RHL with arrays

June 11, 2026

Contents

1 graded hoare logic with arrays (no beta)

1.1 Syntax

Arithmetic Expression	e	::=	$n \mid i \mid x \mid e_1 \uplus e_2 \mid len(u_a) \mid u_a[e] \mid$
Boolean Expression	b	::=	$\texttt{true} \mid \texttt{false} \mid \neg b \mid b_1 \wedge b_2 \mid e_1 \oplus e_2$
Binary Operation	$\oplus$	::=	$(< \mid > \mid = \mid \leq \mid \geq)$
Arithmetic Operation	$\uplus$	::=	$(+ \mid - \mid \times)$
Command	C	::=	$x := e \mid \texttt{while}\ b\ \texttt{do}\ C \mid C_1; C_2$
			$\mid \texttt{if}\ b\ \texttt{then}\ C_1\ \texttt{else}\ C_2 \mid \texttt{skip} \mid$
			$u_a[e_1] := e_2 \mid x := u_a[e] \mid u_a \leftarrow array(e_1)(e_2)$
Array Variables	u_a	=	Var_a
state	s	=	$Var \rightarrow_{fin} \mathbb{Z}$
heap	h	=	$Var_a \rightarrow_{fin} (\mathbb{Z} \rightarrow \mathbb{Z})$
Combined state	Σ	=	$state \times heap$
Grade	m	::=	$n \mid i \mid max(m_1, m_2) \mid m_1 + m_2 \mid m_1 \times m_2 \mid \lvert\beta\rvert \mid \sum_{k=a}^{b} m_k$

Logical Variable	i	$\in$	**LVar**
Sort	**S**	::=	$\mathbb{Z}$
Index	J	::=	$n \mid i \mid max(J_1, J_2) \mid J_1 \uplus J_2$
Unary Assertion	p, q	::=	**True** $\mid$ **False** $\mid e_1 \oplus e_2 \mid$
			$\mid \neg p \mid p_1 \wedge p_2 \mid p_1 \vee p_2 \mid p_1 \implies p_2 \mid \forall i :: S.p \mid \exists i :: S.p$

$\mathbb{N}$ for natural numbers, $K \subseteq \mathbb{N}$, $\mathbb{Z}$ for integers, $\mathbb{P}$ for set of natural numbers. i is the logical variable for integers, and the logical variable j is used for set of natural numbers.

1.2 Unary Logic

We have the logic rule of the form: $\vdash^m \{p\}\ C\ \{q\}$.

$$\frac{}{\vdash^1 \{e_1 = n\}\ u_a \leftarrow \mathit{array}(e_1)(e_2)\ \{\forall k.0 \le k < n.u_a[k] = e_2\}}\ \textbf{arrayAlloc}$$

$$\frac{}{\vdash^1 \{q[u_a[e_1 \mapsto e_2]/u_a]\}\ u_a[e_1] := e_2\ \{q\}}\ \textbf{arrayUpdt}$$

$$\frac{}{\vdash^1 \{q[u_a[e]/x]\}\ x := u_a[e]\ \{q\}}\ \textbf{arrayRead} \qquad \frac{}{\vdash^0 \{p\}\ \texttt{skip}\ \{p\}}\ \textbf{skip}$$

$$\frac{}{\vdash^1 \{q[e/x]\}\ x := e\ \{q\}}\ \textbf{assign}$$

$$\frac{\vdash^{m_1} \{p \wedge b\}\ C_1\ \{q\} \qquad \vdash^{m_2} \{p \wedge \neg b\}\ C_2\ \{q\}}{\vdash^{\max(m_1,m_2)} \{p\}\ \ \texttt{if}\ b\ \texttt{then}\ C_1\ \texttt{else}\ C_2\ \{q\}}\ \textbf{conditional}$$

$$\frac{\vdash^{m'} \{p'\}\ C\ \{q'\} \qquad \vDash p \Rightarrow p' \qquad \vDash q' \Rightarrow q \qquad p \vdash m' \le m}{\vdash^{m} \{p\}\ C\ \{q\}}\ \textbf{conseq}$$

$$\frac{\vdash^{m_1} \{p\}\ C_1\ \{r\} \qquad \vdash^{m_2} \{r\}\ C_2\ \{q\}}{\vdash^{m_1+m_2} \{p\}\ C_1; C_2\ \{q\}}\ \textbf{sequence}$$

$$\frac{\begin{array}{c} n \ge 0 \qquad k \notin FV(p) \cup FV(b) \cup FV(e_v) \\ \vDash p \implies 0 \le e_v \\ \vDash p \implies (e_v \le 0 \iff \neg b) \\ \forall k.\ 1 \le k \le n \Rightarrow 0 \le m_k \\ \forall k.\ 1 \le k \le n \Rightarrow \vdash^{m_k} \{p \wedge b \wedge e_v = k\}\ C\ \{p \wedge e_v < k\} \end{array}}{\vdash^{\sum_{k=1}^{n} m_k} \{p \wedge e_v \le n\}\ \ \texttt{while}\ b\ \texttt{do}\ C\ \{p \wedge \neg b\}}\ \textbf{awhile}$$

$$\frac{\vdash^{m} \{p\}\ C\ \{q\} \qquad \vdash^{m} \{r\}\ C\ \{q\}}{\vdash^{m} \{p \vee r\}\ C\ \{q\}}\ \textbf{disjunction-precond}$$

Conventions for this preliminary section. This no-β fragment proves *partial correctness* only. Allocation, array read, and array update are *partial* commands: an out-of-bounds index, or a non-fresh allocation target $u_a \in \mathit{dom}(h)$, has no transition (Section 2), so the array rules above need no explicit in-bounds or freshness premise and make no memory-safety or termination claim. In particular **arrayAlloc** is the weak, frame-free allocation rule; the stronger framed rule that preserves an arbitrary assertion or effect (side condition $u_a \notin \mathit{arrFV}(\cdot)$) appears in Section 6. Grades denote natural numbers under the interpretation I: n and a logical variable i denote n and $I(i)$; $+$, $\times$, max are the usual operations on $\mathbb{N}$; $|\beta|$ is the cardinality of a finite interpretation-level index set (here β ranges over such sets, e.g. intervals $[a, b]$ and the position sets K_+ of the examples); and $\sum_{k=a}^{b} m_k$ (with natural-number bounds a, b) denotes $\sum_{r=a}^{b} [\![m_k]\!]_i(I[k \mapsto r])$ (the empty range, $b < a$, gives 0). The grade comparison $m' \le m$ in **conseq** compares these denotations. The closed-form grades in the examples below

(e.g. $1 + 2n - |K_+ \cap [1, n]|$) are the natural-number *values* of the corresponding finite-sum grades $1 + \sum_{k=1}^{n} m_k$; subtraction occurs only in evaluating such a sum, never inside the grade language. Finally, i, x, and c in the programs are *program* variables; the logical index variables of the grade and index language are distinct and do not occur inside commands.

Audit note. Under the current cost model, `skip` is cost-free, scalar assignment and array update each cost 1, and guard evaluation itself is not charged unless it is written as an explicit read command. The examples below are therefore normalized for this preliminary no-β syntax; examples that first execute $x := a[i]$ pay one additional read cost in the loop body.

1.3 Example

1.3.1 p1: simple if

Think about the following program $p1$:

```
p1 =
        i = n ;
        if (a[i] > 0)
          then skip;skip
          else a[i] = 0
```

The grade is 2: the assignment $i := n$ costs 1, and the worst branch of the conditional is the array update, which costs 1.

$$\vdash^2 \{len(a) > n \geq 0\}\ p1\ \{len(a) > n \geq 0 \wedge a[n] \geq 0\}$$

If we know that $a[n] > 0$, the update branch is unreachable and the grade is 1 for the scalar assignment alone. If instead $a[n] < 0$, the else branch is forced and the grade remains 2.

$$\vdash^1 \{len(a) > n \geq 0 \wedge a[n] > 0\}\ p1\ \{len(a) > n \geq 0 \wedge a[n] \geq 0\}$$

1.3.2 p2: while if

```
p2 =                        { len(a) > n >=0   }
  i = n ;                   {i = n /\ len(a) > n >=0   }
  while (i>0) do            {forall j.i<j <= n. a[j]>=0} = I
      if (a[i] > 0)
          then skip;skip
          else a[i] = 0 ;
      i = i -1
                              {forall j.0<j <= n. a[j]>=0}
```

The initialization costs 1. Each loop body costs at most 2: the then branch executes only the decrement $i := i - 1$, while the else branch executes one array update and the decrement. Thus the no-information grade is:

$$\vdash^{1+2n} \{len(a) > n \geq 0\}\ p2\ \{\forall j.0 < j <= n.a[j] >= 0\}$$

If we know that some fixed positions are positive, for example $A_+ = a[2] > 0 \wedge a[3] > 0 \wedge a[6] > 0 \wedge a[8] > 0$, then those iterations take the cheaper then branch. When $n \geq 8$, the loop grade improves by one for each known-positive visited position:

$$\vdash^{1+2n-4} \{len(a) > n \geq 8 \wedge A_+\}\ p2\ \{\forall j.0 < j <= n.a[j] >= 0\}$$

In general, if K_+ is the set of positions known to satisfy $a[k] > 0$, then $m_k = 1$ for $k \in K_+ \cap [1, n]$ and $m_k = 2$ otherwise, giving grade $1 + 2n - |K_+ \cap [1, n]|$.
The β-tracking version of this example (using $u_a \mapsto^{\phi} \beta$) appears in Section 2.3.

2 GAAHLogic

2.1 Syntax

Arithmetic Expression	e	::=	$n \mid i \mid x \mid e_1 \uplus e_2 \mid len(u_a) \mid u_a[e] \mid$
Boolean Expression	b	::=	$\texttt{true} \mid \texttt{false} \mid \neg b \mid b_1 \wedge b_2 \mid e_1 \oplus e_2$
Binary Operation	$\oplus$	::=	$(< \mid > \mid = \mid \leq \mid \geq)$
Arithmetic Operation	$\uplus$	::=	$(+ \mid - \mid \times \mid \div)$
Command	C	::=	$x := e \mid \texttt{while}\ b\ \texttt{do}\ C \mid C_1; C_2$
			$\mid \texttt{if}\ b\ \texttt{then}\ C_1\ \texttt{else}\ C_2 \mid \texttt{skip} \mid$
			$u_a[e_1] := e_2 \mid x := u_a[e] \mid u_a \leftarrow array(e_1)(e_2)$
Array Variables	u_a	=	Var_a
state	s	=	$Var \rightarrow_{fin} \mathbb{Z}$
heap	h	=	$Var_a \rightarrow_{fin} (\mathbb{Z} \rightarrow \mathbb{Z})$
Combined state	Σ	=	$state \times heap$
Grade	m	::=	$n \mid i \mid m_1 \uplus m_2 \mid m_1 \times m_2 \mid max(m_1, m_2) \mid min(m_1, m_2) \mid \lvert\beta\rvert \mid \sum_{k=J_1}^{J_2} m_k$

Logical Variable	i, l	$\in$	**LVar**
Sort	$\mathbf{S}$	::=	$\mathbb{Z} \mid \mathbb{P}$
Integer Index	J	::=	$n \mid i \mid J_1 \uplus J_2 \mid J_1 \times J_2 \mid max(J_1, J_2) \mid min(J_1, J_2) \mid \lvert\beta\rvert$
Set Index	β	::=	$l \mid \beta_1 \cup \beta_2 \mid \beta_1 \cap \beta_2 \mid \beta_1 - \beta_2 \mid \emptyset \mid \{J\} \mid [J_1, J_2] \mid \beta \uplus J$
unary property	ϵ	::=	$emp \mid u_a \mapsto^{\phi} \beta \mid \epsilon_1 \star \epsilon_2$
Unary Assertion	p, q	::=	$\mathbf{True} \mid \mathbf{False} \mid e_1 \oplus e_2 \mid J \in \beta \mid \phi(e)$
			$\mid \neg p \mid p_1 \wedge p_2 \mid p_1 \vee p_2 \mid p_1 \implies p_2 \mid \forall i :: S.p \mid \exists i :: S.p$
Binary property	δ	::=	$emp \mid (u_a, u'_a) \mapsto^{\Phi} \beta \mid \delta_1 \star \delta_2$

$\mathbb{N}$ for natural numbers, $K \subseteq \mathbb{N}$, $\mathbb{Z}$ for integers, $\mathbb{P}$ for set of natural numbers. i is the logical variable for integers, and the logical variable j is used for set of natural numbers.

2.2 Semantics

2.2.1 Array update

In the semantics, we have an update operation for arrays which we denote as $u\{i \mapsto v\}$ and it is defined as follows.

$$\frac{}{\vdash u\{i \mapsto v_2\}[i] = v_2} \qquad \frac{i_1 \neq i}{\vdash u\{i_1 \mapsto v_2\}[i] = u[i]}$$

Interpretation I returns the values of the logical variables i, denoted as $I : \mathbf{LVar} \to \mathbf{S}$.

The denotation of e is a partial map, from combined state to integer numbers. $[\![e]\!]_i : \Sigma \times I \to S$

$$\begin{aligned}
[\![n]\!]_i((s,h),I) &= n \\
[\![x]\!]_i((s,h),I) &= s(x) \\
[\![i]\!]_i((s,h),I) &= I(i) \\
[\![len(u_a)]\!]_i((s,h),I) &= length(h(u_a)) \\
[\![u_a(e)]\!]_i((s,h),I) &= h(u_a)[[\![e]\!]_i((s,h),I)]
\end{aligned}$$

The boolean expression denotation: $[\![b]\!]_i : \Sigma \times I \to Bool$

$$\begin{aligned}
[\![true]\!]_i((s,h),I) &= true \\
[\![false]\!]_i((s,h),I) &= false \\
[\![e_1 \oplus e_2]\!]_i((s,h),I) &= [\![e_1]\!]_i((s,h),I) \oplus [\![e_2]\!]_i((s,h),I) \\
[\![\neg b]\!]_i((s,h),I) &= not\ [\![b]\!]_i((s,h),I) \\
[\![b_1 \wedge b_2]\!]_i((s,h),I) &= [\![b_1]\!]_i((s,h),I) \wedge [\![b_2]\!]_i((s,h),I)
\end{aligned}$$

$[\![J]\!]_j$: $I \to \mathbb{Z}$ (J is pure: depends only on interpretation I, not on state)

$$[\![n]\!]_j(I) = n$$

$$[\![i]\!]_j(I) = I(i)$$

$$[\![max(J_1, J_2)]\!]_j(I) = max([\![J_1]\!]_j(I), [\![J_2]\!]_j(I))$$

$$[\![J_1 \uplus J_2]\!]_j(I) = [\![J_1]\!]_j(I) \uplus [\![J_2]\!]_j(I)$$

$$[\![J_1 \times J_2]\!]_j(I) = [\![J_1]\!]_j(I) \times [\![J_2]\!]_j(I)$$

$$[\![|\beta|]\!]_j(I) = |[\![\beta]\!]_b(I)|$$

$[\![\beta]\!]_b$: $I \to \mathbb{P}$ (pure: depends only on interpretation I)

$$[\![l]\!]_b(I) = I(l)$$

$$[\![\beta_1 \cup \beta_2]\!]_b(I) = [\![\beta_1]\!]_b(I) \cup [\![\beta_2]\!]_b(I)$$

$$[\![\beta_1 \cap \beta_2]\!]_b(I) = [\![\beta_1]\!]_b(I) \cap [\![\beta_2]\!]_b(I)$$

$$[\![\beta_1 - \beta_2]\!]_b(I) = [\![\beta_1]\!]_b(I) - [\![\beta_2]\!]_b(I)$$

$$[\![\beta \uplus J]\!]_b(I) = \{n \uplus v \mid n \in [\![\beta]\!]_b(I) \wedge v = [\![J]\!]_j(I)\}$$

$$[\![\emptyset]\!]_b(I) = \{\}$$

$$[\![\{J\}]\!]_b(I) = \{[\![J]\!]_j(I)\}$$

$$[\![[J_1, J_2]]\!]_b(I) = \{n \mid [\![J_1]\!]_j(I) \leq n \leq [\![J_2]\!]_j(I)\}$$

The denotation of a grade is an integer-valued expression over the interpretation:

$$[\![m]\!]_i : I \to \mathbb{Z}.$$

$$[\![n]\!]_i(I) = n$$

$$[\![i]\!]_i(I) = I(i)$$

$$[\![m_1 \uplus m_2]\!]_i(I) = [\![m_1]\!]_i(I) \uplus [\![m_2]\!]_i(I)$$

$$[\![m_1 \times m_2]\!]_i(I) = [\![m_1]\!]_i(I) \times [\![m_2]\!]_i(I)$$

$$[\![max(m_1, m_2)]\!]_i(I) = max([\![m_1]\!]_i(I), [\![m_2]\!]_i(I))$$

$$[\![min(m_1, m_2)]\!]_i(I) = min([\![m_1]\!]_i(I), [\![m_2]\!]_i(I))$$

$$[\![|\beta|]\!]_i(I) = |[\![\beta]\!]_b(I)|$$

$$[\![\textstyle\sum_{k=J_1}^{J_2} m_k]\!]_i(I) = \sum_{r=[\![J_1]\!]_j(I)}^{[\![J_2]\!]_j(I)} [\![m_k]\!]_i(I[k \mapsto r])$$

For graded judgments, the interpretation is well formed only when the grade denotes a natural number: $[\![m]\!]_i(I) \in \mathbb{N}$. The cardinality form $|\beta|$ is defined only when $[\![\beta]\!]_b(I)$ is finite; otherwise the grade term is not well formed. A finite sum $\sum_{k=J_1}^{J_2} m_k$ uses the standard convention that an empty range ($[\![J_2]\!]_i(I) < [\![J_1]\!]_i(I)$) denotes 0. Grade comparisons are semantic: they compare the denotations of grade expressions under the current interpretation.

2.2.2 Assertion validation relation

We express the validation of assertion p as a relation $(s, h) \vDash_I p$, read as p is valid in combined store (s, h) under interpretation I. We write $(s, h) \nvDash_I p$

whenever $(s,h) \vDash_I p$ does not hold.

$$
\begin{array}{lll}
(s,h) \vDash_I \mathit{true} & & \mathit{always} \\
(s,h) \vDash_I e_1 \oplus e_2 & \mathit{if} & [\![e_1]\!]_i((s,h),I) \oplus [\![e_2]\!]_i((s,h),I) \\
(s,h) \vDash_I p_1 \wedge p_2 & \mathit{if} & (s,h) \vDash_I p_1 \ \mathit{and}\ (s,h) \vDash_I p_2 \\
(s,h) \vDash_I p_1 \vee p_2 & \mathit{if} & (s,h) \vDash_I p_1 \ \mathit{or}\ (s,h) \vDash_I p_2 \\
(s,h) \vDash_I p_1 \implies p_2 & \mathit{if} & (s,h) \nvDash_I p_1 \ \mathit{or}\ (s,h) \vDash_I p_2 \\
(s,h) \vDash_I \neg p & \mathit{if} & (s,h) \nvDash_I p \\
(s,h) \vDash_I \forall i :: L.p & \mathit{if} & \forall k \in \mathsf{L}.(s,h) \vDash_{I[i \to k]} p \\
(s,h) \vDash_I \exists i :: L.p & \mathit{if} & \exists k \in \mathsf{L}.(s,h) \vDash_{I[i \to k]} p \\
(s,h) \vDash_I J \in \beta & \mathit{if} & [\![J]\!]_j(I) \in [\![\beta]\!]_b(I) \\
(s,h) \vDash_I \mathbf{False} & & \mathit{never} \\
(s,h) \vDash_I \phi(e) & \mathit{if} & \phi([\![e]\!]_i((s,h),I))
\end{array}
$$

Here ϕ denotes the fixed unary value predicate ($\phi : \mathbb{Z} \to \mathit{Bool}$) parameterizing the ambient effect annotation; it is a rule/derivation parameter, not quantified over inside the assertion language.

We say an assertion p is valid, written $\vDash p$ if it is valid in any combined state, under any interpretation: $\forall s, h, I.(s,h) \vDash_I p$.

Two heap h_1 and h_2 are disjoint, denoted as $h_1 \perp h_2$, defined as $h_1 \perp h_2 = \mathit{dom}(h_1) \cap \mathit{dom}(h_2) = \emptyset$.

We express the validation of effect ϵ as a relation $(s,h) \vDash_I \epsilon$, read as ϵ is valid under state s and heap h under interpretation I. We write $(s,h) \nvDash_I \epsilon$ whenever $(s,h) \vDash_I \epsilon$ does not hold.

$$
\begin{array}{lll}
(s,h) \vDash_I \mathit{emp} & \mathit{if} & h = [] \\
(s,h) \vDash_I u_a \mapsto^{\phi} \beta & \mathit{if} & u_a \in \mathit{dom}(h) \wedge [\![\beta]\!]_b(I) \subseteq [0, \mathit{length}(h(u_a))) \\
 & & \wedge \forall x \in [\![\beta]\!]_b(I).\phi(h(u_a)[x]) \mathit{must} \\
(s,h) \vDash_I \epsilon_1 \star \epsilon_2 & \mathit{if} & \exists h_1, h_2.h_1 \perp h_2 \wedge h = h_1 + h_2 \ \mathit{and}\ (s,h_1) \vDash_I \epsilon_1 \ \mathit{and}\ (s,h_2) \vDash_I \epsilon_2
\end{array}
$$

$$\phi : \mathbb{Z} \to \mathit{Bool}$$

We say an effect ϵ is valid, written $\vDash \epsilon$, if it holds in any state under any interpretation: $\forall s, h, I.\ (s,h) \vDash_I \epsilon$.

2.2.3 Command interpretation

Logical variables are not allowed to appear in object-language commands. Hence for a command expression e in a well-formed command, $[\![e]\!]_i((s,h),I)$ is independent of the interpretation I; we write $[\![e]\!]_e(s,h)$ for this common value. Similarly, $[\![b]\!](s,h)$ abbreviates the interpretation-independent boolean denotation used by commands.

The command denotation is set-valued: $[\![C]\!] : \Sigma \to \mathcal{P}(\Sigma \times \mathbb{N})$. Each element is a final combined state paired with the accumulated cost.

Cost model (per-construct parameterization, ARel/RelCost-style). The semantics is *cost-instrumented*: each command evaluation produces a final state and a nonnegative integer cost. Following the parameterized cost-model approach of RelCost [1] and ARel [2], each language construct carries its own cost weight $c_{construct} \in \mathbb{N}$, and the total cost of an execution is the sum of the construct weights along the trace.

Default weights (used for the examples in this paper):

- $c_{\texttt{skip}} = 0$: skip is cost-free.
- $c_{assign} = 1$: scalar assignment $x := e$ (including increment-style $x := x + 1$).
- $c_{read} = 1$: array read $x := u_a[e]$.
- $c_{update} = 1$: array write $u_a[e_1] := e_2$.
- $c_{alloc} = 1$: array allocation.
- $c_{if} = 0$: conditional control flow (cost of taken branch only).
- $c_{while} = 0$ per iteration of loop control overhead (cost of body only).

The framework is parametric in these weights; other defaults are permissible as long as the rule-compatibility conditions are met (Section 8.9). The default choice above makes cost-relevant differences appear at the level of statement types, mirroring ARel's per-construct cost analysis. In particular, $c_{\texttt{skip}} = 0 < c_{assign} = 1$ means a one-sided rule pairing an assignment on the *left* run with `skip` on the right gives a signed cost-difference $m_1 - m_2 = +1$, which the grade 1 accounts for; the mirror case (assignment on the right) gives $m_1 - m_2 = -1$, bounded by the grade 0 ($-1 \leq 0$, the least natural upper bound).

Ghost instrumentation (zero cost). A *ghost* variable is a meta-level annotation introduced solely to *observe* a quantity, for example the relational cost counter c used in the InplaceMap derivation (Section 8.11), and never influences control flow or any heap/array result. Ghost assignments such as $c := 0$ and $c := c + 1$ are *not* measured program constructs: they carry zero cost and are erased before the cost-instrumented semantics applies, so they do not contribute to the operational cost m. In particular they are *not* governed by $c_{assign} = 1$; only genuine program constructs (assignments to live program variables, reads, updates, allocations) carry the weights above. This convention is what lets an instrumented divergent step charge grade 1 (the real update versus `skip`) rather than 2.

$X \circ Y(s,h) \triangleq \{\big((s'',h''), m_1{+}m_2\big) | \exists s', h'.((s',h'),m_1) \in [\![Y]\!](s,h) \wedge ((s'',h''),m_2) \in [\![X]\!](s',h')\}$

$$[\![\texttt{skip}]\!](s,h) = \{\big((s,h), c_{\texttt{skip}}\big)\}$$

$$[\![C_1; C_2]\!](s,h) = [\![C_2]\!] \circ [\![C_1]\!](s,h)$$

$$[\![x := e]\!](s,h) = \{\big((s[x \mapsto v], h), c_{assign}\big) | v = [\![e]\!]_e(s,h)\}$$

$$[\![x := u_a[e]]\!](s,h) = \{\big((s[x \mapsto v], h), c_{read}\big) | i = [\![e]\!]_e(s,h) \wedge 0 \leq i <$$

$$\mathit{length}(h(u_a)) \wedge v = h(u_a)[i]\}$$

$$[\![u_a[e_1] := e_2]\!](s,h) = \{\big((s, h[u_a \mapsto h(u_a)\{i \mapsto v\}]), c_{update}\big) \mid i =$$

$$[\![e_1]\!]_e(s,h) \wedge 0 \leq i < \mathit{length}(h(u_a)) \wedge v = [\![e_2]\!]_e(s,h)\}$$

$$[\![u_a \leftarrow \mathit{array}(e_1)(e_2)]\!](s,h) = \{\big((s, h[u_a \mapsto [v_2, \ldots, v_2]]), c_{alloc}\big) \mid n =$$

$$[\![e_1]\!]_e(s,h) \wedge v_2 = [\![e_2]\!]_e(s,h) \wedge u_a \notin \mathit{dom}(h) \wedge \mathit{size}([v_2, \ldots, v_2]) = n\}$$

$$[\![\ \texttt{if}\ b\ \texttt{then}\ C_1\ \texttt{else}\ C_2]\!](s,h) = \{((s',h'), c_{if} + m) \mid [\![b]\!](s,h) =$$

$$\texttt{true} \wedge ((s',h'),m) \in [\![C_1]\!](s,h)\} \cup \{((s',h'), c_{if} + m) \mid [\![b]\!](s,h) =$$

$$\texttt{false} \wedge ((s',h'),m) \in [\![C_2]\!](s,h)\}$$

$$[\![\ \texttt{while}\ b\ \texttt{do}\ C]\!](s,h) = \{\big((s',h'),m\big) | \exists n. \mathrm{Iter}(n)(b)(C)(s,h)(s',h')m\}$$

Under the default weights ($c_{\texttt{skip}} = 0$, $c_{assign} = c_{read} = c_{update} = c_{alloc} = 1$, $c_{if} = c_{while} = 0$), the conditional cost equals the cost of the taken branch; the while loop cost is the sum of per-iteration body costs. Array reads and updates are *in-bounds*: when $i \notin [0, \mathit{length}(h(u_a)))$ the read/update command has no transition. This matches the effect-validity requirement $[\![\beta]\!]_b(I) \subseteq [0, \mathit{length}(h(u_a)))$ and the relational update rules' explicit index bound.

$\mathrm{Iter}(n)(b)(C)(s,h)(s',h')m$ is defined (with optional per-iteration overhead c_{while} added at each non-zero step; under default $c_{while} = 0$ this is invisible):

$$\mathrm{Iter}(0)(b)(C)(s,h)(s',h')0 = \neg[\![b]\!](s,h) \wedge (s,h) = (s',h')$$

$$\mathrm{Iter}(n+1)(b)(C)(s,h)(s',h')(c_{while} + m' + m) =$$

$$[\![b]\!](s,h) \wedge \exists (s'',h'').((s'',h''),m') \in [\![C]\!](s,h) \wedge \mathrm{Iter}(n)(b)(C)(s'',h'')(s',h')(m)$$

2.2.4 The order of the unary property

The order on unary effects is *semantic implication*, in the direction **conseq** requires:

$$(s,h) \vDash_I \epsilon_1 \leq \epsilon_2 \quad \textbf{iff} \quad \big((s,h) \vDash_I \epsilon_2 \implies (s,h) \vDash_I \epsilon_1\big).$$

$p \vdash \epsilon_1 \leq \epsilon_2$ is defined to be $\forall s, h, I.\ (s,h) \vDash_I p \implies (s,h) \vDash_I \epsilon_1 \leq \epsilon_2$, i.e. on every p-state, ϵ_2 implies ϵ_1.
For *map* effects this recovers β-weakening as an admissible lemma: $\{u \mapsto^\phi \beta_1\} \leq \{u \mapsto^\phi \beta_2\}$ holds whenever $[\![\beta_1]\!]_b(I) \subseteq [\![\beta_2]\!]_b(I)$ (the more-tracked property implies the less-tracked one), and $\leq$ is monotone under $\star$. Exact-empty *emp* is *not* below an arbitrary effect: $emp \leq \epsilon$ would require $\epsilon \implies emp$, which holds only for $\epsilon \equiv emp$. (The earlier unconditional $emp \leq \epsilon$ row and the purely syntactic *dom*-subset row are dropped: both are unsound for the implication direction **conseq** uses; no live derivation depends on the dropped $emp \leq \epsilon$ step.)

2.2.5 Free variables

We define FLV(p) and FLV(ϵ) for the free logical variables in assertion p and effect assertion ϵ.

We define FV(p) and FV(ϵ) for the free program variables in assertion p and effect assertion ϵ.

$FV(\epsilon)$

$$FV(emp) = \{\}$$
$$FV(\{u \mapsto^\phi \beta) = FV(\beta)$$
$$FV(\epsilon_1 \star \epsilon_2) = FV(\epsilon_1) \cup FV(\epsilon_2)$$

$FV(\beta)$

$$FV(l) = \{\}$$
$$FV(\beta_1 \cup \beta_2) = FV(\beta_1) \cup FV(\beta_2)$$
$$FV(\beta_1 \cap \beta_2) = FV(\beta_1) \cup FV(\beta_2)$$
$$FV(\beta_1 - \beta_2) = FV(\beta_1) \cup FV(\beta_2)$$
$$FV(\beta \uplus J) = FV(\beta) \cup FV(J)$$
$$FV(\emptyset) = \{\}$$
$$FV(\{J\}) = FV(J)$$
$$FV([J_1, J_2]) = FV(J_1) \cup FV(J_2)$$

$FV(J)$

$$FV(e) = FV(e)$$

$$FV(max(J_1, J_2)) = FV(J_1) \cup FV(J_2)$$

$$FV(J_1 \uplus J_2) = FV(J_1) \cup FV(J_2)$$

$$FV(|\beta|) = FV(\beta)$$

$FV(e)$

$$\begin{aligned} FV(n) &= \{\} \\ FV(x) &= \{x\} \\ FV(i) &= \{\} \\ FV(len(u_a)) &= \{\} \\ FV(u_a(e)) &= FV(e) \end{aligned}$$

Free variables of a binary property. The side conditions of the relational rules, in particular **heap-frame** (Section 6.3), constrain $FV(\delta)$, the free *scalar program* variables of a binary property δ (logical variables are tracked separately by FLV, and array names by $arrFV$). It collects the program variables occurring in the index sets of δ's atoms (well-formed value relations contribute none; see below):

$$FV(emp) = \emptyset, \qquad FV(\delta_1 \star \delta_2) = FV(\delta_1) \cup FV(\delta_2),$$
$$FV((u_1, u_2) \mapsto^{\Phi} \beta) = FV(\beta),$$

The value relation Φ contributes no scalar program variables: by well-formedness (Definition 6.3) it mentions only its two cell-value arguments and logical variables, so only β contributes to $FV(\delta)$. The atom heads u_1, u_2 are array variables, recorded by $arrFV(\delta)$ rather than $FV(\delta)$; thus the **heap-frame** side condition $FV(\delta) \cap (mod(C_1)\langle 1\rangle \cup mod(C_2)\langle 2\rangle) = \emptyset$ prevents the framed property from depending on a scalar variable the commands modify, while disjointness of the *array* footprint is handled separately by $arrFV(\delta)$ and the *arrfoot* side conditions.

We separately track array names mentioned by assertions, index sets, and binary properties, because FV intentionally records only scalar program variables. We write $arrFV(R)$ for the array variables read by an assertion R, and $arrFV(\delta)$ for the run-tagged array variables owned or read by a binary property.

The definition is syntactic:

$$
\begin{aligned}
arrFV(n) = arrFV(x) = arrFV(i) &= \emptyset,\\
arrFV(len(u_a)) &= \{u_a\},\\
arrFV(u_a(e)) &= \{u_a\} \cup arrFV(e),\\
arrFV(\emptyset) = \emptyset, \qquad arrFV(\{J\}) &= arrFV(J),\\
arrFV([J_1, J_2]) &= arrFV(J_1) \cup arrFV(J_2),\\
arrFV(\max(J_1, J_2)) &= arrFV(J_1) \cup arrFV(J_2),\\
arrFV(\min(J_1, J_2)) &= arrFV(J_1) \cup arrFV(J_2),\\
arrFV(J_1 \uplus J_2) &= arrFV(J_1) \cup arrFV(J_2),\\
arrFV(\beta_1 \cup \beta_2) &= arrFV(\beta_1) \cup arrFV(\beta_2),\\
arrFV(\beta_1 \cap \beta_2) &= arrFV(\beta_1) \cup arrFV(\beta_2),\\
arrFV(\beta_1 - \beta_2) &= arrFV(\beta_1) \cup arrFV(\beta_2),\\
arrFV(\beta \uplus J) &= arrFV(\beta) \cup arrFV(J),\\
arrFV(|\beta|) &= arrFV(\beta),\\
arrFV(\phi) &= \emptyset,\\
arrFV(\phi(e)) &= arrFV(e),\\
arrFV(emp) &= \emptyset,\\
arrFV(u_a \mapsto^{\phi} \beta) &= \{u_a\} \cup arrFV(\phi) \cup arrFV(\beta),\\
arrFV((u_1, u_2) \mapsto^{\Phi} \beta) &= \{u_1\langle 1\rangle, u_2\langle 2\rangle\} \cup arrFV(\Phi) \cup arrFV(\beta).
\end{aligned}
$$

For run-tagged syntax, the run tag is pushed to each array name: $arrFV(E\langle r\rangle) = \{u\langle r\rangle \mid u \in arrFV(E)\}$. The definition extends homomorphically over the remaining arithmetic expressions, Boolean assertions, logical connectives, relation annotations Φ, and $\star$.

We define the depending variables in commands as $dep(C)$:

$$
\begin{aligned}
dep(x := e) &= \{FV(e)\}\\
dep(C_1; C_2) &= dep(C_1) \cup dep(C_2)\\
dep(\texttt{ if } b \texttt{ then } C_1 \texttt{ else } C_2) &= FV(b) \cup dep(C_1) \cup dep(C_2)\\
dep(\texttt{ while } b \texttt{ do } C) &= FV(b) \cup dep(C)\\
dep(u_a[e_1] := e_2) &= \{FV(e_1)\} \cup \{FV(e_2)\}\\
dep(x := u_a[e]) &= FV(e)\\
dep(u_a \leftarrow array(e_1)(e_2)) &= FV(e_1) \cup FV(e_2)
\end{aligned}
$$

We define the modified variables in commands as $mod(C)$:

$$\begin{aligned}
mod(x := e) &= \{x\} \\
mod(C_1; C_2) &= mod(C_1) \cup mod(C_2) \\
mod(\texttt{ if } b \texttt{ then } C_1 \texttt{ else } C_2) &= mod(C_1) \cup mod(C_2) \\
mod(\texttt{ while } b \texttt{ do } C) &= mod(C) \\
mod(u_a[e_1] := e_2) &= \{\} \\
mod(x := u_a[e]) &= \{x\} \\
mod(u_a \leftarrow array(e_1)(e_2)) &= \{\}
\end{aligned}$$

We define the array variables modified by commands as $arrvars(C)$:

$$\begin{aligned}
arrvars(x := e) &= \emptyset \\
arrvars(C_1; C_2) &= arrvars(C_1) \cup arrvars(C_2) \\
arrvars(\texttt{ if } b \texttt{ then } C_1 \texttt{ else } C_2) &= arrvars(C_1) \cup arrvars(C_2) \\
arrvars(\texttt{ while } b \texttt{ do } C) &= arrvars(C) \\
arrvars(u_a[e_1] := e_2) &= \{u_a\} \\
arrvars(x := u_a[e]) &= \emptyset \\
arrvars(u_a \leftarrow array(e_1)(e_2)) &= \{u_a\}
\end{aligned}$$

We also use a conservative syntactic array footprint $arrfoot(C)$ for the arrays whose contents or identity may be inspected, updated, or rebound by a command:

$$\begin{aligned}
arrfoot(x := e) &= arrFV(e) \\
arrfoot(C_1; C_2) &= arrfoot(C_1) \cup arrfoot(C_2) \\
arrfoot(\texttt{ if } b \texttt{ then } C_1 \texttt{ else } C_2) &= arrFV(b) \cup arrfoot(C_1) \cup arrfoot(C_2) \\
arrfoot(\texttt{ while } b \texttt{ do } C) &= arrFV(b) \cup arrfoot(C) \\
arrfoot(u_a[e_1] := e_2) &= \{u_a\} \cup arrFV(e_1) \cup arrFV(e_2) \\
arrfoot(x := u_a[e]) &= \{u_a\} \cup arrFV(e) \\
arrfoot(u_a \leftarrow array(e_1)(e_2)) &= \{u_a\} \cup arrFV(e_1) \cup arrFV(e_2)
\end{aligned}$$

The distinction between *arrvars* and *arrfoot* separates preservation from locality. The set $arrvars(C)$ records arrays whose contents or identity may change; it is therefore sufficient for ordinary frame rules, where a framed assertion must remain true after the command. In contrast, **heap-frame** is a heap-locality rule: the command must not depend on the framed heap fragment at all. A read such as $x := u_a[e]$ does not modify u_a, so u_a is absent from *arrvars*, but it still inspects the heap cell of u_a at the evaluated index; hence u_a is included in *arrfoot*. The **heap-frame** side condition uses *arrfoot* to ensure that framed binary heap properties are disjoint from every array a command may inspect,

update, or rebind. The *arrFV* terms above account for array names appearing inside guards, indices, and value expressions.

The heap-frame rule (Section 6.3) uses *arrfoot* to confine a command's entire array footprint to the arrays *owned by the active heap property*. Because *arrfoot* already contains every array a command may inspect, update, rebind, *or allocate* (the allocation target u_a is in $\mathit{arrfoot}(u_a \leftarrow \mathit{array}(e_1)(e_2))$), requiring $\mathit{arrfoot}(C) \subseteq \mathit{own}(\delta_a)$ for the active property δ_a both keeps the command off the framed fragment and, as a deliberate side effect, forbids the framed body from allocating a *fresh* array that the frame does not already own. The read-only and in-place corpus (CountPositive, DualCount, InplaceMap) allocates nothing inside a heap-framed body, so this restriction costs nothing there; framing across a body that allocates a frame-complement array is a separate extension requiring a path-sensitive "allocated-before-first-use" refinement, which we do not pursue here.

This locality requirement is stronger than preservation on purpose. Ordinary frame permits the command to read framed data, because reading does not change the framed assertion. Heap-frame does not: if a command could inspect a framed heap cell, changing that framed cell would potentially change the command's control flow or the value it writes into the active heap. The locality side condition therefore says that the framed heap fragment is observationally irrelevant to the command execution, not merely unchanged by it. Because atoms are permissive, the rule additionally confines the command footprint to the active-owned arrays and requires the pre- and postconditions to be frame-local (Definitions 6.4, 6.5), so that on every witnessing split the command and the assertions touch only the active fragment and are insensitive to the framed one. Section 6.3 develops these notions.

2.3 Example: p2 with β-tracking

We revisit the program $p2$ from Section 1.3, now using the β-tracking judgment $\vdash^m \{p \mid \epsilon\} C \{q \mid \epsilon'\}$ to introduce an assertion $\epsilon = u_a \mapsto^{pos} \beta$ tracking the indices $i \in \beta$ where $u_a[i] > 0$ (the cells that take the cheap `skip` branch).

The quantified array assertion used below is an ordinary unary assertion; we write it in the desugared syntax of Section 2 as $\forall j :: \mathbb{Z}.\ ((0 < j \wedge j \leq n) \implies a[j] \geq 0)$, rather than as bounded-quantifier shorthand. Since the effect changes as the loop drops processed positive cells from β, this example uses the evolving while rule, not the fixed-effect rule. All intervals in the effect annotations are inclusive.

```
p2 =                          { len(a) > n >=0  }
    i = n ;                   {i = n /\ len(a) > n >=0  }
    while (i>0) do
        if (a[i] > 0)
            then skip;skip
            else a[i] = 0 ;
        i = i -1
```

$$
\begin{aligned}
p &\triangleq len(a) > n \wedge n \geq 0 \wedge 0 \leq i \leq n \wedge \forall j :: \mathbb{Z}.\ ((i < j \wedge j \leq n) \implies a[j] \geq 0),\\
e_v &\triangleq i,\\
b &\triangleq i > 0,\\
\epsilon_I(k) &\triangleq a \mapsto^{pos} (\beta - [k+1, n]),\\
m_k &\triangleq 2 - |(\beta - [k+1, n]) \cap \{k\}|.
\end{aligned}
$$

At the start of the iteration with $e_v = k$, the effect is $\epsilon_I(k) = a \mapsto^{pos} (\beta - [k+1, n])$: the already processed suffix $[k+1, n]$ has been removed from the tracked positive set. The body processes index k and then executes $i := i-1$, so the post-effect is $\epsilon_I(k-1) = a \mapsto^{pos} (\beta - [k, n])$. The evolving rule's biconditional prevents early exit while $i > 0$, hence the final effect is $\epsilon_I(0) = a \mapsto^{pos} (\beta - [1, n])$.

Thus the β-sensitive judgment is written:

$$
\begin{aligned}
&\vDash^{1+\sum_{k=1}^{n} m_k} \{len(a) > n \wedge n \geq 0 \mid a \mapsto^{pos} \beta\}\ p2\\
&\quad \{\forall j :: \mathbb{Z}.\ ((0 < j \wedge j \leq n) \implies a[j] \geq 0) \mid a \mapsto^{pos} (\beta - [1, n])\}.
\end{aligned}
$$

3 Unary Logic

The unary property ϕ of an array u is stored in heap precondition ϵ and heap post-condition ϵ' as $u \mapsto \beta$. Every item i in β means $\phi(u[i])$ holds. We have the logic rule of the form: $\vdash^m \{p \mid \epsilon\}\ C\ \{q \mid \epsilon'\}$.

$$\frac{\ell \text{ fresh logical index} \qquad u_a \notin arrFV(p) \cup arrFV(\epsilon)}{\vdash^1 \{p \wedge e_1 = \ell \wedge \ell \geq 0 \wedge \phi(e_2) \mid \epsilon\}\ u_a \leftarrow array(e_1)(e_2)\ \{p \mid \epsilon \star \{u_a \mapsto^\phi [0, \ell - 1]\}\}}\ \textbf{arrayAlloc1}$$

$$\frac{u_a \notin arrFV(p) \cup arrFV(\epsilon)}{\vdash^1 \{p \wedge e_1 \geq 0 \mid \epsilon\}\ u_a \leftarrow array(e_1)(e_2)\ \{p \mid \epsilon \star \{u_a \mapsto^\phi \emptyset\}\}}\ \textbf{arrayAlloc2}$$

$$\frac{j \text{ fresh}}{\vdash^1 \{q[u_a[e_1 \mapsto e_2]/u_a] \wedge \phi(e_2) \wedge e_1 = j \mid \epsilon \star \{u_a \mapsto^\phi \beta\}\}\ u_a[e_1] := e_2\ \{q \mid \epsilon \star \{u_a \mapsto^\phi \beta \cup \{j\}\}\}}\ \textbf{arrayUpdt1}$$

$$\frac{j \text{ fresh}}{\vdash^1 \{q[u_a[e_1 \mapsto e_2]/u_a] \wedge e_1 = j \mid \epsilon \star \{u_a \mapsto^\phi \beta\}\}\ u_a[e_1] := e_2\ \{q \mid \epsilon \star \{u_a \mapsto^\phi \beta - \{j\}\}\}}\ \textbf{arrayUpdt2}$$

$$\frac{j \text{ fresh}}{\vdash^1 \{q[u_a[e]/x] \wedge e = j \wedge j \in \beta \mid \epsilon \star \{u_a \mapsto^\phi \beta\}\}\ x := u_a[e]\ \{q \wedge \phi(x) \mid \epsilon \star \{u_a \mapsto^\phi \beta\}\}}\ \textbf{arrayRead1}$$

$$\frac{}{\vdash^1 \{q[u_a[e]/x] \mid \epsilon \star \{u_a \mapsto^\phi \beta\}\}\ x := u_a[e]\ \{q \mid \epsilon \star \{u_a \mapsto^\phi \beta\}\}}\ \textbf{arrayRead2}$$

$$\frac{}{\vdash^0 \{p \mid \epsilon\}\ \texttt{skip}\ \{p \mid \epsilon\}}\ \textbf{skip} \qquad \frac{x \notin FV(\epsilon)}{\vdash^1 \{q[e/x] \mid \epsilon[e/x]\}\ x := e\ \{q \mid \epsilon\}}\ \textbf{assign}$$

$$\frac{\vdash^m \{p \wedge b \mid \epsilon\}\ C_1\ \{q \mid \epsilon'\} \qquad \vdash^m \{p \wedge \neg b \mid \epsilon\}\ C_2\ \{q \mid \epsilon'\}}{\vdash^m \{p \mid \epsilon\}\ \ \texttt{if}\ b\ \texttt{then}\ C_1\ \texttt{else}\ C_2\ \{q \mid \epsilon'\}}\ \textbf{conditional}$$

$$\frac{\begin{array}{c}\vdash^{m'} \{p' \mid \epsilon_1'\}\ C\ \{q' \mid \epsilon_2'\} \qquad \vDash p \Rightarrow p' \qquad \vDash q' \Rightarrow q \\ p \vdash \epsilon_1' \leq \epsilon_1 \qquad q \vdash \epsilon_2 \leq \epsilon_2' \qquad p \vdash m' \leq m\end{array}}{\vdash^m \{p \mid \epsilon_1\}\ C\ \{q \mid \epsilon_2\}}\ \textbf{conseq}$$

$$\frac{\vdash^{m_1} \{p \mid \epsilon_1\}\ C_1\ \{r \mid \epsilon_2\} \qquad \vdash^{m_2} \{r \mid \epsilon_2\}\ C_2\ \{q \mid \epsilon_3\}}{\vdash^{m_1+m_2} \{p \mid \epsilon_1\}\ C_1; C_2\ \{q \mid \epsilon_3\}}\ \textbf{sequence}$$

$$\frac{\vdash^m \{p \mid \epsilon\}\ C\ \{q \mid \epsilon'\} \qquad \vdash^m \{r \mid \epsilon\}\ C\ \{q \mid \epsilon'\}}{\vdash^m \{p \vee r \mid \epsilon\}\ C\ \{q \mid \epsilon'\}}\ \textbf{disjunction-precond}$$

The allocation side condition $u_a \notin arrFV(p) \cup arrFV(\epsilon)$ is a freshness condition for specifications, not merely for heaps. Allocation rebinds the array name u_a to a fresh heap object. If the precondition assertion p or the incoming heap effect ϵ already mentioned u_a, the allocation step would invalidate those pre-state references instead of simply extending the heap with a new array component. The rule therefore applies when u_a is fresh for the specification being extended. Preservation of unrelated assertions is handled separately by the ordinary frame principle; heap-locality framing for heap effects uses the stronger *arrfoot* side condition above.

$$\frac{\begin{array}{c} n \geq 0 \qquad k \notin FV(p) \cup FV(b) \cup FV(e_v) \cup FV(\epsilon_I) \\ \vDash p \implies 0 \leq e_v \\ \vDash p \implies (e_v \leq 0 \iff \neg b) \\ \forall k.\ 1 \leq k \leq n \Rightarrow 0 \leq m_k \\ \forall k.\ 1 \leq k \leq n \Rightarrow \vdash^{m_k} \{p \wedge b \wedge e_v = k \wedge 0 \leq e_v \leq n \mid \epsilon_I\}\ C\ \{p \wedge e_v < k \mid \epsilon_I\} \end{array}}{\vdash^{\sum_{k=1}^{n} m_k} \{p \wedge e_v \leq n \mid \epsilon_I\}\ \texttt{while}\ b\ \texttt{do}\ C\ \{p \wedge \neg b \mid \epsilon_I\}}\ \textbf{awhile-rule}$$

$$\frac{\begin{array}{c} n \geq 0 \qquad k \notin FV(p) \cup FV(b) \cup FV(e_v) \qquad \vDash p \implies 0 \leq e_v \\ \vDash p \implies (e_v \leq 0 \iff \neg b) \\ \epsilon_I(k) \text{ is a meta-level indexed effect family} \\ \forall k.\ 1 \leq k \leq n \implies 0 \leq m_k \\ \forall k.\ 1 \leq k \leq n \implies \vdash^{m_k} \{p \wedge b \wedge e_v = k \mid \epsilon_I(k)\}\ C\ \{p \wedge e_v = k - 1 \mid \epsilon_I(k-1)\} \end{array}}{\vdash^{\sum_{k=1}^{n} m_k} \{p \wedge e_v = n \mid \epsilon_I(n)\}\ \texttt{while}\ b\ \texttt{do}\ C\ \{p \wedge \neg b \mid \epsilon_I(0)\}}\ \textbf{awhile-rule-evolving}$$

In the fixed unary rule, ϵ_I is a single fixed heap/effect annotation, so the snapshot variable k must be absent from $FV(\epsilon_I)$ just as it is absent from $FV(p)$, $FV(b)$, and $FV(e_v)$. In the evolving unary rule, $\epsilon_I(k)$ (equivalently, ϵ_k when the effect is written as ϵ) is a meta-level family of heap/effect annotations indexed by the loop stage. The binder $\forall k$ selects the family member for the current iteration; it does not capture an ordinary free occurrence of k inside a single fixed effect. Fixed-effect unary while rules therefore use one effect annotation throughout the loop, while evolving rules use the indexed family explicitly.

The two rules also make different commitments about the relation between the guard and the variant. The fixed-effect rule above deliberately uses the biconditional $p \implies (e_v \leq 0 \iff \neg b)$, because the stated rule, proof, and weakest-precondition presentation use an exact guard/variant alignment. A weaker fixed-effect lemma with only $p \wedge e_v \leq 0 \implies \neg b$ can be sound for partial correctness and upper-bound costs, provided the per-variant grades are nonnegative: exiting early at a positive variant leaves the fixed effect unchanged and only skips nonnegative grade terms. That weaker statement is a separate derived lemma, not the rule presented here. In contrast, the evolving-effect rule needs the biconditional in its current form. Its conclusion promises the final effect $\epsilon_I(0)$; if the loop could exit at $e_v = j > 0$, the derivation would only justify $\epsilon_I(j)$ unless the conclusion were weakened to expose the actual exit index.

Use the fixed rule when the loop body preserves the same effect annotation at every iteration, for example a read-only traversal or an update whose proof restores the same tracked domain. Use the evolving rule when each iteration intentionally changes the effect, such as consuming processed indices from β, growing a proved region, or shifting a tracked domain. In that case the body postcondition uses the exact stage $e_v = k - 1$, not merely $e_v < k$, because the post-effect is the particular family member $\epsilon_I(k-1)$. A weaker postcondition

$e_v < k$ would prove termination progress but would not identify which effect annotation should be used by the next iteration.

3.1 Unary validity definitions

LV($\{p \mid \epsilon\}C\{q \mid \epsilon'\}$) is the logical variables in the hoare triple, and a well formed interpretation I with respect to $\{p \mid \epsilon\}C\{q \mid \epsilon'\}$ is defined as:

$$I \vDash \{p \mid \epsilon\}C\{q \mid \epsilon'\} \text{ if } dom(I) \supseteq LV(\{p \mid \epsilon\}C\{q \mid \epsilon'\})$$

We define $(s,h) \vDash_I p \mid \epsilon$ to be $(s,h) \vDash_I p$ and $(s,h) \vDash_I \epsilon$.

Definition 3.1. *A partial correctness statement $\{p \mid \epsilon\}C\{q \mid \epsilon'\}$ is valid in the combined store $\sigma = (s,h)$ under interpretation I, written as $\sigma \vDash_I \{p \mid \epsilon\}C\{q \mid \epsilon'\}$ defined as follows:*

$$\forall \sigma' = (s',h'). \text{ if } \sigma \vDash_I p \mid \epsilon \text{ and } (\sigma, \sigma', \_) \in [\![C]\!], \text{then } \sigma' \vDash_I q \mid \epsilon'.$$

Definition 3.2. *A partial correctness statement $\{p \mid \epsilon\}C\{q \mid \epsilon'\}$ is valid in the combined store $\sigma = (s,h)$ under interpretation I with grade m, written as $\sigma \vDash^m_I \{p \mid \epsilon\}C\{q \mid \epsilon'\}$, when $[\![m]\!]_i(I) \in \mathbb{N}$ and:*

$$\forall \sigma' = (s',h'). \text{ if } \sigma \vDash_I p \mid \epsilon \text{ and } (\sigma, \sigma', m_1) \in [\![C]\!], \text{then } \sigma' \vDash_I q \mid \epsilon' \text{ and } m_1 \leq [\![m]\!]_i(I)$$

Definition 3.3. *A partial correctness statement $\{p \mid \epsilon\}C\{q \mid \epsilon'\}$ is valid written as $\vDash \{p \mid \epsilon\}C\{q \mid \epsilon'\}$ defined as follows:*

$$\forall \sigma, I. \sigma \vDash_I \{p \mid \epsilon\}C\{q \mid \epsilon'\}$$

Definition 3.4. *A graded partial correctness statement $\{p \mid \epsilon\}C\{q \mid \epsilon'\}$ is valid with grade m, written as $\vDash^m \{p \mid \epsilon\}C\{q \mid \epsilon'\}$, if for every well-formed interpretation I covering the logical variables of the judgment and satisfying $[\![m]\!]_i(I) \in \mathbb{N}$,*

$$\forall \sigma. \sigma \vDash^m_I \{p \mid \epsilon\}C\{q \mid \epsilon'\}.$$

3.2 Weakest Precondition Style

$$\frac{x \notin FV(\epsilon)}{ewp(x := e, q, \epsilon) = q[e/x], \epsilon}\ \textbf{assign}$$

$$\frac{p_1, \epsilon_1 = ewp(\ C_1, q, \epsilon) \qquad p_2, \epsilon_2 = ewp(\ C_2, q, \epsilon)}{ewp(\ \texttt{if}\ b\ \texttt{then}\ C_1\ \texttt{else}\ C_2, q, \epsilon) = b\implies p_1 \wedge \neg b \implies p_2, min(\epsilon_1, \epsilon_2)}\ \textbf{conditional}$$

$$\frac{p_1, \epsilon_1 = ewp(\ C_2, q, \epsilon)}{ewp(\ C_1; C_2, q, \epsilon) = ewp(\ C_1, p_1, \epsilon_1)}\ \textbf{sequence}$$

$$\frac{q' \Rightarrow q \qquad q' \vdash \epsilon \leq \epsilon_1 \qquad ewp(\ C, q', \epsilon_1)}{ewp(\ C, q, \epsilon)}\ \textbf{monotonicity}$$

$$\frac{\begin{array}{c} n \geq 0 \qquad k \notin FV(I) \cup FV(b) \cup FV(e_v) \cup FV(\epsilon_I) \\ \vDash I \implies 0 \leq e_v \\ \vDash I \implies (e_v \leq 0 \iff \neg b) \\ \forall k.\ 1 \leq k \leq n \implies I \wedge b \wedge e_v = k,\ \epsilon_I \implies ewp(\ C, I \wedge e_v < k, \epsilon_I) \\ I \wedge \neg b \implies q \end{array}}{ewp(\ \texttt{while}\ b\ \texttt{do}\ C, q, \epsilon_I) = I \wedge e_v \leq n,\ \epsilon_I}\ \textbf{awhile}$$

$$\frac{\begin{array}{c} n \geq 0 \qquad k \notin FV(I) \cup FV(b) \cup FV(e_v) \qquad \vDash I \implies 0 \leq e_v \\ \vDash I \implies e_v \leq 0 \iff \neg b \\ \epsilon_I(k) \text{ is a meta-level indexed effect family} \\ \forall k.\ 1 \leq k \leq n \implies I \wedge b \wedge e_v = k,\ \epsilon_I(k) \implies ewp(\ C, I \wedge e_v = k - 1, \epsilon_I(k-1)) \\ I \wedge \neg b \implies q \end{array}}{ewp(\ \texttt{while}\ b\ \texttt{do}\ C, q, \epsilon_I(0)) = I \wedge e_v = n,\ \epsilon_I(n)}\ \textbf{awhile-evolving}$$

$$\frac{\begin{array}{c} \ell \text{ fresh logical index} \\ \ell \notin FV(q) \cup FV(\epsilon') \\ u_a \notin arrFV(q) \cup arrFV(\epsilon') \\ \epsilon = \epsilon' \star \{u_a \mapsto^{\phi} [0, \ell - 1]\} \end{array}}{ewp(u_a \leftarrow array(e_1)(e_2), q, \epsilon) = q \wedge e_1 = \ell \wedge \ell \geq 0 \wedge \phi(e_2), \epsilon'}\ \textbf{arrayAllocFull}$$

$$\frac{\begin{array}{c} u_a \notin arrFV(q) \cup arrFV(\epsilon') \\ \epsilon = \epsilon' \star \{u_a \mapsto^{\phi} \emptyset\} \end{array}}{ewp(u_a \leftarrow array(e_1)(e_2), q, \epsilon) = q \wedge e_1 \geq 0, \epsilon'}\ \textbf{arrayAllocEmpty}$$

$$\frac{\epsilon(u_a) = \beta \qquad j \text{ fresh}}{ewp(u_a[e_1] := e_2, q, \epsilon) = q[u_a[e_1 \mapsto e_2]/u_a] \wedge e_1 = j \wedge (j \in \beta \implies \phi(e_2)), \epsilon[u_a \to \beta - \{j\}]}\ \textbf{arrayUpdt}$$

$$\frac{x \notin FV(\epsilon)}{ewp(x := u_a[e], q, \epsilon) = q[u_a[e]/x], \epsilon}\ \textbf{arrayRead}$$

The two while clauses above are the weakest-precondition counterparts of the Hoare-style fixed and evolving while rules. In the fixed-effect clause, the body precondition omits the explicit conjunct $0 \leq e_v \leq n$ because it follows from the premise snapshot $e_v = k$ together with $1 \leq k \leq n$. The nonnegative per-iteration grade side condition belongs to the Hoare presentation; this weakest-precondition section computes only the precondition and effect annotation.

3.3 Example discipline for evolving effects

The examples in this subsection illustrate a syntactic discipline used by the unary while rules: a heap/effect annotation may mention logical variables and pure set indices, but it must not mention a mutable program variable as if that variable were a set-index parameter. During a loop proof, the program variant is related to a fresh logical stage variable by the premise $e_v = k$; the effect is then selected as a meta-level family member such as $\epsilon_I(k)$.

3.3.1 p2: while if

```
p2 =                              { len(a) > n >=0   }
     i = n ;                      {i = n /\ len(a) > n >=0   }
     while (i>0) do
          x = a[i];
          if (x > 0)
               then skip;skip
               else a[i] = 0 ;
          i = i -1
```

Let

$$\begin{aligned} p &\triangleq len(a) > n \wedge n \geq 0 \wedge 0 \leq i \leq n \wedge \forall j :: \mathbb{Z}.\ ((i < j \wedge j \leq n) \implies a[j] \geq 0), \\ e_v &\triangleq i, \\ b &\triangleq i > 0, \\ \epsilon_I(k) &\triangleq u_a \mapsto^{neg} (\beta - [k+1, n]). \end{aligned}$$

The important point is that $\epsilon_I(k)$ uses the logical stage k, not the program variable i, inside the set expression. At a loop-body premise with $i = k$, the body starts from $u_a \mapsto^{neg} (\beta - [k+1, n])$ and, after processing cell k and executing $i := i - 1$, establishes the next family member $u_a \mapsto^{neg} (\beta - [k, n]) = \epsilon_I(k-1)$. We use the empty-interval convention $[n+1, n] = \emptyset$, so $\epsilon_I(n) = u_a \mapsto^{neg} \beta$ at loop entry. Thus the loop is an instance of **awhile-rule-evolving**, not the fixed-effect rule:

$$\vDash^{1+\sum_{k=1}^{n} 3} \{len(a) > n \wedge n \geq 0 \mid u_a \mapsto^{neg} \beta\}\ p2\ \{\forall j :: \mathbb{Z}.\ ((0 < j \wedge j \leq n) \implies a[j] \geq 0) \mid u_a \mapsto^{neg} (\beta - [1, n])\}.$$

The displayed grade is a conservative default-cost bound for this explicit-read version: initialization costs 1, and each loop iteration costs at most one read, one update, and one decrement. The purpose of the example here is the effect

discipline. More precise beta-sensitive cost bounds require choosing a property that identifies cheaper branches, as in the normalized no-β discussion above or the later relational cost examples.

This example intentionally differs from the β-sensitive cost example in Section 2.3: here we use an explicit read and a negative-cell effect to illustrate evolving-effect discipline. In the normalized no-explicit-read version of Section 2.3, the tight positive-β cost bound is $1 + \sum_k m_k$.

3.4 Soundness of unary logic

The proof below applies the semantic validity definitions from Section 3.1. We keep those definitions before the examples because the example judgments use $\vDash$ notation; this subsection only proves that the rule system is sound for those definitions.

We first state two substitution lemmas used throughout the soundness proofs. All substitutions are capture-avoiding: bound logical variables (e.g. in $\forall i :: S$, $\exists i :: S$) are α-renamed away from the free variables of the substituted term.

Lemma 3.1 (Scalar substitution)**.** *For any assertion q, program variable x, expression e, state (s, h), and interpretation I:*

$$(s, h) \vDash_I q[e/x] \quad \textit{iff} \quad (s[x \mapsto [\![e]\!]_e(s, h)], h) \vDash_I q$$

Lemma 3.2 (Array substitution)**.** *For any assertion q, array variable u_a, index expression e_1, value expression e_2, state (s, h), and interpretation I: let $i = [\![e_1]\!]_e(s, h)$ and $v = [\![e_2]\!]_e(s, h)$, and assume $u_a \in dom(h)$ and $0 \le i < length(h(u_a))$ (so the overwrite is in-bounds and length-preserving). Then:*

$$(s, h) \vDash_I q[u_a[e_1 \mapsto e_2]/u_a] \quad \textit{iff} \quad (s, h[u_a \mapsto h(u_a)\{i \mapsto v\}]) \vDash_I q$$

That is, substituting the updated array $u_a[e_1 \mapsto e_2]$ for u_a in the assertion is equivalent to evaluating the assertion in the updated heap.

Both lemmas follow by structural induction on q. The array substitution lemma is the key step in the arrayUpdt and arrayRead soundness proofs: it bridges the syntactic substitution in the Hoare rule's precondition with the semantic update in the command interpretation.

Theorem 3.3 (Unary soundness under default V1 costs)**.** *Assume the command semantics uses the default V1 weights stated above:*

$$c_{\texttt{skip}} = c_{if} = c_{while} = 0, \qquad c_{assign} = c_{read} = c_{update} = c_{alloc} = 1.$$

If $\vdash^m \{p \mid \epsilon\}\ C\ \{q \mid \epsilon'\}$, then $\vDash^m \{p \mid \epsilon\}\ C\ \{q \mid \epsilon'\}$. Changing these weights requires changing the primitive grades or proving a separate rule-compatibility condition.

Proof. The proof of soundness uses structural induction on the derivation of $\vdash^m \{p \mid \epsilon\}\ C\ \{q \mid \epsilon'\}$.

Case

$$\frac{x \notin FV(\epsilon)}{\vdash^1 \{q[e/x] \mid \epsilon[e/x]\}\ x := e\ \{q \mid \epsilon\}}\ \textbf{assign}$$

For any $\sigma = (s, h)$ and I, assume $\sigma \vDash_I q[e/x]$ and $(s, h) \vDash_I \epsilon[e/x]$, we want to show that: for any $\sigma' = (s', h')$, $(\sigma, \sigma', m_1) \in [\![x := e]\!]$,

1. $\sigma' \vDash_I q$
2. $(s', h') \vDash_I \epsilon$
3. $m_1 \leq [\![1]\!](I)$

According to $[\![x := e]\!]$, we have $\sigma' = (s[x \mapsto v], h)$ and $m_1 = 1$. So (3) is proved.

We know that $x \notin FV(\epsilon)$, so from $(s, h) \vDash_I \epsilon[e/x]$, we know that $(s[x \mapsto v], h) \vDash_I \epsilon$, which proves (2).

We know $(s[x \mapsto v], h) \vDash_I q$ when we have $(s, h) \vDash_I q[e/x]$ when $v = [\![e]\!](s, h)$.

Case

$$\frac{}{\vdash^0 \{p \mid \epsilon\}\ \texttt{skip}\ \{p \mid \epsilon\}}\ \textbf{skip}$$

For any $\sigma = (s, h)$ and I, assume $\sigma \vDash_I p$ and $\sigma \vDash_I \epsilon$. If $(\sigma, \sigma', m_1) \in [\![\texttt{skip}]\!]$, then by the command semantics $\sigma' = \sigma$ and $m_1 = c_{\texttt{skip}} = 0$. Hence $\sigma' \vDash_I p$, $\sigma' \vDash_I \epsilon$, and $m_1 = 0 \leq [\![0]\!](I)$. This proves the case.

Case

$$\frac{\vdash^{m_1} \{p \mid \epsilon_1\}\ C_1\ \{r \mid \epsilon_2\} \qquad \vdash^{m_2} \{r \mid \epsilon_2\}\ C_2\ \{q \mid \epsilon_3\}}{\vdash^{m_1+m_2} \{p \mid \epsilon_1\}\ C_1; C_2\ \{q \mid \epsilon_3\}}\ \textbf{sequence}$$

For any $\sigma = (s, h)$ and I, assume $\sigma \vDash_I p$ and $h \vDash_I \epsilon_1$, we want to show that: for any $\sigma' = (s', h')$, $(\sigma, \sigma', m) \in [\![C_1; C_2]\!]$,

1. $\sigma' \vDash_I q$
2. $(s', h') \vDash_I \epsilon_3$
3. $m \leq [\![m_1 + m_2]\!](I)$

According to the interpretation of $[\![C_1; C_2]\!]$, we know that: there exist $\sigma_1 = (s_1, h_1)$ so that $(\sigma, \sigma_1, m_1') \in [\![C_1]\!]$ and $(\sigma_1, \sigma', m_2') \in [\![C_2]\!]$ and $m = m_1' + m_2'$.

By induction hypothesis on the first premise, we know that $\sigma_1 \vDash_I r$ and $\sigma_1 \vDash_I \epsilon_2$ and $m_1' \leq [\![m_1]\!](I)$.

By induction hypothesis on the second premise, we know that $\sigma' \vDash_I q$ and $\sigma' \vDash_I \epsilon_3$ and $m_2' \leq [\![m_2]\!](I)$. Adding these inequalities gives $m = m_1' + m_2' \leq [\![m_1 + m_2]\!](I)$. This completes the proof.

Case

$$\frac{\vdash^{m'} \{p' \mid \epsilon_1'\}\ C\ \{q' \mid \epsilon_2'\} \qquad \vDash p \Rightarrow p' \qquad \vDash q' \Rightarrow q \qquad p \vdash \epsilon_1' \leq \epsilon_1 \qquad q \vdash \epsilon_2 \leq \epsilon_2' \qquad p \vdash m' \leq m}{\vdash^{m} \{p \mid \epsilon_1\}\ C\ \{q \mid \epsilon_2\}}\ \textbf{conseq}$$

For any $\sigma = (s, h)$ and I, assume $\sigma \vDash_I p$ and $\sigma \vDash_I \epsilon_1$, we want to show that: for any $\sigma' = (s', h')$, $(\sigma, \sigma', m_1) \in [\![C]\!]$,

1. $\sigma' \vDash_I q$
2. $\sigma' \vDash_I \epsilon_2$
3. $m_1 \leq [\![m]\!]_i(I)$

From $\vDash p \Rightarrow p'$ and $\sigma \vDash_I p$ we know that $\sigma \vDash_I p'$.

From $p \vdash \epsilon_1' \leq \epsilon_1$ and $\sigma \vDash_I p$ we have $\sigma \vDash_I \epsilon_1' \leq \epsilon_1$, i.e. (by the semantic order) $\sigma \vDash_I \epsilon_1 \implies \sigma \vDash_I \epsilon_1'$. Since we know $\sigma \vDash_I \epsilon_1$, we conclude $\sigma \vDash_I \epsilon_1'$.

By induction hypothesis, we know that for any $\sigma' = (s', h')$, $(\sigma, \sigma', m_1) \in [\![C]\!]$, we have $\sigma' \vDash_I q'$ and $\sigma' \vDash_I \epsilon_2'$ and $m_1 \leq [\![m']\!]_i(I)$.

From $\vDash q' \Rightarrow q$ and $\sigma' \vDash_I q'$, we know that $\sigma' \vDash_I q$.

From $q \vdash \epsilon_2 \leq \epsilon_2'$ and $\sigma' \vDash_I q$ we have $\sigma' \vDash_I \epsilon_2 \leq \epsilon_2'$, i.e. (by the semantic order) $\sigma' \vDash_I \epsilon_2' \implies \sigma' \vDash_I \epsilon_2$. From $\sigma' \vDash_I \epsilon_2'$ we conclude $\sigma' \vDash_I \epsilon_2$.

From $m_1 \leq [\![m']\!]_i(I)$ and $p \vdash m' \leq m$, from $s, h \vDash_I p$, we get $\vDash_I m' \leq m$, hence $[\![m']\!]_i(I) \leq [\![m]\!]_i(I)$. By transitivity, $m_1 \leq [\![m]\!]_i(I)$. This completes the proof.

Case

$$\frac{\ell \text{ fresh logical index} \qquad u_a \notin \mathit{arrFV}(p) \cup \mathit{arrFV}(\epsilon)}{\vdash^{1} \{p \wedge e_1 = \ell \wedge \ell \geq 0 \wedge \phi(e_2) \mid \epsilon\}\ u_a \leftarrow \mathit{array}(e_1)(e_2)\ \{p \mid \epsilon \star \{u_a \mapsto^{\phi} [0, \ell - 1]\}\}}\ \textbf{arrayAlloc1}$$

For any $\sigma = (s, h)$ and I, assume $\sigma \vDash_I p \wedge e_1 = \ell \wedge \ell \geq 0 \wedge \phi(e_2)$ and $\sigma \vDash_I \epsilon$, we want to show that: for any $\sigma' = (s', h')$, $(\sigma, \sigma', m_1) \in [\![u_a \leftarrow \mathit{array}(e_1)(e_2)]\!]$,

1. $\sigma' \vDash_I p$
2. $\sigma' \vDash_I \epsilon \star \{u_a \mapsto^{\phi} [0, \ell - 1]\}$
3. $m_1 \leq [\![1]\!](I)$

Unfold the definition of $[\![u_a \leftarrow \mathit{array}(e_1)(e_2)]\!](s, h)$, we know: $\{\big((s, h[u_a \mapsto [v_2, \ldots, v_2]]), c_{alloc}\big) \mid n = [\![e_1]\!]_e(s, h) \wedge v_2 = [\![e_2]\!]_e(s, h) \wedge u_a \notin \mathit{dom}(h) \wedge \mathit{size}([v_2, \ldots, v_2]) = n\}$.

Under the default V1 weights, $m_1 = c_{alloc} = 1$, so $m_1 \leq [\![1]\!](I)$ is proved. The side condition $u_a \notin \mathit{arrFV}(p)$ ensures that allocating the fresh heap cell for u_a cannot change the truth of p, so from $\sigma \vDash_I p$ we get $\sigma' \vDash_I p$.

The side condition $u_a \notin \mathit{arrFV}(\epsilon)$ and the operational premise $u_a \notin \mathit{dom}(h)$ let us split the post heap as the old heap satisfying ϵ and the fresh cell for

u_a. From $\sigma \vDash_I \phi(e_2)$, we conclude $\phi(v_2)$ where $v_2 = [\![e_2]\!]_e(s,h)$. From the snapshot premise $\sigma \vDash_I e_1 = \ell$, we have $n = [\![e_1]\!]_e(s,h) = I(\ell)$, and $\ell \geq 0$ gives $n \geq 0$ (the tracked interval $[0, \ell - 1]$ is empty when $\ell = 0$). Since every allocated cell contains v_2, the fresh fragment satisfies $\{u_a \mapsto^{\phi} [0, I(\ell) - 1]\}$. Hence $\sigma' \vDash_I \epsilon \star \{u_a \mapsto^{\phi} [0, \ell - 1]\}$.

Case

$$\frac{u_a \notin arrFV(p) \cup arrFV(\epsilon)}{\vdash^1 \{p \wedge e_1 \geq 0 \mid \epsilon\}\ u_a \leftarrow array(e_1)(e_2)\ \{p \mid \epsilon \star \{u_a \mapsto^{\phi} \emptyset\}\}}\ \textbf{arrayAlloc2}$$

For any $\sigma = (s,h)$ and I, assume $\sigma \vDash_I p \wedge e_1 \geq 0$ and $h \vDash_I \epsilon$, we want to show that: for any $\sigma' = (s',h')$, $(\sigma, \sigma', m_1) \in [\![u_a \leftarrow array(e_1)(e_2)]\!]$,

1. $\sigma' \vDash_I p$
2. $h' \vDash_I \epsilon \star \{u_a \mapsto^{\phi} \emptyset\}$
3. $m_1 \leq [\![1]\!](I)$

Similarly to the case **arrayAlloc1**, unfold the definition of $[\![u_a \leftarrow array(e_1)(e_2)]\!](s,h)$: $\{\big((s, h[u_a \mapsto [v_2, \ldots, v_2]]), c_{alloc}\big) \mid n = [\![e_1]\!]_e(s,h) \wedge v_2 = [\![e_2]\!]_e(s,h) \wedge u_a \notin dom(h) \wedge size([v_2, \ldots, v_2]) = n\}$. The freshness side condition preserves p as in **arrayAlloc1**, so (1) holds. Under the default V1 weights, $m_1 = c_{alloc} = 1$, so (3) holds.

For (2), the fresh u_a fragment satisfies $\{u_a \mapsto^{\phi} \emptyset\}$ vacuously. Combining it with the old heap fragment satisfying ϵ gives $h' \vDash_I \epsilon \star \{u_a \mapsto^{\phi} \emptyset\}$. No length snapshot is needed in **arrayAlloc2**, because the postcondition tracks the empty set of allocated indices.

Case

$$\frac{j \text{ fresh}}{\vdash^1 \{q[u_a[e_1 \mapsto e_2]/u_a] \wedge \phi(e_2) \wedge e_1 = j \mid \epsilon \star \{u_a \mapsto^{\phi} \beta\}\}\ u_a[e_1] := e_2\ \{q \mid \epsilon \star \{u_a \mapsto^{\phi} \beta \cup \{j\}\}\}}\ \textbf{arrayUpdt1}$$

For any $\sigma = (s,h)$ and I with $I(j) = [\![e_1]\!]_e(s,h)$ (from $e_1 = j$ in the precondition), assume $\sigma \vDash_I q[u_a[e_1 \mapsto e_2]/u_a] \wedge \phi(e_2)$ and $\sigma \vDash_I \epsilon \star \{u_a \mapsto^{\phi} \beta\}$. We show that for any $\sigma' = (s',h')$, $(\sigma, \sigma', m_1) \in [\![u_a[e_1] := e_2]\!]$:

1. $\sigma' \vDash_I q$
2. $\sigma' \vDash_I \epsilon \star \{u_a \mapsto^{\phi} \beta \cup \{j\}\}$
3. $m_1 \leq [\![1]\!](I)$

Unfold the command semantics: $s' = s$, $h' = h[u_a \mapsto h(u_a)\{i \mapsto v\}]$ where $i = [\![e_1]\!]_e(s,h) = I(j)$ and $v = [\![e_2]\!]_e(s,h)$, and $m_1 = 1$. So (3) holds.

From $(s,h) \vDash_I q[u_a[e_1 \mapsto e_2]/u_a]$ and the substitution lemma, $(s,h') \vDash_I q$. This proves (1).

From $\sigma \vDash_I \epsilon \star \{u_a \mapsto^{\phi} \beta\}$, there exist h_1, h_2 with $h = h_1 + h_2$, $h_1 \vDash_I \epsilon$, and $h_2 \vDash_I \{u_a \mapsto^{\phi} \beta\}$, i.e., $\forall k \in [\![\beta]\!]_b(I).\phi(h_2(u_a)[k])$.

After the update: $h' = h_1 + h_2[u_a \mapsto h_2(u_a)\{i \mapsto v\}]$. We need $\forall k \in [\![\beta \cup \{j\}]\!]_b(I).\phi(h'(u_a)[k])$.

Since β is pure (J contains no program variables or array lookups), $[\![\beta]\!]_b(I)$ does not depend on the state. And $[\![\{j\}]\!]_b(I) = \{I(j)\} = \{i\}$.

For $k \in [\![\beta]\!]_b(I)$ with $k \neq i$: $h'(u_a)[k] = h_2(u_a)[k]$, so $\phi(h'(u_a)[k])$ holds from the precondition effect.

For $k = i = I(j)$: $h'(u_a)[i] = v = [\![e_2]\!]_e(s,h)$. From $(s,h) \vDash_I \phi(e_2)$, we get $\phi(v)$. So $\phi(h'(u_a)[i])$ holds.

By the in-bounds update semantics the transition requires $0 \leq i < \mathit{length}(h(u_a))$, and the point update is length-preserving, so $i = I(j) \in [0, \mathit{length}(h'(u_a)))$, i.e. $[\![\{j\}]\!]_b(I) \subseteq [0, \mathit{length}(h'(u_a)))$. With $[\![\beta]\!]_b(I) \subseteq [0, \mathit{length}(h(u_a))) = [0, \mathit{length}(h'(u_a)))$ from the precondition effect, $[\![\beta \cup \{j\}]\!]_b(I) \subseteq [0, \mathit{length}(h'(u_a)))$, which is the in-bounds requirement of the post-effect.

Thus $h' \vDash_I \epsilon \star \{u_a \mapsto^\phi \beta \cup \{j\}\}$. This proves (2).

Case

$$\frac{j \text{ fresh}}{\vdash^1 \{q[u_a[e_1 \mapsto e_2]/u_a] \wedge e_1 = j \mid \epsilon \star \{u_a \mapsto^\phi \beta\}\}\ u_a[e_1] := e_2\ \{q \mid \epsilon \star \{u_a \mapsto^\phi \beta - \{j\}\}\}}\ \textbf{arrayUpdt2}$$

Similar to **arrayUpdt1**. From $e_1 = j$ in the precondition, $I(j) = [\![e_1]\!]_e(s,h) = i$. Goals (1) and (3) follow identically. For (2): from $h_2 \vDash_I \{u_a \mapsto^\phi \beta\}$, we know $\forall k \in [\![\beta]\!]_b(I).\phi(h_2(u_a)[k])$. After the update, for $k \in [\![\beta - \{j\}]\!]_b(I)$, we have $k \neq I(j) = i$, so $h'(u_a)[k] = h_2(u_a)[k]$ and ϕ is preserved. (We make no claim about index i in the postcondition effect, since it is removed from β.)

Case

$$\frac{j \text{ fresh}}{\vdash^1 \{q[u_a[e]/x] \wedge e = j \wedge j \in \beta \mid \epsilon \star \{u_a \mapsto^\phi \beta\}\}\ x := u_a[e]\ \{q \wedge \phi(x) \mid \epsilon \star \{u_a \mapsto^\phi \beta\}\}}\ \textbf{arrayRead1}$$

For any $\sigma = (s,h)$ and I with $I(j) = [\![e]\!]_e(s,h)$ (from $e = j$), assume $\sigma \vDash_I q[u_a[e]/x] \wedge j \in \beta$ and $\sigma \vDash_I \epsilon \star \{u_a \mapsto^\phi \beta\}$. We show that for any $(\sigma', m_1) \in [\![x := u_a[e]]\!](s,h)$:

1. $\sigma' \vDash_I q \wedge \phi(x)$
2. $\sigma' \vDash_I \epsilon \star \{u_a \mapsto^\phi \beta\}$
3. $m_1 \leq [\![1]\!](I)$

Let $i = [\![e]\!]_e(s,h) = I(j)$ and $v = h(u_a)[i]$. Then $s' = s[x \mapsto v]$, $h' = h$, $m_1 = 1$. So (3) holds.

Since $h' = h$ and the read does not modify the heap, $\sigma' \vDash_I \epsilon \star \{u_a \mapsto^\phi \beta\}$ follows directly from the precondition. This proves (2).

From $(s,h) \vDash_I q[u_a[e]/x]$ and the substitution lemma, $(s[x \mapsto v], h) \vDash_I q$.

From $j \in \beta$ and $\sigma \vDash_I \epsilon \star \{u_a \mapsto^\phi \beta\}$, we know $I(j) \in [\![\beta]\!]_b(I)$ and $\forall k \in [\![\beta]\!]_b(I).\phi(h(u_a)[k])$. So $\phi(h(u_a)[I(j)]) = \phi(h(u_a)[i]) = \phi(v)$. Since $v = s'(x)$, we have $(s', h') \vDash_I \phi(x)$.

This proves (1).

Case

$$\frac{}{\vdash^1 \{q[u_a[e]/x] \mid \epsilon \star \{u_a \mapsto^\phi \beta\}\}\ x := u_a[e]\ \{q \mid \epsilon \star \{u_a \mapsto^\phi \beta\}\}}\ \textbf{arrayRead2}$$

Similar to **arrayRead1** but without the ϕ conclusion. Let $i = [\![e]\!]_e(s,h)$, $v = h(u_a)[i]$. Then $s' = s[x \mapsto v]$, $h' = h$, $m_1 = 1$. Goal (3) holds. Goal (2) holds since $h' = h$ (heap unchanged). Goal (1): from $q[u_a[e]/x]$ and the substitution lemma, $(s[x \mapsto v], h) \vDash_I q$. (No $\phi(x)$ is claimed since we do not assume $e \in \beta$.)

Case

$$\frac{\vdash^m \{p \wedge b \mid \epsilon\}\ C_1\ \{q \mid \epsilon'\} \qquad \vdash^m \{p \wedge \neg b \mid \epsilon\}\ C_2\ \{q \mid \epsilon'\}}{\vdash^m \{p \mid \epsilon\}\ \texttt{if}\ b\ \texttt{then}\ C_1\ \texttt{else}\ C_2\ \{q \mid \epsilon'\}}\ \textbf{conditional}$$

For any $\sigma = (s,h)$ and I, assume $\sigma \vDash_I p$ and $\sigma \vDash_I \epsilon$, we want to show that: for any $\sigma' = (s',h')$, $(\sigma, \sigma', m_1) \in [\![\ \texttt{if}\ b\ \texttt{then}\ C_1\ \texttt{else}\ C_2]\!]$,

1. $\sigma' \vDash_I q$
2. $\sigma' \vDash_I \epsilon'$
3. $m_1 \leq [\![m]\!](I)$

There are two cases of $[\![\ \texttt{if}\ b\ \texttt{then}\ C_1\ \texttt{else}\ C_2]\!]$:

Subcase 1 ($[\![b]\!](s,h) = \texttt{true}$): Then $((s',h'), m_1)$ belongs to the first union member of $[\![\ \texttt{if}\ b\ \texttt{then}\ C_1\ \texttt{else}\ C_2]\!](s,h)$, so $m_1 = c_{if} + m_t$ for some $((s',h'), m_t) \in [\![C_1]\!](s,h)$. From $(s,h) \vDash_I p$ and $[\![b]\!](s,h) = \texttt{true}$ we have $(s,h) \vDash_I p \wedge b$. By the induction hypothesis on the first premise,

1. $(s',h') \vDash_I q$
2. $(s',h') \vDash_I \epsilon'$
3. $m_t \leq [\![m]\!](I)$

Under the default cost model $c_{if} = 0$, so $m_1 = m_t \leq [\![m]\!](I)$. This proves the subcase.

Subcase 2 ($[\![b]\!](s,h) = \texttt{false}$): Then $((s',h'), m_1)$ belongs to the second union member of $[\![\ \texttt{if}\ b\ \texttt{then}\ C_1\ \texttt{else}\ C_2]\!](s,h)$, so $m_1 = c_{if} + m_f$ for some $((s',h'), m_f) \in [\![C_2]\!](s,h)$. From $(s,h) \vDash_I p$ and $[\![b]\!](s,h) = \texttt{false}$ we have $(s,h) \vDash_I p \wedge \neg b$. By the induction hypothesis on the second premise,

1. $(s',h') \vDash_I q$
2. $(s',h') \vDash_I \epsilon'$
3. $m_f \leq [\![m]\!](I)$

Under the default cost model $c_{if} = 0$, so $m_1 = m_f \leq [\![m]\!](I)$. This proves the subcase.

Case

$$\frac{\vdash^m \{p \mid \epsilon\}\ C\ \{q \mid \epsilon'\} \qquad \vdash^m \{r \mid \epsilon\}\ C\ \{q \mid \epsilon'\}}{\vdash^m \{p \vee r \mid \epsilon\}\ C\ \{q \mid \epsilon'\}}\ \textbf{disjunction-precond}$$

For any $\sigma = (s, h)$ and I, assume $\sigma \vDash_I p \vee r$ and $\sigma \vDash_I \epsilon$. If $(\sigma, \sigma', m_1) \in [\![C]\!]$, then either $\sigma \vDash_I p$ or $\sigma \vDash_I r$. In the first case the induction hypothesis for the first premise gives $\sigma' \vDash_I q$, $\sigma' \vDash_I \epsilon'$, and $m_1 \leq [\![m]\!](I)$. In the second case the induction hypothesis for the second premise gives the same three conclusions. This proves the case.

Case

$$\frac{\begin{array}{c} n \geq 0 \qquad k \notin FV(p) \cup FV(b) \cup FV(e_v) \\ k \text{ not free in the fixed heap/effect annotation } \epsilon_I \\ \vDash p \implies 0 \leq e_v \\ \vDash p \implies (e_v \leq 0 \iff \neg b) \\ \forall k.\ 1 \leq k \leq n \Rightarrow 0 \leq m_k \\ \forall k.\ 1 \leq k \leq n \Rightarrow \vdash^{m_k} \{p \wedge b \wedge e_v = k \wedge 0 \leq e_v \leq n \mid \epsilon_I\}\ C\ \{p \wedge e_v < k \mid \epsilon_I\} \end{array}}{\vdash^{\sum_{k=1}^{n} m_k} \{p \wedge e_v \leq n \mid \epsilon_I\}\ \texttt{while}\ b\ \texttt{do}\ C\ \{p \wedge \neg b \mid \epsilon_I\}}\ \textbf{awhile-rule}$$

For any $\sigma = (s, h)$ and I, assume $\sigma \vDash_I p \wedge e_v \leq n$ and $(s, h) \vDash_I \epsilon_I$. By the premise $p \implies 0 \leq e_v$, the current variant value $k_0 = [\![e_v]\!]_i((s, h), I)$ is a natural number with $k_0 \leq n$.

We prove the stronger claim by well-founded induction on the current variant value k_0. Since $\sigma \vDash_I p$, the biconditional premise gives $\sigma \vDash_I (e_v \leq 0 \iff \neg b)$. If $k_0 = 0$, then $\sigma \vDash_I e_v \leq 0$, hence $\sigma \vDash_I \neg b$; the while semantics takes zero body steps, the state and ϵ_I are unchanged, and the postcondition follows.

If $k_0 > 0$, then $\sigma \vDash_I e_v = k_0$ and k_0 is a natural number, so $\sigma \vDash_I \neg(e_v \leq 0)$. The $\neg b \Rightarrow e_v \leq 0$ direction of the biconditional therefore gives, by contraposition, $\sigma \vDash_I \neg\neg b$, hence classically $\sigma \vDash_I b$. Thus the body-premise precondition $p \wedge b \wedge e_v = k_0 \wedge 0 \leq e_v \leq n$ holds. Since the guard is true, unfolding the given terminating while execution exposes a first body execution to σ_1 with cost m_1, followed by the residual loop execution from σ_1. Instantiate the body premise with the current snapshot $k = k_0$. The side conditions on k make this instantiation only a premise snapshot; it does not capture p, b, e_v, or the fixed effect ϵ_I. Applying the soundness induction hypothesis to the body derivation for this first body execution gives:

$$\sigma_1 \vDash_I p \wedge e_v < k_0, \qquad \sigma_1 \vDash_I \epsilon_I, \qquad m_1 \leq [\![m_{k_0}]\!]_i(I[k \mapsto k_0]).$$

Since p is re-established, the nonnegative-variant premise gives $0 \leq [\![e_v]\!]_i(\sigma_1, I)$. Thus the next variant value is a natural number strictly smaller than k_0, and the well-founded induction hypothesis applies to the remaining loop iterations.

The visited variant values are therefore a strictly decreasing subset of $\{1, \ldots, n\}$. Summing the per-iteration grade bounds over the visited values and weakening with the nonnegative grades for unvisited indices yields:

$$m' \leq \sum_{k \in \mathit{Visited}} [\![m_k]\!]_i(I[k \mapsto k]) \leq [\![\sum_{k=1}^{n} m_k]\!]_i(I).$$

The final state satisfies $p \wedge \neg b$ by the induction claim, and the fixed effect remains ϵ_I throughout. This completes the proof.

Case

$$\frac{\begin{array}{c} n \geq 0 \qquad k \notin FV(p) \cup FV(b) \cup FV(e_v) \qquad \vDash p \implies 0 \leq e_v \\ \vDash p \implies (e_v \leq 0 \iff \neg b) \\ \epsilon_I(k) \text{ is a meta-level indexed effect family} \\ \forall k.\ 1 \leq k \leq n \implies 0 \leq m_k \\ \forall k.\ 1 \leq k \leq n \implies \vdash^{m_k} \{p \wedge b \wedge e_v = k \mid \epsilon_I(k)\}\ C\ \{p \wedge e_v = k-1 \mid \epsilon_I(k-1)\} \end{array}}{\vdash^{\sum_{k=1}^{n} m_k} \{p \wedge e_v = n \mid \epsilon_I(n)\}\ \texttt{while}\ b\ \texttt{do}\ C\ \{p \wedge \neg b \mid \epsilon_I(0)\}}\ \textbf{awhile-rule-evolving}$$

By induction on n (the initial variant value, which equals the iteration count due to $\iff$).

Base case: $n = 0$**.** From $e_v = 0$ and $e_v \leq 0 \iff \neg b$, we have $\neg b$. The loop does not execute: $\sigma' = \sigma$, $m' = 0$. Postcondition $p \wedge \neg b$ holds. Effect: $\epsilon_I(0) = \epsilon_I(0)$. Grade: $0 \leq \sum_{k=1}^{0} m_k = 0$.

Inductive case: $n > 0$**.** From $e_v = n > 0$ and $\iff$, we have b is true. Instantiate the premise with $k = n$: from $\sigma \vDash_I p \wedge b \wedge e_v = n$ and $\sigma \vDash_I \epsilon_I(n)$, after one body execution $(\sigma, \sigma_1, m_1') \in [\![C]\!]$:

(a) $\sigma_1 \vDash_I p \wedge e_v = n - 1$

(b) $\sigma_1 \vDash_I \epsilon_I(n-1)$

(c) $m_1' \leq [\![m_k]\!]_i(I[k \mapsto n])$

Why $k \notin FV(p)$ is critical: Since $k \notin FV(p)$, the invariant p in (a) does not depend on the value of k in I. When we applied the premise with $k = n$, the grade term m_k was interpreted under $I[k \mapsto n]$, while the effect annotation selected the meta-level family member $\epsilon_I(n)$. For the next iteration, we will instantiate with $k = n - 1$. Because p does not mention k, the fact that $\sigma_1 \vDash_I p$ holds regardless of how we re-instantiate k. This is what makes the induction work: p is the *stable* invariant, $e_v = k$ is the *snapshot* binding, and $\epsilon_I(k)$ is the *evolving* meta-level effect family selected at stage k.

By induction hypothesis on $n - 1$: σ_1 satisfies $p \wedge e_v = n - 1$ with effect $\epsilon_I(n-1)$, so the remaining loop produces σ' with:

(i) $\sigma' \vDash_I p \wedge \neg b$

(ii) $\sigma' \vDash_I \epsilon_I(0)$

(iii) $m'' \leq [\![\sum_{k=1}^{n-1} m_k]\!]_i(I)$

Total grade: $m' = m'_1 + m'' \leq [\![m_k]\!]_i(I[k \mapsto n]) + [\![\sum_{k=1}^{n-1} m_k]\!]_i(I) = [\![\sum_{k=1}^{n} m_k]\!]_i(I)$.

Note: the biconditional $\Longleftrightarrow$ is essential. The contrapositive of the ($\Longleftarrow$) direction guarantees that b is true whenever $e_v > 0$, so the loop takes exactly n iterations. The effect $\epsilon_I(k)$ is a meta-level indexed family with binder-scoped substitution, not ordinary free-variable capture by k.

□

4 Unary CHC Translation

This section gives a unary CHC warm-up for the logic of Section 2. It is placed after the unary logic sections, before the relational Level 1 system, so the single-run CHC mechanism is in place before the relational levels. It is not part of the Level 1 relational rule system in Section 5; the construction below explains how Hoare-rule structure can be encoded as CHC obligations for a single run.

This warm-up is intentionally not the full bridge to the PCSAT artifacts. The implementation-facing bridge also needs cell-refocus choices, witness predicates, HIT/MISS case splits, prophecy counters, and example-specific invariants. Those ingredients are introduced later through the Level 2 CountPositive derivation and the relational CHC translation in Section 10.6. Thus Section 4 should be read as notation and mechanism for CHC generation, while the later examples make the PCSAT-to-logic connection explicit.

We sketch a translation scheme from logic-rule structure to CHC constraints, written $T(C, l, l')$ where l, l' are the entry and exit labels and V represents the program variables used. *The clauses in this warm-up are **schematic pseudocode** illustrating the derivation-to-CHC idea, not a formal definition; some displays elide argument lists or join labels. The finalized, formal translation with exact clause signatures is the Level 2 read-only-array translation of Section 10.6.*

4.0.1 Cell morphing, no β, no grade

$T(x = e, l, l + 1) =$

$$(I_l(V) \wedge x' = e \Rightarrow I_{l+1}(V[x'/x]))$$

$T(C_1; C_2, l, l_2) =$

$$\begin{array}{l} (T(C_1, l, l_1)) \\ T(C_2, l_1, l_2) \end{array}$$

$T(\texttt{ if } b \texttt{ then } C_1 \texttt{ else } C_2, l, l_{join}) =$

$$\begin{array}{l} I_l(V) \wedge b \Rightarrow I_{thenl}(V') \\ I_l(V) \wedge \neg b \Rightarrow I_{elsel}(V') \\ T(C_1, thenl, joinl) \\ T(C_2, elsel, joinl) \end{array}$$

$T(\texttt{ while } b \texttt{ do } C, l, exitl) =$

$$\begin{array}{l} I_{entryl}(V) \Rightarrow inv_l(V) \\ inv_l(V) \wedge b \Rightarrow I_{bodyl}(V') \\ T(C, bodyl, bendl) \\ I_{bendl}(V) \Rightarrow inv_l(V) \\ inv_l(V) \wedge \neg b \Rightarrow I_{exitl}(V) \end{array}$$

$T(x = a[i], l, l + 1) =$

$$\begin{array}{l} I_l(V, x, i, k, ak) \wedge I_l(V, x, i, i, ai) \wedge i \neq k \Rightarrow I_{l+1}(V, ai, i, k, ak) \\ I_l(V, x, i, i, ai) \Rightarrow I_{l+1}(V, ai, i, i, ai) \end{array}$$

$T(a[i] = e, l, l+1) =$

$$I_l(V, i, k, ak) \wedge i \neq k \Rightarrow I_{l+1}(V, i, k, ak)$$
$$I_l(V, i, i, ai) \Rightarrow I_{l+1}(V, i, i, e)$$

4.0.2 Cell morphing, no β, with grade

$T(x = e, l, l+1, (L, U), (L+1, U+1)) =$

$$(I_l(V, (L, U)) \wedge x' = e \Rightarrow I_{l+1}(V[x'/x], (L+1, U+1)))$$

$T(C_1; C_2, l, l_2, (L, U), (L_2, U_2)) =$

$$(T(C_1, l, l_1, (L, U), (L_1, U_1)))$$
$$T(C_2, l_1, l_2, (L_1, U_1), (L_2, U_2))$$

$T($ `if` b `then` C_1 `else` $C_2, l, l_{join}, (L, U), (L_1, U_1)) =$

$$I_l(V, (L, U)) \wedge b \Rightarrow I_{thenl}(V', (L, U))$$
$$I_l(V, (L, U)) \wedge \neg b \Rightarrow I_{elsel}(V', (L, U))$$
$$T(C_1, thenl, joinl, (L, U), (L_1, U_1))$$
$$T(C_2, elsel, joinl, (L, U), (L_1, U_1))$$

$T($ `while` b `do` $C, l, exitl, (L, U), (L_{exit}, U_{exit})) =$

$$I_{entryl}(V, (L, U)) \Rightarrow inv_l(V, (L, U))$$
$$inv_l(V, (L, U)) \wedge b \Rightarrow I_{bodyl}(V', (L, U))$$
$$T(C, bodyl, bendl, (L, U), (L', U'))$$
$$I_{bendl}(V, (L', U')) \Rightarrow inv_l(V, (L', U'))$$
$$inv_l(V, (L_{exit}, U_{exit})) \wedge \neg b \Rightarrow I_{exitl}(V, (L_{exit}, U_{exit}))$$

$T(x = a[i], l, l+1, (L, U), (L + cost_{read}, U + cost_{read})) =$

$$I_l(V, x, i, k, ak, (L, U)) \wedge I_l(V, x, i, i, ai, (\_, \_)) \wedge i \neq k$$
$$\Rightarrow I_{l+1}(V, ai, i, k, ak, (L + cost_{read}, U + cost_{read}))$$
$$I_l(V, x, i, i, ai, (L, U)) \Rightarrow I_{l+1}(V, ai, i, i, ai, (L + cost_{read}, U + cost_{read}))$$

$T(a[i] = e, l, l+1, (L, U), (L + cost_{updt}, U + cost_{updt})) =$

$$I_l(V, i, k, ak, (L, U)) \wedge i \neq k \Rightarrow I_{l+1}(V, i, k, ak, (L + cost_{updt}, U + cost_{updt}))$$
$$I_l(V, i, i, ai, (L, U)) \Rightarrow I_{l+1}(V, i, i, e, (L + cost_{updt}, U + cost_{updt}))$$

5 Level 1: Relational Logic (No Grade)

This section presents the relational logic without grades, corresponding to **Level 1** of the relational cell morphing framework. Level 1 covers *ungraded* relational properties, including monotonicity, noninterference, order preservation, and arithmetic robustness bounds stated directly in P and Q, where the relationship between two runs is tracked entirely through the assertion P and the heap relation δ, with no grade or execution-cost accounting.

The relational judgment has the form:

$$\vdash \{\delta\} \mid \{P\}\ C_1 \sim C_2\ \{Q\} \mid \{\delta'\}$$

where $\delta = (u_1, u_2) \mapsto^{\Phi} \beta$ specifies that arrays u_1, u_2 are related by Φ at all positions *in* β: for every $i \in \beta$, $\Phi(u_1[i], u_2[i])$ holds. Positions outside β may have arbitrary (unrelated) values. The assertion P (resp. Q) is a relational precondition (resp. postcondition) over program variables from both runs, using the notation $x\langle 1\rangle, x\langle 2\rangle$.

Connection to Level 1 cell morphing. The CHC encodings for Level 1 can be read as finite focused-cell proof obligations: they track a fixed finite set of distinguished cells (often a singleton k) and their witnessed values, without Level 2 PickK/prophecy-grade accounting. The logic's β plays the analogous role of the semantic tracked domain. The relational **read1**/**read2** rules support the same style of HIT/MISS case split: a **read1** step may use the relation Φ at positions in β, while a **read2** step intentionally learns no pointwise relation outside β. Thus examples that need correlated MISS behavior, asynchronous schedulers, or multi-cell summaries must state the needed fact as an explicit relational invariant, bridge lemma, or **re-sync** recovery obligation. The PCSAT artifacts automate these obligations in the encoding, but they are not themselves a proof of a logic derivation until the corresponding bridge lemmas are stated. No grade is needed because the properties are ungraded: we prove output relations such as $c\langle 1\rangle \le c\langle 2\rangle$ or numeric sensitivity bounds in Q, not semantic execution-cost bounds like $m_1 - m_2 \le n - |\beta|$.

Binary-property weakening and update notation. The structural rule **conseq** uses the order $\delta_a \le \delta_b$ on binary properties. It is semantic weakening:

$$P \vdash \delta_a \le \delta_b \quad \text{iff} \quad \forall \sigma_1, \sigma_2, I.\ (\sigma_1, \sigma_2) \vDash_I^r P \wedge (\sigma_1, \sigma_2) \vDash_I \delta_b \Rightarrow (\sigma_1, \sigma_2) \vDash_I \delta_a.$$

For must properties, $(u_1, u_2) \mapsto^{\Phi} \beta_a \le (u_1, u_2) \mapsto^{\Phi} \beta_b$ when $[\![\beta_a]\!]_b(I) \subseteq [\![\beta_b]\!]_b(I)$; $\star$ is ordered componentwise.

Substitution into β or δ is admissible only when the substituted term is *interpretation-level* (its denotation reads only the interpretation I, never the program store or heap), per the global pure-index convention in Section 6. Heap reads and array updates therefore do not get substituted into β below. For one-sided array writes, $\delta \ominus_{u_r} \{j\}$ removes the pre-state index j from every atomic

binary component whose run-r array is u:

$$\begin{aligned}
emp \ominus_{u_r} \{j\} &= emp, \\
((u_1,u_2) \mapsto^{\Phi} \beta) \ominus_{u_1\langle 1\rangle} \{j\} &= (u_1,u_2) \mapsto^{\Phi} (\beta - \{j\}), \\
((u_1,u_2) \mapsto^{\Phi} \beta) \ominus_{u_2\langle 2\rangle} \{j\} &= (u_1,u_2) \mapsto^{\Phi} (\beta - \{j\}), \\
((u_1,u_2) \mapsto^{\Phi} \beta) \ominus_{v_r} \{j\} &= (u_1,u_2) \mapsto^{\Phi} \beta \quad \text{when } v_r \notin \{u_1\langle 1\rangle, u_2\langle 2\rangle\}, \\
(\delta_1 \star \delta_2) \ominus_{u_r} \{j\} &= (\delta_1 \ominus_{u_r} \{j\}) \star (\delta_2 \ominus_{u_r} \{j\}).
\end{aligned}$$

5.1 Structural rules

$$\frac{}{\vdash \{\delta\} \mid \{\bot\}\ C_1 \sim C_2\ \{P\} \mid \{\delta\}}\ \textbf{false}$$

$$\frac{\begin{array}{c}\vdash \{\delta_1'\} \mid \{P'\}\ C_1 \sim C_2\ \{Q'\} \mid \{\delta_2'\} \qquad \vDash P \Rightarrow P' \qquad \vDash Q' \Rightarrow Q \\ P \vdash \delta_1' \le \delta_1 \qquad Q \vdash \delta_2 \le \delta_2'\end{array}}{\vdash \{\delta_1\} \mid \{P\}\ C_1 \sim C_2\ \{Q\} \mid \{\delta_2\}}\ \textbf{conseq}$$

$$\frac{\begin{array}{c}\vdash^{m_1} \{p_1 \mid \{u_1 \mapsto^{\phi_1} \beta\}\}\ C_1\ \{q_1 \mid \{u_1 \mapsto^{\phi_1} \beta'\}\} \\ \vdash^{m_2} \{p_2 \mid \{u_2 \mapsto^{\phi_2} \beta\}\}\ C_2\ \{q_2 \mid \{u_2 \mapsto^{\phi_2} \beta'\}\} \\ \forall x,y.\Phi(x,y) \implies \phi_1(x) \wedge \phi_2(y) \\ \forall x,y.\phi_1(x) \wedge \phi_2(y) \implies \Phi(x,y)\end{array}}{\vdash \{(u_1,u_2) \mapsto^{\Phi} \beta\} \mid \{p_1\langle 1\rangle \wedge p_2\langle 2\rangle\}\ C_1 \sim C_2\ \{q_1\langle 1\rangle \wedge q_2\langle 2\rangle\} \mid \{(u_1,u_2) \mapsto^{\Phi} \beta'\}}\ \textbf{u2binary-derived}$$

$$\frac{\begin{array}{c}\vdash \{\delta_1\} \mid \{P\}\ C_1 \sim C_2\ \{Q\} \mid \{\delta_2\} \qquad FV(\delta) \cap (mod(C_1)\langle 1\rangle \cup mod(C_2)\langle 2\rangle) = \emptyset \\ arrFV(\delta) \cap (arrfoot(C_1)\langle 1\rangle \cup arrfoot(C_2)\langle 2\rangle) = \emptyset \\ arrfoot(C_i)\langle i\rangle \subseteq own_i(\delta_1) \quad (i \in \{1,2\}) \\ local_{\delta,\delta_1}(P) \text{ and } local_{\delta,\delta_2}(Q) \quad \text{(Definitions 6.4, 6.5)}\end{array}}{\vdash \{\delta \star \delta_1\} \mid \{P\}\ C_1 \sim C_2\ \{Q\} \mid \{\delta \star \delta_2\}}\ \textbf{heap-frame}$$

$$\frac{\begin{array}{c}\vdash \{\delta_1\} \mid \{P\}\ C_1 \sim C_2\ \{Q\} \mid \{\delta_2\} \\ FV(R) \cap (mod(C_1)\langle 1\rangle \cup mod(C_2)\langle 2\rangle) = \emptyset \\ arrFV(R) \cap (arrvars(C_1)\langle 1\rangle \cup arrvars(C_2)\langle 2\rangle) = \emptyset\end{array}}{\vdash \{\delta_1\} \mid \{P \wedge R\}\ C_1 \sim C_2\ \{Q \wedge R\} \mid \{\delta_2\}}\ \textbf{frame}$$

$$\frac{\vdash \{\delta\} \mid \{P\}\ C_1 \sim C_2\ \{R\} \mid \{\delta_1\} \qquad \vdash \{\delta_1\} \mid \{R\}\ C_1' \sim C_2'\ \{Q\} \mid \{\delta'\}}{\vdash \{\delta\} \mid \{P\}\ C_1; C_1' \sim C_2; C_2'\ \{Q\} \mid \{\delta'\}}\ \textbf{sequence}$$

The rule **u2binary-derived** is a convenience bridge, not a primitive needed for the core relational semantics: it packages two unary derivations as a binary judgment when the binary relation is equivalent to the conjunction of the two unary element predicates. It is useful for reusing two unary proofs when the desired relational fact is exactly decomposable into separate unary facts about each run. It is not the intended proof pattern for examples whose key step is branch agreement, correlated reads, or a relational invariant; those examples should use the relational read, conditional, frame, and heap-frame rules directly. By contrast, **heap-frame** is a core structural rule for multi-array heap relations; it composes an unaffected heap component with the component being proved.

5.2 Two-size rules

$$\frac{}{\vdash \{\delta\} \mid \{P\}\ \texttt{skip} \sim \texttt{skip}\ \{P\} \mid \{\delta\}}\ \textbf{skip}$$

$$\frac{x_1\langle 1\rangle \notin FV(\delta) \qquad x_2\langle 2\rangle \notin FV(\delta)}{\vdash \{\delta[e_1\langle 1\rangle/x_1\langle 1\rangle][e_2\langle 2\rangle/x_2\langle 2\rangle]\} \mid \{Q[e_1\langle 1\rangle/x_1\langle 1\rangle][e_2\langle 2\rangle/x_2\langle 2\rangle]\}\ x_1 := e_1 \sim x_2 := e_2\ \{Q\} \mid \{\delta\}}\ \textbf{assign}$$

$$\frac{\begin{array}{c} n \geq 0 \qquad k \notin FV(P) \cup FV(b_1) \cup FV(b_2) \cup FV(e_v) \cup FV(\delta) \\ P \implies 0 \leq e_v\langle 1\rangle \\ P \implies e_v\langle 1\rangle \leq 0 \iff \neg b_1\langle 1\rangle \\ \forall k.\ \vdash \{\delta\} \mid \{P \wedge b_1\langle 1\rangle \wedge b_2\langle 2\rangle \wedge e_v\langle 1\rangle = k \wedge 0 < e_v\langle 1\rangle \leq n\}\ C_1 \sim C_2\ \{P \wedge b_1\langle 1\rangle = b_2\langle 2\rangle \wedge e_v\langle 1\rangle < k\} \mid \{\delta\} \end{array}}{\vdash \{\delta\} \mid \{P \wedge b_1\langle 1\rangle = b_2\langle 2\rangle \wedge e_v\langle 1\rangle \leq n\}\ \texttt{while}\ b_1\ \texttt{do}\ C_1 \sim\ \texttt{while}\ b_2\ \texttt{do}\ C_2\ \{P \wedge \neg b_1\langle 1\rangle \wedge \neg b_2\langle 2\rangle\} \mid \{\delta\}}\ \textbf{awhile}$$

$$\frac{\begin{array}{c} n \geq 0 \qquad k \notin FV(P) \cup FV(b_1) \cup FV(b_2) \cup FV(e_v) \\ P \implies 0 \leq e_v\langle 1\rangle \\ P \implies e_v\langle 1\rangle \leq 0 \iff \neg b_1\langle 1\rangle \\ \forall k.\ 1 \leq k \leq n \implies \vdash \{\delta_k\} \mid \{P \wedge b_1\langle 1\rangle \wedge b_2\langle 2\rangle \wedge e_v\langle 1\rangle = k\}\ C_1 \sim C_2\ \{P \wedge b_1\langle 1\rangle = b_2\langle 2\rangle \wedge e_v\langle 1\rangle = k-1\} \mid \{\delta_{k-1}\} \end{array}}{\vdash \{\delta_n\} \mid \{P \wedge b_1\langle 1\rangle = b_2\langle 2\rangle \wedge e_v\langle 1\rangle = n\}\ \texttt{while}\ b_1\ \texttt{do}\ C_1 \sim\ \texttt{while}\ b_2\ \texttt{do}\ C_2\ \{P \wedge \neg b_1\langle 1\rangle \wedge \neg b_2\langle 2\rangle\} \mid \{\delta_0\}}\ \textbf{awhile-evolving}$$

$$\frac{\begin{array}{c} \vDash P \implies b_1\langle 1\rangle = b_2\langle 2\rangle \\ \vdash \{\delta\} \mid \{P \wedge b_1\langle 1\rangle\}\ C_1 \sim C_2\ \{Q\} \mid \{\delta'\} \qquad \vdash \{\delta\} \mid \{P \wedge \neg b_1\langle 1\rangle\}\ C_1' \sim C_2'\ \{Q\} \mid \{\delta'\} \end{array}}{\vdash \{\delta\} \mid \{P\}\ \texttt{if}\ b_1\ \texttt{then}\ C_1\ \texttt{else}\ C_1' \sim\ \texttt{if}\ b_2\ \texttt{then}\ C_2\ \texttt{else}\ C_2'\ \{Q\} \mid \{\delta'\}}\ \textbf{conditional}$$

$$\frac{\begin{array}{c} j \text{ fresh} \\ P = Q[u_1[e_1\langle 1\rangle \mapsto e_1'\langle 1\rangle]/u_1][u_2[e_2\langle 2\rangle \mapsto e_2'\langle 2\rangle]/u_2] \\ P \implies e_1\langle 1\rangle = e_2\langle 2\rangle = j \\ P \implies 0 \leq j < \min(len(u_1)\langle 1\rangle, len(u_2)\langle 2\rangle) \\ P \implies \Phi(e_1'\langle 1\rangle, e_2'\langle 2\rangle) \end{array}}{\vdash \{(u_1,u_2) \mapsto^{\Phi} \beta\} \mid \{P\}\ u_1[e_1] := e_1' \sim u_2[e_2] := e_2'\ \{Q\} \mid \{(u_1,u_2) \mapsto^{\Phi} \beta \cup \{j\}\}}\ \textbf{update1}$$

$$\frac{\begin{array}{c} j \text{ fresh} \\ P = Q[u_1[e_1\langle 1\rangle \mapsto e_1'\langle 1\rangle]/u_1][u_2[e_2\langle 2\rangle \mapsto e_2'\langle 2\rangle]/u_2] \\ P \implies e_1\langle 1\rangle = e_2\langle 2\rangle = j \\ P \implies 0 \leq j < \min(len(u_1)\langle 1\rangle, len(u_2)\langle 2\rangle) \end{array}}{\vdash \{(u_1,u_2) \mapsto^{\Phi} \beta\} \mid \{P\}\ u_1[e_1] := e_1' \sim u_2[e_2] := e_2'\ \{Q\} \mid \{(u_1,u_2) \mapsto^{\Phi} \beta - \{j\}\}}\ \textbf{update2}$$

$$\frac{\begin{array}{c} j \text{ fresh} \qquad x_1, x_2 \notin FV(\beta) \\ P = Q[u_1[e_1]/x_1\langle 1\rangle][u_2[e_2]/x_2\langle 2\rangle] \qquad P \implies e_1\langle 1\rangle = e_2\langle 2\rangle = j \wedge j \in \beta \end{array}}{\vdash \{(u_1,u_2) \mapsto^{\Phi} \beta\} \mid \{P\}\ x_1 := u_1[e_1] \sim x_2 := u_2[e_2]\ \{Q \wedge \Phi(x_1,x_2)\} \mid \{(u_1,u_2) \mapsto^{\Phi} \beta\}}\ \textbf{read1}$$

$$\frac{\begin{array}{c} j \text{ fresh} \qquad x_1, x_2 \notin FV(\beta) \\ P = Q[u_1[e_1]/x_1\langle 1\rangle][u_2[e_2]/x_2\langle 2\rangle] \qquad P \implies e_1\langle 1\rangle = e_2\langle 2\rangle = j \end{array}}{\vdash \{(u_1,u_2) \mapsto^{\Phi} \beta\} \mid \{P\}\ x_1 := u_1[e_1] \sim x_2 := u_2[e_2]\ \{Q\} \mid \{(u_1,u_2) \mapsto^{\Phi} \beta\}}\ \textbf{read2}$$

Allocation rules (framed). The allocation rules add a fresh array relation to an existing frame δ:

$$\mathit{FreshAlloc}(Q, \delta, u_1, u_2) \equiv \mathit{arrFV}(Q) \cap U = \emptyset \wedge \mathit{arrFV}(\delta) \cap U = \emptyset, \quad U = \{u_1\langle 1\rangle, u_2\langle 2\rangle\}.$$

alloc1 (tracked equal-length allocation).

$$\frac{\begin{array}{c} P \implies Q \\ \mathit{FreshAlloc}(Q, \delta, u_1, u_2) \\ \ell \text{ fresh logical index} \quad \ell \notin FV(Q) \cup FV(\delta) \\ P \implies e_1\langle 1\rangle = \ell \wedge e_2\langle 2\rangle = \ell \wedge 0 \le \ell \\ P \implies \Phi(e_1'\langle 1\rangle, e_2'\langle 2\rangle) \end{array}}{\vdash \{\delta\} \mid \{P\}\ u_1 \leftarrow \mathit{array}(e_1)(e_1')\ \sim\ u_2 \leftarrow \mathit{array}(e_2)(e_2')\ \{Q\} \mid \{\delta \star ((u_1, u_2) \mapsto^{\Phi} [0, \ell - 1])\}}$$

Here ℓ is a ghost snapshot chosen for this rule instance: it may be constrained by the precondition obligation above, but it is not a program variable and no program expression appears inside the β component. The rule intentionally covers the tracked equal-length case; unequal or untracked fresh allocations use **alloc2** or a smaller tracked domain. By the interval semantics, $[0, \ell - 1]$ denotes $\emptyset$ when $\ell = 0$.

alloc2 (fresh but untracked allocation).

$$\frac{\begin{array}{c} P \implies Q \\ \mathit{FreshAlloc}(Q, \delta, u_1, u_2) \\ P \implies 0 \le e_1\langle 1\rangle \wedge 0 \le e_2\langle 2\rangle \end{array}}{\vdash \{\delta\} \mid \{P\}\ u_1 \leftarrow \mathit{array}(e_1)(e_1')\ \sim\ u_2 \leftarrow \mathit{array}(e_2)(e_2')\ \{Q\} \mid \{\delta \star ((u_1, u_2) \mapsto^{\Phi} \emptyset)\}}$$

In **alloc2**, the choice of Φ is immaterial because the tracked domain is empty.

For the fixed **awhile** rule, the binary property δ is a fixed effect annotation; the side condition $k \notin FV(\delta)$ prevents the snapshot binder from changing the meaning of that fixed effect. For **awhile-evolving**, δ_k (equivalently, δ_k in the prose) is a meta-level indexed family of effects. Instantiating the premise at a stage k selects the corresponding family member; it is not ordinary free-variable capture. The fixed rule is the special case where $\delta_k = \delta$ for every stage k.

5.3 One-size rules

We present left-sided rules below. Right-sided variants (e.g., **assignR-skipL**, **conditionalR**, **whileR-skipL**, **readR**, **updateR**) are symmetric and omitted.

$$\frac{x_1\langle 1\rangle \notin FV(\delta)}{\vdash \{\delta[e_1\langle 1\rangle/x_1\langle 1\rangle]\} \mid \{Q[e_1\langle 1\rangle/x_1\langle 1\rangle]\}\ x_1 := e_1 \sim \texttt{skip}\ \{Q\} \mid \{\delta\}}\ \textbf{assignL-skipR}$$

$$\frac{x_1\langle 1\rangle \notin FV(\delta) \qquad \vdash \{\delta\} \mid \{P\}\ \texttt{skip} \sim C_2\ \{Q\} \mid \{\delta'\}}{\vdash \{\delta[e_1\langle 1\rangle/x_1\langle 1\rangle]\} \mid \{P[e_1\langle 1\rangle/x_1\langle 1\rangle]\}\ x_1 := e_1 \sim C_2\ \{Q\} \mid \{\delta'\}}\ \textbf{assignL}$$

$$\frac{\vdash \{\delta\} \mid \{P \wedge b_1\langle 1\rangle\}\ C_1 \sim C_2\ \{Q\} \mid \{\delta'\} \qquad \vdash \{\delta\} \mid \{P \wedge \neg b_1\langle 1\rangle\}\ C_1' \sim C_2\ \{Q\} \mid \{\delta'\}}{\vdash \{\delta\} \mid \{P\}\ \ \texttt{if}\ b_1\ \texttt{then}\ C_1\ \texttt{else}\ C_1' \sim C_2\ \{Q\} \mid \{\delta'\}}\ \textbf{conditionalL}$$

$$\frac{\vdash \{\delta\} \mid \{P \wedge b_1\langle 1\rangle\}\ C_1 \sim \texttt{skip}\ \{P\} \mid \{\delta\}}{\vdash \{\delta\} \mid \{P\}\ \ \texttt{while}\ b_1\ \texttt{do}\ C_1 \sim \texttt{skip}\ \{P \wedge \neg b_1\langle 1\rangle\} \mid \{\delta\}}\ \textbf{whileL-skipR}$$

$$\frac{x_1\langle 1\rangle \notin FV(\delta)}{\vdash \{\delta\} \mid \{Q[u_1[e_1\langle 1\rangle]/x_1\langle 1\rangle]\}\ x_1 := u_1[e_1] \sim \texttt{skip}\ \{Q\} \mid \{\delta\}}\ \textbf{readL-skipR}$$

$$\frac{x_1\langle 1\rangle \notin FV(\delta) \vdash \{\delta\} \mid \{P\}\ \texttt{skip} \sim C_2\ \{Q\} \mid \{\delta'\}}{\vdash \{\delta\} \mid \{P[u_1[e_1\langle 1\rangle]/x_1\langle 1\rangle]\}\ x_1 := u_1[e_1] \sim C_2\ \{Q\} \mid \{\delta'\}}\ \textbf{readL}$$

$$\frac{\begin{array}{c} j \text{ fresh} \\ Q[u_1[e_1\langle 1\rangle \mapsto e_1'\langle 1\rangle]/u_1] \implies e_1\langle 1\rangle = j \\ Q[u_1[e_1\langle 1\rangle \mapsto e_1'\langle 1\rangle]/u_1] \implies 0 \le j < len(u_1)\langle 1\rangle \end{array}}{\vdash \{\delta\} \mid \{Q[u_1[e_1\langle 1\rangle \mapsto e_1'\langle 1\rangle]/u_1]\}\ u_1[e_1] := e_1' \sim \texttt{skip}\ \{Q\} \mid \{\delta \ominus_{u_1\langle 1\rangle} \{j\}\}}\ \textbf{updateL-skipR}$$

$$\frac{\begin{array}{c} j \text{ fresh} \\ P[u_1[e_1\langle 1\rangle \mapsto e_1'\langle 1\rangle]/u_1] \implies e_1\langle 1\rangle = j \\ P[u_1[e_1\langle 1\rangle \mapsto e_1'\langle 1\rangle]/u_1] \implies 0 \le j < len(u_1)\langle 1\rangle \\ \vdash \{\delta \ominus_{u_1\langle 1\rangle} \{j\}\} \mid \{P\}\ \texttt{skip} \sim C_2\ \{Q\} \mid \{\delta'\} \end{array}}{\vdash \{\delta\} \mid \{P[u_1[e_1\langle 1\rangle \mapsto e_1'\langle 1\rangle]/u_1]\}\ u_1[e_1] := e_1' \sim C_2\ \{Q\} \mid \{\delta'\}}\ \textbf{updateL}$$

Derived re-sync recovery principle. When two runs execute independently (e.g., async inner loops with different iteration counts), one-sided rules like **updateL-skipR** may shrink β. Re-sync is not a local primitive rule: it is a derived recovery principle that applies after two one-sided derivations, provided a separate recovery lemma proves the desired relation on the original index set.

This principle is inspired by ARel's R-S rule [2], but the extra feature is β-recovery: ARel's R-S switches from relational to unary reasoning, whereas re-sync allows returning to lockstep only after an explicit recovery proof.

The strictness is deliberate. A proof should use **re-sync-derived** only when later lockstep reasoning needs a relation on the original tracked domain β_0 after one-sided updates have discarded part of that domain. If the remaining proof can proceed over the residual domain, or if the invariant never loses the needed β facts, no re-sync step is required. The current manuscript uses this pattern

for the insertion-sort/asynchronous-recovery style of argument. A legacy bridge inventory found 21 Level 1 examples: 14 use fixed **awhile**-style lockstep reasoning, 0 use evolving effects, and 7 fall into async/re-sync/other patterns. That count is evidence for organizing the examples, not a manuscript theorem and not an audited claim that exactly seven examples must use re-sync.

We use the following abbreviations for the two semantic obligations:

$$\begin{aligned} \mathit{Recover}(P', u_1, u_2, \beta_0, \Phi_{rec}) &\equiv \vDash P' \implies \forall j \in \beta_0.\ \Phi_{rec}(u_1[j]\langle 1\rangle, u_2[j]\langle 2\rangle), \\ \mathit{PreserveLen}(C_1, C_2, u_1, u_2, \beta_0) &\equiv \text{post-state array existence + bounds.} \end{aligned}$$

Here *Recover* is a semantic side condition, not a new assertion constructor and not a CHC-generated obligation. It is read over the final paired states produced by the two one-sided executions: array reads such as $u_1[j]\langle 1\rangle$ and $u_2[j]\langle 2\rangle$ are interpreted in the post-state heaps, whose array-domain and index-bound facts are supplied by *PreserveLen*. The second abbreviation is intentionally named rather than expanded inside the rule display: semantically, it quantifies over the terminating executions of C_1 from the left pre-state and C_2 from the right pre-state covered by the preceding one-sided premises, requires u_1 and u_2 to remain allocated in the two post-state heaps, and requires $j < \mathit{len}(\sigma_1'(u_1))$ and $j < \mathit{len}(\sigma_2'(u_2))$ for every $j \in \beta_0$. The allocation part is non-vacuous even when $\beta_0 = \emptyset$, because Definition 6.2 still requires the two array names to be in the post-state heap domains. In applications this obligation is discharged by a syntactic length-preservation argument or by the same domain lemma that proves recovery.

$$\frac{\begin{array}{c} \vdash \{(u_1, u_2) \mapsto^{\Phi} \beta_0\} \mid \{P\}\ C_1 \sim \texttt{skip}\ \{P_{mid}\} \mid \{\delta_{mid}\} \\ \vdash \{\delta_{mid}\} \mid \{P_{mid}\}\ \texttt{skip} \sim C_2\ \{P'\} \mid \{\delta_{out}\} \\ \mathit{Recover}(P', u_1, u_2, \beta_0, \Phi_{rec}) \\ \mathit{PreserveLen}(C_1, C_2, u_1, u_2, \beta_0) \quad \Phi_{rec} \text{ well-formed} \end{array}}{\vdash \{(u_1, u_2) \mapsto^{\Phi} \beta_0\} \mid \{P\}\ C_1 \sim C_2\ \{P'\} \mid \{(u_1, u_2) \mapsto^{\Phi_{rec}} \beta_0\}}\ \textbf{re-sync-derived}$$

The first two premises establish partial correctness of each run independently (via one-sided rules; β may shrink to an arbitrary residual δ_{out}). This derived principle intentionally concludes only the selected recovered relation on β_0; it does not assert δ_{out} in the conclusion. If later reasoning needs residual facts from δ_{out}, they must be included by an additional framing or consequence step. The recovery obligation proves the target relation over all of β_0, not merely over positions that survived the one-sided updates. PreserveLen ensures those indices still denote cells in the post-state arrays.

*Soundness intuition; full proof in the **re-sync-derived** case of Theorem 6.3.* Since C_1 and C_2 execute independently (Definition 6.6), the composition $C_1 \sim \texttt{skip}; \texttt{skip} \sim C_2$ produces the same final state (σ_1', σ_2') as $C_1 \sim C_2$. The one-sided premises give $(\sigma_1', \sigma_2') \vDash_I P'$. The recovery obligation gives $\forall j \in \beta_0.\ \Phi_{rec}(h_1'(u_1)[j], h_2'(u_2)[j])$, and PreserveLen makes each such read defined. By Definition 6.2, $(\sigma_1', \sigma_2') \vDash_I (u_1, u_2) \mapsto^{\Phi_{rec}} \beta_0$, so the recovered postcondition holds. □

Insertion-sort robustness derivation skeleton. This example uses the standard in-place insertion-sort command, shown schematically at the command-structure level:

$$
\begin{array}{l}
i := 1; \\
\texttt{while } i < n \texttt{ do} \\
\quad key := A[i]; \quad j := i - 1; \\
\quad \texttt{while } j \geq 0 \wedge A[j] > key \texttt{ do} \\
\quad\quad A[j+1] := A[j]; \quad j := j - 1; \\
\quad \textbf{done}; \\
\quad A[j+1] := key; \quad i := i + 1 \\
\textbf{done}.
\end{array}
$$

The relational robustness property is not the unary sorting-correctness theorem. It is the following stability statement for the two executions of the same command on arrays A_1 and A_2. Let $\beta_0 = [0, n)$, where n is a logical snapshot of the common input length, and let $\Phi_\varepsilon(x, y) \equiv |x - y| \leq \varepsilon$. From the precondition

$$P_{pre}(n) \equiv n \geq 0 \wedge len(A_1)\langle 1\rangle = n \wedge len(A_2)\langle 2\rangle = n$$

and the initial binary property $(A_1, A_2) \mapsto^{\Phi_\varepsilon} \beta_0$, the desired postcondition is

$$P_{post}(n) \mid (A_1, A_2) \mapsto^{\Phi_\varepsilon} \beta_0,$$

where $P_{post}(n)$ records the ordinary scalar facts needed after the sort (for example, the length facts and loop exit condition). Correctness of the sorted output can be proved separately as a unary property. The relational goal here is conditional: once the insertion-sort recovery lemma below is supplied, the final arrays can be shown pointwise ε-close over the original domain β_0.

For one outer iteration, write Ins_i for the body that extracts key, runs the inner shifting loop, writes key back, and increments i. At the start of the iteration the lockstep invariant contains $(A_1, A_2) \mapsto^{\Phi_\varepsilon} \beta_0$. The two runs agree on the outer guard and are handled by **awhile**; the difficulty is the inner shifting loop. Because the values in A_1 and A_2 may differ by up to ε, one run may shift more cells than the other. The two inner loops are therefore derived asynchronously:

$$C^1_{shift} \sim \texttt{skip} \qquad \text{and then} \qquad \texttt{skip} \sim C^2_{shift}.$$

Each one-sided array update consumes the written target index from the tracked domain. Thus the two one-sided premises can establish a semantic postcondition P'_i for the completed pair of insertion steps Ins^1_i and Ins^2_i, but their binary post-effect is only a residual δ_{out}; it is not enough to resume the outer lockstep invariant over the full β_0. The postcondition P'_i must contain the domain facts used by the recovery lemma: the processed prefixes after insertion are the sorted/order-statistic images of the corresponding pre-step prefixes, those pre-step prefixes came from arrays that were pointwise ε-close, the unprocessed suffix is unchanged, and the array lengths are preserved.

The re-sync step is therefore the explicit recovery point. The proof must supply the domain lemma

$$\mathit{Recover}(P_i', A_1, A_2, \beta_0, \Phi_\varepsilon),$$

i.e. from the semantic facts after both insertion steps have completed, every original index $j \in \beta_0$ again satisfies $|A_1[j]\langle 1\rangle - A_2[j]\langle 2\rangle| \leq \varepsilon$. For insertion sort this lemma is the domain-specific order-statistic argument: the two prefixes being reinserted come from inputs that were pointwise ε-close, while the unprocessed suffix is unchanged. Shifts and key insertion do not allocate or change array lengths, so *PreserveLen* is discharged by ordinary length preservation. Applying **re-sync-derived** with $\Phi_{rec} = \Phi_\varepsilon$ restores $(A_1, A_2) \mapsto^{\Phi_\varepsilon} \beta_0$, allowing the outer **awhile** invariant to continue. Without this recovery obligation the derivation keeps only the residual domain and proves a strictly weaker, partial-array robustness statement. See Appendix, Gap Example 2.

6 Relational Semantics and Level 1 Soundness

This section defines the common semantic foundation for the relational logics: relational assertion semantics, binary property validation, and Level 1 validity. These semantic definitions are reused by the later graded levels. Level 2 extends the same relational validity judgment with a semantic grade and states its soundness theorem in Section 8.

6.1 Relational Assertion Semantics

Relational assertions P, Q use the notation $e\langle 1\rangle$ and $e\langle 2\rangle$ to refer to expressions evaluated in the first and second run's state, respectively. Given two combined states $\sigma_1 = (s_1, h_1)$ and $\sigma_2 = (s_2, h_2)$, we define the *relational evaluation* of an expression:

$$\begin{aligned} [\![e\langle 1\rangle]\!]_r((\sigma_1, \sigma_2), I) &= [\![e]\!]_i(\sigma_1, I) \\ [\![e\langle 2\rangle]\!]_r((\sigma_1, \sigma_2), I) &= [\![e]\!]_i(\sigma_2, I) \end{aligned}$$

Definition 6.1 (Relational assertion validity)**.** *The validation of a relational assertion P over a pair of combined states (σ_1, σ_2) under interpretation I, written $(\sigma_1, \sigma_2) \vDash_I^r P$, is defined inductively:*

$(\sigma_1, \sigma_2) \vDash_I^r$ **True**		*always*
$(\sigma_1, \sigma_2) \nvDash_I^r$ **False**		*always*
$(\sigma_1, \sigma_2) \vDash_I^r e_1\langle j_1\rangle \oplus e_2\langle j_2\rangle$	*if*	$[\![e_1]\!]_i(\sigma_{j_1}, I) \oplus [\![e_2]\!]_i(\sigma_{j_2}, I)$
$(\sigma_1, \sigma_2) \vDash_I^r P_1 \wedge P_2$	*if*	$(\sigma_1, \sigma_2) \vDash_I^r P_1$ *and* $(\sigma_1, \sigma_2) \vDash_I^r P_2$
$(\sigma_1, \sigma_2) \vDash_I^r P_1 \vee P_2$	*if*	$(\sigma_1, \sigma_2) \vDash_I^r P_1$ *or* $(\sigma_1, \sigma_2) \vDash_I^r P_2$
$(\sigma_1, \sigma_2) \vDash_I^r P_1 \implies P_2$	*if*	$(\sigma_1, \sigma_2) \nvDash_I^r P_1$ *or* $(\sigma_1, \sigma_2) \vDash_I^r P_2$
$(\sigma_1, \sigma_2) \vDash_I^r \neg P$	*if*	$(\sigma_1, \sigma_2) \nvDash_I^r P$
$(\sigma_1, \sigma_2) \vDash_I^r \forall i{::}S.\, P$	*if*	$\forall k \in \mathsf{S}.\, (\sigma_1, \sigma_2) \vDash_{I[i\to k]}^r P$
$(\sigma_1, \sigma_2) \vDash_I^r \exists i{::}S.\, P$	*if*	$\exists k \in \mathsf{S}.\, (\sigma_1, \sigma_2) \vDash_{I[i\to k]}^r P$
$(\sigma_1, \sigma_2) \vDash_I^r J \in \beta$	*if*	$[\![J]\!]_j(I) \in [\![\beta]\!]_b(I)$ *(pure: Section 3.2)*

where $j_1, j_2 \in \{1, 2\}$ select which run's state is used for each sub-expression, and the subscript $_j$ in $[\![J]\!]_j$ denotes the index denotation function (not a run selector).

Note that β-membership is pure (depends only on interpretation I, not on program states), consistent with the unary logic (Section 3.2).

6.2 Binary Property Validation

Definition 6.2 (Binary property validity)**.** *The validation of a binary property δ over a pair of combined states $(\sigma_1, \sigma_2) = ((s_1, h_1), (s_2, h_2))$ under interpretation I, written $(\sigma_1, \sigma_2) \vDash_I \delta$, is defined as:*

$$
\begin{array}{lll}
(\sigma_1, \sigma_2) \vDash_I \mathit{emp} & \mathit{if} & h_1 = [] \wedge h_2 = [] \\
(\sigma_1, \sigma_2) \vDash_I (u_1, u_2) \mapsto^{\Phi} \beta & \mathit{if} & u_1 \in \mathit{dom}(h_1) \wedge u_2 \in \mathit{dom}(h_2) \\
 & & \wedge\ [\![\beta]\!]_b(I) \subseteq [0, \min(\mathit{len}(h_1(u_1)), \mathit{len}(h_2(u_2)))) \\
 & & \wedge\ \forall x \in [\![\beta]\!]_b(I).\, \Phi(h_1(u_1)[x],\, h_2(u_2)[x]) \\
(\sigma_1, \sigma_2) \vDash_I \delta_1 \star \delta_2 & \mathit{if} & \exists h_1^a, h_1^b, h_2^a, h_2^b. \\
 & & h_1^a \perp h_1^b \wedge h_1 = h_1^a + h_1^b \\
 & & \wedge\ h_2^a \perp h_2^b \wedge h_2 = h_2^a + h_2^b \\
 & & \wedge\ ((s_1, h_1^a), (s_2, h_2^a)) \vDash_I \delta_1 \\
 & & \wedge\ ((s_1, h_1^b), (s_2, h_2^b)) \vDash_I \delta_2
\end{array}
$$

Atomic footprint convention. An atomic binary property $(u_1, u_2) \mapsto^{\Phi} \beta$ is permissive with respect to extra heap entries in the fragments assigned to the atom: it requires $u_1 \in \mathit{dom}(h_1)$ and $u_2 \in \mathit{dom}(h_2)$, and it constrains the values of those two arrays at indices in $[\![\beta]\!]_b(I)$, but it does not require those fragments to contain only u_1 and u_2. Extra arrays in the same fragments are ignored by the atom. This does not weaken separation: in $\delta_1 \star \delta_2$, the fragments assigned to δ_1 and δ_2 must still be disjoint on each side. Thus permissiveness is local to an atom's assigned fragment; no heap entry can be shared between the two operands of $\star$ on either side. When Φ contains logical variables, they are interpreted under the same interpretation I as β and the relational assertion P.

Definition 6.3 (Well-formed value relation)**.** *A value relation is written $\Phi(a, b; \vec{\ell})$, where a, b are its two distinguished* cell-value arguments *(bound symbols,* not *program variables) and $\vec{\ell}$ are logical variables. Φ is* well formed *when it mentions only a, b, and logical variables (interpreted by I), and* no scalar program variable*. **We require every value relation in a binary property to be well formed.***

Pure-index discipline (global convention). The index terms J and index sets β occurring in an effect or binary property are *interpretation-level*: their denotations $[\![J]\!]_j(I)$ and $[\![\beta]\!]_b(I)$ are functions of the interpretation I alone, reading neither the program store nor the heap. We require every derivation to preserve

this: a rule that substitutes a term into β or into a value relation Φ (for instance the **assign** and **read** rules) is admissible only when the substituted term is itself an interpretation-level index expression, *never* a heap-reading expression such as $u_a[e]$ or $\mathit{len}(u_a)$. Consequently β-membership atoms $J \in \beta$ contribute no heap reads (used in Lemma 6.1 and the frame-locality proof), and no admissible substitution can turn a binary property heap-dependent. The Level 1 and Level 2 rule blocks rely on this single convention rather than restating it per rule.

For instance $\Phi(a, b) = (a = b)$, $(a \le b)$, or $(a = b + \ell)$ with ℓ logical. Consequently Φ's truth depends only on the two cell values supplied for a, b and on I, never on the program store; its logical variables are tracked by FLV, not FV. A scalar assignment or read can therefore falsify a binary property only through β, so protecting $FV(\beta)$ suffices for the scalar side conditions of the read rules (**read1**, **read2**); the array-update rules (**update1**, ...) maintain the value relation through their own Φ-premises at the updated cell. When we write $\Phi(x_1, x_2)$ in a rule (e.g. the **read1** postcondition), the program variables x_1, x_2 are *supplied as* the cell-value arguments a, b; this is application, not a free occurrence inside Φ. A *state-dependent* Φ mentioning live program variables is possible in principle but would require scalar freshness/preservation side conditions on every property-preserving rule; we do not pursue it, as no example needs it (future work).

The bound condition $[\![\beta]\!]_b(I) \subseteq [0, \min(\mathit{len}(h_1(u_1)), \mathit{len}(h_2(u_2))))$ ensures that every tracked position falls within both arrays. For $\star$-composition, the disjointness is per-side: h_1 splits into h_1^a, h_1^b (for δ_1, δ_2 respectively), and h_2 splits independently. The two sides of δ_1 must name disjoint heap entries from the two sides of δ_2.

Ownership vs. correlation. The $\star$ connective handles *heap ownership separation*: δ_1 and δ_2 govern disjoint arrays. Same-index correlation between arrays (e.g., "position i is equal in *both* A and B") is not expressed by $\star$ itself, but by the shared index domain and the β-intersection in the loop invariant. For example, in the DualCount derivation (Section 8.12), $\delta = \delta_A \star \delta_B$ separates ownership of arrays A and B, while the invariant tracks $|\beta_A \cap \beta_B \cap [0, i)|$ to capture their positional correlation.

Combined validation. We write $(\sigma_1, \sigma_2) \vDash_I P \mid \delta$ to mean $(\sigma_1, \sigma_2) \vDash_I^r P$ and $(\sigma_1, \sigma_2) \vDash_I \delta$.

6.3 Ownership and Frame Locality

The **heap-frame** rule (Section 5) attaches a framed binary property δ to both sides of a triple. Its soundness requires more than the syntactic footprint side conditions on the commands: because atoms are *permissive* (an atom's assigned fragment may hold extra, unnamed entries), the existential $\star$-split is not unique, and we must control which arrays the relational pre/postconditions actually

read. We isolate the arrays a binary property *owns* (pins into its fragment), as distinct from those it merely mentions inside an index set β or a value relation Φ. Throughout this subsection we write δ for the framed property and δ_a for a generic active property (the heap-frame rule instantiates δ_a with the pre-effect δ_1 for the precondition and the post-effect δ_2 for the postcondition); both range over binary properties.

Definition 6.4 (Ownership)**.** *For $j \in \{1,2\}$, the run-j* owned arrays $own_j(\delta)$ *of a binary property are*

$$own_j(emp) = \emptyset, \qquad own_j(\delta_1 \star \delta_2) = own_j(\delta_1) \cup own_j(\delta_2),$$
$$own_1((u_1,u_2) \mapsto^{\Phi} \beta) = \{u_1\}, \quad own_2((u_1,u_2) \mapsto^{\Phi} \beta) = \{u_2\}.$$

By Definition 6.2, $(\sigma_1,\sigma_2) \vDash_I \delta$ forces $own_j(\delta) \subseteq dom(h_j)$, whereas an array that appears only inside β (never as an atom head) need not lie in the fragment, and a well-formed Φ (Definition 6.3) mentions no array at all beyond the two cell values it is supplied. This rests on the purity of those positions: an index set β (and every index term J) is *interpretation-level*; its denotation $[\![\beta]\!]_b(I)$ is a function of I alone and reads no heap, while Φ enters only as a *value relation*: for a well-formed Φ it is applied to the two cell values $h_1(u_1)[x], h_2(u_2)[x]$ of the already-owned atom heads u_1, u_2 (Definition 6.2), depending solely on those two cell arguments and on logical variables interpreted under the same I. Hence neither β nor Φ forces any array beyond the heads u_1, u_2 into the fragment (cf. the note following Definition 6.2). In particular $\delta_1 \star \delta_2$ is satisfiable on side j only when $own_j(\delta_1) \cap own_j(\delta_2) = \emptyset$, since the two operand fragments are disjoint.

The permissive convention means a witnessing $\star$-split may scatter unowned arrays arbitrarily between the framed and active fragments; in particular we may *not* assume the framed fragment contains only δ's owned arrays, nor that an unowned "extra" could be moved into the active fragment (the active property δ_a need not tolerate it; for instance *emp* requires an empty fragment, so rebalancing an extra into an *emp* active part would falsify it). The heap-frame rule therefore does *not* canonicalize the split. Instead it confines the command footprint and the relational pre/postconditions to the arrays *owned by the active property*: on every witnessing split those arrays are pinned into the active fragment, while any unowned extra harmlessly remains in the frame, untouched and unread. We make this precise with a locality notion for assertions (below) and a footprint side condition on commands (stated with the rule).

Definition 6.5 (Frame locality)**.** *A relational assertion P is* local *with respect to a framed property δ and an active property δ_a, written $local_{\delta,\delta_a}(P)$, if for every interpretation I, stacks s_1, s_2, and split $h_j = h_j^a \uplus h_j^f$ ($j \in \{1,2\}$, $h_j^a \perp h_j^f$) with $((s_1,h_1^f),(s_2,h_2^f)) \vDash_I \delta$ and $((s_1,h_1^a),(s_2,h_2^a)) \vDash_I \delta_a$,*

$$((s_1,h_1^a),(s_2,h_2^a)) \vDash_I^r P \iff ((s_1,h_1),(s_2,h_2)) \vDash_I^r P.$$

The biconditional supplies both directions the soundness proof needs: full-to-active to feed the induction hypothesis on the active fragments, and active-to-full to lift the postcondition back to the whole heap. A purely syntactic

disjointness $arrFV(P) \cap arrFV(\delta) = \emptyset$ is *not* sufficient: a third array, named by neither δ nor δ_a, could be hidden as a permissive extra in the framed fragment, so that P reads it on the full heap but not on the active fragment. The sound condition pins P's arrays to the active owner.

Lemma 6.1 (Syntactic sufficient condition for frame locality)**.** *Let $arrFV_j(P)$ be the run-j projection of the arrays P actually reads from the heap: an array-read subterm $u_j[\,\cdot\,]$ and a length term $len(u_j)$ each contribute u_j, whereas an index-membership atom $J \in \beta$ contributes nothing, since β and J are interpretation-level (above) and read no heap. If $arrFV_j(P) \subseteq own_j(\delta_a)$ for $j \in \{1, 2\}$, then $local_{\delta,\delta_a}(P)$ holds for every framed property δ (the conclusion is vacuous when no split with $h^f \vDash_I \delta$ and $h^a \vDash_I \delta_a$ exists).*

Proof. Fix a qualifying split. The active atoms force $own_j(\delta_a) \subseteq dom(h^a_j)$, and $h^a_j \perp h^f_j$ places these arrays outside h^f_j; hence h_j and h^a_j store identical contents for every array in $own_j(\delta_a) \supseteq arrFV_j(P)$. Stacks are shared between the fragments, so scalar and logical reads coincide. By structural induction on P (atomic scalar relations, atomic array relations that read only $arrFV_j(P)$, β-membership atoms that are pure and depend only on I, and the Boolean connectives and quantifiers compositionally), P has the same truth value on $((s_1, h^a_1), (s_2, h^a_2))$ and $((s_1, h_1), (s_2, h_2))$. □

The heap-frame proof also factors a command's execution through the active fragment. We isolate this as a semantic frame property of the operational semantics, so the soundness case can cite it rather than re-argue it inline.

Lemma 6.2 (Command-frame factorization)**.** *Let C be a command and A a set of array names with $arrfoot(C) \subseteq A$. For any state (s, h) and any split $h = h^a \uplus h^f$ ($h^a \perp h^f$) with $A \subseteq dom(h^a)$, every terminating execution $((s, h), (s', h'), m) \in [\![C]\!]$ factors through the active heap: there is a heap $h^{a\prime}$ with $((s, h^a), (s', h^{a\prime}), m) \in [\![C]\!]$, $h' = h^{a\prime} \uplus h^f$, and h^f unchanged.*

Proof. By induction on the structure of C, using throughout that every array a step reads or writes lies in $arrfoot(C) \subseteq A \subseteq dom(h^a)$, disjoint from h^f.

- *Scalar assignment $x := e$.* The arrays read in e form $arrFV(e) = arrfoot(C) \subseteq dom(h^a)$, so e denotes the same value on h and on h^a; the step changes only the stack. Take $h^{a\prime} = h^a$.
- *Read $x := u_a[e]$ and update $u_a[e_1] := e_2$.* The accessed array u_a and the index/value arrays ($arrFV(e)$, resp. $arrFV(e_1) \cup arrFV(e_2)$) all lie in $arrfoot(C) \subseteq dom(h^a)$. The read returns the same value on h and h^a; the update rewrites a cell of $u_a \in dom(h^a)$, leaving h^f untouched. Set $h^{a\prime}$ to the updated active heap.
- *Conditional / while.* The guard's array reads $arrFV(b) \subseteq arrfoot(C) \subseteq dom(h^a)$ evaluate identically on h and h^a, so the same branch is taken (resp. the same number of iterations runs). Apply the induction hypothesis

to the chosen branch. For **while**, each iteration preserves the invariant $A \subseteq \mathit{dom}(h^a)$ (by the domain argument of the sequence case below: no deallocation, allocation impossible), so the hypothesis applies at every iteration and the factorizations compose.

- *Sequence* $C_1; C_2$. As $\mathit{arrfoot}(C_1), \mathit{arrfoot}(C_2) \subseteq \mathit{arrfoot}(C) \subseteq A$, apply the induction hypothesis to C_1 to factor through h^a to an intermediate active heap whose domain still contains A. Domain preservation holds because the language has *no deallocation command*: reads and scalar assignments leave the heap unchanged, an update rewrites a cell of an existing array without changing the domain, and a taken allocation step is impossible (allocation case). Thus no array leaves or is added to the active domain; apply the induction hypothesis to C_2.

- *Allocation* $u_a \leftarrow \mathit{array}(e_1)(e_2)$. A taken allocation step is *impossible* under the hypothesis: its target satisfies $u_a \in \mathit{arrfoot}(C) \subseteq A \subseteq \mathit{dom}(h^a) \subseteq \mathit{dom}(h)$, yet the operational semantics of allocation requires $u_a \notin \mathit{dom}(h)$. Hence no terminating execution covered by the hypothesis takes an allocation step, and the heap is never extended (an allocation may still sit in an untaken conditional branch; the syntactic footprint condition merely keeps its target inside A).

In every case the cost component m is independent of the framed fragment, and the same stack updates are performed, producing the same final stack s'; so the same execution runs from (s, h^a) to $(s', h^{a\prime})$ with $h' = h^{a\prime} \uplus h^f$ and h^f unchanged. □

6.4 Relational Validity (Level 1)

Definition 6.6 (Relational validity, Level 1)**.** *A relational partial correctness statement* $\vdash \{\delta\} \mid \{P\}\ C_1 \sim C_2\ \{Q\} \mid \{\delta'\}$ *is* valid*, written*

$$\vDash \{\delta\} \mid \{P\}\ C_1 \sim C_2\ \{Q\} \mid \{\delta'\}$$

if for all $\sigma_1, \sigma_2, \sigma_1', \sigma_2', I$ *with* $\mathit{dom}(I) \supseteq \mathit{LV}(P, Q, \delta, \delta')$*:*

$$\textit{if } (\sigma_1, \sigma_2) \vDash_I P \mid \delta \textit{ and } (\sigma_1, \sigma_1', \_) \in [\![C_1]\!] \textit{ and } (\sigma_2, \sigma_2', \_) \in [\![C_2]\!],$$

$$\textit{then } (\sigma_1', \sigma_2') \vDash_I Q \mid \delta'.$$

Note that the two programs execute *independently* under $[\![C_1]\!]$ and $[\![C_2]\!]$; there is no paired operational semantics. The relational structure arises entirely from the pre/postconditions and the binary property δ. The third component of $[\![C]\!]$ (the cost) is discarded at Level 1: it is observational metadata that does not influence the produced post-state. The well-formedness condition $\mathit{dom}(I) \supseteq \mathit{LV}(P, Q, \delta, \delta')$ ranges over specification-level logical variables only; commands are closed object-language syntax and do not contain logical symbols.

6.5 Soundness Theorems

Theorem 6.3 (Soundness, Level 1)**.** *If* $\vdash \{\delta\} \mid \{P\}\ C_1 \sim C_2\ \{Q\} \mid \{\delta'\}$ *is derivable in the Level 1 relational logic (Section 5), then* $\vDash \{\delta\} \mid \{P\}\ C_1 \sim C_2\ \{Q\} \mid \{\delta'\}$.

Proof. We prove Level 1 soundness by induction on derivations. The induction invariant is the semantic postcondition and binary post-property required by Definition 6.6. The same case analysis supplies the postcondition/effect component reused by the Level 2 theorem in Section 8; grade obligations are discharged there.

Case ***false****.*
Rule recalled.

$$\frac{}{\vdash \{\delta\} \mid \{\bot\}\ C_1 \sim C_2\ \{P\} \mid \{\delta\}}\ \textbf{false}$$

Fix any interpretation, initial states, and terminating executions satisfying the semantic antecedent of Definition 6.6. The relational assertion component of the precondition is $\bot$, so no such initial pair exists. Hence the semantic implication is vacuous. The same vacuity is reused for the postcondition component of the graded false rule.

Case ***conseq****.*
Rule recalled.

$$\frac{\begin{array}{c}\vdash \{\delta_1'\} \mid \{P'\}\ C_1 \sim C_2\ \{Q'\} \mid \{\delta_2'\} \\ \vDash P \Rightarrow P' \quad \vDash Q' \Rightarrow Q \\ P \vdash \delta_1' \leq \delta_1 \quad Q \vdash \delta_2 \leq \delta_2'\end{array}}{\vdash \{\delta_1\} \mid \{P\}\ C_1 \sim C_2\ \{Q\} \mid \{\delta_2\}}\ \textbf{conseq}$$

Assume the conclusion precondition holds: $(\sigma_1, \sigma_2) \vDash_I P \mid \delta_1$, and assume terminating executions of the two commands. The assertion premise $\vDash P \Rightarrow P'$ gives $(\sigma_1, \sigma_2) \vDash_I^r P'$. The pre-effect order premise is exactly the semantic weakening needed to turn satisfaction of the conclusion pre-effect δ_1 into satisfaction of the premise pre-effect δ_1'. Therefore the induction hypothesis for the premise judgment yields $(\sigma_1', \sigma_2') \vDash_I Q' \mid \delta_2'$. The assertion implication $\vDash Q' \Rightarrow Q$ gives the conclusion assertion Q, and the post-effect order premise weakens δ_2' to the conclusion post-effect δ_2. The graded **conseq** rule reuses the same postcondition and effect argument; its grade weakening is part of the Level 2 cost-bound argument in Section 8.9.

Case ***sequence****.*
Rule recalled.

$$\frac{\vdash \{\delta\} \mid \{P\}\ C_1 \sim C_2\ \{R\} \mid \{\delta_1\} \quad \vdash \{\delta_1\} \mid \{R\}\ C_1' \sim C_2'\ \{Q\} \mid \{\delta'\}}{\vdash \{\delta\} \mid \{P\}\ C_1; C_1' \sim C_2; C_2'\ \{Q\} \mid \{\delta'\}}\ \textbf{sequence}$$

Suppose the conclusion precondition holds and $(\sigma_i, \sigma_i'', m_i) \in [\![C_i; C_i']\!]$ for $i \in \{1, 2\}$. By the operational semantics of sequence, for each side there are intermediate states and costs $(\sigma_i, \widehat{\sigma}_i, m_i^a) \in [\![C_i]\!]$ and $(\widehat{\sigma}_i, \sigma_i'', m_i^b) \in [\![C_i']\!]$ with $m_i = m_i^a + m_i^b$. Applying the first induction hypothesis to the first pair of

executions and the original precondition gives $(\widehat{\sigma}_1, \widehat{\sigma}_2) \vDash_I R \mid \delta_1$. This is exactly the semantic antecedent required by the second premise. Applying the second induction hypothesis to the remaining executions gives $(\sigma_1'', \sigma_2'') \vDash_I Q \mid \delta'$, which is the sequence conclusion. The graded sequence rule reuses this state/effect decomposition; the additive grade calculation is handled in Section 8.9.

Case ***u2binary-derived****.*
Rule recalled.

$$\frac{\begin{array}{c}\vdash^{m_1} \{p_1 \mid \{u_1 \mapsto^{\phi_1} \beta\}\}\ C_1\ \{q_1 \mid \{u_1 \mapsto^{\phi_1} \beta'\}\} \\ \vdash^{m_2} \{p_2 \mid \{u_2 \mapsto^{\phi_2} \beta\}\}\ C_2\ \{q_2 \mid \{u_2 \mapsto^{\phi_2} \beta'\}\} \\ \forall x, y.\ \Phi(x,y) \Rightarrow \phi_1(x) \wedge \phi_2(y) \\ \forall x, y.\ \phi_1(x) \wedge \phi_2(y) \Rightarrow \Phi(x,y)\end{array}}{\vdash \{(u_1,u_2) \mapsto^{\Phi} \beta\} \mid \{p_1\langle 1\rangle \wedge p_2\langle 2\rangle\}\ C_1 \sim C_2\ \{q_1\langle 1\rangle \wedge q_2\langle 2\rangle\} \mid \{(u_1,u_2) \mapsto^{\Phi} \beta'\}}\ \textbf{u2binary-derived}$$

This rule is derived rather than primitive: it packages two unary derivations with a compatibility proof for the binary relation. Assume the conclusion precondition holds, writing $\sigma_i = (s_i, h_i)$. The relational assertion component gives $(s_1, h_1) \vDash_I p_1$ and $(s_2, h_2) \vDash_I p_2$ after erasing the run tags. The binary pre-property gives, for each $j \in [\![\beta]\!]_b(I)$,

$$\Phi(h_1(u_1)[j], h_2(u_2)[j]).$$

By the first compatibility premise, the same cell pair satisfies $\phi_1(h_1(u_1)[j])$ and $\phi_2(h_2(u_2)[j])$. Thus the left and right unary pre-effects required by the two unary judgments hold. Unary soundness applied to the two terminating executions yields unary postconditions q_1, q_2 and unary post-effects over the rule's post-domain β'.

Now fix any $j \in [\![\beta']\!]_b(I)$. The two unary post-effects give the post-state facts $\phi_1(h_1'(u_1)[j])$ and $\phi_2(h_2'(u_2)[j])$, together with the array-existence and in-bounds obligations needed to interpret the atom. By the converse compatibility premise, $\phi_1(x) \wedge \phi_2(y) \Rightarrow \Phi(x,y)$, we obtain

$$\Phi(h_1'(u_1)[j], h_2'(u_2)[j]).$$

Therefore Definition 6.2 gives $(\sigma_1', \sigma_2') \vDash_I (u_1, u_2) \mapsto^{\Phi} \beta'$. The unary postconditions lift back to the relational assertion $q_1\langle 1\rangle \wedge q_2\langle 2\rangle$.

Case ***frame****.*
Rule recalled.

$$\frac{\begin{array}{c}\vdash \{\delta_1\} \mid \{P\}\ C_1 \sim C_2\ \{Q\} \mid \{\delta_2\} \\ FV(R) \cap (mod(C_1)\langle 1\rangle \cup mod(C_2)\langle 2\rangle) = \emptyset \\ arrFV(R) \cap (arrvars(C_1)\langle 1\rangle \cup arrvars(C_2)\langle 2\rangle) = \emptyset\end{array}}{\vdash \{\delta_1\} \mid \{P \wedge R\}\ C_1 \sim C_2\ \{Q \wedge R\} \mid \{\delta_2\}}\ \textbf{frame}$$

Assume the conclusion precondition $(\sigma_1, \sigma_2) \vDash_I P \wedge R \mid \delta_1$ and terminating executions of the two commands. The active part gives $(\sigma_1, \sigma_2) \vDash_I P \mid \delta_1$, so the induction hypothesis yields $(\sigma_1', \sigma_2') \vDash_I Q \mid \delta_2$. It remains to preserve R.

The preservation fact used here is the standard free-variable lemma for relational assertions: if an execution pair leaves every scalar variable in $FV(R)$ unchanged and does not change the interpretation of any array in $arrFV(R)$, then satisfaction of R is invariant from the pre-pair to the post-pair. The proof is by structural induction on R. Atomic scalar relations use equality of the relevant scalar interpretations; atomic array relations use equality of the relevant array identities and contents; β-membership atoms are pure and depend only on I; and the Boolean connectives and quantifiers follow compositionally. The rule's scalar and array side conditions establish exactly these hypotheses. Commands may read arrays mentioned in R, but they cannot update, allocate into, or rebind them. Hence $(\sigma_1', \sigma_2') \vDash_I^r R$. Combining this with the induction-hypothesis postcondition gives $Q \wedge R \mid \delta_2$. The binary property is unchanged.

Case ***heap-frame****.*
Rule recalled.

$$\frac{\begin{array}{c}\vdash \{\delta_1\} \mid \{P\}\ C_1 \sim C_2\ \{Q\} \mid \{\delta_2\} \\ FV(\delta) \cap (mod(C_1)\langle 1\rangle \cup mod(C_2)\langle 2\rangle) = \emptyset \\ arrFV(\delta) \cap (arrfoot(C_1)\langle 1\rangle \cup arrfoot(C_2)\langle 2\rangle) = \emptyset \\ arrfoot(C_i)\langle i\rangle \subseteq own_i(\delta_1) \quad (i \in \{1,2\}) \\ local_{\delta,\delta_1}(P) \text{ and } local_{\delta,\delta_2}(Q)\end{array}}{\vdash \{\delta \star \delta_1\} \mid \{P\}\ C_1 \sim C_2\ \{Q\} \mid \{\delta \star \delta_2\}}\ \textbf{heap-frame}$$

Assume $(\sigma_1, \sigma_2) \vDash_I P \mid \delta \star \delta_1$, with $\sigma_i = (s_i, h_i)$, and terminating executions $((s_i, h_i), (s_i', h_i'), m_i) \in [\![C_i]\!]$.

Witnessing split. By Definition 6.2, the precondition effect $\delta \star \delta_1$ provides, for each side i, a split $h_i = h_i^f \uplus h_i^a$ with $((s_1, h_1^f), (s_2, h_2^f)) \vDash_I \delta$ and $((s_1, h_1^a), (s_2, h_2^a)) \vDash_I \delta_1$. We do *not* canonicalize this split; the footprint and locality side conditions make every witnessing split usable. The active atoms pin $own_i(\delta_1) \subseteq dom(h_i^a)$, and disjointness $h_i^a \perp h_i^f$ keeps these arrays out of h_i^f. The footprint side condition $arrfoot(C_i)\langle i\rangle \subseteq own_i(\delta_1)$ then confines the command's *entire* array footprint, every array it may inspect, update, rebind, or allocate, to $own_i(\delta_1) \subseteq dom(h_i^a)$, hence disjoint from h_i^f. In particular, no terminating execution covered by the rule takes a fresh-allocation step. An allocation target lies in $arrfoot(C_i)$, so the footprint condition forces it into $own_i(\delta_1) \subseteq dom(h_i^a) \subseteq dom(h_i)$; yet the operational semantics of allocation requires that target to be *absent* from the pre-state heap. These are contradictory, so an allocation step cannot fire on a state meeting the rule's premises (the deliberate no-framed-allocation restriction noted with the footprint definition). An allocation may still sit syntactically in an untaken conditional branch; because *arrfoot* collects allocation targets from *both* branches, the footprint condition keeps even such a target inside $own_i(\delta_1)$, but no covered execution actually allocates. Thus the command reads and writes only the active fragment, while any unowned "extra" that the permissive split left in h_i^f is never touched. This is the step the permissive-atom convention would otherwise leave unjustified: a δ_1-disjoint array cannot be silently read from a hidden frame entry, because every array the command touches is forced into the active fragment.

Precondition localizes. The premise $\mathit{local}_{\delta,\delta_1}(P)$ (Definition 6.5), instantiated at this split, turns $(\sigma_1, \sigma_2) \vDash_I^r P$ on the full heaps into $((s_1, h_1^a), (s_2, h_2^a)) \vDash_I^r P$ on the active fragments, giving $((s_1, h_1^a), (s_2, h_2^a)) \vDash_I P \mid \delta_1$, the antecedent the induction hypothesis requires.

Command locality. Apply the command-frame factorization lemma (Lemma 6.2) to the side-i command instance C_i, a unary statement over the single heap h_i, with the run tag on side i erased, taking $A = \mathit{own}_i(\delta_1)$: the footprint side condition supplies the hypothesis $\mathit{arrfoot}(C_i)\langle i\rangle \subseteq \mathit{own}_i(\delta_1) = A$, and the active atoms supply $A = \mathit{own}_i(\delta_1) \subseteq \mathit{dom}(h_i^a)$. The lemma yields an active result $h_i^{a\prime}$ with $((s_i, h_i^a), (s_i', h_i^{a\prime}), m_i) \in [\![C_i]\!]$, $h_i' = h_i^{a\prime} \uplus h_i^f$, and h_i^f unchanged. By the lemma's allocation case no covered execution allocates, so h_i^f is never extended; the recombination is therefore just the unchanged framed fragment h_i^f joined with the active result $h_i^{a\prime}$, on disjoint domains. (The disjointness of scalar modifications from $FV(\delta)$ is used below to preserve the framed property, not for this heap factorization.)

Induction hypothesis and postcondition. Applying the induction hypothesis to the active executions transforms δ_1 into δ_2 and gives $((s_1', h_1^{a'}), (s_2', h_2^{a'})) \vDash_I Q \mid \delta_2$. The framed fragments are unchanged and $FV(\delta)$ is disjoint from the modified variables, so $((s_1', h_1^f), (s_2', h_2^f)) \vDash_I \delta$ still holds; thus $h_i' = h_i^f \uplus h_i^{a'}$ witnesses $(\sigma_1', \sigma_2') \vDash_I \delta \star \delta_2$. Finally $\mathit{local}_{\delta,\delta_2}(Q)$, instantiated at the post-split (framed $h_i^f \vDash_I \delta$, active $h_i^{a'} \vDash_I \delta_2$), lifts Q from the active fragments back to the full post-heaps, giving $(\sigma_1', \sigma_2') \vDash_I^r Q$. Therefore $(\sigma_1', \sigma_2') \vDash_I Q \mid \delta \star \delta_2$.

*Case **skip**.*
Rule recalled.

$$\frac{}{\vdash \{\delta\} \mid \{P\}\ \texttt{skip} \sim \texttt{skip}\ \{P\} \mid \{\delta\}}\ \textbf{skip}$$

Both executions are identity executions: $(\sigma_1, \sigma_1, 0) \in [\![\texttt{skip}]\!]$ and $(\sigma_2, \sigma_2, 0) \in [\![\texttt{skip}]\!]$. Thus the precondition $P \mid \delta$ is already the postcondition $P \mid \delta$.

*Case **assign**.*
Rule recalled.

$$\frac{}{\vdash \{\delta[e_1\langle 1\rangle/x_1\langle 1\rangle][e_2\langle 2\rangle/x_2\langle 2\rangle]\} \mid \{Q[e_1\langle 1\rangle/x_1\langle 1\rangle][e_2\langle 2\rangle/x_2\langle 2\rangle]\}\ x_1 := e_1 \sim x_2 := e_2\ \{Q\} \mid \{\delta\}}\ \textbf{assign}$$

The two operational steps update only x_1 on the left and x_2 on the right to the values of the corresponding pre-state expressions. Applying the scalar substitution lemma (Lemma 3.1) on each run, to $x_1\langle 1\rangle$ with $e_1\langle 1\rangle$ and to $x_2\langle 2\rangle$ with $e_2\langle 2\rangle$, evaluating Q after those two assignments is equivalent to evaluating the substituted precondition before them. The same substitution on δ is allowed only when it keeps the binary effect well formed, so the binary post-property is exactly δ.

*Case **read1**.*

Rule recalled.

$$\frac{\begin{array}{c} j \text{ fresh} \quad x_1, x_2 \notin FV(\beta) \\ P = Q[u_1[e_1]/x_1\langle 1\rangle][u_2[e_2]/x_2\langle 2\rangle] \\ P \implies e_1\langle 1\rangle = e_2\langle 2\rangle = j \wedge j \in \beta \end{array}}{\vdash \{(u_1, u_2) \mapsto^{\Phi} \beta\} \mid \{P\}\ x_1 := u_1[e_1] \sim x_2 := u_2[e_2]\ \{Q \wedge \Phi(x_1, x_2)\} \mid \{(u_1, u_2) \mapsto^{\Phi} \beta\}}\ \textbf{read1}$$

Setup. Assume $(\sigma_1, \sigma_2) \vDash_I P \mid (u_1, u_2) \mapsto^{\Phi} \beta$, with $\sigma_i = (s_i, h_i)$, and terminating read executions $((s_i, h_i), (s_i', h_i'), m_i) \in [\![x_i := u_i[e_i]]\!]$ for $i \in \{1, 2\}$. The goal is $(\sigma_1', \sigma_2') \vDash_I Q \wedge \Phi(x_1, x_2) \mid (u_1, u_2) \mapsto^{\Phi} \beta$.

Common index. Set $k = I(j)$. From $(\sigma_1, \sigma_2) \vDash_I^r P$ and the premise $P \implies e_1\langle 1\rangle = e_2\langle 2\rangle = j \wedge j \in \beta$, both index expressions evaluate to this shared value and the value is tracked:

$$[\![e_1]\!]_e((s_1, h_1), I) = [\![e_2]\!]_e((s_2, h_2), I) = k \quad \text{and} \quad k \in [\![\beta]\!]_b(I).$$

Pre-property at k. From $(\sigma_1, \sigma_2) \vDash_I (u_1, u_2) \mapsto^{\Phi} \beta$ and Definition 6.2, the atom supplies the array-existence facts $u_1 \in dom(h_1)$, $u_2 \in dom(h_2)$ and the in-bounds clause $[\![\beta]\!]_b(I) \subseteq [0, \min(len(h_1(u_1)), len(h_2(u_2))))$; since $k \in [\![\beta]\!]_b(I)$, both cells at k are defined and

$$\Phi(h_1(u_1)[k], h_2(u_2)[k]).$$

Read steps. By the operational semantics of the array-read assignment $x_i := u_i[e_i]$, side i leaves the heap unchanged and updates only the scalar x_i:

$$\sigma_1' = (s_1[x_1 \mapsto h_1(u_1)[k]],\, h_1), \qquad \sigma_2' = (s_2[x_2 \mapsto h_2(u_2)[k]],\, h_2),$$

using $[\![e_i]\!]_e((s_i, h_i), I) = k$ from the common-index step.

Postcondition $Q \wedge \Phi(x_1, x_2)$. The reads assign x_1, x_2 exactly the values denoted by the substituted read expressions $u_i[e_i]$, so the scalar substitution lemma (Lemma 3.1) applied on each run, to $x_1\langle 1\rangle$ with the array-read expression $u_1[e_1]\langle 1\rangle$ and to $x_2\langle 2\rangle$ with $u_2[e_2]\langle 2\rangle$, together with the premise $P = Q[u_1[e_1]/x_1\langle 1\rangle][u_2[e_2]/x_2\langle 2\rangle]$ and $(\sigma_1, \sigma_2) \vDash_I^r P$, yields $(\sigma_1', \sigma_2') \vDash_I^r Q$. The pre-property fact $\Phi(h_1(u_1)[k], h_2(u_2)[k])$, together with $s_1'(x_1) = h_1(u_1)[k]$ and $s_2'(x_2) = h_2(u_2)[k]$, gives $(\sigma_1', \sigma_2') \vDash_I^r \Phi(x_1, x_2)$. (The j-freshness premise ensures the naming index collides with no free variable of Q or β.)

Binary post-property preserved. The reads modify only the scalars x_1, x_2 and leave both heaps unchanged ($h_i' = h_i$); the atom heads u_1, u_2 and their contents at every index are untouched. The value-relation clause $\forall x \in [\![\beta]\!]_b(I).\, \Phi(h_1(u_1)[x], h_2(u_2)[x])$ is therefore preserved: its cell values are unchanged, and by well-formedness (Definition 6.3) Φ mentions no program variable, so the updates to x_1, x_2 cannot alter its truth. The tracked domain is unchanged too: by the side condition $x_1, x_2 \notin FV(\beta)$ the scalar updates leave $[\![\beta]\!]_b(I)$ fixed. Hence $(\sigma_1', \sigma_2') \vDash_I (u_1, u_2) \mapsto^{\Phi} \beta$.

Conclusion. Combining the last two steps, $(\sigma_1', \sigma_2') \vDash_I Q \wedge \Phi(x_1, x_2) \mid (u_1, u_2) \mapsto^{\Phi} \beta$, the **read1** conclusion. At Level 1 the read cost m_i is discarded by Definition 6.6, so it carries no obligation here.

*Case **read2**.*
Rule recalled.

$$\frac{\begin{array}{c} j \text{ fresh} \quad x_1, x_2 \notin FV(\beta) \\ P = Q[u_1[e_1]/x_1\langle 1\rangle][u_2[e_2]/x_2\langle 2\rangle] \\ P \implies e_1\langle 1\rangle = e_2\langle 2\rangle = j \end{array}}{\vdash \{(u_1, u_2) \mapsto^{\Phi} \beta\} \mid \{P\}\ x_1 := u_1[e_1] \sim x_2 := u_2[e_2]\ \{Q\} \mid \{(u_1, u_2) \mapsto^{\Phi} \beta\}}\ \textbf{read2}$$

This proof is the same read-substitution argument as **read1**, except the rule does not assume $j \in \beta$. Therefore Definition 6.2 does not provide a pointwise relation between the two values read, and the postcondition contains only Q. Since reads leave heaps unchanged and the fresh result variables are not in β, the binary property is preserved.

*Case **update1**.*
Rule recalled.

$$\frac{\begin{array}{c} j \text{ fresh} \\ P = Q[u_1[e_1\langle 1\rangle \mapsto e_1'\langle 1\rangle]/u_1][u_2[e_2\langle 2\rangle \mapsto e_2'\langle 2\rangle]/u_2] \\ P \implies e_1\langle 1\rangle = e_2\langle 2\rangle = j \\ P \implies 0 \le j < \min(len(u_1)\langle 1\rangle, len(u_2)\langle 2\rangle) \\ P \implies \Phi(e_1'\langle 1\rangle, e_2'\langle 2\rangle) \end{array}}{\vdash \{(u_1, u_2) \mapsto^{\Phi} \beta\} \mid \{P\}\ u_1[e_1] := e_1' \sim u_2[e_2] := e_2'\ \{Q\} \mid \{(u_1, u_2) \mapsto^{\Phi} (\beta \cup \{j\})\}}\ \textbf{update1}$$

The side conditions make the two updates occur at the same in-bounds index j. Heap-update semantics changes only cell j of the two named arrays. For all tracked indices already in β and different from j, the old binary property still supplies Φ and the cells are unchanged. For the new tracked index j, the premise $P \implies \Phi(e_1'\langle 1\rangle, e_2'\langle 2\rangle)$ gives the relation between the two values written. Thus the post-heap satisfies $(u_1, u_2) \mapsto^{\Phi} (\beta \cup \{j\})$, and the substitution premise gives Q.

*Case **update2**.*
Rule recalled.

$$\frac{\begin{array}{c} j \text{ fresh} \\ P = Q[u_1[e_1\langle 1\rangle \mapsto e_1'\langle 1\rangle]/u_1][u_2[e_2\langle 2\rangle \mapsto e_2'\langle 2\rangle]/u_2] \\ P \implies e_1\langle 1\rangle = e_2\langle 2\rangle = j \\ P \implies 0 \le j < \min(len(u_1)\langle 1\rangle, len(u_2)\langle 2\rangle) \end{array}}{\vdash \{(u_1, u_2) \mapsto^{\Phi} \beta\} \mid \{P\}\ u_1[e_1] := e_1' \sim u_2[e_2] := e_2'\ \{Q\} \mid \{(u_1, u_2) \mapsto^{\Phi} (\beta - \{j\})\}}\ \textbf{update2}$$

The two runs update the same in-bounds index j, but no premise relates the two values written. The rule therefore removes j from the tracked domain. For every remaining tracked index in $\beta - \{j\}$, heap-update semantics leaves both arrays' cells unchanged; the old binary property still proves Φ there. The substitution premise gives Q.

*Case **alloc1**.*

Rule recalled.

$$\frac{\begin{array}{c} P \implies Q \\ \mathit{FreshAlloc}(Q, \delta, u_1, u_2) \\ \ell \text{ fresh logical index} \quad \ell \notin FV(Q) \cup FV(\delta) \\ P \implies e_1\langle 1\rangle = \ell \wedge e_2\langle 2\rangle = \ell \wedge 0 \le \ell \\ P \implies \Phi(e_1'\langle 1\rangle, e_2'\langle 2\rangle) \end{array}}{\vdash \{\delta\} \mid \{P\}\ u_1 \leftarrow \mathit{array}(e_1)(e_1') \sim u_2 \leftarrow \mathit{array}(e_2)(e_2')\ \{Q\} \mid \{\delta \star ((u_1, u_2) \mapsto^{\Phi} [0, \ell - 1])\}}\ \textbf{alloc1}$$

The allocation semantics chooses fresh heap entries for u_1 and u_2, with lengths given by the pre-state lengths $e_1\langle 1\rangle$ and $e_2\langle 2\rangle$. The ghost snapshot premise makes those lengths equal to the same logical ℓ, and $0 \le \ell$ makes the interval well formed. Freshness ensures the old heap component satisfying δ is disjoint from the newly allocated arrays and that Q and δ do not depend on the fresh array names. Hence $P \implies Q$ gives the assertion postcondition and preserves the old effect δ. Every allocated cell contains the corresponding initial value, and $P \implies \Phi(e_1'\langle 1\rangle, e_2'\langle 2\rangle)$ gives Φ for every index in $[0, \ell - 1]$; when $\ell = 0$, the interval denotes $\emptyset$ and the value obligation is vacuous. Recombining the old and fresh heap fragments gives the post-effect.

Case ***alloc2****.*

Rule recalled.

$$\frac{\begin{array}{c} P \implies Q \\ \mathit{FreshAlloc}(Q, \delta, u_1, u_2) \\ P \implies 0 \le e_1\langle 1\rangle \wedge 0 \le e_2\langle 2\rangle \end{array}}{\vdash \{\delta\} \mid \{P\}\ u_1 \leftarrow \mathit{array}(e_1)(e_1') \sim u_2 \leftarrow \mathit{array}(e_2)(e_2')\ \{Q\} \mid \{\delta \star ((u_1, u_2) \mapsto^{\Phi} \emptyset)\}}\ \textbf{alloc2}$$

The proof is the same freshness and heap-extension argument as **alloc1**, except the fresh binary atom tracks the empty domain. Thus no relation between initial values is required, and unequal allocation lengths are allowed as long as both lengths are nonnegative. Definition 6.2 still requires the two fresh arrays to exist in the post heaps, which the allocation steps provide.

Case ***conditional****.*

Rule recalled.

$$\frac{\begin{array}{c} \vDash P \implies b_1\langle 1\rangle = b_2\langle 2\rangle \\ \vdash \{\delta\} \mid \{P \wedge b_1\langle 1\rangle\}\ C_1 \sim C_2\ \{Q\} \mid \{\delta'\} \quad \vdash \{\delta\} \mid \{P \wedge \neg b_1\langle 1\rangle\}\ C_1' \sim C_2'\ \{Q\} \mid \{\delta'\} \end{array}}{\vdash \{\delta\} \mid \{P\}\ \texttt{if}\ b_1\ \texttt{then}\ C_1\ \texttt{else}\ C_1' \sim\ \texttt{if}\ b_2\ \texttt{then}\ C_2\ \texttt{else}\ C_2'\ \{Q\} \mid \{\delta'\}}\ \textbf{conditional}$$

The conclusion precondition entails $b_1\langle 1\rangle = b_2\langle 2\rangle$, so both guards evaluate to the same Boolean in the two initial states. If both guards are true, the executions are executions of the then-commands and the then-branch induction hypothesis applies. If both guards are false, the else-branch induction hypothesis applies. The branch postconditions are the same $Q \mid \delta'$, so the conditional conclusion follows.

Case ***assignL-skipR****.*

Rule recalled.

$$\frac{}{\vdash \{\delta[e_1\langle 1\rangle/x_1\langle 1\rangle]\} \mid \{Q[e_1\langle 1\rangle/x_1\langle 1\rangle]\}\ x_1 := e_1 \sim \texttt{skip}\ \{Q\} \mid \{\delta\}}\ \textbf{assignL-skipR}$$

The left assignment changes only x_1; the right skip leaves the right state unchanged. The usual substitution lemma proves Q in the final pair from the substituted precondition. The binary property is transformed by the same substitution and then interpreted in the post-state as δ.

*Case **assignL** (left assignment with continuation).*
Rule recalled.

$$\frac{\vdash \{\delta\} \mid \{P\}\ \texttt{skip} \sim C_2\ \{Q\} \mid \{\delta'\}}{\vdash \{\delta[e_1\langle 1\rangle/x_1\langle 1\rangle]\} \mid \{P[e_1\langle 1\rangle/x_1\langle 1\rangle]\}\ x_1 := e_1 \sim C_2\ \{Q\} \mid \{\delta'\}}\ \textbf{assignL}$$

Decompose the left execution as the active assignment followed by the empty left continuation, and view the right execution as the continuation C_2. The assignment-substitution argument establishes the premise precondition $P \mid \delta$ for the intermediate pair. The induction hypothesis for the premise then proves $Q \mid \delta'$.

*Case **readL-skipR**.*
Rule recalled.

$$\frac{x_1\langle 1\rangle \notin FV(\delta)}{\vdash \{\delta\} \mid \{Q[u_1[e_1\langle 1\rangle]/x_1\langle 1\rangle]\}\ x_1 := u_1[e_1] \sim \texttt{skip}\ \{Q\} \mid \{\delta\}}\ \textbf{readL-skipR}$$

The left read assigns the selected left heap value to x_1, and the right state is unchanged. The substitution premise proves Q after the read. Because reads do not mutate heaps and the freshness side condition keeps x_1 out of the binary property, δ is preserved exactly.

*Case **readL** (left read with continuation).*
Rule recalled.

$$\frac{\begin{array}{c} x_1\langle 1\rangle \notin FV(\delta) \\ \vdash \{\delta\} \mid \{P\}\ \texttt{skip} \sim C_2\ \{Q\} \mid \{\delta'\} \end{array}}{\vdash \{\delta\} \mid \{P[u_1[e_1\langle 1\rangle]/x_1\langle 1\rangle]\}\ x_1 := u_1[e_1] \sim C_2\ \{Q\} \mid \{\delta'\}}\ \textbf{readL}$$

After the active read, the substitution premise establishes the premise assertion P for the intermediate pair. Since the read does not change the heap relation and x_1 is not free in δ, the premise effect is still δ. The induction hypothesis for the continuation premise gives $Q \mid \delta'$.

*Case **updateL-skipR**.*
Rule recalled.

$$\frac{\begin{array}{c} j \text{ fresh} \\ Q[u_1[e_1\langle 1\rangle \mapsto e_1'\langle 1\rangle]/u_1] \implies e_1\langle 1\rangle = j \\ Q[u_1[e_1\langle 1\rangle \mapsto e_1'\langle 1\rangle]/u_1] \implies 0 \le j < len(u_1)\langle 1\rangle \end{array}}{\vdash \{\delta\} \mid \{Q[u_1[e_1\langle 1\rangle \mapsto e_1'\langle 1\rangle]/u_1]\}\ u_1[e_1] := e_1' \sim \texttt{skip}\ \{Q\} \mid \{\delta \ominus_{u_1\langle 1\rangle} \{j\}\}}\ \textbf{updateL-skipR}$$

The active update changes exactly the left cell at the pre-state index j; the side conditions make that index well defined and in bounds. The post-effect removes j from every atom whose run-1 array is u_1. Therefore every index that remains tracked either belongs to another atom or is an index of u_1 different from j; in both cases the corresponding cells are unchanged by the left update and the right skip. The substituted precondition gives Q after the heap update. The result is well formed because $\ominus$ preserves the structure of δ while shrinking only the affected domains.

*Case **updateL** (left update with continuation).*
Rule recalled.

$$\frac{\begin{array}{c} j \text{ fresh} \\ P[u_1[e_1\langle 1\rangle \mapsto e_1'\langle 1\rangle]/u_1] \implies e_1\langle 1\rangle = j \\ P[u_1[e_1\langle 1\rangle \mapsto e_1'\langle 1\rangle]/u_1] \implies 0 \le j < len(u_1)\langle 1\rangle \\ \vdash \{\delta \ominus_{u_1\langle 1\rangle} \{j\}\} \mid \{P\}\ \texttt{skip} \sim C_2\ \{Q\} \mid \{\delta'\} \end{array}}{\vdash \{\delta\} \mid \{P[u_1[e_1\langle 1\rangle \mapsto e_1'\langle 1\rangle]/u_1]\}\ u_1[e_1] := e_1' \sim C_2\ \{Q\} \mid \{\delta'\}}\ \textbf{updateL}$$

Let the initial left state be $\sigma_1 = (s_1, h_1)$ and let the active update produce $\widehat{\sigma}_1 = (\widehat{s}_1, \widehat{h}_1)$ by changing exactly the cell of u_1 at the pre-state index j; the side conditions identify the evaluated update index with j and make it in bounds. The substitution premise turns the original assertion into P at the intermediate pair $(\widehat{\sigma}_1, \sigma_2)$. For the binary property, the operation $\delta \ominus_{u_1\langle 1\rangle} \{j\}$ removes the affected run-1 obligations involving array u_1 at index j, according to the definition of $\ominus$. Every remaining tracked obligation is either about a different run-1 array, or about a different index of u_1; those cells are unchanged by the update, and the right state has not moved yet. Hence the intermediate pair satisfies $P \mid \delta \ominus_{u_1\langle 1\rangle} \{j\}$.

The premise judgment now runs $\texttt{skip} \sim C_2$ from this intermediate pair. Applying the induction hypothesis to that premise and to the remaining right execution yields the final postcondition $Q \mid \delta'$.

*Case **conditionalL**.*
Rule recalled.

$$\frac{\vdash \{\delta\} \mid \{P \wedge b_1\langle 1\rangle\}\ C_1 \sim C_2\ \{Q\} \mid \{\delta'\} \quad \vdash \{\delta\} \mid \{P \wedge \neg b_1\langle 1\rangle\}\ C_1' \sim C_2\ \{Q\} \mid \{\delta'\}}{\vdash \{\delta\} \mid \{P\}\ \ \texttt{if}\ b_1\ \texttt{then}\ C_1\ \texttt{else}\ C_1' \sim C_2\ \{Q\} \mid \{\delta'\}}\ \textbf{conditionalL}$$

Only the left run branches. If the left guard is true, the left execution is an execution of C_1 and the first premise induction hypothesis applies with precondition $P \wedge b_1\langle 1\rangle$. If the guard is false, the second premise applies. The right execution is the same C_2 in both cases, and both premises conclude $Q \mid \delta'$.

*Case **whileL-skipR**.*
Rule recalled.

$$\frac{\vdash \{\delta\} \mid \{P \wedge b_1\langle 1\rangle\}\ C_1 \sim \texttt{skip}\ \{P\} \mid \{\delta\}}{\vdash \{\delta\} \mid \{P\}\ \ \texttt{while}\ b_1\ \texttt{do}\ C_1 \sim \texttt{skip}\ \{P \wedge \neg b_1\langle 1\rangle\} \mid \{\delta\}}\ \textbf{whileL-skipR}$$

The proof is a one-sided variant of the fixed **awhile** argument. Induct on the number of left loop iterations in the terminating execution. If the left guard is

false, both the left while and the right skip are identity steps, and $P \wedge \neg b_1\langle 1\rangle \mid \delta$ follows directly. If the left guard is true, the left while decomposes into one body step followed by the remaining left while; the right side is still skip. The body-premise induction hypothesis gives $P \mid \delta$ after one body iteration. The outer induction applies to the remaining left iterations and the right skip, yielding the final postcondition. The graded one-sided loop rule reuses this postcondition argument; its summed cost-difference bound is handled in Section 8.9.

*Case **right-sided one-sided variants**.* Section 5 presents the left-sided rules and states that **assignR-skipL**, **conditionalR**, **whileR-skipL**, **readR**, and **updateR** are obtained symmetrically. Their soundness proofs are the same arguments above with run tags 1 and 2 exchanged.

One-sided allocation status. The current Level 1 rule section has paired allocation rules **alloc1**/**alloc2**, but no primitive **allocL** or **allocR** rule. Therefore there is no one-sided allocation case in the present induction. If a future version adds such a rule, its proof must state fresh heap-domain extension, preservation of all unmentioned binary atoms, and the absence or explicit framing of any binary relation mentioning the newly allocated array.

*Case **awhile** (fixed δ).* We prove: if the **awhile** judgment is derivable, then it is valid (Definition 6.6).
Rule recalled.

awhile premises:

(1) $n \geq 0$,
(2) $k \notin FV(P) \cup FV(b_1) \cup FV(b_2) \cup FV(e_v) \cup FV(\delta)$,
(3) $P \implies 0 \leq e_v\langle 1\rangle$,
(4) $P \implies \big(e_v\langle 1\rangle \leq 0 \iff \neg b_1\langle 1\rangle\big)$,
(5) $\forall k.\ \vdash \{\delta\} \mid \{P \wedge b_1\langle 1\rangle \wedge b_2\langle 2\rangle \wedge e_v\langle 1\rangle = k \wedge 0 < e_v\langle 1\rangle \leq n\}$ $C_1 \sim C_2$ $\{P \wedge b_1\langle 1\rangle = b_2\langle 2\rangle \wedge e_v\langle 1\rangle < k\} \mid \{\delta\}$. **awhile**

awhile conclusion:

$\vdash \{\delta\} \mid \{P \wedge b_1\langle 1\rangle = b_2\langle 2\rangle \wedge e_v\langle 1\rangle \leq n\}$ `while` b_1 `do` $C_1 \sim$ `while` b_2 `do` C_2 $\{P \wedge \neg b_1\langle 1\rangle \wedge \neg b_2\langle 2\rangle\} \mid \{\delta\}$.

Assume $(\sigma_1, \sigma_2) \vDash_I P \wedge b_1\langle 1\rangle = b_2\langle 2\rangle \wedge e_v\langle 1\rangle \leq n \mid \delta$, and $(\sigma_1, \sigma_1', \_) \in [\![$ `while` b_1 `do` $C_1]\!]$ and $(\sigma_2, \sigma_2', \_) \in [\![$ `while` b_2 `do` $C_2]\!]$. We show $(\sigma_1', \sigma_2') \vDash_I P \wedge \neg b_1\langle 1\rangle \wedge \neg b_2\langle 2\rangle \mid \delta$.

Let $k_0 = [\![e_v\langle 1\rangle]\!]_i(\sigma_1, I)$. We prove a state-parametric claim $S(k)$ by well-founded induction on $k \in \mathbb{N}$, then instantiate it with $k = k_0$. Arithmetic expressions are integer-valued, and the rule premise $P \implies 0 \leq e_v\langle 1\rangle$ makes the current variant value natural for every state satisfying the loop invariant. Thus $(\mathbb{N}, <)$ provides the well-founded order, with base case $k = 0$ (the biconditional $e_v\langle 1\rangle \leq 0 \iff \neg b_1\langle 1\rangle$ then forces the loop to exit). Variants that may become negative are outside this rule; they should be wrapped in a nonnegative measure before applying **awhile**.

The side condition $k \notin FV(\delta)$ is what makes δ genuinely fixed: when the body premise is instantiated with the current snapshot value, the interpretation

of the pre/post effect remains the same δ rather than a stage-indexed family selected by k.

Claim $S(k)$. For all state pairs (τ_1, τ_2) such that $(\tau_1, \tau_2) \vDash_I P \wedge b_1\langle 1\rangle = b_2\langle 2\rangle \wedge e_v\langle 1\rangle \leq n \mid \delta$ and $[\![e_v\langle 1\rangle]\!]_i(\tau_1, I) = k$: if $(\tau_1, \tau_1', \_) \in [\![\texttt{while } b_1 \texttt{ do } C_1]\!]$ and $(\tau_2, \tau_2', \_) \in [\![\texttt{while } b_2 \texttt{ do } C_2]\!]$, then $(\tau_1', \tau_2') \vDash_I P \wedge \neg b_1\langle 1\rangle \wedge \neg b_2\langle 2\rangle \mid \delta$.

To prove $S(k)$, fix an arbitrary state pair and executions satisfying the antecedent of the claim, and write them again as (σ_1, σ_2), $(\sigma_1, \sigma_1', \_)$, and $(\sigma_2, \sigma_2', \_)$.

Case $k = 0$: From premise (4) and the current P, $e_v\langle 1\rangle \leq 0$ implies $\neg b_1\langle 1\rangle$, so guard $b_1\langle 1\rangle$ is false. From $b_1\langle 1\rangle = b_2\langle 2\rangle$: $b_2\langle 2\rangle$ is also false. By the while semantics (Iter(0)): $\sigma_1' = \sigma_1$ and $\sigma_2' = \sigma_2$. The postcondition $P \wedge \neg b_1\langle 1\rangle \wedge \neg b_2\langle 2\rangle \mid \delta$ holds directly from the precondition. ✓

Case $k > 0$: From the biconditional (contrapositive): $e_v\langle 1\rangle > 0 \implies b_1\langle 1\rangle$. So $b_1\langle 1\rangle$ is true. From $b_1\langle 1\rangle = b_2\langle 2\rangle$: $b_2\langle 2\rangle$ is also true. By the while semantics (Iter($n' + 1$)): both loops execute one body step.

Step 1: One body iteration. Since both guards are true, both loops execute one iteration:

$$(\sigma_1, \sigma_1^{(1)}, \_) \in [\![C_1]\!], \quad (\sigma_2, \sigma_2^{(1)}, \_) \in [\![C_2]\!]$$

The remaining iterations give $(\sigma_1^{(1)}, \sigma_1', \_) \in [\![\texttt{while } b_1 \texttt{ do } C_1]\!]$ and $(\sigma_2^{(1)}, \sigma_2', \_) \in [\![\texttt{while } b_2 \texttt{ do } C_2]\!]$.

Step 2: Apply the body premise. Since the current variant value is k, with $k > 0$ and $k \leq n$, the antecedent $0 < e_v\langle 1\rangle \leq n$ of the body premise is satisfied. The body premise (instantiated with the current snapshot value k) has been proved derivable. By the soundness induction hypothesis for the body premise:

$$(\sigma_1^{(1)}, \sigma_2^{(1)}) \vDash_I P \wedge b_1\langle 1\rangle = b_2\langle 2\rangle \wedge e_v\langle 1\rangle < k \mid \delta$$

Step 3: Apply the induction hypothesis. Let $k_1 = [\![e_v\langle 1\rangle]\!]_i(\sigma_1^{(1)}, I)$. From the body postcondition: $k_1 < k$ and P is re-established. From the claim precondition, $k \leq n$; hence $k_1 < k \leq n$, so $k_1 \leq n$. From the nonnegative-variant premise applied to the re-established P, $0 \leq k_1$. The intermediate state $(\sigma_1^{(1)}, \sigma_2^{(1)})$ therefore satisfies the claim's precondition with natural variant value $k_1 < k$. By well-founded induction, $S(k_1)$ applies:

$$(\sigma_1', \sigma_2') \vDash_I P \wedge \neg b_1\langle 1\rangle \wedge \neg b_2\langle 2\rangle \mid \delta \quad \square$$

Instantiating $S(k_0)$ with the original initial state pair proves the Level 1 fixed-**awhile** semantic obligation. The Level 2 loop rule reuses this postcondition/effect argument; its signed cost-bound calculation is in Section 8.9.

Key points.

- The induction is on the current variant value, NOT on the iteration count. This avoids assuming $n_1 = n_2$ upfront.
- Guard agreement ($b_1\langle 1\rangle = b_2\langle 2\rangle$) is re-established by the body premise at each step. This ensures both loops step together inductively.

- The binary property δ is fixed: the body premise preserves δ.

*Case **awhile-evolving** ($\delta_k \to \delta_{k-1}$).*
Rule recalled.

awhile-evolving premises:
(1) $n \geq 0$,
(2) $k \notin FV(P) \cup FV(b_1) \cup FV(b_2) \cup FV(e_v)$,
(3) $P \implies 0 \leq e_v\langle 1\rangle$,
(4) $P \implies \big(e_v\langle 1\rangle \leq 0 \iff \neg b_1\langle 1\rangle\big)$,
(5) $\forall k.\ 1 \leq k \leq n \implies \vdash \{\delta_k\} \mid \{P \wedge b_1\langle 1\rangle \wedge b_2\langle 2\rangle \wedge e_v\langle 1\rangle = k\}$
$C_1 \sim C_2$
$\{P \wedge b_1\langle 1\rangle = b_2\langle 2\rangle \wedge e_v\langle 1\rangle = k-1\} \mid \{\delta_{k-1}\}$. **awhile-evolving**

awhile-evolving conclusion:
$\vdash \{\delta_n\} \mid \{P \wedge b_1\langle 1\rangle = b_2\langle 2\rangle \wedge e_v\langle 1\rangle = n\}$
$\texttt{while}\ b_1\ \texttt{do}\ C_1 \sim \texttt{while}\ b_2\ \texttt{do}\ C_2$
$\{P \wedge \neg b_1\langle 1\rangle \wedge \neg b_2\langle 2\rangle\} \mid \{\delta_0\}$.

In this rule, δ_k denotes a meta-level family of binary properties indexed by the remaining variant value k; the notation δ_k is the same family when discussing heap effects abstractly. Binding the premise with $\forall k$ selects the family member for that stage. It is not ordinary free-variable capture inside a single fixed δ.

Recall the essential premises of the rule. The bound n is a meta-level natural number (if represented by a logical symbol in an implementation, read the following induction as being on $[\![n]\!]_i(I)$). We require $n \geq 0$, the same freshness and nonnegative-variant premises as the fixed rule, the exit biconditional $P \implies (e_v\langle 1\rangle \leq 0 \iff \neg b_1\langle 1\rangle)$, and for every stage t with $1 \leq t \leq n$ a body judgment from pre-effect δ_t and precondition $P \wedge b_1\langle 1\rangle \wedge b_2\langle 2\rangle \wedge e_v\langle 1\rangle = t$ to post-effect δ_{t-1} and postcondition $P \wedge b_1\langle 1\rangle = b_2\langle 2\rangle \wedge e_v\langle 1\rangle = t-1$. The conclusion starts from δ_n and $e_v\langle 1\rangle = n$ and exits with δ_0.

By induction on the meta-level initial variant value n. The side condition $n \geq 0$ and the precondition $e_v\langle 1\rangle = n$ put the initial variant in $\mathbb{N}$; the exact-decrement premise keeps the recursive variant values in $\mathbb{N}$. The exact-decrement premise together with the biconditional $e_v\langle 1\rangle \leq 0 \iff \neg b_1\langle 1\rangle$ ensures the loop executes exactly n iterations.

Assume $(\sigma_1, \sigma_2) \vDash_I P \wedge b_1\langle 1\rangle = b_2\langle 2\rangle \wedge e_v\langle 1\rangle = n \mid \delta_n$, and both loops execute from these states to produce (σ_1', σ_2').

Base case ($n = 0$): From $e_v\langle 1\rangle = 0$ and the biconditional ($\Rightarrow$ direction), $\neg b_1\langle 1\rangle$ holds. From $b_1\langle 1\rangle = b_2\langle 2\rangle$: $\neg b_2\langle 2\rangle$. By while semantics (Iter(0)): $\sigma_r' = \sigma_r$. Postcondition: $P \wedge \neg b_1\langle 1\rangle \wedge \neg b_2\langle 2\rangle$ from precondition. Binary property: $\delta_n = \delta_0$. ✓

Inductive case ($n > 0$): From $e_v\langle 1\rangle = n > 0$ and the biconditional ($\Leftarrow$ direction, contrapositive): $b_1\langle 1\rangle$ is true. From $b_1\langle 1\rangle = b_2\langle 2\rangle$: $b_2\langle 2\rangle$ is true. Both loops execute one body step:

$$(\sigma_1, \sigma_1^{(1)}, \_) \in [\![C_1]\!], \quad (\sigma_2, \sigma_2^{(1)}, \_) \in [\![C_2]\!]$$

Instantiate the body premise with the stage value $t = n$. By the soundness induction hypothesis for the body premise:

$$(\sigma_1^{(1)}, \sigma_2^{(1)}) \vDash_I P \wedge b_1\langle 1\rangle = b_2\langle 2\rangle \wedge e_v\langle 1\rangle = n-1 \mid \delta_{n-1}$$

Why $k \notin FV(P)$ is critical: The notation δ_{n-1} denotes the meta-level family member selected for the next stage. If a mechanization represents this family syntactically by a k-indexed β, then δ_{n-1} means the result of substituting the stage value $n-1$ before interpreting the effect. Since $k \notin FV(P) \cup FV(b_1) \cup FV(b_2) \cup FV(e_v)$, the assertion and guard/variant facts are stable under the corresponding stage rebinding; only the selected effect family member changes.

The remaining loop iterations execute from $(\sigma_1^{(1)}, \sigma_2^{(1)})$ with $e_v\langle 1\rangle = n-1$ and effect δ_{n-1}. By induction on $n-1$:

$$(\sigma_1', \sigma_2') \vDash_I P \wedge \neg b_1\langle 1\rangle \wedge \neg b_2\langle 2\rangle \mid \delta_0$$

The effect chain composes: $\delta_n \to \delta_{n-1} \to \cdots \to \delta_0$. □

The Level 2 evolving loop rule reuses this postcondition/effect argument; its signed cost-bound calculation is in Section 8.9.

Note. The fixed-δ **awhile** is a special case where $\delta_k = \delta$ for all k.

Use case: Programs where f creates new equalities (e.g., constant $f(x) = 0$), so β grows per iteration as **update1** confirms Φ at each processed position. For general f, InplaceMap uses the fixed-δ **awhile** (see Section 8.11).

*Case **re-sync-derived** (β-recovery after async one-sided operations). Rule recalled for reference.*

re-sync-derived premises:

(1) $\vdash \{(u_1, u_2) \mapsto^{\Phi} \beta_0\} \mid \{P\}$
$C_1 \sim \texttt{skip} \quad \{P_{mid}\} \mid \{\delta_{mid}\}$,

(2) $\vdash \{\delta_{mid}\} \mid \{P_{mid}\}$
$\texttt{skip} \sim C_2 \quad \{P'\} \mid \{\delta_{out}\}$,

(3) $Recover(P', u_1, u_2, \beta_0, \Phi_{rec})$,

(4) $PreserveLen(C_1, C_2, u_1, u_2, \beta_0)$,

(5) Φ_{rec} well-formed.

re-sync-derived

re-sync-derived conclusion:

$\vdash \{(u_1, u_2) \mapsto^{\Phi} \beta_0\} \mid \{P\}$
$C_1 \sim C_2 \quad \{P'\} \mid \{(u_1, u_2) \mapsto^{\Phi_{rec}} \beta_0\}$.

We prove validity of the derived conclusion from the validity of its two one-sided premises. Let

$$\vdash \{(u_1, u_2) \mapsto^{\Phi} \beta_0\} \mid \{P\}\ C_1 \sim \texttt{skip}\ \{P_{mid}\} \mid \{\delta_{mid}\} \quad \text{and} \quad \vdash \{\delta_{mid}\} \mid \{P_{mid}\}\ \texttt{skip} \sim C_2\ \{P'\} \mid \{\delta_{out}\}$$

be the two premise judgments. By the induction hypothesis for the Level 1 soundness theorem, or equivalently by assuming semantic validity of the recalled premise judgments in this admissibility proof, both one-sided premises are semantically valid.

Fix any well-formed interpretation I covering the free logical variables, and fix terminating executions $(\sigma_1, \sigma_1', m_1) \in [\![C_1]\!]$ and $(\sigma_2, \sigma_2', m_2) \in [\![C_2]\!]$ such that

$$(\sigma_1, \sigma_2) \vDash_I P \mid (u_1, u_2) \mapsto^{\Phi} \beta_0.$$

The skip command leaves the other side unchanged, so $(\sigma_2, \sigma_2, 0) \in [\![\texttt{skip}]\!]$. Applying validity of the first premise to the left execution and this skip execution gives

$$(\sigma_1', \sigma_2) \vDash_I P_{mid} \mid \delta_{mid}.$$

Similarly, $(\sigma_1', \sigma_1', 0) \in [\![\texttt{skip}]\!]$. Applying validity of the second premise to this skip execution and the right execution gives

$$(\sigma_1', \sigma_2') \vDash_I P' \mid \delta_{out}.$$

This is the same final pair as the independent execution of $C_1 \sim C_2$ in Definition 6.6; no paired operational step is being assumed.

It remains to prove the recovered post-effect named in the conclusion. The third recalled premise supplies value recovery, and the fourth recalled premise supplies the post-state allocation and index-bound facts needed to interpret the recovered binary property. Since $(\sigma_1', \sigma_2') \vDash_I^r P'$, the semantic recovery premise yields

$$\forall j \in [\![\beta_0]\!]_b(I).\ \Phi_{rec}(h_1'(u_1)[j], h_2'(u_2)[j]),$$

where $\sigma_i' = (s_i', h_i')$. The *PreserveLen* side condition states, for these same terminating executions, that $u_1 \in dom(h_1')$ and $u_2 \in dom(h_2')$, and that every $j \in [\![\beta_0]\!]_b(I)$ remains in bounds for both post-state arrays:

$$j < len(h_1'(u_1)) \quad \text{and} \quad j < len(h_2'(u_2)).$$

Well-formedness of β_0 as an index domain supplies $0 \le j$ for every $j \in [\![\beta_0]\!]_b(I)$. Hence

$$[\![\beta_0]\!]_b(I) \subseteq [0, \min(len(h_1'(u_1)), len(h_2'(u_2)))).$$

Together with post-state array existence and well-formedness of Φ_{rec}, Definition 6.2 therefore gives

$$(\sigma_1', \sigma_2') \vDash_I (u_1, u_2) \mapsto^{\Phi_{rec}} \beta_0.$$

Combining this binary-property fact with $(\sigma_1', \sigma_2') \vDash_I^r P'$ proves

$$(\sigma_1', \sigma_2') \vDash_I P' \mid (u_1, u_2) \mapsto^{\Phi_{rec}} \beta_0,$$

which is exactly the semantic validity of the **re-sync-derived** conclusion. The conclusion deliberately states the recovered post-effect rather than δ_{out}. No affine-resource assumption is used here: semantic validity only requires proving the post-effect named in the conclusion. Any residual binary facts needed after recovery must be included by an additional framing or consequence step. □

The graded derived rule in Section 8 reuses this postcondition/effect argument. Its additive grade calculation is handled with the other Level 2 cost-bound cases in Section 8.9.

Note. The recovery premise is an explicit proof obligation discharged by the user. It is NOT checked by the CHC solver. For insertion sort, the obligation is the semantic lemma that the postcondition of the two completed insertion steps entails pointwise ε-closeness again over the original domain β_0. The rule separates the generic mechanism (β-recovery) from the domain-specific correctness argument.

7 Level 1 Example: Monotonicity of CountPositive

We demonstrate the Level 1 relational logic on a *qualitative* property: monotonicity of CountPositive. This contrasts with the Level 2 *quantitative* derivation in Section 8, which bounds the cost difference $c\langle 1\rangle - c\langle 2\rangle \leq n - |\beta|$.

7.1 The Program

```
CountPositive(A, n):
    c := 0; i := 0
    while (i < n):
        x := A[i]
        if x > 0: c++
        i++
    return c
```

Two runs: P_1 on array A_1, P_2 on array A_2.

7.2 The Property (Monotonicity)

If every element of A_1 is at most the corresponding element of A_2, then $\mathit{CountPositive}(A_1) \leq \mathit{CountPositive}(A_2)$.

Formally, with $\Phi(x, y) = (x \leq y)$ and $\beta = [0, n)$ (all positions satisfy Φ), where n is a logical snapshot of the common input length ($n\langle 1\rangle = n\langle 2\rangle = N$) so that β is interpretation-level:

$$\vdash \{(A_1, A_2) \mapsto^{\leq} [0, n)\} \mid \{n \geq 0\}\ \ P_1 \sim P_2\ \ \{c\langle 1\rangle \leq c\langle 2\rangle\} \mid \{(A_1, A_2) \mapsto^{\leq} [0, n)\}$$

Key difference from Level 2. In the Level 2 cost derivation (Section 8), $\Phi(x, y) = (x = y)$ and $\beta \subsetneq [0, n)$ (some positions differ). This requires a grade to bound the cost difference at non-β positions. Here, $\Phi(x, y) = (x \leq y)$ and $\beta = [0, n)$ (every position satisfies Φ), so no grade is needed; the monotonicity relation is tracked purely through the assertion P.

7.3 Derivation

Step 1: Initialization. Using **assign** and **sequence**:

$$\vdash \{\delta\} \mid \{n \geq 0\} \;\; c_1 := 0; i_1 := 0 \sim c_2 := 0; i_2 := 0 \;\; \{I\} \mid \{\delta\}$$

where the loop invariant is:

$$I \equiv i\langle 1\rangle = i\langle 2\rangle \wedge 0 \leq i \leq n \wedge c\langle 1\rangle \leq c\langle 2\rangle$$

Step 2: While loop. Using **awhile** (lockstep, δ fixed since arrays are read-only):

$$\frac{\vdash \{\delta\} \mid \{I \wedge i < n\} \;\; \mathit{body}_1 \sim \mathit{body}_2 \;\; \{I\} \mid \{\delta\}}{\vdash \{\delta\} \mid \{I\} \;\; \textbf{while } (i < n) \textbf{ do } \mathit{body}_1 \sim \textbf{while } (i < n) \textbf{ do } \mathit{body}_2 \;\; \{I \wedge i \geq n\} \mid \{\delta\}}$$

Guards agree: $i\langle 1\rangle = i\langle 2\rangle$ (from I), so $(i < n)\langle 1\rangle = (i < n)\langle 2\rangle$. ✓

The **awhile** variant is $e_v = n - i\langle 1\rangle$: under I we have $0 \leq e_v \leq n$, and $e_v \leq 0 \iff i\langle 1\rangle \geq n \iff \neg(i < n)$, so the variant reaches 0 exactly when the guard fails; each iteration executes $i{+}{+}$, decreasing e_v by one, which discharges the strict-decrease premise of **awhile**.

Step 3: Loop body: Read. Using **read1** (since $i \in \beta = [0, n)$ at every iteration):

$$\vdash \{\delta\} \mid \{I \wedge i < n\} \;\; x_1 := A_1[i] \sim x_2 := A_2[i] \;\; \{I \wedge i < n \wedge \Phi(x_1, x_2)\} \mid \{\delta\}$$

We learn $x_1 \leq x_2$. Note: in Level 2 with $\Phi(x, y) = (x = y)$, some positions would use **read2** (no Φ info), requiring grade accounting. Here, **read1** applies everywhere.

Step 4: Loop body: Conditional (case split). From $x_1 \leq x_2$, the guards $(x_1 > 0)$ and $(x_2 > 0)$ may *differ*: $x_1 \leq 0 < x_2$ is possible. So we cannot use the two-sided **conditional** rule (which requires $b\langle 1\rangle = b\langle 2\rangle$). Instead, we use **conditionalL** on $x_1 > 0$:

Case $x_1 > 0$: Since $x_1 \leq x_2$ and $x_1 > 0$, we have $x_2 > 0$. Under **conditionalL** the right command is still `if` $x_2 > 0$ `then` $c_2{+}{+}$ `else skip`; we discharge it with **conditionalR**, whose $x_2 \leq 0$ premise carries the unsatisfiable precondition $x_1 \leq x_2 \wedge x_1 > 0 \wedge x_2 \leq 0$, leaving only the $x_2 > 0$ branch in which both runs increment c:

$$c_1' = c_1 + 1, \quad c_2' = c_2 + 1 \quad \Longrightarrow \quad c_1' \leq c_2' \quad \checkmark$$

Case $x_1 \leq 0$: Guard $x_1 > 0$ is false on the left, so $C_1 = \textbf{skip}$. We use **conditionalR** on $x_2 > 0$ for the right side:

- **Sub-case $x_2 > 0$:** Left skips, right increments. Using **assignR-skipL**: $c_1' = c_1$, $c_2' = c_2 + 1$, so $c_1' \leq c_2'$ (gap widens). ✓
- **Sub-case $x_2 \leq 0$:** Both skip. $c_1' = c_1 \leq c_2 = c_2'$. ✓

In all cases, $c\langle 1\rangle \leq c\langle 2\rangle$ is preserved. After incrementing i (using **assign**): I is restored with $i' = i + 1$.

Step 5: Termination. At loop exit: $I \wedge i \geq n$, giving $c\langle 1\rangle \leq c\langle 2\rangle$. ✓

7.4 Rules Used

Proof Step	Rule	Why
Array read	**read1**	$i \in \beta = [0, n)$ always; learn $\Phi(x_1, x_2)$
Both increment ($x_1 > 0$)	**conditionalL** then **conditionalR**	$x_1 > 0 \Rightarrow x_2 > 0$ (from Φ); the $x_2 \leq 0$ premise is vacuous
Different branch	**conditionalL** + **conditionalR**	When $x_1 \leq 0$ but x_2 may be positive
Left skips, right acts	**assignR-skipL**	c_2 increments, c_1 unchanged
Lockstep loop	**awhile**	Guards agree (from $i\langle 1\rangle = i\langle 2\rangle$)

No grade needed. Every iteration uses **read1** (since $\beta = [0, n)$) and the output relation $c\langle 1\rangle \leq c\langle 2\rangle$ is maintained through P; no cost accounting is required. This is the hallmark of Level 1: qualitative properties via δ and P alone.

7.5 Connection to Level 1 CHC

The derivation maps to Level 1 cell morphing as a derivation-guided correspondence, not as a completeness theorem from CHC solutions back to logic proofs. The logic proof fixes $\beta = [0, n)$ and uses **read1** at every iteration. The Level 1 CHC encoding instead focuses on a distinguished cell k and checks the local HIT/MISS obligations generated by that same proof shape. Because k is arbitrary, the fixed-cell obligations represent the all-index property required by the logical binary property. The CHC clauses are universally quantified over their free variables, so this is not one chosen constant; every admissible focused cell must satisfy the local obligation.

	Logic	**Level 1 CHC**
β	Known: $\beta = [0, n)$	Arbitrary focused cell k; prove the local obligation for that cell
At the focused cell	**read1** gives $\Phi(x_1, x_2)$	HIT at $i = k$: witness gives $ak_1 \leq ak_2$
Away from the focused cell	Same proof obligation, by arbitrary k	MISS branch carries no cell fact for the current focus
Property	$c\langle 1\rangle \leq c\langle 2\rangle$ in P	Goal clause rules out or asserts violation of $c_1 \leq c_2$
Cost/grade	None	No cost variable
PickK	None (logic knows β)	None (fixed k)

Invariant correspondence. The logic loop invariant I corresponds to the CHC invariant predicate $Inv(\ldots)$: shared scalar facts such as $i\langle 1\rangle = i\langle 2\rangle$ and $c\langle 1\rangle \leq c\langle 2\rangle$ become invariant arguments or constraints, while the binary property provides the witness relation at the focused cell. The **read1** proof case corresponds to the HIT transition where the witness supplies $ak_1 \leq ak_2$. The conditional proof splits correspond to CHC transition clauses: when both guards agree, the synchronized branch preserves the assertion; when only the right side increments, the one-sided assignment case preserves $c\langle 1\rangle \leq c\langle 2\rangle$ because the gap widens in the safe direction. The final CHC goal is just the loop-exit use of the postcondition, written either as a direct assertion or, in violation-polarity encodings, as an unsatisfiable clause for $c_1 > c_2$.

Why Level 1 suffices. For monotonicity, Φ holds at *every* position ($\beta = [0, n)$). The fixed-k cell morphing discharges the per-cell HIT obligation $\Phi(ak_1, ak_2)$ at the focused k; since the CHC clauses are universally quantified over k, this stands in for the all-index property. A single HIT does not by itself discharge the other loop iterations; those rely on the derivation-guided MISS summary supplied by the universal $\beta = [0, n)$ fact. No PickK is needed because there is nothing to "search" for: every position is in β.

Existing Level 1 CHC encodings for monotonicity include `monotone_arraysum_final.clp` (ArraySum) and `threshold_filter_mono.clp` (threshold filter), which follow this same pattern.

8 Level 2: Graded Relational Logic

This section extends the Level 1 relational logic with a *semantic grade*, corresponding to **Level 2** of the relational cell morphing framework. Level 2 covers *quantitative* relational properties, including relational cost bounds like $c\langle 1\rangle - c\langle 2\rangle \leq n - |\beta|$ (signed, in the spirit of ARel), where the grade m semantically bounds the relational cost difference $m_1 - m_2$ via the cost-instrumented operational semantics defined below.

8.1 Graded Judgment

The graded relational judgment has the form (we write m as a superscript on the turnstile, matching ARel/RelCost notation):

$$\vdash^m \{\delta\} \mid \{P\}\ C_1 \sim C_2\ \{Q\} \mid \{\delta'\}$$

where m is a grade term (a natural-valued derived grade term over logical/index variables; Lemma 8.1). All other components are as in the Level 1 judgment (Section 5). The grade carries the *semantic signed upper bound on the relational cost difference*: a derivable judgment guarantees $m_1 - m_2 \leq [\![m]\!]_i(I)$ for any pair of cost-instrumented executions of C_1 and C_2 from a precondition state (Definition 8.1).

(For typesetting compatibility with the unary logic in this manuscript, the macro form $\vdash \{\delta\} \mid \{P\}\ C_1 \sim C_2\ \{Q\} \mid \{\delta'\} \mid m$ is used in the rule statements below; it denotes the same judgment as the $\vdash^m$ form above.)

8.2 Graded Relational Validity (Level 2)

Grade interpretation. The relational grade m is the *semantic signed upper bound on the relational cost difference* $m_1 - m_2$. A derivable judgment with grade m guarantees that, for every pair of executions of C_1 and C_2 from a state pair satisfying the precondition $P \mid \delta$, the operational cost difference $m_1 - m_2$ between the two runs is bounded by $[\![m]\!]_i(I)$. The grade is *not* erased: it carries the same kind of semantic content as the unary grade in the unary logic (Section 2), lifted to the relational setting. We follow ARel/RelCost in using a signed (oriented) relational cost difference: the bound applies to "left run minus right run." A symmetric bound $|m_1 - m_2| \leq m$ is recovered by deriving both the judgment and its swapped variant ($C_2 \sim C_1$ with the same grade).

Cost-instrumented operational semantics. The validity definition uses the same cost-instrumented operational semantics as the unary logic. Every command evaluation $(\sigma, \sigma', m_1) \in [\![C]\!]$ produces both a final state σ' and a nonnegative integer cost m_1 accumulated from the per-construct weights (Section 3.2.3). At Level 1 the cost component is discarded; at Level 2 the signed difference $m_1 - m_2$ between the two runs' costs is the quantity bounded by $[\![m]\!]$.

For example, in CountPositive, the derivation has grade $m = n - |\beta|$ (one one-sided cost-relevant rule application per non-β position), and validity guarantees

$c\langle 1\rangle - c\langle 2\rangle \leq n - |\beta \cap [0, n)|$ for any pair of executions consistent with the precondition. The bound is stated semantically by the grade rather than encoded in the postcondition Q, and corresponds to the encoding's CHC goal $c \leq n - d_0$ (also signed) as the threshold weakening of the exact grade $n - |\beta \cap [0, n)|$ under $d_0 \leq |\beta \cap [0, n)|$ (they coincide only when $d_0 = |\beta \cap [0, n)|$).

Definition 8.1 (Graded relational validity, Level 2)**.** *A graded relational statement* $\vdash \{\delta\} \mid \{P\}\ C_1 \sim C_2\ \{Q\} \mid \{\delta'\} \mid m$ *is* valid*, written*

$$\vDash^m \{\delta\} \mid \{P\}\ C_1 \sim C_2\ \{Q\} \mid \{\delta'\}$$

if for every well-formed *interpretation* I *(with* $dom(I) \supseteq LV(P, Q, \delta, \delta', m)$ *and* $[\![m]\!]_i(I) \in \mathbb{N}$ *defined under* I*), and every pair of* finite terminating *cost-instrumented executions* $(\sigma_1, \sigma_1', m_1) \in [\![C_1]\!]$ *and* $(\sigma_2, \sigma_2', m_2) \in [\![C_2]\!]$*:*

$$\text{if } (\sigma_1, \sigma_2) \vDash_I P \mid \delta, \quad \text{then} \quad (\sigma_1', \sigma_2') \vDash_I Q \mid \delta' \quad \text{and} \quad m_1 - m_2 \leq [\![m]\!]_i(I).$$

Well-formedness of grade terms. *The grade term* m *is restricted to depend only on logical/index variables in* $\mathrm{LV}(P, Q, \delta, \delta')$ *and grade-language constants, not on mutable program variables of the runs (so that* $[\![m]\!]_i(I)$ *is determined by* I *alone, independent of* σ_1, σ_2*). This restriction guarantees that the grade has a stable denotation across all states satisfying the precondition. In particular, the array size* n *that appears in grades and indices (e.g.* $m = n - |\beta|$*) is a* read-only size parameter*: it is fixed for the run and never assigned, so we treat it as a logical/index variable rather than a mutable program variable, which is what admits its use in grade and index terms.*
Sign convention. *The bound is* signed *(left run minus right run) and* one-sided*: it bounds* $m_1 - m_2$ *from above, not from below. The grade* m *ranges over* $\mathbb{N}$ *(the grade-algebra monoid; see "Grade algebra" below). When the right run is structurally costlier than the left,* $m_1 - m_2$ *may be negative; the bound* $m_1 - m_2 \leq m$ *still holds (since* $m \geq 0$*) but is not informative in that direction. To bound the symmetric direction* $m_2 - m_1 \leq m'$*, derive the swapped judgment* $\vdash \{\delta\} \mid \{P\}\ C_2 \sim C_1\ \{Q\} \mid \{\delta'\} \mid m'$ *separately.*

The two programs continue to execute *independently*; the relational structure arises from the pre/postconditions, the binary property δ, and now the grade-based bound on the cost difference. There is no paired operational semantics; the grade bound is a relational fact about the two cost components produced by the two independent runs.

Scope: default execution-cost effect instance. The semantic grade m formally bounds the signed difference of the total execution costs accumulated from the *default per-construct costs*: $m_1 - m_2$ under the cost model $c_{\texttt{skip}} = 0$, $c_{assign} = c_{read} = c_{update} = c_{alloc} = 1$, and $c_{if} = c_{while} = 0$. The proof obligation reduces to bounding how many asymmetric one-sided cost-bearing constructs fire, and adding their per-rule bounds. Under this literal execution-cost reading, the *logic* soundness theorem (Theorem 8.2) covers 15 of the 22

Level 2 encodings: 13 K=1 unit-asymmetric counters, the K=2 **DualCount** example (two independent unit counter updates in one loop body), and the pure execution-cost **InplaceMap** example. (This is the *logic* soundness scope. The separate *CHC translation* theorem of Section 10.6 (Theorem 10.6) is proved for the read-only single-array fragment; the mutable-array InplaceMap is covered by the mutable-array extension, Theorem 10.11 (Section 10.8), which uses the same unknown signature with a net cell-update map in Clause 2.)

Among the seven non-execution-cost cases detailed below, **ClampPositive** shares the relational invariant skeleton (lockstep loop, PickK/Wit cell observation, 0/1 equal/unequal observation driving an additive scalar recurrence) used by the Theorem 8.2 examples, but its property is an output-Hamming upper bound, not a per-construct execution-cost difference. We therefore treat ClampPositive as a **Level 1 quantitative-invariant** case: the encoding's counter c *upper-bounds* the output-Hamming distance via the cell-morphing abstraction (search-phase tight; settled/MISS conservatively over-approximates), and the bound $c \leq n - d_0$ is established by a relational invariant over that counter, *not* via the execution-cost grade. Concretely, and this is a generic feature of the output-Hamming effect, not specific to ClampPositive's trace, Hamming can increase on a paired same-construct *write* step where both runs execute the same update but store different values, which contributes zero to the execution-cost grade: exactly the discrepancy that forbids reading the cost grade as a Hamming bound. ClampPositive is the $M = 1$ instance of the value-accumulator pattern described next and is excluded from Theorem 8.2.

The remaining seven Level 2 encodings are Level 1 quantitative-invariant cases. Their encoding variable c tracks a quantity other than per-construct execution cost. This includes **ClampPositive** (the $M = 1$ instance: c upper-bounds the output-Hamming distance via the cell-morphing abstraction), **NestedBranch** (a source counter updated by `cost += 2`; under the default execution-cost model this is one assignment, not two construct costs), **CompressPositive** (output-length difference), and the four value-weighted accumulators **arraysum_mixed**, **conditional_cost**, **threshold_filter_bounded**, and **ArrayDoubleOpNI_robust_hFT**. These are verified by plain relational Hoare reasoning with quantitative invariants over the relevant scalar (output-disagreement counter, source variable, or value-weighted accumulator); their soundness is the Level-1 relational soundness theorem (Theorem 6.3) applied to that quantitative invariant, *not* the graded Theorem 8.2. For value-weighted examples the invariant supplies a tracked equal subset and a pointwise complement bound M, yielding bounds of the form $M \cdot (n - d_0)$; ClampPositive is the $M = 1$ specialization. The execution-cost grade theorem (Theorem 8.2) itself is not invoked for any of these cases; they are covered by the generic graded-effect instances of Section 9 (Corollaries 9.4 and 9.5), which formalize exactly these quantitative-invariant arguments.

Scope sentence. *We bound relational execution-cost divergence for instrumented unit or explicitly bounded cost events; data-valued resource accumula-*

tion is outside the execution-cost instance unless reduced by independent unary or relational numeric bounds. It is brought in scope by the affine-accumulator instance of the generic graded-effect theorem (Section 9), whose precondition-bounded increments realize exactly those numeric bounds.

Grade algebra. Grades form an additive monoid $(\mathbb{N}, 0, +)$ with a partial order $\leq$:

- **Identity**: 0 (two-sided rule, no divergent cost).
- **Composition**: $m_1 + m_2$ (sequential composition).
- **Branching**: $\max(m_1, m_2)$ (conditional, worst-case branch).
- **Iteration**: $\sum_{k=1}^{n} m_k$ (loop, per-iteration sum).
- **Weakening**: if $m' \leq m$, then a derivation with grade m' can be weakened to grade m (via **conseq**).

Comparison with ARel's grade D. In ARel [2], the relational judgment $\vdash t_1 \ominus t_2 \lesssim D : \tau$ includes a grade D whose soundness theorem (Theorem 4.2) guarantees $\mathit{cost}(t_1) - \mathit{cost}(t_2) \leq D$ via a step-indexed logical relation. Our grade m now plays the same role: a semantic upper bound on the relational cost difference. Both grades are propagated additively through sequencing, via max through conditionals, and via $\sum$ through loops.

The differences are in framing rather than substance. ARel is a type-and-effect system over a higher-order language with a monadic relational type $\{P\}\ \exists \vec{y}.\tau\ \{Q\}$ and $\mathit{diff}(D)$ separating spatial from quantitative information. Our setting is a Hoare-style logic over an imperative array language, with δ playing the spatial role analogous to ARel's P/Q and m playing the role of D. The cell-morphing assertion $(u_1, u_2) \mapsto^{\Phi} \beta$ and the prophecy-style existential β-choice are first-class in our δ framework, and the CHC translation in Section 10.6 realizes them algorithmically.

The scope where our grade and ARel's D behave equivalently is the default execution-cost effect instance above: cost events with unit or rule-compatible statically bounded weight. Value-weighted source accumulation is handled separately by independent numeric invariants; neither framework encodes array-value sums directly in the grade language. In examples such as CountPositive, the derivation grade $m = n - |\beta|$ coincides numerically with the cost-difference bound from ARel's R-S–style reasoning.

Rule-level grading principle. The proof system realizes the semantic grade compositionally:

- **Two-sided** operations (both runs execute the same construct): grade 0.
- **One-sided** operations (one run acts, the other idles): grade 1 when the *left* run acts (signed effect $+1$), grade 0 when the *right* run acts (signed effect -1, least natural upper bound 0).

- **Composition**: sequence adds ($m_1 + m_2$), conditionals take the max under guard agreement, loops sum per iteration ($\sum_k m_k$).

The per-rule cost-difference bounds and their soundness are discharged in Section 8.9.

8.3 Graded Two-Sided Rules

All two-sided rules carry grade 0, since both runs execute the same construct. We present the key rules; the remaining two-sided rules (**alloc1**, **alloc2**) extend analogously.

$$\frac{}{\vdash \{\delta\} \mid \{P\}\ \texttt{skip} \sim \texttt{skip}\ \{P\} \mid \{\delta\}\ \mid 0}\ \textbf{skip}$$

$$\frac{x_1\langle 1\rangle \notin FV(\delta) \qquad x_2\langle 2\rangle \notin FV(\delta)}{\vdash \{\delta[e_1\langle 1\rangle/x_1\langle 1\rangle][e_2\langle 2\rangle/x_2\langle 2\rangle]\} \mid \{Q[e_1\langle 1\rangle/x_1\langle 1\rangle][e_2\langle 2\rangle/x_2\langle 2\rangle]\}\ x_1 := e_1 \sim x_2 := e_2\ \{Q\} \mid \{\delta\}\ \mid 0}\ \textbf{assign}$$

$$\frac{\models P \implies b_1\langle 1\rangle = b_2\langle 2\rangle \quad \vdash \{\delta\} \mid \{P \wedge b_1\langle 1\rangle\}\ C_1 \sim C_2\ \{Q\} \mid \{\delta'\}\ \mid m \qquad \vdash \{\delta\} \mid \{P \wedge \neg b_1\langle 1\rangle\}\ C_1' \sim C_2'\ \{Q\} \mid \{\delta'\}\ \mid m}{\vdash \{\delta\} \mid \{P\}\ \ \texttt{if}\ b_1\ \texttt{then}\ C_1\ \texttt{else}\ C_1' \sim\ \texttt{if}\ b_2\ \texttt{then}\ C_2\ \texttt{else}\ C_2'\ \{Q\} \mid \{\delta'\}\ \mid m}\ \textbf{conditional}$$

$$\frac{\begin{array}{c} j \text{ fresh} \qquad x_1, x_2 \notin FV(\beta) \\ P = Q[u_1[e_1]/x_1\langle 1\rangle][u_2[e_2]/x_2\langle 2\rangle] \qquad P \implies e_1\langle 1\rangle = e_2\langle 2\rangle = j \wedge j \in \beta \end{array}}{\vdash \{(u_1, u_2) \mapsto^{\Phi} \beta\} \mid \{P\}\ x_1 := u_1[e_1] \sim x_2 := u_2[e_2]\ \{Q \wedge \Phi(x_1, x_2)\} \mid \{(u_1, u_2) \mapsto^{\Phi} \beta\}\ \mid 0}\ \textbf{read1}$$

$$\frac{\begin{array}{c} j \text{ fresh} \qquad x_1, x_2 \notin FV(\beta) \\ P = Q[u_1[e_1]/x_1\langle 1\rangle][u_2[e_2]/x_2\langle 2\rangle] \qquad P \implies e_1\langle 1\rangle = e_2\langle 2\rangle = j \end{array}}{\vdash \{(u_1, u_2) \mapsto^{\Phi} \beta\} \mid \{P\}\ x_1 := u_1[e_1] \sim x_2 := u_2[e_2]\ \{Q\} \mid \{(u_1, u_2) \mapsto^{\Phi} \beta\}\ \mid 0}\ \textbf{read2}$$

$$\frac{\begin{array}{c} j \text{ fresh} \\ P = Q[u_1[e_1\langle 1\rangle \mapsto e_1'\langle 1\rangle]/u_1][u_2[e_2\langle 2\rangle \mapsto e_2'\langle 2\rangle]/u_2] \\ P \implies e_1\langle 1\rangle = e_2\langle 2\rangle = j \\ P \implies 0 \le j < \min(len(u_1)\langle 1\rangle, len(u_2)\langle 2\rangle) \\ P \implies \Phi(e_1'\langle 1\rangle, e_2'\langle 2\rangle) \end{array}}{\vdash \{(u_1, u_2) \mapsto^{\Phi} \beta\} \mid \{P\}\ u_1[e_1] := e_1' \sim u_2[e_2] := e_2'\ \{Q\} \mid \{(u_1, u_2) \mapsto^{\Phi} \beta \cup \{j\}\}\ \mid 0}\ \textbf{update1}$$

$$\frac{\begin{array}{c} j \text{ fresh} \\ P = Q[u_1[e_1\langle 1\rangle \mapsto e_1'\langle 1\rangle]/u_1][u_2[e_2\langle 2\rangle \mapsto e_2'\langle 2\rangle]/u_2] \\ P \implies e_1\langle 1\rangle = e_2\langle 2\rangle = j \\ P \implies 0 \le j < \min(len(u_1)\langle 1\rangle, len(u_2)\langle 2\rangle) \end{array}}{\vdash \{(u_1, u_2) \mapsto^{\Phi} \beta\} \mid \{P\}\ u_1[e_1] := e_1' \sim u_2[e_2] := e_2'\ \{Q\} \mid \{(u_1, u_2) \mapsto^{\Phi} \beta - \{j\}\}\ \mid 0}\ \textbf{update2}$$

$$\frac{\begin{array}{c} P \implies Q \\ arrFV(Q) \cap \{u_1\langle 1\rangle, u_2\langle 2\rangle\} = \emptyset \\ \ell \text{ fresh logical index} \quad \ell \notin FV(Q) \\ P \implies e_1\langle 1\rangle = \ell \wedge e_2\langle 2\rangle = \ell \wedge 0 \le \ell \\ P \implies \Phi(e_1'\langle 1\rangle, e_2'\langle 2\rangle) \end{array}}{\vdash \{emp\} \mid \{P\}\ u_1 \leftarrow array(e_1)(e_1') \sim u_2 \leftarrow array(e_2)(e_2')\ \{Q\} \mid \{(u_1, u_2) \mapsto^{\Phi} [0, \ell - 1]\}\ \mid 0}\ \textbf{alloc1}$$

$$\frac{\begin{array}{c} P \implies Q \\ arrFV(Q) \cap \{u_1\langle 1\rangle, u_2\langle 2\rangle\} = \emptyset \\ P \implies 0 \le e_1\langle 1\rangle \wedge 0 \le e_2\langle 2\rangle \end{array}}{\vdash \{emp\} \mid \{P\}\ u_1 \leftarrow array(e_1)(e_1') \sim u_2 \leftarrow array(e_2)(e_2')\ \{Q\} \mid \{(u_1, u_2) \mapsto^{\Phi} \emptyset\}\ \mid 0}\ \textbf{alloc2}$$

In **alloc1**, ℓ is a ghost logical snapshot of the (equal) allocation length, not a program variable, so no program expression appears inside the β component; $[0, \ell - 1]$ denotes $\emptyset$ when $\ell = 0$. This mirrors the Level 1 **alloc1** rule and keeps the set index heap-independent.

Graded awhile (synchronous, fixed δ). Two loops execute in lockstep with synchronized guards. The nonnegative variant $e_v\langle 1\rangle$ controls the loop-exit condition (left run); guard agreement transfers it to the right run. Each iteration's body is checked at every variant value k, with body grade m_k. The total grade is the sum $\sum_{k=1}^{n} m_k$. The binary property δ is invariant across iterations.

Reading guide:

- Premise 1: nonnegative variant pinpoints loop exit on left run via biconditional.
- Premise 2: each m_k is nonnegative, so grades for unvisited variant values may be added safely when weakening to the full sum.
- Premise 3: $\forall k$, body re-establishes P, re-establishes guard agreement, decreases variant strictly. Body grade is m_k.
- Fixed δ must be k-free; otherwise the rule is the evolving-δ_k rule below.
- Conclusion: while-loop with total grade $\sum_{k=1}^{n} m_k$.

$$\frac{\begin{array}{c} n \geq 0 \qquad k \notin FV(P) \cup FV(b_1) \cup FV(b_2) \cup FV(e_v) \cup FV(\delta) \\ P \implies 0 \leq e_v\langle 1\rangle \\ P \implies e_v\langle 1\rangle \leq 0 \iff \neg b_1\langle 1\rangle \\ \forall k.\ 1 \leq k \leq n \implies 0 \leq m_k \\ \forall k.\ 1 \leq k \leq n \implies \vdash \{\delta\} \mid \{P \wedge b_1\langle 1\rangle \wedge b_2\langle 2\rangle \wedge e_v\langle 1\rangle = k \wedge 0 < e_v\langle 1\rangle \leq n\}\ C_1 \sim C_2\ \{P \wedge b_1\langle 1\rangle = b_2\langle 2\rangle \wedge e_v\langle 1\rangle < k\} \mid \{\delta\} \mid m_k \end{array}}{\begin{array}{c} \vdash \{\delta\} \mid \{P \wedge b_1\langle 1\rangle = b_2\langle 2\rangle \wedge e_v\langle 1\rangle \leq n\} \\ \texttt{while}\ b_1\ \texttt{do}\ C_1 \ \sim\ \texttt{while}\ b_2\ \texttt{do}\ C_2 \\ \{P \wedge \neg b_1\langle 1\rangle \wedge \neg b_2\langle 2\rangle\} \mid \{\delta\} \mid \sum_{k=1}^{n} m_k \end{array}}\ \textbf{awhile}$$

Graded awhile-evolving (synchronous, evolving δ_k). Same lockstep structure, but the binary property changes per iteration: $\delta_k \to \delta_{k-1}$. The variant decreases by exactly 1 each iteration (so the loop runs exactly n times). Used when the property genuinely changes during the loop, e.g., InplaceMap with a constant function $f(x) = 0$ where new equalities are created.

Difference from awhile:

- Variant decreases by 1 exactly (not just strictly), so loop runs exactly n iterations.
- Binary property is δ_k at iteration k, δ_{k-1} after; chain is $\delta_n \to \cdots \to \delta_0$.
- Side condition $k \notin FV(P) \cup FV(b_1) \cup FV(b_2) \cup FV(e_v)$ ensures k binds only the indexed family of premises/effects.

$$\frac{\begin{array}{c} n \geq 0 \qquad k \notin FV(P) \cup FV(b_1) \cup FV(b_2) \cup FV(e_v) \\ P \implies 0 \leq e_v\langle 1\rangle \\ P \implies e_v\langle 1\rangle \leq 0 \iff \neg b_1\langle 1\rangle \\ \forall k.\ 1 \leq k \leq n \implies \vdash \{\delta_k\} \mid \{P \wedge b_1\langle 1\rangle \wedge b_2\langle 2\rangle \wedge e_v\langle 1\rangle = k\}\ C_1 \sim C_2\ \{P \wedge b_1\langle 1\rangle = b_2\langle 2\rangle \wedge e_v\langle 1\rangle = k-1\} \mid \{\delta_{k-1}\} \mid m_k \end{array}}{\begin{array}{c} \vdash \{\delta_n\} \mid \{P \wedge b_1\langle 1\rangle = b_2\langle 2\rangle \wedge e_v\langle 1\rangle = n\} \\ \texttt{while}\ b_1\ \texttt{do}\ C_1 \ \sim \ \texttt{while}\ b_2\ \texttt{do}\ C_2 \\ \{P \wedge \neg b_1\langle 1\rangle \wedge \neg b_2\langle 2\rangle\} \mid \{\delta_0\} \mid \sum_{k=1}^{n} m_k \end{array}}\ \textbf{awhile-evolving}$$

8.4 Graded One-Sided Rules

A one-sided rule pairs an active construct on one run with `skip` on the other. Grades are natural-valued upper bounds on the *signed* cost difference $m_1 - m_2$ (left run minus right run). A *left*-active step has signed effect $+1$, so its grade is 1; a *right*-active step has signed effect -1, whose least natural upper bound is 0, so its grade is 0. Thus **assignL-skipR** carries grade 1 while its right-active mirror **assignR-skipL** carries grade 0. (A fully signed, $\mathbb{Z}$-graded variant would recover the literal ARel grade -1 for the right-active step; we keep grades in $\mathbb{N}$ here and revisit the signed variant in the remark of Section 8.9.)

$$\frac{x_1\langle 1\rangle \notin FV(\delta)}{\vdash \{\delta[e_1\langle 1\rangle/x_1\langle 1\rangle]\} \mid \{Q[e_1\langle 1\rangle/x_1\langle 1\rangle]\}\ x_1 := e_1 \sim \texttt{skip}\ \{Q\} \mid \{\delta\}\ \mid 1}\ \textbf{assignL-skipR}$$

$$\frac{x_2\langle 2\rangle \notin FV(\delta)}{\vdash \{\delta[e_2\langle 2\rangle/x_2\langle 2\rangle]\} \mid \{Q[e_2\langle 2\rangle/x_2\langle 2\rangle]\}\ \texttt{skip} \sim x_2 := e_2\ \{Q\} \mid \{\delta\}\ \mid 0}\ \textbf{assignR-skipL}$$

$$\frac{\vdash \{\delta\} \mid \{P \wedge b_1\langle 1\rangle\}\ C_1 \sim C_2\ \{Q\} \mid \{\delta'\}\ \mid m_1 \qquad \vdash \{\delta\} \mid \{P \wedge \neg b_1\langle 1\rangle\}\ C_1' \sim C_2\ \{Q\} \mid \{\delta'\}\ \mid m_2}{\vdash \{\delta\} \mid \{P\}\ \texttt{if}\ b_1\ \texttt{then}\ C_1\ \texttt{else}\ C_1' \sim C_2\ \{Q\} \mid \{\delta'\}\ \mid \max(m_1, m_2)}\ \textbf{conditiona}$$

$$\frac{x_1\langle 1\rangle \notin FV(\delta) \qquad \vdash \{\delta\} \mid \{P\}\ \texttt{skip} \sim C_2\ \{Q\} \mid \{\delta'\}\ \mid m}{\vdash \{\delta[e_1\langle 1\rangle/x_1\langle 1\rangle]\} \mid \{P[e_1\langle 1\rangle/x_1\langle 1\rangle]\}\ x_1 := e_1 \sim C_2\ \{Q\} \mid \{\delta'\}\ \mid 1 + m}\ \textbf{assignL}$$

$$\frac{x_1\langle 1\rangle \notin FV(\delta)}{\vdash \{\delta\} \mid \{Q[u_1[e_1\langle 1\rangle]/x_1\langle 1\rangle]\}\ x_1 := u_1[e_1] \sim \texttt{skip}\ \{Q\} \mid \{\delta\}\ \mid 1}\ \textbf{readL-skipR}$$

$$\frac{\begin{array}{c} j \text{ fresh} \\ Q[u_1[e_1\langle 1\rangle \mapsto e_1'\langle 1\rangle]/u_1] \implies e_1\langle 1\rangle = j \\ Q[u_1[e_1\langle 1\rangle \mapsto e_1'\langle 1\rangle]/u_1] \implies 0 \le j < len(u_1)\langle 1\rangle \end{array}}{\vdash \{\delta\} \mid \{Q[u_1[e_1\langle 1\rangle \mapsto e_1'\langle 1\rangle]/u_1]\}\ u_1[e_1] := e_1' \sim \texttt{skip}\ \{Q\} \mid \{\delta \ominus_{u_1\langle 1\rangle} \{j\}\}\ \mid 1}\ \textbf{updateL-skipR}$$

$$\frac{\begin{array}{c} n \ge 0 \qquad k \notin FV(P) \cup FV(b_1) \cup FV(e_v) \cup FV(\delta) \\ P \implies 0 \le e_v\langle 1\rangle \\ P \implies e_v\langle 1\rangle \le 0 \iff \neg b_1\langle 1\rangle \\ \forall k.\ 1 \le k \le n \implies 0 \le m_k \\ \forall k.\ 1 \le k \le n \implies \vdash \{\delta\} \mid \{P \wedge b_1\langle 1\rangle \wedge e_v\langle 1\rangle = k \wedge 0 < e_v\langle 1\rangle \le n\}\ C_1 \sim \texttt{skip}\ \{P \wedge e_v\langle 1\rangle < k\} \mid \{\delta\}\ \mid m_k \end{array}}{\vdash \{\delta\} \mid \{P \wedge e_v\langle 1\rangle \le n\}\ \texttt{while}\ b_1\ \texttt{do}\ C_1 \sim \texttt{skip}\ \{P \wedge \neg b_1\langle 1\rangle\} \mid \{\delta\}\ \mid \sum_{k=1}^{n} m_k}\ \textbf{wh}$$

$$\textbf{readL}\ \frac{\begin{array}{c} x_1\langle 1\rangle \notin FV(\delta) \\ \vdash \{\delta\} \mid \{P\}\ \texttt{skip} \sim C_2\ \{Q\} \mid \{\delta'\}\ \mid m \end{array}}{\vdash \{\delta\} \mid \{P[u_1[e_1\langle 1\rangle]/x_1\langle 1\rangle]\}\ x_1 := u_1[e_1] \sim C_2\ \{Q\} \mid \{\delta'\}\ \mid 1 + m}$$

$$\textbf{updateL}\ \frac{\begin{array}{c} j \text{ fresh} \\ P[u_1[e_1\langle 1\rangle \mapsto e_1'\langle 1\rangle]/u_1] \implies e_1\langle 1\rangle = j \\ P[u_1[e_1\langle 1\rangle \mapsto e_1'\langle 1\rangle]/u_1] \implies 0 \le j < len(u_1)\langle 1\rangle \\ \vdash \{\delta \ominus_{u_1\langle 1\rangle} \{j\}\} \mid \{P\}\ \texttt{skip} \sim C_2\ \{Q\} \mid \{\delta'\}\ \mid m \end{array}}{\vdash \{\delta\} \mid \{P[u_1[e_1\langle 1\rangle \mapsto e_1'\langle 1\rangle]/u_1]\}\ u_1[e_1] := e_1' \sim C_2\ \{Q\} \mid \{\delta'\}\ \mid 1 + m}$$

Right-sided variants (**assignR**, **readR-skipL**, **updateR-skipL**, **whileR-skipL**, **conditionalR**, **readR**, **updateR**) are structurally symmetric and omitted. Their grades, however, follow the signed bound rather than mirroring the left-active grades: a single right-active step (**readR-skipL**, **updateR-skipL**) has grade 0 like **assignR-skipL** (signed difference -1, least natural bound 0),

and a right-active continuation rule combines that grade 0 with the continuation grade. The branch-maximum rule **conditionalR** keeps grade $\max(m_1, m_2)$, since the branch maximum is direction-agnostic.

Graded re-sync-derived (β-recovery). The graded version of **re-sync-derived** accumulates the grades of the two one-sided phases. The recovery premise is semantic and contributes no grade:

$$\frac{\begin{array}{c}\vdash \{(u_1,u_2) \mapsto^{\Phi} \beta_0\} \mid \{P\}\ C_1 \sim \texttt{skip}\ \{P_{mid}\} \mid \{\delta_{mid}\}\ \mid m_1 \\ \vdash \{\delta_{mid}\} \mid \{P_{mid}\}\ \texttt{skip} \sim C_2\ \{P'\} \mid \{\delta_{out}\}\ \mid m_2 \\ \mathit{Recover}(P', u_1, u_2, \beta_0, \Phi_{rec}) \\ \mathit{PreserveLen}(C_1, C_2, u_1, u_2, \beta_0) \quad \Phi_{rec} \text{ well-formed}\end{array}}{\vdash \{(u_1,u_2) \mapsto^{\Phi} \beta_0\} \mid \{P\}\ C_1 \sim C_2\ \{P'\} \mid \{(u_1,u_2) \mapsto^{\Phi_{rec}} \beta_0\}\ \mid m_1 + m_2}\ \textbf{re-sync-derived}$$

Grade $= m_1 + m_2$ (additive, same as **sequence**). The second one-sided phase may end in an arbitrary residual δ_{out}; the recovered relation is established only by the explicit semantic recovery obligation, and the conclusion does not assert δ_{out} unless it is separately framed or recovered. The recovery premise contributes no grade; it is a logical side condition, not a program step. See the **re-sync-derived** case in the proof of Theorem 6.3 for the postcondition soundness proof; the grade obligation follows by adding the two one-sided grade bounds.

8.5 Graded Structural Rules

$$\frac{\begin{array}{c}\vdash \{\delta_1'\} \mid \{P'\}\ C_1 \sim C_2\ \{Q'\} \mid \{\delta_2'\}\ \mid m' \qquad \vDash P \Rightarrow P' \qquad \vDash Q' \Rightarrow Q \\ P \vdash \delta_1' \le \delta_1 \qquad Q \vdash \delta_2 \le \delta_2' \qquad m' \le m \qquad m \in \mathcal{G}\end{array}}{\vdash \{\delta_1\} \mid \{P\}\ C_1 \sim C_2\ \{Q\} \mid \{\delta_2\}\ \mid m}\ \textbf{conseq}$$

$$\frac{\vdash \{\delta\} \mid \{P\}\ C_1 \sim C_2\ \{R\} \mid \{\delta_1\}\ \mid m_1 \qquad \vdash \{\delta_1\} \mid \{R\}\ C_1' \sim C_2'\ \{Q\} \mid \{\delta'\}\ \mid m_2}{\vdash \{\delta\} \mid \{P\}\ C_1; C_1' \sim C_2; C_2'\ \{Q\} \mid \{\delta'\}\ \mid m_1 + m_2}\ \textbf{sequence}$$

$$\frac{m \in \mathcal{G}}{\vdash \{\delta\} \mid \{\bot\}\ C_1 \sim C_2\ \{P\} \mid \{\delta\}\ \mid m}\ \textbf{false}$$

$$\frac{\begin{array}{c}\vdash \{\delta_1\} \mid \{P\}\ C_1 \sim C_2\ \{Q\} \mid \{\delta_2\}\ \mid m \qquad FV(\delta) \cap (\mathit{mod}(C_1)\langle 1\rangle \cup \mathit{mod}(C_2)\langle 2\rangle) = \emptyset \\ \mathit{arrFV}(\delta) \cap (\mathit{arrfoot}(C_1)\langle 1\rangle \cup \mathit{arrfoot}(C_2)\langle 2\rangle) = \emptyset \\ \mathit{arrfoot}(C_i)\langle i\rangle \subseteq \mathit{own}_i(\delta_1) \quad (i \in \{1,2\}) \\ \mathit{local}_{\delta,\delta_1}(P) \text{ and } \mathit{local}_{\delta,\delta_2}(Q) \quad \text{(Definitions 6.4, 6.5)}\end{array}}{\vdash \{\delta \star \delta_1\} \mid \{P\}\ C_1 \sim C_2\ \{Q\} \mid \{\delta \star \delta_2\}\ \mid m}\ \textbf{heap-frame}$$

$$\frac{\begin{array}{c}\vdash \{\delta_1\} \mid \{P\}\ C_1 \sim C_2\ \{Q\} \mid \{\delta_2\}\ \mid m \qquad FV(R) \cap (\mathit{mod}(C_1)\langle 1\rangle \cup \mathit{mod}(C_2)\langle 2\rangle) = \emptyset \\ \mathit{arrFV}(R) \cap (\mathit{arrvars}(C_1)\langle 1\rangle \cup \mathit{arrvars}(C_2)\langle 2\rangle) = \emptyset\end{array}}{\vdash \{\delta_1\} \mid \{P \wedge R\}\ C_1 \sim C_2\ \{Q \wedge R\} \mid \{\delta_2\}\ \mid m}\ \textbf{frame}$$

8.6 Grade Summary

Rule	Grade	Side	Rationale
skip, assign	0	Two-sided	Same operation both runs
read1, read2	0	Two-sided	Both runs read
update1, update2	0	Two-sided	Both runs update
alloc1, alloc2	0	Two-sided	Both runs allocate
conditional	m	Two-sided	From body (same branch)
awhile	$\sum m_k$	Two-sided	Per-iteration sum
awhile-evolving	$\sum m_k$	Two-sided	Evolving δ_k, per-iteration sum
assignL-skipR	1	One-sided	Left acts, right idles
assignR-skipL	0	One-sided	Right acts (signed -1); least natural bound 0
assignL	$1 + m$	One-sided	Left assigns, then continuation
readL-skipR	1	One-sided	Left reads, right idles
updateL-skipR	1	One-sided	Left updates, right idles
conditionalL	$\max(m_1, m_2)$	One-sided	From branches
whileL-skipR	$\sum m_k$	One-sided	Left loops, right idles
sequence	$m_1 + m_2$	Composition	Additive
conseq	$m' \leq m$	Structural	Weakening
false	m	Structural	Any $\mathcal{G}$ grade from $\bot$
heap-frame	m	Structural	Grade preserved under $\star$
frame	m	Structural	Grade preserved under $\wedge$

8.7 Grade Well-Formedness

Theorem 8.2 presupposes that the grade m denotes a natural number determined by the interpretation. The following lemma discharges that presupposition. We impose, as an explicit *side condition* on the structural rules, that every grade externally supplied to a structural rule, namely the weakened target m of **conseq** (with $m' \leq m$) and the arbitrary grade of **false**, is a term of the derived sublanguage $\mathcal{G}$ defined in the lemma below ($m \in \mathcal{G}$); the composition rules (sequence, conditional, awhile) build their grades from $\mathcal{G}$ automatically. This $m \in \mathcal{G}$ side condition is what keeps every derivable grade inside $\mathcal{G}$, and hence (Lemma 8.1) natural-valued; it is part of the well-formedness hypothesis of Theorem 8.2.

Lemma 8.1 (Grade well-formedness)**.** *Every grade assigned by a Level 2 rule belongs to the* derived grade sublanguage

$$\mathcal{G} ::= 0 \mid 1 \mid \chi_\varphi \mid \mathcal{G} + \mathcal{G} \mid \max(\mathcal{G}_1, \mathcal{G}_2) \mid \textstyle\sum_{k=a}^{b} \mathcal{G}_k,$$

where $\sum_{k=a}^{b}$ *is a finite sum over a bounded integer range* $a \leq k \leq b$ *(so both* $\sum_{k=1}^{n}$ *and* $\sum_{k=0}^{n-1}$ *are instances), and* $\chi_{J\in\beta}, \chi_{J\notin\beta} \in \{0, 1\}$ *are* index-set membership indicators *for an interpretation-level index set* β *(denotation* $[\![\beta]\!]_b(I)$

fixed by I). The indicator is pure and heap-independent*, reading no program variable and no array, so it is determined by I alone. Crucially, a $\mathcal{G}$ grade is built only from the program's* structural *quantities: the array size n, interpretation-level index sets β, and bound index variables, and* not *from the free prophecy/threshold parameters (such as d_0) that parameterize a property bound. Concretely, no summation bound a, b, no index set β, and no indicator predicate occurring in a $\mathcal{G}$ grade may mention d_0. This closes every route to the threshold: $\sum_{k=d_0}^{n-1} 1$ (bound mentions d_0), $\sum_{k=0}^{n-1} \chi_{d_0 \le k}$ (predicate mentions d_0), and $\sum_{k=0}^{n-1} \chi_{k \notin [0,d_0)}$ (β mentions d_0) all evaluate to $n - d_0$ but are* excluded*; so $n - d_0$ with free d_0 is genuinely outside $\mathcal{G}$. The same discipline governs* weight parameters*: a bounded-sum limit may be a nonnegative, d_0-free, read-only logical/index parameter (e.g. the increment weight 2, an L^∞ bound ϵ, a threshold h_2, a pointwise bound M, or a fixed affine coefficient with the same read-only treatment as the array size n in the well-formedness note of Definition 8.1), and we write $w \cdot m$ as definitional sugar for the bounded sum $\sum_{j=1}^{w} m$; weights are never mutable program variables and never mention d_0. The grammar uses only natural-number-preserving constructs (no subtraction, negation, or product); a finite index-set cardinality is then expressible within $\mathcal{G}$, e.g. $|\beta \cap [0, n)| = \sum_{k=0}^{n-1} \chi_{k \in \beta}$. Consequently, for every derivable graded judgment $\vdash \{\delta\} \mid \{P\}\ C_1 \sim C_2\ \{Q\} \mid \{\delta'\} \mid m$, the grade m is a term of $\mathcal{G}$ whose free variables are logical or index variables of the judgment, and for every interpretation I over those variables, $[\![m]\!]_i(I) \in \mathbb{N}$ is determined by I alone.*

Proof. By induction on the derivation, showing that each rule's grade is a term of $\mathcal{G}$; $\mathbb{N}$-valuedness then follows because every constructor of $\mathcal{G}$ is total on $\mathbb{N}$. *Base cases.* The primitive grades 0 (two-sided rules and right-active one-sided rules) and 1 (left-active one-sided rules) are the constants of $\mathcal{G}$, in $\mathbb{N}$, with no free variables. An indicator atom χ_φ is valued in $\{0, 1\} \subseteq \mathbb{N}$, determined by I through the decidable predicate φ whose free variables are logical/index variables; hence $\chi_\varphi \in \mathcal{G}$ is natural-valued. *Sequence* and *re-sync-derived* (both grade $m_1 + m_2$, additive): by the induction hypothesis $m_1, m_2 \in \mathcal{G}$, so $m_1 + m_2 \in \mathcal{G}$. *Two-sided conditional* inherits a single common grade m from its two equal-graded branches, in $\mathcal{G}$ by the induction hypothesis; the one-sided *conditionalL* takes $\max(m_1, m_2)$, also in $\mathcal{G}$. In each case a sum or maximum of two naturals is a natural, and the free-variable set is the union of the premises'. *The substantive case is* ***awhile*** (grade $\sum_{k=1}^{n} m_k$): by the induction hypothesis each per-iteration grade $m_k \in \mathcal{G}$ is natural-valued in the extended interpretation $I[k \mapsto \cdot]$; the summation variable k and the loop count n are index/logical variables, as are weight parameters used as summation limits (ϵ, h_2, M; statement side condition above), and a finite sum (a terminating run has $n \in \mathbb{N}$) of naturals is a natural. Hence

$$[\![\textstyle\sum_{k=1}^{n} m_k]\!]_i(I) = \sum_{k=1}^{n} [\![m_k]\!]_i(I[k \mapsto k]) \in \mathbb{N},$$

determined by I. The one-sided loop rule **whileL-skipR** (grade $\sum_k m_k$) and **awhile-evolving** are identical. *conseq* (grade m with side condition $m' \le m$): the weakening target m is itself required to be a term of $\mathcal{G}$, and the side

condition $m' \leq m$ is a comparison in the natural-number order (presupposing $m \in \mathbb{N}$); the premise grade m' is in $\mathcal{G}$ by the induction hypothesis. *heap-frame*, *frame*: the grade is inherited unchanged from the sub-derivation, in $\mathcal{G}$ by the induction hypothesis. *false*: the grade is an arbitrary term of $\mathcal{G}$, vacuously sound. Each rule preserves membership in $\mathcal{G}$ and $\mathbb{N}$-totality, so $[\![m]\!]_i(I) \in \mathbb{N}$ for every derivable judgment.

Finally, closed forms such as $n-|\beta|$ that appear in the examples (e.g. CountPositive) are *evaluations* of a bounded $\sum$ term over indicator atoms, namely the count of divergent positions over the index range $[0,n)$, $\sum_{k=0}^{n-1} \chi_{k\notin\beta} = n - |\beta \cap [0,n)|$ (the range $[0,n)$ has n elements, each $\chi_{k\notin\beta} \in \{0,1\}$), and not primitive subtraction terms of the grade grammar; the subtraction lives in the arithmetic of the denotation, not in the grade syntax, so $\mathbb{N}$-valuedness is preserved. A bound of the form $n - d_0$ with d_0 a *free* threshold variable is not itself a $\mathcal{G}$ term; where it appears (InplaceMap, DualCount, the CHC goal) it is a *numeric upper bound* on the exact grade $n - |\beta \cap [0,n)| = \sum_{k=0}^{n-1} \chi_{k\notin\beta}$, valid when $d_0 \leq |\beta \cap [0,n)|$. The property $c\langle 1\rangle - c\langle 2\rangle \leq n - d_0$ then follows from the grade's semantic guarantee $c\langle 1\rangle - c\langle 2\rangle \leq n - |\beta \cap [0,n)|$ together with the arithmetic $n - |\beta \cap [0,n)| \leq n - d_0$; $n - d_0$ is never supplied as a grade. □

8.8 Level 2 Soundness

Theorem 8.2 (Soundness, Level 2 execution-cost effect instance)**.** *For every well-formed derivation of* $\vdash \{\delta\} \mid \{P\}\ C_1 \sim C_2\ \{Q\} \mid \{\delta'\} \mid m$ *in the default execution-cost instance of the Level 2 graded relational logic (Section 8),*

$$\vDash^m \{\delta\} \mid \{P\}\ C_1 \sim C_2\ \{Q\} \mid \{\delta'\}.$$

That is, for every well-formed interpretation I covering the free logical variables of the judgment (with m a grade of the derived sublanguage $\mathcal{G}$ of Lemma 8.1, so that $[\![m]\!]_i(I) \in \mathbb{N}$ is defined under I) and every pair of finite terminating cost-instrumented executions $(\sigma_1, \sigma_1', m_1) \in [\![C_1]\!]$ and $(\sigma_2, \sigma_2', m_2) \in [\![C_2]\!]$ from a precondition state, the postcondition holds and $m_1 - m_2 \leq [\![m]\!]_i(I)$ (signed bound, one-sided, in the spirit of ARel's $cost(t_1) - cost(t_2) \leq D$).

Scope of Theorem 8.2 (execution-cost effect instance). *Theorem 8.2 is restricted to the default execution-cost effect instance.* Specifically: m_1, m_2 are the total execution costs accumulated by the two runs C_1, C_2 from the per-construct cost model of Section 3.2.3 ($c_{\texttt{skip}} = 0$, $c_{assign} = c_{read} = c_{update} = c_{alloc} = 1$, $c_{if} = c_{while} = 0$); and the cost model satisfies the *rule-compatibility* conditions of Section 8.9 (grade-0 paired rules; grade-1 left-active and grade-0 right-active one-sided rules, the latter bounding the signed difference $-1 \leq 0$). Both restrictions are satisfied by this cost model.

The remaining seven Level 2 encodings (**ClampPositive**, **NestedBranch**, **CompressPositive**, **arraysum_mixed**, **conditional_cost**, **threshold_filter_bounded**, **ArrayDoubleOpNI_robust_hFT**) are Level 1 quantitative-invariant cases. They share the relational invariant skeleton with the Theorem 8.2 examples

(lockstep loop, PickK/Wit cell observation, 0/1 equal/unequal observation driving an additive scalar recurrence), but accumulate over an effect other than per-construct execution cost: ClampPositive upper-bounds the output-Hamming distance (the $M = 1$ instance), NestedBranch and CompressPositive accumulate source-variable counters, and the four value-weighted examples accumulate weighted source quantities. Their soundness is established by relational invariants over the respective scalar counter, *not* by Theorem 8.2. In particular, the execution-cost grade does not bound Hamming: a paired same-construct *write* (both runs update the same cell with different values) increases Hamming by one while contributing zero to the cost grade. A generic graded-effect soundness theorem parametric in the effect functional is proved in Section 9; its output-disagreement instance (Corollary 9.4) covers ClampPositive and its affine-accumulator instance (Corollary 9.5) covers the six accumulator cases.

Proof. By induction on the Level 2 derivation of $\vdash \{\delta\} \mid \{P\}\ C_1 \sim C_2\ \{Q\} \mid \{\delta'\} \mid m$. Each rule discharges two obligations.

(i) Postcondition and binary-property preservation. This component is identical to the Level 1 obligation: every Level 2 rule reuses the relational assertions, binary properties, and state/effect transformation of its Level 1 counterpart, and the grade annotation does not influence the produced post-state. It is therefore discharged by the corresponding case of the Level 1 soundness proof (Theorem 6.3).

(ii) Signed cost bound. The new obligation, $m_1 - m_2 \leq [\![m]\!]_i(I)$ of Definition 8.1, is exactly the conclusion of Lemma 8.3 (per-rule cost-difference preservation), whose hypotheses are the induction hypotheses of the present induction (each premise judgment is graded-valid at its premise grade: postcondition/effect preservation together with its signed cost bound). Lemma 8.1 supplies $[\![m]\!]_i(I) \in \mathbb{N}$, so the bound is well-posed, and the executions are the finite terminating ones quantified in the theorem statement (the **awhile** case uses finiteness to evaluate the per-iteration sum).

Composing (i) and (ii) at each rule over the derivation yields $\vDash^m \{\delta\} \mid \{P\}\ C_1 \sim C_2\ \{Q\} \mid \{\delta'\}$. □

8.9 Level 2 Per-Rule Cost-Difference Bounds

This subsection discharges the cost-difference obligation $m_1 - m_2 \leq [\![m]\!]_i(I)$ of Theorem 8.2 for the rules of the Level 2 graded relational logic. The bound is signed (left run minus right run), matching ARel/RelCost. For each rule, the postcondition obligation reduces to the common postcondition/effect argument already proved for Level 1 (Theorem 6.3); we focus here on the new cost-difference component. Throughout, m_i denotes the operational cost of the run of C_i, given by $(\sigma_i, \sigma_i', m_i) \in [\![C_i]\!]$ in the cost-instrumented operational semantics (Section 3.2.3, with default per-construct weights $c_{\texttt{skip}} = 0$, $c_{assign} = c_{read} = c_{update} = c_{alloc} = 1$, $c_{if} = c_{while} = 0$).

Lemma 8.3 (Per-rule cost-difference preservation)**.** *Under the default execution-cost model, every Level 2 rule preserves the signed cost bound: if each premise*

judgment is graded-valid *at its premise grade m_ℓ, i.e. on all finite terminating runs it preserves the relational postcondition and binary property (Theorem 6.3)* and *satisfies the signed cost bound $m_1^\ell - m_2^\ell \leq [\![m_\ell]\!]_i(I)$, with all grades in $\mathcal{G}$ (Lemma 8.1), then every pair of finite terminating executions of the conclusion's two commands from a precondition state satisfies $m_1 - m_2 \leq [\![m]\!]_i(I)$, where m is the rule's conclusion grade. (We invoke the full graded-validity of the premises, not only their cost bounds, since the sequence and continuation cases use the premise postcondition/effect to apply the next premise's bound.)*

Proof. By cases on the Level 2 rule; we discharge the cost-difference component (the postcondition/effect component of each case is the corresponding Level 1 case, Theorem 6.3).

Two-sided rules (grade 0). The two-sided rules **skip**, **assign**, **read1**, **read2**, **update1**, **update2**, **alloc1**, **alloc2** all have grade 0. In each case both runs perform the same kind of construct, and the per-construct cost model assigns the same cost to that construct on both sides. Hence $m_1 = m_2$ for the single step contributed by the rule, so $m_1 - m_2 = 0 \leq 0 = [\![0]\!]_i(I)$. □

One-sided rules. A one-sided rule pairs an active construct on one run with `skip` on the other; its grade is the least natural upper bound on the signed cost difference $m_1 - m_2$.

- **assignL-skipR** (grade 1): $C_1 = (x := e)$ has cost $c_{assign} = 1$; $C_2 = \texttt{skip}$ has cost $c_{\texttt{skip}} = 0$. Hence $m_1 - m_2 = 1 - 0 = 1 \leq 1 = [\![1]\!]_i(I)$. Tight. □
- **readL-skipR**, **updateL-skipR** (grade 1): $c_{read} = c_{update} = 1$, same analysis as **assignL-skipR**: $m_1 - m_2 = 1 \leq 1$. □
- **assignR-skipL** (grade 0): $m_1 - m_2 = 0 - 1 = -1 \leq 0 = [\![0]\!]_i(I)$. The exact signed effect is -1; its least natural upper bound is 0. □
- Right-active variants **readR-skipL**, **updateR-skipL** (grade 0): same analysis as **assignR-skipL**. □

Remark (grade of right-active rules). Grades are natural numbers bounding the signed difference $m_1 - m_2$ from above. A right-active unit step has exact signed effect -1, so its least natural upper bound is 0, not 1: charging 1 would be sound but loose. A fully signed, $\mathbb{Z}$-valued grade would assign the literal ARel grade -1; we keep grades in $\mathbb{N}$, accepting one unit of slack on right-active steps. The signed-grade variant is characterized in the cost-model remark below: the right-active grade is $-lo_2$, equal to -1 only under strict execution cost. Primitive unit left-active grades remain tight at 1.

Sequence (grade $m_1 + m_2$). For $C_1; C_1' \sim C_2; C_2'$ with grade $m_1 + m_2$, the cost-instrumented semantics decomposes each run's cost as $m_1 = m_1^{(a)} + m_1^{(b)}$ and $m_2 = m_2^{(a)} + m_2^{(b)}$. Applying the first premise's graded validity to the

first pair of sub-executions gives both the intermediate relational assertion R at $(\widehat{\sigma}_1, \widehat{\sigma}_2)$, which is exactly the precondition of the second premise, and $m_1^{(a)} - m_2^{(a)} \leq [\![m_1]\!]_i(I)$; the second premise, applied from that intermediate state, then gives $m_1^{(b)} - m_2^{(b)} \leq [\![m_2]\!]_i(I)$. Adding the two inequalities:

$$m_1 - m_2 = (m_1^{(a)} - m_2^{(a)}) + (m_1^{(b)} - m_2^{(b)}) \leq [\![m_1 + m_2]\!]_i(I). \quad \square$$

(Signed bound is preserved under addition; no triangle inequality needed.)

Conditional with branch agreement (common grade m). The two-sided **conditional** rule requires $P \implies b_1\langle 1\rangle = b_2\langle 2\rangle$ as a premise, and the rule states a *common* grade m shared by both branches and the conclusion. So under the precondition both runs evaluate the guard to the same boolean and take the same branch, and the chosen branch's premise gives $m_1 - m_2 \leq [\![m]\!]_i(I)$. Under default weights $c_{if} = 0$, the conditional adds no cost overhead. The common-grade form means the prover must weaken individual branch grades to a common m via **conseq** before applying **conditional**; the upper bound $m = \max(m_1, m_2)$ is the natural choice. $\square$

One-sided conditional conditionalL (grade $\max(m_1, m_2)$). The one-sided **conditionalL** rule handles the case where only the left run branches. The two branch premises may have grades m_1 and m_2, and the conclusion uses $\max(m_1, m_2)$. The cost-difference bound follows by case analysis on which branch the left run takes, applying the IH to the corresponding sub-derivation and weakening that branch's bound to the maximum. The bound accounts for the cost difference contributed by the entire executed branch (including any internal one-sided steps within it). $\square$

Synchronous awhile, fixed δ (grade $\sum_{k=1}^{n} m_k$). The relational **awhile** rule (Section 5) requires guard agreement and runs both loops lockstep. The variant bound establishes that the loops execute *at most* n body iterations and that the variant $e_v\langle 1\rangle$ strictly decreases (in $\mathbb{N}$) on each iteration. Because Theorem 8.2 quantifies only over *finite terminating* executions, each run performs finitely many iterations $n_* \leq n$. With $c_{while} = 0$, the cost-instrumented semantics then decomposes each run's total cost as the finite sum $m_i = \sum_{j=1}^{n_*} m_i^{(j)}$, where $m_i^{(j)}$ is the cost of the j-th iteration. Let $k_j = [\![e_v\langle 1\rangle]\!](\sigma_1^{(j-1)})$ be the variant value at the start of iteration j; the per-iteration premise instantiated by the substitution $k \mapsto k_j$ (i.e., evaluated under the extended interpretation $I[k \mapsto k_j]$) gives $m_1^{(j)} - m_2^{(j)} \leq [\![m_k]\!]_i(I[k \mapsto k_j])$. The visited values $\{k_1, k_2, \ldots, k_{n_*}\} \subseteq \{1, 2, \ldots, n\}$ form a subset of the index range (the variant strictly decreases through $\mathbb{N}$, so the values are distinct and in range). Summing over visited iterations and weakening with nonnegative grades $m_k \geq 0$ for k outside the visited subset:

$$m_1 - m_2 = \sum_{j=1}^{n_*} (m_1^{(j)} - m_2^{(j)}) \leq \sum_{j=1}^{n_*} [\![m_k]\!]_i(I[k \mapsto k_j]) \leq \sum_{k=1}^{n} [\![m_k]\!]_i(I[k \mapsto k]) = [\![\sum_k m_k]\!]_i(I). \quad \square$$

(Signed bound preserved under summation. The visited-subset inclusion uses strict monotonicity of the variant; weakening to the full $[1, n]$ range uses $m_k \geq 0$.)

awhile-evolving (grade $\sum_k m_k$ with evolving δ_k, $e_v = n$, $k \to k-1$). The evolving variant fixes the iteration count exactly (decreasing variant from n to 0, so exactly n body iterations). Same cost-decomposition argument as synchronous **awhile**, but without the weakening step: the trace sum equals $\sum_{k=1}^{n}$ exactly. The evolution of δ_k affects the postcondition obligation (handled by the common postcondition/effect argument in Section 6) but not the cost-difference component, which proceeds rule-by-rule on the per-iteration premises. □

Consequence (conseq). Suppose the premise gives $m_1 - m_2 \leq [\![m']\!]_i(I)$ and the conclusion has grade m with side condition $m' \leq m$. Then $m_1 - m_2 \leq [\![m']\!]_i(I) \leq [\![m]\!]_i(I)$. □

Omitted cases (reduced to existing arguments). The following rules are omitted from the explicit sketches because they reduce to the patterns above:

- **heap-frame** and **frame**: these structural rules attach a framing assertion that is independent of the modified variables and arrays. They do not affect cost since both runs still execute the same body; the cost-difference bound is inherited unchanged from the framed sub-derivation.
- **false** (precondition $\bot$): the obligation is vacuously satisfied since no precondition state exists.
- **whileL-skipR** and **whileR-skipL**: these one-sided loop rules charge the body grade m_k at each visited variant value k. For **whileL-skipR** (left loops, right idle) each iteration runs the body *derivation* on the left against `skip` on the right, contributing its premise grade m_k (for the canonical unit-cost left-active body this is $+1$, bounded by grade 1); for **whileR-skipL** the per-iteration contribution is the mirror body grade (for a unit-cost right-active body, signed -1, bounded by grade 0). The cost-difference reasoning is otherwise the same finite-sum argument as the synchronous **awhile** above, applied to the one-sided body derivation.
- Continuation rules **assignL**, **readL**, **updateL** (and right-symmetric variants **assignR**, **readR**, **updateR**): these chain a one-sided action with a relational continuation. The cost-difference reasoning combines the one-sided cost contribution (signed $+1 \leq 1$ for a left-active step, signed $-1 \leq 0$ for a right-active step) with the IH on the continuation premise, exactly like the sequence rule above.
- **re-sync-derived**: this derived rule chains $C_1 \sim \texttt{skip}$ and $\texttt{skip} \sim C_2$ via sequential composition. The cost-difference reasoning follows the sequence

rule applied to the two one-sided sub-derivations (each contributes its own grade); the semantic recovery premise affects the postcondition obligation only, not the cost-difference component.

□

Remark on cost models (rule-compatibility condition). The soundness sketches above require the cost-instrumented semantics to be *rule-compatible* with the grade arithmetic. Specifically, the cost model must satisfy:

- **Grade-0 paired rules:** every two-sided construct (**skip**, **assign**, etc.) has equal cost on both runs when both runs perform that construct. Automatic when per-construct cost depends only on the construct's syntactic shape, not on run-specific runtime values.
- **One-sided rules:** for a *left*-active one-sided rule (**assignL-skipR** and analogues), the signed cost difference between the active construct on the left and the idle `skip` on the right must be bounded by the rule's grade 1:

$$\mathit{cost}(\mathit{active})\langle 1\rangle - \mathit{cost}(\texttt{skip})\langle 2\rangle \leq 1.$$

  Under default weights ($c_{assign} = c_{read} = c_{update} = c_{alloc} = 1$, $c_{\texttt{skip}} = 0$), this is exactly $1 - 0 = 1$, so the grade-1 bound is *tight*. For a *right*-active one-sided rule (**assignR-skipL** and analogues), the signed difference is $\mathit{cost}(\texttt{skip})\langle 1\rangle - \mathit{cost}(\mathit{active})\langle 2\rangle = 0 - 1 = -1$, bounded by the rule's grade 0 ($-1 \leq 0$, the least natural bound, as discussed in the one-sided rules remark above).

The default per-construct cost model in Section 3.2.3 satisfies these conditions, and the resulting grades are tight on V1 execution-cost examples (e.g., $m = n - |\beta|$ for CountPositive). Other cost models are permissible as long as the rule-compatibility conditions hold.

Neutral vs. signed one-sided grades ($U_1 - L_2$; branch-counting vs. execost). The one-sided grades instantiate the relational-cost identity $m = U_1 - L_2$: a left-active step contributes $+up_1$ (an *upper* bound on the active run's per-step cost) and a right-active step $-lo_2$ (a *lower* bound on the active run's per-step cost). Two cost models bracket the cases. Under *strict execution cost* every active step performs an unconditional cost-bearing construct, so $up_1 = lo_2 = 1$ and the tight right-active grade is the signed -1 (rounded up to 0 in $\mathbb{N}$, as above). Under *branch-counting cost* (cost increments only when a guard holds) the active step may cost 0, so $lo_2 = 0$ and the tight right-active grade is already 0 in both $\mathbb{N}$ and $\mathbb{Z}$; here a hard -1 would be *unsound*, since the actual signed difference can be 0 and $0 \leq -1$ fails. The natural-valued grade 0 is therefore the neutral, cost-model-agnostic choice, sound for both models because costs are nonnegative ($m_1 - m_2 = 0 - w_2 \leq 0$ for run-2 step cost $w_2 \geq 0$); the signed -1 is recoverable only in the execost instance ($lo_2 = 1$). This is the logic-side image of the scheduler rule

for an asynchronous `SchFT` step, $c' = c - lo_2$, settled on the experimental side (`working-folder/FT_COST_CONCLUSIONS_2026-06-01.html`): the *primitive* one-sided rule keeps the neutral grade 0, while the cost-model-dependent $-lo_2$ belongs to the scheduler/signed accounting, not the primitive grade table. Left-active grades scale with up_1: unit steps give $+1$, and multi-unit accumulations (e.g. the $+2$ and $+4$ of DualCount and QuadCount) give $+2$, $+4$. The neutral choice is *not* loose for the bounds of this paper: every published bound provable with the signed grade is provable with the neutral one (e.g. Stride-2v3 proves $2c + 2d_0 \leq n + 1$ under either); the signed -1 is needed only for tight "right-run-strictly-cheaper" bounds ($m_1 - m_2 \leq -1$) outside our target class.

8.10 Level 2 Derivation: Relational Cost of CountPositive

We now derive the *quantitative* bound $c\langle 1\rangle - c\langle 2\rangle \leq n - |\beta|$ using the graded logic. Compare with the Level 1 *qualitative* derivation (Section 7).

Setup. $\Phi(x, y) = (x = y)$, $\beta \subseteq [0, n)$ (positions where $A_1[i] = A_2[i]$), $\delta = (A_1, A_2) \mapsto^{=} \beta$.

Invariant.

$$I \equiv i\langle 1\rangle = i\langle 2\rangle \wedge 0 \leq i \leq n \wedge c\langle 1\rangle - c\langle 2\rangle \leq i - |\beta \cap [0, i)|$$

Loop body at $i \in \beta$ (grade 0). **read1** gives $\Phi(x_1, x_2)$, i.e., $x_1 = x_2$. Then **conditional** (same guard since $x_1 = x_2$): both take the same branch. All two-sided, body grade = 0.

Invariant: $c' = c$, $|\beta \cap [0, i')| = |\beta \cap [0, i)| + 1$. So $c' \leq i' - |\beta \cap [0, i')| = (i - |\beta \cap [0, i)|)$. ✓

Loop body at $i \notin \beta$ (grade 1). **read2** gives no Φ info ($m = 0$). Guards may differ, so use **conditionalL** on $x_1 > 0$:

- $x_1 > 0, x_2 > 0$: **assign∼assign** ($c_1{+}{+}$, $c_2{+}{+}$). Grade 0.
- $x_1 > 0, x_2 \leq 0$: **assignL-skipR** ($c_1{+}{+}$, skip). Grade 1. ← *divergence*
- $x_1 \leq 0, x_2 > 0$: **assignR-skipL** (skip, $c_2{+}{+}$). Grade 0 (signed $-1 \leq 0$).
- $x_1 \leq 0, x_2 \leq 0$: **skip∼skip**. Grade 0.

The left guard drives **conditionalL**: its $x_1 > 0$ branch has grade $\max(0, 1) = 1$ (binding sub-case **assignL-skipR**) and its $x_1 \leq 0$ branch has grade $\max(0, 0) = 0$, so **conditionalL** grade $= \max(1, 0) = 1$ and the body grade is 0 (**read2**) $+ 1$ (**conditionalL**) $= 1$. The right-diverging sub-case **assignR-skipL** (signed -1) is dominated by the left-diverging $+1$, so the upper bound on $c\langle 1\rangle - c\langle 2\rangle$ is 1 per divergent position, the left-heavy character of the bound.

Invariant: $c' \leq c + 1$, $|\beta \cap [0, i')| = |\beta \cap [0, i)|$. So $c' \leq (i - |\beta \cap [0, i)|) + 1 = i' - |\beta \cap [0, i')|$. ✓

Total grade.

$$\sum_{k=1}^{n} m_k = 0 \cdot |\beta \cap [0,n)| + 1 \cdot (n - |\beta \cap [0,n)|) = n - |\beta \cap [0,n)|$$

At termination: $c\langle 1\rangle - c\langle 2\rangle \leq n - |\beta \cap [0,n)|$. ✓

Level 1 vs. Level 2 contrast.

	Level 1 (Section 7)	**Level 2 (this section)**
$\Phi(x,y)$	$x \leq y$	$x = y$
β	$[0,n)$ (all positions)	$\subsetneq [0,n)$ (some differ)
Property	$c\langle 1\rangle \leq c\langle 2\rangle$ (qualitative)	$c\langle 1\rangle - c\langle 2\rangle \leq n - \lvert\beta\rvert$ (quantitative)
Grade	Not needed	$n - \lvert\beta\rvert$
Key rule	**conditionalL** (monotonicity)	**assignL-skipR** (grade 1)
CHC analog	Fixed k, no PickK	PickK + prophecy d_0

8.11 Level 2 Derivation: InplaceMap (Mutable Array)

This derivation exercises the graded **update1** and **update2** rules on a mutable-array example, the running example from ARel [2].

```
InplaceMap(A, n):
    i := 0
    c := 0                                              // ghost: counts update-branch
    while (i < n):
        x := A[i]
        if x > 0 then { A[i] := x + 1; c := c+1 } else skip
// asymmetric: only positives are updated
        i++
```

Two runs on different arrays A_1, A_2. *The variable c is ghost (instrumentation): it does not affect control flow or the array result, and is incremented exactly when the update branch fires; it materializes the per-construct execution-cost difference so the relational bound can be stated as* $c\langle 1\rangle - c\langle 2\rangle \leq n - d_0$. The ghost update $c := c+1$ is itself *not* counted in the operational cost m of Theorem 8.2: m accumulates only the real program constructs, and the cost-relevant step at a position is the array *update* in the if-branch (weight $c_{update} = 1$) versus the `skip` in the else-branch (weight 0), giving the per-position operational difference of $+1$ that the grade 1 bounds; the ghost c merely records this difference and does not add to m. Under the per-construct cost model with $c_{\texttt{skip}} = 0$ and $c_{update} = 1$, the per-iteration *update/branch cost contribution* is 1 when the guard $x > 0$ holds (the if-branch performs an array update) and 0 when it fails (the else-branch is `skip`). The fixed read cost (one read of $A[i]$ per iteration) is the same in both runs and cancels in the cost difference.

Note on source modification. The original ARel-style InplaceMap source was $A[i] := f(x)$ with $f(x) = x > 0?x + 1 : 1$, where both branches write to $A[i]$. Under the per-construct cost model both runs always do n updates, so operational cost diff is 0 and the encoding's relational counter c would be a ghost. Modifying the else-branch to `skip` makes the operational cost diff non-trivial and aligns the encoding's c with the signed cost-diff bound.

Property (signed relational cost). If A_1 and A_2 agree at d_0 positions in $[0, n)$, then $c\langle 1\rangle - c\langle 2\rangle \leq n - d_0$ (signed). Here c counts the number of positions where the if-branch fires (i.e., the number of positives in the run's array).

$\Phi(x, y) = (x = y)$, $\beta_0 \subseteq [0, n)$ with $|\beta_0| \geq d_0$:

$$\vdash \{(A_1, A_2) \mapsto^{=} \beta_0\} \mid \{n > 0\}\ P_1 \sim P_2\ \{c\langle 1\rangle - c\langle 2\rangle \leq n - d_0\} \mid \{(A_1, A_2) \mapsto^{=} \beta_0\} \mid n - |\beta_0|$$

The derived grade is the $\mathcal{G}$ term $\sum_k m_k = n - |\beta_0| = \sum_{k=0}^{n-1} \chi_{k \notin \beta_0}$ (grade 0 on the $|\beta_0|$ equal positions, grade 1 on the $n - |\beta_0|$ divergent ones); by soundness it bounds $c\langle 1\rangle - c\langle 2\rangle \leq n - |\beta_0|$. Since $|\beta_0| \geq d_0$ (i.e. $-|\beta_0| \leq -d_0$), arithmetic gives $n - |\beta_0| \leq n - d_0$, yielding the stated property $c\langle 1\rangle - c\langle 2\rangle \leq n - d_0$. The threshold bound $n - d_0$ is therefore a numeric weakening of the exact grade, *not* itself a supplied grade ($n - d_0$ with free d_0 is not a $\mathcal{G}$ term, so it cannot be a **conseq** target).

Key difference from CountPositive. CountPositive has read-only arrays. InplaceMap *mutates* the arrays (when the guard fires), requiring **update1** on the cell-morphing β_0 positions where equality is preserved. For the asymmetric source above, δ is actually **fixed** at β_0: when $x_1 = x_2$ at $i \in \beta_0$, both runs take the same branch (both update or both skip), and equality is preserved. When $i \notin \beta_0$, neither run gains new Φ information; **update2**/**skipL-skipR** preserves the same β_0. The **awhile-evolving** rule would be needed only if the guard's branch could create new equalities (out of scope for this asymmetric variant).

Invariant.

$$I \equiv i\langle 1\rangle = i\langle 2\rangle \wedge 0 \leq i \leq n \wedge c\langle 1\rangle - c\langle 2\rangle \leq i - |\beta_0 \cap [0, i)|$$

Evolving δ. At iteration k (where $k = n - i$, the remaining iterations), δ_k tracks positions where equality is *confirmed* in the post-update arrays. Only positions $i \in \beta_0$ gain confirmed equality after processing (because $x_1 = x_2 \Rightarrow f(x_1) = f(x_2)$). Positions $i \notin \beta_0$ use **update2**, which does *not* add i to the equality set.

$$\delta_k = (A_1, A_2) \mapsto^{=} \beta_0 \cap [0, n - k)\ \cup\ (\beta_0 - [0, n - k))$$

That is, δ_k starts as β_0 and stays β_0: the already-processed β_0 positions retain their equality (confirmed by **update1**), and unprocessed β_0 positions retain their original equality. So $\delta_k = (A_1, A_2) \mapsto^{=} \beta_0$ for all k.

Simplification: Since f preserves equality at β_0 positions and **update2** preserves β at non-β_0 positions, δ is actually **fixed** (not evolving). We can use the simpler **awhile** rule instead of **awhile-evolving**.

Why awhile-evolving is still relevant. If f could *create* new equalities (e.g., a constant function $f(x) = 0$), then positions $i \notin \beta_0$ would satisfy $f(x_1) = f(x_2) = 0$ after processing, and δ would genuinely grow. In that case, **awhile-evolving** with $\delta_k \to \delta_{k-1}$ where $\delta_{k-1} = \delta_k \cup \{n-k\}$ is needed. For general f, we use the fixed-δ **awhile** and the derivation below.

Step 1: Initialization. Using **assign**: $i_1 := 0; c_1 := 0 \sim i_2 := 0; c_2 := 0$. Establishes I with $\delta = (A_1, A_2) \mapsto^{=} \beta_0$.

Step 2: While loop. Using **awhile** (fixed δ, since β_0 is preserved):

$$\frac{\forall k.\ \vdash \{\delta\} \mid \{I \wedge i < n\}\ \ \mathit{body}_1 \sim \mathit{body}_2\ \ \{I\} \mid \{\delta\} \mid m_k}{\vdash \{\delta\} \mid \{I\}\ \ \textbf{while} \ldots\ \ \{I \wedge i \geq n\} \mid \{\delta\} \mid \sum m_k}$$

Step 3: Loop body at $i \in \beta_0$ (grade 0).

1. **read1**: $i \in \beta_0$, so δ gives $\Phi(x_1, x_2)$, i.e., $x_1 = x_2$. Grade 0.
2. Since $x_1 = x_2$, both runs take the same branch of `if x > 0`.
3. Both branches: either both `A[i] := x+1` (**update1**, grade 0) or both `skip`(grade 0). In either case, the cell pair stays equal at i, so δ is preserved.
4. Cost unchanged: $c\langle 1\rangle - c\langle 2\rangle$ doesn't change. Body grade = 0. ✓

Step 4: Loop body at $i \notin \beta_0$ (grade 1).

1. **read2**: $i \notin \beta_0$, no Φ info. Grade 0.
2. Each run independently evaluates `if x > 0`; branches may differ.
3. Worst case for $c\langle 1\rangle - c\langle 2\rangle$: run 1 enters the if-branch and updates ($+1$ to c_1), run 2 takes the else and skips (0 to c_2). Cost diff contribution ≤ 1. Captured by **conditionalL** (or analogously **updateL-skipR** on the if-branch instance) with grade 1.
4. Body grade = 1. Invariant: $(c\langle 1\rangle - c\langle 2\rangle)' \leq (i - |\beta_0 \cap [0, i)|) + 1 = (i+1) - |\beta_0 \cap [0, i+1)|$. ✓

Step 5: Termination. At exit: $I \wedge i \geq n$, so $c\langle 1\rangle - c\langle 2\rangle \leq n - |\beta_0 \cap [0, n)| \leq n - d_0$. Total grade $= \sum m_k = n - |\beta_0|$. ✓

Rules exercised (beyond CountPositive).

Rule	Why needed
update1 (grade 0)	Mutable array write where $\Phi(f(x_1), f(x_2))$ confirmed; β_0 preserved
update2 (grade 0)	Mutable array write where Φ not confirmed; β_0 preserved
read1/read2 (grade 0)	Same as CountPositive
awhile (fixed δ)	$\delta = \beta_0$ is invariant since both update rules preserve β_0

8.12 Level 2 Derivation: DualCount (Two-Array, Heap-Frame)

This derivation exercises **heap-frame** and the $\star$ composition for programs that take *two* array inputs.

```
DualCountPositive(A, B, n):
    c := 0
    for i = 0 to n-1:
        if A[i] > 0: c++
        if B[i] > 0: c++
    return c
```

Two runs: run 1 on (A_1, B_1), run 2 on (A_2, B_2).

Property. If both arrays agree *simultaneously* at $\geq d_0$ positions of $[0, n)$ (formally $|\beta_A \cap \beta_B \cap [0, n)| \geq d_0$, i.e. $A_1[i] = A_2[i] \wedge B_1[i] = B_2[i]$ holds at every $i \in \beta_A \cap \beta_B$ and there are at least d_0 such positions), then $c\langle 1\rangle - c\langle 2\rangle \leq 2(n - d_0)$.

Heap relation. Two arrays require a *composed* heap relation:

$$\delta = \underbrace{(A_1, A_2) \mapsto^{=} \beta_A}_{\delta_A} \star \underbrace{(B_1, B_2) \mapsto^{=} \beta_B}_{\delta_B}$$

where $\beta_A = \{i : A_1[i] = A_2[i]\}$ and $\beta_B = \{i : B_1[i] = B_2[i]\}$ are pure, interpretation-fixed specification sets (not values the program computes from heap reads). A position i is "fully equal" iff $i \in \beta_A \cap \beta_B$.

Key rule: heap-frame. The **heap-frame** rule allows us to reason about one array while framing the other:

$$\frac{\vdash \{\delta_A\} \mid \{P\}\ C_1 \sim C_2\ \{Q\} \mid \{\delta_A\} \mid m \qquad FV(\delta_B) \cap mod = \emptyset \qquad arrFV(\delta_B) \cap arrfoot = \emptyset \qquad arrfoot(C_j)\langle j\rangle \subseteq own_j(\delta_A) \quad local_{\delta_B,\delta_A}(P),\ local_{\delta_B,\delta_A}(Q)}{\vdash \{\delta_A \star \delta_B\} \mid \{P\}\ C_1 \sim C_2\ \{Q\} \mid \{\delta_A \star \delta_B\} \mid m}$$

Since A and B are independent arrays and the program only reads (no mutations), both δ_A and δ_B are preserved. When **heap-frame** is applied to the sub-derivation that processes array A while framing δ_B (and symmetrically to the sub-derivation over B framing δ_A), its side conditions hold because that sub-derivation's commands and assertions touch only the active array: it reads A without allocating, so $\mathit{arrfoot}(C_j)\langle j\rangle \subseteq \{A_j\} = \mathit{own}_j(\delta_A)$ (the footprint condition), and $\mathit{arrFV}_j(P), \mathit{arrFV}_j(Q) \subseteq \{A_j\} = \mathit{own}_j(\delta_A)$, which discharges the frame-locality premises $\mathit{local}_{\delta_B,\delta_A}(P)$, $\mathit{local}_{\delta_B,\delta_A}(Q)$ (Definition 6.5) by the syntactic sufficient condition (Lemma 6.1), all disjoint from $\mathit{own}_j(\delta_B) = \{B_j\}$. The positional correlation $\beta_A \cap \beta_B$ enters only through pure β-membership, not heap reads, so it does not affect locality.

Invariant.

$$I \equiv i\langle 1\rangle = i\langle 2\rangle \wedge 0 \le i \le n \wedge c\langle 1\rangle - c\langle 2\rangle \le 2i - 2|\beta_A \cap \beta_B \cap [0, i)|$$

Loop body at $i \in \beta_A \cap \beta_B$ (grade 0).

- **read1** on A: $i \in \beta_A$, learn $x_{A1} = x_{A2}$. Grade 0.
- First **conditional** (on $A[i] > 0$): guards agree ($x_{A1} = x_{A2}$). Both take the same branch. Two-sided, grade 0.
- **read1** on B: $i \in \beta_B$, learn $x_{B1} = x_{B2}$. Grade 0.
- Second **conditional** (on $B[i] > 0$): guards agree. Two-sided, grade 0.
- Total body grade: 0. Cost unchanged: $c' = c$.

Loop body at $i \notin \beta_A \cap \beta_B$ (grade ≤ 2). At least one of $i \notin \beta_A$ or $i \notin \beta_B$ (or both). Consider the worst case $i \notin \beta_A \wedge i \notin \beta_B$:

- **read2** on A: no Φ info. Grade 0.
- First **conditionalL**: guards may differ. Worst case: **assignL-skipR** (grade 1).
- **read2** on B: no Φ info. Grade 0.
- Second **conditionalL**: guards may differ. Worst case: **assignL-skipR** (grade 1).
- Total body grade: $0 + 1 + 0 + 1 = 2$. Cost: $c' \le c + 2$.

If only one array differs ($i \notin \beta_A$ but $i \in \beta_B$, or vice versa), the body grade is 1: one conditional uses two-sided (grade 0), the other uses one-sided (grade 1).

Total grade. The per-position grade is the indicator sum $m_i = \chi_{i\notin\beta_A} + \chi_{i\notin\beta_B}$, realizing the four cases of the table above (0 when both agree, 1 when exactly one disagrees, 2 when both disagree). Using the pointwise inequality $\chi_{i\notin\beta_A} + \chi_{i\notin\beta_B} \le 2\,\chi_{i\notin(\beta_A\cap\beta_B)}$:

$$\sum_{i=0}^{n-1} m_i \;\le\; 2\sum_{i=0}^{n-1} \chi_{i\notin(\beta_A\cap\beta_B)} \;=\; 2\big(n - |\beta_A\cap\beta_B\cap[0,n)|\big) \;\le\; 2(n-d_0),$$

the last step by the threshold $|\beta_A\cap\beta_B\cap[0,n)| \ge d_0$. The grade $\sum_i m_i$ is a $\mathcal{G}$ term (an indicator sum); $2(n-d_0)$ is its numeric upper bound, not a supplied grade.

Invariant maintenance. At $i \in \beta_A\cap\beta_B$: $c' = c$, $|\beta_A\cap\beta_B\cap[0,i')| = |\beta_A\cap\beta_B\cap[0,i)| + 1$.

$$2(i+1) - 2(|\beta_A\cap\beta_B\cap[0,i)|+1) = 2i - 2|\beta_A\cap\beta_B\cap[0,i)| \ge c = c' \quad \checkmark$$

At $i \notin \beta_A\cap\beta_B$: $c' \le c+2$, $|\beta_A\cap\beta_B\cap[0,i')| = |\beta_A\cap\beta_B\cap[0,i)|$.

$$c' \le (2i - 2|\beta_A\cap\beta_B\cap[0,i)|) + 2 = 2(i+1) - 2|\beta_A\cap\beta_B\cap[0,i')| \quad \checkmark$$

CHC correspondence. In the CHC encoding, the two arrays are tracked by a *single* distinguished position k with *two* abstraction bits: bk (for A) and bj (for B). The four HIT sub-cases correspond to the logic's case analysis on β_A and β_B membership:

Logic	**CHC**	**Cost**	**Grade**
$i \in \beta_A\cap\beta_B$	bk=1, bj=1	$c' = c$	0
$i \in \beta_A, i \notin \beta_B$	bk=1, bj=0	$c' \le c+1$	1
$i \notin \beta_A, i \in \beta_B$	bk=0, bj=1	$c' \le c+1$	1
$i \notin \beta_A \wedge i \notin \beta_B$	bk=0, bj=0	$c' \le c+2$	2

Rules exercised (beyond CountPositive/InplaceMap).

Rule	**Why needed**
heap-frame (grade m)	Compose $\delta_A \star \delta_B$; frame one while reasoning about the other
$\star$ **composition**	Two independent array relations in δ
Multiple **read1**/**read2**	Different β membership per array
Two **conditionalL**s in sequence	Grade $1+1=2$ per MISS position

Generalization to K counters (a design strength). DualCount's worst-case grade of 2 per asymmetric position arises purely from **sequence**-rule additivity: the loop body sequences two independent unit-counter updates, and

the graded **sequence** rule adds their grades ($1 + 1 = 2$ in the worst case). No K=2-specific rule is involved. The same compositional pattern extends to any fixed number K of independent per-position counters: sequencing K unit-counter updates in the body gives a worst-case body grade of K. By the same indicator-sum bound as above, now with $m_i = \sum_{r=1}^{K} \chi_{i \notin \beta_r} \leq K\, \chi_{i \notin \bigcap_r \beta_r}$, the total grade is $m = \sum_{i=0}^{n-1} m_i \leq K\big(n - |\bigcap_{r=1}^{K} \beta_r \cap [0, n)|\big) \leq K\,(n - d_0)$ *provided the single threshold d_0 lower-bounds the K-way simultaneous agreement*, $|\bigcap_{r=1}^{K} \beta_r \cap [0, n)| \geq d_0$; it is discharged by the same **sequence**/**awhile** composition (grade additivity is the **sequence** case of the per-rule soundness argument in Section 8.9). The framework thus handles K-counter relational cost for arbitrary K with *no new rules*. The bound is sound but not always tight: the factor K assumes every asymmetric position fails agreement on *all* K arrays at once, whereas a position lying in the symmetric difference of two agreement sets (e.g. $i \in \beta_A \triangle \beta_B$ for K=2) contributes grade 1 rather than 2 and realizes a strictly smaller cost difference. Tightening would replace the single threshold d_0 with per-array thresholds $d_0^{(1)}, \ldots, d_0^{(K)}$; we state the single-threshold $K\,(n - d_0)$ form as the strength claim and leave the per-array refinement as a presentation option.

9 Generic Graded-Effect Soundness

Theorem 8.2 is the *execution-cost* instance of a more general statement. Its proof has a characteristic shape: the composition and structural cases (sequence, conditionals, loops, consequence, frames) never inspect the cost model: they use only telescoping of signed differences, case analysis with max, finite per-iteration sums with nonnegative weakening, and transitivity, while the cost model enters *only* at the primitive leaves (the per-rule bounds of Lemma 8.3) and through the control-neutrality facts $c_{if} = c_{while} = 0$. This section isolates that structure. We parametrize the graded judgment by an *effect functional* over pairs of states, replace the primitive grade table by a per-application semantic obligation, and prove one generic soundness theorem. Three instances then cover all 22 Level 2 encodings: the execution-cost instance recovers Theorem 8.2 and its 15 examples, an output-disagreement instance covers **ClampPositive**, and an affine-accumulator instance covers the six remaining counter and value-weighted examples. Throughout, one judgment is graded for one effect instance at a time (no mixing of effect functionals inside a single derivation); every example below is single-effect.

Intuition: the grade is a budget, the effect is what it bounds. It helps to read the two roles separately. The *grade* m is a *syntactic budget*: a term of the fixed grade language $\mathcal{G}$ (naturals with $+$, max, bounded sums, and the order $\leq$), interpretation-determined and barred from reading mutable state. The *effect* $\mathcal{E}$ is the *semantic quantity the budget bounds*: a reading of the two final states as a signed integer. Soundness is the single inequality $\mathcal{E}(\sigma_1', \sigma_2') - \mathcal{E}(\sigma_1, \sigma_2) \leq [\![m]\!]_i(I)$, "the quantity moved by no more than the budget spent." Execution cost is one reading ($\mathcal{E}$ the cost difference, Instance E1); output disagreement (Hamming, Instance E2) and accumulator differences (Instance E3) are others. Across instances the grade algebra and the target of the measurement stay fixed: effect *changes* $\Delta\mathcal{E}$ always live in $(\mathbb{Z}, +)$ and add along a sequence by telescoping, so the grade's $+$ (sequence), max (branch), and bounded sum (loop) bound them uniformly. We do *not* require $\mathcal{E}$ itself to be additive or a monoid homomorphism; the telescoping identity supplies the only compositionality the proof uses, which is what lets a state-dependent reading such as the processed-prefix disagreement count of Instance E2 qualify without proving any additivity law for $\mathcal{E}$ itself. Generalizing the grade algebra itself to some other ordered monoid is a separate axis we do not pursue here. The effect functional adds no source-level transition rules to the operational semantics: the Instance E2 and E3 functionals read ordinary heap and scalar state, while Instance E1 uses a conservative ghost accumulator gc that presents the existing Section 3.2.3 cost component m and is erased from program behavior.

Notation: cardinality expressions in invariants. The instances below and the Level 2 derivations write prefix-cardinality expressions such as $|\beta \cap [0, i)|$ (and $|\beta \cap [0, n)|$, $|\beta|$) inside relational invariants and grades. These are

interpretation-determined shorthands, not object-level assertion terms: for a fixed well-formed interpretation I, β is an interpretation-level index set (Definition 6.2), so for each loop index i, $|\beta \cap [0,i)| = \sum_{j\in[0,i)}[\,j \in \beta\,] \in \mathbb{N}$ is a determined natural number. The object-level assertion grammar carries no set-cardinality constructor (set algebra in assertions is not part of the object language), so such an occurrence abbreviates this I-determined value, in the same interpretation-level reading already used for β itself. On the grade side the cardinality convention is the one fixed for the no-β fragment (a closed-form grade is the natural-number value of a finite-sum grade $\sum_k m_k$): the grade semantics defines $[\![|\beta|]\!]_i(I) = |[\![\beta]\!]_b(I)|$ for finite β, and Lemma 8.1 licenses the resulting closed forms. This convention is pre-existing (it is what the Count-Positive and InplaceMap invariants already use), and nothing in Theorem 9.2 depends on it.

Definition 9.1 (Effect functional; carrier)**.** *An* effect functional *is a map $\mathcal{E}$ assigning to every pair of states (σ_1, σ_2) an integer $\mathcal{E}(\sigma_1, \sigma_2) \in \mathbb{Z}$, equipped with a declared* carrier*: a set of scalar variables and array identifiers (possibly including a loop index) such that $\mathcal{E}(\sigma_1, \sigma_2)$ depends only on the carrier components of σ_1, σ_2 (*carrier locality*). We additionally require, as a formal hypothesis,* frame-extension invariance *(*C-frame*), stated precisely as Definition 9.2 below. The functional is a semantic object, not a grade term: it may read mutable program variables and heap cells, which grade terms may not (Definition 8.1, well-formedness of grade terms). For a pair of finite terminating executions $(\sigma_1, \sigma_1', m_1) \in [\![C_1]\!]$ and $(\sigma_2, \sigma_2', m_2) \in [\![C_2]\!]$, the* effect change *is $\mathcal{E}(\sigma_1', \sigma_2') - \mathcal{E}(\sigma_1, \sigma_2)$.*

Definition 9.2 (Frame-extension invariance (C-frame))**.** *An effect functional $\mathcal{E}$ satisfies* frame-extension invariance *when its effect change is unaffected by a preserved frame heap. Concretely, let the two full executions be $((s_j, h_j), (s_j', h_j'), m_j) \in [\![C_j]\!]$ for $j \in \{1, 2\}$, and suppose each initial heap splits as $h_j = h_j^a \uplus h_j^f$ ($h_j^a \perp h_j^f$) so that Lemma 6.2 factors the execution through the active fragment,*

$$((s_j, h_j^a), (s_j', h_j^{a\prime}), m_j) \in [\![C_j]\!] \quad \text{and} \quad h_j' = h_j^{a\prime} \uplus h_j^f,$$

with the frame heap h_j^f unchanged. Then C-frame requires

$$\begin{aligned}\mathcal{E}\big((s_1', h_1^{a\prime} \uplus h_1^f), (s_2', h_2^{a\prime} \uplus h_2^f)\big) - \mathcal{E}\big((s_1, h_1^a \uplus h_1^f), (s_2, h_2^a \uplus h_2^f)\big)\\ = \mathcal{E}\big((s_1', h_1^{a\prime}), (s_2', h_2^{a\prime})\big) - \mathcal{E}\big((s_1, h_1^a), (s_2, h_2^a)\big).\end{aligned}$$

That is, the effect change computed on the extended states equals the effect change computed on the active fragments. Carrier locality (Definition 9.1) supplies C-frame whenever the carrier is contained in the active fragments; an instance with a scalar-only carrier satisfies it trivially, since stacks are shared unchanged between a state and its frame extension.

Definition 9.3 ($\mathcal{E}$-graded relational validity)**.** *Let $\mathcal{E}$ be an effect functional. A graded relational statement $\vdash \{\delta\} \mid \{P\}\ C_1 \sim C_2\ \{Q\} \mid \{\delta'\} \mid m$ is* $\mathcal{E}$-valid*,*

written

$$\vDash^{m}_{\mathcal{E}} \{\delta\} \mid \{P\}\ C_1 \sim C_2\ \{Q\} \mid \{\delta'\},$$

if for every well-formed interpretation I (with $m \in \mathcal{G}$ and $[\![m]\!]_i(I) \in \mathbb{N}$ defined under I, exactly as in Definition 8.1) and every pair of finite terminating executions $(\sigma_1, \sigma_1', m_1) \in [\![C_1]\!]$, $(\sigma_2, \sigma_2', m_2) \in [\![C_2]\!]$ with $(\sigma_1, \sigma_2) \vDash_I P \mid \delta$:

$$(\sigma_1', \sigma_2') \vDash_I Q \mid \delta' \quad \textit{and} \quad \mathcal{E}(\sigma_1', \sigma_2') - \mathcal{E}(\sigma_1, \sigma_2) \leq [\![m]\!]_i(I).$$

Two differences are in play and should not be conflated. The functional $\mathcal{E}$ already encodes the relational *(left-run versus right-run) comparison: for the cost instance $\mathcal{E}(\sigma_1, \sigma_2) = gc\langle 1\rangle - gc\langle 2\rangle$, for the accumulator instance $\ell_1(\sigma_1) - \ell_2(\sigma_2)$. The grade bounds the* temporal *change of that relational reading, $\Delta\mathcal{E} = \mathcal{E}(\sigma_1', \sigma_2') - \mathcal{E}(\sigma_1, \sigma_2)$. When the precondition forces the initial relational reading to vanish, $\mathcal{E}(\sigma_1, \sigma_2) = 0$ (equal initial costs or accumulators, or an empty processed prefix, as every instance below arranges), the change-bound is exactly a bound on the final relational quantity $\mathcal{E}(\sigma_1', \sigma_2') \leq [\![m]\!]_i(I)$, recovering the $m_1 - m_2 \leq [\![m]\!]_i(I)$ form of Definition 8.1 (where the costs m_j likewise accumulate from 0). The bound is signed and one-sided, exactly as in Definition 8.1; the symmetric direction is obtained by deriving the swapped judgment $\vdash \{\delta\} \mid \{P\}\ C_2 \sim C_1\ \{Q\} \mid \{\delta'\} \mid m'$ for the* swapped *effect functional $\mathcal{E}^{\mathsf{sw}}(\sigma_2, \sigma_1) = -\mathcal{E}(\sigma_1, \sigma_2)$, and an absolute-value property $|\mathcal{E}| \leq B$ is established by the conjunction of the two one-sided judgments. Every absolute-value example below uses this swap convention; no single judgment carries a two-sided bound.*

Generic primitive obligation. In the $\mathcal{E}$-instanced logic, a *primitive* rule application (a two-sided or one-sided axiom leaf: skip, assign, read, update, alloc, and their one-sided variants) with precondition $P \mid \delta$ may carry any grade $m \in \mathcal{G}$ satisfying the side condition

$$\text{for all well-formed } I \text{ and all } (\sigma_1, \sigma_2) \vDash_I P \mid \delta: \quad \mathcal{E}(\sigma_1', \sigma_2') - \mathcal{E}(\sigma_1, \sigma_2) \leq [\![m]\!]_i(I),$$

where (σ_1', σ_2') are the post-states of the leaf's single paired step (for a one-sided leaf, the idle run's state is unchanged). This is a semantic side condition of the same kind as the guard-agreement premise $\vDash P \implies b_1\langle 1\rangle = b_2\langle 2\rangle$ of the two-sided **conditional** rule. The composition and structural rules are unchanged: sequence adds grades, conditionals take the common grade or $\max(m_1, m_2)$, loops sum per-iteration grades, **conseq** weakens, frames and **false** are as before. The instances below supply *canonical tables* discharging this obligation syntactically for the rule shapes that actually occur in the corpus.

Lemma 9.1 (Control neutrality)**.** *Let $\mathcal{E}$ be any effect functional. Pure control transitions (boolean guard evaluation of **if** and **while**, loop re-entry and exit, and* `skip`*) leave $\mathcal{E}$ unchanged on both runs.*

Proof. Guard evaluation is pure: boolean and arithmetic expressions, possibly reading array cells $u_a[e]$ or lengths $len(u_a)$, evaluate without modifying the store

or the heap, and `skip` has no state effect. A state-pair functional depends only on the states, so its value is unchanged. (Contrast the execution-cost setting, where control steps could in principle carry cost and the default model must stipulate $c_{if} = c_{while} = 0$; for state-pair effect functionals the neutrality is structural, not stipulated.) □

Theorem 9.2 (Soundness, generic graded effect)**.** *Let $\mathcal{E}$ be an effect functional. For every well-formed derivation of $\vdash \{\delta\} \mid \{P\}\ C_1 \sim C_2\ \{Q\} \mid \{\delta'\} \mid m$ in the $\mathcal{E}$-instanced Level 2 logic (the Level 2 rules of Section 8 with every primitive leaf's grade discharging the generic primitive obligation for $\mathcal{E}$, all grades in the derived sublanguage $\mathcal{G}$ of Lemma 8.1)*

$$\vDash^{m}_{\mathcal{E}} \{\delta\} \mid \{P\}\ C_1 \sim C_2\ \{Q\} \mid \{\delta'\}.$$

Primitive leaves. The leaves carrying the generic primitive obligation are the two-sided axiom rules **skip**, **assign**, **read1**/**read2**, **update1**/**update2**, **alloc1**/**alloc2**, and the one-sided axiom leaves **assignL-skipR**/**assignR-skipL**, **readL-skipR**/**readR-skipL**, **updateL-skipR**/**updateR-skipL** (there is no one-sided allocation rule). The generic primitive obligation is the per-leaf criterion each must satisfy; the canonical per-rule grade tables for the three corpus instances are given in Corollaries 9.3, 9.4, and 9.5.

Proof. By induction on the derivation, with the two obligations of Definition 9.3.

(i) Postcondition and binary-property preservation. Identical to the corresponding case of Theorem 6.3: grades do not influence post-states, and the $\mathcal{E}$-instanced rules differ from the Level 1 rules only in their grade annotations.

(ii) Effect-change bound. Each case of Lemma 8.3 is reused with the signed cost difference $m_1 - m_2$ replaced by the effect change $\Delta\mathcal{E} = \mathcal{E}(\sigma_1', \sigma_2') - \mathcal{E}(\sigma_1, \sigma_2)$:

- *Primitive leaves.* Immediate from the generic primitive obligation, whose hypothesis $(\sigma_1, \sigma_2) \vDash_I P \mid \delta$ is provided by the induction context.
- *Sequence* $(m_1 + m_2)$. The effect change telescopes *exactly* through the intermediate state pair $(\widehat{\sigma}_1, \widehat{\sigma}_2)$: $\Delta\mathcal{E} = [\mathcal{E}(\sigma_1', \sigma_2') - \mathcal{E}(\widehat{\sigma}_1, \widehat{\sigma}_2)] + [\mathcal{E}(\widehat{\sigma}_1, \widehat{\sigma}_2) - \mathcal{E}(\sigma_1, \sigma_2)]$. The first premise, applied to the initial pair, establishes the intermediate assertion (the second premise's precondition) and bounds the second bracket by $[\![m_1]\!]_i(I)$; the second premise, applied from the intermediate pair, bounds the first bracket by $[\![m_2]\!]_i(I)$. Note no additivity or subadditivity assumption on $\mathcal{E}$ is needed: telescoping is an identity of differences, as in the cost case (Lemma 8.3, sequence).
- *Two-sided conditional* (common grade m). Guard agreement forces both runs into the same branch; the chosen branch's premise bounds $\Delta\mathcal{E}$. Guard evaluation itself is $\mathcal{E}$-neutral by Lemma 9.1.
- *One-sided conditional* ***conditionalL*** $(\max(m_1, m_2))$. Case analysis on the left guard; the executed branch's premise bound is weakened to the maximum.

- *Synchronous* ***awhile*** *and* ***awhile-evolving*** ($\sum_k m_k$). Both runs perform finitely many lockstep iterations (finite terminating executions; variant argument as in Lemma 8.3). The effect change telescopes exactly over the iteration boundaries; between body executions only guard evaluation and loop re-entry occur, which are $\mathcal{E}$-neutral by Lemma 9.1, so $\Delta\mathcal{E} = \sum_j \Delta\mathcal{E}^{(j)}$ over the visited iterations. Each $\Delta\mathcal{E}^{(j)} \leq [\![m_k]\!]_i(I[k \mapsto k_j])$ by the per-iteration premise; the visited-subset weakening to $\sum_{k=1}^{n}$ uses $m_k \geq 0$, which holds because grades denote naturals (Lemma 8.1). Note that an in-body index assignment (e.g. $i := i + 1$) is a *body command*, not loop bookkeeping: if the index is in the carrier, that assignment is an effect-relevant primitive leaf and carries its obligation like any other (this matters for the output-disagreement instance below).

- ***conseq*** ($m' \leq m$): transitivity.

- ***heap-frame***. By command-frame factorization (Lemma 6.2), every conclusion execution factors through the active heap, namely $((s_j, h_j^a), (s_j', h_j^{a\prime}), m_j) \in [\![C_j]\!]$ with the frame heaps left unchanged by execution, and the rule's footprint side conditions confine the commands' array footprint to the active fragments. The premise bound applies to the factored (active) executions; frame-extension invariance (C-frame, Definition 9.2) equates the effect change on the extended states with the effect change on the active states. The bound is inherited.

- ***frame*** (assertion framing, no heap split). The conclusion executes literally the same commands on the same states as the premise, so $\Delta\mathcal{E}$ is that of the premise and the bound is inherited unchanged.

- ***false***: vacuous. *Continuation rules and* ***re-sync-derived***, ***whileL-skipR***, ***whileR-skipL***: reduce to the sequence and one-sided patterns exactly as in Lemma 8.3.

Composing (i) and (ii) at each rule yields $\mathcal{E}$-validity. □

Relation to the design-level conditions. The design of this section stated two instance conditions. *C-control* (control steps are effect-neutral) is *derived* for state-pair effect functionals (Lemma 9.1) rather than hypothesized. *C-frame* is retained as a formal hypothesis (the frame-extension invariance of Definition 9.2, cited by the **heap-frame** case of the proof), and its carrier-locality companion is what the instances' canonical tables use to give grade 0 to steps that do not touch the carrier (e.g. the **CompressPositive** output writes below).

9.1 Instance E1: Execution Cost

Corollary 9.3 (Execution-cost instance)**.** *Augment each run's state with a ghost accumulator gc, initialized to* 0*, incremented by the per-construct weight of the default cost model (*$c_{assign} = c_{read} = c_{update} = c_{alloc} = 1$, $c_{\texttt{skip}} = c_{if} = c_{while} =$

0*) at each construct, the standard ghost-instrumentation presentation of the cost semantics of Section 3.2.3, with* $(\sigma, \sigma', m) \in [\![C]\!]$ *iff the instrumented run takes* $\sigma[gc \mapsto 0]$ *to* $\sigma'[gc \mapsto m]$*. Take* $\mathcal{E}_{\text{cost}}(\sigma_1, \sigma_2) = gc\langle 1\rangle - gc\langle 2\rangle$ *(carrier:* gc*). Then the canonical primitive grades (two-sided rules* 0*, left-active one-sided rules* 1*, right-active one-sided rules* 0*) discharge the generic primitive obligation, and Theorem 9.2 yields exactly the conclusion of Theorem 8.2:* $m_1 - m_2 = \Delta\mathcal{E}_{\text{cost}} \leq [\![m]\!]_i(I)$ *for the* 15 *execution-cost examples.*

Proof. The obligation check per leaf is the corresponding primitive case of Lemma 8.3: a two-sided construct increments both gc by the same weight ($\Delta\mathcal{E}_{\text{cost}} = 0$); a left-active unit step gives $\Delta\mathcal{E}_{\text{cost}} = 1$; a right-active unit step gives $-1 \leq 0$. Initial ghost values are 0, so $\mathcal{E}_{\text{cost}}(\sigma_1, \sigma_2) = 0$ on initial states and the effect change equals $m_1 - m_2$. C-frame holds trivially: the carrier gc is a stack scalar, and stacks are shared unchanged between a state and its frame extension. □

Theorem 8.2 remains the primary statement of the execution-cost instance; the corollary records that it is literally an instance of the generic theorem under the ghost presentation, so nothing in the existing development changes.

9.2 Instance E2: Output Disagreement (ClampPositive)

Corollary 9.4 (Output-disagreement instance)**.** *Let* u *be a designated array and* i *a designated index variable, lockstep (*$i\langle 1\rangle = i\langle 2\rangle$ *maintained by the relational invariant). Take*

$$\mathcal{E}_{\text{dis}}(\sigma_1, \sigma_2) \;=\; \#\{\, j < i\langle 1\rangle : h_1(u)[j] \neq h_2(u)[j] \,\}$$

(carrier: u *and* i*), the number of* disagreeing positions in the processed prefix. *Suppose the derivation satisfies:*

(SP) single-pass*: the loop body writes* u *only at the current position* $i\langle 1\rangle$ *and never at positions* $< i\langle 1\rangle$*, and* i *strictly increases by* 1 *per iteration, so a position committed into the prefix is never re-written;*

(Z) *the precondition entails* $i\langle 1\rangle = 0$*, so* $\mathcal{E}_{\text{dis}} = 0$ *initially.*

Canonical primitive grades: every leaf that does not advance i *has grade* 0 *(writes at position* $i\langle 1\rangle$ *are outside the prefix, and non-carrier steps are neutral by carrier locality); the index-advance leaf* $i := i + 1 \sim i := i + 1$ *has grade* 0 *when its precondition forces* $h_1(u)[i] = h_2(u)[i]$ *(post-write values), and grade* 1 *otherwise. Then Theorem 9.2 bounds the final whole-output Hamming distance: at loop exit* $i\langle 1\rangle = n$*, so* $\#\{j < n : h_1(u)[j] \neq h_2(u)[j]\} \leq [\![m]\!]_i(I)$*.*

Proof. Obligation check. A write $u[i\langle 1\rangle] := \cdot$ changes no position $j < i\langle 1\rangle$, so $\Delta\mathcal{E}_{\text{dis}} = 0$ (SP excludes writes below $i\langle 1\rangle$). The index advance extends the prefix by the single position i, so $\Delta\mathcal{E}_{\text{dis}} = [\, h_1(u)[i] \neq h_2(u)[i] \,] \in \{0, 1\}$: when the leaf's precondition forces the post-write values equal, the indicator is 0; otherwise it

is *at most* 1, so the grade-1 entry is sound because the prefix extension adds a disagreement *indicator*, covering both the divergent and the healing outcome. (It is *not* claimed that disagreement is entailed off β: in ClampPositive the one-sided write case $A_1[i] < 0 \leq A_2[i]$ with $A_2[i] = 0$ writes 0 on the left and leaves 0 on the right; the cell *heals*, and the indicator is $0 \leq 1$.) By (Z) the initial effect is 0, so the effect-change bound is a bound on the final prefix count, which at exit is the whole-output Hamming distance. C-frame: $\mathcal{E}_{\text{dis}}$ reads only the scalar i and the array u; in any command-frame factorization of the loop, u belongs to the commands' array footprint and hence to the active fragment, where its contents agree with the unsplit heap, so the effect changes on extended and active states coincide. □

ClampPositive. $\Phi(x, y) = (x = y)$, $\beta \subseteq [0, n)$ the agreeing-input positions, in-place clamp ($u = A$, single-pass, lockstep). At $i \in \beta$: equal inputs take the same branch and perform equal writes (or both skip), so the index-advance precondition forces equal post-values, giving grade 0. At $i \notin \beta$: grade 1 by the indicator argument. Total grade $\sum_{k=1}^{n} m_k = n - |\beta \cap [0, n)| \leq n - d_0$ under $d_0 \leq |\beta \cap [0, n)|$, bounding the output Hamming distance *directly*. The initial effect is 0 by (Z); no initial-Hamming budget enters. This matches the encoding's counter discipline (`clamppositive_robustness.clp`: c initialized to 0; $c' = c$ at HIT-equal, $c' = c + 1$ at HIT-unequal and MISS).
The loop-body transition realizes this charge directly (the CaseSplit disjunction on the observed cell):

```
% clamppositive_robustness.clp   goal: c <= n - d0   (sat = verified; 1.3s, pcsa
%   effect c = #{ i : out1[i] <> out2[i] } (output Hamming); d0 = #agreeing inpu
( (i = kp and bkp = 1 and c' = c       and d' = d+1)    % HIT-equal: same clamp, n
or (i = kp and bkp = 0 and c' = c + 1 and d' = d)       % HIT-unequal: +1 disagree
or (i <> kp              and c' = c + 1 and d' = d) )   % MISS: worst case +1
```

Remark (separated alternative; not used by ClampPositive). For in-place, value-nonexpansive maps one can instead take raw whole-array Hamming $H(\sigma_1, \sigma_2)$ as the effect functional, prove every per-iteration effect change ≤ 0 (clamping never creates disagreement at an already-equal cell, and may heal an unequal one), and import $H(\text{initial}) \leq n - d_0$ from the precondition. The two routes must not be mixed: combining the prefix charges with an initial-Hamming budget would double-count to $2(n - d_0)$.

9.3 Instance E3: Affine Accumulator Difference

Corollary 9.5 (Affine-accumulator instance)**.** *Let ℓ_1, ℓ_2 be* fixed *linear combinations of designated scalar accumulators with nonnegative integer coefficients, the coefficients interpretation-determined, d_0-free, read-only parameters in the sense of the read-only-parameter convention (Definition 8.1, well-formedness of grade terms). Take*

$$\mathcal{E}_{\text{acc}}(\sigma_1, \sigma_2) \;=\; \ell_1(\sigma_1) - \ell_2(\sigma_2)$$

(carrier: the designated scalars). Canonical primitive grades:

- *steps not assigning a designated scalar: grade* 0 *(carrier locality; this includes heap writes, e.g. the* ***CompressPositive*** *output-array writes);*

- *paired increments* $c := c + e_1 \sim c := c + e_2$ *on a designated scalar* c *(coefficients* λ, μ *in* ℓ_1, ℓ_2*): any* $m \in \mathcal{G}$ *with* $\vDash P \implies \lambda\, e_1\langle 1\rangle - \mu\, e_2\langle 2\rangle \leq m$*;*

- *left-active increment (right idles): any* m *with* $\vDash P \implies \lambda\, e\langle 1\rangle \leq m$*; right-active: any* m *with* $\vDash P \implies -\mu\, e\langle 2\rangle \leq m$ *(grade* 0 *suffices when the increment is nonnegative);*

- *guarded conditionals over increments use* ***conditional*** *(guard agreement) or* ***conditionalL*** *with the sub-case table of the four guard combinations, exactly as in the CountPositive derivation (Section 8.10).*

Then Theorem 9.2 bounds $\Delta(\ell_1(\sigma_1) - \ell_2(\sigma_2))$*; when the precondition entails* $\ell_1(\sigma_1) = \ell_2(\sigma_2)$ *(accumulators initialized to* 0*, as in all six examples below), the bound is a bound on the final value* $\ell_1(\sigma_1') - \ell_2(\sigma_2')$*.*

Proof. Obligation check per leaf: an assignment to a non-designated variable or a heap write leaves both $\ell_1(\sigma_1)$ and $\ell_2(\sigma_2)$ unchanged (carrier locality). A paired increment changes the effect by $\lambda\, [\![e_1]\!](\sigma_1) - \mu\, [\![e_2]\!](\sigma_2)$, bounded by $[\![m]\!]_i(I)$ under P by the stated side condition; one-sided increments are the μ- (resp. λ-) omitted cases. The side conditions are semantic implications of the same form as the generic obligation, here made syntactic because increments are program expressions. C-frame holds trivially: the carrier consists of stack scalars only. □

The six examples. With β the relational agreement set tracked by the invariant (value equality, or an agreement *predicate* such as inclusion agreement for **threshold_filter_bounded**, legitimate since β is an interpretation-level index set):

- **NestedBranch** ($\ell_1 = \ell_2 = id$ on *cost*, increments +2 guarded): off-β **conditionalL** sub-cases give $\max(2 - 2,\, 2,\, -2,\, 0) \leq 2$; per-iteration grade $2 \cdot \chi_{k \notin \beta}$; total $2(n - |\beta|) \leq 2(n - d_0)$. The absolute-value property uses the swap convention.

- **CompressPositive** ($\ell = id$ on the write pointer j): off-β worst case 1; total $n - |\beta| \leq n - d_0$ (signed, one-sided, as in the encoding).

- **arraysum_mixed** ($\ell = id$ on s; $|a_1[i] - a_2[i]| \leq \epsilon$ off β): paired increment bound ϵ off β, 0 on β; total $(n - |\beta|)\, \epsilon \leq (n - d_0)\, \epsilon$; swap for $|s_1 - s_2|$.

- **conditional_cost** (guarded $+A[i]$, ϵ-L^∞ at all positions): the one-sided-branch sub-case has $a_2[i] \leq 0 < a_1[i] \leq a_2[i] + \epsilon \leq \epsilon$, so all four sub-cases are $\leq \epsilon$ off β; total $(n - d_0)\, \epsilon$.

- **threshold_filter_bounded** (shared array, $h_1 \le h_2$, $A[i] \ge 0$; β = inclusion agreement): both-include gives $A[i] - A[i] = 0$; disagreement entails $h_1 < A[i] \le h_2$ with only the left run including, bound h_2; right-only inclusion is impossible ($h_1 \le h_2$); total $(n - d_0)\, h_2$.

- **ArrayDoubleOpNI_robust_hFT**: judgment (a) takes $\ell_1 = 2\, y\langle 1\rangle$, $\ell_2 = y\langle 2\rangle$, bounding $2y_1 - y_2$; the swapped judgment (b) bounds $y_2 - 2y_1$; this swapped orientation is the encoding's ghost $c = y_2 - 2y_1$ (`ArrayDoubleOpNI_robust_hFT.clp`); judgment (a) alone does not match the ghost. Per-step paired increment $y := y + e_1 \sim y := y + 2\, e_2$ changes the effect by $\pm 2(a_1[i] - a_2[i])$ depending on orientation, bounded by $2\epsilon \cdot \chi_{k \notin \beta}$ in both; totals $2(n - d_0)\, \epsilon$ each; together $|2y_1 - y_2| \le 2(n - d_0)\, \epsilon$. The *graded segment ends at loop exit*: the post-loop one-sided doubling $y_1 := 2y_1$ lies outside the graded judgment, and the final property is recovered by post-condition re-expression: $y_1^f = 2\, y\langle 1\rangle$, so $|y_1^f - y_2^f| = |2y_1 - y_2|$ evaluated at loop exit. This is an ungraded Level 1 substitution step.

Encoding excerpts (the six examples). Each loop-body transition charges the accumulator c per processed cell exactly as the affine table above prescribes: HIT-equal is neutral, and off-β steps are bounded by the example's weight (2, ϵ, h_2, or 2ϵ). A direct `sat` goal is the positive-form verification record; a violation-head `unsat` goal is the corpus's expected violation probe (corroborating, not by itself a universal-bound proof; see the goal-polarity convention of Section 10.6). The universal bound for each example is the affine-accumulator argument above.

```
% nested_branch_sync.clp            goal: c > 2*(n - d0)
(violation-head unsat)
%   cost += 2 per positive;  bound |cost1 - cost2| <= 2*(n - d0)
( (i=kp and bkp=1 and c' = c                    and d'=d+1)
% HIT-equal: neutral
or (i=kp and bkp=0 and c' <= c+2 and c' >= c-2 and d'=d)
% HIT-unequal: +-2
or (i<>kp          and c' <= c+2 and c' >= c-2 and d'=d) )
% MISS: +-2

% compresspositive_sync.clp         goal: j1 - j2 <= n - d0
(sat = verified; 1.4s, pcsat_tbq_ar.json)
%   c = write-pointer difference j1 - j2 (output length);
bound n - d0
( (i=kp and bkp=1 and c' = c     and d'=d+1)   % HIT-equal: no length gap
or (i=kp and bkp=0 and c' = c+1 and d'=d)     % HIT-unequal: +1
or (i<>kp          and c' = c+1 and d'=d) )   % MISS: +1

% arraysum_mixed_sync.clp           goal: c > (n - d0)*eps
(violation-head unsat)
%   c = s1 - s2;  inputs eps-close off beta (Linf);  bound |s1 - s2| <= (n - d0)
```

```
( (i=kp and bkp=1 and c' = c                          and d'=d+1)
% HIT-equal: 0 contribution
or (i=kp and bkp=0 and c' <= c+eps and c' >= c-eps and d'=d)
% HIT-unequal: <= eps
or (i<>kp          and c' <= c+eps and c' >= c-eps and d'=d) )
% MISS: <= eps

% conditional_cost_sync.clp       goal: c > (n - d0)*eps
(violation-head unsat)
%   c = cost1 - cost2;  cost += max(A[i],0), all positions eps-close;
bound (n - d0)*eps
( (i=kp and bkp=1 and c' = c                          and d'=d+1)
% HIT-equal: same contribution
or (i=kp and bkp=0 and c' <= c+eps and c' >= c-eps and d'=d)
% HIT-unequal: <= eps (1-Lipschitz)
or (i<>kp          and c' <= c+eps and c' >= c-eps and d'=d) )
% MISS: <= eps

% threshold_filter_bounded.clp    goal: c > (n - d0)*h2
(violation-head unsat; 5.6s, CONFIG pending)
%   c = y1 - y2;  shared array, h1 <= h2;  bound y1 - y2 <= (n - d0)*h2
( (i=kp and bkp=1 and c' = c                    and d'=d+1)
% HIT-agree (akp>h2 or akp<=h1)
or (i=kp and bkp=0 and c' = c + akp              and d'=d)
% HIT-disagree (h1<akp<=h2): +akp <= h2
or (i<>kp          and c' >= c and c' <= c+h2 and d'=d) )
% MISS: worst case <= h2

% ArrayDoubleOpNI_robust_hFT.clp goal: c > 2*(n - d0)*eps
(violation-head unsat; 26 it, ~1min, pcsat_tb_hyperprop2.json)
%   ghost c = y2 - 2*y1 (swapped affine);  inputs eps-close;
bound |2*y1 - y2| <= 2*(n - d0)*eps
( (i=kp and bkp=1 and c' = c                                and d'=d+1)
% HIT-equal: 0
or (i=kp and bkp=0 and c' <= c+2*eps and c' >= c-2*eps and d'=d)
% HIT-unequal: <= 2*eps
or (i<>kp          and c' <= c+2*eps and c' >= c-2*eps and d'=d) )
% MISS: <= 2*eps
```

Weighted grades. The weighted grades above ($2\cdot\chi$, $\epsilon\cdot\chi$, $h_2\cdot\chi$, and the affine coefficients) require no new grade-language constructor: for a weight w that is a nonnegative, d_0-free, read-only logical/index parameter, $w\cdot m$ abbreviates the bounded sum $\sum_{j=1}^{w} m$, already a term of $\mathcal{G}$ (Lemma 8.1); the well-formedness side condition on weights is recorded there.

Coverage. Instances E1–E3 place all 22 Level 2 encodings under the graded soundness umbrella: 15 under the execution-cost instance (Theorem 8.2 / Corollary 9.3), **ClampPositive** under Corollary 9.4, and the six accumulator examples under Corollary 9.5. This is the *logic-side* coverage statement; it neither alters nor upgrades the empirical Tier-3 records of Section 10.5 (in particular the four convergence-limited *partial* value-weighted positives remain partial).

10 Connecting to Relational Cell Morphing (Level 2)

Section 8 gives the Level 2 logic, validity definition, and soundness theorem. This section shows how those logical derivations connect to the Level 2 cell morphing CHC encoding. We use CountPositive as the canonical example. Compare with the Level 1 monotonicity derivation in Section 7, which uses the *same* program but proves a qualitative property without grades.

10.1 The Program

```
CountPositive(A, n):
    c := 0; i := 0
    while (i < n):
        x := A[i]
        if x > 0: c++
        i++
    return c
```

Two runs: P_1 on array A_1, P_2 on array A_2.

10.2 The Relational Property

If A_1 and A_2 share at least d_0 equal positions, then $c\langle 1\rangle - c\langle 2\rangle \leq n - d_0$.

Formally, with $\Phi(x, y) = (x = y)$ and $\beta = \{i \in [0, n) : A_1[i] = A_2[i]\}$:

$$\vdash \{(A_1, A_2) \mapsto^{=} \beta\} \mid \{n > 0\}\ P_1 \sim P_2\ \{c\langle 1\rangle - c\langle 2\rangle \leq n - |\beta \cap [0, n)|\} \mid \{(A_1, A_2) \mapsto^{=} \beta\}$$

The source judgment generalizes to $n \geq 0$ (the $n = 0$ case is trivial: both runs execute no loop iteration, $\beta \cap [0, n) = \emptyset$, and the bound is $0 \leq 0$). We keep $n > 0$ here because the cell-morphing CHC correspondence below tracks an in-bounds distinguished cell $0 \leq k < n$, which does not exist when $n = 0$; the empty-array case would require a separate, vacuous zero-length encoding.

10.3 Derivation

Step 1: Initialization. Using **assign** and **sequence**:

$$\vdash \{\delta\} \mid \{n > 0\}\ \ c_1 := 0; i_1 := 0 \sim c_2 := 0; i_2 := 0\ \ \{I\} \mid \{\delta\}$$

where the **loop invariant** is:

$$I \equiv i\langle 1\rangle = i\langle 2\rangle \wedge 0 \leq i \leq n \wedge c\langle 1\rangle - c\langle 2\rangle \leq i - |\beta \cap [0, i)|$$

This maps to **CHC Clause 1 (Init)**: $Inv(0, n, k, ak_1, ak_2, bk, 0, d_0, 0)$.

Logic	CHC Encoding
$i\langle 1\rangle = i\langle 2\rangle$	single variable i (lockstep)
$\lvert\beta \cap [0, i)\rvert$ (source path)	d (abstract credit; the true/discovered count only on source-corresponding search paths)
$\lvert\beta \cap [0, n)\rvert \geq d_0$ (source adequacy)	$d_0 \geq 0$ (CLP syntactic constraint)
$c\langle 1\rangle - c\langle 2\rangle \leq i - d$	c (CHC charge counter, $c = i - d$)

Step 2: While loop. Using **awhile** (lockstep, $b_1\langle 1\rangle = b_2\langle 2\rangle$ since $i\langle 1\rangle = i\langle 2\rangle$):

$$\frac{\vdash \{\delta\} \mid \{I \wedge i < n\} \;\; body_1 \sim body_2 \;\; \{I\} \mid \{\delta\}}{\vdash \{\delta\} \mid \{I\} \;\; \textbf{while } (i < n) \textbf{ do } body_1 \sim \textbf{while } (i < n) \textbf{ do } body_2 \;\; \{I \wedge i \geq n\} \mid \{\delta\}}$$

$\delta = (A_1, A_2) \mapsto^{=} \beta$ is preserved because arrays are *read-only*.
This maps to **CHC structure**: Clause 2 (transition) feeds back into *Inv*.

Step 3: Loop body: Case $i \in \beta$ (HIT-equal). Using **read1**: since $i \in \beta$, we learn $\Phi(x_1, x_2)$, i.e., $x_1 = x_2$.

Then using **conditional**: since $x_1 = x_2$, both branches agree: $(x_1 > 0) = (x_2 > 0)$.

- Both positive: c_1++, c_2++, so $c' = c$ (cost neutral).
- Both non-positive: skip, skip, so $c' = c$.

Invariant: $c' = c$, $i' = i + 1$, $\lvert\beta \cap [0, i')\rvert = \lvert\beta \cap [0, i)\rvert + 1$.
$c' \leq i' - \lvert\beta \cap [0, i')\rvert = (i + 1) - (\lvert\beta \cap [0, i)\rvert + 1) = i - \lvert\beta \cap [0, i)\rvert \geq c$. ✓

CHC correspondence: HIT-equal branch ($i = kp$, $bkp = 1$, $c' = c$, $d' = d + 1$).

Step 4: Loop body: Case $i \notin \beta$ (HIT-unequal or MISS). Using **read2**: since $i \notin \beta$, no Φ information. Worst case: $c' \leq c + 1$. (In the CHC this position fires the HIT-unequal branch when it is observed under the search phase, $kp = i$ with $bkp = 0$, and the MISS branch when it is unobserved under the settled phase, $kp = k \neq i$; both charge $c' = c + 1$. The header label "MISS" in earlier drafts conflated these two CaseSplit branches.)

Invariant: $c' \leq c + 1$, $i' = i + 1$, $\lvert\beta \cap [0, i')\rvert = \lvert\beta \cap [0, i)\rvert$.
$c' \leq (i - \lvert\beta \cap [0, i)\rvert) + 1 = (i + 1) - \lvert\beta \cap [0, i)\rvert = i' - \lvert\beta \cap [0, i')\rvert$. ✓

CHC correspondence: HIT-unequal ($bkp = 0$, $c' = c + 1$, $d' = d$) or MISS ($i \neq kp$, $c' = c + 1$).

Step 5: Termination. At loop exit: $I \wedge i \geq n$, so $c \leq n - \lvert\beta \cap [0, n)\rvert$.

CHC Clause 6 (Goal): $c \leq n - d_0 \leftarrow Inv(\ldots), n \leq i, d \geq d_0$ (direct goal; `sat` = verified, matching the formal T_R of Section 10.6).

10.4 The PickK–Logic Correspondence

The key insight connecting the logic to the CHC:

	Logic (manual proof)	**CHC (PCSAT automation)**
β known?	Yes (part of δ)	No; PCSAT must discover
At $i \in \beta$	Apply **read1** $\to \Phi(x_1, x_2)$	PickK searching: $kp = i$, check equality
At $i \notin \beta$	Apply **read2** $\to$ no info	PickK: search $kp{=}i$ (HIT-unequal) or settled $kp{=}k$ (MISS); both charged
Cost bound	$\Delta c \leq i - \lvert\beta \cap [0, i)\rvert$ from I	$c_{chc} = i - d$ (synthesized by PCSAT)
Grade $\sum_k m_k$	$\sum_{i \notin \beta} 1 = n - \lvert\beta\rvert$	Total charged (HIT-unequal/MISS) steps

PickK is a cell refocus strategy. In the logic, β is known and **read1**/**read2** are applied deterministically based on β-membership. In the CHC, PickK chooses *where to observe*: during searching ($d < d_0$), it focuses the current position ($kp = i$), forcing an observation that reveals whether Φ holds (via bkp from Wit). During settled ($d \geq d_0$), it keeps the current cell ($kp = k$), so untracked positions yield MISS. (Under a *threshold* prophecy $d_0 < d^*$, a settled *tracked* position $i \in \beta$ with $k \neq i$ can also fire MISS, leaving an equal position uncredited; the formal soundness argument in Section 10.6 handles this by treating d as a loose abstract credit rather than the true count, so this exact-prophecy/search table is intuition only.) PickK does not itself decide β-membership; it chooses where to look, and the case split ($bkp = 1$ vs $bkp = 0$ vs $i \neq kp$) reveals the answer.

TransBetaSync (historical intuition). *This paragraph records an earlier sketch; the finalized translation in Section 10.6 uses the loose invariant with an abstract credit d and the actual PickK/Wit/CaseSplit clauses, not a concrete mutable β' state.* The earlier CHC sketch wrote the read1/read2 case split as:

$$\mathit{TransBetaSync}(\beta, \beta', i) = (\Phi(ai_1, ai_2) \wedge \beta' = \beta + \{i\}) \vee \beta' = \beta - \{i\}$$

encodes the automation-side witness update for the same case split: **read1** is the proof case where the current index is in the semantic tracked set and Φ is available; **read2** is the case where no pointwise relation is learned. The logic's β is a fixed semantic set for read-only arrays; the CHC variables track the solver's discovered approximation to that set.

10.5 Coverage of Level 2 Examples

The Level 2 CHC corpus shares one cell-morphing transition pattern, but only the V1 execution-cost effect instances are covered by Theorem 8.2 (the *logic*

soundness theorem; the separate read-only CHC *translation* theorem of Section 10.6 is narrower still; see its scope); all 22 are covered by the generic graded-effect theorem of Section 9. The partition by instance is:

Bucket	**Examples (count)**	**Soundness route**
V1 execution cost, K=1	13 unit-asymmetric counters (13)	Theorem 8.2
V1 execution cost, K=2	DualCount (1)	Theorem 8.2; two unit branches
V1 execution cost, mutable	InplaceMap (1)	Theorem 8.2 (logic); update vs skip
L1 quantitative invariant	ClampPositive (M=1 Hamming), NestedBranch, CompressPositive, arraysum_mixed, conditional_cost, threshold_filter_bounded, ArrayDoubleOpNI_robust_hFT (7)	Relational invariant over scalar counter; outside Theorem 8.2, covered by Cor. 9.4 (ClampPositive) and Cor. 9.5 (the six accumulator cases)

Empirical Tier-3 evidence. The 22 Level 2 examples have been evaluated under the strict Tier-3 methodology (positive sat + negation unsat + family-specific vacuity unsat + minimal-d_0 vacuity unsat) using PCSAT/CoAR. Of the 22, **18 are FULL QUAD** (all four probes pass as expected). For the remaining **4**, the negation and both vacuity probes pass but the positive-form probe is PCSAT convergence-limited within our time budget; we therefore record these as **partial** (violation-form sanity-checked), *not* as positive-form verified. This is deliberately weaker than FULL: as the Goal-polarity-convention paragraph of Section 10.6 establishes, a violation-form `unsat` certifies only that *some* reachable terminal meets the bound, so it is not by itself a universal-bound proof. The universal bound for these four (the value-weighted cases outside Theorem 8.2) is carried by the relational quantitative-invariant route recorded in the table above, not by the violation-form probe. Per-example records, with the same caveat, are in `working-folder/TIER3_FINAL_RESULTS.md`.

Thus **DualCount**, not NestedBranch, is the canonical K=2 V1 execution-cost example. NestedBranch may be read as operational at the encoding level because its source variable is named `cost`, but under literal V1 semantics the assignment `cost` += 2 is one scalar assignment. ConditionalCost is likewise value-weighted source-state reasoning: the update `cost` += $A[i]$ contributes one V1 assignment cost, while the accumulated value difference is proved by a quantitative invariant.

10.6 Formal CHC Translation (Level 2, Synchronous)

We now formalize the translation from a Level 2 derivation to a CHC clause system. The translation is *derivation-pattern-guided*: it maps the proof structure (which rules were applied) to CHC clauses, not the program syntax alone.

10.6.1 CHC Unknown Signature

For a synchronous Level 2 derivation over a single shared array, the CHC system uses three unknown predicates:

$Inv(V_s, k, ak_1, ak_2, bk, d, d_0, c)$	:	loop invariant over cell morphing state
$PickK(V_s, k, ak_1, ak_2, bk, d, d_0, c, kp)$	:	cell refocus strategy
$Wit(V_s, kp, ak_{1p}, ak_{2p}, bkp, d, d_0, c)$	:	cell value witness at position kp

where V_s collects the shared program variables (e.g., i, n in lockstep programs), k is the current distinguished cell, ak_1, ak_2 are the cell values from each run, $bk = [\Phi(ak_1, ak_2)]$ is the relation bit, d is the credited-count slot (an *abstract credit* in the inductive model $\mathcal{I}$, equal to the observed-equal count $\leq |\beta \cap [0, i)|$ on source-corresponding paths), d_0 is the *free threshold prophecy* ($d_0 \geq 0$ in the CHC, with the source-level threshold $d_0 \leq d^* = |\beta \cap [0, n)|$), and c is the CHC charge counter $c_{chc} = i - d$ (it over-approximates the source cost difference $\Delta c = c\langle 1\rangle - c\langle 2\rangle$, related only by $\Delta c \leq c_{chc}$; see the Notation in Section 10.6).

Correspondence to logic concepts.

CHC Unknown	Logic Concept	Role
Inv	Loop invariant P_{loop} + binary property δ	Combined relational state
$PickK$	Cell refocus strategy	Where to observe: $kp{=}i$ (search) or $kp{=}k$ (settled)
Wit	Cell value after refocus	Witnessed ak_{1p}, ak_{2p}, bkp at position kp
d, d_0	abstract credit ($d \leq \lvert\beta \cap [0, i)\rvert$); free threshold prophecy ($0 \leq d_0 \leq d^*$)	Credited count and target; source reading $d{=}\lvert\beta \cap [0, i)\rvert$, $d_0{=}d^*{=}\lvert\beta \cap [0, n)\rvert$
c (c_{chc})	CHC charge counter $i - d$; bounds source Δc	Grade-enabled cost bound

10.6.2 Translation T_R: Derivation to CHC Clauses

Given a Level 2 derivation $\mathcal{D}$ for a synchronous loop program, the translation $T_R(\mathcal{D})$ produces a CHC clause system consisting of six clause families:

Clause 1 (Init). From the **assign** + **sequence** rules establishing the loop precondition, matching the shipped `.clp`:

$$Inv(V_0, k, ak_1, ak_2, bk, 0, d_0, 0) \leftarrow Pre(V_0) \wedge 0 \le k < n \wedge ConsistBk(ak_1, ak_2, bk) \wedge d_0 \ge 0$$

where *Pre* encodes the precondition P and *ConsistBk* enforces $(ak_1 = ak_2 \wedge bk = 1) \vee (ak_1 \neq ak_2 \wedge bk = 0)$. The prophecy d_0 is a free variable constrained only by $d_0 \ge 0$; the source-level precondition additionally guarantees the *threshold* $d_0 \le |\beta \cap [0, n)|$, used in part (1) and Corollary 10.7. The inductive model $\mathcal{I}$ defined below satisfies this relaxed Init directly, so we prove satisfiability of the translated system $T_R(\mathcal{D})$ itself; no exact-prophecy specialization is needed.

Clause 2 (Transition). From the **awhile** body premise, combining **read1**/**read2** case splits:

$$Inv(V_s', kp, ak_{1p}, ak_{2p}, bk', d', d_0, c') \leftarrow Inv(V_s, k, ak_1, ak_2, bk, d, d_0, c) \wedge Guard(V_s) \wedge Step(V_s, V_s')$$

$$\wedge\, PickK(\ldots, kp) \wedge\, Wit(\ldots, ak_{1p}, ak_{2p}, bkp, \ldots) \wedge\, CaseSplit(i, kp, bkp, c, c', d, d') \wedge\, ConsistBk(ak_{1p}, ak_{2p}, bk')$$

where *CaseSplit* encodes the three-way disjunction from **read1**/**read2**:

$$\begin{aligned} CaseSplit = {} & (i = kp \wedge bkp = 1 \wedge c' = c \wedge d' = d + 1) && \text{(HIT-equal: } \mathbf{read1}\text{)} \\ \vee\ & (i = kp \wedge bkp = 0 \wedge c' = c + 1 \wedge d' = d) && \text{(HIT-unequal: } \mathbf{read2}\text{)} \\ \vee\ & (i \neq kp \wedge c' = c + 1 \wedge d' = d) && \text{(MISS: } \mathbf{read2}\text{)} \end{aligned}$$

Clause 3 (Witness). The witness clause exposes the cell values at the distinguished position. Following the `.clp` encoding, its head cell variables *coincide* with the body's (there are no free head variables):

$$Wit(V_s, k, ak_1, ak_2, bk, d, d_0, c) \leftarrow Inv(V_s, k, ak_1, ak_2, bk, d, d_0, c) \wedge i < n$$

Refocusing is performed not inside Clause 3 but by Clause 2's body call $Wit(V_s, kp, ak_{1p}, ak_{2p}, bkp, d, d_0, c)$ (with kp bound by *PickK*, $kp \in \{i, k\} \subseteq [0, n)$). Unifying that call against this clause demands $Inv(V_s, kp, ak_{1p}, ak_{2p}, bkp, d, d_0, c)$: the invariant must hold with the distinguished cell set to kp and cell values ak_{1p}, ak_{2p}, bkp. This is satisfiable because $\mathcal{I}(Inv)$ is the *over-approximation* (see Intended Interpretations below): it admits any *ConsistBk* cell triple at any $kp \in [0, n)$, so the refocused obligation holds immediately. The intended witness pins $bkp = [\, kp \in \beta \,]$ (equivalently the heap values $ak_{1p} = h_1(u_1)[kp]$, $ak_{2p} = h_2(u_2)[kp]$, $bkp = [ak_{1p} = ak_{2p}]$). (The earlier draft wrote free head variables $kp, ak_{1p}, ak_{2p}, bkp$ with the body keeping the old $k, ak_1, \ldots$, which the restrictive witness interpretation did not satisfy; matching the `.clp` head-equals-body shape removes that mismatch.) This head-equals-body shape suffices for the *exhibited*-model statement of Theorem 10.6; for *arbitrary*-model forward soundness it is strengthened to a refocus-completing form (Section 10.7).

Clauses 4–5 (PickK). From the two-phase β-discovery strategy:

$$PickK(\ldots, i) \leftarrow Inv(\ldots) \wedge d < d_0 \quad \text{(searching: refocus to current position)}$$
$$PickK(\ldots, k) \leftarrow Inv(\ldots) \wedge d \geq d_0 \quad \text{(settled: keep current cell)}$$

The searching phase forces observation at the current position ($kp = i$), yielding either **read1** (if $bkp = 1$, i.e., Φ holds) or **read2** (if $bkp = 0$). The settled phase keeps the distinguished cell fixed ($kp = k$), so positions $i \neq k$ yield MISS (no new information).

Clause 6 (Goal). From **conseq** at loop exit, encoding the cost bound. Writing c_{chc} for the last *Inv* slot (the CHC charge counter; see the Notation paragraph below):

$$c_{chc} \leq n - d_0 \leftarrow Inv(V_s, k, ak_1, ak_2, bk, d, d_0, c_{chc}) \wedge \neg Guard(V_s) \wedge d \geq d_0$$

The right-hand bound $n - d_0$ is the CHC *threshold* bound (in terms of the free prophecy d_0); the exact derivation grade is $[\![m]\!] = n - d^*$ with $d^* = |\beta \cap [0, n)|$, and $n - d_0 = [\![m]\!]$ only in the exact-prophecy case $d_0 = d^*$. The generic goal proved in Theorem 10.6 is $c_{chc} \leq n - d_0$, which the source grade $[\![m]\!] = n - d^*$ implies under the precondition threshold $d_0 \leq d^*$ (Corollary 10.7).

Goal convention of T_R vs. the corpus files. The Goal clause family T_R emits (Clause 6 above) is the *direct* form $c_{chc} \leq n - d_0$ (satisfiability = verified), and every part (2) satisfiability claim in this section and its extensions (Theorems 10.6, 10.11, 10.16) is about this direct system. The default corpus mains differ by example: `dualcount_twoarray.clp` ships the direct goal $c \leq 2(n - d_0)$ (`sat` = verified), matching T_R exactly, whereas `demo2a_countpositive_pickk_sync.clp` and `inplacemap_sync.clp` ship the *violation-head* goal $c > n - d_0$ (`unsat` = verified), with the direct goal realized by their `_pos` probe files. By the Goal-polarity-convention paragraph below, the violation-head form is the strictly weaker obligation (its `unsat` certifies only that *some* reachable terminal meets the bound); we therefore state T_R's soundness for the direct goal it emits, and read a violation-head `unsat` on a corpus main as corroborating evidence, not as the universal-bound proof itself.

10.6.3 Translation Soundness

End-to-end pipeline. The full verification pipeline is:

$$\text{Level 2 derivation } \mathcal{D} \xrightarrow{\text{Thm 8.2}} \text{valid judgment} \xrightarrow{T_R} \text{CHC clause system} \xrightarrow{\text{PCSAT}} \text{sat/unsat}$$

For derivations inside the V1 execution-cost fragment, Theorem 8.2 (Section 8) guarantees that a derivation yields a valid relational judgment. The translation produces the translated CHC clause system $T_R(\mathcal{D})$; we exhibit an inductive invariant $\mathcal{I}$ that satisfies it (so the system is verifiable) for the lockstep/equality fragment below.

Scope and assumptions. We prove the translation for **single-pass lock-step loops with** $\Phi = (=)$ (equality) over **read-only arrays**, restricted to the V1 execution-cost encodings of this shape, with CountPositive as the running instance. Source-state/value-accumulator examples such as CompressPositive use the same CHC idiom but are outside Theorem 8.2 unless a separate source-state graded-effect theorem is supplied. *Mutable*-array examples such as InplaceMap are outside *this* (read-only) statement, whose Clause 2 passes the focused cell values through unchanged ($h' = h$); they are handled by the mutable-array extension (Theorem 10.11, Section 10.8), which folds a net cell update τ into Clause 2 and adds an exact-β preservation lemma. There the closure argument (Lemma 10.4) is itself insensitive to the cell-update map; the change is confined to the source-corresponding inclusion direction. For this translation we take β to be the *exact* agreement set $\beta = \{j \in [0, n) : u_1[j] = u_2[j]\}$, so $i \in \beta \iff u_1[i] = u_2[i]$ and $i \notin \beta \implies u_1[i] \neq u_2[i]$. This exactness is an *additional representation choice* for the CountPositive translation, *not* a consequence of Definition 6.2: a binary property only gives the must-direction $i \in \beta \implies u_1[i] = u_2[i]$, and here we additionally pin its converse by choosing β to be the full agreement set under $\Phi = (=)$. We assume $n > 0$ (non-empty array). Writing $d^* := |\beta \cap [0, n)|$ for the exact count, the inductive invariant below uses the shipped *threshold* prophecy directly: d_0 is free with $d_0 \geq 0$ in the CHC, and the source-level precondition supplies the threshold $d_0 \leq d^*$ that Corollary 10.7 and part (1) consume. The scope is the CountPositive CaseSplit shape, whose accounting is $c + d = i$ (equivalently $c = i - d$); other Level-2 encodings are covered only insofar as they share this accounting. Extension to a general Φ with a genuine must-interpretation of β (where $i \notin \beta$ does not imply $\neg\Phi$) requires a modified CaseSplit that treats HIT-unequal conservatively; we discuss this at the end.

Notation. We write P_{loop} for the **awhile** rule's loop invariant (to avoid collision with interpretation I). We write η for the interpretation of logical variables.

Source cost versus CHC charge. We distinguish two quantities. The *source signed cost difference* $\Delta c := c\langle 1\rangle - c\langle 2\rangle$ is what the Level-2 judgment bounds ($\Delta c \leq [\![m]\!]$, part (1)). The *CHC charge counter* is the last *Inv* argument, written c_{chc} (named c in the `.clp`). The CaseSplit leaves it unchanged on HIT-equal and increments it by 1 on HIT-unequal/MISS, while the credit d rises by 1 on HIT-equal and is unchanged otherwise; the transition therefore maintains the purely algebraic invariant

$$c_{chc} = i - d,$$

where d counts the *credited* (HIT-equal) steps so far. On the *source-corresponding* path, where the witness reports the true bit $bkp = [\, kp \in \beta \,]$, d equals $|\beta \cap [0, i)|$ *while searching*; once PickK settles, d is an *observed-credit subcount* that need not track $|\beta \cap [0, i)|$ (a settled step with $kp{=}k{=}i$ can still HIT-equal and increment d, while other settled steps leave later β-positions uncredited), so in general

$d \le |\beta \cap [0,i)|$, a strict subcount when $d_0 < d^*$. It is an abstract credit, *not* literally capped at d_0. Under-crediting only *raises* $c_{chc} = i - d \ge i - |\beta \cap [0,i)|$, so the Level-2 loop-invariant bound $\Delta c \le i - |\beta \cap [0,i)|$ gives

$$\Delta c \;\le\; c_{chc} \qquad \text{(on the source-corresponding path).}$$

We never identify Δc with c_{chc}: at a charged position the source difference may change by $+1$, 0, or -1 (the last is the right-active **assignR-skipL** case, grade 0), whereas c_{chc} rises by exactly 1; and on *non*-source witness paths d need not track β at all (it is then a free abstract credit). Crucially, the CHC Goal bounds c_{chc} *terminally and algebraically* (Lemma 10.5: at exit $c_{chc} = n - d \le n - d_0$ whenever $d \ge d_0$), *not* via any per-step charge-vs-grade comparison and *not* by claiming bad witnesses only lower c_{chc} (a mis-crediting witness can raise c_{chc} on a prefix; the bound is purely about the terminal state). The source bound $\Delta c \le [\![m]\!]$ is carried independently by the logic (part (1)); identifying c_{chc} with Δc component-by-component was the original defect this section repairs.

State encoding. Given a relational state (σ_1, σ_2) satisfying P_{loop} at iteration i under interpretation η, and given $\delta = (u_1, u_2) \mapsto^{=} \beta$ with $\beta = \{j \in [0,n) : h_1(u_1)[j] = h_2(u_2)[j]\}$, define:

$$enc(\sigma_1,\sigma_2,\eta,\beta,i) = \big(\; \underbrace{V_s(\sigma_1,\sigma_2)}_{\text{shared, incl. } i,n},\; k,\; \underbrace{h_1(u_1)[k]}_{ak_1},\; \underbrace{h_2(u_2)[k]}_{ak_2},\; \underbrace{[ak_1{=}ak_2]}_{bk},\; \underbrace{|\beta \cap [0,i)|}_{d},\; \underbrace{|\beta \cap [0,n)|}_{d_0},\; \underbrace{i - |\beta \cap [0,i)|}_{c_{chc}} \big)$$

where V_s collects the shared program variables (those appearing in P_{loop} from both runs), k is the distinguished cell ($0 \le k < n$), and the last slot is the CHC charge counter $c_{chc} = i - d$, *not* the source difference $\Delta c = c\langle 1\rangle - c\langle 2\rangle$ (see the Notation above). Here *enc* records the concrete, *source-corresponding* tuple, in which $d = |\beta \cap [0,i)|$ is the true count; the inductive invariant $\mathcal{I}(Inv)$ below *loosens* this to an abstract credit $0 \le d \le i$ with $c_{chc} = i - d$, which is what makes it closed under the Clause-2 transition for *every* witness, not only the source-corresponding one. The displayed *enc* also pins the d_0 slot to the exact count $d^* = |\beta \cap [0,n)|$, so it is the *exact-prophecy* embedding ($d_0 = d^*$); the inductive model $\mathcal{I}$ and Theorem 10.6 instead use a *free* threshold d_0 with only $0 \le d_0 \le d^*$ on source-corresponding instances.

Intended interpretations (inductive over-approximation). A CHC solution is an inductive invariant that *over-approximates* the *CHC-reachable* states (the least model of the definite clauses); from satisfiability alone it need not contain the image of every source execution, which is the separate adequacy direction, established for the exhibited $\mathcal{I}$ below (Lemma 10.1). We take $\mathcal{I}(Inv)$ to be the loop invariant lifted to the CHC variables, with the discovery slot relaxed to an *abstract credit* $d \in [0,i]$ (not pinned to the true β-count) and the

prophecy left free:

$$\begin{aligned}
\mathcal{I}(\mathit{Inv})(V_s, k, ak_1, ak_2, bk, d, d_0, c) &\iff 0 \le k < n \ \wedge\ \mathit{ConsistBk}(ak_1, ak_2, bk) \ \wedge\ d_0 \ge 0 \\
&\qquad \wedge\ 0 \le i \le n \ \wedge\ 0 \le d \le i \ \wedge\ c = i - d, \\
\mathcal{I}(\mathit{PickK})(\vec{v}, kp) &\iff \mathcal{I}(\mathit{Inv})(\vec{v}) \wedge \big((d < d_0 \wedge kp = i) \vee (d \ge d_0 \wedge kp = k)\big), \\
\mathcal{I}(\mathit{Wit})(\vec{v}) &\iff \mathcal{I}(\mathit{Inv})(\vec{v}),
\end{aligned}$$

where $i \in V_s$ is the shared loop counter and $\vec{v} = (V_s, k, ak_1, ak_2, bk, d, d_0, c)$. Three points make this a genuine model of the *translated* system $T_R(\mathcal{D})$:

- *Relaxed Init and prophecy.* $\mathcal{I}(\mathit{Inv})$ requires only $d_0 \ge 0$ (not $d_0 = d^*$) and admits *any* *ConsistBk* focus cell (ak_1, ak_2, bk), since it constrains only d, c, d_0 and the ranges, none of which depend on the focus cell's contents. So every tuple satisfying the shipped Init body lies in $\mathcal{I}(\mathit{Inv})$; no exact-prophecy specialization is needed.
- *Witness.* Since $\mathcal{I}(\mathit{Wit}) = \mathcal{I}(\mathit{Inv})$ admits every *ConsistBk* triple, the refocused call $\mathit{Wit}(V_s, kp, \ldots)$ raised by Clause 2 is satisfied for any $kp \in [0, n)$, matching the `.clp` shape $\mathit{Wit} \leftarrow \mathit{Inv}$, which likewise does not pin the witness bit.
- *Abstract credit.* d counts the *credited* (HIT-equal) steps, not the true $|\beta \cap [0, i)|$; the invariant constrains only $0 \le d \le i$ and $c = i - d$. This is exactly what makes $\mathcal{I}(\mathit{Inv})$ closed under *arbitrary* witnesses (Lemma 10.4): a witness that mis-reports a bit changes which CaseSplit branch fires, but the successor still satisfies $0 \le d' \le i'$ and $c' = i' - d'$.

The *source-corresponding* witness pins $bkp = [\, kp \in \beta \,]$ (equivalently $bkp = [\, h_1(u_1)[kp] = h_2(u_2)[kp] \,]$ under $\Phi = (=)$); along that path d is the credited (observed-equal) count, equal to $|\beta \cap [0, i)|$ while searching and an observed-credit subcount once settled (not literally capped at d_0), hence a subcount in general, and the concrete states sit inside $\mathcal{I}(\mathit{Inv})$ (Lemma 10.1). Closure and the Goal do *not* depend on the witness being source-corresponding; they hold for every *ConsistBk* witness.

Lemma 10.1 (Concrete states satisfy the invariant)**.** *If the* ***awhile*** *rule's invariant P_{loop} holds at iteration i, i.e., $(\sigma_1, \sigma_2) \vDash_\eta P_{loop} \mid \delta$ with $0 \le i \le n$, then $\mathcal{I}(\mathit{Inv})(\mathit{enc}(\sigma_1, \sigma_2, \eta, \beta, i))$ holds. That is, the concrete enc-image is* contained in *the over-approximation $\mathcal{I}(\mathit{Inv})$.*

Proof. The encoded tuple satisfies each defining conjunct of $\mathcal{I}(\mathit{Inv})$: $0 \le k < n$ (by initialization); $\mathit{ConsistBk}(ak_1, ak_2, bk)$ since $bk = [ak_1 = ak_2]$ by construction; $d_0 \ge 0$ (the chosen threshold satisfies $0 \le d_0 \le d^*$); the credit d (the observed-equal count, $\le |\beta \cap [0, i)| \le i$, so $d \in [0, i]$); and the charge slot $c_{chc} = i - d$. Hence the tuple lies in $\mathcal{I}(\mathit{Inv})$. This is *inclusion only*: $\mathcal{I}(\mathit{Inv})$ also contains guessed-cell and mis-credited tuples with no concrete pre-image (that is what makes it an over-approximation closed under arbitrary witnesses); we do not claim the concrete states are all of $\mathcal{I}(\mathit{Inv})$, nor that the concrete path is

the only operational CHC path (when $d_0 < d^*$ the shipped PickK settles early). As always, $c_{chc} = i - d$ is the CHC charge counter, not the source difference Δc, which P_{loop} bounds by $i - d$ on this path. □

Lemma 10.2 (Rule-local CHC satisfiability)**.** *For each logic rule applied in the loop body at iteration i under derivation $\mathcal{D}$, at least one CaseSplit branch of Clause 2 is satisfied under $\mathcal{I}$, and the invariant P_{loop} is preserved with the correct cost and discovery updates.*

Proof. This lemma concerns the *source-corresponding* witness path (where $bkp = [\, kp \in \beta \,]$), along which the credit d is the observed-equal count (equal to $|\beta \cap [0, i)|$ during searching, an observed-credit subcount once settled, not literally capped at d_0, so a subcount in general); it supports the inclusion of concrete states (Lemma 10.1), not the CHC closure (which holds for arbitrary witnesses, Lemma 10.4). We perform case analysis on (a) whether $i \in \beta$ or $i \notin \beta$, and (b) whether PickK is searching ($d < d_0$) or settled ($d \geq d_0$). Under exact β (from $\Phi = (=)$), $i \in \beta \iff h_1(u_1)[i] = h_2(u_2)[i]$. The slot c updated by CaseSplit is the charge counter c_{chc} ($c'_{chc} = c_{chc}$ on HIT-equal, $c'_{chc} = c_{chc} + 1$ on HIT-unequal/MISS).

Case 1: $i \in \beta$, searching ($d < d_0$). The logic applies **read1**: since $i \in \beta$, we learn $h_1(u_1)[i] = h_2(u_2)[i]$. $\mathcal{I}(PickK)$ sets $kp = i$. The witness provides $ak_{1p} = ak_{2p}$, so $bkp = 1$. The HIT-equal branch fires: $c' = c$, $d' = d + 1$. The **conditional** (synchronized, grade 0) preserves c because equal array values imply guard agreement (e.g., $(x_1 > 0) = (x_2 > 0)$ when $x_1 = x_2$). Discovery: $d' = d + 1 = |\beta \cap [0, i)| + 1 = |\beta \cap [0, i + 1)|$. ✓

Case 2: $i \notin \beta$, searching ($d < d_0$). The logic applies **read2**: no equality information. $\mathcal{I}(PickK)$ sets $kp = i$ (still searching). Since $i \notin \beta$ and β is exact, $h_1(u_1)[i] \neq h_2(u_2)[i]$, so $bkp = 0$. The HIT-unequal branch fires: $c' = c+1$, $d' = d$. The **conditionalL** (one-sided, grade 1) allows worst-case $c' = c + 1$ (runs may take different branches). Discovery: $d' = d = |\beta \cap [0, i)| = |\beta \cap [0, i + 1)|$ (since $i \notin \beta$). ✓

Case 3: $i \in \beta$, settled ($d \geq d_0$). Under *exact* prophecy ($d_0 = d^*$) this case is unreachable: with d the true count, $d \geq d_0 = |\beta \cap [0, n)|$ and $[0, i) \subseteq [0, n)$ force $\beta \cap [0, i) = \beta \cap [0, n)$, contradicting $i \in \beta$. Under the *threshold* prophecy ($d_0 < d^*$) the case *is* reachable; PickK is settled, so $kp = k$, with two sub-cases:

- $k = i$: the witness reports $bkp = [\, i \in \beta \,] = 1$, so HIT-equal fires ($c' = c$, $d' = d + 1$) and the position *is* credited.
- $k \neq i$: the MISS branch fires ($c' = c + 1$, $d' = d$) and the β-position at i is left *uncredited*, so the credit d under-counts $|\beta \cap [0, i)|$, exactly why $\mathcal{I}(Inv)$ treats d as an abstract credit rather than the true count.

Both sub-cases maintain $0 \leq d' \leq i'$ and $c' = i' - d'$. ✓

Case 4: $i \notin \beta$, settled ($d \geq d_0$). $\mathcal{I}(PickK)$ sets $kp = k$. Two sub-cases:

- If $k = i$: then $kp = i$, and the witness reveals $bkp = 0$ (since $i \notin \beta$, exact). The HIT-unequal branch fires: $c' = c + 1$, $d' = d$. ✓

- If $k \neq i$: the MISS branch fires. $c' = c + 1$, $d' = d$. ✓

In all cases, $\mathit{ConsistBk}(ak_{1p}, ak_{2p}, bk')$ holds: $bk' = [ak_{1p} = ak_{2p}]$ by definition of the witness interpretation. □

Lemma 10.3 (Witness adequacy)**.** *Under $\mathcal{I}$, Clauses 3–5 of $T_R(\mathcal{D})$ are satisfied:*

1. Clause 3 (Witness): *the clause $\mathit{Wit} \leftarrow \mathit{Inv} \wedge i < n$ holds under $\mathcal{I}$ as a* universal inclusion*: since $\mathcal{I}(\mathit{Wit}) = \mathcal{I}(\mathit{Inv})$, every tuple with $\mathcal{I}(\mathit{Inv})(\vec{v})$ (and $i < n$) satisfies $\mathcal{I}(\mathit{Wit})(\vec{v})$. The refocused witnesses (ak_{1p}, ak_{2p}, bkp) that Clause 2 then consumes are arbitrary ConsistBk triples, handled by Transition closure (Lemma 10.4).*
2. Clause 4 (PickK-search): *If $\mathcal{I}(\mathit{Inv})(\vec{v})$, $i < n$, and $d < d_0$, then $\mathcal{I}(\mathit{PickK})(\vec{v}, i)$.*
3. Clause 5 (PickK-settled): *If $\mathcal{I}(\mathit{Inv})(\vec{v})$ and $d \geq d_0$, then $\mathcal{I}(\mathit{PickK})(\vec{v}, k)$.*

Proof. All three follow from the definitions of $\mathcal{I}(\mathit{PickK})$ and $\mathcal{I}(\mathit{Wit})$ together with the fact that $\mathcal{I}(\mathit{Inv})$ is the over-approximation, which admits any *ConsistBk* cell triple. For Clause 3, recall its head cell variables equal its body's; the refocused call $\mathit{Wit}(V_s, kp, ak_{1p}, ak_{2p}, bkp, \ldots)$ raised by Clause 2 unifies to the obligation $\mathit{Inv}(V_s, kp, ak_{1p}, ak_{2p}, bkp, \ldots)$, which holds for any $kp \in [0, n)$ because $\mathcal{I}(\mathit{Inv})$ does not constrain the focus-cell values beyond *ConsistBk*. The intended witness pins $bkp = [\, kp \in \beta \,]$, equivalently the heap values $ak_{1p} = h_1(u_1)[kp]$, $ak_{2p} = h_2(u_2)[kp]$, $bkp = [ak_{1p} = ak_{2p}]$, which exist because $kp \in \{i, k\} \subseteq [0, n)$ and $n \leq \min(\mathit{len}(h_1(u_1)), \mathit{len}(h_2(u_2)))$ (Def 6.2). The search/settled split (Clauses 4–5) depends only on (d, d_0), which is well-defined since β is exact and pure (depends only on array contents and interpretation η, not on mutable scalar state). □

Lemma 10.4 (Invariant preservation)**.** *If $\mathcal{I}(\mathit{Inv})$ holds at iteration i with $i < n$, then after one application of Clause 2 (Transition), $\mathcal{I}(\mathit{Inv})$ holds at iteration $i + 1$.*

Proof. Assume an arbitrary tuple satisfying $\mathcal{I}(\mathit{Inv})$ at iteration i with $i < n$ (not necessarily a concrete *enc*-image; closure must hold for every admitted tuple).

Step 1: CHC obligations. Closure is established purely from the clause relations on the assumed tuple, with no appeal to a concrete source step. By Lemma 10.3, Clauses 3–5 admit a refocus kp and witness (ak_{1p}, ak_{2p}, bkp) (any *ConsistBk* triple, since $\mathcal{I}(\mathit{Wit}) = \mathcal{I}(\mathit{Inv})$); whichever *CaseSplit* branch the witness selects, Step 3 below shows the successor tuple lies in $\mathcal{I}(\mathit{Inv})$.

Step 2: Source-corresponding instance (for inclusion only). When the assumed tuple is a concrete *enc*-image and the witness is source-corresponding, the body premise of the **awhile** rule (soundness of the body sub-derivation, Theorem 8.2) transforms (σ_1, σ_2) to $(\sigma_1', \sigma_2') \vDash_\eta P_{loop} \mid \delta$ at iteration $i + 1$, so the concrete successor is one of the tuples Step 3 places in $\mathcal{I}(\mathit{Inv})$ (supporting Lemma 10.1). This step is *not* needed for closure, which holds for every admitted witness.

Step 3: Post-state tuple lies in $\mathcal{I}(Inv)$ (closure under any *witness).* We check the Clause 2 head tuple $Inv(V'_s, kp, ak_{1p}, ak_{2p}, bk', d', d_0, c'_{chc})$ satisfies the defining conjuncts of $\mathcal{I}(Inv)$ at iteration $i{+}1$, for an *arbitrary ConsistBk* witness (ak_{1p}, ak_{2p}, bkp); no β-correctness of the witness is assumed:

- $0 \le kp < n$: the refocused cell is chosen by *PickK* from $\{i, k\} \subseteq [0, n)$. ✓
- $ConsistBk(ak_{1p}, ak_{2p}, bk')$: enforced by the Clause 2 conjunct. ✓
- $d_0 \ge 0$: unchanged (d_0 is a parameter, untouched by the transition). ✓
- $0 \le i' \le n$: $i' = i + 1$ from *Step*, and $i < n$, so $i' \le n$. ✓
- $0 \le d' \le i'$: HIT-equal sets $d' = d + 1$, and $d \le i$ gives $d' \le i + 1 = i'$; a charged step sets $d' = d \le i \le i'$. In both cases $d' \ge 0$. ✓
- $c'_{chc} = i' - d'$: HIT-equal gives $c'_{chc} = c_{chc} = i - d = (i{+}1) - (d{+}1) = i' - d'$; a charged step gives $c'_{chc} = c_{chc} + 1 = (i - d) + 1 = (i{+}1) - d = i' - d'$. ✓

Thus the head tuple lies in $\mathcal{I}(Inv)$ at $i{+}1$: the invariant is closed under the transition for *every ConsistBk* witness, including mis-crediting ones. The concrete post-state $enc(\sigma'_1, \sigma'_2, \eta, \beta, i{+}1)$ is one such tuple (Lemma 10.1), but closure does not depend on the witness being source-corresponding, which is what repairs the gap where a loose witness produced a successor outside a β-pinned invariant.

Source-corresponding charge (the repair of the original c-conflation). On the source-corresponding path the charge relates to the source difference by an *inequality*, not equality: the earlier draft identified the slot with $\Delta c = c\langle 1\rangle - c\langle 2\rangle$, which is false (at a right-active divergence Δc changes by -1 via **assignR-skipL**, grade 0, while c_{chc} rises by $+1$). The preserved relationship is $\Delta c \le c_{chc}$: HIT-equal leaves both unchanged; a charged step gives $\Delta c' \le \Delta c{+}1 \le c_{chc}{+}1 = c'_{chc}$, using $\Delta c \le c_{chc}$ from P_{loop}. This inequality is what the logic (part (1)) certifies; the CHC closure above is independent of it.

Step 4: Conclude. By Step 3 the Clause 2 head tuple satisfies $\mathcal{I}(Inv)$ at $i + 1$ for every *ConsistBk* focus-cell choice; that is, $\mathcal{I}(Inv)$ is closed under the transition. The logic body step (Step 1) guarantees that the concrete post-state (σ'_1, σ'_2) is itself such a tuple (Lemma 10.1), so the reachable states remain inside the invariant. □

Lemma 10.5 (Charge bound at exit)**.** *For every tuple in $\mathcal{I}(Inv)$ at loop exit, i.e. with $n \le i$ and $d \ge d_0$, the charge satisfies $c_{chc} \le n - d_0$, the CHC threshold bound, so the Goal clause holds under $\mathcal{I}$. (The exact source grade is $[\![m]\!] = n - d^*$ with $d^* = |\beta \cap [0, n)|$; it equals $n - d_0$ only when $d_0 = d^*$, and otherwise $n - d^* \le n - d_0$ weakens to the threshold bound by Corollary 10.7.)*

Proof. Purely algebraic from the invariant. $\mathcal{I}(Inv)$ gives $c_{chc} = i - d$ and $i \le n$; with $n \le i$ this forces $i = n$, so $c_{chc} = n - d$, and $d \ge d_0$ yields $c_{chc} = n - d \le n - d_0$. The bound is *terminal*: it constrains only the exit tuple, and needs no per-step charge-vs-grade comparison and no assumption on witness quality. In particular we do *not* claim that mis-crediting witnesses only lower c_{chc}; a

witness that under-credits a truly equal position can raise c_{chc} on a prefix, but *any* exit tuple in $\mathcal{I}(Inv)$ with $d \geq d_0$ satisfies the bound regardless. On the source-corresponding path the credit reaches $d \geq d_0$ because the precondition supplies $d_0 \leq d^* = |\beta|$ discoverable β-positions; there $\Delta c \leq c_{chc} \leq n - d_0$ recovers the source bound, which part (1) also gives directly. □

Theorem 10.6 (Translation soundness, Level 2 sync)**.** *If $\mathcal{D}$ is a Level 2 derivation of $\vdash \{\delta\} \mid \{P\}\ C_1 \sim C_2\ \{Q\} \mid \{\delta'\}\ \mid m$ in the V1 execution-cost, single-pass lockstep/equality fragment over read-only arrays described above (with CountPositive as the running instance), then:*

1. *By Theorem 8.2, the* graded *relational property $\vDash^m\ \{\delta\} \mid \{P\}\ C_1 \sim C_2\ \{Q\} \mid \{\delta'\}$ holds; in particular the source cost difference satisfies $c\langle 1\rangle - c\langle 2\rangle \leq [\![m]\!]_i(I)$.*
2. *The translated clause system $T_R(\mathcal{D})$ (direct-goal convention; relaxed Init, free $d_0 \geq 0$) is satisfiable: the inductive over-approximation $\mathcal{I}$ defined above is a model, and its Goal clause holds as the CHC threshold bound $c_{chc} \leq n - d_0$ (in terms of the CHC's own free prophecy d_0). Under the precondition threshold $d_0 \leq d^* = |\beta \cap [0, n)|$, this matches the weakened source bound: the derivation grade $[\![m]\!] = n - d^*$ weakens to $n - d_0$ by Corollary 10.7, recovering the source-level bound of part (1).*

Part (2) is an inductive-invariant (over-approximation) satisfiability claim for the CountPositive CaseSplit accounting $c + d = i$; it does not *assert that the CHC charge tracks the exact source grade under arbitrary witnesses, nor CHC-to-source completeness (the converse). The source bound is carried by part (1).*

Proof. Part (1) is Theorem 8.2. For Part (2), we verify each clause of the translated system $T_R(\mathcal{D})$ under $\mathcal{I}$:

Clause 1 (Init): We show every tuple satisfying the relaxed Init body lies in $\mathcal{I}(Inv)$. The body, consisting of $Pre(V_0)$, $0 \leq k < n$, $ConsistBk(ak_1, ak_2, bk)$, and $d_0 \geq 0$, instantiated at loop entry ($i = 0$, hence $d = 0$ and $c_{chc} = 0$) satisfies every defining conjunct of $\mathcal{I}(Inv)$: the cell, range, and $d_0 \geq 0$ constraints are the Init body verbatim; $0 \leq i = 0 \leq n$ (as $n > 0$); $0 \leq d = 0 \leq i$; and $c_{chc} = 0 = i - d$. Since $\mathcal{I}(Inv)$ requires only $d_0 \geq 0$ and admits any *ConsistBk* cell, *every* Init-body tuple is in $\mathcal{I}(Inv)$; thus $\mathcal{I}$ satisfies the shipped relaxed Init directly, with no exact-prophecy specialization. (By Lemma 10.1 the concrete entry state is one such Init tuple, so the reachable states start inside $\mathcal{I}$.)

Clause 2 (Transition): By Lemma 10.4, each iteration preserves $\mathcal{I}(Inv)$. By induction on i from 0 to $n - 1$, $\mathcal{I}(Inv)$ holds at every iteration.

Clauses 3–5 (Witness, PickK): By Lemma 10.3.

Clause 6 (Goal): The Goal clause is $c_{chc} \leq n - d_0 \leftarrow Inv(\ldots) \wedge n \leq i \wedge d \geq d_0$. By the Charge bound (Lemma 10.5), every $\mathcal{I}(Inv)$ tuple with $n \leq i$ and $d \geq d_0$ has $c_{chc} = n - d \leq n - d_0$ (the CHC threshold bound; the exact grade $[\![m]\!] = n - d^*$ weakens to it under $d_0 \leq d^*$), a purely terminal, algebraic consequence of $c_{chc} = i - d$, requiring no witness-quality assumption. So the Goal holds under $\mathcal{I}$. Independently, Part (1) gives the *source* bound $c\langle 1\rangle - c\langle 2\rangle \leq [\![m]\!]$ from

the graded judgment, consistent with the charge on the source-corresponding path via $c\langle 1\rangle - c\langle 2\rangle \leq c_{chc}$ (Lemma 10.4). The CHC certifies that the over-approximation is safe; the graded judgment certifies the source bound. □

Converse direction (CHC → Logic). The converse, that a satisfying CHC assignment yields a logic derivation, requires reconstructing β from the *PickK* assignment and building a derivation tree from the invariant. This is a *completeness* claim and is left as future work. We do *not* claim CHC-to-logic completeness: a satisfying PCSAT assignment certifies the CHC *abstraction* (the charge threshold bound $c_{chc} \leq n - d_0$), not a reconstruction of β or of a derivation tree, and because the charge over-approximates the source difference, a model of the CHC does not by itself exhibit the source-level cost behavior. Connecting a PCSAT result back to the source semantics requires the separate adequacy/simulation argument proved above (the forward direction), not merely the existence of a model. In particular, the forward source bound is established here for the *exhibited* model $\mathcal{I}$ (the source *enc*-images lie in $\mathcal{I}$ by Lemma 10.1, bounded by Lemma 10.5); transferring it to an *arbitrary* solver-found model requires a simulation placing the source-corresponding states in the CHC least model (so that they lie in every model): for the original head-equals-body translation this is unavailable (the least model is not focus-complete), but for the Wit-completion variant T_R^{WC} it is proved as Theorem 10.8 (Section 10.7). This scopes what a bare `sat` verdict certifies on its own; it does not weaken the exhibited-model theorems above.

Scope. This (read-only) translation covers the synchronous Level 2 pattern (single shared array, lockstep iteration). Two extensions are now proved: the *mutable* single-array case as Theorem 10.11 (Section 10.8), via a Clause 2 net cell-update map and an exact-β preservation lemma with the *same* unknown signature (no signature extension needed); and the *two read-only array* case as Theorem 10.16 (Section 10.9), which extends the signature with the second array's cell slots bj_1, bj_2, bj, uses a five-way Clause 2, and carries the inequality charge invariant $c_{chc} \leq 2(i - d)$. The remaining extension, Level 4 async (replace Clauses 2–5 with scheduler-mediated variants SchTT/SchTF/SchFT), follows the same derivation-guided principle with a scheduler-extended signature. The translation as stated also targets a *single* synchronous loop: the logic's **sequence** and **awhile** composition are sound at the judgment level (Theorem 8.2), but a clause-level construction that *composes* or *nests* loop CHC systems (chaining their pre/postconditions and sharing *Inv*) is not given here and is left as future work.

Goal polarity convention. The formal translation uses the *direct* convention: the bound $c \leq n - d_0$ is a constraint-clause head, and the synthesized inductive invariant verifies the property by *satisfiability* (SAT = verified). This is equivalent to the standard refutation/query encoding $\mathit{false} \leftarrow \mathit{Inv}(\ldots) \wedge n \leq i \wedge d \geq d_0 \wedge c > n - d_0$: both assert $\mathit{Inv} \cap \{\, \mathrm{exit}, c > n - d_0 \,\} = \emptyset$. It is *not* inter-

changeable with a *violation-as-head* goal $c > n - d_0 \leftarrow \mathit{Inv}(\ldots) \wedge n \leq i \wedge d \geq d_0$: that clause asserts $\mathit{Inv} \rightarrow (\text{exit} \Rightarrow c > n - d_0)$, i.e. *every* invariant exit state exceeds the bound, a *different* universal obligation from the safety condition $\mathit{Inv} \cap \{\, \text{exit}, c > n - d_0 \,\} = \emptyset$ (and *not* its negation). The direct and query forms are equivalent *unconditionally* (including under predicate synthesis, since the two clauses denote the same implication Body $\Rightarrow c \leq n - d_0$); the violation-as-head form is equivalent to neither. All formal results in this section use the direct (equivalently, query-form) convention; we do not appeal to a blanket direct/violation equivalence.

Threshold prophecy as a corollary (alignment with the `.clp` encodings). The `.clp` encodings in `encodings/level-2/` use the *threshold* convention: the encoding's d_0 is a free variable, and the source-level precondition guarantees only $|\beta \cap [0, n)| \geq d_0$. The source-level threshold bound follows from the *logic* (part (1)) by weakening, independently of the CHC model:

Corollary 10.7 (Threshold prophecy soundness)**.** *Let $d^* = |\beta \cap [0, n)|$ be the exact count, and let t be a threshold with $0 \leq t \leq d^*$ (the encoding's free prophecy d_0, for which the source-level precondition guarantees $d^* \geq t$). Then the source cost difference satisfies the threshold bound $c\langle 1\rangle - c\langle 2\rangle \leq n - t$, which has the same* numeric shape *as the translated CHC goal $c_{chc} \leq n - t$.*

Proof. By Theorem 10.6 part (1) (the graded judgment), the derivation gives the tight source bound $c\langle 1\rangle - c\langle 2\rangle \leq [\![m]\!] = n - d^*$. Since $t \leq d^*$, i.e. $-d^* \leq -t$, we obtain $c\langle 1\rangle - c\langle 2\rangle \leq n - d^* \leq n - t$. □

This corollary matches the *numeric shape* of the encoding pattern: PCSAT chooses $t = d_0$ as a free variable, the source-level precondition ensures the threshold $d^* \geq t$ is met, and the bound $c\langle 1\rangle - c\langle 2\rangle \leq n - t$ holds with possible slack $d^* - t$ between the tight source bound $n - d^*$ and the threshold bound $n - t$. We do *not* claim that the CHC verification *simulates* the source semantics clause-for-clause, nor CHC-to-source completeness: the threshold bound is a source-level weakening from part (1), while part (2) separately certifies that the translated CHC system has an inductive model (so PCSAT's check is sound on its own terms).

10.7 Forward Soundness: Any Solver Model Verifies the Source Bound

Theorem 10.6 establishes part (2) by exhibiting a specific over-approximation model $\mathcal{I}$. The automation claim wants more: that *any* satisfying assignment PCSAT returns certifies the source bound, not merely that some model exists. This subsection proves that, after one refocus-completing change to the Witness clause.

The focus-availability obstruction. The standard CHC route is: a `sat` certificate is a model M; the least model $\mathcal{L}$ (the CHC-reachable states) is contained in every M; so if the source *enc*-images lie in $\mathcal{L}$, the Goal clause bounds them in every M. The gap is at the premise "*enc*-images lie in $\mathcal{L}$". With the head-equals-body Witness clause (Clause 3), Clause 2's body call $\mathit{Wit}(V_s, kp, \ldots)$ demands $\mathit{Inv}(V_s, kp, \ldots)$ at the *same* iteration. While searching, *PickK* sets $kp = i$, so a step at iteration i needs *Inv* focused at i at iteration i; but the transition into iteration i produces a focus in $\{i-1, k\}$ (the previous kp), never i. Hence $\mathcal{L}$ is *not* focus-complete for $i \geq 1$, and the searching refocus is unavailable in the least model. The exhibited $\mathcal{I}$ escapes this only because it admits *every* *ConsistBk* focus triple (its defining clause constrains d, c, d_0 and the ranges, not the focus contents), which is precisely why Theorem 10.6 is an exhibited-model statement and not yet an arbitrary-model one.

Wit-completion. We close the gap by strengthening Clause 3 to the *Wit-completion* form (WC), which supplies the refocus premise directly:

$$\mathit{Wit}(V_s, kp, ak_{1p}, ak_{2p}, bkp, d, d_0, c) \leftarrow \mathit{Inv}(V_s, k, ak_1, ak_2, bk, d, d_0, c) \wedge i < n \wedge 0 \leq kp < n \wedge \mathit{ConsistBk}(ak_{1p}, a$$

Given *Inv* at any current focus at iteration i, (WC) makes *Wit* hold at every cell $kp \in [0, n)$ with any consistent value triple, at the same (i, d, d_0, c). Write T_R^{WC} for the translation variant in which Clause 3 is *replaced* by (WC) (the head-equals-body and (WC) Witness clauses do not coexist). Two facts make this the right change. First, the exhibited model still works for the variant: since $\mathcal{I}(\mathit{Wit}) = \mathcal{I}(\mathit{Inv})$ already admits every *ConsistBk* triple at every focus, $\mathcal{I} \vDash$ (WC), so re-proving the Clause 3 case of Lemma 10.3 for (WC) (the other clauses are unchanged) shows $\mathcal{I}$ models T_R^{WC}, i.e. Theorem 10.6 part (2) carries over to the variant. Second, the least model becomes focus-complete: whenever $\mathcal{L}$ holds *Inv* at some focus at iteration i, (WC) places *Wit* at every $kp \in [0, n)$ at iteration i into $\mathcal{L}$, so the Clause 2 refocus premise is always available. Crucially, (WC) adds tuples only to *Wit*, which the Goal clause does not range over, so the Goal obligation on *Inv* is unchanged. As corroboration (not part of the proof), on the tested read-only CountPositive encoding (WC) preserves the full probe quad (positive $c \leq n - d_0$ `sat`; violation $c > n - d_0$ `unsat`; vacuity `unsat` at $n = 2$ for both $d_0 = 0$ and $d_0 = 1$), evidence that on that encoding (WC) neither breaks the bound nor makes the terminal unreachable.

Theorem 10.8 (Forward soundness, read-only Level 2, direct-goal convention)**.** *Let $\mathcal{D}$ be an in-scope read-only Level 2 derivation (the fragment of Theorem 10.6); let $T_R^{\mathrm{WC}}(\mathcal{D})$ be its translation with Clause 3 in the Wit-completion form (WC) and the* direct *Goal $c_{chc} \leq n - d_0$; and suppose the threshold precondition $0 \leq d_0 \leq d^* = |\beta \cap [0, n)|$ holds. Then for* every *model M of $T_R^{\mathrm{WC}}(\mathcal{D})$, the source signed cost difference at loop exit satisfies $c\langle 1\rangle - c\langle 2\rangle \leq n - d_0$.*

Proof. Let $\mathcal{L}$ be the least model of the definite clauses (Init, Transition, Witness, PickK), and fix the threshold $t = d_0$ with $0 \leq t \leq d^*$. The *source-corresponding trajectory at t* is the family of *enc*-images carrying prophecy slot $d_0 = t$ and

credit slot d equal to the observed-credit count ($|\beta \cap [0, i)|$ while searching, an observed subcount once settled), as in the abstract-credit reading of Section 10.6 and Lemma 10.1; this is the threshold embedding, not the exact-prophecy *enc* that pins $d_0 = d^*$.

1. *Init adequacy.* The trajectory's initial image ($i = 0$, $d = 0$, $d_0 = t$, $c = 0$) satisfies the Init body (Lemma 10.1), so it lies in $\mathcal{L}$.

2. *Refocus availability.* At iteration i with the source *Inv*-image in $\mathcal{L}$, (WC) derives $Wit(V_s, kp, h_1(u_1)[kp], h_2(u_2)[kp], [kp \in \beta], d, d_0, c) \in \mathcal{L}$ for the kp chosen by *PickK* ($kp \in \{i, k\} \subseteq [0, n)$, and the source cell values form a *ConsistBk* triple). This is exactly the premise the head-equals-body clause could not supply in $\mathcal{L}$.

3. *Step simulation.* With the refocus available, the source body step matches the corresponding Clause 2 branch by the rule-local analysis (Lemma 10.2): HIT-equal, HIT-unequal, or MISS on ($i \in \beta$?, searching/settled), with the source-corresponding witness $bkp = [kp \in \beta]$. The successor *enc*-image therefore lies in $\mathcal{L}$, and by induction the entire source-corresponding trajectory is in $\mathcal{L}$.

4. *Discovery progress.* Under $d_0 \leq d^*$, the search phase credits d up to d_0 before $[0, n)$ is exhausted (there are $d^* \geq d_0$ discoverable β-positions), so the trajectory reaches the exit $i = n$ with $d \geq d_0$.

5. *Charge dominance.* On the source-corresponding trajectory, $\Delta c = c\langle 1\rangle - c\langle 2\rangle \leq c_{chc}$ (Notation paragraph of Section 10.6), an inequality maintained step-by-step and independent of the CHC model.

6. *Model transfer.* M satisfies all definite clauses, so $\mathcal{L} \subseteq M$. The source exit *enc*-image (in $\mathcal{L}$ by steps 1–4, with $n \leq i$ and $d \geq d_0$) thus lies in $M(Inv)$ and satisfies the Goal clause of M, giving $c_{chc} \leq n - d_0$. With step 5, $\Delta c \leq c_{chc} \leq n - d_0$.

The bound holds for an *arbitrary* model M, so a PCSAT `sat` verdict on $T_R^{\mathrm{WC}}(\mathcal{D})$ certifies the source bound itself, not merely the existence of a model. As in Theorem 10.6 this is the direct-goal convention; CHC-to-logic completeness is still not claimed. □

Scope of forward soundness. Theorem 10.8 is stated for the read-only fragment with CountPositive as the running instance. The mutable and two read-only array cases carry over by the same six-step argument once Clause 3 is placed in the (WC) form matching their signatures; the model-transfer step is identical, and only the refocus-premise and step-simulation bookkeeping change. We record them as Corollary 10.12 (after Theorem 10.11) and Corollary 10.17 (after Theorem 10.16).

10.8 Formal CHC Translation: Mutable Single Array

Theorem 10.6 proves the translation for *read-only* arrays. We now extend it to *in-place mutation* of a single array, with **InplaceMap** (Section 8.11) as the running instance. The extension reuses the read-only unknown signature, the over-approximation $\mathcal{I}$, the closure lemma (Lemma 10.4), and the goal lemma (Lemma 10.5) *unchanged*. The only new ingredients are an exact-β preservation lemma (under an injectivity hypothesis) and a current-index write-frame, both confined to the *source-corresponding* (inclusion) direction. We deliberately keep the over-approximation closure (O1, arbitrary witnesses) separate from the source-adequacy argument (O2, source-corresponding witnesses), as in Theorem 10.6.

Setting and hypotheses. We keep the read-only setting of Section 10.6: a single shared array, a single-pass lockstep loop with $\Phi = (=)$, and the *exact* index-agreement set taken over the *initial* heaps,

$$\beta_0 = \{j \in [0, n) : h_1^{(0)}(u_1)[j] = h_2^{(0)}(u_2)[j]\},$$

which is the same exact-β representation choice fixed in the Scope-and-assumptions paragraph of Section 10.6. Let $\tau : \mathbb{Z} \to \mathbb{Z}$ denote the *net per-cell transformation* the loop body applies to a cell, $\tau(x) = \mathit{guard}(x)\,?\,\mathit{upd}(x) : x$ (the value the cell holds after the iteration that processes it). We add two hypotheses, both discharged by InplaceMap:

(H1) Current-index write. The body reads $x := A[i]$ and performs at most one array write, to the *current* cell, of the form $A[i] := \mathit{upd}(x)$ guarded by $\mathit{guard}(x)$ (and `skip` otherwise). No cell other than $A[i]$ is written in iteration i, so $h^{(i+1)}(u)[j] = h^{(i)}(u)[j]$ for every $j \neq i$.

(H2) Injective net update. τ is injective on the reachable cell-value domain (global injectivity is a simple sufficient condition). Equivalently τ is equality-preserving ($x_1 = x_2 \Rightarrow \tau(x_1) = \tau(x_2)$, automatic for a function) and non-equality-creating ($x_1 \neq x_2 \Rightarrow \tau(x_1) \neq \tau(x_2)$).

For InplaceMap, $\mathit{guard}(x) = (x > 0)$ and the if-branch writes $x + 1$, so $\tau(x) = (x > 0\,?\,x + 1 : x)$. This τ is injective over $\mathbb{Z}$ (it maps $x \leq 0$ into $(-\infty, 0]$ by the identity and $x > 0$ into $[2, \infty)$ by $x \mapsto x + 1$, each branch injective with disjoint images), so (H2) holds; and each cell is written only at its own index, so (H1) holds.

Clause 2 (mutable transition). The read-only Clause 2 passes the witnessed cell values through unchanged ($ak_1' = ak_{1p}$, $ak_2' = ak_{2p}$). The mutable Clause 2 instead folds the net update τ into the focused cell:

$$\begin{aligned}
\text{(HIT-equal)} \quad & i = kp \wedge bkp = 1 \wedge ak_1' = \tau(ak_{1p}) \wedge ak_2' = ak_1' \wedge c' = c \wedge d' = d + 1,\\
\text{(HIT-unequal)} \quad & i = kp \wedge bkp = 0 \wedge ak_1' = \tau(ak_{1p}) \wedge ak_2' = \tau(ak_{2p}) \wedge c' = c + 1 \wedge d' = d,\\
\text{(MISS)} \quad & i \neq kp \wedge ak_1' = ak_{1p} \wedge ak_2' = ak_{2p} \wedge c' = c + 1 \wedge d' = d,
\end{aligned}$$

followed, exactly as in the read-only clause, by $\mathit{ConsistBk}(ak'_1, ak'_2, bk')$ pinning $bk' = [ak'_1 = ak'_2]$ on the post-write values. The cost/credit updates (c', d') are *identical* to the read-only clause; only the cell-value slots change. On HIT-equal, $ak_{1p} = ak_{2p}$ gives $\tau(ak_{1p}) = \tau(ak_{2p})$, so writing $ak'_2 = ak'_1$ is sound. On MISS the focus is the settled cell $k \neq i$, which (H1) leaves untouched, so the pass-through $ak'_r = ak_{rp}$ is the true post-state value of that cell.

What is inherited unchanged. The unknown signature $(\mathit{Inv}, \mathit{PickK}, \mathit{Wit})$, Init, Witness, PickK, and the Goal clause are identical to the read-only system. Two of the three soundness tracks carry over with no new content:

- *Closure (O1).* Lemma 10.4 establishes closure of $\mathcal{I}(\mathit{Inv})$ under Clause 2 for an *arbitrary ConsistBk* witness, checking only the range conjuncts, $\mathit{ConsistBk}(ak'_1, ak'_2, bk')$, and the algebra $0 \leq d' \leq i'$, $c'_{chc} = i' - d'$ (its Step 3). None of these inspect *how* ak'_1, ak'_2 are produced; they require only that the clause re-pin bk' via *ConsistBk* (it does) and leave c, d, i as before (they are). Hence Lemma 10.4 applies to the mutable Clause 2 unchanged: the over-approximation is insensitive to the cell-update map.
- *Goal (O3).* Lemma 10.5 is purely algebraic ($c_{chc} = i - d$ at exit gives $c_{chc} \leq n - d_0$) and never refers to cell values; it applies verbatim.

Only the *inclusion* track (concrete states lie inside $\mathcal{I}$, supporting Lemmas 10.1 and 10.2) needs new content, supplied by the two lemmas below.

Lemma 10.9 (Exact-β preservation under injective current-index writes)**.** *Under (H1) and (H2), the exact agreement set is invariant across the loop: writing* $\beta^{(i)} = \{j \in [0, n) : h_1^{(i)}(u_1)[j] = h_2^{(i)}(u_2)[j]\}$*, we have* $\beta^{(i)} = \beta_0$ *for every* $i \in [0, n]$.

Proof. By induction on i. By (H1) the only cell modified between $h^{(i)}$ and $h^{(i+1)}$ is index i, so $\beta^{(i+1)}$ and $\beta^{(i)}$ can differ only at $j = i$; it suffices to show the write at i preserves the agreement status of i. Both runs apply the *same* net transformation τ to their own cell at index i (each run takes its own branch, but the resulting cell value is τ of its own input). If $i \in \beta^{(i)}$, then $h_1^{(i)}(u_1)[i] = h_2^{(i)}(u_2)[i]$, so the two post-values are τ of equal inputs, hence equal (the must-half; this is the fixed-δ congruence argument of the InplaceMap derivation, Section 8.11). If $i \notin \beta^{(i)}$, the two inputs differ, so the two post-values are τ of distinct inputs, hence distinct by injectivity (H2) (the converse-half, which the must-set binary property does *not* supply on its own; Definition 6.2 fixes only the must-direction). Either way the status of i is unchanged, so $\beta^{(i+1)} = \beta^{(i)} = \beta_0$. □

Lemma 10.10 (Mutable inclusion adequacy)**.** *Under (H1) and (H2), define the mutable state encoding* $\mathit{enc}^{(i)}$ *exactly as in Section 10.6 but reading cell values from the* current *heap* $h^{(i)}$ *and using the fixed exact set* β_0 *(Lemma 10.9). Then (i)* $\mathcal{I}(\mathit{Inv})(\mathit{enc}^{(i)})$ *holds (Lemma 10.1 carries over), and (ii) the source-corresponding witness and the Clause 2 head agree with the concrete post-state*

heap $h^{(i+1)}$: on HIT ($kp = i$) the focused (written) cell becomes $\tau(ak_{rp})$, and on MISS ($kp = k \neq i$) the focused cell is unchanged, matching $ak'_r = ak_{rp}$.

Proof. (i) The defining conjuncts of $\mathcal{I}(Inv)$ are the range constraints, *ConsistBk*, and the credit algebra; the concrete tuple at iteration i satisfies all of them as in Lemma 10.1, since β_0 is exact and fixed (Lemma 10.9) and $bk = [ak_1 = ak_2]$ by construction. (ii) On HIT the refocus is $kp = i$, the written index, and the source step sets that cell to τ of its read value, matching the HIT branches' $ak'_r = \tau(ak_{rp})$. On MISS the refocus is the settled cell $k \neq i$; by (H1) the iteration-i write does not touch index k, so $h^{(i+1)}(u_r)[k] = h^{(i)}(u_r)[k]$, matching $ak'_r = ak_{rp}$. The rule-local case analysis (Lemma 10.2, cases $i \in \beta / i \notin \beta$ against searching/settled, including the exactness premise and the settled $k = i$ subcase) then applies with $\beta := \beta_0$, its exactness discharged by Lemma 10.9. □

Theorem 10.11 (Translation soundness, Level 2 mutable single-array)**.** *Let $\mathcal{D}$ be a Level 2 derivation of $\vdash \{\delta\} \mid \{P\}\ C_1 \sim C_2\ \{Q\} \mid \{\delta'\} \mid m$ in the V1 execution-cost, single-pass lockstep/equality fragment over a* single in-place mutated *array satisfying (H1) and (H2), with InplaceMap as the running instance. Then:*

1. *By Theorem 8.2, the graded relational property $\vDash^m \{\delta\} \mid \{P\}\ C_1 \sim C_2\ \{Q\} \mid \{\delta'\}$ holds; in particular $c\langle 1\rangle - c\langle 2\rangle \leq [\![m]\!]_i(I)$.*
2. *The translated clause system $T_R(\mathcal{D})$ (direct-goal convention) with the mutable Clause 2 is satisfiable: the same inductive over-approximation $\mathcal{I}$ as in Theorem 10.6 is a model, and its Goal clause holds as the threshold bound $c_{chc} \leq n - d_0$.*

As in Theorem 10.6, part (2) is an over-approximation satisfiability claim, not a CHC-to-source completeness claim; the source bound is carried by part (1).

Proof. Part (1) is Theorem 8.2, whose execution-cost scope already includes InplaceMap (Section 8). For part (2) we verify each clause under $\mathcal{I}$. Init, Witness, PickK, and the Goal are the read-only clauses, satisfied as in Theorem 10.6. Closure of $\mathcal{I}(Inv)$ under the mutable Clause 2 is Lemma 10.4 (insensitive to the cell-update map, as noted above), and the Goal bound is Lemma 10.5 (algebraic). That the reachable concrete states remain inside $\mathcal{I}$, so the model is non-vacuous and the source-corresponding accounting holds, is Lemma 10.10, whose exactness premise is supplied by Lemma 10.9 under (H1) and (H2). The over-approximation closure (O1, arbitrary witnesses) and the source-adequacy argument (O2, source-corresponding witnesses) thereby stay separate. □

Corollary 10.12 (Forward soundness, mutable single array)**.** *Let $\mathcal{D}$ be an in-scope mutable Level 2 derivation (the fragment of Theorem 10.11, satisfying (H1) and (H2)), and let $T_R^{\mathrm{WC}}(\mathcal{D})$ be its translation with the (WC) Witness clause and the direct Goal. Then for every model M of $T_R^{\mathrm{WC}}(\mathcal{D})$, the source bound $c\langle 1\rangle - c\langle 2\rangle \leq n - d_0$ holds at loop exit (under $0 \leq d_0 \leq d^* = |\beta_0 \cap [0, n)|$, with β_0 the initial agreement set of Theorem 10.11).*

Proof. The six steps of Theorem 10.8 apply; only refocus availability (step 2) and step simulation (step 3) change. (WC) supplies *Wit* at the *current* (pre-update) cell values $h^{(i)}(u)[kp]$, and the mutable Clause 2 folds the net update τ into its *head* ($ak'_r = \tau(ak_{rp})$), so the successor *enc*-image carries the post-write values exactly as in Lemma 10.10. Exact-β preservation (Lemma 10.9, under (H1) and (H2)) keeps the agreement set fixed across the loop, so discovery progress (step 4) and charge dominance (step 5) are unchanged. Model transfer (step 6) is identical, since the Goal ranges over *Inv* and (WC) adds only *Wit* facts. □

Scope of the mutable extension. Theorem 10.11 covers single-array, current-index, injective in-place updates, i.e. (H1) and (H2). Three boundaries are explicit. (a) A *non-injective* or equality-*creating* net update (e.g. a constant map $\tau(x) = 0$) violates (H2): the agreement set then grows rather than staying fixed, which is the **awhile-evolving** regime (Section 8); such maps still admit the weak bound $n - d_0$ only via a must-set variant with a redesigned source-correspondence (the witness bit then denoting actual equality rather than β_0-membership), which we do not develop here. (b) A *shifted* or multi-cell write (e.g. $A[i+1] := \cdots$, compaction) violates (H1) and breaks the MISS frame, requiring an explicit write-index comparison; out of scope. (c) Two-array mutation requires the heap-frame machinery of the (read-only) two-array pattern composed with this cell-update map; we leave it as the next extension. As in Theorem 10.6, the converse (CHC-to-logic completeness) is not claimed.

10.9 Formal CHC Translation: Two Read-Only Arrays

We extend Theorem 10.6 from one array to *two read-only* arrays, with **Dual-Count** (Section 8.12) as the running instance. Unlike the mutable extension (which kept the unknown signature and reused the closure and goal lemmas verbatim), the two-array case *extends* the unknown signature with the second array's cell slots, and *changes* the cost algebra from the single-array equality $c_{chc} = i - d$ to the inequality $c_{chc} \le 2(i - d)$. What is reused is the *method*: the two-phase *PickK*, the *Wit* clause shape, the abstract-credit reading of d, and the over-approximation proof structure. The two arrays are read-only, so this is independent of the mutable extension.

Setting and signature. Two read-only shared arrays A, B, a single-pass lockstep loop with $\Phi = (=)$, and *one shared* distinguished index k observed in both arrays. We use exact agreement sets $\beta_A = \{j \in [0, n) : h_1(A_1)[j] = h_2(A_2)[j]\}$ and $\beta_B = \{j \in [0, n) : h_1(B_1)[j] = h_2(B_2)[j]\}$ (the per-array exact-β choice of Section 10.6, applied to each array), a single abstract credit d for the simultaneous-agreement set $\beta_A \cap \beta_B$, and a single threshold d_0 with $d_0 \le |\beta_A \cap \beta_B \cap [0, n)|$. The unknown signature extends the single-array one (Section 10.6) with the B-cell slots bj_1, bj_2, bj:

$$Inv(V_s, k, ak_1, ak_2, bk, bj_1, bj_2, bj, d, d_0, c), \quad PickK(\ldots, kp), \quad Wit(\ldots),$$

where (ak_1, ak_2, bk) is the A-cell triple at k and (bj_1, bj_2, bj) the B-cell triple at the *same* k, with $bk = [ak_1 = ak_2]$, $bj = [bj_1 = bj_2]$. As in the single-array case, c is the CHC *charge counter* c_{chc}, over-approximating the source signed difference $\Delta c = c\langle 1\rangle - c\langle 2\rangle$ ($\Delta c \le c_{chc}$, see the Notation paragraph of Section 10.6), and d is an abstract credit, equal to $|\beta_A \cap \beta_B \cap [0, i)|$ only on the source-corresponding search path.

Clause 2 (two-array transition). The CaseSplit is five-way, over the two bits at the focused cell plus MISS:

$$
\begin{array}{lll}
i = kp \wedge bkp = 1 \wedge bjp = 1 \wedge c' = c \quad \wedge d' = d+1 & & \text{(both agree: credit)} \\
i = kp \wedge bkp = 1 \wedge bjp = 0 \wedge c' = c+1 \wedge d' = d & & (A \text{ agree, } B \text{ differ}) \\
i = kp \wedge bkp = 0 \wedge bjp = 1 \wedge c' = c+1 \wedge d' = d & & (A \text{ differ, } B \text{ agree}) \\
i = kp \wedge bkp = 0 \wedge bjp = 0 \wedge c' = c+2 \wedge d' = d & & \text{(both differ)} \\
i \neq kp \quad \wedge c' = c+2 \wedge d' = d & & \text{(MISS)}
\end{array}
$$

followed by $\mathit{ConsistBk}(ak_1', ak_2', bk')$ and the B-analogue $\mathit{ConsistBj}(bj_1', bj_2', bj')$ pinning the two bits. The arrays are read-only, so the cell values pass through ($ak_r' = ak_{rp}$, $bj_r' = bj_{rp}$). The credit d rises only when *both* bits are 1; each disagreeing array contributes at most 1 to the charge, so a step costs 0, 1, or 2.

Over-approximation. The model is the single-array $\mathcal{I}$ of Section 10.6 with the B-triple added and the credit relation *loosened to an inequality*:

$$
\begin{aligned}
\mathcal{I}(\mathit{Inv})(V_s, k, ak_1, ak_2, bk, bj_1, bj_2, bj, d, d_0, c) \iff\ & 0 \le k < n \wedge \mathit{ConsistBk}(ak_1, ak_2, bk) \wedge \mathit{ConsistBj}(bj_1, bj_2, bj) \\
& \wedge\ d_0 \ge 0 \wedge 0 \le i \le n \wedge 0 \le d \le i \wedge c \le 2(i-d).
\end{aligned}
$$

The single-array equality $c = i - d$ becomes the inequality $c \le 2(i-d)$: a one-disagreement step adds only 1 to c but the bound charges 2, so equality cannot hold (and is unnecessary; the loose bound $2(n-d_0)$ is what the encoding proves, matching the logic's worst-case grade 2 per asymmetric position, Section 8.12). $\mathcal{I}(\mathit{PickK})$ and $\mathcal{I}(\mathit{Wit})$ are as in the single-array case, carrying the B-slots.

Heap-frame correspondence. On the logic side the two arrays are composed by the separating conjunction $\delta_A \star \delta_B$ and reasoned about via **heap-frame** (Section 8.12, around the heap-frame rule): one reasons about the A-counter under δ_A while framing δ_B, symmetrically for B, and the worst-case per-position grade 2 is **sequence**-rule additivity of the two unit counters $(1 + 1)$, not a K=2-specific rule. The CHC does *not* replicate this framing: it observes both arrays at the single cell k with two independent bits and *adds* the disagreement costs. We therefore prove CHC satisfiability *directly* via $\mathcal{I}(\mathit{Inv})$ (Theorem 10.16 part (2)), and let heap-frame remain a *part-(1) rule* discharged by Theorem 8.2 in the DualCount derivation; we do *not* translate it into a CHC clause. The one place the separation is consumed on the CHC side is the inclusion lemma below: the $\star$ *disjointness* of the two owned array fragments (the $\star$ heap-split,

Definition 6.2 and the $\star$ clause of validity) is what licenses treating bk and bj as *independent* simultaneous observations, so that the per-array disagreement costs add. (Were A and B aliased, the two bits would not be independent and the additive accounting would be unjustified.) The remaining heap-frame side conditions used in the *source* derivation, namely footprint, locality, read-only preservation, and pure β-membership, are discharged there (Section 8.12) and are not needed by the CHC argument.

Lemma 10.13 (Two-array closure)**.** *$\mathcal{I}(Inv)$ is closed under the two-array Clause 2 for every pair of ConsistBk/ConsistBj witnesses.*

Proof. Take any tuple in $\mathcal{I}(Inv)$ at iteration $i < n$. The head re-pins bk', bj' via *ConsistBk*/*ConsistBj*, so those conjuncts hold; $0 \le kp < n$ since $kp \in \{i, k\} \subseteq [0, n)$; $i' = i+1 \le n$; and $d' \in \{d, d+1\} \subseteq [0, i+1] = [0, i']$. It remains to check $c' \le 2(i'-d')$ on each branch, using $c \le 2(i-d)$ and $i' = i+1$: *both agree* ($c' = c$, $d' = d+1$): $c' = c \le 2(i-d) = 2((i+1)-(d+1)) = 2(i'-d')$. *one disagreement* ($c' = c+1$, $d' = d$): $c' = c+1 \le 2(i-d)+1 \le 2(i-d)+2 = 2(i'-d')$. *both differ / MISS* ($c' = c+2$, $d' = d$): $c' = c+2 \le 2(i-d)+2 = 2(i'-d')$. In all cases the head tuple lies in $\mathcal{I}(Inv)$ at $i+1$. As in the single-array case, closure is witness-agnostic. □

Lemma 10.14 (Two-array charge bound at exit)**.** *For every $\mathcal{I}(Inv)$ tuple at loop exit ($n \le i$, $d \ge d_0$), the charge satisfies $c_{chc} \le 2(n - d_0)$.*

Proof. Algebraic, paralleling Lemma 10.5. From $0 \le i \le n$ and $n \le i$ we get $i = n$, so $c_{chc} \le 2(n-d)$; with $d \ge d_0$ this gives $c_{chc} \le 2(n-d) \le 2(n-d_0)$. Terminal and witness-quality-independent. □

Lemma 10.15 (Two-array inclusion)**.** *With enc encoding both arrays' cells at k and using the exact sets β_A, β_B, the concrete states satisfy $\mathcal{I}(Inv)$, and the source-corresponding rule-local analysis carries over.*

Proof. The defining conjuncts of $\mathcal{I}(Inv)$ are the ranges, the two *ConsistBk*/*ConsistBj* predicates, and $c \le 2(i-d)$; the concrete tuple satisfies them as in Lemma 10.1, with $bk = [ak_1{=}ak_2]$, $bj = [bj_1{=}bj_2]$ by construction. The $\star$ disjointness of A and B (heap-frame correspondence above) makes bk and bj independent observations. Rule-local mirrors the single-array proof (Lemma 10.2), now over the four membership combinations ($i \in \beta_A?, i \in \beta_B?$) against the three *PickK* subcases: *searching* ($kp = i$), *settled* $k = i$, and *settled* $k \ne i$ (MISS). The credit fires ($d' = d+1$) exactly on a both-agree observation ($i \in \beta_A \cap \beta_B$ while searching, or settled $k = i$ both-agree); a settled $k \ne i$ step MISSes and leaves a both-agree position at i *uncredited*, so d is an abstract subcount, $d \le |\beta_A \cap \beta_B \cap [0, i)|$ in general (exactly the single-array abstract-credit treatment). MISS is load-bearing in the settled phase. Exactness of *both* β_A and β_B is used, per array. □

Theorem 10.16 (Translation soundness, Level 2 two read-only arrays)**.** *Let $\mathcal{D}$ be a Level 2 derivation of $\vdash \{\delta\} \mid \{P\}\ C_1 \sim C_2\ \{Q\} \mid \{\delta'\}$ $\mid m$ in the V1 execution-cost, single-pass lockstep/equality fragment over* two read-only *shared*

arrays observed at one common index, with DualCount as the running instance and threshold $d_0 \le |\beta_A \cap \beta_B \cap [0,n)|$. Then:

1. *By Theorem 8.2 (the DualCount derivation uses **heap-frame**, $\star$, and **sequence**), the graded property holds; in particular $c\langle 1\rangle - c\langle 2\rangle \le [\![m]\!]_i(I)$. For the canonical DualCount grade $[\![m]\!] = \sum_{i=0}^{n-1}(\chi_{i\notin\beta_A} + \chi_{i\notin\beta_B}) \le 2\big(n - |\beta_A \cap \beta_B \cap [0,n)|\big)$ (Section 8.12), this weakens to $c\langle 1\rangle - c\langle 2\rangle \le 2(n-d_0)$ under the threshold $d_0 \le |\beta_A \cap \beta_B \cap [0,n)|$, paralleling Corollary 10.7. (As in the single-array case, the chain to the numeric bound is specific to this canonical indicator-sum grade, not an arbitrary **conseq**-weakened grade.)*
2. *The translated clause system $T_R(\mathcal{D})$ (direct-goal convention) with the two-array signature and the five-way Clause 2 is satisfiable: $\mathcal{I}$ above is a model, and its Goal clause holds as $c_{chc} \le 2(n-d_0)$.*

As in Theorem 10.6, part (2) is an over-approximation satisfiability claim, not CHC-to-logic completeness; the source bound is carried by part (1).

Proof. Part (1) is Theorem 8.2 applied to the DualCount derivation (Section 8.12), whose heap-frame, $\star$, and sequence steps yield the canonical indicator-sum grade $\sum_i m_i$, numerically bounded by $2(n-d_0)$ under the threshold (Section 8.12); $2(n-d_0)$ with free d_0 is a numeric upper bound, not a supplied grade. For part (2): Init instantiates $\mathcal{I}(Inv)$ at $i=0$ ($d=0$, $c=0 \le 2(i-d)$, both bits *ConsistBk*/*ConsistBj*); Clause 2 preserves it by Lemma 10.13; Witness and PickK hold as in the single-array case carrying the B-slots; and the Goal is Lemma 10.14. That the reachable concrete states lie in $\mathcal{I}$ is Lemma 10.15. Closure (witness-agnostic) and inclusion (source-corresponding) are kept separate, as in Theorem 10.6. □

Corollary 10.17 (Forward soundness, two read-only arrays)**.** *Let $\mathcal{D}$ be an in-scope two-array Level 2 derivation (the fragment of Theorem 10.16), and let $T_R^{\mathrm{WC}}(\mathcal{D})$ be its translation with the two-array Witness clause in the (WC) form carrying both ConsistBk and ConsistBj, and the direct Goal $c_{chc} \le 2(n-d_0)$. Then for every model M of $T_R^{\mathrm{WC}}(\mathcal{D})$, the source bound $c\langle 1\rangle - c\langle 2\rangle \le 2(n-d_0)$ holds at loop exit (under $0 \le d_0 \le |\beta_A \cap \beta_B \cap [0,n)|$).*

Proof. The six steps apply with the two-array signature. Refocus availability (step 2): the two-array (WC) supplies *Wit* at any $kp \in [0,n)$ with an A-cell triple (*ConsistBk*) and a B-cell triple (*ConsistBj*), so the source-corresponding refocus (the actual A- and B-cell values at kp) lies in the least model. Step simulation (step 3) uses the five-way Clause 2 (Lemma 10.15). Discovery progress (step 4) credits d on simultaneous-agreement positions ($\beta_A \cap \beta_B$); under $d_0 \le |\beta_A \cap \beta_B \cap [0,n)|$ the credit reaches d_0 at exit. Charge dominance (step 5) is the two-array source-to-CHC bridge $\Delta c \le c_{chc}$ (the five-way cost accounting of the two-array signature paragraph, each disagreeing array contributing at most 1). Model transfer (step 6): $\mathcal{L} \subseteq M$, so the source exit image lies in $M(Inv)$ and satisfies M's direct Goal $c_{chc} \le 2(n-d_0)$; with step 5, $\Delta c \le c_{chc} \le 2(n-d_0)$. (Lemma 10.14 is the algebraic exit bound for the *exhibited* model $\mathcal{I}$, not the arbitrary-model transfer step.) □

Scope of the two-array extension. Theorem 10.16 covers *two read-only* arrays observed at one common index, with a single credit and threshold for simultaneous agreement, bound $2(n - d_0)$. The K-counter generalization (Section 8.12) gives the source bound $\Delta c \leq K(n - d_0)$ by the same **sequence**-additivity, with *QuadCount* (K=4) as a corpus instance; we state the theorem at K=2. Boundaries: *mutable* two-array would compose this with the cell-update map τ of Theorem 10.11 (out of scope); *distinct per-array indices* (e.g. A at i, B at $j \neq i$) would need a second distinguished cell (out of scope); *per-array thresholds* d_0^A, d_0^B would tighten the bound below $2(n - d_0)$ but are not pursued. As in Theorem 10.6, the converse (CHC-to-logic completeness) is not claimed.

11 Level 4: Certified Async Relational Logic with Arrays

Level 4 is the async extension of Level 2 for loops that make progress at different paces. It is not a new grade algebra. Level 2 already supplies the local reasoning principles: paired body steps, left-only catch-up steps, and right-only catch-up steps. Level 4 adds one wrapper around those local proofs: an admissible scheduler certificate that constructs a coupled trace for the ordinary terminating executions considered by the judgment, checks that the trace projects exactly back to those executions, records concrete shared-agreement credit, accounts for one-sided catch-up, and establishes the terminal cost bound.

Motivating example: stride 1v2. The simplest place where Level 2 is not enough is the following pair of scans:

```
Run 1: for i1 = 0 to n-1:
           if A1[i1] > 0: c1++
Run 2: for i2 = 0 to n-1 step 2:
           if A2[i2] > 0: c2++
```

Run 1 visits $0, 1, 2, 3, \ldots$, while Run 2 visits only $0, 2, 4, \ldots$. The shared aligned positions are the even indices. At an even index k, both runs can step together and compare $A_1[k]$ with $A_2[k]$. After that paired step, Run 2 has advanced by two positions, while Run 1 has advanced by one, so Run 1 must take one catch-up step at the intervening odd index before the runs are aligned again.

Level 2 supplies exactly the local reasoning one expects for these source steps. A paired local triple proves that a `TT` body step at an aligned even index preserves the relational invariant. A one-sided local triple proves that a `TF` body step, where Run 1 alone processes the odd catch-up position, also preserves the invariant and updates the signed cost ledger. These facts are necessary, but they are still only facts about one selected body step. By themselves they do not package a whole-loop schedule, do not prove that the scheduled steps erase back to the two ordinary executions, and do not record which equal even positions are allowed to pay for the d_0 discount.

The Level 4 scheduler certificate is the missing global object. In this example it can be read as a finite-state policy over the current loop counters, credit counter, and credit set, written informally as (i_1, i_2, d, G):

1. If both loops are done, choose `done`.
2. If $i_1 = i_2 < n$ and $d < d_0$, choose `TT_search`: both runs step at the aligned even index. If the focused cells are equal and the index is fresh, insert that index into G and increase d.
3. If $i_1 = i_2 < n$ and $d \geq d_0$, choose `TT_settled`: both runs step together, but no new discount credit is claimed.

4. If $i_1 < i_2$ and $i_1 < n$, choose `TF`: Run 1 processes the intervening catch-up index while Run 2 stutters. For the 1v2 schedule this is the only one-sided catch-up mode that occurs; `FT` is needed by more general stride pairs.

Iterating this policy produces the trace

$$\texttt{TT\_search},\ \texttt{TF},\ \texttt{TT\_search},\ \texttt{TF}, \ldots$$

until the credit threshold is reached, followed by the same paired/catch-up shape in the settled phase. The certificate is useful because it carries the three global checks that the local Level 2 triples do not provide on their own. First, exact projection says that erasing the coupled trace gives precisely the ordinary stride-1 execution on the left and the ordinary stride-2 execution on the right, so the proof is about the real program pair rather than a convenient invented trace. Second, the credit set G contains only fresh aligned even indices where the concrete focused cells were HIT-equal, so the discount is paid for by real shared events and cannot be counted twice. Third, the terminal check combines the invariant and credit evidence: at exit, $i_1 = n$, $c \leq i_1 - |G|$, and $|G| \geq d_0$, hence

$$c \leq n - d_0.$$

This is the mental model for the rest of the section: Level 2 proves the local body triples, and Level 4 adds the certificate that schedules those triples, projects them back to the two concrete executions, records distinct credit, and checks the terminal bound.

Reader path. The section first states the judgment and the source-facing AWHILE-ASYNC soundness principle. The principle should be read as local Level 2-style body reasoning plus one admissible scheduler certificate. It then works the stride examples to show what the certificate does. The certificate kernel follows after the examples: coupled traces, obligations grouped by purpose, witnesses, and Theorem 11.4. The final part explains how CHC encodings define and check such certificates.

Part A: Soundness Principle and Examples

11.1 The Level-4 Judgment

The Level 4 judgment used in Theorems 11.1–11.3 and in the soundness theorem of Section 11.7 is fixed as follows.

Definition 11.1 (Relational validity, Level 4)**.** *Fix a cost model M assigning a weight in $\mathbb{N}$ to each primitive execution event (the per-construct parameterization of Section 3, applied per occurrence; the default weights and branch-counting are both instances), such that the top-level while-guard evaluations weigh* 0*. This makes M* iteration-additive*: a terminating run's cost is exactly the sum of its*

iterations' costs. Let Q be a relational postcondition that may additionally mention the cost observables m_1, m_2. Then

$$\models_\nu \{P\}\ \texttt{while}\, b_1\ \texttt{do}\ C_1 \sim \texttt{while}\, b_2\ \texttt{do}\ C_2\, \{Q\}\ \mid\ \iota$$

holds iff for every well-formed interpretation I (with $dom(I) \supseteq LV(P,Q)$, as in Definition 8.1) and every pair of finite terminating cost-instrumented runs $(\sigma_1, \sigma_1', m_1) \in [\![\, \texttt{while}\, b_1\ \texttt{do}\ C_1]\!]$ and $(\sigma_2, \sigma_2', m_2) \in [\![\, \texttt{while}\, b_2\ \texttt{do}\ C_2]\!]$ under M: if $(\sigma_1, \sigma_2) \vDash_I P$ then $(\sigma_1', \sigma_2') \vDash_I Q$, where the observables m_1, m_2 in Q denote the two runs' costs.

The two programs execute independently, exactly as at Levels 1–2. The coupled trace and scheduler introduced below are proof machinery, not a new operational semantics. Relative to the Level 2 graded judgment (Definition 8.1), the cost observables move into the postcondition; the judgment is parametric in the iteration-additive cost model M (branch-counting for the stride instances, with the default execution-cost weights also qualifying); and the slot ι records the certificate's declared re-pacing budget. The slot ι has *no* validity clause. It is scheduler bookkeeping, bounded by the index-accounting obligation, and is not the signed source cost. For the threshold bounds used in this section, the Level 2 grade sublanguage excludes free-d_0 terms (Lemma 8.1), so a bound such as $\lceil n/a \rceil - d_0$ cannot ride the grade slot; it belongs in the postcondition.

What the certificate wrapper does. Definition 11.1 fixes where the cost bound lives, namely in Q. The scheduler certificate explains how that bound is proved. It does not prove source termination; it is checked against the finite terminating executions considered by the judgment. At each coupled step it selects one mode among `TT_search`, `TT_settled`, `TF`, `FT`, and `done`. Iterating those choices produces a coupled trace. The certificate is accepted only when the trace projects exactly to the two concrete executions with the same terminal states and cost ledgers, uses only causal observations, is finite and joint-terminal, preserves the phase invariant, records distinct aligned HIT-equal events sufficient for the required d_0 credit, accounts for one-sided catch-up steps through ι, and satisfies the terminal verification condition. Thus the source-facing principle below has two layers: local body triples justify each selected source step, while the certificate wrapper justifies the interleaving and the discounted terminal bound.

11.2 The Source-Facing awhile-async Soundness Principle

Level 4 exposes one source-facing soundness principle. The four behaviors are paired progress, left catch-up, right catch-up, and terminal acceptance. The displayed body triples are local Level 2-style preservation facts over the loop invariant I. The only genuinely new premise is that S is an admissible scheduler certificate for the invariant, the terminal postcondition, the entry-fixed threshold d_0, the catch-up budget ι, and the bound B. Part B discharges this premise with the kernel-side package $\mathsf{Adm} \wedge \mathsf{O1\text{-}O10/KC}$. The display is not a syntactic $\vdash$-rule: its conclusion is the semantic judgment $\models_\nu$. It is a source-facing presentation

of the soundness theorem, not a new semantic judgment and not a replacement for the kernel theorem. A future syntactic Level 4 derivability system could internalize this principle as a $\vdash_{\mathrm{L4}}$ proof rule; this section proves only the semantic soundness direction. The side-condition package also carries the initial and terminal connections: the initial P-states satisfy the certificate invariant, and accepted terminal traces satisfying that invariant yield Q and the cost bound.

$$\frac{\begin{array}{c}\{I \wedge b_1\langle 1\rangle \wedge b_2\langle 2\rangle\}\ C_1 \sim C_2\ \{I\} \quad (\texttt{TT}) \\ \{I \wedge b_1\langle 1\rangle\}\ C_1 \sim \texttt{skip}\ \{I\} \quad (\texttt{TF}) \\ \{I \wedge b_2\langle 2\rangle\}\ \texttt{skip} \sim C_2\ \{I\} \quad (\texttt{FT}) \\ S \text{ is an admissible scheduler certificate for } (P, Q, I, d_0, \iota, B)\end{array}}{\models_\nu\ \{P\}\ \texttt{while}\, b_1\ \texttt{do}\ C_1 \sim \texttt{while}\, b_2\ \texttt{do}\ C_2\ \{Q \wedge c \le B(C_1, C_2, n, d_0)\} \mid \iota}\ \textbf{awhile-async}$$

The single displayed `TT` body premise is refined by the scheduler into two kernel modes. This compression is presentation-only; the kernel still distinguishes `TT_search` from `TT_settled`. `TT_search` is the phase in which the certificate searches for fresh shared-agreement credit, while `TT_settled` is paired progress after the entry-fixed threshold has been met. This split is a feature of the asynchronous scheduler, not of the prophecy itself: a lockstep Level 2 loop may carry the same d_0 promise, but it steps both runs at one shared index, so there is no scheduler phase to distinguish. The phase policy ($d < d_0$ versus $d \ge d_0$), the distinct fresh HIT-equal credit claims, and the threshold/accounting proof are discharged by the admissibility side condition, concretely by the forcing clauses, O8, and O9 in Section 11.6. They are not postconditions of the displayed body triples.

Part B explains why this readable principle is sound. The kernel uses phase-indexed invariants, a mode-step relation, and the obligations Adm∧O1-O10/KC; Theorem 11.4 is the unchanged soundness theorem that discharges the principle above.

11.3 Scheduler Certificate by Example: Stride-1v2

This example shows the wrapper contract before the formal kernel. The local body triples are simple: paired steps preserve the invariant, left catch-up preserves it while possibly increasing the signed ledger by one, and right catch-up cannot increase the left-minus-right ledger. The scheduler certificate is what decides when those local facts are used and why the final discount is valid.

Programs. Run 1 is the stride-1 scan ($i_1 = 0, 1, \ldots, n-1$). Run 2 is the stride-2 scan ($i_2 = 0, 2, 4, \ldots$ below n). Both arrays are read-only, and the cost model is branch-counting, with $m_1 - m_2$ written as c inside the invariant.

Precondition and target. The precondition P_{1v2} says $n > 0$, $0 \le d_0 \le \lceil n/2 \rceil$, and at least d_0 even positions $p < n$ satisfy $A_1[p] = A_2[p]$. The desired postcondition is

$$Q_{1v2} \wedge m_1 - m_2 \le n - d_0.$$

Invariant intuition. Let d be the number of credited fresh equal even positions discovered so far. The useful scalar part of the invariant is

$$0 \le i_1 \le n \wedge 0 \le i_2 \le n+1 \wedge c \le i_1 - d.$$

For odd n, the stride-2 run may overshoot from $n-1$ to $n+1$; the cost bound depends only on i_1 and d, so that overshoot is harmless.

Scheduler modes. `TT_search` is used when both runs are at the same even position and the certificate has not yet justified the threshold. If the focused cells are equal and the position is fresh, both branches agree and the certificate may add one credit. If they are unequal, no credit is added and the signed cost can increase by at most one; the inequality $c \le i_1 - d$ is still preserved because i_1 also advances.

`TF` catches run 1 up after a paired stride-2 step leaves it behind. It processes the intervening odd position alone, so c may increase by at most one and i_1 increases by one.

`TT_settled` is paired progress after the certificate has met the entry-fixed credit threshold. It preserves the invariant without claiming new credit. The terminal proof uses the fact, supplied by the admissibility obligations, that the settled phase is reached only after enough justified credits have been found.

`FT` lets run 2 step alone when it is behind. This cannot increase $m_1 - m_2$, so it is harmless for the upper bound.

Termination. At exit, the invariant gives $i_1 = n$ and $c \le n - d$. The admissibility obligations ensure $d \ge d_0$ at terminal traces, hence $m_1 - m_2 = c \le n - d_0$.

CHC correspondence.

Logic role	CHC clause family	Purpose
`TT_search` body	TT search clause	Pick focused aligned cells
`TT_settled` body	TT settled clause	Paired progress after threshold
`TF` body	TF clause	Left catch-up
`FT` body	FT clause	Right catch-up
Alignment forcing	realignment constraint	Return to paired positions
Progress	progress clauses	Finite terminal trace

11.4 Proved Stride Instances

For strides a, b, let $E = lcm(a, b)$, $\alpha = E/a$, and $\beta = E/b$. The stride certificate $\sigma_{\alpha,\beta}$ repeats a bounded realignment pattern inside each epoch: paired search at a shared point, lagging-pointer catch-up, and paired progress after the threshold is met. The proved general bound pattern is

$$B(C_1, C_2, n, d_0) = \left\lceil \frac{n}{a} \right\rceil - d_0,$$

where d_0 counts jointly aligned shared agreements, not all positions visited by run 1.

The three stride results below are proved unconditionally for the branch-counting cost model. All three proofs are *by direct construction*: the concrete scheduler certificate is built directly from the concrete run pair, with the focused-cell values set to the concrete array cells at each `TT_search`, so the over-approximation cannot manufacture agreement credit. They are hand proofs, not mechanically checked. They do *not* use the general encoding-to-logic bridge of Section 11.8 and make no solver-model soundness claim: a satisfying PCSAT model is corroborating evidence only.

Theorem 11.1 (1v2 relational cost; branch-counting; unconditional)**.** *Let C_1 be the stride-1 scan and C_2 the stride-2 scan over read-only arrays A_1, A_2, with branch-counting costs m_1, m_2. Assume $P_{1v2}(A_1, A_2, n, d_0)$: $n > 0$; d_0 is a natural number with $0 \le d_0 \le \lceil n/2 \rceil$ (the count of shared even positions below n); and at least d_0 even positions $p < n$ satisfy $A_1[p] = A_2[p]$. Then*

$$\models_\nu \{P_{1v2}\}\ \texttt{while}\, b_1\ \texttt{do}\ C_1 \sim \texttt{while}\, b_2\ \texttt{do}\ C_2\ \{Q_{1v2} \wedge m_1 - m_2 \le n - d_0\}\ \mid\ \iota_{1v2}, \qquad \iota_{1v2} = n + \lceil n/2 \rceil.$$

Proof (direct construction). v1 proves the local lemmas (`LocalScalar_1v2`, `GhostShape_1v2`, `Coverage_1v2`); v2 discharges `BridgeReal_1v2` (exact projection, focus realization, and whole-shadow coherence) and the derived cost and credit-fulfillment targets. Full hand proof: `popl/L4_1V2_V1_LOCAL_PROOFS_2026-06-09.md` and `popl/L4_BRIDGE_1V2_2026-06-09.md`. □

Theorem 11.2 (2v3 relational cost; branch-counting; unconditional)**.** *Let C_1 be the stride-2 scan and C_2 the stride-3 scan over read-only arrays A_1, A_2, with branch-counting costs m_1, m_2. Assume $P_{2v3}(A_1, A_2, n, d_0)$: $n > 0$; d_0 is a natural number with $0 \le d_0 \le \lceil n/6 \rceil$ (the count of shared positions below n, namely the multiples of 6); and at least d_0 multiples of 6, $p < n$, satisfy $A_1[p] = A_2[p]$. Then*

$$\models_\nu \{P_{2v3}\}\ \texttt{while}\, b_1\ \texttt{do}\ C_1 \sim \texttt{while}\, b_2\ \texttt{do}\ C_2\ \{Q_{2v3} \wedge m_1 - m_2 \le \lceil n/2 \rceil - d_0\}\ \mid\ \iota_{2v3}, \qquad \iota_{2v3} = \lceil n/2 \rceil + \lceil n/3 \rceil$$

Proof (consolidated direct construction). The local lemmas (bidirectional shape using both `TF` and `FT` catch-up, coverage, and the ghost-set shape) and the bridge are established by direct construction; the no-double-credit fact cites the distinct-credit lemma. Full hand proof: `popl/L4_2V3_SOUNDNESS_2026-06-09.md`, citing `popl/L4_2V3_DISTINCT_CREDIT_LEMMA_2026-06-09.md`. □

Theorem 11.3 (General stride relational cost; branch-counting; unconditional)**.** *Let $1 \le a < b$, $E = lcm(a, b)$, $\alpha = E/a$, $\beta = E/b$ (so $\gcd(\alpha, \beta) = 1$); let C_1 be the stride-a scan and C_2 the stride-b scan over read-only arrays A_1, A_2, with branch-counting costs m_1, m_2. Assume $P_{\text{stride}}(A_1, A_2, n, d_0)$: $n > 0$; d_0 a natural number with $0 \le d_0 \le \lceil n/E \rceil$ (the count of shared positions below n, the multiples of E); and at least d_0 multiples of E, $p < n$, satisfy $A_1[p] = A_2[p]$. Then*

$$\models_\nu \{P_{\text{stride}}\}\ \texttt{while}\, b_1\ \texttt{do}\ C_1 \sim \texttt{while}\, b_2\ \texttt{do}\ C_2\ \{Q \wedge m_1 - m_2 \le \lceil n/a \rceil - d_0\}\ \mid\ \iota, \qquad \iota = \lceil n/a \rceil + \lceil n/b \rceil.$$

Theorems 11.1 and 11.2 are the instances $a{=}1, b{=}2$ *and* $a{=}2, b{=}3$.

Proof (direct construction). The local lemmas (invariant $a(c{+}d) \le i_1$, coprimality alignment, the lagging-pointer merge with bounded realignment per search-phase epoch, scan, distinctness $d \le \lceil n/E \rceil$, and coverage) and the bridge are established by direct construction, generalizing the 1v2/2v3 arguments with a, b, E substituted. Full hand proof: `popl/L4_GENERAL_STRIDE_SOUNDNESS_2026-06-09.md`. □

These are the three proved stride results (1v2, 2v3, and the general corollary). They are re-derived as *instantiations* of the general AWhile-Product rule (Theorem 11.4; companion artifact, Section 6): the stride certificate discharges the kernel obligations, and the rule yields exactly the statements above.

Level 2 vs. Level 4.

	Level 2 (sync)	**Level 4 (async)**
Counters	One shared loop counter	Two concrete loop counters
Guard agreement	Required by lockstep	Mediated by the scheduler
Scheduler	None	`TT_search`, `TT_settled`, `TF`, `FT`
Alignment	Every body step	Only certified paired steps
Credit source	Local relational body fact	Fresh focused shared agreements
Cost accounting	Grade slot	Postcondition bound over m_1, m_2
awhile rule	One synchronous body rule	Source rule plus certificate soundness kernel

What ARel cannot express. ARel's product rule types each program independently, yielding a bound of the form $c_1 - c_2 \le U_1 - L_2$. For stride 1v2, this gives the trivial upper bound $U_1 = n$ for run 1 and the trivial lower bound $L_2 = 0$ for run 2, hence $c_1 - c_2 \le n$. The Level 4 certificate can use concrete agreement at shared even positions, so it proves the strictly tighter bound $c_1 - c_2 \le n - d_0$ when $d_0 > 0$.

Part B: Why the Rule Is Sound

11.5 Scheduler Certificates

A Level 4 scheduler certificate has the form

$$S = (\Phi, \phi_0, \mathit{mode}, \mathit{next}, \mathit{enabled})$$

over phases Φ and causally available product observations. The mode function ranges over

$$\{\texttt{TT\_search}, \texttt{TT\_settled}, \texttt{TF}, \texttt{FT}, \texttt{done}\}.$$

The certificate is accompanied by a phase-indexed invariant I, a ghost set G of credited shared positions, an entry-fixed prophecy d_0, a scheduler-index bound ι, a terminal cost-bound parameter $B(C_1, C_2, n, d_0)$, and a declared *credit domain*

SA. The domain SA is the certificate-fixed set of jointly aligned positions where agreement credit may be earned. For stride instances, SA is the set of multiples of $E = lcm(a, b)$ below n. The precondition promises at least d_0 agreeing positions in SA; credit is earned only at fresh positions of SA, one credit per position. Well-formedness requires $0 \le d_0 \le |SA|$ and requires B to be defined and nonnegative on that range.

Scheduler control is causal. The functions *mode* and *next* are total functions of entry-fixed certificate parameters, phase, and the current observation state: current scalar stores, current guard values, the derived credit count $d = |G|$, and logged array observations from already consumed body runs or focused cells. The predicate *enabled* is the guard column of the mode-step table in Section 11.5. The entry-fixed threshold comparison $d < d_0$ is permitted, just as comparisons with n are permitted; scheduler control may not depend on the future cells that realize the d_0 promise.

The following table relates certificate modes to the pfwCSP encoding predicates. These predicates are templates for candidate certificates extracted from encodings; a satisfying PCSAT model is not by itself a concrete Level 4 proof.

Kernel mode	**Encoding predicate**	**Concrete movement**	**Role**
`TT_search`	`SchTT_search`	both runs step	aligned comparison, possible credit
`TT_settled`	`SchTT_settled`	both runs step	post-search paired progress
`TF`	`SchTF`	left steps, right stutters	left catch-up
`FT`	`SchFT`	right steps, left stutters	right catch-up

Coupled traces and mode steps. Fix two finite terminating concrete executions

$$\rho_1 = ((C_1, s_1) \Rightarrow^{m_1} s_1') \qquad \rho_2 = ((C_2, s_2) \Rightarrow^{m_2} s_2').$$

Because the top-level while guards have weight 0, each run decomposes into its body-iteration derivations, and the terminal cost is the sum of those iteration costs. A product configuration is $(\sigma_1, \sigma_2, \phi, G, l_1, l_2)$, where l_1, l_2 are the concrete cost ledgers consumed so far; the signed ledger is $c = l_1 - l_2$.

A *coupled S-trace* for ρ_1, ρ_2 is a sequence of mode steps prescribed by S from the initial product state. A *candidate finite coupled S-trace* is a finite such sequence. Candidate traces are neutral objects: they need not yet be accepted, need not yet be known to project to the two concrete executions, and need not yet establish the terminal postcondition. The *produced* trace is the candidate obtained by iterating S against the fixed concrete runs until the scheduler reaches `done`, if it does.

The mode-step semantics used by the obligations is the following table.

Mode	Guard	Step
`TT_search`	$b_1(\sigma_1) \wedge b_2(\sigma_2)$	Both body runs step, with ledgers $l_1 + w_1, l_2 + w_2$; on a fresh HIT-equal focus observation, $G' = G \cup \{k\}$, otherwise $G' = G$.
`TT_settled`	$b_1(\sigma_1) \wedge b_2(\sigma_2)$	Both step as above; $G' = G$.
`TF`	$b_1(\sigma_1)$	Left body run only; ledger $l_1 + w_1$; right side stutters, whether b_2 is true or false.
`FT`	$b_2(\sigma_2)$	Symmetric: right body run only; ledger $l_2 + w_2$; left side stutters.
`done`	$\neg b_1(\sigma_1) \wedge \neg b_2(\sigma_2)$	Joint terminal; no source step.

The two `TT` mode guards are only the two loop guards. For stride certificates, an additional condition such as $i_1 = i_2$ is scheduler-selection policy, enforced by the certificate's mode/next choices and by the forcing clauses of the encoding bridge; it is not part of the generic mode guard.

11.6 Kernel Obligations

The obligations below, together with the well-formedness package Adm, are the formal expansion of "S admissible." They are easier to read by purpose first:

Purpose	Obligations	What the reader should track
Well-formed certificate	Adm	The program pair, invariant, credit domain, threshold, scheduler index, and bound are in scope.
Projection	O1	The coupled trace erases to exactly the two ordinary runs, with the same terminal states and costs.
Scheduler control	O2, O3, O4, O8	The scheduler is defined when needed, uses only causal observations, and selects enabled modes with permitted prophecy use.
Invariant preservation	O5	Each selected mode preserves the phase-indexed invariant.
Accepted produced trace	O6, KC	O6 gives the finite joint-terminal produced trace; KC connects that trace to the per-trace checks.
Credit and catch-up accounting	O9, O10	Distinct HIT-equal events justify the d_0 credit, and one-sided steps are bounded by ι.
Terminal bound	O7	The accepted terminal witness implies Q and $c \leq B$.

O1, O3, O4, O5, O8, O9, and O10 are predicates on a candidate finite coupled trace. O2, O6, and KC are global obligations about the trace actually produced by S. O7 is a terminal verification condition applied after a terminal accepted witness has been built.

O1: exact projection. The stutter-erasure of τ is exactly ρ_1, ρ_2, including the same initial states, terminal states, and costs. Under an iteration-additive cost model (Definition 11.1) the trace's two cost ledgers at the joint terminal therefore equal m_1, m_2; O1 is what transfers the certificate's terminal cost relation to the concrete runs (companion artifact, Section 2).

O2: applicability / non-stuckness. Along the produced coupled run, S is defined and selects an enabled transition until a joint terminal state is reached. This obligation does not by itself assert finiteness or terminal acceptance.

O3: causal choices. The choices of *mode*, *next*, and *enabled* use only causally available observations. They may not branch on future array values, future branch outcomes, or the future-dependent realization of d_0.

O4: enabledness. Every mode used in τ fires only when its guard in Section 11.5's mode-step table holds.

O5: invariant preservation. The initial product state satisfies $I(\phi_0, \sigma_1, \sigma_2)$, and every mode step preserves the phase-indexed invariant at $\mathit{next}(\phi, \mathit{obs})$, including mixed-terminal tails.

O6: finite joint terminality. For every finite terminating concrete run pair covered by the certificate assumptions, the produced coupled trace is finite and ends in a joint terminal state. The intended proof uses productivity and bounded realignment of the certificate, not a well-founded source-termination measure and not an unspecified liveness assumption.

O7: terminal adequacy. At the joint terminal of a candidate trace, the invariant facts, the ghost evidence G with $|G| \geq d_0$, and that trace's index accounting $\#\mathsf{TF} + \#\mathsf{FT} \leq \iota$ imply the postcondition Q and the bound $c \leq B(C_1, C_2, n, d_0)$. The witness alone does not establish this; O7 applies the terminal verification condition to the witness.

O8: PickK / prophecy. Focus-cell choices are realizable in the concrete run and use only present information. The prophecy d_0 is entry-fixed, may appear in I, O7, O9, and O10, and may not be used by scheduler control as future knowledge. Entry-fixed *threshold comparisons* (such as $d < d_0$) in scheduler control are permitted: d_0 is an instantiation parameter, like n; what is excluded is any dependence on the future cells that realize d_0. This matches bridge condition A5 and corrects the stricter wording of the kernel spec (companion artifact, Section 1.4).

O9: distinct shared-credit evidence. Whenever the certificate claims a d_0-discounted bound, the trace carries a ghost set G of credited positions with $|G| \geq d_0$. Every credited position is justified by a fresh concrete `TT_search` HIT-equal event at a position in the declared credit domain SA visited by both runs; hence $|G| \leq |SA|$. MISS cases, HIT-unequal cases, and unfocused summary facts do not credit G.

O10: index accounting. The produced trace satisfies $\#\mathsf{TF} + \#\mathsf{FT} \leq \iota$.

KC: kernel coverage of the produced trace. For each finite terminating run pair in the certificate-local scope, the trace produced by S satisfies the per-trace obligations O1, O3, O4, O5, O8, O9, and O10. Together with O2 and O6, KC makes the produced trace a terminal accepted witness. KC is a proof obligation for the certificate, not a claim that every solver model or every possible program pair is covered.

The adequacy package $\mathsf{SchedOK\_adeq}$. Write

$$\mathsf{SchedOK\_adeq}(P, