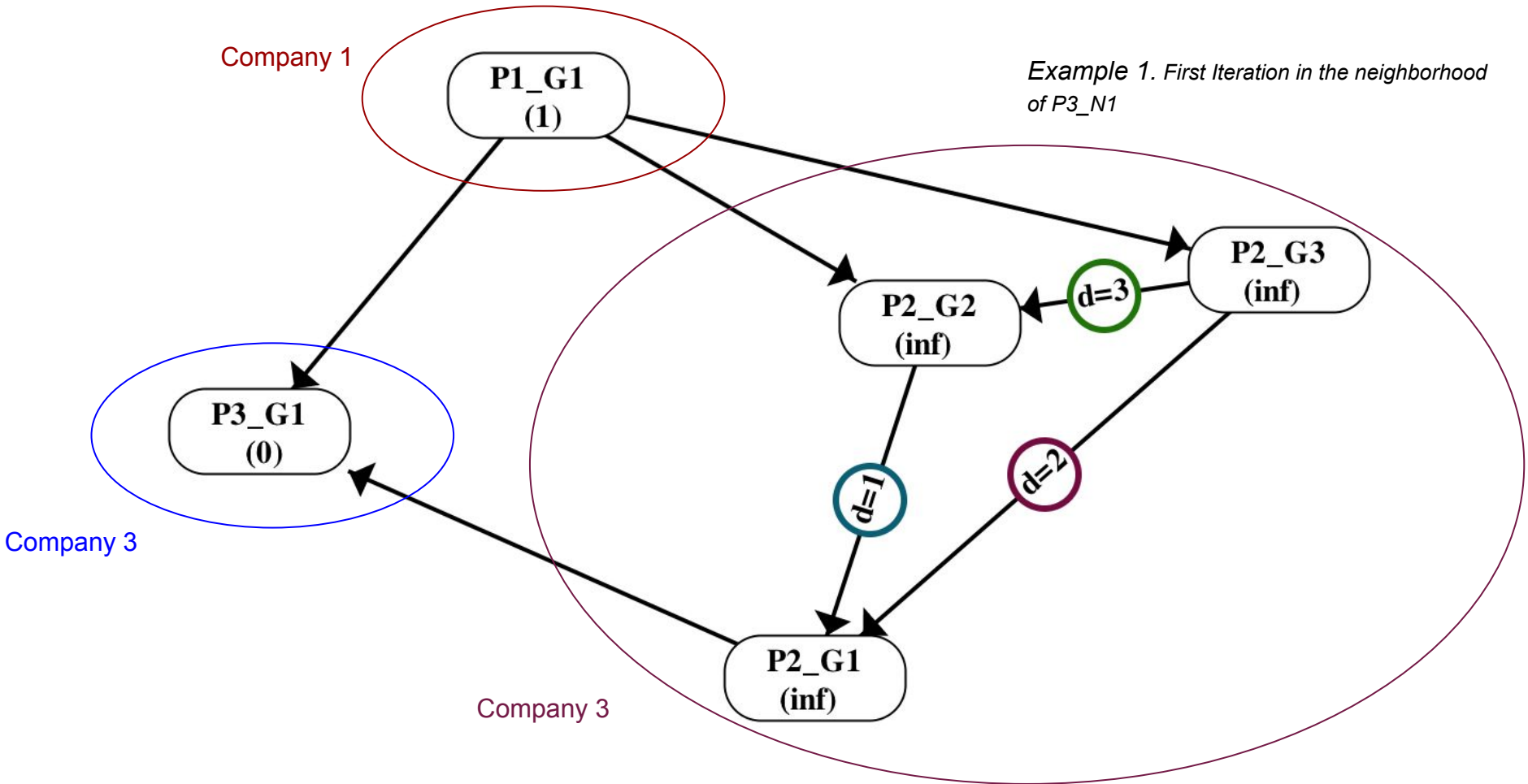
Example 1. First Iteration in the neighborhood of P3_N1

Security

- Passive Security (for now).
- Hide internal topology of a company’s network.
- Final distance is known but not any intermediate values.
- Final distance is known but not the causing vulnerable nodes.
- Gateways graph is public.
- Final output is known to everyone (for now).

