# Static Profiling of Parametric Resource Usage as a Valuable Aid for Hot-spot Detection

P. López-García[1,2] M. Klemen[1] U. Liqat[1] Manuel Hermenegildo[1,3]

[1]IMDEA Software Institute
[2]Spanish Research Council (CSIC)
[3]T. U. of Madrid (UPM)

Melbourne, CICLOPS, August 28, 2017

# Introduction and Motivation

- Resources: non-func. numerical properties about the execution of a program.
  - Examples: resolution steps, execution time, energy consumption, # of calls to a predicate, # of network accesses, # of transactions, ...
- Goal of static analysis:
  estimating the resource usage of the execution of a program without running it with concrete data, as function of input data sizes and possibly other parameters.

  Typical size metrics → actual value of a number, the length of a list, the number of constant and function symbols of a term, etc.
- Significant work done in logic programming.
  - + Allows analysis of other languages via transformation into Horn Clauses.
- Resource analysis is very useful:
  - Automatic program optimization.
  - Verification of resource-related specifications.
  - Detection of performance bugs, help guiding software design, ...
    Example: developing energy-efficient software.

# Inferring Accumulated Cost [TPLP'16, FLOPS'16]

- Helping developers make (resource-related) design decisions:
  - Which parts of the program are the most resource-consuming?
  - Which predicates should be optimized first?
- The standard/classical notion of cost only partially meets these objectives:
  - Predicates w/highest (standard) costs may not need to be optimized first.
  - E.g., perhaps predicates with lower costs but which are called more often.
  - The input sizes to such calls are also relevant.
- Need info resulting from a static profiling of the program to:
  - identify the parts of a program responsible for highest fractions of the cost $\rightarrow$ accumulated cost.
  - I.e., how the total resource usage of the execution of a program is *distributed* over selected parts of it (cost centers $\rightarrow$ predicates).

  Static profiling $\rightarrow$ *static inference of the kinds of information that are usually obtained at run-time by profilers*.

## Main contribution

Novel, general, and flexible framework for setting up cost equations/relations.
$\rightarrow$ can be instantiated for performing a wide range of static resource usage analyses, including both accumulated cost and standard cost.

# Overview of the Classical Cost Analysis

1. Perform all the required supporting analyses (examples):
   - Types (shapes) for inferring size metrics (list-length, term-depth, ...).
   - Mode analysis to determine input/output arguments.
   - Sharing analysis for correctness (conservative: only when there is no sharing among data structures).
   - *Non-failure* (no exceptions) inferred for non-trivial lower bounds.
   - *Determinacy* (mutual exclusion) to obtain tighter bounds.
2. Set up recurrence equations representing the size of each (relevant) output argument as a function of the input data sizes.
   - Size metrics are derived from inferred type (shape) information.
   - Data dependency graphs used to determine *relative* sizes of variable contents.
3. Compute (lower/upper) bounds to the solutions of these recurrence equations to obtain output argument sizes as (closed-form) functions of input sizes.
   - Using internal recurrence solver, or the interfaces with Mathematica, Parma, PUBS, Matlab, etc.

[PLDI'90, SAS'94, PASCO'94]

# Overview of the Classical Cost Analysis

1. Perform all the required supporting analyses (examples):
   - Types (shapes) for inferring size metrics (list-length, term-depth, ...).
   - Mode analysis to determine input/output arguments.
   - Sharing analysis for correctness (conservative: only when there is no sharing among data structures).
   - *Non-failure* (no exceptions) inferred for non-trivial lower bounds.
   - *Determinacy* (mutual exclusion) to obtain tighter bounds.
2. Set up recurrence equations representing the size of each (relevant) output argument as a function of the input data sizes.
   - Size metrics are derived from inferred type (shape) information.
   - Data dependency graphs used to determine *relative* sizes of variable contents.
3. Compute (lower/upper) bounds to the solutions of these recurrence equations to obtain output argument sizes as (closed-form) functions of input sizes.
   - Using internal recurrence solver, or the interfaces with Mathematica, Parma, PUBS, Matlab, etc.

[PLDI'90, SAS'94, PASCO'94]

# Overview of the Classical Cost Analysis

1. Perform all the required supporting analyses (examples):
   - Types (shapes) for inferring size metrics (list-length, term-depth, ...).
   - Mode analysis to determine input/output arguments.
   - Sharing analysis for correctness (conservative: only when there is no sharing among data structures).
   - *Non-failure* (no exceptions) inferred for non-trivial lower bounds.
   - *Determinacy* (mutual exclusion) to obtain tighter bounds.
2. Set up recurrence equations representing the size of each (relevant) output argument as a function of the input data sizes.
   - Size metrics are derived from inferred type (shape) information.
   - Data dependency graphs used to determine *relative* sizes of variable contents.
3. Compute (lower/upper) bounds to the solutions of these recurrence equations to obtain output argument sizes as (closed-form) functions of input sizes.
   - Using internal recurrence solver, or the interfaces with Mathematica, Parma, PUBS, Matlab, etc.

[PLDI'90, SAS'94, PASCO'94]

# Overview of the Classical Cost Analysis

1. Perform all the required supporting analyses (examples):
   - Types (shapes) for inferring size metrics (list-length, term-depth, ...).
   - Mode analysis to determine input/output arguments.
   - Sharing analysis for correctness (conservative: only when there is no sharing among data structures).
   - *Non-failure* (no exceptions) inferred for non-trivial lower bounds.
   - *Determinacy* (mutual exclusion) to obtain tighter bounds.
2. Set up recurrence equations representing the size of each (relevant) output argument as a function of the input data sizes.
   - Size metrics are derived from inferred type (shape) information.
   - Data dependency graphs used to determine *relative* sizes of variable contents.
3. Compute (lower/upper) bounds to the solutions of these recurrence equations to obtain output argument sizes as (closed-form) functions of input sizes.
   - Using internal recurrence solver, or the interfaces with Mathematica, Parma, PUBS, Matlab, etc.
4. E.g.:

```
:- true pred append(A,B,C) : list * list * var
                       => ( size_lb(C, length(A)+length(B)),
                            size_ub(C, length(A)+length(B)) ).
```

[PLDI'90, SAS'94, PASCO'94]

# Overview of the Classical Cost Analysis

1. Perform all the required supporting analyses (examples):
   - Types (shapes) for inferring size metrics (list-length, term-depth, ...).
   - Mode analysis to determine input/output arguments.
   - Sharing analysis for correctness (conservative: only when there is no sharing among data structures).
   - *Non-failure* (no exceptions) inferred for non-trivial lower bounds.
   - *Determinacy* (mutual exclusion) to obtain tighter bounds.
2. Set up recurrence equations representing the size of each (relevant) output argument as a function of the input data sizes.
   - Size metrics are derived from inferred type (shape) information.
   - Data dependency graphs used to determine *relative* sizes of variable contents.
3. Compute (lower/upper) bounds to the solutions of these recurrence equations to obtain output argument sizes as (closed-form) functions of input sizes.
   - Using internal recurrence solver, or the interfaces with Mathematica, Parma, PUBS, Matlab, etc.

5. Use the size information to set up recurrence equations representing the computational cost of each clause and compute bounds to their solutions to obtain **cost functions**.

[PLDI'90, SAS'94, PASCO'94]

# Size Metrics

- Various size metrics can be used to determine the size of an input:
  - the actual value of a number,
  - the length of a list,
  - the number of constant and function symbols in a term.
  - the depth of a term,
  - etc.
- These are automatically inferred based on type (shape) analysis and other information (program control flow and operations).
- The function $size_m(t)$ defines the size of a term $t$ under the metric $m$:
  $size_{length}(\texttt{[4,2,7]}) = 3$
  $size_{length}(\texttt{[]}) = 0$
  $size_{term\_depth}(\texttt{f(a,g(b))}) = 2$
- The function $diff_m(t_1, t_2)$ gives the size difference between two terms $t_1$ and $t_2$ under the metric $m$:
  $diff_{length}([2, 3|\texttt{L}], [4|\texttt{L}]) = 1$
  $diff_{length}(\texttt{L}, [\texttt{H}|\texttt{L}]) = -1$
  $diff_{term\_depth}(\texttt{f}(\texttt{a}, \texttt{g}(\texttt{X})), \texttt{X}) = 2$

# Size Metrics

- Various size metrics can be used to determine the size of an input:
  - the actual value of a number,
  - the length of a list,
  - the number of constant and function symbols in a term.
  - the depth of a term,
  - etc.
- These are automatically inferred based on type (shape) analysis and other information (program control flow and operations).
- The function $size_m(t)$ defines the size of a term $t$ under the metric $m$:
  $size_{length}(\texttt{[4,2,7]}) = 3$
  $size_{length}(\texttt{[]}) = 0$
  $size_{term\_depth}(\texttt{f(a,g(b))}) = 2$
- The function $diff_m(t_1, t_2)$ gives the size difference between two terms $t_1$ and $t_2$ under the metric $m$:
  $diff_{length}([2,3|\mathtt{L}],[4|\mathtt{L}]) = 1$
  $diff_{length}(\mathtt{L},[\mathtt{H}|\mathtt{L}]) = -1$
  $diff_{term\_depth}(\mathtt{f}(\mathtt{a},\mathtt{g}(\mathtt{X})),\mathtt{X}) = 2$

# Size Analysis (size relations): Example

```
:- entry nrev/2 : list(num) * var.

nrev([],[]).
nrev([H|L],R) :- nrev(L,R1), app(R1,[H],R).

app([],L,L).
app([H|L],L1,[H|R]) :- app(L,L1,R).
```

- The automatically inferred size metric is *length* (list length) for all arguments.
- All arguments inferred to be input except last ones (output). No aliasing.
- Let $\langle b.i \rangle$ denote (a bound on) the size of the term(s) appearing in the $i^{th}$ argument position in the head of a clause defining predicate $b$.
- Let $\langle p.j.i \rangle$ denote (a bound on) the size of the term(s) appearing in the $i^{th}$ argument position in the $j^{th}$ body call of a clause defining predicate $b$.

  $\rightarrow$ $p$ denotes the predicate called (for readability).
- Example (2nd clause): $\langle app.1.1 \rangle = length($`L`$)$ and $\langle app.1 \rangle = length($`[H|L]`$)$.
- First, we consider predicate `app(A,B,C)` (third arg is output).
- We want to obtain intra-predicate argument size relations: $Sz_3^{app}(x, y)$ represents the size of the third argument of `app` as a function of its input data sizes ($x = length($`A`$)$ and $y = length($`B`$)$).

# Size Analysis (size relations): Example

```
:- entry nrev/2 : list(num) * var.

nrev([],[]).
nrev([H|L],R) :- nrev(L,R1), app(R1,[H],R).

app([],L,L).
app([H|L],L1,[H|R]) :- app(L,L1,R).
```

- The automatically inferred size metric is *length* (list length) for all arguments.
- All arguments inferred to be input except last ones (output). No aliasing.
- Let $\langle b.i \rangle$ denote (a bound on) the size of the term(s) appearing in the $i^{th}$ argument position in the head of a clause defining predicate $b$.
- Let $\langle p.j.i \rangle$ denote (a bound on) the size of the term(s) appearing in the $i^{th}$ argument position in the $j^{th}$ body call of a clause defining predicate $b$.

  $\rightarrow$ $p$ denotes the predicate called (for readability).
- Example (2nd clause): $\langle app.1.1 \rangle = length($`L`$)$ and $\langle app.1 \rangle = length($`[H|L]`$)$.
- First, we consider predicate `app(A,B,C)` (third arg is output).
- We want to obtain intra-predicate argument size relations:
  $Sz_3^{app}(x, y)$ represents the size of the third argument of `app` as a function of its input data sizes ($x = length($`A`$)$ and $y = length($`B`$)$).

# Size Analysis (size relations): Example

```
:- entry nrev/2 : list(num) * var.

nrev([],[]).
nrev([H|L],R) :- nrev(L,R1), app(R1,[H],R).

app([],L,L).
app([H|L],L1,[H|R]) :- app(L,L1,R).
```

- The automatically inferred size metric is *length* (list length) for all arguments.
- All arguments inferred to be input except last ones (output). No aliasing.
- Let $\langle b.i \rangle$ denote (a bound on) the size of the term(s) appearing in the $i^{th}$ argument position in the head of a clause defining predicate $b$.
- Let $\langle p.j.i \rangle$ denote (a bound on) the size of the term(s) appearing in the $i^{th}$ argument position in the $j^{th}$ body call of a clause defining predicate $b$.

  $\rightarrow$ $p$ denotes the predicate called (for readability).
- Example (2nd clause): $\langle app.1.1 \rangle = length($`L`$)$ and $\langle app.1 \rangle = length($`[H|L]`$)$.
- First, we consider predicate `app(A,B,C)` (third arg is output).
- We want to obtain intra-predicate argument size relations:
  $Sz_3^{app}(x, y)$ represents the size of the third argument of `app` as a function of its input data sizes ($x = length($`A`$)$ and $y = length($`B`$)$).

# Size Analysis (size relations): Example

```
app([],L,L).
app([H|L],L1,[H|R]) :- app(L,L1,R).
```

- Argument size relations for the recursive clause:
  $\langle app.1.1\rangle = \langle app.1\rangle + \mathit{diff}(\texttt{L}, \texttt{[H|L]})$ *(inter-predicate)*

# Size Analysis (size relations): Example

```
app([],L,L).
app([H|L],L1,[H|R]) :- app(L,L1,R).
```

- Argument size relations for the recursive clause:
  $\mathit{length}(\texttt{L}) = \mathit{length}(\texttt{[H|L]}) + \mathit{diff}(\texttt{L}, \texttt{[H|L]})$

# Size Analysis (size relations): Example

```
app([],L,L).
app([H|L],L1,[H|R]) :- app(L,L1,R).
```

- Argument size relations for the recursive clause:
  $\mathit{length}(\texttt{L}) = \mathit{length}(\texttt{[H|L]}) - 1$

# Size Analysis (size relations): Example

```
app([],L,L).
app([H|L],L1,[H|R]) :- app(L,L1,R).
```

- Argument size relations for the recursive clause:
  $\langle app.1.1 \rangle = \langle app.1 \rangle - 1 \quad \equiv \quad length(\texttt{L}) = length(\texttt{[H|L]}) - 1$

# Size Analysis (size relations): Example

```
app([],L,L).
app([H|L],L1,[H|R]) :- app(L,L1,R).
```

- Argument size relations for the recursive clause:
  $\langle app.1.1 \rangle = \langle app.1 \rangle - 1$

# Size Analysis (size relations): Example

```
app([],L,L).
app([H|L],L1,[H|R]) :- app(L,L1,R).
```

- Argument size relations for the recursive clause:
  $\langle app.1.1 \rangle = \langle app.1 \rangle - 1$
  $\langle app.1.2 \rangle = \langle app.2 \rangle + \mathit{diff}(\mathtt{L1}, \mathtt{L1})$ *(inter-predicate)*

# Size Analysis (size relations): Example

```
app([],L,L).
app([H|L],L1,[H|R]) :- app(L,L1,R).
```

- Argument size relations for the recursive clause:
  $\langle app.1.1 \rangle = \langle app.1 \rangle - 1$
  $\langle app.1.2 \rangle = \langle app.2 \rangle \quad \equiv \quad length(\mathtt{L1}) = length(\mathtt{L1}) + 0$

# Size Analysis (size relations): Example

```
app([],L,L).
app([H|L],L1,[H|R]) :- app(L,L1,R).
```

- Argument size relations for the recursive clause:
  $\langle app.1.1 \rangle = \langle app.1 \rangle - 1$
  $\langle app.1.2 \rangle = \langle app.2 \rangle \quad \equiv \quad length(\mathtt{L1}) = length(\mathtt{L1})$

# Size Analysis (size relations): Example

```
app([],L,L).
app([H|L],L1,[H|R]) :- app(L,L1,R).
```

- Argument size relations for the recursive clause:
  $\langle app.1.1 \rangle = \langle app.1 \rangle - 1$
  $\langle app.1.2 \rangle = \langle app.2 \rangle$
  $\langle app.1.3 \rangle = Sz_3^{app}(\langle app.1.1 \rangle, \langle app.1.2 \rangle)$ *(intra-predicate)*

# Size Analysis (size relations): Example

```
app([],L,L).
app([H|L],L1,[H|R]) :- app(L,L1,R).
```

- Argument size relations for the recursive clause:
  $\langle app.1.1\rangle = \langle app.1\rangle - 1$
  $\langle app.1.2\rangle = \langle app.2\rangle$
  $\langle app.1.3\rangle = Sz_3^{app}(\langle app.1\rangle - 1, \langle app.1.2\rangle)$ *(normalizing)*

# Size Analysis (size relations): Example

```
app([],L,L).
app([H|L],L1,[H|R]) :- app(L,L1,R).
```

- Argument size relations for the recursive clause:
  $\langle app.1.1\rangle = \langle app.1\rangle - 1$
  $\langle app.1.2\rangle = \langle app.2\rangle$
  $\langle app.1.3\rangle = Sz_3^{app}(\langle app.1\rangle - 1, \langle app.2\rangle)$ *(normalizing)*

# Size Analysis (size relations): Example

```
app([],L,L).
app([H|L],L1,[H|R]) :- app(L,L1,R).
```

- Argument size relations for the recursive clause:

  $\langle app.1.1 \rangle = \langle app.1 \rangle - 1$

  $\langle app.1.2 \rangle = \langle app.2 \rangle$

  $\langle app.1.3 \rangle = Sz_3^{app}(\langle app.1 \rangle - 1, \langle app.2 \rangle)$

  $\langle app.3 \rangle = \langle app.1.3 \rangle + diff(\texttt{[H|R]}, \texttt{R})$ *(inter-predicate)*

# Size Analysis (size relations): Example

```
app([],L,L).
app([H|L],L1,[H|R]) :- app(L,L1,R).
```

- Argument size relations for the recursive clause:

  $\langle app.1.1\rangle = \langle app.1\rangle - 1$

  $\langle app.1.2\rangle = \langle app.2\rangle$

  $\langle app.1.3\rangle = Sz_3^{app}(\langle app.1\rangle - 1, \langle app.2\rangle)$

  $\langle app.3\rangle = \langle app.1.3\rangle + 1$

# Size Analysis (size relations): Example

```
app([],L,L).
app([H|L],L1,[H|R]) :- app(L,L1,R).
```

- Argument size relations for the recursive clause:
  $\langle app.1.1\rangle = \langle app.1\rangle - 1$
  $\langle app.1.2\rangle = \langle app.2\rangle$
  $\langle app.1.3\rangle = Sz_3^{app}(\langle app.1\rangle - 1, \langle app.2\rangle)$
  $\langle app.3\rangle = Sz_3^{app}(\langle app.1\rangle - 1, \langle app.2\rangle) + 1$ *(normalizing)*

# Size Analysis (size relations): Example

```
app([],L,L).
app([H|L],L1,[H|R]) :- app(L,L1,R).
```

- Argument size relations for the recursive clause:

  $\langle app.1.1\rangle = \langle app.1\rangle - 1$

  $\langle app.1.2\rangle = \langle app.2\rangle$

  $\langle app.1.3\rangle = Sz_3^{app}(\langle app.1\rangle - 1, \langle app.2\rangle)$

  $\langle app.3\rangle = Sz_3^{app}(\langle app.1\rangle - 1, \langle app.2\rangle) + 1$

  $Sz_3^{app}(\langle app.1\rangle, \langle app.2\rangle) = \langle app.3\rangle$ *(intra-predicate)*

# Size Analysis (size relations): Example

```
app([],L,L).
app([H|L],L1,[H|R]) :- app(L,L1,R).
```

- Argument size relations for the recursive clause:

  $\langle app.1.1 \rangle = \langle app.1 \rangle - 1$

  $\langle app.1.2 \rangle = \langle app.2 \rangle$

  $\langle app.1.3 \rangle = Sz_3^{app}(\langle app.1 \rangle - 1, \langle app.2 \rangle)$

  $\langle app.3 \rangle = Sz_3^{app}(\langle app.1 \rangle - 1, \langle app.2 \rangle) + 1$

  $Sz_3^{app}(\langle app.1 \rangle, \langle app.2 \rangle) = Sz_3^{app}(\langle app.1 \rangle - 1, \langle app.2 \rangle) + 1 \quad \textit{(normalizing)}$

# Size Analysis (size relations): Example

```
app([],L,L).
app([H|L],L1,[H|R]) :- app(L,L1,R).
```

- Argument size relations for the recursive clause:

  $\langle app.1.1\rangle = \langle app.1\rangle - 1$

  $\langle app.1.2\rangle = \langle app.2\rangle$

  $\langle app.1.3\rangle = Sz_3^{app}(\langle app.1\rangle - 1, \langle app.2\rangle)$

  $\langle app.3\rangle = Sz_3^{app}(\langle app.1\rangle - 1, \langle app.2\rangle) + 1$

  $Sz_3^{app}(\langle app.1\rangle, \langle app.2\rangle) = Sz_3^{app}(\langle app.1\rangle - 1, \langle app.2\rangle) + 1$

# Size Analysis (size relations): Example

```
app([],L,L).
app([H|L],L1,[H|R]) :- app(L,L1,R).
```

- Argument size relations for the recursive clause:

  $\langle app.1.1\rangle = \langle app.1\rangle - 1$

  $\langle app.1.2\rangle = \langle app.2\rangle$

  $\langle app.1.3\rangle = Sz_3^{app}(\langle app.1\rangle - 1, \langle app.2\rangle)$

  $\langle app.3\rangle = Sz_3^{app}(\langle app.1\rangle - 1, \langle app.2\rangle) + 1$

  $Sz_3^{app}(\langle app.1\rangle, \langle app.2\rangle) = Sz_3^{app}(\langle app.1\rangle - 1, \langle app.2\rangle) + 1$

- From the first clause of `app`, we obtain the equation:

  $Sz_3^{app}(0, \langle app.2\rangle) = \langle app.2\rangle$

# Size Analysis (size relations): Example

```
app([],L,L).
app([H|L],L1,[H|R]) :- app(L,L1,R).
```

- Argument size relations for the recursive clause:
  $\langle app.1.1\rangle = \langle app.1\rangle - 1$
  $\langle app.1.2\rangle = \langle app.2\rangle$
  $\langle app.1.3\rangle = Sz_3^{app}(\langle app.1\rangle - 1, \langle app.2\rangle)$
  $\langle app.3\rangle = Sz_3^{app}(\langle app.1\rangle - 1, \langle app.2\rangle) + 1$
  $Sz_3^{app}(\langle app.1\rangle, \langle app.2\rangle) = Sz_3^{app}(\langle app.1\rangle - 1, \langle app.2\rangle) + 1$
- From the first clause of `app`, we obtain the equation:
  $Sz_3^{app}(0, \langle app.2\rangle) = \langle app.2\rangle$
- The equations:
  $Sz_3^{app}(0, \langle app.2\rangle) = \langle app.2\rangle$
  $Sz_3^{app}(\langle app.1\rangle, \langle app.2\rangle) = Sz_3^{app}(\langle app.1\rangle - 1, \langle app.2\rangle) + 1$

# Size Analysis (size relations): Example

```
app([],L,L).
app([H|L],L1,[H|R]) :- app(L,L1,R).
```

- Argument size relations for the recursive clause:
  $\langle app.1.1 \rangle = \langle app.1 \rangle - 1$
  $\langle app.1.2 \rangle = \langle app.2 \rangle$
  $\langle app.1.3 \rangle = Sz_3^{app}(\langle app.1 \rangle - 1, \langle app.2 \rangle)$
  $\langle app.3 \rangle = Sz_3^{app}(\langle app.1 \rangle - 1, \langle app.2 \rangle) + 1$
  $Sz_3^{app}(\langle app.1 \rangle, \langle app.2 \rangle) = Sz_3^{app}(\langle app.1 \rangle - 1, \langle app.2 \rangle) + 1$
- From the first clause of `app`, we obtain the equation:
  $Sz_3^{app}(0, \langle app.2 \rangle) = \langle app.2 \rangle$
- The equations ($n = \langle app.1 \rangle$, $m = \langle app.2 \rangle$):
  $Sz_3^{app}(n, m) = m \quad \text{if } n = 0$
  $Sz_3^{app}(n, m) = Sz_3^{app}(n-1, m) + 1 \quad \text{if } n > 0$

# Size Analysis (size relations): Example

```
app([],L,L).
app([H|L],L1,[H|R]) :- app(L,L1,R).
```

- Argument size relations for the recursive clause:
  $\langle app.1.1\rangle = \langle app.1\rangle - 1$
  $\langle app.1.2\rangle = \langle app.2\rangle$
  $\langle app.1.3\rangle = Sz_3^{app}(\langle app.1\rangle - 1, \langle app.2\rangle)$
  $\langle app.3\rangle = Sz_3^{app}(\langle app.1\rangle - 1, \langle app.2\rangle) + 1$
  $Sz_3^{app}(\langle app.1\rangle, \langle app.2\rangle) = Sz_3^{app}(\langle app.1\rangle - 1, \langle app.2\rangle) + 1$
- From the first clause of `app`, we obtain the equation:
  $Sz_3^{app}(0, \langle app.2\rangle) = \langle app.2\rangle$
- The equations ($n = \langle app.1\rangle$, $m = \langle app.2\rangle$):
  $Sz_3^{app}(n, m) = m \quad \text{if } n = 0$
  $Sz_3^{app}(n, m) = Sz_3^{app}(n-1, m) + 1 \quad \text{if } n > 0$
  are solved, obtaining the closed-form function:
  $Sz_3^{app}(n, m) = n + m \quad \text{if } n \geq 0$

# Size Analysis (size relations): Example

```
app([],L,L).
app([H|L],L1,[H|R]) :- app(L,L1,R).
```

- Argument size relations for the recursive clause:
  $\langle app.1.1\rangle = \langle app.1\rangle - 1$
  $\langle app.1.2\rangle = \langle app.2\rangle$
  $\langle app.1.3\rangle = Sz_3^{app}(\langle app.1\rangle - 1, \langle app.2\rangle)$
  $\langle app.3\rangle = Sz_3^{app}(\langle app.1\rangle - 1, \langle app.2\rangle) + 1$
  $Sz_3^{app}(\langle app.1\rangle, \langle app.2\rangle) = Sz_3^{app}(\langle app.1\rangle - 1, \langle app.2\rangle) + 1$
- From the first clause of `app`, we obtain the equation:
  $Sz_3^{app}(0, \langle app.2\rangle) = \langle app.2\rangle$
- The equations ($n = \langle app.1\rangle$, $m = \langle app.2\rangle$):
  $Sz_3^{app}(n, m) = m \quad \text{if } n = 0$
  $Sz_3^{app}(n, m) = Sz_3^{app}(n-1, m) + 1 \quad \text{if } n > 0$
  are solved, obtaining the closed-form function:
  $Sz_3^{app}(n, m) = n + m \quad \text{if } n \geq 0$
  which is used for the analysis of predicate `nrev`.

# Size Analysis (size relations): Example

```
nrev([],[]).
nrev([H|L],R) :- nrev(L,R1), app(R1,[H],R).
```

- We now switch to predicate `nrev(A, B)`, where `A` and `B` are input and output arguments respectively.
- Argument size relations for the recursive clause:
  $\langle \mathit{nrev}.1.1 \rangle = \langle \mathit{nrev}.1 \rangle + \mathit{diff}(\mathtt{L}, \mathtt{[H|L]})$ *(inter-predicate)*

# Size Analysis (size relations): Example

```
nrev([],[]).
nrev([H|L],R) :- nrev(L,R1), app(R1,[H],R).
```

- We now switch to predicate `nrev(A, B)`, where `A` and `B` are input and output arguments respectively.
- Argument size relations for the recursive clause:
  $\mathit{length}(\texttt{L}) = \mathit{length}(\texttt{[H|L]}) + \mathit{diff}(\texttt{L}, \texttt{[H|L]})$

# Size Analysis (size relations): Example

```
nrev([],[]).
nrev([H|L],R) :- nrev(L,R1), app(R1,[H],R).
```

- We now switch to predicate `nrev(A, B)`, where `A` and `B` are input and output arguments respectively.
- Argument size relations for the recursive clause:
  $\mathit{length}(\texttt{L}) = \mathit{length}(\texttt{[H|L]}) - 1$

# Size Analysis (size relations): Example

```
nrev([],[]).
nrev([H|L],R) :- nrev(L,R1), app(R1,[H],R).
```

- We now switch to predicate `nrev(A, B)`, where `A` and `B` are input and output arguments respectively.
- Argument size relations for the recursive clause:
  $\langle nrev.1.1\rangle = \langle nrev.1\rangle - 1 \quad \equiv \quad length(\mathtt{L}) = length(\mathtt{[H|L]}) - 1$

# Size Analysis (size relations): Example

```
nrev([],[]).
nrev([H|L],R) :- nrev(L,R1), app(R1,[H],R).
```

- We now switch to predicate `nrev(A, B)`, where `A` and `B` are input and output arguments respectively.
- Argument size relations for the recursive clause:
  $\langle nrev.1.1\rangle = \langle nrev.1\rangle - 1$

# Size Analysis (size relations): Example

```
nrev([],[]).
nrev([H|L],R) :- nrev(L,R1), app(R1,[H],R).
```

- We now switch to predicate `nrev(A, B)`, where `A` and `B` are input and output arguments respectively.
- Argument size relations for the recursive clause:
  $\langle nrev.1.1 \rangle = \langle nrev.1 \rangle - 1$
  $\langle nrev.1.2 \rangle = Sz_2^{nrev}(\langle nrev.1.1 \rangle)$ *(intra-predicate)*

# Size Analysis (size relations): Example

```
nrev([],[]).
nrev([H|L],R) :- nrev(L,R1), app(R1,[H],R).
```

- We now switch to predicate `nrev(A, B)`, where `A` and `B` are input and output arguments respectively.
- Argument size relations for the recursive clause:
  $\langle nrev.1.1\rangle = \langle nrev.1\rangle - 1$
  $\langle nrev.1.2\rangle = Sz_2^{nrev}(\langle nrev.1.1\rangle) \quad \equiv \quad length(\mathtt{R1}) = Sz_2^{nrev}(length(\mathtt{L}))$

# Size Analysis (size relations): Example

```
nrev([],[]).
nrev([H|L],R) :- nrev(L,R1), app(R1,[H],R).
```

- We now switch to predicate `nrev(A, B)`, where `A` and `B` are input and output arguments respectively.
- Argument size relations for the recursive clause:
  $\langle nrev.1.1 \rangle = \langle nrev.1 \rangle - 1$
  $\langle nrev.1.2 \rangle = Sz_2^{nrev}(\langle nrev.1 \rangle - 1)$ *(normalizing)*

# Size Analysis (size relations): Example

```
nrev([],[]).
nrev([H|L],R) :- nrev(L,R1), app(R1,[H],R).
```

- We now switch to predicate `nrev(A, B)`, where `A` and `B` are input and output arguments respectively.
- Argument size relations for the recursive clause:
  $\langle nrev.1.1 \rangle = \langle nrev.1 \rangle - 1$
  $\langle nrev.1.2 \rangle = Sz_2^{nrev}(\langle nrev.1 \rangle - 1)$
  $\langle app.2.1 \rangle = \langle nrev.1.2 \rangle$ *(inter-predicate)*

# Size Analysis (size relations): Example

```
nrev([],[]).
nrev([H|L],R) :- nrev(L,R1), app(R1,[H],R).
```

- We now switch to predicate `nrev(A, B)`, where `A` and `B` are input and output arguments respectively.
- Argument size relations for the recursive clause:
  $\langle nrev.1.1 \rangle = \langle nrev.1 \rangle - 1$
  $\langle nrev.1.2 \rangle = Sz_2^{nrev}(\langle nrev.1 \rangle - 1)$
  $\langle app.2.1 \rangle = \langle nrev.1.2 \rangle \quad \equiv \quad length(\texttt{R1}) = length(\texttt{R1})$

# Size Analysis (size relations): Example

```
nrev([],[]).
nrev([H|L],R) :- nrev(L,R1), app(R1,[H],R).
```

- We now switch to predicate `nrev(A, B)`, where `A` and `B` are input and output arguments respectively.
- Argument size relations for the recursive clause:

  $\langle nrev.1.1 \rangle = \langle nrev.1 \rangle - 1$

  $\langle nrev.1.2 \rangle = Sz_2^{nrev}(\langle nrev.1 \rangle - 1)$

  $\langle app.2.1 \rangle = Sz_2^{nrev}(\langle nrev.1 \rangle - 1)$ *(normalizing)*

# Size Analysis (size relations): Example

```
nrev([],[]).
nrev([H|L],R) :- nrev(L,R1), app(R1,[H],R).
```

- We now switch to predicate `nrev(A, B)`, where `A` and `B` are input and output arguments respectively.
- Argument size relations for the recursive clause:
  $\langle nrev.1.1 \rangle = \langle nrev.1 \rangle - 1$
  $\langle nrev.1.2 \rangle = Sz_2^{nrev}(\langle nrev.1 \rangle - 1)$
  $\langle app.2.1 \rangle = Sz_2^{nrev}(\langle nrev.1 \rangle - 1)$
  $\langle app.2.2 \rangle = length($`[H]`$)$ *(explicit size)*

# Size Analysis (size relations): Example

```
nrev([],[]).
nrev([H|L],R) :- nrev(L,R1), app(R1,[H],R).
```

- We now switch to predicate `nrev(A, B)`, where `A` and `B` are input and output arguments respectively.
- Argument size relations for the recursive clause:

  $\langle nrev.1.1 \rangle = \langle nrev.1 \rangle - 1$

  $\langle nrev.1.2 \rangle = Sz_2^{nrev}(\langle nrev.1 \rangle - 1)$

  $\langle app.2.1 \rangle = Sz_2^{nrev}(\langle nrev.1 \rangle - 1)$

  $\langle app.2.2 \rangle = 1$

# Size Analysis (size relations): Example

```
nrev([],[]).
nrev([H|L],R) :- nrev(L,R1), app(R1,[H],R).
```

- We now switch to predicate `nrev(A, B)`, where `A` and `B` are input and output arguments respectively.
- Argument size relations for the recursive clause:

$$\langle nrev.1.1 \rangle = \langle nrev.1 \rangle - 1$$
$$\langle nrev.1.2 \rangle = Sz_2^{nrev}(\langle nrev.1 \rangle - 1)$$
$$\langle app.2.1 \rangle = Sz_2^{nrev}(\langle nrev.1 \rangle - 1)$$
$$\langle app.2.2 \rangle = 1$$
$$\langle app.2.3 \rangle = Sz_3^{app}(\langle app.2.1 \rangle, \langle app.2.2 \rangle) \quad \textit{(intra-predicate)}$$

# Size Analysis (size relations): Example

```
nrev([],[]).
nrev([H|L],R) :- nrev(L,R1), app(R1,[H],R).
```

- We now switch to predicate `nrev(A, B)`, where `A` and `B` are input and output arguments respectively.
- Argument size relations for the recursive clause:

$$\langle nrev.1.1\rangle = \langle nrev.1\rangle - 1$$
$$\langle nrev.1.2\rangle = Sz_2^{nrev}(\langle nrev.1\rangle - 1)$$
$$\langle app.2.1\rangle = Sz_2^{nrev}(\langle nrev.1\rangle - 1)$$
$$\langle app.2.2\rangle = 1$$
$$\langle app.2.3\rangle = \langle app.2.1\rangle + \langle app.2.2\rangle \quad \textit{using} \quad Sz_3^{app}(x, y) = x + y$$

# Size Analysis (size relations): Example

```
nrev([],[]).
nrev([H|L],R) :- nrev(L,R1), app(R1,[H],R).
```

- We now switch to predicate `nrev(A, B)`, where `A` and `B` are input and output arguments respectively.
- Argument size relations for the recursive clause:

$$\langle nrev.1.1\rangle = \langle nrev.1\rangle - 1$$
$$\langle nrev.1.2\rangle = Sz_2^{nrev}(\langle nrev.1\rangle - 1)$$
$$\langle app.2.1\rangle = Sz_2^{nrev}(\langle nrev.1\rangle - 1)$$
$$\langle app.2.2\rangle = 1$$
$$\langle app.2.3\rangle = Sz_2^{nrev}(\langle nrev.1\rangle - 1) + 1 \quad \textit{(normalizing)}$$

# Size Analysis (size relations): Example

```
nrev([],[]).
nrev([H|L],R) :- nrev(L,R1), app(R1,[H],R).
```

- We now switch to predicate `nrev(A, B)`, where `A` and `B` are input and output arguments respectively.
- Argument size relations for the recursive clause:

  $\langle nrev.1.1\rangle = \langle nrev.1\rangle - 1$
  $\langle nrev.1.2\rangle = Sz_2^{nrev}(\langle nrev.1\rangle - 1)$
  $\langle app.2.1\rangle = Sz_2^{nrev}(\langle nrev.1\rangle - 1)$
  $\langle app.2.2\rangle = 1$
  $\langle app.2.3\rangle = Sz_2^{nrev}(\langle nrev.1\rangle - 1) + 1$
  $\langle nrev.2\rangle = \langle app.2.3\rangle + diff(\mathtt{R}, \mathtt{R})$ *(inter-predicate)*

# Size Analysis (size relations): Example

```
nrev([],[]).
nrev([H|L],R) :- nrev(L,R1), app(R1,[H],R).
```

- We now switch to predicate `nrev(A, B)`, where `A` and `B` are input and output arguments respectively.
- Argument size relations for the recursive clause:

$$\langle nrev.1.1 \rangle = \langle nrev.1 \rangle - 1$$
$$\langle nrev.1.2 \rangle = Sz_2^{nrev}(\langle nrev.1 \rangle - 1)$$
$$\langle app.2.1 \rangle = Sz_2^{nrev}(\langle nrev.1 \rangle - 1)$$
$$\langle app.2.2 \rangle = 1$$
$$\langle app.2.3 \rangle = Sz_2^{nrev}(\langle nrev.1 \rangle - 1) + 1$$
$$\langle nrev.2 \rangle = \langle app.2.3 \rangle + 0$$

# Size Analysis (size relations): Example

```
nrev([],[]).
nrev([H|L],R) :- nrev(L,R1), app(R1,[H],R).
```

- We now switch to predicate `nrev(A, B)`, where `A` and `B` are input and output arguments respectively.
- Argument size relations for the recursive clause:

$\langle nrev.1.1\rangle = \langle nrev.1\rangle - 1$
$\langle nrev.1.2\rangle = Sz_2^{nrev}(\langle nrev.1\rangle - 1)$
$\langle app.2.1\rangle = Sz_2^{nrev}(\langle nrev.1\rangle - 1)$
$\langle app.2.2\rangle = 1$
$\langle app.2.3\rangle = Sz_2^{nrev}(\langle nrev.1\rangle - 1) + 1$
$\langle nrev.2\rangle = Sz_2^{nrev}(\langle nrev.1\rangle - 1) + 1 \quad$ *(normalizing)*

# Size Analysis (size relations): Example

```
nrev([],[]).
nrev([H|L],R) :- nrev(L,R1), app(R1,[H],R).
```

- We now switch to predicate `nrev(A, B)`, where `A` and `B` are input and output arguments respectively.
- Argument size relations for the recursive clause:

$$\langle nrev.1.1\rangle = \langle nrev.1\rangle - 1$$
$$\langle nrev.1.2\rangle = Sz_2^{nrev}(\langle nrev.1\rangle - 1)$$
$$\langle app.2.1\rangle = Sz_2^{nrev}(\langle nrev.1\rangle - 1)$$
$$\langle app.2.2\rangle = 1$$
$$\langle app.2.3\rangle = Sz_2^{nrev}(\langle nrev.1\rangle - 1) + 1$$
$$\langle nrev.2\rangle = Sz_2^{nrev}(\langle nrev.1\rangle - 1) + 1$$
$$Sz_2^{nrev}(\langle nrev.1\rangle) = \langle nrev.2\rangle \quad \textit{(intra-predicate)}$$

# Size Analysis (size relations): Example

```
nrev([],[]).
nrev([H|L],R) :- nrev(L,R1), app(R1,[H],R).
```

- We now switch to predicate `nrev(A, B)`, where `A` and `B` are input and output arguments respectively.
- Argument size relations for the recursive clause:

$$\langle nrev.1.1\rangle = \langle nrev.1\rangle - 1$$
$$\langle nrev.1.2\rangle = Sz_2^{nrev}(\langle nrev.1\rangle - 1)$$
$$\langle app.2.1\rangle = Sz_2^{nrev}(\langle nrev.1\rangle - 1)$$
$$\langle app.2.2\rangle = 1$$
$$\langle app.2.3\rangle = Sz_2^{nrev}(\langle nrev.1\rangle - 1) + 1$$
$$\langle nrev.2\rangle = Sz_2^{nrev}(\langle nrev.1\rangle - 1) + 1$$
$$Sz_2^{nrev}(\langle nrev.1\rangle) = Sz_2^{nrev}(\langle nrev.1\rangle - 1) + 1 \quad \textit{(normalizing)}$$

