Towards Statically Reasoning About R Vectors

$$\mathcal{D}_{\mathcal{V}_{\mathcal{B}}^{\sharp}} \stackrel{\text{def}}{=} \left\langle \mathcal{V}_{\mathcal{B}}^{\sharp},\ \sqsubseteq_{\mathcal{V}_{\mathcal{B}}^{\sharp}},\ \sqcup_{\mathcal{V}_{\mathcal{B}}^{\sharp}},\ \sqcap_{\mathcal{V}_{\mathcal{B}}^{\sharp}},\ \bot_{\mathcal{V}_{\mathcal{B}}^{\sharp}},\ \top_{\mathcal{V}_{\mathcal{B}}^{\sharp}} \right\rangle$$

Manuel Di Agostino[1], Florian Sihler[2], Vincenzo Arceri[1], Oliver Gerstl[2], Matthias Tichy[2]

2026-10-07

[1] University of Parma, [2] Ulm University

manuel.diagostino@studenti.unipr.it

1. Motivation

1.1 Why R?

1.1 Why R?

UNIVERSITÀ DI PARMA

- Dominant in statistical computing
- Over 23,938 community packages[1]
- No type declarations. Every variable can hold any value at any time
- Interpreted, dynamically typed: errors surface only at runtime
- flowR[2]: data-flow, slicing and data-frame analysis[3], no vector abstraction

```
1 gdef <- function() {
2   x <<- "hello!"
3 }
4
5 x <- 3
6 gdef()
7 x + 1 # Runtime error
```

[1] CRAN. *Contributed Packages*. 2026. https://cran.r-project.org/web/packages/
[2] F. Sihler and M. Tichy. *Statically Analyzing the Dataflow of R Programs*. Proc. ACM Program. Lang. 9(OOPSLA2), 2025. https://doi.org/10.1145/3763087
[3] O. Gerstl. *Tracking the Shape of Data Frames in R Programs Using Abstract Interpretation*. Master's thesis, Universität Ulm, 2025. https://doi.org/10.18725/OPARU-58621

1.2 R's Data Model

1.2 R's Data Model

Everything is a vector:

- `TRUE`, *logical* of length 1
- `3` or `3.14`, *double* of length 1
- `c(1L, 2L, 3L)`, *integer* of length 3

1.2 R's Data Model

Everything is a vector:

- `TRUE`, *logical* of length 1
- `3` or `3.14`, *double* of length 1
- `c(1L, 2L, 3L)`, *integer* of length 3

So what about...?

- `c()`
- `"hello"`
- `c(TRUE, 3L, 2.5)`

1.2 R's Data Model

Everything is a vector:

- `TRUE`, *logical* of length 1
- `3` or `3.14`, *double* of length 1
- `c(1L, 2L, 3L)`, *integer* of length 3

So what about...?

- `c()`
- `"hello"`
- `c(TRUE, 3L, 2.5)`

Notation

In the following, we denote vectors as $[e_1, e_2, ..., e_n]$.

`c()` is equivalent to the reserved keyword `NULL`:

```
is.null(c()) # TRUE
length(c()) # 0
```

`c()` is equivalent to the reserved keyword `NULL`:

```
is.null(c()) # TRUE
length(c()) # 0
```

`"hello"` is a *character* vector of length 1:

```
length("hello") # 1
```

`c()` is equivalent to the reserved keyword `NULL`:

```
is.null(c()) # TRUE
length(c()) # 0
```

`"hello"` is a *character* vector of length 1:

```
length("hello") # 1
```

`c(TRUE, 3L, 2.5)` implicitly converted to a *double* vector:

```
typeof(c(TRUE, 3L, 2.5)) # "double"
```

Remark Type Hierarchy

Vectors elements must be homogeneous in their type:

$$\text{logical} < \text{integer} < \text{double} < \text{character}$$

Implicit coercion occurs:

```
c(TRUE, "hello")      # ["TRUE", "hello"]
```

Special logical value (*one per-type*): `NA`, representing missing data:

```
c(NA, "hello")        # [NA, "hello"]
```

1.3 Vector Operations

1.3.1 Recycling

UNIVERSITÀ DI PARMA

- R operations are element-wise across vectors.
- When two vectors have different lengths, the shorter is recycled to match the longer.

Example

```
c(1, 2, 3) + c(10, 20) # [10, 20] recycled to [10, 20, 10]
```

The resulting vector is **[11, 22, 13]**.

1.3.2 Attributes

UNIVERSITÀ DI PARMA

- Any R vector can carry extra metadata via *attributes*
- Most used are `dim`, `class` and `names`

Example From vector to matrix

```
x <- 1:10
dim(x) <- c(2, 5)
#      [,1] [,2] [,3] [,4] [,5]
# [1,]    1    3    5    7    9
# [2,]    2    4    6    8   10
```

1.3.3 Slicing

UNIVERSITÀ DI PARMA

- Three indexing modes: positive, negative, and logical
- Access selects by position, exclusion, or TRUE mask
- Positive access: *out-of-bounds* yields NA, index 0 contributes nothing

Example Access

```
v <- c(10, 20, 30, 40)
v[c(2, 0, 5)]           # [20, NA]
v[c(-1, -3)]            # [20, 40]
v[c(TRUE, FALSE)]       # [10, 30]
```

1.3.3 Slicing

UNIVERSITÀ DI PARMA

- Three indexing modes: positive, negative, and logical
- Access selects by position, exclusion, or TRUE mask
- Positive access: *out-of-bounds* yields NA, index 0 contributes nothing

Example Access

```
v <- c(10, 20, 30, 40)
v[c(2, 0, 5)]           # [20, NA]
v[c(-1, -3)]            # [20, 40]
v[c(TRUE, FALSE)]       # [10, 30]
```

- Update may grow the vector, padding with NA
- Logical update recycles the mask

Example Update

```
v <- c(10, 20, 30)
v[5] <- 99              # [10, 20, 30, NA, 99]
v[-2] <- 5              # [ 5, 20,  5,  5,  5]
v[c(TRUE, FALSE)] <- 0  # [ 0, 20,  0,  5,  0]
```

2. The μR Language

2.1 Concrete Syntax

μR is a subset of R focused on vector operations; it omits environments, promises, and lazy evaluation.

$$
\begin{array}{lcl}
P & ::= & S \\
S & ::= & E \\
 & | & \texttt{if}\,(E)\,\{S_1\}\,\texttt{else}\,\{S_2\} \\
 & | & \texttt{while}\,(E)\,\{S\} \\
 & | & S_1\,S_2 \\
 & | & \{\ S\ \} \\
\bowtie_{arith} & ::= & + \mid - \mid * \mid / \\
\bowtie_{rel} & ::= & > \mid < \mid == \mid \neq \mid \leqslant \mid \geqslant \\
\triangleright_{unary} & ::= & ! \mid -
\end{array}
$$

$$
\begin{array}{lcl}
E & ::= & Atm \mid \triangleright\, E \\
 & | & e_1 \bowtie_{arith} e_2 \\
 & | & e_1 \bowtie_{rel} e_2 \\
 & | & \texttt{c}() \\
 & | & \texttt{c}(E_1, \ldots, E_n) \\
 & | & x \leftarrow e \\
 & | & x[E_{idx}] \\
 & | & x[E_{idx}] \leftarrow e \\
 & | & \texttt{setAttr}(x,\ \{s_1, \ldots, s_n\}) \\
 & | & \texttt{getAttr}(x)
\end{array}
$$

Table 1: Syntax of μR (subset of the full grammar).

$$Atm ::= Var \mid Float \mid Int \mid Bool \mid \texttt{NULL}$$

Definition Concrete Vector

Given the set of atomic values $\mathbb{V}_{Atm}$ and attribute names $\mathcal{A}$, a concrete vector $v \in \mathcal{V}$ is a pair $([e_1, ..., e_n], a) \in (\mathbb{N} \to \mathbb{V}_{Atm}) \times \mathcal{A}$: a total map from indices to atomic values, plus a set of attribute names.

Notation

$$([e_1, ..., e_n], \mathcal{A}) \equiv [e_1, ..., e_n]_{\mathcal{A}}$$

$$([e_1, ..., e_n], \varnothing) \equiv [e_1, ..., e_n]$$

$\mathbb{E}[\![e]\!]: \mathcal{E} \to (\mathcal{E} \times (\mathcal{V} \cup \mathcal{P}(\text{String})))$, where $x \in \mathit{Var}, e \in E$

$$\mathbb{E}[\![e_1 \diamond e_2]\!]\rho \stackrel{\text{def}}{=} (\rho_2, v_1' \diamond v_2') \text{ where } (\rho_1, v_1) = \mathbb{E}[\![e_1]\!]\rho, (\rho_2, v_2) = \mathbb{E}[\![e_2]\!]\rho_1, (v_1', v_2') = \text{Recycle}((v_1, v_2)), \diamond \in \{\bowtie_{arith}, \bowtie_{rel}\}$$

$$\mathbb{E}[\![\text{c}()]\!]\rho \stackrel{\text{def}}{=} (\rho, [\,])$$

$$\mathbb{E}[\![\text{c}(E_1, \ldots, E_n)]\!]\rho_0 \stackrel{\text{def}}{=} (\rho_n, v_1 \oplus^\sharp \cdots \oplus^\sharp v_n) \text{ where } \forall i \in [1, n].(\rho_i, v_i) = \mathbb{E}[\![e_i]\!]\rho_{i-1}$$

$$\mathbb{E}[\![x[e_{idx}]]\!]\rho \stackrel{\text{def}}{=} (\rho', \text{Select}(\rho'(x), v_{\text{idx}})) \text{ where } (\rho', v_{\text{idx}}) = \mathbb{E}[\![e_{idx}]\!]\rho$$