Problems

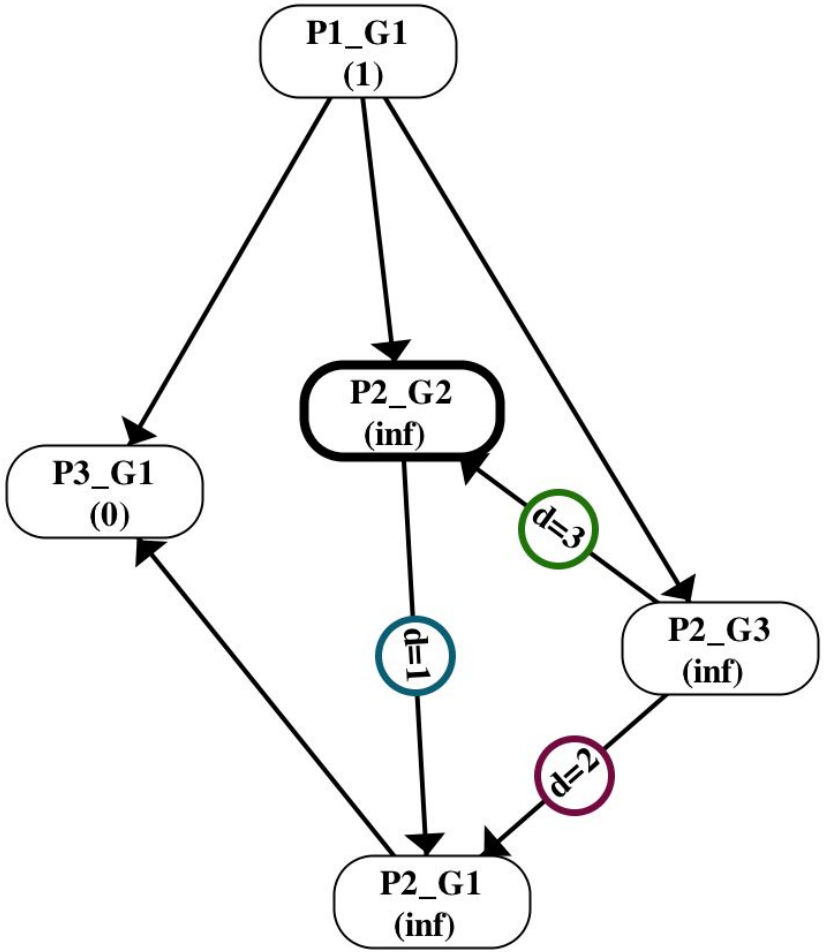
- Graph can be very large
  - Cannot perform MPC on the entire graph.
  - Companies perform direct computation on their private network.
  - MPC computation on the subgraph containing only gateways of companies.
  - Direct computation to propagate gateways results back into private network.



Example 1. First Iteration in the neighborhood of P3_N1

Problems

- Not revealing any intermediate values
  - Can be done with viff, but with huge cost (min is expensive).
  - Run algorithm with “symbolic” inputs.
  - Unwraps iteration/execution yielding an expression.
  - Secret share initial distances, evaluate expression in one shot.
  - Only final answer is revealed.
  - Expression can be optimized. It can be stored and re-used.



$P2\_G2_1$

= **min**($P2\_G2_0$, $P1\_G1_0$+1, $P2\_G3_0$+3, $P2\_G1_0$+1)

= **min**(*inf*, *2*, *inf*, *inf*)

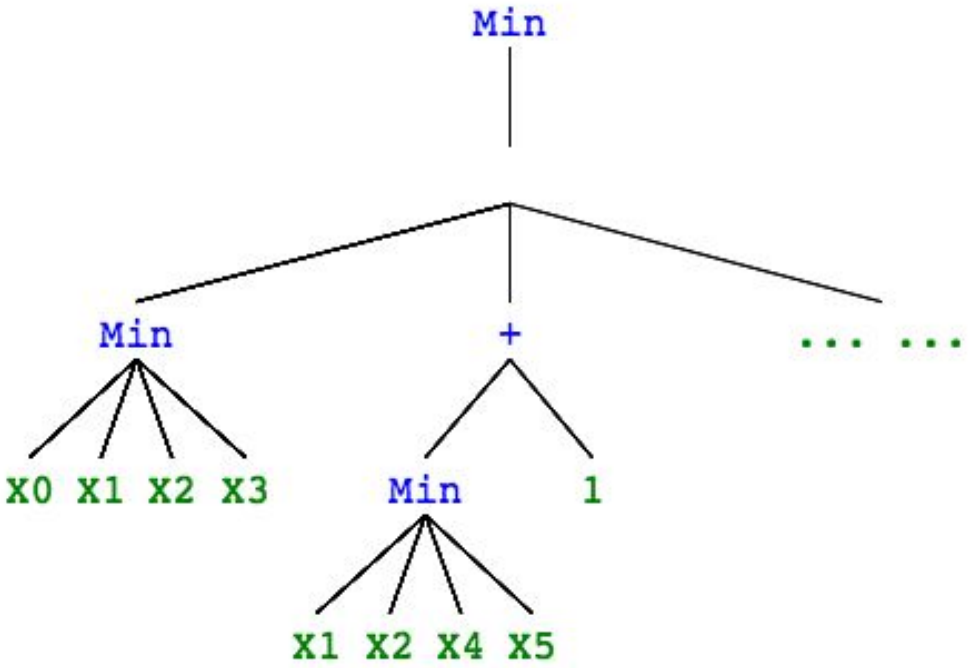
$P2\_G2_2$

= **min**($P2\_G2_1$, $P1\_G1_1$+1, $P2\_G3_1$+3, $P2\_G1_1$+1)

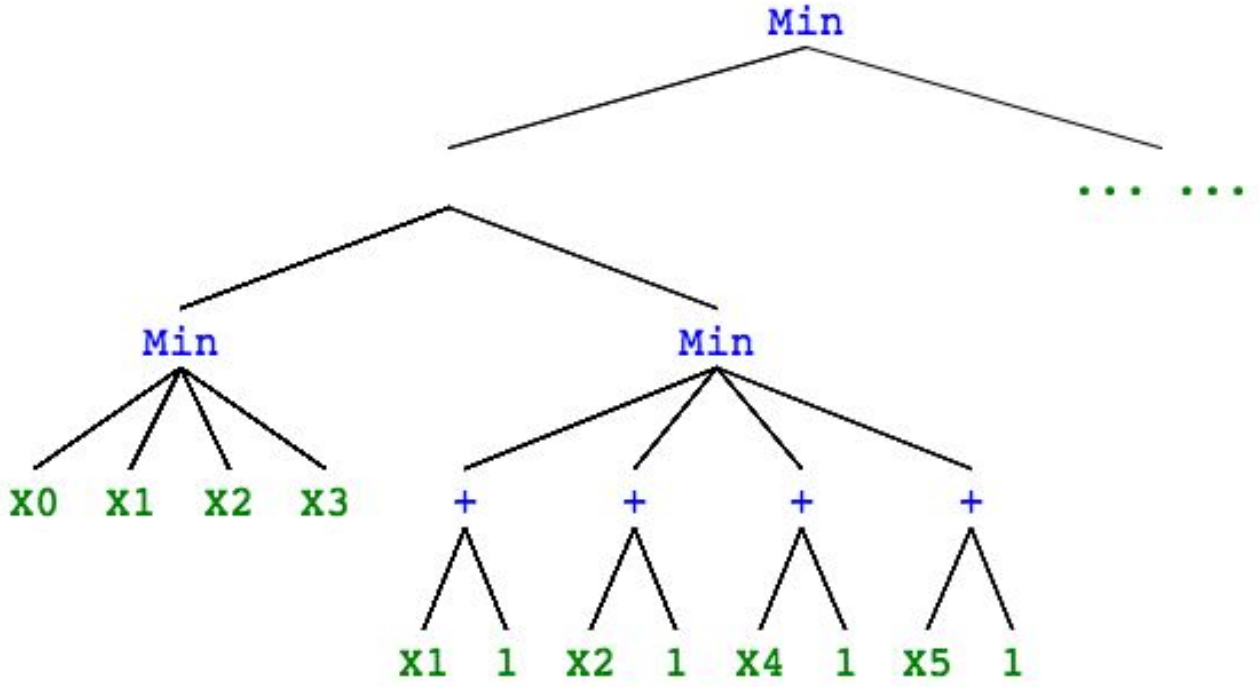
= **min**(**min**(*inf*, *2*, *inf*, *inf*), **min**(*1*, *1*, *inf*, *inf*)+1, …)

Expression Tree

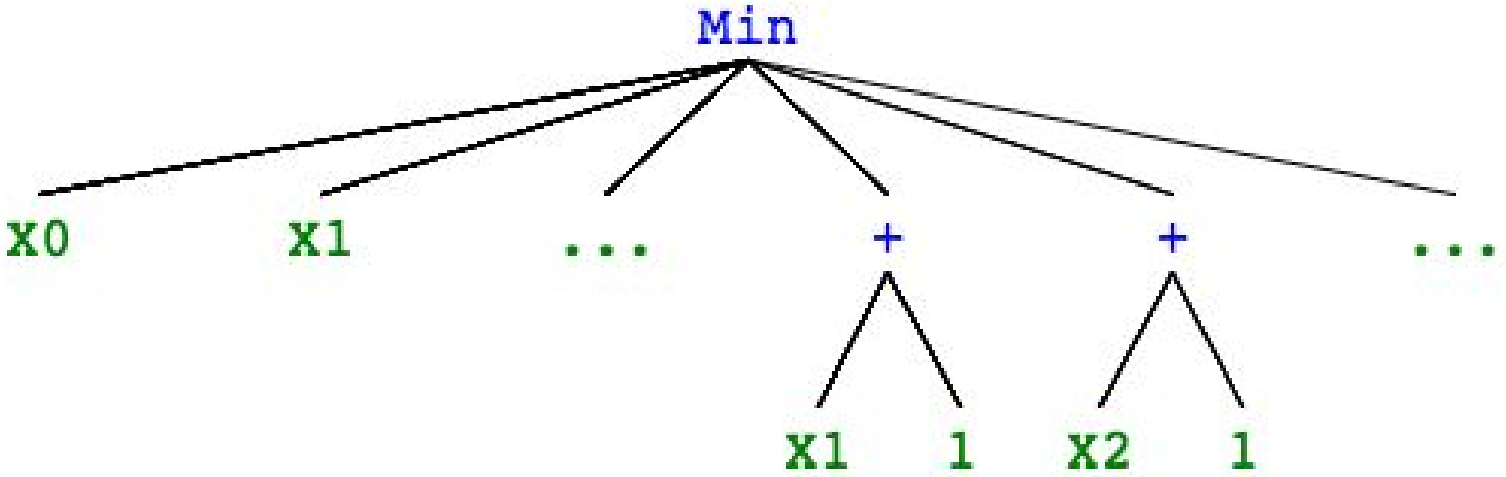
- Expressions as parse trees.
- Levels of + and Min.
- Optimizing the syntax tree is equivalent to simplifying the expression.
- Simplification utilizes distribution properties of + and Min.
- Simplification yields a single Min with compact additions.



Add-Min Reduction



Min-Min Reduction



Library Modules

- Implemented as a Python Library.
- Operator overloading to construct Expressions.
- Important Modules
  - Expression : AddExpression, MinExpression, VarExpression …
  - Simplifiers: BaseSimplifier, Reductions, Optimizing Simplifiers …
  - Evaluators: BaseEvaluator (Direct), ViffEvaluator.
- Customize expression, simplifiers, and evaluators (inheritance).
- Code does not reveal MPC, Expressions, or Twisted.

Remaining Work

- Optimizing Min expression by removing obvious terms.
  - Min(X1 + 1, X1 + 2, ...) => Min(X1 + 1, …)
- Speed-up evaluation by exploiting similar terms (memoization).
- MPC protocols for Min
  - Currently we are using **(x * y/x + y * x/y) / (x/y + y/x)**
  - Slow (due to division), but does not reveal any information.
  - Look for faster division or min protocols with similar guarantees.
- SPDZ Evaluator ?
- Investigate producing different outputs for each company.

Future Extensions

- Provide builtin support for more expressions and simplifications.
- Apply the library to more problems.
- Include conditionals, loops, and other python statements
  - Overload the python parse tree instead of only operators.
  - Generic MPC protocols for these constructs.

Thank You

Questions?