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

P. López-García\textsuperscript{1,2} M. Klemen\textsuperscript{1} U. Liqat\textsuperscript{1} Manuel Hermenegildo\textsuperscript{1,3}

\textsuperscript{1}IMDEA Software Institute
\textsuperscript{2}Spanish Research Council (CSIC)
\textsuperscript{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 $\rightarrow$ 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 → accumulated cost.
- I.e., how the total resource usage of the execution of a program is distributed over selected parts of it (cost centers → predicates).

**Static profiling → 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.
→ 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.:
   
   \[
   \text{:- true pred append(A,B,C) : list * list * var} \\
   \Rightarrow ( \text{size\_lb(C, length(A)+length(B))}, \\
   \text{size\_ub(C, length(A)+length(B))} ).
   \]

[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.

4. 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 $m$:
  $size_{length}([4, 2, 7]) = 3$
  $size_{length}([]) = 0$
  $size_{term\_depth}(f(a, g(b))) = 2$

- The function $diff_m(t_1, t_2)$ gives the size difference between two terms $t_1$ and $t_2$ under the metric $m$:
  $diff_{length}([2, 3|L], [4|L]) = 1$
  $diff_{length}(L, [H|L]) = -1$
  $diff_{term\_depth}(f(a, g(X)), X) = 2$
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 $m$:
  
  $size_{\text{length}}([4, 2, 7]) = 3$
  
  $size_{\text{length}}([[]]) = 0$
  
  $size_{\text{term\_depth}}(f(a, g(b))) = 2$

- The function $diff_m(t_1, t_2)$ gives the size difference between two terms $t_1$ and $t_2$ under the metric $m$:
  
  $diff_{\text{length}}([2, 3|\text{L}], [4|\text{L}]) = 1$
  
  $diff_{\text{length}}(\text{L}, [\text{H}|\text{L}]) = -1$
  
  $diff_{\text{term\_depth}}(f(a, g(X)), X) = 2$
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 \textit{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): \(\langle \text{app.1.1} \rangle = \text{length}(L)\) and \(\langle \text{app.1} \rangle = \text{length}([H|L])\).
- First, we consider predicate \texttt{app} (\(A, B, C\)) (third arg is output).

- We want to obtain intra-predicate argument size relations:

\[ Sz^\text{app}_3(x,y) \] represents the size of the third argument of \texttt{app} as a function of its input data sizes \((x = \text{length}(A)\) and \(y = \text{length}(B))\).
Size Analysis (size relations): Example

\[\text{:- entry nrev/2 : list(num) * var.}\]

\[\text{nrev([],[]).}\]
\[\text{nrev([H\mid L],R) :- nrev(L,R1), app(R1,[H],R).}\]
\[\text{app([],L,L).}\]
\[\text{app([H\mid L],L1,[H\mid R]) :- app(L,L1,R).}\]

- The automatically inferred size metric is \textit{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): \(\langle \text{app.1.1} \rangle = \text{length}(\text{L})\) and \(\langle \text{app.1} \rangle = \text{length}(\text{[H \mid L]})\).
  - First, we consider predicate \(\text{app(A,B,C)}\) (third arg is output).
  - We want to obtain intra-predicate argument size relations:
    \(Sz_{3}^{\text{app}}(x,y)\) represents the size of the third argument of \(\text{app}\) as a function of its input data sizes \((x = \text{length}(A)\) and \(y = \text{length}(B))\).
Size Analysis (size relations): Example

```prolog
:- 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 `<b.i>` 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 `<p.j.i>` 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 $b$.

→ $p$ denotes the predicate called (for readability).

- Example (2nd clause): `<app.1.1> = length(L)` and `<app.1> = length([H|L])`.
- First, we consider predicate `app(A,B,C)` (third arg is output).
- We want to obtain intra-predicate argument size relations: $Sz_{app}^3(x,y)$ 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:
  \[
  \langle app.1.1 \rangle = \langle app.1 \rangle + \text{diff}(L, [H|L]) \quad (\text{inter-predicate})
  \]
**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:**
  
  \[
  \text{length}(L) = \text{length}([H|L]) + \text{diff}(L, [H|L])
  \]
Size Analysis (size relations): Example

\[
\text{app}([], L, L). \\
\text{app}([H | L], L_1, [H | R]) :- \text{app}(L, L_1, R).
\]

- Argument size relations for the recursive clause:
  
  \[
  \text{length}(L) = \text{length}([H | L]) - 1
  \]
Size Analysis (size relations): Example

\begin{verbatim}
app([], L, L).
app([H|L], L1, [H|R]) :- app(L, L1, R).
\end{verbatim}

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

\[
\text{app}([], L, L). \\
\text{app}([H|L], L1, [H|R]) :- \text{app}(L, L1, R).
\]

- Argument size relations for the recursive clause:

\[
\langle app.1.1 \rangle = \langle app.1 \rangle - 1
\]
Size Analysis (size relations): Example

\[
\text{app}([], L, L).
\text{app}([H|L], L1, [H|R]) :- \text{app}(L, L1, R).
\]

- Argument size relations for the recursive clause:
  \[
  \langle app.1.1 \rangle = \langle app.1 \rangle - 1
  \]
  \[
  \langle app.1.2 \rangle = \langle app.2 \rangle + \text{diff}(L1, L1) \quad (\text{inter-predicate})
  \]
**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:**
  
  \[
  \langle app.1.1 \rangle = \langle app.1 \rangle - 1 \\
  \langle app.1.2 \rangle = \langle app.2 \rangle \equiv length(L1) = length(L1) + 0
  \]
Size Analysis (size relations): Example

\begin{verbatim}
app([],L,L).
app([H|L],L1,[H|R]) :- app(L,L1,R).
\end{verbatim}

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

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

\[
\text{Sz}_{app}^3(0, \langle app.2 \rangle) = \langle app.2 \rangle
\]

The equations are solved, obtaining the closed-form function:

\[
\text{Sz}_{app}^3(n, m) = n + m \text{ if } n \geq 0
\]

which is used for the analysis of predicate `nrev`.
Size Analysis (size relations): Example

\[\text{app}([], L, L).\]
\[\text{app}([H|L], L1, [H|R]) :- \text{app}(L, L1, R).\]

- Argument size relations for the recursive clause:
  \[
  \langle app.1.1 \rangle = \langle app.1 \rangle - 1 \\
  \langle app.1.2 \rangle = \langle app.2 \rangle \\
  \langle app.1.3 \rangle = Sz^3_{app}(\langle app.1.1 \rangle, \langle app.1.2 \rangle) \quad (\text{intra-predicate})
  \]

From the first clause of \text{app}, we obtain the equation:
\[Sz^3_{app}(0, \langle app.2 \rangle) = \langle app.2 \rangle\]
The equations are solved, obtaining the closed-form function:
\[Sz^3_{app}(n, m) = n + m \text{ if } n \geq 0\]
which is used for the analysis of predicate \text{nrev}. 

Lopez-Garcia, Klemen, Liqat, Hermenegildo
Static Profiling of Parametric Resource Usage
Melbourne, CICLOPS, August 28, 2017 7 / 36
Size Analysis (size relations): Example

\begin{verbatim}
app([], L, L).
app([H|L], L1, [H|R]) :- app(L, L1, R).
\end{verbatim}

- 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:
  \[
  \langle app.1.1 \rangle = \langle app.1 \rangle - 1 \\
  \langle app.1.2 \rangle = \langle app.2 \rangle \\
  \langle app.1.3 \rangle = Sz^{app}_3(\langle app.1 \rangle - 1, \langle app.2 \rangle) \quad (normalizing)
  \]
Size Analysis (size relations): Example

\[
\begin{align*}
\text{app}([], L, L). \\
\text{app}([H|L], L1, [H|R]) & : - \text{app}(L, L1, R).
\end{align*}
\]

- Argument size relations for the recursive clause:
  \[
  \langle app.1.1 \rangle = \langle app.1 \rangle - 1 \\
  \langle app.1.2 \rangle = \langle app.2 \rangle \\
  \langle app.1.3 \rangle = Sz^\text{app}_3 (\langle app.1 \rangle - 1, \langle app.2 \rangle) \\
  \langle app.3 \rangle = \langle app.1.3 \rangle + \text{diff}([H|R], R) \quad (\text{inter-predicate})
  \]
Size Analysis (size relations): Example

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

- **Argument size relations for the recursive clause:**
  - $\langle app.1.1 \rangle = \langle app.1 \rangle - 1$
  - $\langle app.1.2 \rangle = \langle app.2 \rangle$
  - $\langle app.1.3 \rangle = Sz^app_3(\langle app.1 \rangle - 1, \langle app.2 \rangle)$
  - $\langle app.3 \rangle = \langle app.1.3 \rangle + 1$

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

$$Sz^app_3(0, \langle app.2 \rangle) = \langle app.2 \rangle$$

The equations are solved, obtaining the closed-form function:

$$Sz^app_3(n, m) = n + m$$ if $n \geq 0$

which is used for the analysis of predicate `nrev`.
Argument size relations for the recursive clause:

\[
\langle \text{app.1.1} \rangle = \langle \text{app.1} \rangle - 1 \\
\langle \text{app.1.2} \rangle = \langle \text{app.2} \rangle \\
\langle \text{app.1.3} \rangle = Sz_3^{\text{app}}(\langle \text{app.1} \rangle - 1, \langle \text{app.2} \rangle) \\
\langle \text{app.3} \rangle = Sz_3^{\text{app}}(\langle \text{app.1} \rangle - 1, \langle \text{app.2} \rangle) + 1 \quad \text{(normalizing)}
\]
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:
  \[
  \begin{align*}
  \langle \text{app.1.1} \rangle &= \langle \text{app.1} \rangle - 1 \\
  \langle \text{app.1.2} \rangle &= \langle \text{app.2} \rangle \\
  \langle \text{app.1.3} \rangle &= Sz^\text{app}_3 (\langle \text{app.1} \rangle - 1, \langle \text{app.2} \rangle) \\
  \langle \text{app.3} \rangle &= Sz^\text{app}_3 (\langle \text{app.1} \rangle - 1, \langle \text{app.2} \rangle) + 1 \\
  Sz^\text{app}_3 (\langle \text{app.1} \rangle, \langle \text{app.2} \rangle) &= \langle \text{app.3} \rangle \quad \text{(intra-predicate)}
  \end{align*}
  \]
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:
\[
\begin{align*}
\langle app.1.1 \rangle &= \langle app.1 \rangle - 1 \\
\langle app.1.2 \rangle &= \langle app.2 \rangle \\
\langle app.1.3 \rangle &= Sz_3^{app}(\langle app.1 \rangle - 1, \langle app.2 \rangle) \\
\langle app.3 \rangle &= Sz_3^{app}(\langle app.1 \rangle - 1, \langle app.2 \rangle) + 1 \\
Sz_3^{app}(\langle app.1 \rangle, \langle app.2 \rangle) &= Sz_3^{app}(\langle app.1 \rangle - 1, \langle app.2 \rangle) + 1 \quad \text{(normalizing)}
\end{align*}
\]
Size Analysis (size relations): Example

\[
\text{app}([], L, L). \\
\text{app}([H|L], L1, [H|R]) :- \text{app}(L, L1, R).
\]

- Argument size relations for the recursive clause:
  \[
  \langle app.1.1 \rangle = \langle app.1 \rangle - 1 \\
  \langle app.1.2 \rangle = \langle app.2 \rangle \\
  \langle app.1.3 \rangle = \text{Sz}^{app}_3(\langle app.1 \rangle - 1, \langle app.2 \rangle) \\
  \langle app.3 \rangle = \text{Sz}^{app}_3(\langle app.1 \rangle - 1, \langle app.2 \rangle) + 1 \\
  \text{Sz}^{app}_3(\langle app.1 \rangle, \langle app.2 \rangle) = \text{Sz}^{app}_3(\langle app.1 \rangle - 1, \langle app.2 \rangle) + 1
  \]

From the first clause of \text{app}, we obtain the equation:

\[
\text{Sz}^{app}_3(0, \langle app.2 \rangle) = \langle app.2 \rangle
\]

The equations are solved, obtaining the closed-form function:

\[
\text{Sz}^{app}_3(n, m) = n + m \text{ if } n \geq 0
\]

which is used for the analysis of predicate \text{nrev}.
Argument size relations for the recursive clause:

\[
\begin{align*}
\langle \text{app.1.1} \rangle &= \langle \text{app.1} \rangle - 1 \\
\langle \text{app.1.2} \rangle &= \langle \text{app.2} \rangle \\
\langle \text{app.1.3} \rangle &= Sz_3^{\text{app}}(\langle \text{app.1} \rangle - 1, \langle \text{app.2} \rangle) \\
\langle \text{app.3} \rangle &= Sz_3^{\text{app}}(\langle \text{app.1} \rangle - 1, \langle \text{app.2} \rangle) + 1 \\
Sz_3^{\text{app}}(\langle \text{app.1} \rangle, \langle \text{app.2} \rangle) &= Sz_3^{\text{app}}(\langle \text{app.1} \rangle - 1, \langle \text{app.2} \rangle) + 1
\end{align*}
\]

From the first clause of \texttt{app}, we obtain the equation:

\[
Sz_3^{\text{app}}(0, \langle \text{app.2} \rangle) = \langle \text{app.2} \rangle
\]
Size Analysis (size relations): Example

\texttt{app([],L,L).}
\texttt{app([H|L],L1,[H|R]) :- app(L,L1,R).}

- Argument size relations for the recursive clause:
  \[
  \langle \text{app.1.1} \rangle = \langle \text{app.1} \rangle - 1 \\
  \langle \text{app.1.2} \rangle = \langle \text{app.2} \rangle \\
  \langle \text{app.1.3} \rangle = Sz^\text{app}_3 (\langle \text{app.1} \rangle - 1, \langle \text{app.2} \rangle) \\
  \langle \text{app.3} \rangle = Sz^\text{app}_3 (\langle \text{app.1} \rangle - 1, \langle \text{app.2} \rangle) + 1 \\
  Sz^\text{app}_3 (\langle \text{app.1} \rangle, \langle \text{app.2} \rangle) = Sz^\text{app}_3 (\langle \text{app.1} \rangle - 1, \langle \text{app.2} \rangle) + 1
  \]

- From the first clause of \text{app}, we obtain the equation:
  \[
  Sz^\text{app}_3 (0, \langle \text{app.2} \rangle) = \langle \text{app.2} \rangle
  \]

- The equations:
  \[
  Sz^\text{app}_3 (0, \langle \text{app.2} \rangle) = \langle \text{app.2} \rangle \\
  Sz^\text{app}_3 (\langle \text{app.1} \rangle, \langle \text{app.2} \rangle) = Sz^\text{app}_3 (\langle \text{app.1} \rangle - 1, \langle \text{app.2} \rangle) + 1
  \]
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:**
  \[\langle app.1 \rangle = \langle app.1 \rangle - 1\]
  \[\langle app.1.2 \rangle = \langle app.2 \rangle\]
  \[\langle app.1.3 \rangle = \text{Sz}^{app}_3 (\langle app.1 \rangle - 1, \langle app.2 \rangle)\]
  \[\langle app.3 \rangle = \text{Sz}^{app}_3 (\langle app.1 \rangle - 1, \langle app.2 \rangle) + 1\]
  \[\text{Sz}^{app}_3 (\langle app.1 \rangle, \langle app.2 \rangle) = \text{Sz}^{app}_3 (\langle app.1 \rangle - 1, \langle app.2 \rangle) + 1\]

- **From the first clause of app**, we obtain the equation:
  \[\text{Sz}^{app}_3 (0, \langle app.2 \rangle) = \langle app.2 \rangle\]

- **The equations** \((n = \langle app.1 \rangle, m = \langle app.2 \rangle):\)
  \[\text{Sz}^{app}_3 (n, m) = m \quad \text{if } n = 0\]
  \[\text{Sz}^{app}_3 (n, m) = \text{Sz}^{app}_3 (n - 1, m) + 1 \quad \text{if } n > 0\]
Size Analysis (size relations): Example

\[
\text{app}([], L, L).
\]
\[
\text{app}([H|L], L1, [H|R]) :- \text{app}(L, L1, R).
\]

- Argument size relations for the recursive clause:
  \[
  \langle \text{app}.1.1 \rangle = \langle \text{app}.1 \rangle - 1 \\
  \langle \text{app}.1.2 \rangle = \langle \text{app}.2 \rangle \\
  \langle \text{app}.1.3 \rangle = \text{Sz}_{3}^{\text{app}}(\langle \text{app}.1 \rangle - 1, \langle \text{app}.2 \rangle) \\
  \langle \text{app}.3 \rangle = \text{Sz}_{3}^{\text{app}}(\langle \text{app}.1 \rangle - 1, \langle \text{app}.2 \rangle) + 1 \\
  \text{Sz}_{3}^{\text{app}}(\langle \text{app}.1 \rangle, \langle \text{app}.2 \rangle) = \text{Sz}_{3}^{\text{app}}(\langle \text{app}.1 \rangle - 1, \langle \text{app}.2 \rangle) + 1
  \]

- From the first clause of \text{app}, we obtain the equation:
  \[
  \text{Sz}_{3}^{\text{app}}(0, \langle \text{app}.2 \rangle) = \langle \text{app}.2 \rangle
  \]

- The equations (\( n = \langle \text{app}.1 \rangle, \ m = \langle \text{app}.2 \rangle \)):
  \[
  \text{Sz}_{3}^{\text{app}}(n, m) = m \quad \text{if } n = 0 \\
  \text{Sz}_{3}^{\text{app}}(n, m) = \text{Sz}_{3}^{\text{app}}(n - 1, m) + 1 \quad \text{if } n > 0
  \]
  are solved, obtaining the closed-form function:
  \[
  \text{Sz}_{3}^{\text{app}}(n, m) = n + m \quad \text{if } n \geq 0
  \]
Size Analysis (size relations): Example

\[ \text{app}([], L, L). \]
\[ \text{app}([H | L], L1, [H | R]) :- \text{app}(L, L1, R). \]

- Argument size relations for the recursive clause:
  \[
  \langle app.1.1 \rangle = \langle app.1 \rangle - 1
  \]
  \[
  \langle app.1.2 \rangle = \langle app.2 \rangle
  \]
  \[
  \langle app.1.3 \rangle = Sz^{app}_3 (\langle app.1 \rangle - 1, \langle app.2 \rangle)
  \]
  \[
  \langle app.3 \rangle = Sz^{app}_3 (\langle app.1 \rangle - 1, \langle app.2 \rangle) + 1
  \]
  \[
  Sz^{app}_3 (\langle app.1 \rangle, \langle app.2 \rangle) = Sz^{app}_3 (\langle app.1 \rangle - 1, \langle app.2 \rangle) + 1
  \]

- From the first clause of \text{app}, we obtain the equation:
  \[
  Sz^{app}_3 (0, \langle app.2 \rangle) = \langle app.2 \rangle
  \]

- The equations \((n = \langle app.1 \rangle, m = \langle app.2 \rangle)\):
  \[
  Sz^{app}_3 (n, m) = m \quad \text{if} \ n = 0
  \]
  \[
  Sz^{app}_3 (n, m) = Sz^{app}_3 (n - 1, m) + 1 \quad \text{if} \ n > 0
  \]
are solved, obtaining the closed-form function:
\[
Sz^{app}_3 (n, m) = n + m \quad \text{if} \ n \geq 0
\]
which is used for the analysis of predicate \text{nrev}.
Size Analysis (size relations): Example

\[
\text{nrev}([], []). \\
\text{nrev}([H|L], R) :- \text{nrev}(L, R1), \text{app}(R1, [H], R).
\]

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

\[
\text{nrev([],[]).}
\]
\[
\text{nrev([H|L],R) :- nrev(L,R1), app(R1,[H],R).}
\]

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

Argument size relations for the recursive clause:

\[
\text{length}(L) = \text{length}([H|L]) - 1
\]
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:
  \[ \langle nrev.1.1 \rangle = \langle nrev.1 \rangle - 1 \equiv \text{length}(L) = \text{length}([H|L]) - 1 \]
Size Analysis (size relations): Example

\[
\text{nrev}([],[]).
\text{nrev}([H|L],R) :- \text{nrev}(L,R1), \text{app}(R1,[H],R).
\]

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

Argument size relations for the recursive clause:

\[
\langle nrev.1.1 \rangle = \langle nrev.1 \rangle - 1
\]

\[
\langle nrev.1.2 \rangle = Sz_{2}^{nrev}(\langle nrev.1.1 \rangle) \quad (\text{intra-predicate})
\]
Size Analysis (size relations): Example

\begin{verbatim}
nrev([],[]).
nrev([H|L],R) :- nrev(L,R1), app(R1,[H],R).
\end{verbatim}

- We now switch to predicate \texttt{nrev(A, B)}, where \( A \) and \( B \) are input and output arguments respectively.
- Argument size relations for the recursive clause:
  \( \langle nrev.1.1 \rangle = \langle nrev.1 \rangle - 1 \)
  \( \langle nrev.1.2 \rangle = Sz_{nrev}^2(\langle nrev.1.1 \rangle) \equiv length(R1) = Sz_{nrev}^2(length(L)) \)
Size Analysis (size relations): Example

\[
\text{nrev([],[]).}
\text{nrev([H|L],R) :- nrev(L,R1), app(R1,[H],R).}
\]

- We now switch to predicate \( \text{nrev}(A, B) \), where \( A \) and \( B \) are input and output arguments respectively.
- Argument size relations for the recursive clause:
  \[
  \langle \text{nrev.1.1} \rangle = \langle \text{nrev.1} \rangle - 1
  \text{\hspace{1cm} (normalizing)}
  \]
  \[
  \langle \text{nrev.1.2} \rangle = Sz^2_{\text{nrev}}(\langle \text{nrev.1} \rangle - 1)
  \text{\hspace{1cm} (normalizing)}
  \]
Size Analysis (size relations): Example

\[
\text{nrev}([],[]) . \\
\text{nrev}([H|L], R) :- \text{nrev}(L, R1), \text{app}(R1, [H], R).
\]

- We now switch to predicate \text{nrev}(A, B), where \text{A} and \text{B} are input and output arguments respectively.
- Argument size relations for the recursive clause:
  \[
  \langle \text{nrev.1.1} \rangle = \langle \text{nrev.1} \rangle - 1 \\
  \langle \text{nrev.1.2} \rangle = \text{Sz}_2^{nrev}(\langle \text{nrev.1} \rangle - 1) \\
  \langle \text{app.2.1} \rangle = \langle \text{nrev.1.2} \rangle \quad \text{(inter-predicate)}
  \]

\[
\text{Sz}_2^{nrev}(\langle \text{nrev.1} \rangle) = \langle \text{nrev.2} \rangle (\text{intra-predicate})
\]

From the first clause of \text{nrev}, we obtain the equation:
\[
\text{Sz}_2^{nrev}(0) = 0
\]

The equations are solved, obtaining the closed-form function:
\[
\text{Sz}_2^{nrev}(n) = n
\]
Size Analysis (size relations): Example

\[
\text{nrev}([], []).
\text{nrev}([H|L], R) :- \text{nrev}(L, R1), \text{app}(R1, [H], R).
\]

- We now switch to predicate \text{nrev}(A, B), where A and B are input and output arguments respectively.
- Argument size relations for the recursive clause:
  \[
  \langle \text{nrev.1.1} \rangle = \langle \text{nrev.1} \rangle - 1
  \]
  \[
  \langle \text{nrev.1.2} \rangle = \text{Sz}_2^{\text{nrev}}(\langle \text{nrev.1} \rangle - 1)
  \]
  \[
  \langle \text{app.2.1} \rangle = \langle \text{nrev.1.2} \rangle \equiv \text{length}(R1) = \text{length}(R1)
  \]
Size Analysis (size relations): Example

\[
\text{nrev([],[]).} \\
\text{nrev([H|L],R) :- nrev(L,R1), app(R1,[H],R).}
\]

- We now switch to predicate \( \text{nrev}(A, B) \), where \( A \) and \( B \) are input and output arguments respectively.

- Argument size relations for the recursive clause:
  \[
  \langle nrev.1.1 \rangle = \langle nrev.1 \rangle - 1 \\
  \langle nrev.1.2 \rangle = S_{nrev}^2(\langle nrev.1 \rangle - 1) \\
  \langle app.2.1 \rangle = S_{nrev}^2(\langle nrev.1 \rangle - 1) \quad (\text{normalizing})
  \]
Size Analysis (size relations): Example

\texttt{nrev([],[]).}
\texttt{nrev([H|L],R) :- nrev(L,R1), app(R1,[H],R).}

- We now switch to predicate \texttt{nrev(A, B)}, where \texttt{A} and \texttt{B} are input and output arguments respectively.
- Argument size relations for the recursive clause:
  \begin{align*}
  \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 &= \text{length}([H]) \quad (explicit \ size)
  \end{align*}
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:

- `<nrev.1.1> = <nrev.1> - 1`
- `<nrev.1.2> = Sz_2^{nrev}(<nrev.1> - 1)`
- `<app.2.1> = Sz_2^{nrev}(<nrev.1> - 1)`
- `<app.2.2> = 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:
  \[
  \langle nrev.1.1 \rangle = \langle nrev.1 \rangle - 1
  \]
  \[
  \langle nrev.1.2 \rangle = Sz_2^{nrev}(\langle nrev.1 \rangle - 1)
  \]
  \[
  \langle app.2.1 \rangle = Sz_2^{nrev}(\langle nrev.1 \rangle - 1)
  \]
  \[
  \langle app.2.2 \rangle = 1
  \]
  \[
  \langle app.2.3 \rangle = Sz_3^{app}(\langle app.2.1 \rangle, \langle app.2.2 \rangle) \quad (intra-predicate)
  \]
Size Analysis (size relations): Example

\[
\text{nrev}([], []). \\
\text{nrev}([H|L], R) :- \text{nrev}(L, R1), \text{app}(R1, [H], R).
\]

- We now switch to predicate \text{nrev}(A, B), where A and B are input and output arguments respectively.
- Argument size relations for the recursive clause:
  \[
  \langle nrev.1.1 \rangle = \langle nrev.1 \rangle - 1 \\
  \langle nrev.1.2 \rangle = Sz_{2}^{\text{nrev}}(\langle nrev.1 \rangle - 1) \\
  \langle app.2.1 \rangle = Sz_{2}^{\text{nrev}}(\langle nrev.1 \rangle - 1) \\
  \langle app.2.2 \rangle = 1 \\
  \langle app.2.3 \rangle = \langle app.2.1 \rangle + \langle app.2.2 \rangle \quad \text{using} \quad Sz_{3}^{\text{app}}(x, y) = x + y
  \]
Size Analysis (size relations): Example

\[
\text{nrev}([],[]).
\text{nrev}([H|L],R) :- \text{nrev}(L,R1), \text{app}(R1,[H],R).
\]

- We now switch to predicate \( \text{nrev}(A, B) \), 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_{nrev}^{2} (\langle nrev.1 \rangle - 1)
  
  \langle app.2.1 \rangle = Sz_{nrev}^{2} (\langle nrev.1 \rangle - 1)
  
  \langle app.2.2 \rangle = 1
  
  \langle app.2.3 \rangle = Sz_{nrev}^{2} (\langle nrev.1 \rangle - 1) + 1 \quad (normalizing)
  \]
Size Analysis (size relations): Example

\begin{verbatim}
 nrev([], []). 
 nrev([H|L], R) :- nrev(L, R1), app(R1, [H], R).
\end{verbatim}

- We now switch to predicate \texttt{nrev}(A, B), where A and B are input and output arguments respectively.

- Argument size relations for the recursive clause:
  \[
  \langle nrev.1.1 \rangle = \langle nrev.1 \rangle - 1 \\
  \langle nrev.1.2 \rangle = Sz_{nrev}^2 (\langle nrev.1 \rangle - 1) \\
  \langle app.2.1 \rangle = Sz_{nrev}^2 (\langle nrev.1 \rangle - 1) \\
  \langle app.2.2 \rangle = 1 \\
  \langle app.2.3 \rangle = Sz_{nrev}^2 (\langle nrev.1 \rangle - 1) + 1 \\
  \langle nrev.2 \rangle = \langle app.2.3 \rangle + \text{diff}(R, R) \quad \text{(inter-predicate)}
  \]

From the first clause of \texttt{nrev}, we obtain the equation:
\[
Sz_{nrev}^2 (0) = 0
\]

The equations are solved, obtaining the closed-form function:
\[
Sz_{nrev}^2 (n) = n
\]
Size Analysis (size relations): Example

\[
\text{nrev([],[]).}
\]
\[
\text{nrev([H|L], R) :- nrev(L, R1), app(R1, [H], R).}
\]

- We now switch to predicate \text{nrev(A, B)}, where \text{A} and \text{B} are input and output arguments respectively.
- Argument size relations for the recursive clause:
  \[
  \langle nrev.1.1 \rangle = \langle nrev.1 \rangle - 1
  \]
  \[
  \langle nrev.1.2 \rangle = Sz_n^{nrev}(\langle nrev.1 \rangle - 1)
  \]
  \[
  \langle app.2.1 \rangle = Sz_n^{nrev}(\langle nrev.1 \rangle - 1)
  \]
  \[
  \langle app.2.2 \rangle = 1
  \]
  \[
  \langle app.2.3 \rangle = Sz_n^{nrev}(\langle nrev.1 \rangle - 1) + 1
  \]
  \[
  \langle nrev.2 \rangle = \langle app.2.3 \rangle + 0
  \]
Size Analysis (size relations): Example

\[
\text{\texttt{nrev([],[]).}}
\]
\[
\text{\texttt{nrev([H\mid L],R) :- nrev(L,R1), app(R1,[H],R).}}
\]

- We now switch to predicate \texttt{nrev(A, B)}, where \texttt{A} and \texttt{B} are input and output arguments respectively.
- Argument size relations for the recursive clause:
  \[
  \langle nrev.1.1 \rangle = \langle nrev.1 \rangle - 1
  \]
  \[
  \langle nrev.1.2 \rangle = \text{Sz}_{\text{nrev}}^{nrev}(\langle nrev.1 \rangle - 1)
  \]
  \[
  \langle app.2.1 \rangle = \text{Sz}_{\text{nrev}}^{nrev}(\langle nrev.1 \rangle - 1)
  \]
  \[
  \langle app.2.2 \rangle = 1
  \]
  \[
  \langle app.2.3 \rangle = \text{Sz}_{\text{nrev}}^{nrev}(\langle nrev.1 \rangle - 1) + 1
  \]
  \[
  \langle nrev.2 \rangle = \text{Sz}_{\text{nrev}}^{nrev}(\langle nrev.1 \rangle - 1) + 1 \quad (\text{normalizing})
  \]
We now switch to predicate \( \text{nrev}(A, B) \), where \( A \) and \( B \) are input and output arguments respectively.

Argument size relations for the recursive clause:

\[
\begin{align*}
\langle \text{nrev.1.1} \rangle &= \langle \text{nrev.1} \rangle - 1 \\
\langle \text{nrev.1.2} \rangle &= Sz_{\text{nrev}}^2 (\langle \text{nrev.1} \rangle - 1) \\
\langle \text{app.2.1} \rangle &= Sz_{\text{nrev}}^2 (\langle \text{nrev.1} \rangle - 1) \\
\langle \text{app.2.2} \rangle &= 1 \\
\langle \text{app.2.3} \rangle &= Sz_{\text{nrev}}^2 (\langle \text{nrev.1} \rangle - 1) + 1 \\
\langle \text{nrev.2} \rangle &= Sz_{\text{nrev}}^2 (\langle \text{nrev.1} \rangle - 1) + 1 \\
Sz_{\text{nrev}}^2 (\langle \text{nrev.1} \rangle) &= \langle \text{nrev.2} \rangle \quad (\text{intra-predicate})
\end{align*}
\]
We now switch to predicate \texttt{nrev}(A, B), where A and B are input and output arguments respectively.

Argument size relations for the recursive clause:
\[
\begin{align*}
\langle nrev.1.1 \rangle &= \langle nrev.1 \rangle - 1 \\
\langle nrev.1.2 \rangle &= Sz_2^{nrev}(\langle nrev.1 \rangle - 1) \\
\langle app.2.1 \rangle &= Sz_2^{nrev}(\langle nrev.1 \rangle - 1) \\
\langle app.2.2 \rangle &= 1 \\
\langle app.2.3 \rangle &= Sz_2^{nrev}(\langle nrev.1 \rangle - 1) + 1 \\
\langle nrev.2 \rangle &= Sz_2^{nrev}(\langle nrev.1 \rangle - 1) + 1 \\
Sz_2^{nrev}(\langle nrev.1 \rangle) &= Sz_2^{nrev}(\langle nrev.1 \rangle - 1) + 1 \quad (normalizing)