# Size Analysis (size relations): Example

```
nrev([],[]).
nrev([H|L],R) :- nrev(L,R1), app(R1,[H],R).
```

- We now switch to predicate `nrev(A, B)`, where `A` and `B` are input and output arguments respectively.
- Argument size relations for the recursive clause:

$$\langle nrev.1.1\rangle = \langle nrev.1\rangle - 1$$
$$\langle nrev.1.2\rangle = Sz_2^{nrev}(\langle nrev.1\rangle - 1)$$
$$\langle app.2.1\rangle = Sz_2^{nrev}(\langle nrev.1\rangle - 1)$$
$$\langle app.2.2\rangle = 1$$
$$\langle app.2.3\rangle = Sz_2^{nrev}(\langle nrev.1\rangle - 1) + 1$$
$$\langle nrev.2\rangle = Sz_2^{nrev}(\langle nrev.1\rangle - 1) + 1$$
$$Sz_2^{nrev}(\langle nrev.1\rangle) = Sz_2^{nrev}(\langle nrev.1\rangle - 1) + 1$$

# Size Analysis (size relations): Example

```
nrev([],[]).
nrev([H|L],R) :- nrev(L,R1), app(R1,[H],R).
```

- We now switch to predicate `nrev(A, B)`, where `A` and `B` are input and output arguments respectively.
- Argument size relations for the recursive clause:

  $\langle nrev.1.1 \rangle = \langle nrev.1 \rangle - 1$
  $\langle nrev.1.2 \rangle = Sz_2^{nrev}(\langle nrev.1 \rangle - 1)$
  $\langle app.2.1 \rangle = Sz_2^{nrev}(\langle nrev.1 \rangle - 1)$
  $\langle app.2.2 \rangle = 1$
  $\langle app.2.3 \rangle = Sz_2^{nrev}(\langle nrev.1 \rangle - 1) + 1$
  $\langle nrev.2 \rangle = Sz_2^{nrev}(\langle nrev.1 \rangle - 1) + 1$
  $Sz_2^{nrev}(\langle nrev.1 \rangle) = Sz_2^{nrev}(\langle nrev.1 \rangle - 1) + 1$
- From the first clause of `nrev`, we obtain the equation:
  $Sz_2^{nrev}(0) = 0$

# Size Analysis (size relations): Example

```
nrev([],[]).
nrev([H|L],R) :- nrev(L,R1), app(R1,[H],R).
```

- We now switch to predicate `nrev(A, B)`, where `A` and `B` are input and output arguments respectively.
- Argument size relations for the recursive clause:

  $\langle nrev.1.1 \rangle = \langle nrev.1 \rangle - 1$
  $\langle nrev.1.2 \rangle = Sz_2^{nrev}(\langle nrev.1 \rangle - 1)$
  $\langle app.2.1 \rangle = Sz_2^{nrev}(\langle nrev.1 \rangle - 1)$
  $\langle app.2.2 \rangle = 1$
  $\langle app.2.3 \rangle = Sz_2^{nrev}(\langle nrev.1 \rangle - 1) + 1$
  $\langle nrev.2 \rangle = Sz_2^{nrev}(\langle nrev.1 \rangle - 1) + 1$
  $Sz_2^{nrev}(\langle nrev.1 \rangle) = Sz_2^{nrev}(\langle nrev.1 \rangle - 1) + 1$
- From the first clause of `nrev`, we obtain the equation:
  $Sz_2^{nrev}(0) = 0$
- The equations:

  $Sz_2^{nrev}(0) = 0$
  $Sz_2^{nrev}(\langle nrev.1 \rangle) = Sz_2^{nrev}(\langle nrev.1 \rangle - 1) + 1$

# Size Analysis (size relations): Example

```
nrev([],[]).
nrev([H|L],R) :- nrev(L,R1), app(R1,[H],R).
```

- We now switch to predicate `nrev(A, B)`, where `A` and `B` are input and output arguments respectively.
- Argument size relations for the recursive clause:
  $\langle nrev.1.1 \rangle = \langle nrev.1 \rangle - 1$
  $\langle nrev.1.2 \rangle = Sz_2^{nrev}(\langle nrev.1 \rangle - 1)$
  $\langle app.2.1 \rangle = Sz_2^{nrev}(\langle nrev.1 \rangle - 1)$
  $\langle app.2.2 \rangle = 1$
  $\langle app.2.3 \rangle = Sz_2^{nrev}(\langle nrev.1 \rangle - 1) + 1$
  $\langle nrev.2 \rangle = Sz_2^{nrev}(\langle nrev.1 \rangle - 1) + 1$
  $Sz_2^{nrev}(\langle nrev.1 \rangle) = Sz_2^{nrev}(\langle nrev.1 \rangle - 1) + 1$
- From the first clause of `nrev`, we obtain the equation:
  $Sz_2^{nrev}(0) = 0$
- The equations ($n = \langle nrev.1 \rangle$):
  $Sz_2^{nrev}(0) = 0$
  $Sz_2^{nrev}(n) = Sz_2^{nrev}(n-1) + 1$

# Size Analysis (size relations): Example

```
nrev([],[]).
nrev([H|L],R) :- nrev(L,R1), app(R1,[H],R).
```

- We now switch to predicate `nrev(A, B)`, where `A` and `B` are input and output arguments respectively.
- Argument size relations for the recursive clause:
  $\langle nrev.1.1 \rangle = \langle nrev.1 \rangle - 1$
  $\langle nrev.1.2 \rangle = Sz_2^{nrev}(\langle nrev.1 \rangle - 1)$
  $\langle app.2.1 \rangle = Sz_2^{nrev}(\langle nrev.1 \rangle - 1)$
  $\langle app.2.2 \rangle = 1$
  $\langle app.2.3 \rangle = Sz_2^{nrev}(\langle nrev.1 \rangle - 1) + 1$
  $\langle nrev.2 \rangle = Sz_2^{nrev}(\langle nrev.1 \rangle - 1) + 1$
  $Sz_2^{nrev}(\langle nrev.1 \rangle) = Sz_2^{nrev}(\langle nrev.1 \rangle - 1) + 1$
- From the first clause of `nrev`, we obtain the equation:
  $Sz_2^{nrev}(0) = 0$
- The equations ($n = \langle nrev.1 \rangle$):
  $Sz_2^{nrev}(0) = 0$
  $Sz_2^{nrev}(n) = Sz_2^{nrev}(n-1) + 1$
  are solved, obtaining the closed-form function:
  $Sz_2^{nrev}(n) = n$

# Size Analysis (size relations): Example

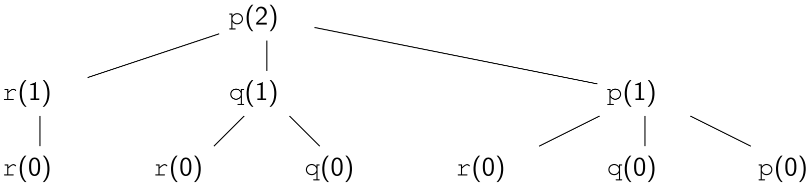
- The size of the output argument of `nrev(A, B)` is given by the following equations (where $n = length(\mathtt{A})$):
  $Sz_2^{nrev}(0) = 0$
  $Sz_2^{nrev}(n) = Sz_2^{nrev}(n-1) + 1$
- Solution: $Sz_2^{nrev}(n) = n$.
  The length (size) of the output argument of `nrev` is equal to the length of its input.

# Standard Cost: Intuition

```
p(0).
p(X):- X > 0, Y is X - 1, r(Y), q(Y), p(Y).

q(0).
q(X):- X > 0, Y is X - 1, r(Y), q(Y).

r(0).
r(X):- X > 0, Y is X - 1, r(Y).
```

The standard cost of a call $\mathtt{p}(2)$ (in number of resolution steps): $\mathtt{C_p}(2)$.

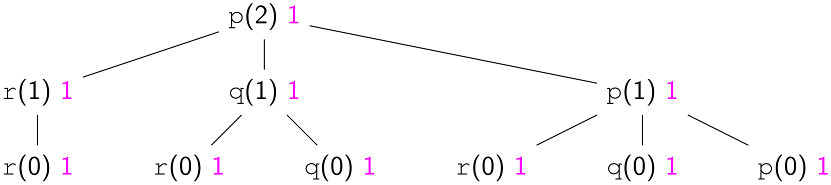
(assume the builtins $>$ /2 and *is*/2 have zero cost)



# Standard Cost: Intuition

```
p(0).
p(X):- X > 0, Y is X - 1, r(Y), q(Y), p(Y).

q(0).
q(X):- X > 0, Y is X - 1, r(Y), q(Y).

r(0).
r(X):- X > 0, Y is X - 1, r(Y).
```

The standard cost of a call $\mathtt{p}(2)$ (in number of resolution steps): $\mathtt{C}_{\mathtt{p}}(2) = 10$.
(also: $\mathtt{C}_{\mathtt{r}}(1) = 2$ and $\mathtt{C}_{\mathtt{q}}(1) = 3$).



# Standard Cost Relations Framework: Intuition

```
p(0).
p(X):- X > 0, Y is X - 1, r(Y), q(Y), p(Y).

q(0).
q(X):- X > 0, Y is X - 1, r(Y), q(Y).

r(0).
r(X):- X > 0, Y is X - 1, r(Y).
```

Cost relations $\quad n = size(X) = X$ (actual value of $X$)

Standard cost of `p`:

$$\begin{aligned} \mathtt{C_p}(0) &= 1 \\ \mathtt{C_p}(n) &= 1 + \mathtt{C_r}(n-1) + \mathtt{C_q}(n-1) + \mathtt{C_p}(n-1) \quad \text{if } n > 0 \end{aligned}$$

Standard cost of `q`:

$$\begin{aligned} \mathtt{C_q}(0) &= 1 \\ \mathtt{C_q}(n) &= 1 + \mathtt{C_r}(n-1) + \mathtt{C_q}(n-1) \quad \text{if } n > 0 \end{aligned}$$

Standard cost of `r`:

$$\begin{aligned} \mathtt{C_r}(0) &= 1 \\ \mathtt{C_r}(n) &= 1 + \mathtt{C_r}(n-1) \quad \text{if } n > 0 \end{aligned}$$

# Standard Cost Relations Framework: Intuition

```
p(0).
p(X):- X > 0, Y is X - 1, r(Y), q(Y), p(Y).

q(0).
q(X):- X > 0, Y is X - 1, r(Y), q(Y).

r(0).
r(X):- X > 0, Y is X - 1, r(Y).
```

Cost relations $\quad n = size(X) = X$ (actual value of $X$)

Standard cost of p:

$$\begin{aligned} \mathtt{C}_{\mathtt{p}}(0) &= 1 \\ \mathtt{C}_{\mathtt{p}}(n) &= 1 + \mathtt{C}_{\mathtt{r}}(n-1) + \mathtt{C}_{\mathtt{q}}(n-1) + \mathtt{C}_{\mathtt{p}}(n-1) \quad \text{if } n > 0 \end{aligned}$$

Standard cost of q:

$$\begin{aligned} \mathtt{C}_{\mathtt{q}}(0) &= 1 \\ \mathtt{C}_{\mathtt{q}}(n) &= 1 + \mathtt{C}_{\mathtt{r}}(n-1) + \mathtt{C}_{\mathtt{q}}(n-1) \quad \text{if } n > 0 \end{aligned}$$

Standard cost of r $\rightarrow$ closed-form: $\mathtt{C}_{\mathtt{r}}(n) = n + 1$, for $n \geq 0$.

$$\begin{aligned} \mathtt{C}_{\mathtt{r}}(0) &= 1 \\ \mathtt{C}_{\mathtt{r}}(n) &= 1 + \mathtt{C}_{\mathtt{r}}(n-1) \quad \text{if } n > 0 \end{aligned}$$

# Standard Cost Relations Framework: Intuition

```
p(0).
p(X):- X > 0, Y is X - 1, r(Y), q(Y), p(Y).

q(0).
q(X):- X > 0, Y is X - 1, r(Y), q(Y).

r(0).
r(X):- X > 0, Y is X - 1, r(Y).
```

Cost relations $\quad n = size(X) = X$ (actual value of $X$)

Standard cost of $\mathtt{p}$:

$$\begin{aligned} \mathtt{C}_{\mathtt{p}}(0) &= 1 \\ \mathtt{C}_{\mathtt{p}}(n) &= 1 + \mathtt{C}_{\mathtt{r}}(n-1) + \mathtt{C}_{\mathtt{q}}(n-1) + \mathtt{C}_{\mathtt{p}}(n-1) \quad \text{if } n > 0 \end{aligned}$$

Standard cost of $\mathtt{q}$:

$$\begin{aligned} \mathtt{C}_{\mathtt{q}}(0) &= 1 \\ \mathtt{C}_{\mathtt{q}}(n) &= 1 + n + \mathtt{C}_{\mathtt{q}}(n-1) \quad \text{if } n > 0 \end{aligned}$$

Standard cost of $\mathtt{r}$ $\rightarrow$ closed-form: $\mathtt{C}_{\mathtt{r}}(n) = n + 1$, for $n \geq 0$.

$$\begin{aligned} \mathtt{C}_{\mathtt{r}}(0) &= 1 \\ \mathtt{C}_{\mathtt{r}}(n) &= 1 + \mathtt{C}_{\mathtt{r}}(n-1) \quad \text{if } n > 0 \end{aligned}$$

# Standard Cost Relations Framework: Intuition

```
p(0).
p(X):- X > 0, Y is X - 1, r(Y), q(Y), p(Y).

q(0).
q(X):- X > 0, Y is X - 1, r(Y), q(Y).

r(0).
r(X):- X > 0, Y is X - 1, r(Y).
```

Cost relations $\quad n = size(X) = X$ (actual value of $X$)

Standard cost of `p`:

$$\begin{aligned} \mathtt{C}_{\mathtt{p}}(0) = &\quad 1 \\ \mathtt{C}_{\mathtt{p}}(n) = &\quad 1 + \mathtt{C}_{\mathtt{r}}(n-1) + \mathtt{C}_{\mathtt{q}}(n-1) + \mathtt{C}_{\mathtt{p}}(n-1) \quad \text{if } n > 0 \end{aligned}$$

Standard cost of `q` $\rightarrow$ closed form: $\mathtt{C}_{\mathtt{q}}(n) = \frac{1}{2}\, n^2 + \frac{3}{2}\, n + 2$ for $n \geq 0$.

$$\begin{aligned} \mathtt{C}_{\mathtt{q}}(0) = &\quad 1 \\ \mathtt{C}_{\mathtt{q}}(n) = &\quad 1 + n + \mathtt{C}_{\mathtt{q}}(n-1) \quad \text{if } n > 0 \end{aligned}$$

Standard cost of `r` $\rightarrow$ closed-form: $\mathtt{C}_{\mathtt{r}}(n) = n + 1$, for $n \geq 0$.

$$\begin{aligned} \mathtt{C}_{\mathtt{r}}(0) = &\quad 1 \\ \mathtt{C}_{\mathtt{r}}(n) = &\quad 1 + \mathtt{C}_{\mathtt{r}}(n-1) \quad \text{if } n > 0 \end{aligned}$$

# Standard Cost Relations Framework: Intuition

```
p(0).
p(X):- X > 0, Y is X - 1, r(Y), q(Y), p(Y).

q(0).
q(X):- X > 0, Y is X - 1, r(Y), q(Y).

r(0).
r(X):- X > 0, Y is X - 1, r(Y).
```

Cost relations $n = size(X) = X$ (actual value of $X$)

Standard cost of `p`:

$$\mathrm{C_p}(0) = 1$$
$$\mathrm{C_p}(n) = 1 + n + \frac{1}{2}(n-1)^2 + \frac{3}{2}(n-1) + 2 + \mathrm{C_p}(n-1) \quad \text{if } n > 0$$

Standard cost of `q` $\rightarrow$ closed form: $\mathrm{C_q}(n) = \frac{1}{2}n^2 + \frac{3}{2}n + 2$ for $n \geq 0$.

$$\mathrm{C_q}(0) = 1$$
$$\mathrm{C_q}(n) = 1 + n + \mathrm{C_q}(n-1) \quad \text{if } n > 0$$

Standard cost of `r` $\rightarrow$ closed-form: $\mathrm{C_r}(n) = n + 1$, for $n \geq 0$.

$$\mathrm{C_r}(0) = 1$$
$$\mathrm{C_r}(n) = 1 + \mathrm{C_r}(n-1) \quad \text{if } n > 0$$

# Standard Cost Relations Framework: Intuition

```
p(0).
p(X):- X > 0, Y is X - 1, r(Y), q(Y), p(Y).

q(0).
q(X):- X > 0, Y is X - 1, r(Y), q(Y).

r(0).
r(X):- X > 0, Y is X - 1, r(Y).
```

Cost relations $n = size(X) = X$ (actual value of $X$)

Standard cost of $\mathtt{p}$ → closed form: $\mathtt{C}_{\mathtt{p}}(n) = \frac{1}{6}n^3 + n^2 + \frac{17}{6}n + 1$, for $n \geq 0$.

$$\begin{aligned} \mathtt{C}_{\mathtt{p}}(0) = & \ 1 \\ \mathtt{C}_{\mathtt{p}}(n) = & \ 1 + n + \frac{1}{2}\ (n-1)^2 + \frac{3}{2}\ (n-1) + 2 + \mathtt{C}_{\mathtt{p}}(n-1) \quad \text{if } n > 0 \end{aligned}$$

Standard cost of $\mathtt{q}$ → closed form: $\mathtt{C}_{\mathtt{q}}(n) = \frac{1}{2}\ n^2 + \frac{3}{2}\ n + 2$ for $n \geq 0$.

$$\begin{aligned} \mathtt{C}_{\mathtt{q}}(0) = & \ 1 \\ \mathtt{C}_{\mathtt{q}}(n) = & \ 1 + n + \mathtt{C}_{\mathtt{q}}(n-1) \quad \text{if } n > 0 \end{aligned}$$

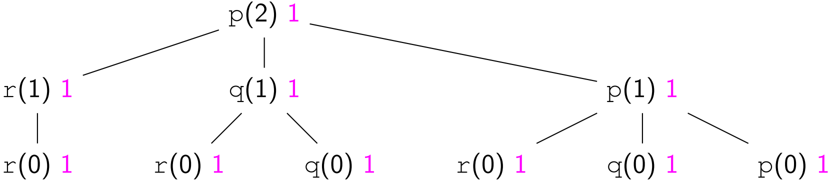
Standard cost of $\mathtt{r}$ → closed-form: $\mathtt{C}_{\mathtt{r}}(n) = n + 1$, for $n \geq 0$.

$$\begin{aligned} \mathtt{C}_{\mathtt{r}}(0) = & \ 1 \\ \mathtt{C}_{\mathtt{r}}(n) = & \ 1 + \mathtt{C}_{\mathtt{r}}(n-1) \quad \text{if } n > 0 \end{aligned}$$

# Accumulated-cost: Intuition

```
p(0).
p(X):- X > 0, Y is X - 1, r(Y), q(Y), p(Y).

q(0).
q(X):- X > 0, Y is X - 1, r(Y), q(Y).

r(0).
r(X):- X > 0, Y is X - 1, r(Y).
```

We want to know how the standard/total cost of p is distributed between the predicates of the program.



# Accumulated-cost: Intuition

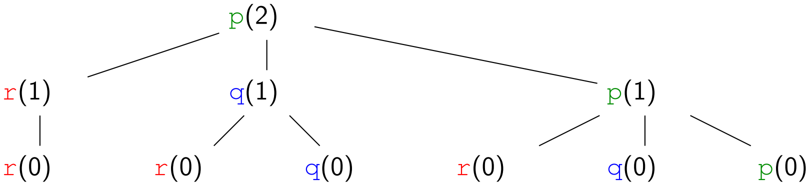
```
p(0).
p(X):- X > 0, Y is X - 1, r(Y), q(Y), p(Y).

q(0).
q(X):- X > 0, Y is X - 1, r(Y), q(Y).

r(0).
r(X):- X > 0, Y is X - 1, r(Y).
```

Set of cost centers:

$$\Diamond = \{\mathtt{p}, \mathtt{q}, \mathtt{r}\}$$

We declare that predicates p, q, and r are cost centers.

Cost centers are user-defined program points (predicates, in our case) to which execution costs are assigned during the execution of a program.