Table 2: Selected rules of the concrete semantics.

3. The Abstract Domain

Definition Raw Abstract Element

A *raw abstract vector* in $\tilde{\mathcal{V}}^{\sharp}_{\mathcal{B}}$ is a 4-tuple

$$v = \left([l, u], \langle p_1 \cdots p_k \rangle, s, a\right) \in \mathsf{Itv}_0 \times \mathcal{B}^{\sharp *} \times \mathcal{B}^{\sharp} \times \mathsf{Attr}^{\sharp}$$

with the invariant:

$$l \leq k \leq u \wedge u \neq +\infty \Rightarrow s = \bot_{\mathcal{B}^{\sharp}} \wedge u = k$$

For example, setting $\mathcal{B}^{\sharp} = \mathsf{Itv}$:

- $\left([1, 2], \langle [1, 1][0, 0] \rangle, \bot_{\mathsf{Itv}}, \bot_{\mathsf{Attr}^{\sharp}}\right) \overset{\gamma_{\mathcal{V}^{\sharp}_{\mathsf{Itv}}}}{\longmapsto} \{[1], [1, 0]\}$
- $\left([1, +\infty], \langle [42, 42] \rangle, [42, 42], \bot_{\mathsf{Attr}^{\sharp}}\right) \overset{\gamma_{\mathcal{V}^{\sharp}_{\mathsf{Itv}}}}{\longmapsto} \{[42], [42, 42], [42, 42, 42], \ldots\}$
- $\left([1, +\infty], \langle [42, 42][42, 42] \rangle, [42, 42], \bot_{\mathsf{Attr}^{\sharp}}\right) \overset{\gamma_{\mathcal{V}^{\sharp}_{\mathsf{Itv}}}}{\longmapsto} \{[42], [42, 42], [42, 42, 42], \ldots\}$

Definition Raw Abstract Element

A *raw abstract vector* in $\tilde{\mathcal{V}}^{\sharp}_{\mathcal{B}}$ is a 4-tuple

$$v = \left([l, u], \langle p_1 \cdots p_k \rangle, s, a\right) \in \mathsf{Itv}_0 \times \mathcal{B}^{\sharp *} \times \mathcal{B}^{\sharp} \times \mathsf{Attr}^{\sharp}$$

with the invariant:

$$l \leq k \leq u \wedge u \neq +\infty \Rightarrow s = \bot_{\mathcal{B}^{\sharp}} \wedge u = k$$

For example, setting $\mathcal{B}^{\sharp} = \mathsf{Itv}$:

- $\left([1,2], \langle [1,1][0,0] \rangle, \bot_{\mathsf{Itv}}, \bot_{\mathsf{Attr}^{\sharp}}\right) \overset{\gamma_{\mathcal{V}^{\sharp}_{\mathsf{Itv}}}}{\longmapsto} \{[1], [1,0]\}$
- $\left([1,+\infty], \langle [42,42] \rangle, [42,42], \bot_{\mathsf{Attr}^{\sharp}}\right) \overset{\gamma_{\mathcal{V}^{\sharp}_{\mathsf{Itv}}}}{\longmapsto} \{[42], [42,42], [42,42,42], \ldots\}$
- $\left([1,+\infty], \langle [42,42][42,42] \rangle, [42,42], \bot_{\mathsf{Attr}^{\sharp}}\right) \overset{\gamma_{\mathcal{V}^{\sharp}_{\mathsf{Itv}}}}{\longmapsto} \{[42], [42,42], [42,42,42], \ldots\}$

Definition Abstract Domain of R Vectors

The abstract domain is the complete lattice:

$$\mathcal{D}_{\mathcal{V}^\sharp_{\mathcal{B}}} \stackrel{\text{def}}{=} \left\langle \mathcal{V}^\sharp_{\mathcal{B}},\ \sqsubseteq_{\mathcal{V}^\sharp_{\mathcal{B}}},\ \sqcup_{\mathcal{V}^\sharp_{\mathcal{B}}},\ \sqcap_{\mathcal{V}^\sharp_{\mathcal{B}}},\ \bot_{\mathcal{V}^\sharp_{\mathcal{B}}},\ \top_{\mathcal{V}^\sharp_{\mathcal{B}}} \right\rangle$$

where $\mathcal{V}^\sharp_{\mathcal{B}} = \left\{ [v]_{\equiv} \;\middle|\; v \in \tilde{\mathcal{V}}^\sharp_{\mathcal{B}} \right\} \cup \left\{ \bot_{\mathcal{V}^\sharp_{\mathcal{B}}}, \top_{\mathcal{V}^\sharp_{\mathcal{B}}} \right\}$ and:

- $\bot_{\mathcal{V}^\sharp_{\mathcal{B}}} = \left(\bot_{\mathsf{Itv}_0}, \varepsilon, \bot_{\mathcal{B}^\sharp}, \bot_{\mathsf{Attr}^\sharp} \right)$
- $\top_{\mathcal{V}^\sharp_{\mathcal{B}}} = \left(\top_{\mathsf{Itv}_0}, \varepsilon, \top_{\mathcal{B}^\sharp}, \top_{\mathsf{Attr}^\sharp} \right)$

Definition

The concretization function $\gamma_{\mathcal{V}^\sharp_\mathcal{B}} : \mathcal{V}^\sharp_\mathcal{B} \to \mathcal{P}(\mathcal{V})$ is defined as:

$$\gamma_{\mathcal{V}^\sharp_\mathcal{B}}(([l,u],\boldsymbol{p},s,a)) \stackrel{\text{def}}{=} \left\{ [c_1,\ldots,c_n]_\mathcal{A} \in \mathcal{V} \;\middle|\; \begin{array}{l} n \in \gamma_{\mathsf{Itv}_0}([l,u]),\ \mathcal{A} \in \gamma_{\mathsf{Attr}^\sharp}(a), \\ \forall i \in [1,\min(n,|\boldsymbol{p}|)] : c_i \in \gamma_{\mathcal{B}^\sharp}(p_i), \\ \forall j \in [|\boldsymbol{p}|+1,n] : c_j \in \gamma_{\mathcal{B}^\sharp}(s) \end{array} \right\}$$

4. Examples of Abstract Operators

UNIVERSITÀ DI PARMA

$$v_i = \left([l_i, u_i], \left\langle p_{i1} \cdots p_{i\,|\boldsymbol{p}_i|} \right\rangle, s_i, a_i \right)$$

$$\text{Recycle}^\sharp : \mathcal{V}_{\mathcal{B}}^\sharp \times \mathcal{V}_{\mathcal{B}}^\sharp \to \mathcal{V}_{\mathcal{B}}^\sharp \times \mathcal{V}_{\mathcal{B}}^\sharp$$

The resulting recycled length is $[l', u'] = \left[\max(l_1, l_2), \max(u_1, u_2)\right]$.

$$v_i' = \left([l', u'], \bigsqcup_{d=l'}^{\max(|\boldsymbol{p}_1|, |\boldsymbol{p}_2|)} {}_{\mathcal{B}^{\sharp *}} \; \rho_f^\sharp(\boldsymbol{p}_i, s_i, l_i, d),\ s_i',\ a_i \right)$$

Two sources of uncertainty for the known content:

- the target recycled length d lives in $\gamma_{\text{Itv}}([l', \max(|\boldsymbol{p}_1|, |\boldsymbol{p}_2|)])$
  ↳ *solved joining over* $\rho_f^\sharp$
- given a target length d, the original v_i may abstract multiple lengths in $[l_i, u_i]$
  ↳ *solved joining over* $\rho_c^\sharp$

Example

Let $\boldsymbol{p} = \langle[1,1][2,2]\rangle$. Then:

$$\rho^{\sharp}_{f}(\boldsymbol{p}, \bot_{\mathsf{ltv}}, 1, 3) = \rho^{\sharp}_{c}(\boldsymbol{p}, 1, 3) \sqcup_{\mathsf{ltv}^*} \rho^{\sharp}_{c}(\boldsymbol{p}, 2, 3) \sqcup_{\mathsf{ltv}^*} \rho^{\sharp}_{c}(\boldsymbol{p}, 3, 3)$$

$$= \langle[1,1][1,1][1,1]\rangle \sqcup_{\mathsf{ltv}^*} \langle[1,1][2,2][1,1]\rangle = \langle[1,1][1,2][1,1]\rangle$$

4.2 Abstract Selection

$$\text{Select}^\sharp : \mathcal{V}^\sharp_{\mathcal{B}} \times \mathcal{V}^\sharp_{\text{ltv}} \to \mathcal{V}^\sharp_{\mathcal{B}}$$

The concrete **Select** handles three indexing modes. The abstract operator **Select**$^\sharp$ analyzes each mode separately:

Positive $\text{Select}^\sharp_{\text{pos}}$

Indices ≥ 0 or NA. Extracts elements by position.

Negative $\text{Select}^\sharp_{\text{neg}}$

Indices ≤ -0. Excludes elements by position.

Logical $\text{Select}^\sharp_{\text{log}}$

Mask $\in \{\text{NA}, \text{True}, \text{False}\}^+$. Selects where mask is true.

$v_2 = ([l_2, u_2], \boldsymbol{p}_2, s_2, a_2)$

The selector content $\boldsymbol{p}_2 \in \text{Itv}^*$ is partitioned by concretization sign:

$$\boldsymbol{p}_2^+ = \langle p \in \boldsymbol{p}_2 \mid \gamma_{\text{Itv}}(p) \subseteq \mathbb{Z}_{\geq 0} \cup \{\text{NA}\} \rangle$$

$$\boldsymbol{p}_2^- = \langle p \in \boldsymbol{p}_2 \mid \gamma_{\text{Itv}}(p) \subseteq \mathbb{Z}_{\leq 0} \rangle$$

The overall operator joins the two integer-indexing branches:

$$\text{Select}^\sharp(v_1, v_2) = \text{Select}^\sharp_{\text{pos}}(v_1, v_2^+) \sqcup_{\mathcal{V}^\sharp_{\mathcal{B}}} \text{Select}^\sharp_{\text{neg}}(v_1, v_2^-)$$

5. Implementation & Validation

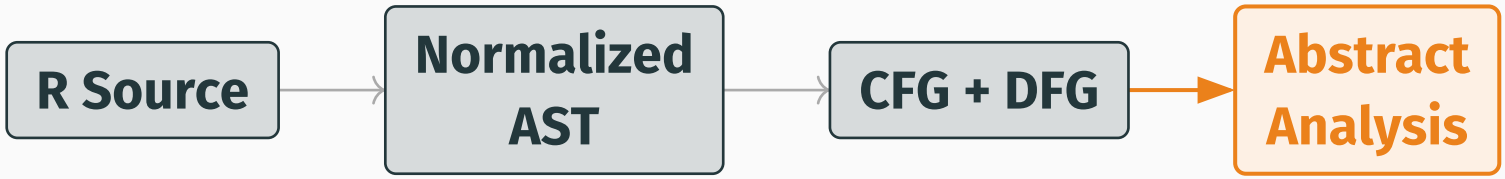
5.1 Analysis Pipeline

61 handcrafted test programs, covering 6 categories:

- *Creation*, *Recycling*, *Selection*, *Update*, *Loops*, *Unbounded Vectors*

Domain integrated[1] in `flowR`[2]:

- Worklist fixpoint algorithm, join at merge points, widening $\nabla_{\mathcal{V}_{\mathcal{B}}^{\sharp}}$ at loops
- Dispatches each CFG vertex to the corresponding abstract operator



[1] Available at https://github.com/manueldiagostino/flowr/tree/r-vectors
[2] F. Sihler and M. Tichy. *Statically Analyzing the Dataflow of R Programs*. Proc. ACM Program. Lang. 9(OOPSLA2), 2025. https://doi.org/10.1145/3763087

5.2 Analysis Example

```
1 v <- 1
  ↳ v ≔ ([1, 1], ⟨[1, 1]⟩, ⊥ltv)
2 if (*) {
3   v <- c(1, 2, 3)
4 } else {
5   v <- c(0, 1)
6 }
  ↳ v ≔ ([2, 3], ⟨[0, 1][1, 2][3, 3]⟩, ⊥ltv)
7 r <- v + c(9, 10, 11)
  ↳ r ≔ ([3, 3], ⟨[9, 10][11, 12][11, 14]⟩, ⊥ltv)
```

Figure 1: An example from the handcrafted test-suite.

6. Conclusion

6. Conclusion

UNIVERSITÀ DI PARMA

Slides available

Contributions:

- R vector semantics formalization → the μR chore calculus
- $\mathcal{V}^{\sharp}_{\mathcal{B}}$: a parametric abstract domain → lattice, widening, abstract operators
- Proof-of-concept in `flowR` → 61 test cases validated

Limitations & Future Work:

- Not modeled: lazy evaluation, metaprogramming, S3/S4 dispatch
- Future: relational domain, real R packages, full-language analysis