\end{align*}
\]
Size Analysis (size relations): Example

\[
\text{nrev}([],[]).
\text{nrev}([H|L], R) :- \text{nrev}(L, R1), \text{app}(R1, [H], R).
\]

- We now switch to predicate \text{nrev}(A, B), where A and B are input and output arguments respectively.

- Argument size relations for the recursive clause:
  \[
  \langle \text{nrev.1.1} \rangle = \langle \text{nrev.1} \rangle - 1 \\
  \langle \text{nrev.1.2} \rangle = \text{Sz}_{\text{nrev}}^{2}(\langle \text{nrev.1} \rangle - 1) \\
  \langle \text{app.2.1} \rangle = \text{Sz}_{\text{nrev}}^{2}(\langle \text{nrev.1} \rangle - 1) \\
  \langle \text{app.2.2} \rangle = 1 \\
  \langle \text{app.2.3} \rangle = \text{Sz}_{\text{nrev}}^{2}(\langle \text{nrev.1} \rangle - 1) + 1 \\
  \langle \text{nrev.2} \rangle = \text{Sz}_{\text{nrev}}^{2}(\langle \text{nrev.1} \rangle - 1) + 1 \\
  \text{Sz}_{\text{nrev}}^{2}(\langle \text{nrev.1} \rangle) = \text{Sz}_{\text{nrev}}^{2}(\langle \text{nrev.1} \rangle - 1) + 1
  \]
We now switch to predicate \texttt{nrev(A, B)}, where \texttt{A} and \texttt{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 \texttt{nrev}, we obtain the equation:
\[
Sz_{2}^{nrev}(0) = 0
\]
Size Analysis (size relations): Example

\[\text{nrev}([],[]).\]
\[\text{nrev}([H|L],R) : - \text{nrev}(L,R1), \text{app}(R1,[H],R).\]

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

- From the first clause of \( nrev \), we obtain the equation:
  \( Sz_{nrev}^2 (0) = 0 \)

- The equations \( (n = \langle nrev.1 \rangle) \):
  \( Sz_{nrev}^2 (0) = 0 \)
  \( Sz_{nrev}^2 (n) = Sz_{nrev}^2 (n - 1) + 1 \)
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 \texttt{nrev(A, B)} is given by the following equations (where \( n = \text{length(A)} \)):
  \[
  Sz_2^{\text{nrev}}(0) = 0 \\
  Sz_2^{\text{nrev}}(n) = Sz_2^{\text{nrev}}(n - 1) + 1
  \]

- Solution: \( Sz_2^{\text{nrev}}(n) = n \).
  The length (size) of the output argument of \texttt{nrev} is equal to the length of its input.
Standard Cost: Intuition

\[\text{p}(0).\]
\[\text{p}(X) \leftarrow X > 0, \ Y \text{ is } X - 1, \ r(Y), \ q(Y), \ p(Y).\]

\[\text{q}(0).\]
\[\text{q}(X) \leftarrow X > 0, \ Y \text{ is } X - 1, \ r(Y), \ q(Y).\]

\[\text{r}(0).\]
\[\text{r}(X) \leftarrow X > 0, \ Y \text{ is } X - 1, \ r(Y).\]

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

(assume the builtins \(> /2\) and \(is /2\) have zero cost)
Standard Cost: Intuition

\begin{verbatim}
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).
\end{verbatim}

The standard cost of a call \( p(2) \) (in number of resolution steps): \( C_{p}(2) = 10 \). (also: \( C_{r}(1) = 2 \) and \( C_{q}(1) = 3 \)).
Standard Cost Relations Framework: Intuition

\[ \begin{align*}
  p(0). \\
  p(X) & :- X > 0, Y \text{ is } X - 1, r(Y), q(Y), p(Y). \\
  q(0). \\
  q(X) & :- X > 0, Y \text{ is } X - 1, r(Y), q(Y). \\
  r(0). \\
  r(X) & :- X > 0, Y \text{ is } X - 1, r(Y).
\end{align*} \]

Cost relations

\[ n = \text{size}(X) = X \text{ (actual value of } X) \]

Standard cost of \( p \):
\[
C_p(0) = 1 \\
C_p(n) = 1 + C_r(n - 1) + C_q(n - 1) + C_p(n - 1) \text{ if } n > 0
\]

Standard cost of \( q \):
\[
C_q(0) = 1 \\
C_q(n) = 1 + C_r(n - 1) + C_q(n - 1) \text{ if } n > 0
\]

Standard cost of \( r \):
\[
C_r(0) = 1 \\
C_r(n) = 1 + C_r(n - 1) \text{ if } n > 0
\]
Standard Cost Relations Framework: Intuition

\begin{verbatim}
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).
\end{verbatim}

Cost relations \[ n = \text{size}(X) = X \] (actual value of \( X \))

**Standard cost of \( p \):**
\[
C_p(0) = 1 \\
C_p(n) = 1 + C_r(n - 1) + C_q(n - 1) + C_p(n - 1) \quad \text{if } n > 0
\]

**Standard cost of \( q \):**
\[
C_q(0) = 1 \\
C_q(n) = 1 + C_r(n - 1) + C_q(n - 1) \quad \text{if } n > 0
\]

**Standard cost of \( r \rightarrow \) closed-form:** \[ C_r(n) = n + 1, \text{ for } n \geq 0. \]
\[
C_r(0) = 1 \\
C_r(n) = 1 + C_r(n - 1) \quad \text{if } n > 0
\]
Standard Cost Relations Framework: Intuition

Cost relations

\[ n = \text{size}(X) = X \text{ (actual value of } X) \]

Standard cost of \( p \):
\[
\begin{align*}
C_p(0) &= 1 \\
C_p(n) &= 1 + C_r(n-1) + C_q(n-1) + C_p(n-1) \quad \text{if } n > 0
\end{align*}
\]

Standard cost of \( q \):
\[
\begin{align*}
C_q(0) &= 1 \\
C_q(n) &= 1 + n + C_q(n-1) \quad \text{if } n > 0
\end{align*}
\]

Standard cost of \( r \) → closed-form: \( C_r(n) = n + 1 \), for \( n \geq 0 \).
\[
\begin{align*}
C_r(0) &= 1 \\
C_r(n) &= 1 + C_r(n-1) \quad \text{if } n > 0
\end{align*}
\]
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 \( n = \text{size}(X) = X \) (actual value of \( X \))

Standard cost of \( p \):
\[
C_p(0) = 1 \\
C_p(n) = 1 + C_r(n - 1) + C_q(n - 1) + C_p(n - 1) \quad \text{if } n > 0
\]

Standard cost of \( q \rightarrow \) closed form: \( C_q(n) = \frac{1}{2} n^2 + \frac{3}{2} n + 2 \) for \( n \geq 0 \).
\[
C_q(0) = 1 \\
C_q(n) = 1 + n + C_q(n - 1) \quad \text{if } n > 0
\]

Standard cost of \( r \rightarrow \) closed-form: \( C_r(n) = n + 1 \), for \( n \geq 0 \).
\[
C_r(0) = 1 \\
C_r(n) = 1 + C_r(n - 1) \quad \text{if } n > 0
\]
Standard Cost Relations Framework: Intuition

\[
\begin{align*}
\text{p(0).} \\
p(X) & : - X > 0, \ Y \text{ is } X - 1, \ r(Y), \ q(Y), \ p(Y).
\end{align*}
\]

\[
\begin{align*}
\text{q(0).} \\
q(X) & : - X > 0, \ Y \text{ is } X - 1, \ r(Y), \ q(Y).
\end{align*}
\]

\[
\begin{align*}
\text{r(0).} \\
r(X) & : - X > 0, \ Y \text{ is } X - 1, \ r(Y).
\end{align*}
\]

Cost relations

\[n = \text{size}(X) = X \text{ (actual value of } X)\]

Standard cost of p:

\[
\begin{align*}
C_p(0) &= 1 \\
C_p(n) &= 1 + n + \frac{1}{2} (n - 1)^2 + \frac{3}{2} (n - 1) + 2 + C_p(n - 1) \quad \text{if } n > 0
\end{align*}
\]

Standard cost of q → closed form:

\[
\begin{align*}
C_q(0) &= 1 \\
C_q(n) &= 1 + n + C_q(n - 1) \quad \text{if } n > 0
\end{align*}
\]

Standard cost of r → closed-form:

\[
\begin{align*}
C_r(0) &= 1 \\
C_r(n) &= 1 + C_r(n - 1) \quad \text{if } n > 0
\end{align*}
\]
Standard Cost Relations Framework: Intuition

\[ p(0). \]
\[ p(X) :- X > 0, Y \text{ is } X - 1, r(Y), q(Y), p(Y). \]

\[ q(0). \]
\[ q(X) :- X > 0, Y \text{ is } X - 1, r(Y), q(Y). \]

\[ r(0). \]
\[ r(X) :- X > 0, Y \text{ is } X - 1, r(Y). \]

Cost relations

\[ n = \text{size}(X) = X \text{ (actual value of } X) \]

Standard cost of \( p \rightarrow \) closed form:
\[ C_p(n) = \frac{1}{6} n^3 + n^2 + \frac{17}{6} n + 1, \text{ for } n \geq 0. \]
\[ C_p(0) = 1 \]
\[ C_p(n) = 1 + n + \frac{1}{2} (n - 1)^2 + \frac{3}{2} (n - 1) + 2 + C_p(n - 1) \text{ if } n > 0 \]

Standard cost of \( q \rightarrow \) closed form:
\[ C_q(n) = \frac{1}{2} n^2 + \frac{3}{2} n + 2 \text{ for } n \geq 0. \]
\[ C_q(0) = 1 \]
\[ C_q(n) = 1 + n + C_q(n - 1) \text{ if } n > 0 \]

Standard cost of \( r \rightarrow \) closed-form:
\[ C_r(n) = n + 1, \text{ for } n \geq 0. \]
\[ C_r(0) = 1 \]
\[ C_r(n) = 1 + C_r(n - 1) \text{ if } n > 0 \]
Accumulated-cost: Intuition

\[
\begin{align*}
p(0). \\
p(X) &\leftarrow X > 0, \ Y \text{ is } X - 1, \ r(Y), \ q(Y), \ p(Y). \\
q(0). \\
q(X) &\leftarrow X > 0, \ Y \text{ is } X - 1, \ r(Y), \ q(Y). \\
r(0). \\
r(X) &\leftarrow X > 0, \ Y \text{ is } X - 1, \ r(Y). 
\end{align*}
\]

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

\[
\begin{array}{c}
p(2) \\
| \\
r(1) \\
| \\
r(0) \\
r(0) \\
r(0) \\
q(1) \\
q(0) \\
r(0) \\
r(0) \\
q(0) \\
r(0) \\
p(0) \\
\end{array}
\]
Accumulated-cost: Intuition

\[
\begin{align*}
\text{Set of cost centers:} & & \diamond = \{ p, q, r \} \\
\text{We declare that predicates } & & \text{p, q, and r are cost centers.} \\
\text{Cost centers are user-defined program points (predicates, in our case) to} & & \text{which execution costs are assigned during the execution of a program.}
\end{align*}
\]

\[
\begin{align*}
p(0). \\
p(X) & : - X > 0, \ Y \text{ is } X - 1, \ r(Y), \ q(Y), \ p(Y). \\
q(0). \\
q(X) & : - X > 0, \ Y \text{ is } X - 1, \ r(Y), \ q(Y). \\
r(0). \\
r(X) & : - X > 0, \ Y \text{ is } X - 1, \ r(Y).
\end{align*}
\]
Accumulated-cost: Intuition

Set of cost centers:
\[ \diamond = \{ \text{p, q, r} \} \]

The cost of a call \( p(2) \) accumulated in cost center \( r \), denoted \( C_p^r(2) \)

Is the sum of the resolution steps that are descendant (in the call stack) of \( p(2) \), and whose closest ancestor in the call stack that is a cost center, is \( r \)
Accumulated-cost: Intuition

\[ p(0). \]
\[ p(X) :- X > 0, Y \text{ is } X - 1, r(Y), q(Y), p(Y). \]
\[ q(0). \]
\[ q(X) :- X > 0, Y \text{ is } X - 1, r(Y), q(Y). \]
\[ r(0). \]
\[ r(X) :- X > 0, Y \text{ is } X - 1, r(Y). \]

Set of cost centers:
\[ \diamondsuit = \{ p, q, r \} \]

The cost of a call \( p(2) \) accumulated in cost center \( r \rightarrow C^r_p(2) = 4 \)

Is the sum of the resolution steps that are descendant (in the call stack) of \( p(2) \), and whose closest ancestor in the call stack that is a cost center, is \( r \)
Accumulated-cost: Intuition

\[
\begin{align*}
p(0). \\
p(X) & : - \ X > 0, \ Y \text{ is } X - 1, \ r(Y), \ q(Y), \ p(Y). \\
q(0). \\
q(X) & : - \ X > 0, \ Y \text{ is } X - 1, \ r(Y), \ q(Y). \\
r(0). \\
r(X) & : - \ X > 0, \ Y \text{ is } X - 1, \ r(Y).
\end{align*}
\]

Set of cost centers:
\[\diamondsuit = \{p, q, r\}\]

The cost of a call \(p(2)\) accumulated in cost center \(q\), denoted \(C_{q}^{p}(2)\), is the sum of the resolution steps that are descendant (in the call stack) of \(p(2)\), and whose closest ancestor in the call stack that is a cost center, is \(q\).
Accumulated-cost: Intuition

Set of cost centers:
\[ \diamondsuit = \{ p, q, r \} \]

The cost of a call \( p(2) \) accumulated in cost center \( q \rightarrow c^q_p(2) = 3 \)

Is the sum of the resolution steps that are descendant (in the call stack) of \( p(2) \), and whose closest ancestor in the call stack that is a cost center, is \( q \)
Accumulated-cost: Intuition

\[
p(0).
p(X) \leftarrow X > 0, \ Y \text{ is } X - 1, \ r(Y), \ q(Y), \ p(Y).
\]

\[
q(0).
q(X) \leftarrow X > 0, \ Y \text{ is } X - 1, \ r(Y), \ q(Y).
\]

\[
r(0).
r(X) \leftarrow X > 0, \ Y \text{ is } X - 1, \ r(Y).
\]

Set of cost centers:
\[\diamondsuit = \{p, q, r\}\]

The cost of a call \(p(2)\) accumulated in cost center \(p\), denoted \(C_p^{p}(2)\), is the sum of the resolution steps that are descendant (in the call stack) of \(p(2)\), and whose closest ancestor in the call stack that is a cost center, is \(p\).
Accumulated-cost: Intuition

Set of cost centers: \( \diamondsuit = \{ p, q, r \} \)

The cost of a call \( p(2) \) accumulated in cost center \( p \rightarrow c_p^p(2) = 3 \)

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

\[
\begin{align*}
p(0).
p(X):=& \; X > 0, \; Y \; is \; X - 1, \; r(Y), \; q(Y), \; p(Y).
q(0).
q(X):=& \; X > 0, \; Y \; is \; X - 1, \; r(Y), \; q(Y).
r(0).
r(X):=& \; X > 0, \; Y \; is \; X - 1, \; r(Y).
\end{align*}
\]
Accumulated-cost: Intuition

\[ p(0). \]
\[ p(X) :- X > 0, Y \text{ is } X - 1, r(Y), q(Y), p(Y). \]
\[ q(0). \]
\[ q(X) :- X > 0, Y \text{ is } X - 1, r(Y), q(Y). \]
\[ r(0). \]
\[ r(X) :- X > 0, Y \text{ is } X - 1, r(Y). \]

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

\[ C_p(2) = C_p^p(2) + C_p^q(2) + C_p^r(2) \]
\[ 10 = 3 + 3 + 4 \]
Accumulated-cost: Intuition

\[ p(0). \]
\[ p(X) : - X > 0, \ Y \text{ is } X - 1, \ r(Y), \ q(Y), \ p(Y). \]
\[ q(0). \]
\[ q(X) : - X > 0, \ Y \text{ is } X - 1, \ r(Y), \ q(Y). \]
\[ r(0). \]
\[ r(X) : - X > 0, \ Y \text{ is } X - 1, \ r(Y). \]

Set of cost centers:
\[ \diamondsuit = \{p, q\} \]

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

\[
\begin{align*}
p(2) & \quad 1 \\
r(1) & \quad 1 \\
r(0) & \quad 1 \\
p(0) & \quad 1 \\
q(1) & \quad 1 \\
q(0) & \quad 1 \\
r(0) & \quad 1 \\
q(0) & \quad 1 \\
p(0) & \quad 1 \\
\end{align*}
\]
**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:

\[
\diamondsuit = \{p, q\}
\]

\[
C_p(2) = C_p^p(2) + C_p^q(2)
\]

\[
10 = 6 + 4
\]

Diagram:

```
      p(2)
     /   \
    p(2)
   /     \
 r(1)   q(1)
 /     /     \
 r(0) r(0) q(0) r(0)
 |   |     |     |   |
 r(0) r(0) q(0) r(0) p(0)
```

Lopez-Garcia, Klemen, Liqat, Hermenegildo

Static Profiling of Parametric Resource Usage

Melbourne, CICLOPS, August 28, 2017
Accumulated-cost: Definition

**Definition: Accumulated Cost**

The cost of a (single) call $p(n)$ accumulated in cost center $q$, denoted $C^q_p(n)$:

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

![Call Stack Diagram](image-url)
Cost Relations for Accumulated-costs in Cost Center \( r \)

\[
\begin{align*}
p(0). \\
p(X) & : \quad X > 0, \; Y \; \text{is} \; X - 1, \; r(Y), \; q(Y), \; p(Y). \\
q(0). \\
q(X) & : \quad X > 0, \; Y \; \text{is} \; X - 1, \; r(Y), \; q(Y). \\
r(0). \\
r(X) & : \quad X > 0, \; Y \; \text{is} \; X - 1, \; r(Y). \\
\end{align*}
\]

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

Cost relations

The cost of \( p \) accumulated in \( r \):
\[
\begin{align*}
C^r_{rp}(0) &= 0 \\
C^r_{rp}(n) &= 0 + C^r_{rp}(n-1) + C^r_{rq}(n-1) + C^r_{rp}(n-1) \quad \text{if } n > 0
\end{align*}
\]

The cost of \( q \) accumulated in \( r \):
\[
\begin{align*}
C^r_{rq}(0) &= 0 \\
C^r_{rq}(n) &= 0 + C^r_{rq}(n-1) \quad \text{if } n > 0
\end{align*}
\]

The cost of \( r \) accumulated in \( r \):
\[
\begin{align*}
C^r_{rr}(0) &= 1 \\
C^r_{rr}(n) &= 1 + C^r_{rr}(n-1) \quad \text{if } n > 0
\end{align*}
\]

\( n = \text{size}(X) = X \) (actual value of \( X \))
Cost Relations for Accumulated-costs in Cost Center \( r \)

Set of cost centers:

\[
\diamondsuit = \{ p, q, r \}
\]

Cost relations

The cost of \( p \) accumulated in \( r \):

\[
C_r^p(0) = 0
\]

\[
C_r^p(n) = 0 + C_r^r(n - 1) + C_q^r(n - 1) + C_p^r(n - 1) \quad \text{if } n > 0
\]

E.g. \((n = 2)\):

\[
C_r^p(2) = 0 + C_r^r(1) + C_q^r(1) + C_p^r(1) = 0 + 2 + 1 + 1 = 4
\]
Cost Relations for Accumulated-costs in Cost Center $r$

$p(0)$.
$p(X) :- X > 0, Y \text{ is } X - 1, r(Y), q(Y), p(Y)$.

$q(0)$.
$q(X) :- X > 0, Y \text{ is } X - 1, r(Y), q(Y)$.

$r(0)$.
$r(X) :- X > 0, Y \text{ is } X - 1, r(Y)$.

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

Cost relations

The cost of $p$ accumulated in $r$:
\[
C_{rp}(0) = 0
\]
\[
C_{rp}(n) = 0 + C_{rr}(n - 1) + C_{rq}(n - 1) + C_{rp}(n - 1) \quad \text{if } n > 0
\]

The cost of $q$ accumulated in $r$:
\[
C_{rq}(0) = 0
\]
\[
C_{rq}(n) = 0 + C_{rr}(n - 1) + C_{rq}(n - 1) \quad \text{if } n > 0
\]

The cost of $r$ accumulated in $r$:
\[
C_{rr}(0) = 1
\]
\[
C_{rr}(n) = 1 + C_{rr}(n - 1) \quad \text{if } n > 0
\]
Cost Relations for Accumulated-costs in Cost Center \( r \)

Set of cost centers:
\[ \diamondsuit = \{ p, q, r \} \]

Cost relations

The cost of \( p \) accumulated in \( r \):
\[
\begin{align*}
C_{rp}(0) & = 0 \\
C_{rp}(n) & = 0 + C_{rr}(n-1) + C_{rq}(n-1) + C_{rp}(n-1) \quad \text{if } n > 0
\end{align*}
\]

The cost of \( q \) accumulated in \( r \):
\[
\begin{align*}
C_{rq}(0) & = 0 \\
C_{rq}(n) & = 0 + C_{rr}(n-1) + C_{rq}(n-1) \quad \text{if } n > 0
\end{align*}
\]

The cost of \( r \) accumulated in \( r \) → closed form: \( C_{rr}(n) = n + 1 \), for \( n \geq 0 \).
\[
\begin{align*}
C_{rr}(0) & = 1 \\
C_{rr}(n) & = 1 + C_{rr}(n-1) \quad \text{if } n > 0
\end{align*}
\]
Cost Relations for Accumulated-costs in Cost Center $r$

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

Cost relations  

The cost of $p$ accumulated in $r$:  
\[
C_{rp}(0) = 0 \\
C_{rp}(n) = 0 + C_{rr}(n - 1) + C_{rq}(n - 1) + C_{rp}(n - 1) \quad \text{if } n > 0
\]

The cost of $q$ accumulated in $r$:  
\[
C_{rq}(0) = 0 \\
C_{rq}(n) = 0 + n + C_{rq}(n - 1) \quad \text{if } n > 0
\]

The cost of $r$ accumulated in $r$ $\rightarrow$ closed form:  
\[
C_{rr}(n) = n + 1, \quad \text{for } n \geq 0.
\]

\[
C_{rr}(0) = 1 \\
C_{rr}(n) = 1 + C_{rr}(n - 1) \quad \text{if } n > 0
\]
Cost Relations for Accumulated-costs in Cost Center $r$

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

Cost relations

The cost of $p$ accumulated in $r$:
\[
C_r^p(0) = 0 \\
C_r^p(n) = 0 + C_r^r(n-1) + C_q^r(n-1) + C_p^r(n-1) \quad \text{if } n > 0
\]

The cost of $q$ accumulated in $r$:
\[
C_q^r(n) = \frac{1}{2}n^2 + \frac{1}{2}n \quad \text{for } n \geq 0
\]

The cost of $r$ accumulated in $r$:
\[
C_r^r(n) = n + 1 \quad \text{for } n \geq 0.
\]

$n = \text{size}(X) = X$ (actual value of $X$)
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:
$\diamondsuit = \{p, q, r\}$

Cost relations

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

The cost of $p$ accumulated in $r$:

$C^r_p(0) = 0$

$C^r_p(n) = 0 + n + \frac{1}{2}(n - 1)^2 + \frac{1}{2}(n - 1) + C^r_p(n - 1)$ if $n > 0$

The cost of $q$ accumulated in $r$:

$C^r_q(0) = 0$

$C^r_q(n) = 0 + n + C^r_q(n - 1)$ if $n > 0$

The cost of $r$ accumulated in $r$:

Closed form: $C^r_r(n) = n + 1$, for $n \geq 0$.

$C^r_r(0) = 1$

$C^r_r(n) = 1 + C^r_r(n - 1)$ if $n > 0$
Cost Relations for Accumulated-costs in Cost Center \( r \)

\[
\begin{align*}
p(0). \\
p(X) & : - X > 0, \ Y \text{ is } X - 1, \ r(Y), \ q(Y), \ p(Y). \\
q(0). \\
q(X) & : - X > 0, \ Y \text{ is } X - 1, \ r(Y), \ q(Y). \\
r(0). \\
r(X) & : - X > 0, \ Y \text{ is } X - 1, \ r(Y).
\end{align*}
\]

Set of cost centers:
\[ \diamondsuit = \{ p, q, r \} \]

Cost relations

\( n = \text{size}(X) = X \) (actual value of \( X \))

The cost of \( p \) accumulated in \( r \):

\[
C_r^p(n) = \frac{1}{6} n^3 + \frac{1}{2} n^2 + \frac{1}{3} n, \quad \text{for } n \geq 0.
\]

\[
C_r^p(0) = 0
\]

\[
C_r^p(n) = 0 + n + \frac{1}{2} (n - 1)^2 + \frac{1}{2} (n - 1) + C_r^p(n - 1) \quad \text{if } n > 0
\]

The cost of \( q \) accumulated in \( r \):

\[
C_r^q(n) = \frac{1}{2} n^2 + \frac{1}{2} n, \quad \text{for } n \geq 0
\]

\[
C_r^q(0) = 0
\]

\[
C_r^q(n) = 0 + n + C_r^q(n - 1) \quad \text{if } n > 0
\]

The cost of \( r \) accumulated in \( r \) (closed form):

\[
C_r^r(n) = n + 1, \quad \text{for } n \geq 0.
\]

\[
C_r^r(0) = 1
\]

\[
C_r^r(n) = 1 + C_r^r(n - 1) \quad \text{if } n > 0
\]
Cost Relations for Accumulated-costs in Cost Center $q$

Set of cost centers:
\[ \diamondsuit = \{p, q, r\} \]

Cost relations

The cost of $p$ accumulated in $q$:
\[
\begin{align*}
C_{q}^{p}(0) &= 0 \\
C_{q}^{p}(n) &= 0 + C_{r}^{q}(n - 1) + C_{q}^{q}(n - 1) + C_{p}^{q}(n - 1) \quad \text{if } n > 0
\end{align*}
\]

The cost of $q$ accumulated in $q$:
\[
\begin{align*}
C_{q}^{q}(0) &= 1 \\
C_{q}^{q}(n) &= 1 + C_{r}^{q}(n - 1) + C_{q}^{q}(n - 1) \quad \text{if } n > 0
\end{align*}
\]

The cost of $r$ accumulated in $q$:
\[
\begin{align*}
C_{r}^{q}(0) &= 0 \\
C_{r}^{q}(n) &= 0 + C_{r}^{q}(n - 1) \quad \text{if } n > 0
\end{align*}
\]
Cost Relations for Accumulated-costs in Cost Center $q$

Set of cost centers:

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

Cost relations

The cost of $p$ accumulated in $q$:

- $C^q_{p}(0) = 0$
- $C^q_{p}(n) = 0 + C^q_{r}(n - 1) + C^q_{q}(n - 1) + C^q_{p}(n - 1)$ if $n > 0$

The cost of $q$ accumulated in $q$:

- $C^q_{q}(0) = 1$
- $C^q_{q}(n) = 1 + C^q_{r}(n - 1) + C^q_{q}(n - 1)$ if $n > 0$

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

$$\forall r, q \in \Diamond, \text{ if } r \not\Rightarrow^\ast_{\alpha} q \text{ then } C^q_{r}(\overline{x}) = 0 \quad \text{(Lemma 3)}$$
Cost Relations for Accumulated-costs in Cost Center $q$

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

Cost relations

The cost of $p$ accumulated in $q$:

$\forall p, q \in \diamondsuit$, if $p \neq q \in \alpha$ then $C_{q}^{\alpha}(\bar{x}) = 0$ (Lemma 3)
Cost Relations for Accumulated-costs in Cost Center $q$

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

Cost relations

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

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

The cost of $r$ accumulated in $q$ \(\rightarrow\)
\[ C^q_r(n) = 0, \text{ for } n \geq 0. \]

\(\forall r, q \in \Diamond, \text{ if } r \not\xrightarrow{\alpha}^* q \text{ then } C^q_r(\bar{x}) = 0 \) (Lemma 3)
Cost Relations for Accumulated-costs in Cost Center \( p \)

\[
\begin{align*}
\text{p}(0). \\
\text{p}(X) & : - \quad X > 0, \text{ Y is } X - 1, \text{ r(Y), q(Y), p(Y).} \\
\text{q}(0). \\
\text{q}(X) & : - \quad X > 0, \text{ Y is } X - 1, \text{ r(Y), q(Y).} \\
\text{r}(0). \\
\text{r}(X) & : - \quad X > 0, \text{ Y is } X - 1, \text{ r(Y).}
\end{align*}
\]

Set of cost centers:

\[
\diamondsuit = \{ p, q, r \}
\]

Cost relations

The cost of \( p \) accumulated in \( p \):

\[
\begin{align*}
C_p^p(0) &= 1 \\
C_p^p(n) &= 1 + C_r^n(n - 1) + C_q^n(n - 1) + C_p^n(n - 1) \quad \text{if } n > 0 \\
C_q^p(n) &= 0 \quad \text{(by Lemma 3, since } q \not\xrightarrow{\alpha}^* p) \\
C_r^p(n) &= 0 \quad \text{(by Lemma 3, since } r \not\xrightarrow{\alpha}^* p).
\end{align*}
\]

\( n = \text{size}(X) = X \) (actual value of \( X \))
Cost Relations for Accumulated-costs in Cost Center \( p \)

\[
\begin{align*}
p(0). \\
p(X) &\leftarrow X > 0, \text{ Y is } X - 1, \ r(Y), \ q(Y), \ p(Y). \\
q(0). \\
q(X) &\leftarrow X > 0, \text{ Y is } X - 1, \ r(Y), \ q(Y). \\
r(0). \\
r(X) &\leftarrow X > 0, \text{ Y is } X - 1, \ r(Y).
\end{align*}
\]

Set of cost centers:
\[ \diamondsuit = \{ p, q, r \} \]

Cost relations

The cost of \( p \) accumulated in \( p \):

\[
\begin{align*}
C_p^p(0) &= 1 \\
C_p^p(n) &= 1 + C_p^p(n - 1) \quad \text{if } n > 0 \\
C_p^q(n) &= 0 \text{ (by Lemma 3, since } q \not\Rightarrow^* p). \\
C_p^r(n) &= 0 \text{ (by Lemma 3, since } r \not\Rightarrow^* p). \\
n &= \text{size}(X) = X \text{ (actual value of } X) 
\end{align*}
\]
Cost Relations for Accumulated-costs in Cost Center \( p \)

\[
\begin{align*}
\text{Cost relations} & \quad n = \text{size}(X) = X \text{ (actual value of } X) \\
\text{The cost of } p \text{ accumulated in } p & \rightarrow C_p^n(n) = n + 1, \text{ for } n \geq 0.
\end{align*}
\]

\[
\begin{align*}
C_p^p(0) &= 1 \\
C_p^p(n) &= 1 + C_p^p(n - 1) \quad \text{if } n > 0 \\
C_p^q(n) &= 0 \text{ (by Lemma 3, since } q \not\Rightarrow^*_\alpha p) \\
C_p^r(n) &= 0 \text{ (by Lemma 3, since } r \not\Rightarrow^*_\alpha p).
\end{align*}
\]
Need for Tracking the “Environment:” Example

\[
\begin{align*}
p(0). \\
p(X) & : - X > 0, \ Y \ is \ X - 1, \ r(Y), \ q(Y), \ p(Y). \\
q(0). \\
q(X) & : - X > 0, \ Y \ is \ X - 1, \ r(Y), \ q(Y). \\
r(0). \\
r(X) & : - X > 0, \ Y \ is \ X - 1, \ r(Y).
\end{align*}
\]

Set of cost centers:
\[\diamond = \{p, q\}\]

The cost of \(p\) accumulated in \(q\):
\[
\begin{align*}
C^q_p(0) & = 0 \\
C^q_p(n) & = 0 + C^q_r(n - 1) + C^q_q(n - 1) + C^q_p(n - 1) \quad \text{if } n > 0
\end{align*}
\]

The cost of \(q\) accumulated in \(q\):
\[
\begin{align*}
C^q_q(0) & = 1 \\
C^q_q(n) & = 1 + C^q_r(n - 1) + C^q_q(n - 1) \quad \text{if } n > 0
\end{align*}
\]

We have two versions for the cost of \(r\) accumulated in \(q\):
Need for Tracking the “Environment:” Example

\[ p(0). \]
\[ p(X) : - X > 0, Y \text{ is } X - 1, r(Y), q(Y), p(Y). \]
\[ q(0). \]
\[ q(X) : - X > 0, Y \text{ is } X - 1, r(Y), q(Y). \]
\[ r(0). \]
\[ r(X) : - X > 0, Y \text{ is } X - 1, r(Y). \]

Set of cost centers:
\[ \diamond = \{ p, q \} \]

The cost of \( p \) accumulated in \( q \):
\[
C^q_p(0) = 0 \\
C^q_p(n) = 0 + C^q_r,0(n - 1) + C^q_q(n - 1) + C^q_p(n - 1) \text{ if } n > 0
\]

The cost of \( q \) accumulated in \( q \):
\[
C^q_q(0) = 1 \\
C^q_q(n) = 1 + C^q_r,1(n - 1) + C^q_q(n - 1) \text{ if } n > 0
\]

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

Under the scope of \( q \)
\[
C^q_{r,0}(0) = 1 \\
C^q_{r,0}(n) = 1 + C^q_{r,0}(n - 1) \text{ if } n > 0
\]

NOT under the scope of \( q \)
\[
C^q_{r,1}(0) = 0 \\
C^q_{r,1}(n) = 0 + C^q_{r,1}(n - 1) \text{ if } n > 0
\]
Our Extended Cost Relations for Accumulated-cost

The standard cost of a clause

\[ C \equiv p(\bar{x}) : - q_1(\bar{x}_1), \ldots, q_n(\bar{x}_n) \]

for a (single) call to \( p \):

\[ c_p(\bar{x}) = \varphi(p(\bar{x})) + \sum_{i=1}^{\text{lim}(C, \bar{x})} \text{sols}_i \times c_{q_i}(\bar{x}_i) \]

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

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

\[ \text{sols}_i = \prod_{j=1}^{i-1} s_{\text{pred}}(q_j(\bar{x}_j)) \]

\[ s_{\text{pred}}(q_j(\bar{x}_j)) \] def = number of solutions produced by \( q_j(\bar{x}_j) \).

The cost of a body literal \( q_i(\bar{x}_i) \) is obtained from the costs of all clauses applicable to it that are executed, by using an aggregation operator \( \odot \).
Our Extended Cost Relations for Accumulated-cost

The accumulated cost of a clause
\[ C \equiv p(\bar{x}) : - q_1(\bar{x}_1), \ldots, q_n(\bar{x}_n) \]

for a (single) call to \( p \):
\[ C_{c,c}^{p,e}(\bar{x}) = B_{\varphi}(p,c,e) \times \varphi(p(\bar{x})) + \sum_{i=1}^{\text{lim}(C,\bar{x})} \text{sols}_i \times C_{c,e'}^{q_i,e'}(\bar{x}_i) \times B(p,c,e,q_i) \]

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

- \( \text{lim}(C, \bar{x}) \) \( \overset{\text{def}}{=} \) index of the last body literal that is called in the execution of \( C \).
- \( \text{sols}_i \) \( \overset{\text{def}}{=} \) product of the number of solutions produced by the ancestor literals of \( q_i(\bar{x}_i) \) in the clause body:
\[ \text{sols}_i = \prod_{j=1}^{i-1} s_{\text{pred}}(q_j(\bar{x}_j)) \]
\[ s_{\text{pred}}(q_j(\bar{x}_j)) \overset{\text{def}}{=} \text{number of solutions produced by } q_j(\bar{x}_j) \]

The cost of a body literal \( q_i(\bar{x}_i) \) is obtained from the costs of all clauses applicable to it that are executed, by using an aggregation operator \( \odot \).
Our Extended Cost Relations for Accumulated-cost

The accumulated cost of a clause
\[ C \equiv p(\bar{x}) \equiv q_1(\bar{x}_1), \ldots, q_n(\bar{x}_n) \]

for a (single) call to \( p \):
\[ C^c_{p,e}(\bar{x}) = B_\varphi(p, c, e) \times \varphi(p(\bar{x})) + \sum_{i=1}^{\lim(C, \bar{x})} \text{sols}_i \times C^c_{q_i, e'}(\bar{x}_i) \times B(p, c, e, q_i) \]

- The environment \( e \) is a Boolean value (1 \( \equiv \) true and 0 \( \equiv \) false):
  \[ e = \begin{cases} 
    1 & \text{if the call to } p \text{ is under the scope of cost center } c \\
    0 & \text{otherwise}
  \end{cases} \]

- Boolean functions:
  \( B_\varphi(p, c, e) \) is 1 iff “the computation” is under the scope of \( c \).
  \[ B_\varphi(p, c, e) \overset{\text{def}}{=} (p = c \lor (p \not\in \Diamond \land e)) \]

  \( B(p, c, e, q) \) is 1 iff the body literal is under the scope of \( c \), or it may call \( c \).
  \[ B(p, c, e, q) \overset{\text{def}}{=} B_\varphi(p, c, e) \lor (q \rightsquigarrow^*_\alpha c) \]

- \( e' = \mathcal{E}(p, c, e, q_i(\bar{x}_i)) \), and \( \mathcal{E} \) is the environment change function:
  \[ \mathcal{E}(p, c, e, \_ ) \overset{\text{def}}{=} (p = c \lor (p \not\in \Diamond \land e)) \]
Our Extended Cost Relations for Accumulated-cost

The accumulated cost of a clause

\[ C \equiv p(\vec{x}) : - q_1(\vec{x}_1), \ldots, q_n(\vec{x}_n) \]

for a (single) call to \( p \):

\[
C_{p,e}^c(\vec{x}) = B_{\varphi}(p, c, e) \times \varphi(p(\vec{x})) + \sum_{i=1}^{\lim(C,\vec{x})} sols_i \times C_{q_i, e'}(\vec{x}_i) \times B(p, c, e, q_i)
\]

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

\[
C_p(\vec{x}) = \Psi(p)(\vec{x})
\]
Our Extended Cost Relations for Accumulated-cost

The accumulated cost of a clause

\[ C \equiv p(\bar{x}) : - q_1(\bar{x}_1), \ldots, q_n(\bar{x}_n) \]

for a (single) call to \( p \):

\[
C^c_{p,e}(\bar{x}) = B_\varphi(p, c, e) \times \varphi(p(\bar{x})) + \sum_{i=1}^{\text{lim}(C, \bar{x})} \text{sols}_i \times C^c_{q_i,e'}(\bar{x}_i) \times B(p, c, e, q_i)
\]

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

\[
C^c_{p,e}(\bar{x}) = \Psi(p)(\bar{x}) \times B_\varphi(p, c, e)
\]
Genericity of our Cost Relations Framework

The cost a clause for a (single) call to $p$:

$$
C^c_{p,e}(\bar{x}) = B\varphi(p, c, e) \times \varphi(p(\bar{x})) + \sum_{i=1}^{\text{lim}(C,\bar{x})} \text{sols}_i \times B(p, c, e, q_i) \times C^c_{q_i, e'}(\bar{x}_i)
$$

- 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 $B\varphi(p, c, e)$ and $B(p, c, e, q)$ to control which terms of the cost relations should be considered.

- $C^c_{p,e}(\bar{x}) \overset{\text{def}}{=} \text{part of } C_p(\bar{x})$, performed in an *environment* $e$, that is attributed to *cost center* $c$ of the program.
Some Properties of the Accumulated-cost

Definition of the calls relation, $\leadsto_\alpha$

- $p \leadsto_\alpha q$, iff a literal with predicate symbol $q$ appears in the body of a clause defining $p$.
- $\leadsto_\alpha^*$ is the reflexive transitive closure of $\leadsto_\alpha$.
- It is an abstraction (over-approximation) of the concrete “calls” relation, $\leadsto$.
Some Properties of the Accumulated-cost

- $\forall p, c \in \diamondsuit, \forall e \in \{0, 1\}$, it holds that:
  - $E(p, c, e, \bot) \overset{\text{def}}{=} (p = c)$
    (recall that $E(p, c, e, \bot) \overset{\text{def}}{=} (p = c \lor (p \not\in \diamondsuit \land e)))$
  - $B\varphi(p, c, e) \overset{\text{def}}{=} (p = c)$
    (recall that $B\varphi(p, c, e) \overset{\text{def}}{=} (p = c \lor (p \not\in \diamondsuit \land e)))$
  - $B(p, c, e, q) \overset{\text{def}}{=} (p = c) \lor (q \xrightarrow{\alpha}^* c)$
    (recall that $B(p, c, e, q) \overset{\text{def}}{=} B\varphi(p, c, e) \lor (q \xrightarrow{\alpha}^* c)$)

- This implies that $\forall p, c \in \diamondsuit$ it holds that $C^c_{p,0}(\bar{x}) = C^c_{p,1}(\bar{x})$.
- Thus, if $p \in \diamondsuit$ we omit the environment $e$ and write $C^c_p(\bar{x})$.
- (Lemma 3) $\forall p, c \in \diamondsuit$, if $p \not\xrightarrow{\alpha}^* c$ then $C^c_p(\bar{x}) = 0$.
- (Lemma 4) $\forall p \not\in \diamondsuit, \forall c \in \diamondsuit$, if $p \not\xrightarrow{\alpha}^* c$ then $C^c_{p,0}(\bar{x}) = 0$. 
Consider the following program, where predicates $p$, $q$ and $r$ are cost centers.

\[
\begin{align*}
    p(X, Y, Z) & : - X > 0, q(X, Y, Z1), Z \text{ is } Z1 \times 2. \\
    q(0, \_, 0). \\
    q(X, Y, Z) & : - r(Y, Y1), X1 \text{ is } X - 1, q(X1, Y, Z1), Z \text{ is } Z1 + Y1. \\
    r(0, 0). \\
    r(X, Y) & : - X1 \text{ is } X - 1, r(X1, Y1), Y \text{ is } Y1 + X.
\end{align*}
\]

**Standard Cost**

- $C_p(x, y) = y \times x + 2 \times x + 2$
- $C_q(x, y) = y \times x + 2 \times x + 1$
- $C_r(x, y) = x + 1$

**Accumulated Cost**

- $C^p_r(x, y) = x \times y$
- $C^r_p(x, y) = 1$
- $C^q_p(x, y) = x$

$\quad C_r(x, y) = x + 1 \rightarrow r \text{ is not costly by itself.}$

$\quad C^r_p(x, y) = x \times y \rightarrow r \text{ is responsible for the non-linear complexity of } p.$
Usefulness of the Accumulated Cost

Consider the following program, where predicates \( p \), \( q \) and \( r \) are cost centers.

\[
\begin{align*}
\text{p}(X, Y, Z) & : - X > 0, q(X, Y, Z1), Z \text{ is } Z1 \times 2. \\
q(0, _, 0) & .
\end{align*}
\]

\[
\begin{align*}
\text{q}(X, Y, Z) & : - r(Y, Y1), X1 \text{ is } X - 1, q(X1, Y, Z1), Z \text{ is } Z1 + Y1. \\
r(0, 0) & .
\end{align*}
\]

\[
\begin{align*}
r(X, Y) & : - X1 \text{ is } X - 1, r(X1, Y1), Y \text{ is } Y1 + X.
\end{align*}
\]

\[C_p(x, y) = y \times x + 2 \times x + 2 \]
\[C_q(x, y) = y \times x + 2 \times x + 1 \]
\[C_r(x, y) = x + 1 \]

\[C_r(x, y) = x + 1 \rightarrow r \text{ is not costly by itself.} \]
\[C_p(x, y) = x \times y \rightarrow r \text{ is responsible for the non-linear complexity of } p. \]
Usefulness of the Accumulated Cost

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

$$p(X, Y, Z) :- X \geq 0, \ r(Y, Y1), \ q(X, Y1, Z1), \ Z \ is \ Z1 \times 2.$$

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

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

Standard Cost
- $C_p(x, y) = x + y + 3$
- $C_q(x, y) = x + 1$
- $C_r(x, y) = x + 1$

Accumulated Cost
- $C^p_p(x, y) = 1$
- $C^q_p(x, y) = x$
- $C^r_p(x, y) = y$
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

<table>
<thead>
<tr>
<th>Cost-Centers &amp; Input Sizes</th>
<th>Accumulated Cost UB</th>
<th>Static vs. Dyn</th>
<th>Standard Cost UB</th>
<th>#Calls</th>
</tr>
</thead>
<tbody>
<tr>
<td>variance((n))*</td>
<td>1</td>
<td>0%</td>
<td>2(n^2)</td>
<td>1</td>
</tr>
<tr>
<td>sq_diff((m_1, m_2))</td>
<td>(n - 1)</td>
<td>0%</td>
<td>2(m_1 m_2 - 2m_2)</td>
<td>(n - 1)</td>
</tr>
<tr>
<td>mean((u))</td>
<td>(2n^2 - n)</td>
<td>0%</td>
<td>2(u + 1)</td>
<td>(n)</td>
</tr>
<tr>
<td>is_prime((n))*</td>
<td>1</td>
<td>0%</td>
<td>((n - 1)! + n + 3)</td>
<td>(n)</td>
</tr>
<tr>
<td>fact((m))</td>
<td>(n)</td>
<td>0%</td>
<td>(m)</td>
<td>(n)</td>
</tr>
<tr>
<td>mult((u))</td>
<td>((n - 1)! + 2)</td>
<td>0%</td>
<td>(u + 1)</td>
<td>((n - 1)! + 2)</td>
</tr>
<tr>
<td>app1((n_1, n_2, n_3))*</td>
<td>(n_1)</td>
<td>0%</td>
<td>(O(n_1 n_2 n_3))</td>
<td>1</td>
</tr>
<tr>
<td>app2((m_1, m_2))</td>
<td>(n_1 n_2)</td>
<td>0%</td>
<td>(m_1 m_2)</td>
<td>(n_1)</td>
</tr>
<tr>
<td>app3((u))</td>
<td>(2n_1 n_2 n_3)</td>
<td>0%</td>
<td>(u)</td>
<td>(n_1 n_2 + n_1)</td>
</tr>
<tr>
<td>dyade((n_1, n_2))*</td>
<td>(n_1)</td>
<td>0%</td>
<td>(n_1 (n_2 + 1))</td>
<td>1</td>
</tr>
<tr>
<td>mult((m))</td>
<td>(n_1 n_2)</td>
<td>0%</td>
<td>(m)</td>
<td>(n_1)</td>
</tr>
<tr>
<td>minsort((n))*</td>
<td>(n + 1)</td>
<td>0%</td>
<td>((n+1)^2 + \frac{n-1}{2})</td>
<td>1</td>
</tr>
<tr>
<td>findmin((m))</td>
<td>((n+1)^2 + \frac{n-1}{2})</td>
<td>7%</td>
<td>(m)</td>
<td>(n + 1)</td>
</tr>
<tr>
<td>hanoi((n))*</td>
<td>(2^n - 1)</td>
<td>0%</td>
<td>(2^{n+1} - 2)</td>
<td>1</td>
</tr>
<tr>
<td>move((m))</td>
<td>(2^n - 1)</td>
<td>0%</td>
<td>(1)</td>
<td>(2^n - 1)</td>
</tr>
<tr>
<td>coupled((n))*</td>
<td>1</td>
<td>0%</td>
<td>(n + 2)</td>
<td>1</td>
</tr>
<tr>
<td>p((m))</td>
<td>(\frac{n}{2} + \frac{(-1)^n}{4} + \frac{3}{4})</td>
<td>1.2%</td>
<td>(m + 1)</td>
<td>(\frac{n}{2} - \frac{(-1)^n}{4} + \frac{1}{4})</td>
</tr>
<tr>
<td>q((u))</td>
<td>(\frac{n}{2} - \frac{(-1)^n}{4} + \frac{1}{4})</td>
<td>0%</td>
<td>(u + 1)</td>
<td>(\frac{n}{2} + \frac{(-1)^n}{4} - \frac{1}{4})</td>
</tr>
<tr>
<td>search((n))*</td>
<td>1</td>
<td>0%</td>
<td>(2n + 2)</td>
<td>1</td>
</tr>
<tr>
<td>member((m))</td>
<td>(2n + 1)</td>
<td>0%</td>
<td>(2m + 1)</td>
<td>(2n + 1)</td>
</tr>
<tr>
<td>sublist((n_1, n_2))*</td>
<td>(n_2 + 3)</td>
<td>5%</td>
<td>(n_1 n_2 + 3n_2 + 2)</td>
<td>2</td>
</tr>
<tr>
<td>append((m))</td>
<td>(n_1 n_2 + 2n_2 - 1)</td>
<td>40%</td>
<td>(2m - 1)</td>
<td>(n_1 n_2 + 2n_2 - 1)</td>
</tr>
</tbody>
</table>
### Experimental Results: Times (milliseconds)

<table>
<thead>
<tr>
<th>Cost-Center</th>
<th>Accumulated Cost UB Transformation (FLOPS’16)</th>
<th>Standard Cost UB</th>
<th>Acc Cost/Std Cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>variance</td>
<td>3283 (-45%)</td>
<td>6038</td>
<td>3066</td>
</tr>
<tr>
<td>sq_diff</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>mean</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>isprime*</td>
<td>1245 (-42%)</td>
<td>2172</td>
<td>1231</td>
</tr>
<tr>
<td>fact</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>mult</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>app1*</td>
<td>4150 (-34%)</td>
<td>6328</td>
<td>3757</td>
</tr>
<tr>
<td>app2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>app3</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>minsort*</td>
<td>3400 (-29%)</td>
<td>4845</td>
<td>3300</td>
</tr>
<tr>
<td>findmin</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>dyade*</td>
<td>3097 (-24%)</td>
<td>4117</td>
<td>2853</td>
</tr>
<tr>
<td>mult</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>hanoi*</td>
<td>1605 (-19%)</td>
<td>1996</td>
<td>1376</td>
</tr>
<tr>
<td>move</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>coupled*</td>
<td>2407 (-14%)</td>
<td>3112</td>
<td>1877</td>
</tr>
<tr>
<td>f</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>g</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>search*</td>
<td>1079</td>
<td>N/A</td>
<td>1071</td>
</tr>
<tr>
<td>member</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>sublist*</td>
<td>3674</td>
<td>N/A</td>
<td>3610</td>
</tr>
<tr>
<td>append</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Average</td>
<td>2652 (-33%)</td>
<td>4125</td>
<td>2542</td>
</tr>
</tbody>
</table>
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
  - Alejandro Serrano
  - Nai-wei Lin
  - Mario Méndez-Lojo
  - Jorge Navas
  - 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](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 \([\text{AADEBUG'97, LOPSTR'99, ILPS-WS'97, LNCS'00, SAS'03}]\)

2003 simple function comparisons (orders).

2004 Abstraction carrying code for resources. \([\text{PPDP'05, LPAR'04}]\)

2006 Probabilistic Cost Analysis. \([\text{CLEI'06}]\)

2007 User-definable resources. \([\text{ICLP'07}]\)

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

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

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

2008 Application to energy consumption of Java bytecode. \([\text{NASA FM'08}]\)

2007-2009 User-definable resources for Java bytecode. \([\text{Bytecode'09}]\)
<table>
<thead>
<tr>
<th>Year</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>2004</td>
<td>Abstraction carrying code for resources. [PPDP’05, LPAR’04]</td>
</tr>
<tr>
<td>2006</td>
<td>Probabilistic Cost Analysis. [CLEI’06]</td>
</tr>
<tr>
<td>2007</td>
<td><strong>User-definable</strong> resources. [ICLP’07]</td>
</tr>
<tr>
<td>2007</td>
<td><strong>Multi-language</strong> support (Java bytecode, C#, FP, CLP) via <strong>Horn clause-based IR.</strong> [LOPSTR’07]</td>
</tr>
<tr>
<td>2007-2009</td>
<td>User-definable resources for Java bytecode. [Bytecode’09]</td>
</tr>
<tr>
<td>2008</td>
<td>Application to <strong>execution time</strong> (using bytecode-level models, obtained by regression). [PPDP’08]</td>
</tr>
<tr>
<td>2008</td>
<td>Application to <strong>energy consumption</strong> of Java bytecode. [NASA FM’08]</td>
</tr>
<tr>
<td>2007-2009</td>
<td>User-definable resources for Java bytecode. [Bytecode’09]</td>
</tr>
</tbody>
</table>

- 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].
<table>
<thead>
<tr>
<th>Year</th>
<th>Description</th>
<th>References</th>
</tr>
</thead>
<tbody>
<tr>
<td>2003</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2004</td>
<td>Abstraction carrying code for resources.</td>
<td>PPDP’05, LPAR’04</td>
</tr>
<tr>
<td>2006</td>
<td>Probabilistic Cost Analysis.</td>
<td>CLEI’06</td>
</tr>
<tr>
<td>2007</td>
<td><strong>User-definable resources.</strong></td>
<td>ICLP’07</td>
</tr>
<tr>
<td>2007</td>
<td>Multi-language support (Java bytecode, C#, FP, CLP) via Horn clause-based IR.</td>
<td>LOPSTR’07</td>
</tr>
<tr>
<td></td>
<td><em>Combined with user-definable resources:</em></td>
<td></td>
</tr>
<tr>
<td></td>
<td>- no need to develop specific analyzers for specific languages!</td>
<td></td>
</tr>
<tr>
<td></td>
<td>- Instrumental analyses: sharing/nullity/class dependence shape</td>
<td></td>
</tr>
<tr>
<td>2008</td>
<td>Application to <strong>execution time</strong> (using bytecode-level models, obtained by regression).</td>
<td>PPDP’08</td>
</tr>
<tr>
<td>2008</td>
<td>Application to <strong>energy consumption</strong> of Java bytecode.</td>
<td>NASA FM’08</td>
</tr>
<tr>
<td>2007-</td>
<td><strong>User-definable resources for Java bytecode.</strong></td>
<td>Bytecode’09</td>
</tr>
<tr>
<td>2009</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
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]
<table>
<thead>
<tr>
<th>Year</th>
<th>Description</th>
<th>References</th>
</tr>
</thead>
<tbody>
<tr>
<td>2010-</td>
<td>Resource verification: interval-based, improved function comparison.</td>
<td>[ICLP’10, FOPARA’12, HIP3ES’15]</td>
</tr>
<tr>
<td>2015</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2012-</td>
<td>Cost analysis as multivariant abstract interpretation.</td>
<td>[TPLP’14]</td>
</tr>
<tr>
<td>2014</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>➔ Multivariant, integrated with assertion checking, modular, incremental.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Domain: interval (piece-wise) functions.</td>
<td>[ICLP’10, FOPARA’12]</td>
</tr>
<tr>
<td>2013</td>
<td>Using sized shapes (sized types).</td>
<td>[ICLP’13]</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2013-</td>
<td>Analysis and verification of Energy:</td>
<td></td>
</tr>
<tr>
<td>2016</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>• At the ISA level</td>
<td>[LOPSTR’13]</td>
</tr>
<tr>
<td></td>
<td>• Comparing LLMV and ISA levels</td>
<td>[FOPARA’15]</td>
</tr>
<tr>
<td></td>
<td>• At the block level</td>
<td>[HIP3ES’16]</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2016</td>
<td>Static profiling / accumulated cost.</td>
<td>[FLOPS’16, TPLP’16]</td>
</tr>
</tbody>
</table>
Selected Bibliography on CiaoPP
(and Additional Slides)
CiaoPP References – Analysis and Verification of Energy


CiaoPP References – Intermediate Repr. / Multi-Lingual Support

A Flexible (C)LP-Based Approach to the Analysis of Object-Oriented Programs.  

CiaoPP References – Analysis and Verification of Resources in General


A Transformational Approach to Parametric Accumulated-cost Static Profiling.  

Resource Usage Analysis of Logic Programs via Abstract Interpretation Using Sized Types.  

Sized Type Analysis of Logic Programs (Technical Communication).  
Interval-Based Resource Usage Verification: Formalization and Prototype


User-Definable Resource Usage Bounds Analysis for Java Bytecode.

Towards Execution Time Estimation in Abstract Machine-Based Languages.

User-Definable Resource Bounds Analysis for Logic Programs.

Probabilistic Cost Analysis of Logic Programs: A First Case Study.
In XXXII Latin-American Conference on Informatics (CLEI 2006), August 2006.


CiaoPP References – Assertion Language


CiaoPP References – Overall Debugging and Verification Model

Reducing the Overhead of Assertion Run-time Checks via static analysis.

Practical Run-time Checking via Unobtrusive Property Caching.

[ICLP’09] E. Mera, P. López-García, and M. Hermenegildo.
Integrating Software Testing and Run-Time Checking in an Assertion Verification Framework.

Integrated Program Debugging, Verification, and Optimization Using Abstract Interpretation

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.

A Generic Preprocessor for Program Validation and Debugging.


CiaoPP References – Abstraction-Carrying Code


CiaoPP References – Fixpoint-based Analyzer (Abstract Interpreter)

An Efficient, Context and Path Sensitive Analysis Framework for Java Programs.

Incremental Analysis of Constraint Logic Programs.

Effectiveness of Abstract Interpretation in Automatic Parallelization: A Case Study in Logic Programming.

Optimized Algorithms for the Incremental Analysis of Logic Programs.
Springer-Verlag, September 1996.

Analyzing Logic Programs with Dynamic Scheduling.

Compile-time Derivation of Variable Dependency Using Abstract Interpretation.

On the Practicality of Global Flow Analysis of Logic Programs
CiaoPP References – Modular Analysis, Analysis of Concurrency

A Practical Type Analysis for Verification of Modular Prolog Programs.

Context-Sensitive Multivariant Assertion Checking in Modular Programs.

A Model for Inter-module Analysis and Optimizing Compilation.

Some Issues in Analysis and Specialization of Modular Ciao-Prolog Programs.

Analyzing Logic Programs with Dynamic Scheduling.

Global Analysis of Standard Prolog Programs.
CiaoPP References – Domains: Sharing/Aliasing

Identification of Logically Related Heap Regions.

Efficient Set Sharing using ZBDDs.
In 21st Int’l. WS on Languages and Compilers for Parallel Computing (LCPC’08), LNCS. Springer-Verlag, August 2008.

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.

Sharing Analysis of Arrays, Collections, and Recursive Structures.

Precise Set Sharing Analysis for Java-style Programs.

Efficient top-down set-sharing analysis using cliques.
Combined Determination of Sharing and Freeness of Program Variables Through Abstract Interpretation.

Determination of Variable Dependence Information at Compile-Time Through Abstract Interpretation.

CiaoPP References – Domains: Shape/Type Analysis

Sized Type Analysis of Logic Programs (Technical Communication).

Efficient context-sensitive shape analysis with graph-based heap models.

Heap Analysis in the Presence of Collection Libraries.

More Precise yet Efficient Type Inference for Logic Programs.
CiaoPP References – Domains: Non-failure, Determinacy

Automatic Inference of Determinacy and Mutual Exclusion for Logic Programs Using Mode and Type Information. 

Determinacy Analysis for Logic Programs Using Mode and Type Information. 

Multivariant Non-Failure Analysis via Standard Abstract Interpretation. 

Non-Failure Analysis for Logic Programs. 
Additional slides
“Classical” Cost Analysis (cost relations): Example

\begin{verbatim}
 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).
\end{verbatim}

- Cost relations for resolution steps \( n = \text{length}(X) \) (length of list \( X \))

- Cost of \( \text{nrev} \):
  \[
  C_{\text{nrev}}(0) = 1 \\
  C_{\text{nrev}}(n) = 1 + C_{\text{nrev}}(n-1) + C_{\text{app}}(n-1, 1) \quad \text{if } n > 0
  \]

- Cost of \( \text{app} \):
  \[
  C_{\text{app}}(0, m) = 1 \\
  C_{\text{app}}(n, m) = 1 + C_{\text{app}}(n-1, m) \quad \text{if } n > 0
  \]

- Approach described in \([\text{PLDI'90}], [\text{ILPS'97}]\) (for lower bounds, nondet relations, balanced costs), \([\text{ICLP'07}, \text{Bytecode'09}]\) (for user-defined resources).
“Classical” Cost Analysis (cost relations): Example

\begin{verbatim}
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).
\end{verbatim}

- Cost relations for resolution steps \( n = \text{length}(X) \) (length of list \( X \))

- Cost of \( nrev \):
  \[
  C_{nrev}(0) = 1 \\
  C_{nrev}(n) = 1 + C_{nrev}(n-1) + C_{app}(n-1,1) \quad \text{if } n > 0
  \]

- Cost of \( app \) \( \rightarrow \) closed form: \( C_{app}(n, m) = n + 1 \) for \( n \geq 0 \).
  \[
  C_{app}(0, m) = 1 \\
  C_{app}(n, m) = 1 + C_{app}(n-1, m) \quad \text{if } n > 0
  \]

- 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 \( n = \text{length}(X) \) (length of list \( X \))

Cost of \( nrev \):
\[
\begin{align*}
C_{nrev}(0) &= 1 \\
C_{nrev}(n) &= 1 + C_{nrev}(n - 1) + n \quad \text{if } n > 0
\end{align*}
\]

Cost of \( app \) → closed form: \( C_{app}(n,m) = n + 1 \) for \( n \geq 0 \).
\[
\begin{align*}
C_{app}(0,m) &= 1 \\
C_{app}(n,m) &= 1 + C_{app}(n - 1,m) \quad \text{if } n > 0
\end{align*}
\]

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

\[
\text{nrev}([], []). \\
\text{nrev}([H|L], R) :- \text{nrev}(L, R1), \text{app}(R1, [H], R).
\]

\[
\text{app}([], L, L). \\
\text{app}([H|L], L1, [H|R]) :- \text{app}(L, L1, R).
\]

- Cost relations for resolution steps \( n = \text{length}(X) \) (length of list \( X \))

- Cost of \text{nrev} \( \rightarrow \) closed form: \( C_{\text{nrev}}(n) = \frac{1}{2} n^2 + \frac{3}{2} n + 1 \), for \( n \geq 0 \).
  \[
  C_{\text{nrev}}(0) = 1 \\
  C_{\text{nrev}}(n) = 1 + C_{\text{nrev}}(n - 1) + n \quad \text{if } n > 0
  \]

- Cost of \text{app} \( \rightarrow \) closed form: \( C_{\text{app}}(n, m) = n + 1 \) for \( n \geq 0 \).
  \[
  C_{\text{app}}(0, m) = 1 \\
  C_{\text{app}}(n, m) = 1 + C_{\text{app}}(n - 1, m) \quad \text{if } n > 0
  \]

- 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

\[
\text{nrev}([], []). \\
\text{nrev}([H|L], R) :- \text{nrev}(L, R1), \text{app}(R1, [H], R).
\]

\[
\text{app}([], L, L). \\
\text{app}([H|L], L1, [H|R]) :- \text{app}(L, L1, R).
\]

- Cost relations for resolution steps \( n = \text{length}(X) \) (length of list \( X \))

- Cost of \text{nrev} \( \rightarrow \) closed form: \( C_{\text{nrev}}(n) = \frac{1}{2} n^2 + \frac{3}{2} n + 1 \), for \( n \geq 0 \).
  \[
  C_{\text{nrev}}(0) = 1 \\
  C_{\text{nrev}}(n) = 1 + C_{\text{nrev}}(n - 1) + n \quad \text{if } n > 0
  \]

- Cost of \text{app} \( \rightarrow \) closed form: \( C_{\text{app}}(n, m) = n + 1 \) for \( n \geq 0 \).
  \[
  C_{\text{app}}(0, m) = 1 \\
  C_{\text{app}}(n, m) = 1 + C_{\text{app}}(n - 1, m) \quad \text{if } n > 0
  \]

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