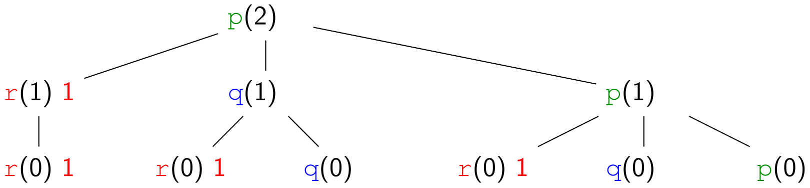
# Accumulated-cost: Intuition

```
p(0).
p(X):- X > 0, Y is X - 1, r(Y), q(Y), p(Y).

q(0).
q(X):- X > 0, Y is X - 1, r(Y), q(Y).

r(0).
r(X):- X > 0, Y is X - 1, r(Y).
```

Set of cost centers:
$\Diamond = \{\mathtt{p}, \mathtt{q}, \mathtt{r}\}$

The cost of a call $\mathtt{p}(2)$ accumulated in cost center $\mathtt{r}$, denoted $\mathtt{C}_{\mathtt{p}}^{\mathtt{r}}(2)$

Is the sum of the resolution steps that are descendant (in the call stack) of $\mathtt{p}(2)$, and whose closest ancestor in the call stack that is a cost center, is $\mathtt{r}$



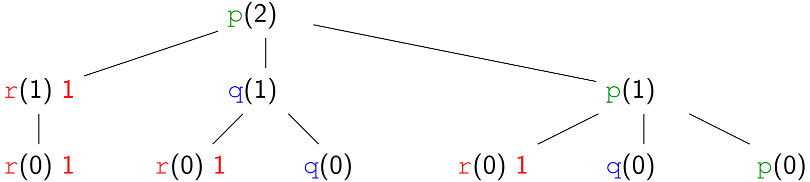
# Accumulated-cost: Intuition

```
p(0).
p(X):- X > 0, Y is X - 1, r(Y), q(Y), p(Y).

q(0).
q(X):- X > 0, Y is X - 1, r(Y), q(Y).

r(0).
r(X):- X > 0, Y is X - 1, r(Y).
```

Set of cost centers:
$\lozenge = \{\mathtt{p}, \mathtt{q}, \mathtt{r}\}$

The cost of a call $\mathtt{p}(2)$ accumulated in cost center $\mathtt{r} \rightarrow \mathtt{C}_{\mathtt{p}}^{\mathtt{r}}(2) = 4$

Is the sum of the resolution steps that are descendant (in the call stack) of $\mathtt{p}(2)$, and whose closest ancestor in the call stack that is a cost center, is $\mathtt{r}$



# Accumulated-cost: Intuition

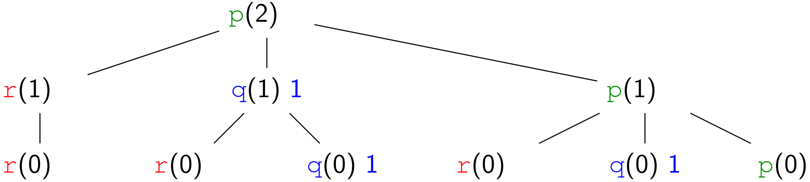
```
p(0).
p(X):- X > 0, Y is X - 1, r(Y), q(Y), p(Y).

q(0).
q(X):- X > 0, Y is X - 1, r(Y), q(Y).

r(0).
r(X):- X > 0, Y is X - 1, r(Y).
```

Set of cost centers:

$$\lozenge = \{\mathtt{p}, \mathtt{q}, \mathtt{r}\}$$

The cost of a call $\mathtt{p}(2)$ accumulated in cost center $\mathtt{q}$, denoted $\mathtt{C}_{\mathtt{p}}^{\mathtt{q}}(2)$

Is the sum of the resolution steps that are descendant (in the call stack) of $\mathtt{p}(2)$, and whose closest ancestor in the call stack that is a cost center, is $\mathtt{q}$



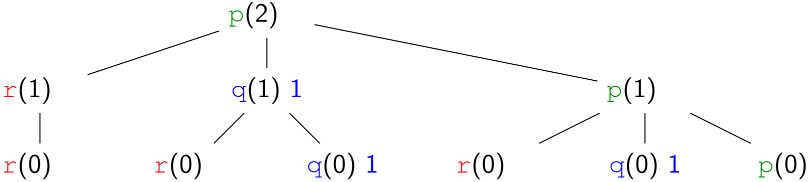
# Accumulated-cost: Intuition

```
p(0).
p(X) :- X > 0, Y is X - 1, r(Y), q(Y), p(Y).

q(0).
q(X) :- X > 0, Y is X - 1, r(Y), q(Y).

r(0).
r(X) :- X > 0, Y is X - 1, r(Y).
```

Set of cost centers:
$\lozenge = \{\mathtt{p}, \mathtt{q}, \mathtt{r}\}$

The cost of a call $\mathtt{p}(2)$ accumulated in cost center $\mathtt{q} \rightarrow \mathtt{C}_{\mathtt{p}}^{\mathtt{q}}(2) = 3$

Is the sum of the resolution steps that are descendant (in the call stack) of $\mathtt{p}(2)$, and whose closest ancestor in the call stack that is a cost center, is $\mathtt{q}$



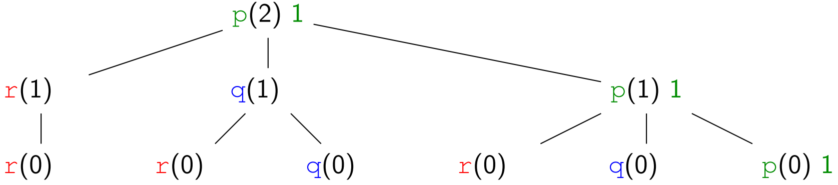
# Accumulated-cost: Intuition

```
p(0).
p(X):- X > 0, Y is X - 1, r(Y), q(Y), p(Y).

q(0).
q(X):- X > 0, Y is X - 1, r(Y), q(Y).

r(0).
r(X):- X > 0, Y is X - 1, r(Y).
```

Set of cost centers:
$$\Diamond = \{\mathtt{p}, \mathtt{q}, \mathtt{r}\}$$

The cost of a call $\mathtt{p}(2)$ accumulated in cost center $\mathtt{p}$, denoted $\mathtt{C}_{\mathtt{p}}^{\mathtt{p}}(2)$

Is the sum of the resolution steps that are descendant (in the call stack) of $\mathtt{p}(2)$, and whose closest ancestor in the call stack that is a cost center, is $\mathtt{p}$



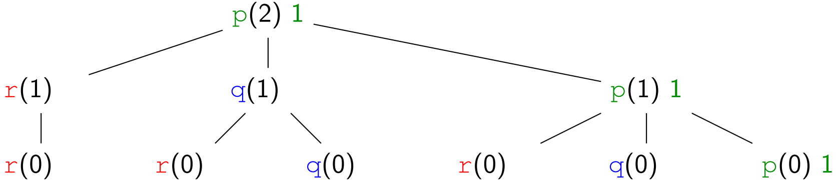
# Accumulated-cost: Intuition

```
p(0).
p(X):- X > 0, Y is X - 1, r(Y), q(Y), p(Y).

q(0).
q(X):- X > 0, Y is X - 1, r(Y), q(Y).

r(0).
r(X):- X > 0, Y is X - 1, r(Y).
```

Set of cost centers:
$\lozenge = \{\mathtt{p}, \mathtt{q}, \mathtt{r}\}$

The cost of a call $\mathtt{p}(2)$ accumulated in cost center $\mathtt{p} \rightarrow \mathtt{C}^{\mathtt{p}}_{\mathtt{p}}(2) = 3$

Is the sum of the resolution steps that are descendant (in the call stack) of $\mathtt{p}(2)$, and whose closest ancestor in the call stack that is a cost center, is $\mathtt{p}$



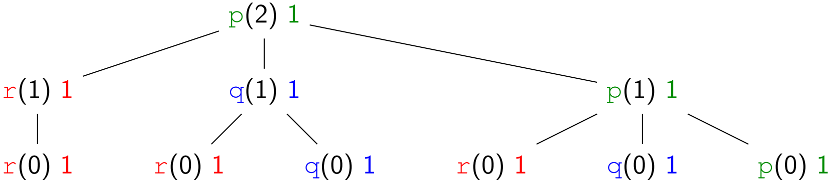
# Accumulated-cost: Intuition

```
p(0).
p(X):- X > 0, Y is X - 1, r(Y), q(Y), p(Y).

q(0).
q(X):- X > 0, Y is X - 1, r(Y), q(Y).

r(0).
r(X):- X > 0, Y is X - 1, r(Y).
```

Set of cost centers:

$$\Diamond = \{\mathtt{p}, \mathtt{q}, \mathtt{r}\}$$

$$\begin{array}{ccccccc} \mathtt{C_p}(2) & = & \mathtt{C_p^p}(2) & + & \mathtt{C_p^q}(2) & + & \mathtt{C_p^r}(2) \\ 10 & = & 3 & + & 3 & + & 4 \end{array}$$

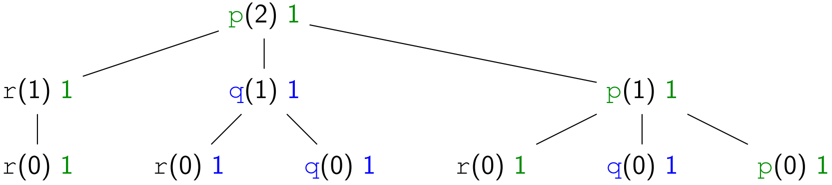


# Accumulated-cost: Intuition

```
p(0).
p(X):- X > 0, Y is X - 1, r(Y), q(Y), p(Y).

q(0).
q(X):- X > 0, Y is X - 1, r(Y), q(Y).

r(0).
r(X):- X > 0, Y is X - 1, r(Y).
```

Set of cost centers:
$\Diamond = \{p, q\}$

We declare that predicates p, q, are cost centers, and r is not.



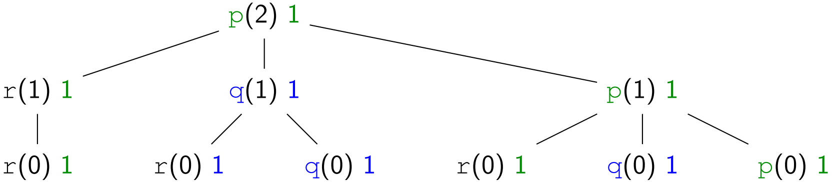
# Accumulated-cost: Intuition

```
p(0).
p(X):- X > 0, Y is X - 1, r(Y), q(Y), p(Y).

q(0).
q(X):- X > 0, Y is X - 1, r(Y), q(Y).

r(0).
r(X):- X > 0, Y is X - 1, r(Y).
```

Set of cost centers:

$$\Diamond = \{\mathtt{p}, \mathtt{q}\}$$

$$\begin{array}{ccccc} \mathtt{C_p}(2) & = & \mathtt{C_p^p}(2) & + & \mathtt{C_p^q}(2) \\ 10 & = & 6 & + & 4 \end{array}$$

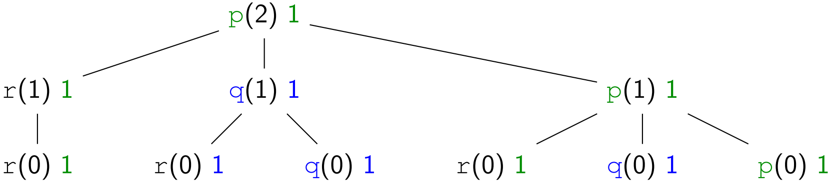


# Accumulated-cost: Definition

## Definition: Accumulated Cost

The cost of a (single) call $\mathtt{p}(n)$ accumulated in cost center $\mathtt{q}$, denoted $\mathtt{C}_{\mathtt{p}}^{\mathtt{q}}(n)$:

- Is the sum of the costs of all the computations that are descendants (in the call stack) of the call $\mathtt{p}(n)$, and are under the scope of any call to $\mathtt{q}$.
- We say that a computation is under the scope of a call to cost center $\mathtt{q}$, if the closest ancestor of such computation in the call stack that is a cost center, is $\mathtt{q}$.
- Expresses how much of the standard cost of the call to $\mathtt{p}$ is attributed to $\mathtt{q}$.



# Cost Relations for Accumulated-costs in Cost Center r

```
p(0).
p(X):- X > 0, Y is X - 1, r(Y), q(Y), p(Y).

q(0).
q(X):- X > 0, Y is X - 1, r(Y), q(Y).

r(0).
r(X):- X > 0, Y is X - 1, r(Y).
```

Set of cost centers:
$\Diamond = \{\mathtt{p}, \mathtt{q}, \mathtt{r}\}$

Cost relations

$n = size(X) = X$ (actual value of $X$)

The cost of p accumulated in r:

$$\mathtt{C}_{\mathtt{p}}^{\mathtt{r}}(0) = 0$$
$$\mathtt{C}_{\mathtt{p}}^{\mathtt{r}}(n) = 0 + \mathtt{C}_{\mathtt{r}}^{\mathtt{r}}(n-1) + \mathtt{C}_{\mathtt{q}}^{\mathtt{r}}(n-1) + \mathtt{C}_{\mathtt{p}}^{\mathtt{r}}(n-1) \quad \text{if } n > 0$$

The cost of q accumulated in r

$$\mathtt{C}_{\mathtt{q}}^{\mathtt{r}}(0) = 0$$
$$\mathtt{C}_{\mathtt{q}}^{\mathtt{r}}(n) = 0 + \ + \mathtt{C}_{\mathtt{q}}^{\mathtt{r}}(n-1) \quad \text{if } n > 0$$

The cost of r accumulated in r

$$\mathtt{C}_{\mathtt{r}}^{\mathtt{r}}(0) = 1$$
$$\mathtt{C}_{\mathtt{r}}^{\mathtt{r}}(n) = 1 + \mathtt{C}_{\mathtt{r}}^{\mathtt{r}}(n-1) \quad \text{if } n > 0$$

# Cost Relations for Accumulated-costs in Cost Center r

```
p(0).
p(X):- X > 0, Y is X - 1, r(Y), q(Y), p(Y).

q(0).
q(X):- X > 0, Y is X - 1, r(Y), q(Y).

r(0).
r(X):- X > 0, Y is X - 1, r(Y).
```

Set of cost centers:
$\lozenge = \{\mathtt{p}, \mathtt{q}, \mathtt{r}\}$

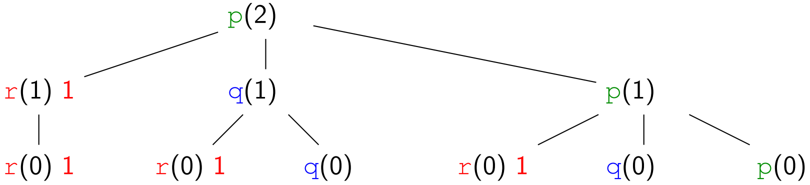
Cost relations $\qquad n = \mathit{size}(X) = X$ (actual value of $X$)

The cost of p accumulated in r:

$$\begin{aligned} \mathtt{C}_{\mathtt{p}}^{\mathtt{r}}(0) &= \quad 0 \\ \mathtt{C}_{\mathtt{p}}^{\mathtt{r}}(n) &= \quad 0 + \mathtt{C}_{\mathtt{r}}^{\mathtt{r}}(n-1) + \mathtt{C}_{\mathtt{q}}^{\mathtt{r}}(n-1) + \mathtt{C}_{\mathtt{p}}^{\mathtt{r}}(n-1) \quad \text{if } n > 0 \end{aligned}$$

---

E.g. ($n = 2$): $\mathtt{C}_{\mathtt{p}}^{\mathtt{r}}(2) = \quad 0 + \mathtt{C}_{\mathtt{r}}^{\mathtt{r}}(1) + \mathtt{C}_{\mathtt{q}}^{\mathtt{r}}(1) + \mathtt{C}_{\mathtt{p}}^{\mathtt{r}}(1) = \quad 0 + 2 + 1 + 1 = 4$



# Cost Relations for Accumulated-costs in Cost Center r

```
p(0).
p(X):- X > 0, Y is X - 1, r(Y), q(Y), p(Y).

q(0).
q(X):- X > 0, Y is X - 1, r(Y), q(Y).

r(0).
r(X):- X > 0, Y is X - 1, r(Y).
```

Set of cost centers:
$\Diamond = \{\mathtt{p}, \mathtt{q}, \mathtt{r}\}$

Cost relations $\qquad n = size(X) = X$ (actual value of $X$)

The cost of p accumulated in r:

$$\mathtt{C}_{\mathtt{p}}^{\mathtt{r}}(0) = 0$$
$$\mathtt{C}_{\mathtt{p}}^{\mathtt{r}}(n) = 0 + \mathtt{C}_{\mathtt{r}}^{\mathtt{r}}(n-1) + \mathtt{C}_{\mathtt{q}}^{\mathtt{r}}(n-1) + \mathtt{C}_{\mathtt{p}}^{\mathtt{r}}(n-1) \quad \text{if } n > 0$$

The cost of q accumulated in r:

$$\mathtt{C}_{\mathtt{q}}^{\mathtt{r}}(0) = 0$$
$$\mathtt{C}_{\mathtt{q}}^{\mathtt{r}}(n) = 0 + \mathtt{C}_{\mathtt{r}}^{\mathtt{r}}(n-1) + \mathtt{C}_{\mathtt{q}}^{\mathtt{r}}(n-1) \quad \text{if } n > 0$$

The cost of r accumulated in r:

$$\mathtt{C}_{\mathtt{r}}^{\mathtt{r}}(0) = 1$$
$$\mathtt{C}_{\mathtt{r}}^{\mathtt{r}}(n) = 1 + \mathtt{C}_{\mathtt{r}}^{\mathtt{r}}(n-1) \quad \text{if } n > 0$$

# Cost Relations for Accumulated-costs in Cost Center $\texttt{r}$

```
p(0).
p(X):- X > 0, Y is X - 1, r(Y), q(Y), p(Y).

q(0).
q(X):- X > 0, Y is X - 1, r(Y), q(Y).

r(0).
r(X):- X > 0, Y is X - 1, r(Y).
```

Set of cost centers:

$$\lozenge = \{\texttt{p}, \texttt{q}, \texttt{r}\}$$

Cost relations $\qquad n = size(X) = X$ (actual value of $X$)

The cost of $\texttt{p}$ accumulated in $\texttt{r}$:

$$\begin{aligned} \mathtt{C}^{\mathtt{r}}_{\mathtt{p}}(0) &= 0 \\ \mathtt{C}^{\mathtt{r}}_{\mathtt{p}}(n) &= 0 + \mathtt{C}^{\mathtt{r}}_{\mathtt{r}}(n-1) + \mathtt{C}^{\mathtt{r}}_{\mathtt{q}}(n-1) + \mathtt{C}^{\mathtt{r}}_{\mathtt{p}}(n-1) \quad \text{if } n > 0 \end{aligned}$$

The cost of $\texttt{q}$ accumulated in $\texttt{r}$:

$$\begin{aligned} \mathtt{C}^{\mathtt{r}}_{\mathtt{q}}(0) &= 0 \\ \mathtt{C}^{\mathtt{r}}_{\mathtt{q}}(n) &= 0 + \mathtt{C}^{\mathtt{r}}_{\mathtt{r}}(n-1) + \mathtt{C}^{\mathtt{r}}_{\mathtt{q}}(n-1) \quad \text{if } n > 0 \end{aligned}$$

The cost of $\texttt{r}$ accumulated in $\texttt{r}$ $\rightarrow$ closed form: $\mathtt{C}^{\mathtt{r}}_{\mathtt{r}}(n) = n + 1$, for $n \geq 0$.

$$\begin{aligned} \mathtt{C}^{\mathtt{r}}_{\mathtt{r}}(0) &= 1 \\ \mathtt{C}^{\mathtt{r}}_{\mathtt{r}}(n) &= 1 + \mathtt{C}^{\mathtt{r}}_{\mathtt{r}}(n-1) \quad \text{if } n > 0 \end{aligned}$$

# Cost Relations for Accumulated-costs in Cost Center $\mathtt{r}$

```
p(0).
p(X) :- X > 0, Y is X - 1, r(Y), q(Y), p(Y).

q(0).
q(X) :- X > 0, Y is X - 1, r(Y), q(Y).

r(0).
r(X) :- X > 0, Y is X - 1, r(Y).
```

Set of cost centers:
$\Diamond = \{\mathtt{p}, \mathtt{q}, \mathtt{r}\}$

Cost relations $\qquad n = size(X) = X$ (actual value of $X$)

The cost of $\mathtt{p}$ accumulated in $\mathtt{r}$:

$$\mathtt{C}^{\mathtt{r}}_{\mathtt{p}}(0) = 0$$
$$\mathtt{C}^{\mathtt{r}}_{\mathtt{p}}(n) = 0 + \mathtt{C}^{\mathtt{r}}_{\mathtt{r}}(n-1) + \mathtt{C}^{\mathtt{r}}_{\mathtt{q}}(n-1) + \mathtt{C}^{\mathtt{r}}_{\mathtt{p}}(n-1) \quad \text{if } n > 0$$

The cost of $\mathtt{q}$ accumulated in $\mathtt{r}$:

$$\mathtt{C}^{\mathtt{r}}_{\mathtt{q}}(0) = 0$$
$$\mathtt{C}^{\mathtt{r}}_{\mathtt{q}}(n) = 0 + n + \mathtt{C}^{\mathtt{r}}_{\mathtt{q}}(n-1) \quad \text{if } n > 0$$

The cost of $\mathtt{r}$ accumulated in $\mathtt{r}$ $\rightarrow$ closed form: $\mathtt{C}^{\mathtt{r}}_{\mathtt{r}}(n) = n + 1$, for $n \geq 0$.

$$\mathtt{C}^{\mathtt{r}}_{\mathtt{r}}(0) = 1$$
$$\mathtt{C}^{\mathtt{r}}_{\mathtt{r}}(n) = 1 + \mathtt{C}^{\mathtt{r}}_{\mathtt{r}}(n-1) \quad \text{if } n > 0$$

# Cost Relations for Accumulated-costs in Cost Center $\mathtt{r}$

```
p(0).
p(X):- X > 0, Y is X - 1, r(Y), q(Y), p(Y).

q(0).
q(X):- X > 0, Y is X - 1, r(Y), q(Y).

r(0).
r(X):- X > 0, Y is X - 1, r(Y).
```

Set of cost centers:
$\Diamond = \{\mathtt{p}, \mathtt{q}, \mathtt{r}\}$

Cost relations $\qquad n = size(X) = X$ (actual value of $X$)

The cost of $\mathtt{p}$ accumulated in $\mathtt{r}$:

$$\begin{aligned} \mathtt{C}_{\mathtt{p}}^{\mathtt{r}}(0) &= \quad 0 \\ \mathtt{C}_{\mathtt{p}}^{\mathtt{r}}(n) &= \quad 0 + \mathtt{C}_{\mathtt{r}}^{\mathtt{r}}(n-1) + \mathtt{C}_{\mathtt{q}}^{\mathtt{r}}(n-1) + \mathtt{C}_{\mathtt{p}}^{\mathtt{r}}(n-1) \quad \text{if } n > 0 \end{aligned}$$

The cost of $\mathtt{q}$ accumulated in $\mathtt{r}$ $\rightarrow$ $\mathtt{C}_{\mathtt{q}}^{\mathtt{r}}(n) = \frac{1}{2}n^2 + \frac{1}{2}n$, for $n \geq 0$

$$\begin{aligned} \mathtt{C}_{\mathtt{q}}^{\mathtt{r}}(0) &= \quad 0 \\ \mathtt{C}_{\mathtt{q}}^{\mathtt{r}}(n) &= \quad 0 + n + \mathtt{C}_{\mathtt{q}}^{\mathtt{r}}(n-1) \quad \text{if } n > 0 \end{aligned}$$

The cost of $\mathtt{r}$ accumulated in $\mathtt{r}$ $\rightarrow$ closed form: $\mathtt{C}_{\mathtt{r}}^{\mathtt{r}}(n) = n + 1$, for $n \geq 0$.

$$\begin{aligned} \mathtt{C}_{\mathtt{r}}^{\mathtt{r}}(0) &= \quad 1 \\ \mathtt{C}_{\mathtt{r}}^{\mathtt{r}}(n) &= \quad 1 + \mathtt{C}_{\mathtt{r}}^{\mathtt{r}}(n-1) \quad \text{if } n > 0 \end{aligned}$$

# Cost Relations for Accumulated-costs in Cost Center $\mathtt{r}$

```
p(0).
p(X) :- X > 0, Y is X - 1, r(Y), q(Y), p(Y).

q(0).
q(X) :- X > 0, Y is X - 1, r(Y), q(Y).

r(0).
r(X) :- X > 0, Y is X - 1, r(Y).
```

Set of cost centers:
$\Diamond = \{\mathtt{p}, \mathtt{q}, \mathtt{r}\}$

Cost relations $\qquad n = size(X) = X$ (actual value of $X$)

The cost of $\mathtt{p}$ accumulated in $\mathtt{r}$:

$$\begin{aligned} \mathtt{C}^{\mathtt{r}}_{\mathtt{p}}(0) &= 0 \\ \mathtt{C}^{\mathtt{r}}_{\mathtt{p}}(n) &= 0 + n + \tfrac{1}{2}(n-1)^2 + \tfrac{1}{2}(n-1) + \mathtt{C}^{\mathtt{r}}_{\mathtt{p}}(n-1) \quad \text{if } n > 0 \end{aligned}$$

The cost of $\mathtt{q}$ accumulated in $\mathtt{r}$ $\rightarrow$ $\mathtt{C}^{\mathtt{r}}_{\mathtt{q}}(n) = \frac{1}{2}n^2 + \frac{1}{2}n$, for $n \geq 0$

$$\begin{aligned} \mathtt{C}^{\mathtt{r}}_{\mathtt{q}}(0) &= 0 \\ \mathtt{C}^{\mathtt{r}}_{\mathtt{q}}(n) &= 0 + n + \mathtt{C}^{\mathtt{r}}_{\mathtt{q}}(n-1) \quad \text{if } n > 0 \end{aligned}$$

The cost of $\mathtt{r}$ accumulated in $\mathtt{r}$ $\rightarrow$ closed form: $\mathtt{C}^{\mathtt{r}}_{\mathtt{r}}(n) = n + 1$, for $n \geq 0$.

$$\begin{aligned} \mathtt{C}^{\mathtt{r}}_{\mathtt{r}}(0) &= 1 \\ \mathtt{C}^{\mathtt{r}}_{\mathtt{r}}(n) &= 1 + \mathtt{C}^{\mathtt{r}}_{\mathtt{r}}(n-1) \quad \text{if } n > 0 \end{aligned}$$

# Cost Relations for Accumulated-costs in Cost Center $\mathtt{r}$

```
p(0).
p(X):- X > 0, Y is X - 1, r(Y), q(Y), p(Y).

q(0).
q(X):- X > 0, Y is X - 1, r(Y), q(Y).

r(0).
r(X):- X > 0, Y is X - 1, r(Y).
```

Set of cost centers:
$\lozenge = \{\mathtt{p}, \mathtt{q}, \mathtt{r}\}$

Cost relations $n = size(X) = X$ (actual value of $X$)

The cost of $\mathtt{p}$ accumulated in $\mathtt{r} \rightarrow \mathtt{C}^{\mathtt{r}}_{\mathtt{p}}(n) = \frac{1}{6}n^3 + \frac{1}{2}n^2 + \frac{1}{3}n$, for $n \geq 0$.

$$\begin{aligned} \mathtt{C}^{\mathtt{r}}_{\mathtt{p}}(0) &= 0 \\ \mathtt{C}^{\mathtt{r}}_{\mathtt{p}}(n) &= 0 + n + \tfrac{1}{2}(n-1)^2 + \tfrac{1}{2}(n-1) + \mathtt{C}^{\mathtt{r}}_{\mathtt{p}}(n-1) \quad \text{if } n > 0 \end{aligned}$$

The cost of $\mathtt{q}$ accumulated in $\mathtt{r} \rightarrow \mathtt{C}^{\mathtt{r}}_{\mathtt{q}}(n) = \frac{1}{2}n^2 + \frac{1}{2}n$, for $n \geq 0$

$$\begin{aligned} \mathtt{C}^{\mathtt{r}}_{\mathtt{q}}(0) &= 0 \\ \mathtt{C}^{\mathtt{r}}_{\mathtt{q}}(n) &= 0 + n + \mathtt{C}^{\mathtt{r}}_{\mathtt{q}}(n-1) \quad \text{if } n > 0 \end{aligned}$$

The cost of $\mathtt{r}$ accumulated in $\mathtt{r} \rightarrow$ closed form: $\mathtt{C}^{\mathtt{r}}_{\mathtt{r}}(n) = n + 1$, for $n \geq 0$.

$$\begin{aligned} \mathtt{C}^{\mathtt{r}}_{\mathtt{r}}(0) &= 1 \\ \mathtt{C}^{\mathtt{r}}_{\mathtt{r}}(n) &= 1 + \mathtt{C}^{\mathtt{r}}_{\mathtt{r}}(n-1) \quad \text{if } n > 0 \end{aligned}$$

# Cost Relations for Accumulated-costs in Cost Center $\mathtt{q}$

```
p(0).
p(X):- X > 0, Y is X - 1, r(Y), q(Y), p(Y).

q(0).
q(X):- X > 0, Y is X - 1, r(Y), q(Y).

r(0).
r(X):- X > 0, Y is X - 1, r(Y).
```

Set of cost centers:
$\lozenge = \{\mathtt{p}, \mathtt{q}, \mathtt{r}\}$

Cost relations $\qquad n = size(X) = X$ (actual value of $X$)

The cost of $\mathtt{p}$ accumulated in $\mathtt{q}$:

$$\mathtt{C}^{\mathtt{q}}_{\mathtt{p}}(0) = 0$$
$$\mathtt{C}^{\mathtt{q}}_{\mathtt{p}}(n) = 0 + \mathtt{C}^{\mathtt{q}}_{\mathtt{r}}(n-1) + \mathtt{C}^{\mathtt{q}}_{\mathtt{q}}(n-1) + \mathtt{C}^{\mathtt{q}}_{\mathtt{p}}(n-1) \quad \text{if } n > 0$$

The cost of $\mathtt{q}$ accumulated in $\mathtt{q}$:

$$\mathtt{C}^{\mathtt{q}}_{\mathtt{q}}(0) = 1$$
$$\mathtt{C}^{\mathtt{q}}_{\mathtt{q}}(n) = 1 + \mathtt{C}^{\mathtt{q}}_{\mathtt{r}}(n-1) + \mathtt{C}^{\mathtt{q}}_{\mathtt{q}}(n-1) \quad \text{if } n > 0$$

The cost of $\mathtt{r}$ accumulated in $\mathtt{q}$:

$$\mathtt{C}^{\mathtt{q}}_{\mathtt{r}}(0) = 0$$
$$\mathtt{C}^{\mathtt{q}}_{\mathtt{r}}(n) = 0 + \mathtt{C}^{\mathtt{q}}_{\mathtt{r}}(n-1) \quad \text{if } n > 0$$

# Cost Relations for Accumulated-costs in Cost Center q

```
p(0).
p(X) :- X > 0, Y is X - 1, r(Y), q(Y), p(Y).

q(0).
q(X) :- X > 0, Y is X - 1, r(Y), q(Y).

r(0).
r(X) :- X > 0, Y is X - 1, r(Y).
```

Set of cost centers:
$\Diamond = \{p, q, r\}$

Cost relations $\qquad n = size(X) = X$ (actual value of $X$)

The cost of p accumulated in q:

$$
\begin{aligned}
C_p^q(0) &= \quad 0 \\
C_p^q(n) &= \quad 0 + C_r^q(n-1) + C_q^q(n-1) + C_p^q(n-1) \quad \text{if } n > 0
\end{aligned}
$$

The cost of q accumulated in q:

$$
\begin{aligned}
C_q^q(0) &= \quad 1 \\
C_q^q(n) &= \quad 1 + C_r^q(n-1) + C_q^q(n-1) \quad \text{if } n > 0
\end{aligned}
$$

The cost of r accumulated in q $\rightarrow C_r^q(n) = 0$, for $n \geq 0$.

$\forall r, q \in \Diamond$, if $r \not\rightarrow_\alpha^\star q$ then $C_r^q(\bar{x}) = 0$ (Lemma 3)

# Cost Relations for Accumulated-costs in Cost Center q

```
p(0).
p(X):- X > 0, Y is X - 1, r(Y), q(Y), p(Y).

q(0).
q(X):- X > 0, Y is X - 1, r(Y), q(Y).

r(0).
r(X):- X > 0, Y is X - 1, r(Y).
```

Set of cost centers:
$\lozenge = \{\mathtt{p}, \mathtt{q}, \mathtt{r}\}$

Cost relations $\qquad n = size(X) = X$ (actual value of $X$)

The cost of p accumulated in q:

$$\mathtt{C}_{\mathtt{p}}^{\mathtt{q}}(0) = 0$$
$$\mathtt{C}_{\mathtt{p}}^{\mathtt{q}}(n) = 0 + \mathtt{C}_{\mathtt{r}}^{\mathtt{q}}(n-1) + \mathtt{C}_{\mathtt{q}}^{\mathtt{q}}(n-1) + \mathtt{C}_{\mathtt{p}}^{\mathtt{q}}(n-1) \quad \text{if } n > 0$$

The cost of q accumulated in q $\rightarrow \mathtt{C}_{\mathtt{q}}^{\mathtt{q}}(n) = n + 1$, for $n \geq 0$.

$$\mathtt{C}_{\mathtt{q}}^{\mathtt{q}}(0) = 1$$
$$\mathtt{C}_{\mathtt{q}}^{\mathtt{q}}(n) = 1 + \mathtt{C}_{\mathtt{r}}^{\mathtt{q}}(n-1) + \mathtt{C}_{\mathtt{q}}^{\mathtt{q}}(n-1) \quad \text{if } n > 0$$

The cost of r accumulated in q $\rightarrow \mathtt{C}_{\mathtt{r}}^{\mathtt{q}}(n) = 0$, for $n \geq 0$.

$\forall \mathtt{r}, \mathtt{q} \in \lozenge$, if $\mathtt{r} \not\rightsquigarrow_{\alpha}^{\star} \mathtt{q}$ then $\mathtt{C}_{\mathtt{r}}^{\mathtt{q}}(\bar{\mathtt{x}}) = 0$ (Lemma 3)

# Cost Relations for Accumulated-costs in Cost Center $\mathtt{q}$

```
p(0).
p(X) :- X > 0, Y is X - 1, r(Y), q(Y), p(Y).

q(0).
q(X) :- X > 0, Y is X - 1, r(Y), q(Y).

r(0).
r(X) :- X > 0, Y is X - 1, r(Y).
```

Set of cost centers:
$\lozenge = \{\mathtt{p}, \mathtt{q}, \mathtt{r}\}$

Cost relations $\quad n = size(X) = X$ (actual value of $X$)

The cost of $\mathtt{p}$ accumulated in $\mathtt{q} \rightarrow \mathtt{C}_{\mathtt{p}}^{\mathtt{q}}(n) = \frac{1}{2}\ n^2 + \frac{1}{2}\ n$.

$$\begin{aligned} \mathtt{C}_{\mathtt{p}}^{\mathtt{q}}(0) &= \quad 0 \\ \mathtt{C}_{\mathtt{p}}^{\mathtt{q}}(n) &= \quad 0 + \mathtt{C}_{\mathtt{r}}^{\mathtt{q}}(n-1) + \mathtt{C}_{\mathtt{q}}^{\mathtt{q}}(n-1) + \mathtt{C}_{\mathtt{p}}^{\mathtt{q}}(n-1) \quad \text{if } n > 0 \end{aligned}$$

The cost of $\mathtt{q}$ accumulated in $\mathtt{q} \rightarrow \mathtt{C}_{\mathtt{q}}^{\mathtt{q}}(n) = n + 1$, for $n \geq 0$.

$$\begin{aligned} \mathtt{C}_{\mathtt{q}}^{\mathtt{q}}(0) &= \quad 1 \\ \mathtt{C}_{\mathtt{q}}^{\mathtt{q}}(n) &= \quad 1 + \mathtt{C}_{\mathtt{r}}^{\mathtt{q}}(n-1) + \mathtt{C}_{\mathtt{q}}^{\mathtt{q}}(n-1) \quad \text{if } n > 0 \end{aligned}$$

The cost of $\mathtt{r}$ accumulated in $\mathtt{q} \rightarrow \mathtt{C}_{\mathtt{r}}^{\mathtt{q}}(n) = 0$, for $n \geq 0$.

$\forall \mathtt{r}, \mathtt{q} \in \lozenge$, if $\mathtt{r} \not\rightsquigarrow_{\alpha}^{\star} \mathtt{q}$ then $\mathtt{C}_{\mathtt{r}}^{\mathtt{q}}(\bar{\mathtt{x}}) = 0$ (Lemma 3)

# Cost Relations for Accumulated-costs in Cost Center p

```
p(0).
p(X) :- X > 0, Y is X - 1, r(Y), q(Y), p(Y).

q(0).
q(X) :- X > 0, Y is X - 1, r(Y), q(Y).

r(0).
r(X) :- X > 0, Y is X - 1, r(Y).
```

Set of cost centers:
$\lozenge = \{\mathtt{p}, \mathtt{q}, \mathtt{r}\}$

Cost relations $\qquad n = size(X) = X$ (actual value of $X$)

The cost of p accumulated in p:

$$\mathtt{C}_{\mathtt{p}}^{\mathtt{p}}(0) = 1$$
$$\mathtt{C}_{\mathtt{p}}^{\mathtt{p}}(n) = 1 + \mathtt{C}_{\mathtt{r}}^{\mathtt{p}}(n-1) + \mathtt{C}_{\mathtt{q}}^{\mathtt{p}}(n-1) + \mathtt{C}_{\mathtt{p}}^{\mathtt{p}}(n-1) \quad \text{if } n > 0$$

$\mathtt{C}_{\mathtt{q}}^{\mathtt{p}}(n) = 0$ (by Lemma 3, since $\mathtt{q} \not\rightarrowtail_{\alpha}^{\star} \mathtt{p}$).

$\mathtt{C}_{\mathtt{r}}^{\mathtt{p}}(n) = 0$ (by Lemma 3, since $\mathtt{r} \not\rightarrowtail_{\alpha}^{\star} \mathtt{p}$).

# Cost Relations for Accumulated-costs in Cost Center $\mathtt{p}$

```
p(0).
p(X):- X > 0, Y is X - 1, r(Y), q(Y), p(Y).

q(0).
q(X):- X > 0, Y is X - 1, r(Y), q(Y).

r(0).
r(X):- X > 0, Y is X - 1, r(Y).
```

Set of cost centers:
$\Diamond = \{\mathtt{p}, \mathtt{q}, \mathtt{r}\}$

Cost relations

$n = size(X) = X$ (actual value of $X$)

The cost of $\mathtt{p}$ accumulated in $\mathtt{p}$:

$$\begin{aligned} \mathtt{C}_{\mathtt{p}}^{\mathtt{p}}(0) &= 1 \\ \mathtt{C}_{\mathtt{p}}^{\mathtt{p}}(n) &= 1 + \mathtt{C}_{\mathtt{p}}^{\mathtt{p}}(n-1) \quad \text{if } n > 0 \end{aligned}$$

$\mathtt{C}_{\mathtt{q}}^{\mathtt{p}}(n) = 0$ (by Lemma 3, since $\mathtt{q} \not\rightsquigarrow_{\alpha}^{\star} \mathtt{p}$).

$\mathtt{C}_{\mathtt{r}}^{\mathtt{p}}(n) = 0$ (by Lemma 3, since $\mathtt{r} \not\rightsquigarrow_{\alpha}^{\star} \mathtt{p}$).

# Cost Relations for Accumulated-costs in Cost Center $\mathtt{p}$

```
p(0).
p(X):- X > 0, Y is X - 1, r(Y), q(Y), p(Y).

q(0).
q(X):- X > 0, Y is X - 1, r(Y), q(Y).

r(0).
r(X):- X > 0, Y is X - 1, r(Y).
```

Set of cost centers:
$\Diamond = \{\mathtt{p}, \mathtt{q}, \mathtt{r}\}$

Cost relations $\quad n = size(X) = X$ (actual value of $X$)

The cost of $\mathtt{p}$ accumulated in $\mathtt{p} \rightarrow \mathtt{C}^{\mathtt{p}}_{\mathtt{p}}(n) = n + 1$, for $n \geq 0$.

$$\begin{aligned} \mathtt{C}^{\mathtt{p}}_{\mathtt{p}}(0) &= \quad 1 \\ \mathtt{C}^{\mathtt{p}}_{\mathtt{p}}(n) &= \quad 1 + \mathtt{C}^{\mathtt{p}}_{\mathtt{p}}(n-1) \quad \text{if } n > 0 \end{aligned}$$

$\mathtt{C}^{\mathtt{p}}_{\mathtt{q}}(n) = 0$ (by Lemma 3, since $\mathtt{q} \not\rightsquigarrow^{\star}_{\alpha} \mathtt{p}$).

$\mathtt{C}^{\mathtt{p}}_{\mathtt{r}}(n) = 0$ (by Lemma 3, since $\mathtt{r} \not\rightsquigarrow^{\star}_{\alpha} \mathtt{p}$).

# Need for Tracking the "Environment:" Example

```
p(0).
p(X) :- X > 0, Y is X - 1, r(Y), q(Y), p(Y).

q(0).
q(X) :- X > 0, Y is X - 1, r(Y), q(Y).

r(0).
r(X) :- X > 0, Y is X - 1, r(Y).
```

Set of cost centers:

$$\Diamond = \{\mathtt{p}, \mathtt{q}\}$$

The cost of p accumulated in q:

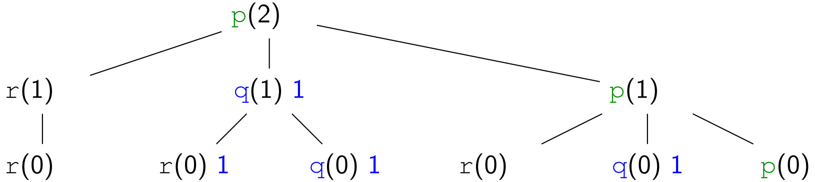
$$\mathtt{C}_{\mathtt{p}}^{\mathtt{q}}(0) = 0$$

$$\mathtt{C}_{\mathtt{p}}^{\mathtt{q}}(n) = 0 + \mathtt{C}_{\mathtt{r},0}^{\mathtt{q}}(n-1) + \mathtt{C}_{\mathtt{q}}^{\mathtt{q}}(n-1) + \mathtt{C}_{\mathtt{p}}^{\mathtt{q}}(n-1) \quad \text{if } n > 0$$

The cost of q accumulated in q:

$$\mathtt{C}_{\mathtt{q}}^{\mathtt{q}}(0) = 1$$

$$\mathtt{C}_{\mathtt{q}}^{\mathtt{q}}(n) = 1 + \mathtt{C}_{\mathtt{r},1}^{\mathtt{q}}(n-1) + \mathtt{C}_{\mathtt{q}}^{\mathtt{q}}(n-1) \quad \text{if } n > 0$$




We have two versions for the cost of r accumulated in q:

# Need for Tracking the "Environment:" Example

```
p(0).
p(X):- X > 0, Y is X - 1, r(Y), q(Y), p(Y).

q(0).
q(X):- X > 0, Y is X - 1, r(Y), q(Y).

r(0).
r(X):- X > 0, Y is X - 1, r(Y).
```

Set of cost centers:

$$\Diamond = \{\mathtt{p}, \mathtt{q}\}$$

The cost of p accumulated in q:

$$\begin{aligned} \mathtt{C}^{\mathtt{q}}_{\mathtt{p}}(0) &= 0 \\ \mathtt{C}^{\mathtt{q}}_{\mathtt{p}}(n) &= 0 + \mathtt{C}^{\mathtt{q}}_{\mathtt{r},0}(n-1) + \mathtt{C}^{\mathtt{q}}_{\mathtt{q}}(n-1) + \mathtt{C}^{\mathtt{q}}_{\mathtt{p}}(n-1) \quad \text{if } n > 0 \end{aligned}$$

The cost of q accumulated in q:

$$\begin{aligned} \mathtt{C}^{\mathtt{q}}_{\mathtt{q}}(0) &= 1 \\ \mathtt{C}^{\mathtt{q}}_{\mathtt{q}}(n) &= 1 + \mathtt{C}^{\mathtt{q}}_{\mathtt{r},1}(n-1) + \mathtt{C}^{\mathtt{q}}_{\mathtt{q}}(n-1) \quad \text{if } n > 0 \end{aligned}$$

---

We have two versions for the cost of r accumulated in q:

Under the scope of q

$$\boxed{\begin{aligned} \mathtt{C}^{\mathtt{q}}_{\mathtt{r},1}(0) &= 1 \\ \mathtt{C}^{\mathtt{q}}_{\mathtt{r},1}(n) &= 1 + \mathtt{C}^{\mathtt{q}}_{\mathtt{r},1}(n-1) \quad \text{if } n > 0 \end{aligned}}$$

NOT under the scope of q

$$\boxed{\begin{aligned} \mathtt{C}^{\mathtt{q}}_{\mathtt{r},0}(0) &= 0 \\ \mathtt{C}^{\mathtt{q}}_{\mathtt{r},0}(n) &= 0 + \mathtt{C}^{\mathtt{q}}_{\mathtt{r},0}(n-1) \quad \text{if } n > 0 \end{aligned}}$$

# Our Extended Cost Relations for Accumulated-cost

The standard cost of a clause

$$C \equiv \mathrm{p}(\bar{x}) :- \mathrm{q}_1(\bar{x}_1), \ldots, \mathrm{q}_n(\bar{x}_n)$$

for a (single) call to $\mathrm{p}$:

$$\mathrm{C}_{\mathrm{p}}(\bar{\mathrm{x}}) \quad = \varphi(\mathrm{p}(\bar{\mathrm{x}})) + \sum_{i=1}^{lim(C,\bar{\mathrm{x}})} sols_i \times \mathrm{C}_{\mathrm{q}_i}(\bar{\mathrm{x}}_i)$$

E.g., for resolutions steps $\rightarrow \varphi(\mathrm{p}(\bar{\mathrm{x}})) = 1$.

- $lim(C, \bar{\mathrm{x}}) \stackrel{\text{def}}{=}$ index of the last body literal that is called in the execution of $C$.
- $sols_i \stackrel{\text{def}}{=}$ product of the number of solutions produced by the ancestor literals of $\mathrm{q}_i(\bar{x}_i)$ in the clause body:

$$sols_i = \prod_{j=1}^{i-1} s_{pred}(\mathrm{q}_j(\bar{x}_j))$$

$$s_{pred}(\mathrm{q}_j(\bar{x}_j)) \stackrel{\text{def}}{=} \text{number of solutions produced by } \mathrm{q}_j(\bar{x}_j)$$

The cost of a body literal $\mathrm{q}_i(\bar{x}_i)$ is obtained from the costs of all clauses applicable to it that are executed, by using an aggregation operator $\bigodot$

# Our Extended Cost Relations for Accumulated-cost

The accumulated cost of a clause

$$C \equiv \mathrm{p}(\bar{x}) :- \mathrm{q}_1(\bar{x}_1), \ldots, \mathrm{q}_n(\bar{x}_n)$$

for a (single) call to p:

$$\mathrm{C}^{\mathrm{c}}_{\mathrm{p},e}(\bar{\mathrm{x}}) \quad = B_{\varphi}(\mathrm{p}, \mathrm{c}, e) \times \varphi(\mathrm{p}(\bar{\mathrm{x}})) + \sum_{i=1}^{lim(C,\bar{\mathrm{x}})} sols_i \times \mathrm{C}^{\mathrm{c}}_{\mathrm{q}_i,e'}(\bar{\mathrm{x}}_i) \times B(\mathrm{p}, \mathrm{c}, e, \mathrm{q}_i)$$

E.g., for resolutions steps $\rightarrow \varphi(\mathrm{p}(\bar{\mathrm{x}})) = 1$.

- $lim(C, \bar{\mathrm{x}}) \stackrel{\text{def}}{=}$ index of the last body literal that is called in the execution of $C$.
- $sols_i \stackrel{\text{def}}{=}$ product of the number of solutions produced by the ancestor literals of $\mathrm{q}_i(\bar{x}_i)$ in the clause body:

$$sols_i = \prod_{j=1}^{i-1} s_{pred}(\mathrm{q}_j(\bar{x}_j))$$

$$s_{pred}(\mathrm{q}_j(\bar{x}_j)) \stackrel{\text{def}}{=} \text{number of solutions produced by } \mathrm{q}_j(\bar{x}_j)$$

The cost of a body literal $\mathrm{q}_i(\bar{x}_i)$ is obtained from the costs of all clauses applicable to it that are executed, by using an aggregation operator $\bigodot$

# Our Extended Cost Relations for Accumulated-cost

The accumulated cost of a clause

$$C \equiv \mathrm{p}(\bar{x}) :- \mathrm{q}_1(\bar{x}_1), \ldots, \mathrm{q}_n(\bar{x}_n)$$

for a (single) call to $\mathrm{p}$:

$$\mathrm{C}^{\mathrm{c}}_{\mathrm{p},e}(\bar{\mathrm{x}}) = B_\varphi(\mathrm{p},\mathrm{c},e) \times \varphi(\mathrm{p}(\bar{\mathrm{x}})) + \sum_{i=1}^{lim(C,\bar{\mathrm{x}})} sols_i \times \mathrm{C}^{\mathrm{c}}_{\mathrm{q}_i,e'}(\bar{\mathrm{x}}_i) \times B(\mathrm{p},\mathrm{c},e,\mathrm{q}_i)$$

- The environment $e$ is a Boolean value ($1 \equiv \mathtt{true}$ and $0 \equiv \mathtt{false}$):
  $$e = \begin{cases} 1 & \text{if the call to } \mathrm{p} \text{ is under the scope of cost center } \mathrm{c} \\ 0 & \textit{otherwise} \end{cases}$$
- Boolean functions:
  $B_\varphi(\mathrm{p},\mathrm{c},e)$ is 1 iff "the computation" is under the scope of $\mathrm{c}$.
  $$B_\varphi(\mathrm{p},\mathrm{c},e) \stackrel{\text{def}}{=} (\mathrm{p} = \mathrm{c} \vee (\mathrm{p} \notin \Diamond \wedge e))$$
  $B(p,\mathrm{c},e,q)$ is 1 iff the body literal is under the scope of $\mathrm{c}$, or it may call $\mathrm{c}$.
  $$B(p,\mathrm{c},e,q) \stackrel{\text{def}}{=} B_\varphi(p,\mathrm{c},e) \vee (q \rightsquigarrow^{\star}_{\alpha} c)$$
- $e' = \mathcal{E}(\mathrm{p},\mathrm{c},e,\mathrm{q}_i(\bar{\mathrm{x}}_i))$, and $\mathcal{E}$ is the *environment change* function:
  $\mathcal{E}(\mathrm{p},\mathrm{c},e,\_) \stackrel{\text{def}}{=} (\mathrm{p} = \mathrm{c} \vee (\mathrm{p} \notin \Diamond \wedge e))$

# Our Extended Cost Relations for Accumulated-cost

The accumulated cost of a clause

$$C \equiv \mathrm{p}(\bar{x}) :- \mathrm{q}_1(\bar{x}_1), \ldots, \mathrm{q}_n(\bar{x}_n)$$

for a (single) call to p:

$$\mathrm{C}^{\mathrm{c}}_{\mathrm{p},e}(\bar{\mathrm{x}}) \quad = B_{\varphi}(\mathrm{p}, \mathrm{c}, e) \times \varphi(\mathrm{p}(\bar{\mathrm{x}})) + \sum_{i=1}^{lim(C,\bar{\mathrm{x}})} sols_i \times \mathrm{C}^{\mathrm{c}}_{\mathrm{q}_i,e'}(\bar{\mathrm{x}}_i) \times B(\mathrm{p}, \mathrm{c}, e, \mathrm{q}_i)$$

If a trust assertion gives the cost of p as a function $\Psi(\mathrm{p})(\bar{\mathrm{x}})$, then:
$\mathrm{C}_{\mathrm{p}}(\bar{\mathrm{x}}) = \Psi(\mathrm{p})(\bar{\mathrm{x}})$

# Our Extended Cost Relations for Accumulated-cost

The accumulated cost of a clause

$$C \equiv \mathrm{p}(\bar{x}) :- \mathrm{q}_1(\bar{x}_1), \ldots, \mathrm{q}_n(\bar{x}_n)$$

for a (single) call to p:

$$\mathrm{C}^{\mathrm{c}}_{\mathrm{p},e}(\bar{\mathrm{x}}) \quad = B_{\varphi}(\mathrm{p}, \mathrm{c}, e) \times \varphi(\mathrm{p}(\bar{\mathrm{x}})) + \sum_{i=1}^{lim(C,\bar{\mathrm{x}})} sols_i \times \mathrm{C}^{\mathrm{c}}_{\mathrm{q}_i,e'}(\bar{\mathrm{x}}_i) \times B(\mathrm{p}, \mathrm{c}, e, \mathrm{q}_i)$$

If a trust assertion gives the cost of p as a function $\Psi(\mathrm{p})(\bar{\mathrm{x}})$, then:

$$\mathrm{C}^{\mathrm{c}}_{\mathrm{p},e}(\bar{\mathrm{x}}) = \Psi(\mathrm{p})(\bar{\mathrm{x}}) \times B_{\varphi}(\mathrm{p}, \mathrm{c}, e)$$

# Genericity of our Cost Relations Framework

The cost a clause for a (single) call to $\mathtt{p}$:

$$\mathtt{C}^{\mathtt{c}}_{\mathtt{p},e}(\bar{\mathtt{x}}) \quad = B_{\varphi}(\mathtt{p},\mathtt{c},e) \times \varphi(\mathtt{p}(\bar{\mathtt{x}})) + \sum_{i=1}^{lim(C,\bar{\mathtt{x}})} sols_i \times B(\mathtt{p},\mathtt{c},e,\mathtt{q}_i) \times \mathtt{C}^{\mathtt{c}}_{\mathtt{q}_i,e'}(\bar{\mathtt{x}}_i)$$

- A broad notion of *environment* $e$. E.g., for energy consumption:
  - state of the hardware or the whole system,
  - the last instruction executed (for modeling the *switching cost*), temperature, voltage, cache state, and pipeline state.
- Suitable definitions of the Boolean functions $B_{\varphi}(\mathtt{p},\mathtt{c},e)$ and $B(p,\mathtt{c},e,q)$ to control which terms of the cost relations should be considered.
- $\mathtt{C}^{\mathtt{c}}_{\mathtt{p},e}(\bar{\mathtt{x}}) \stackrel{\text{def}}{=}$ part of $\mathtt{C}_{\mathtt{p}}(\bar{\mathtt{x}})$, performed in an *environment* $e$, that is attributed to *cost center* $\mathtt{c}$ of the program.

# Some Properties of the Accumulated-cost

### Definition of the *calls* relation, $\rightsquigarrow_\alpha$

- $\mathtt{p} \rightsquigarrow_\alpha \mathtt{q}$, iff a literal with predicate symbol $\mathtt{q}$ appears in the body of a clause defining $\mathtt{p}$.
- $\rightsquigarrow_\alpha^\star$ is the reflexive transitive closure of $\rightsquigarrow_\alpha$.
- It is an abstraction (over-approximation) of the concrete "calls" relation, $\rightsquigarrow$.

# Some Properties of the Accumulated-cost

- $\forall \mathrm{p}, \mathrm{c} \in \lozenge$, $\forall e \in \{0, 1\}$, it holds that:
  - $\mathcal{E}(\mathrm{p}, \mathrm{c}, e, \_) \stackrel{\text{def}}{=} (\mathrm{p} = \mathrm{c})$

    (recall that $\mathcal{E}(\mathrm{p}, \mathrm{c}, e, \_) \stackrel{\text{def}}{=} (\mathrm{p} = \mathrm{c} \vee (\mathrm{p} \notin \lozenge \wedge e)))$
  - $B_\varphi(\mathrm{p}, \mathrm{c}, e) \stackrel{\text{def}}{=} (\mathrm{p} = \mathrm{c})$.

    (recall that $B_\varphi(\mathrm{p}, \mathrm{c}, e) \stackrel{\text{def}}{=} (\mathrm{p} = \mathrm{c} \vee (\mathrm{p} \notin \lozenge \wedge e)))$
  - $B(p, \mathrm{c}, e, q) \stackrel{\text{def}}{=} (\mathrm{p} = \mathrm{c}) \vee (q \rightsquigarrow_\alpha^\star c)$.

    (recall that $B(p, \mathrm{c}, e, q) \stackrel{\text{def}}{=} B_\varphi(p, \mathrm{c}, e) \vee (q \rightsquigarrow_\alpha^\star c))$
- This implies that $\forall \mathrm{p}, \mathrm{c} \in \lozenge$ it holds that $\mathrm{C}^{\mathrm{c}}_{\mathrm{p},0}(\bar{\mathrm{x}}) = \mathrm{C}^{\mathrm{c}}_{\mathrm{p},1}(\bar{\mathrm{x}})$.
- Thus, if $\mathrm{p} \in \lozenge$ we omit the environment $e$ and write $\mathrm{C}^{\mathrm{c}}_{\mathrm{p}}(\bar{\mathrm{x}})$.
- (Lemma 3) $\forall \mathrm{p}, \mathrm{c} \in \lozenge$, if $\mathrm{p} \not\rightsquigarrow_\alpha^\star \mathrm{c}$ then $\mathrm{C}^{\mathrm{c}}_{\mathrm{p}}(\bar{\mathrm{x}}) = 0$.
- (Lemma 4) $\forall \mathrm{p} \notin \lozenge, \forall \mathrm{c} \in \lozenge$, if $\mathrm{p} \not\rightsquigarrow_\alpha^\star \mathrm{c}$ then $\mathrm{C}^{\mathrm{c}}_{\mathrm{p},0}(\bar{\mathrm{x}}) = 0$.

# Usefulness of the Accumulated Cost

Consider the following program, where predicates $p$, $q$ and $r$ are cost centers.

```
p(X, Y, Z):- X > 0, q(X, Y, Z1), Z is Z1 * 2.

q(0, _, 0).
q(X, Y, Z) :- r(Y, Y1), X1 is X - 1, q(X1, Y, Z1), Z is Z1 + Y1.

r(0, 0).
r(X, Y):- X1 is X - 1, r(X1, Y1), Y is Y1 + X.
```

Standard Cost

- $\boxed{\mathcal{C}_p(x,y) = y * x + 2 * x + 2}$
- $\boxed{\mathcal{C}_q(x,y) = y * x + 2 * x + 1}$
- $\boxed{\mathcal{C}_r(x,y) = x + 1}$

Accumulated Cost

- $\boxed{\mathcal{C}_p^p(x,y) = 1}$
- $\boxed{\mathcal{C}_p^q(x,y) = x}$
- $\boxed{\mathcal{C}_p^r(x,y) = x * y}$

- $\mathcal{C}_r(x,y) = x + 1 \rightarrow$ *r is not costly by itself.*
- $\mathcal{C}_p^r(x,y) = x * y \rightarrow$ *r is responsible for the non-linear complexity of p.*

# Usefulness of the Accumulated Cost

Consider the following program, where predicates $p$, $q$ and $r$ are cost centers.

```
p(X, Y, Z):- X > 0, q(X, Y, Z1), Z is Z1 * 2.

q(0, _, 0).
q(X, Y, Z) :- r(Y, Y1), X1 is X - 1, q(X1, Y, Z1), Z is Z1 + Y1.

r(0, 0).
r(X, Y):- X1 is X - 1, r(X1, Y1), Y is Y1 + X.
```

Standard Cost

- $\boxed{\mathcal{C}_p(x,y) = y * x + 2 * x + 2}$
- $\boxed{\mathcal{C}_q(x,y) = y * x + 2 * x + 1}$
- $\boxed{\mathcal{C}_r(x,y) = x + 1}$

Accumulated Cost

- $\boxed{\mathcal{C}_p^p(x,y) = 1}$
- $\boxed{\mathcal{C}_p^q(x,y) = x}$
- $\boxed{\mathcal{C}_p^r(x,y) = x * y}$

- $\mathcal{C}_r(x,y) = x + 1 \rightarrow$ *r is not costly by itself.*
- $\mathcal{C}_p^r(x,y) = x * y \rightarrow$ *r is responsible for the non-linear complexity of p.*

# Usefulness of the Accumulated Cost

Obvious improvement:
move the call $r(Y, Y1)$ outside the (recursive) definition of the predicate $q$.

```
p(X, Y, Z):- X >= 0, r(Y, Y1) , q(X, Y1, Z1), Z is Z1 * 2.

q(0, _, 0).
q(X, Y, Z) :- X1 is X - 1, q(X1, Y, Z1), Z is Z1 + Y.

r(0, 0).
r(X, Y):- X1 is X - 1, r(X1, Y1), Y is Y1 + X.
```

Standard Cost

- $\boxed{\mathcal{C}_p(x, y) = x + y + 3}$
- $\boxed{\mathcal{C}_q(x, y) = x + 1}$
- $\boxed{\mathcal{C}_r(x, y) = x + 1}$

Accumulated Cost

- $\boxed{\mathcal{C}_p^p(x, y) = 1}$
- $\boxed{\mathcal{C}_p^q(x, y) = x}$
- $\boxed{\mathcal{C}_p^r(x, y) = y}$

# Implementation

- Implementation within CiaoPP, directly as an abstract *domain*.
- The information abstracted at each program point includes the state + non-functional props.
- Cost relations are built *incrementally, in the abstract domain*.
- Features inherited for free:
  - Multivariance: separate equations built for each procedure version.
  - Equations are not built for unreachable parts of the program.
  - Easy combination with other abstract domains (reduced product based), in particular, the new sized types and a novel cardinality analysis.
  - Assertion verification.
  - Etc.

# Accumulated Cost: Experimental Results

| Cost-Centers & Input Sizes | Accumulated Cost UB | Static vs. Dyn | Standard Cost UB | #Calls |
|---|---|---|---|---|
| $variance(n)^*$ | $1$ | 0% | $2n^2$ | $1$ |
| $sq\_diff(m_1, m_2)$ | $n-1$ | 0% | $2m_1 m_2 - 2m_2$ | $n-1$ |
| $mean(u)$ | $2n^2 - n$ | 0% | $2u+1$ | $n$ |
| $is\_prime(n)^*$ | $1$ | 0% | $(n-1)! + n + 3$ | $1$ |
| $fact(m)$ | $n$ | 0% | $m$ | $n$ |
| $mult(u)$ | $(n-1)! + 2$ | 0% | $u+1$ | $(n-1)! + 2$ |
| $app1(n_1, n_2, n_3)^*$ | $n_1$ | 0% | $\mathcal{O}(n_1 n_2 n_3)^\dagger$ | $1$ |
| $app2(m_1, m_2)$ | $n_1 n_2$ | 0% | $m_1 m_2$ | $n_1$ |
| $app3(u)$ | $2n_1 n_2 n_3$ | 0% | $u$ | $n_1 n_2 + n_1$ |
| $dyade(n_1, n_2)^*$ | $n_1$ | 0% | $n_1(n_2+1)$ | $1$ |
| $mult(m)$ | $n_1 n_2$ | 0% | $m$ | $n_1$ |
| $minsort(n)^*$ | $n+1$ | 0% | $\frac{(n+1)^2}{2} + \frac{n+1}{2}$ | $1$ |
| $findmin(m)$ | $\frac{(n+1)^2}{2} + \frac{n-1}{2}$ | 7% | $m$ | $n+1$ |
| $hanoi(n)^*$ | $2^n - 1$ | 0% | $2^{n+1} - 2$ | $1$ |
| $move(m)$ | $2^n - 1$ | 0% | $1$ | $2^n - 1$ |
| $coupled(n)^*$ | $1$ | 0% | $n+2$ | $1$ |
| $p(m)$ | $\frac{n}{2} + \frac{(-1)^n}{4} + \frac{3}{4}$ | 1.2% | $m+1$ | $\frac{n}{2} - \frac{(-1)^n}{4} + \frac{1}{4}$ |
| $q(u)$ | $\frac{n}{2} - \frac{(-1)^n}{4} + \frac{1}{4}$ | 0% | $u+1$ | $\frac{n}{2} + \frac{(-1)^n}{4} - \frac{1}{4}$ |
| $search(n)^*$ | $1$ | 0% | $2n+2$ | $1$ |
| $member(m)$ | $2n+1$ | 0% | $2m+1$ | $2n+1$ |
| $sublist(n_1, n_2)^*$ | $n_2 + 3$ | 5% | $n_1 n_2 + 3n_2 + 2$ | $2$ |
| $append(m)$ | $n_1 n_2 + 2n_2 - 1$ | 40% | $2m-1$ | $n_1 n_2 + 2n_2 - 1$ |

# Experimental Results: Times (milliseconds)

| Cost-Center | Accumulated Cost UB | | Standard Cost UB | Acc Cost/ Std Cost |
|---|---|---|---|---|
| | Cost Relations | Transformation (FLOPS'16) | | |
| *variance** *sq_diff* *mean* | 3283 (-45%) | 6038 | 3066 | 1.07 |
| *isprime** *fact* *mult* | 1245 (-42%) | 2172 | 1231 | 1.01 |
| *app1** *app2* *app3* | 4150 (-34%) | 6328 | 3757 | 1.11 |
| *minsort** *findmin* | 3400 (-29%) | 4845 | 3300 | 1.03 |
| *dyade** *mult* | 3097 (-24%) | 4117 | 2853 | 1.08 |
| *hanoi** *move* | 1605 (-19%) | 1996 | 1376 | 1.16 |
| *coupled** *f* *g* | 2407 (-14%) | 3112 | 1877 | 1.28 |
| *search** *member* | 1079 | N/A | 1071 | 1.00 |
| *sublist** *append* | 3674 | N/A | 3610 | 1.01 |
| **Average** | **2652 (-33%)** | **4125** | **2542** | **1.05** |

# Conclusions

- Novel, general, and flexible framework for setting up cost relations which can be instantiated for performing a wide range of resource usage analyses, including both accumulated cost and standard cost.
- Advantages over our previous work (specific to accumulated cost) based on a program transformation:
  - More general.
  - Can deal with non-deterministic/multiple-solution predicates.
  - More efficient.
  - Implementation based on a direct application of abstract interpretation and integration into CiaoPP → many useful features are inherited for free.
  - Also inherits the capability of analyzing for several resources at the same time.
- Experiments → accurate inference of accumulated cost.
- Static profiling is a *more valuable aid for resource-aware software development* than standard resource usage analysis.

  → identify parts that should be optimized first.
- Our approach can be easily applied to other paradigms:

  → including imperative programs, functional programs, CHR, etc.,

  → by compilation to Horn Clauses (as in our previous work with Java or XC).

# Demo!

Please see examples in the CiaoPP playground.

(`http://play.ciao-lang.org`)

# The Team

- Working specifically in CiaoPP resource analysis:


Pedro López-García


Manuel Hermenegildo


Maximiliano Klemen


Umer Liqat

- CiaoPP overall:


José-Francisco Morales


Nataliia Stulova


Isabel García-Contreras

- Previous main contributors to CiaoPP resource analysis:

Saumya Debray　Nai-wei Lin　Jorge Navas
Alejandro Serrano　Mario Méndez-Lojo　Edison Mera

---

Work currently at: IMDEA Software Institute, T.U. Madrid (UPM).

And previously at: U. T. Austin, MCC, U. of Arizona, U. of New Mexico.

Playground at: `http://play.ciao-lang.org`

Thank you!

# Timeline of our Work

**1990** Method for static inference of upper-bound functions on execution cost and data structure sizes [PLDI'90] (building on Wegbreit):

- Techniques for setting up, solving/approximating recurrence relations.
- For Horn-clause programs → used widely as IR for other languages.
- Motivation: task granularity control in automatic parallelization.
- Experimental results (resulting in improved parallel speedups).
- Implementation (leading to CASLOG) but I/O arguments, types, measures, etc. had to be provided by the user.

**1993-1994** **First fully automatic system**, including all auxiliary analyses: **GraCos** (Granularity Control System), implemented within **CiaoPP** [SAS'94, PASCO'94].

- Reducing data size computation overhead. [ICLP'95]
- Further improvements. [JSC'96]
- Precision improved w/determinacy, partial eval. ... [LOPSTR'04, NGC'10]

**1997** **Lower bounds** cost analysis; **divide-and-conquer**. [ILPS'97]

- Lower bounds required developing non-failure (no-exceptions) analysis, guard coverage, ... [ICLP'97, FLOPS'04]
- Also in [ILPS'97]: proposed *non-deterministic recurrence relations*, special for divide-and-conquer programs: looking at sets of computation trees and balancing/bounding node cost (e.g., quadratic bound for qsort).

1990 Method for static inference of upper-bound functions on execution cost and data structure sizes [PLDI'90] (building on Wegbreit):

- Techniques for setting up, solving/approximating recurrence relations.
- For Horn-clause programs → used widely as IR for other languages.
- Motivation: task granularity control in automatic parallelization.
- Experimental results (resulting in improved parallel speedups).
- Implementation (leading to CASLOG) but I/O arguments, types, measures, etc. had to be provided by the user.

1993-1994 **First fully automatic system**, including all auxiliary analyses: **GraCos** (Granularity Control System), implemented within **CiaoPP** [SAS'94, PASCO'94].

- Reducing data size computation overhead. [ICLP'95]
- Further improvements. [JSC'96]
- Precision improved w/determinacy, partial eval. ... [LOPSTR'04, NGC'10]

1997 **Lower bounds** cost analysis; **divide-and-conquer**. [ILPS'97]

- Lower bounds required developing non-failure (no-exceptions) analysis, guard coverage, ... [ICLP'97, FLOPS'04]
- Also in [ILPS'97]: proposed *non-deterministic recurrence relations*, special for divide-and-conquer programs: looking at sets of computation trees and balancing/bounding node cost (e.g., quadratic bound for qsort).

**1997-2003** **Verification:** assert. lang, comp./run-time simple function comparisons (orders). [AADEBUG'97, LOPSTR'99, ILPS-WS'97, LNCS'00, SAS'03]

**2004** Abstraction carrying code for resources. [PPDP'05, LPAR'04]

**2006** Probabilistic Cost Analysis. [CLEI'06]

**2007** **User-definable** resources. [ICLP'07]

**2007** **Multi-language** support (Java bytecode, C#, FP, CLP) via **Horn clause-based IR**. [LOPSTR'07]

- Combined with user-definable resources: no need to develop specific analyzers for specific languages!
- Instrumental analyses: sharing/nullity/class [VMCAI'08, PASTE'08] dependence [LCPC'08] shape [CC'08, SAS'02].

**2008** Application to **execution time** (using bytecode-level models, obtained by regression). [PPDP'08]

**2008** Application to **energy consumption** of Java bytecode. [NASA FM'08]

**2007-2009** User-definable resources for Java bytecode. [Bytecode'09]

1997–2003 **Verification:** assert. lang, comp./run-time simple function comparisons (orders). [AADEBUG'97, LOPSTR'99, ILPS-WS'97, LNCS'00, SAS'03]

2004 Abstraction carrying code for resources. [PPDP'05, LPAR'04]

2006 Probabilistic Cost Analysis. [CLEI'06]

2007 $\boxed{\textbf{User-definable}\text{ resources.}}$ [ICLP'07]

2007 **Multi-language** support (Java bytecode, C#, FP, CLP) via **Horn clause-based IR**. [LOPSTR'07]

- Combined with user-definable resources:
  no need to develop specific analyzers for specific languages!
- Instrumental analyses: sharing/nullity/class [VMCAI'08, PASTE'08] dependence [LCPC'08] shape [CC'08, SAS'02].

2008 Application to **execution time** (using bytecode-level models, obtained by regression). [PPDP'08]

2008 Application to **energy consumption** of Java bytecode. [NASA FM'08]

2007–2009 User-definable resources for Java bytecode. [Bytecode'09]

1997- **Verification:** assert. lang, comp./run-time [AADEBUG'97, LOPSTR'99, ILPS-WS'97, LNCS'00, SAS'03]
2003 simple function comparisons (orders).

2004 Abstraction carrying code for resources. [PPDP'05, LPAR'04]

2006 Probabilistic Cost Analysis. [CLEI'06]

2007 **User-definable** resources. [ICLP'07]

2007 **Multi-language** support (Java bytecode, C#, FP, CLP) via **Horn clause-based IR**. [LOPSTR'07]

- Combined with user-definable resources: no need to develop specific analyzers for specific languages!
- Instrumental analyses: sharing/nullity/class [VMCAI'08, PASTE'08] dependence [LCPC'08] shape [CC'08, SAS'02].

2008 Application to **execution time** (using bytecode-level models, obtained by regression). [PPDP'08]

2008 Application to **energy consumption** of Java bytecode. [NASA FM'08]

2007- User-definable resources for Java bytecode. [Bytecode'09]
2009

**2010-2015** Resource verification: **interval-based, improved function comparison**. [ICLP'10, FOPARA'12, HIP3ES'15]

**2012-2014** **Cost analysis as multivariant abstract interpretation**. [TPLP'14]

→ Multivariant, integrated with assertion checking, modular, incremental.

Domain: interval (piece-wise) functions. [ICLP'10, FOPARA'12]

**2013** Using **sized shapes (sized types)**. [ICLP'13]

**2013-2016** **Analysis and verification of Energy**:

- At the ISA level [LOPSTR'13]
- Comparing LLMV and ISA levels [FOPARA'15]
- At the block level [HIP3ES'16]

**2016** **Static profiling** / accumulated cost. [FLOPS'16, TPLP'16]

2010-2015 Resource verification: **interval-based, improved function comparison.** [ICLP'10, FOPARA'12, HIP3ES'15]

2012-2014 **Cost analysis as multivariant abstract interpretation.** [TPLP'14]

→ Multivariant, integrated with assertion checking, modular, incremental.

Domain: interval (piece-wise) functions. [ICLP'10, FOPARA'12]

2013 Using **sized shapes (sized types).** [ICLP'13]

2013-2016 **Analysis and verification of Energy**:

- At the ISA level [LOPSTR'13]
- Comparing LLMV and ISA levels [FOPARA'15]
- At the block level [HIP3ES'16]

2016 **Static profiling** / accumulated cost. [FLOPS'16, TPLP'16]

# Selected Bibliography on CiaoPP (and Additional Slides)

# CiaoPP References – Analysis and Verification of Energy


[HIP3ES'16] U. Liqat, Z. Banković, P. Lopez-Garcia, and M. V. Hermenegildo.
Inferring Energy Bounds Statically by Evolutionary Analysis of Basic Blocks.
In *(HIP3ES'16)*, 2016. arXiv:1601.02800.

[FOPARA'15] U. Liqat, K. Georgiou, S. Kerrison, P. Lopez-Garcia, M. V. Hermenegildo, J. P. Gallagher, and K. Eder.
Inferring Parametric Energy Consumption Functions at Different Software Levels: ISA vs. LLVM IR.
In M. Van Eekelen and U. Dal Lago, editors, *Foundational and Practical Aspects of Resource Analysis: 4th Intl. Workshop, FOPARA 2015, London, UK, April 11, 2015. Revised Selected Papers*, volume 9964 LNCS, pages 81–100. Springer, 2016.

[HIP3ES'15] P. Lopez-Garcia, R. Haemmerlé, M. Klemen, U. Liqat, and M. V. Hermenegildo
Towards Energy Consumption Verification via Static Analysis.
In *HIPEAC Workshop on High Performance Energy Efficient Embedded Systems (HIP3ES 2015)*, Amsterdam.

[LOPSTR'13] U. Liqat, S. Kerrison, A. Serrano, K. Georgiou, P. López-Garcia, N. Grech, M.V. Hermenegildo, and K. Eder.
Energy Consumption Analysis of Programs based on XMOS ISA-Level Models.
In *Pre-proceedings of the 23rd Intl. Symposium on Logic-Based Program Synthesis and Transformation (LOPSTR'13)*, LNCS, Springer, September 2013.

[NASA FM'08] J. Navas, M. Méndez-Lojo, and M. Hermenegildo.
Safe Upper-bounds Inference of Energy Consumption for Java Bytecode Applications.
In *6th NASA Langley Formal Methods Workshop (LFM 08)*, April 2008. Extended Abstract.

# CiaoPP References – Intermediate Repr. / Multi-Lingual Support


[LOPSTR'07] M. Méndez-Lojo, J. Navas, and M. Hermenegildo.
A Flexible (C)LP-Based Approach to the Analysis of Object-Oriented Programs.
In *17th Intl. Symposium on Logic-based Program Synthesis and Transformation (LOPSTR 2007)*, number 4915 in LNCS, pages 154–168. Springer-Verlag, August 2007.


# CiaoPP References – Analysis and Verification of Resources in General


[TPLP'16] P. Lopez-Garcia, M. Klemen, U. Liqat, and M.V. Hermenegildo.
A General Framework for Static Profiling of Parametric Resource Usage.
*Theory and Practice of Logic Programming, 32nd Int'l. Conference on Logic Programming (ICLP'16) Special Issue*, 16(5-6):849–865, October 2016.

[FLOPS'16] R. Haemmerlé, P. Lopez-Garcia, U. Liqat, M. Klemen, J. P. Gallagher, and M. V. Hermenegildo.
A Transformational Approach to Parametric Accumulated-cost Static Profiling.
In *FLOPS'16*, volume 9613 of *LNCS*, pages 163–180. Springer, 2016.

[TPLP'14] A. Serrano, P. López-Garcia, and M. Hermenegildo.
Resource Usage Analysis of Logic Programs via Abstract Interpretation Using Sized Types.
In *Theory and Practice of Logic Programming, 30th Int'l. Conference on Logic Programming (ICLP'14) Special Issue*, Vol. 14, Num. 4-5, pages 739-754, Cambridge U. Press, 2014.

[ICLP'13] A. Serrano, P. López-Garcia, F. Bueno, and M. Hermenegildo.
Sized Type Analysis of Logic Programs (Technical Communication).
In *Theory and Practice of Logic Programming, 29th Int'l. Conference on Logic Programming (ICLP'13) Special Issue, On-line Supplement,* pages 1–14, Cambridge U. Press, August 2013.

[FOPARA'12] P. Lopez-Garcia, L. Darmawan, F. Bueno, and M. Hermenegildo
Interval-Based Resource Usage Verification: Formalization and Prototype
In *Foundational and Practical Aspects of Resource Analysis. Second International Workshop FOPARA 2011, Revised Selected Papers*. Lecture Notes in Computer Science, 2012, 7177, 54–71, Springer.

[ICLP'10] P. López-García, L. Darmawan, and F. Bueno.
A Framework for Verification and Debugging of Resource Usage Properties.
In *Technical Communications of the 26th ICLP. Leibniz Int'l. Proc. in Informatics (LIPIcs)*, Vol. 7, pages 104–113, Dagstuhl, Germany, July 2010.

[Bytecode'09] J. Navas, M. Méndez-Lojo, and M. Hermenegildo.
User-Definable Resource Usage Bounds Analysis for Java Bytecode.
In *Workshop on Bytecode Semantics, Verification, Analysis and Transformation (BYTECODE'09)*, volume 253 of *ENTCS*, pages 6–86. Elsevier, March 2009.

[PPDP'08] E. Mera, P. López-García, M. Carro, and M. Hermenegildo.
Towards Execution Time Estimation in Abstract Machine-Based Languages.
In *10th Int'l. ACM SIGPLAN Symposium on Principles and Practice of Declarative Programming (PPDP'08)*, pages 174–184. ACM Press, July 2008.

[ICLP'07] J. Navas, E. Mera, P. López-García, and M. Hermenegildo.
User-Definable Resource Bounds Analysis for Logic Programs.
In *23rd International Conference on Logic Programming (ICLP'07)*, LNCS Vol. 4670. Springer, 2007. **ICLP 2017 10-year Test of Time Award.**

[CLEI'06] H. Soza, M. Carro, and P. López-García.
Probabilistic Cost Analysis of Logic Programs: A First Case Study.
In *XXXII Latin-American Conference on Informatics* (CLEI 2006), August 2006.

[ILPS'97] S. K. Debray, P. López-García, M. Hermenegildo, and N.-W. Lin.
Lower Bound Cost Estimation for Logic Programs.
In *1997 Intl. Logic Programming Symposium*, pages 291–305. MIT Press, Cambridge, MA, October 1997.

[SAS'94] S.K. Debray, P. López-García, M. Hermenegildo, and N.-W. Lin.
Estimating the Computational Cost of Logic Programs.
In *Static Analysis Symposium, SAS'94*, number 864 in LNCS, pages 255–265, Sep 1994. Springer.

[ICLP'95] P. López-García and M. Hermenegildo.
Efficient Term Size Computation for Granularity Control.
In *International Conference on Logic Programming*, pages 647–661, Cambridge, MA, June 1995. MIT Press, Cambridge, MA.

[JSC'96] P. López-García, M. Hermenegildo, and S. K. Debray.
A Methodology for Granularity Based Control of Parallelism in Logic Programs.
*Journal of Symbolic Computation, Special Issue on Parallel Symbolic Computation*, 21(4–6):715–734, 1996.

[PASCO'94] P. López-García, M. Hermenegildo, and S.K. Debray.
Towards Granularity Based Control of Parallelism in Logic Programs.
In Hoon Hong, editor, *Proc. of First Intl. Symposium on Parallel Symbolic Computation, PASCO'94*, pages 133–144. World Scientific, September 1994.

[PLDI'90] S. K. Debray, N.-W. Lin, and M. Hermenegildo.
Task Granularity Analysis in Logic Programs.
In *Proc. of the 1990 ACM Conf. on Programming Language Design and Implementation (PLDI)*, pages 174–188. ACM Press, June 1990.

# CiaoPP References – Assertion Language


[PPDP'14] N. Stulova, J. F. Morales, M. V. Hermenegildo.
Assertion-based Debugging of Higher-Order (C)LP Programs.
In *16th Int'l. ACM SIGPLAN Symposium on Principles and Practice of Declarative Programming (PPDP'14)*, ACM Press, September 2014.

[ICLP'09] E. Mera, P. López-García, and M. Hermenegildo.
Integrating Software Testing and Run-Time Checking in an Assertion Verification Framework.
In *25th Intl. Conference on Logic Programming (ICLP'09)*, number 5649 in LNCS, pages 281–295. Springer-Verlag, July 2009.

[LNCS'00] G. Puebla, F. Bueno, and M. Hermenegildo.
An Assertion Language for Constraint Logic Programs.
In P. Deransart, M. Hermenegildo, and J. Maluszynski, editors, *Analysis and Visualization Tools for Constraint Programming*, number 1870 in LNCS, pages 23–61. Springer-Verlag, September 2000.

[ILPS-WS'97] G. Puebla, F. Bueno, and M. Hermenegildo.
An Assertion Language for Debugging of Constraint Logic Programs.
In *Proceedings of the ILPS'97 Workshop on Tools and Environments for (Constraint) Logic Programming*, October 1997. Available from `ftp://cliplab.org/pub/papers/assert_lang_tr_discipldeliv.ps.gz` as tech. report CLIP2/97.1.

# CiaoPP References – Overall Debugging and Verification Model


[PPDP'16] N. Stulova, J. F. Morales, and M. V. Hermenegildo.
Reducing the Overhead of Assertion Run-time Checks via static analysis.
In *18th Int'l. ACM SIGPLAN Symposium on Principles and Practice of Declarative Programming (PPDP'16)*, pages 90–103. ACM Press, September 2016.

[TPLP'15] N. Stulova, J. F. Morales, and M. V. Hermenegildo.
Practical Run-time Checking via Unobtrusive Property Caching.
*Theory and Practice of Logic Programming, 31st Int'l. Conference on Logic Programming (ICLP'15) Special Issue*, 15(04-05):726–741, September 2015.

[ICLP'09] E. Mera, P. López-García, and M. Hermenegildo.
Integrating Software Testing and Run-Time Checking in an Assertion Verification Framework.
In *25th Intl. Conference on Logic Programming (ICLP'09)*, number 5649 in LNCS, pages 281–295. Springer-Verlag, July 2009.

[SCP'05] M. Hermenegildo, G. Puebla, F. Bueno, and P. López-García.
Integrated Program Debugging, Verification, and Optimization Using Abstract Interpretation
*Science of Computer Programming*, 58(1–2), 2005.

[SAS'03] M. Hermenegildo, G. Puebla, F. Bueno, and P. López-García.
Program Development Using Abstract Interpretation (and The Ciao System Preprocessor).
In *10th International Static Analysis Symposium (SAS'03)*, number 2694 in LNCS, pages 127–152. Springer-Verlag, June 2003.

[PBH00a] G. Puebla, F. Bueno, and M. Hermenegildo.
A Generic Preprocessor for Program Validation and Debugging.
In P. Deransart, M. Hermenegildo, and J. Maluszynski, editors, *Analysis and Visualization Tools for Constraint Programming*, number 1870 in LNCS, pages 63–107. Springer-Verlag, Sept. 2000.

[LOPSTR'99] G. Puebla, F. Bueno, and M. Hermenegildo.
Combined Static and Dynamic Assertion-Based Debugging of Constraint Logic Programs.
In *Logic-based Program Synthesis and Transformation (LOPSTR'99)*, number 1817 in LNCS, pages 273–292. Springer-Verlag, March 2000.

[LNCS'99] M. Hermenegildo, G. Puebla, and F. Bueno.
Using Global Analysis, Partial Specifications, and an Extensible Assertion Language for Program Validation and Debugging.
In K. R. Apt, V. Marek, M. Truszczynski, and D. S. Warren, editors, *The Logic Programming Paradigm: a 25–Year Perspective*, pages 161–192. Springer-Verlag, July 1999.

[AADEBUG'97] F. Bueno, P. Deransart, W. Drabent, G. Ferrand, M. Hermenegildo, J. Maluszynski, and G. Puebla.
On the Role of Semantic Approximations in Validation and Diagnosis of Constraint Logic Programs.
In *Proc. of the 3rd. Int'l Workshop on Automated Debugging–AADEBUG'97*, pages 155–170, Linköping, Sweden, May 1997. U. of Linköping Press.

# CiaoPP References – Abstraction-Carrying Code

[ICLP'06] E. Albert, P. Arenas, G. Puebla, and M. Hermenegildo.
Reduced Certificates for Abstraction-Carrying Code.
In *22nd Intl. Conference on Logic Programming (ICLP 2006)*, number 4079 in LNCS, pages 163–178. Springer-Verlag, August 2006.

[PPDP'05] M. Hermenegildo, E. Albert, P. López-García, and G. Puebla.
Abstraction Carrying Code and Resource-Awareness.
In *PPDP*. ACM Press, 2005.

[LPAR'04] E. Albert, G. Puebla, and M. Hermenegildo.
Abstraction-Carrying Code.
In *Proc. of LPAR'04*, volume 3452 of *LNAI*. Springer, 2005.

# CiaoPP References – Fixpoint-based Analyzer (Abstract Interpreter)


[FTfJP'07] J. Navas, M. Méndez-Lojo, and M. Hermenegildo.
An Efficient, Context and Path Sensitive Analysis Framework for Java Programs.
In *9th Workshop on Formal Techniques for Java-like Programs FTfJP 2007*, July 2007.

[TOPLAS'00] M. Hermenegildo, G. Puebla, K. Marriott, and P. Stuckey.
Incremental Analysis of Constraint Logic Programs.
*ACM Transactions on Programming Languages and Systems*, 22(2):187–223, March 2000.

[TOPLAS'99] F. Bueno, M. García de la Banda, and M. Hermenegildo.
Effectiveness of Abstract Interpretation in Automatic Parallelization: A Case Study in Logic Programming.
*ACM Transactions on Programming Languages and Systems*, 21(2):189–238, March 1999.

[SAS'96] G. Puebla and M. Hermenegildo.
Optimized Algorithms for the Incremental Analysis of Logic Programs.
In *Intl. Static Analysis Symposium (SAS 1996)*, number 1145 in LNCS, pages 270–284. Springer-Verlag, September 1996.

[POPL'94] K. Marriott, M. García de la Banda, and M. Hermenegildo.
Analyzing Logic Programs with Dynamic Scheduling.
In *20th. Annual ACM Conf. on Principles of Programming Languages*, pages 240–254. ACM, January 1994.

[JLP'92] K. Muthukumar and M. Hermenegildo.
Compile-time Derivation of Variable Dependency Using Abstract Interpretation.
*Journal of Logic Programming*, 13(2/3):315–347, July 1992.

[ICLP'88] R. Warren, M. Hermenegildo, and S. K. Debray
On the Practicality of Global Flow Analysis of Logic Programs
In *5th Intl. Conf. and Symp. on Logic Programming*, pp. 684–699, MIT Press, August 1988.

# CiaoPP References – Modular Analysis, Analysis of Concurrency


[PEPM'08] P. Pietrzak, J. Correas, G. Puebla, and M. Hermenegildo.
A Practical Type Analysis for Verification of Modular Prolog Programs.
In *PEPM'08*, pages 61–70. ACM Press, January 2008.

[LPAR'06] P. Pietrzak, J. Correas, G. Puebla, and M. Hermenegildo.
Context-Sensitive Multivariant Assertion Checking in Modular Programs.
In *LPAR'06*, number 4246 in LNCS, pages 392–406. Springer-Verlag, November 2006.

[LOPSTR'01] F. Bueno, M. García de la Banda, M. Hermenegildo, K. Marriott, G. Puebla, and P. Stuckey.
A Model for Inter-module Analysis and Optimizing Compilation.
In *Logic-based Program Synthesis and Transformation*, number 2042 in LNCS, pages 86–102. Springer-Verlag, March 2001.

[ENTCS'00] G. Puebla and M. Hermenegildo.
Some Issues in Analysis and Specialization of Modular Ciao-Prolog Programs.
In *Special Issue on Optimization and Implementation of Declarative Programming Languages*, volume 30 of *Electronic Notes in Theoretical Computer Science*. Elsevier - North Holland, March 2000.

[POPL'94] K. Marriott, M. García de la Banda, and M. Hermenegildo.
Analyzing Logic Programs with Dynamic Scheduling.
In *20th. Annual ACM Conf. on Principles of Programming Languages (POPL)*, pages 240–254. ACM, January 1994.

[ESOP'96] F. Bueno, D. Cabeza, M. Hermenegildo, and G. Puebla.
Global Analysis of Standard Prolog Programs.
In *European Symposium on Programming*, number 1058 in LNCS, pages 108–124, Sweden, April 1996. Springer-Verlag.

# CiaoPP References – Domains: Sharing/Aliasing


[ISMM'09] M. Marron, D. Kapur, and M. Hermenegildo.
Identification of Logically Related Heap Regions.
In *ISMM'09: Proceedings of the 8th international symposium on Memory management*, New York, NY, USA, June 2009. ACM Press.

[LCPC'08] M. Méndez-Lojo, O. Lhoták, and M. Hermenegildo.
Efficient Set Sharing using ZBDDs.
In *21st Int'l. WS on Languages and Compilers for Parallel Computing (LCPC'08)*, LNCS. Springer-Verlag, August 2008.

[LCPC'08] M. Marron, D. Kapur, D. Stefanovic, and M. Hermenegildo.
Identification of Heap-Carried Data Dependence Via Explicit Store Heap Models.
In *21st Int'l. WS on Languages and Compilers for Parallel Computing (LCPC'08)*, LNCS. Springer-Verlag, August 2008.

[PASTE'08] M. Marron, M. Méndez-Lojo, M. Hermenegildo, D. Stefanovic, and D. Kapur.
Sharing Analysis of Arrays, Collections, and Recursive Structures.
In *ACM WS on Program Analysis for SW Tools and Engineering (PASTE'08)*. ACM, November 2008.

[VMCAI'08] M. Méndez-Lojo and M. Hermenegildo.
Precise Set Sharing Analysis for Java-style Programs.
In *9th Intl. Conference on Verification, Model Checking and Abstract Interpretation (VMCAI'08)*, number 4905 in LNCS, pages 172–187. Springer-Verlag, January 2008.

[PADL'06] J. Navas, F. Bueno, and M. Hermenegildo.
Efficient top-down set-sharing analysis using cliques.
In *Eight Intl. Symposium on Practical Aspects of Declarative Languages*, number 2819 in LNCS, pages 183–198. Springer-Verlag, January 2006.

[ICLP'91] K. Muthukumar and M. Hermenegildo.
Combined Determination of Sharing and Freeness of Program Variables Through Abstract Interpretation.
In *Intl. Conference on Logic Programming (ICLP 1991)*, pages 49–63. MIT Press, June 1991.

[NACLP'89] K. Muthukumar and M. Hermenegildo.
Determination of Variable Dependence Information at Compile-Time Through Abstract Interpretation.
In *1989 N. American Conf. on Logic Programming*, pages 166–189. MIT Press, October 1989.

## CiaoPP References – Domains: Shape/Type Analysis

[ICLP'13] A. Serrano, P. López-Garcia, F. Bueno, and M. Hermenegildo.
Sized Type Analysis of Logic Programs (Technical Communication).
In *Theory and Practice of Logic Programming, 29th Int'l. Conference on Logic Programming (ICLP'13) Special Issue, On-line Supplement,* pages 1–14, Cambridge U. Press, August 2013.

[CC'08] M. Marron, M. Hermenegildo, D. Kapur, and D. Stefanovic.
Efficient context-sensitive shape analysis with graph-based heap models.
In Laurie Hendren, editor, *Intl. Conference on Compiler Construction (CC 2008)*, Lecture Notes in Computer Science. Springer, April 2008.

[PASTE'07] M. Marron, D. Stefanovic, M. Hermenegildo, and D. Kapur.
Heap Analysis in the Presence of Collection Libraries.
In *ACM WS on Program Analysis for SW Tools and Engineering (PASTE'07)*. ACM, June 2007.

[SAS'02] C. Vaucheret and F. Bueno.
More Precise yet Efficient Type Inference for Logic Programs.
In *Intl. Static Analysis Symposium*, volume 2477 of *Lecture Notes in Computer Science*, pages 102–116. Springer-Verlag, September 2002.

# CiaoPP References – Domains: Non-failure, Determinacy


[NGC'10] P. López-García, F. Bueno, and M. Hermenegildo.
Automatic Inference of Determinacy and Mutual Exclusion for Logic Programs Using Mode and Type Information.
*New Generation Computing*, 28(2):117–206, 2010.

[LOPSTR'04] P. López-García, F. Bueno, and M. Hermenegildo.
Determinacy Analysis for Logic Programs Using Mode and Type Information.
In *Proceedings of the 14th Intl. Symposium on Logic-based Program Synthesis and Transformation (LOPSTR'04)*, number 3573 in LNCS, pages 19–35. Springer-Verlag, August 2005.

[FLOPS'04] F. Bueno, P. López-García, and M. Hermenegildo.
Multivariant Non-Failure Analysis via Standard Abstract Interpretation.
In *7th Intl. Symposium on Functional and Logic Programming (FLOPS 2004)*, number 2998 in LNCS, pages 100–116, Heidelberg, Germany, April 2004. Springer-Verlag.

[ICLP'97] S.K. Debray, P. López-García, and M. Hermenegildo.
Non-Failure Analysis for Logic Programs.
In *1997 Intl. Conference on Logic Programming*, pages 48–62, Cambridge, MA, June 1997. MIT Press, Cambridge, MA.

# Additional slides

# "Classical" Cost Analysis (cost relations): Example

```
nrev([],[]).
nrev([H|L],R) :- nrev(L,R1), app(R1,[H],R).

app([],L,L).
app([H|L],L1,[H|R]) :- app(L,L1,R).
```

- Cost relations for resolution steps $\quad n = \mathit{length}(X)$ (length of list $X$)
- Cost of *nrev*:
$$\begin{aligned} \mathrm{C}_{nrev}(0) &= 1 \\ \mathrm{C}_{nrev}(n) &= 1 + \mathrm{C}_{nrev}(n-1) + \mathrm{C}_{app}(n-1, 1) \quad \text{if } n > 0 \end{aligned}$$
- Cost of *app*:
$$\begin{aligned} \mathrm{C}_{app}(0, m) &= 1 \\ \mathrm{C}_{app}(n, m) &= 1 + \mathrm{C}_{app}(n-1, m) \quad \text{if } n > 0 \end{aligned}$$
- Approach described in [PLDI'90], [ILPS'97] (for lower bounds, nondet relations, balanced costs), [ICLP'07, Bytecode'09] (for user-defined resources).

# "Classical" Cost Analysis (cost relations): Example

```
nrev([],[]).
nrev([H|L],R) :- nrev(L,R1), app(R1,[H],R).

app([],L,L).
app([H|L],L1,[H|R]) :- app(L,L1,R).
```

- Cost relations for resolution steps $n = length(X)$ (length of list $X$)
- Cost of *nrev*:
  $$\mathrm{C}_{nrev}(0) = 1$$
  $$\mathrm{C}_{nrev}(n) = 1 + \mathrm{C}_{nrev}(n-1) + \mathrm{C}_{app}(n-1,1) \quad \text{if } n > 0$$
- Cost of *app* $\rightarrow$ closed form: $\mathrm{C}_{app}(n,m) = n + 1$ for $n \geq 0$.
  $$\mathrm{C}_{app}(0,m) = 1$$
  $$\mathrm{C}_{app}(n,m) = 1 + \mathrm{C}_{app}(n-1,m) \quad \text{if } n > 0$$
- Approach described in [PLDI'90], [ILPS'97] (for lower bounds, nondet relations, balanced costs), [ICLP'07, Bytecode'09] (for user-defined resources).

# "Classical" Cost Analysis (cost relations): Example

```
nrev([],[]).
nrev([H|L],R) :- nrev(L,R1), app(R1,[H],R).

app([],L,L).
app([H|L],L1,[H|R]) :- app(L,L1,R).
```

- Cost relations for resolution steps $n = length(X)$ (length of list $X$)
- Cost of *nrev*:

$$\begin{aligned} \mathrm{C}_{nrev}(0) &= 1 \\ \mathrm{C}_{nrev}(n) &= 1 + \mathrm{C}_{nrev}(n-1) + n \quad \text{if } n > 0 \end{aligned}$$

- Cost of *app* $\rightarrow$ closed form: $\mathrm{C}_{app}(n, m) = n + 1$ for $n \geq 0$.

$$\begin{aligned} \mathrm{C}_{app}(0, m) &= 1 \\ \mathrm{C}_{app}(n, m) &= 1 + \mathrm{C}_{app}(n-1, m) \quad \text{if } n > 0 \end{aligned}$$

- Approach described in [PLDI'90], [ILPS'97] (for lower bounds, nondet relations, balanced costs), [ICLP'07, Bytecode'09] (for user-defined resources).

# "Classical" Cost Analysis (cost relations): Example

```
nrev([],[]).
nrev([H|L],R) :- nrev(L,R1), app(R1,[H],R).

app([],L,L).
app([H|L],L1,[H|R]) :- app(L,L1,R).
```

- Cost relations for resolution steps $\quad n = \mathit{length}(X)$ (length of list $X$)
- Cost of *nrev* $\rightarrow$ closed form: $\mathrm{C}_{nrev}(n) = \frac{1}{2}\ n^2 + \frac{3}{2}\ n + 1$, for $n \geq 0$.

  $$\begin{aligned} \mathrm{C}_{nrev}(0) &= \quad 1 \\ \mathrm{C}_{nrev}(n) &= \quad 1 + \mathrm{C}_{nrev}(n-1) + n \quad \text{if } n > 0 \end{aligned}$$

- Cost of *app* $\rightarrow$ closed form: $\mathrm{C}_{app}(n, m) = n + 1$ for $n \geq 0$.

  $$\begin{aligned} \mathrm{C}_{app}(0, m) &= \quad 1 \\ \mathrm{C}_{app}(n, m) &= \quad 1 + \mathrm{C}_{app}(n-1, m) \quad \text{if } n > 0 \end{aligned}$$

- Approach described in [PLDI'90], [ILPS'97] (for lower bounds, nondet relations, balanced costs), [ICLP'07, Bytecode'09] (for user-defined resources).

# "Classical" Cost Analysis (cost relations): Example

```
nrev([],[]).
nrev([H|L],R) :- nrev(L,R1), app(R1,[H],R).

app([],L,L).
app([H|L],L1,[H|R]) :- app(L,L1,R).
```

- Cost relations for resolution steps $n = length(X)$ (length of list $X$)
- Cost of *nrev* $\rightarrow$ closed form: $\mathrm{C}_{nrev}(n) = \frac{1}{2}\, n^2 + \frac{3}{2}\, n + 1$, for $n \geq 0$.
  $$\mathrm{C}_{nrev}(0) = 1$$
  $$\mathrm{C}_{nrev}(n) = 1 + \mathrm{C}_{nrev}(n-1) + n \quad \text{if } n > 0$$
- Cost of *app* $\rightarrow$ closed form: $\mathrm{C}_{app}(n, m) = n + 1$ for $n \geq 0$.
  $$\mathrm{C}_{app}(0, m) = 1$$
  $$\mathrm{C}_{app}(n, m) = 1 + \mathrm{C}_{app}(n-1, m) \quad \text{if } n > 0$$
- Approach described in [PLDI'90], [ILPS'97] (for lower bounds, nondet relations, balanced costs), [ICLP'07, Bytecode'09] (for user-defined resources).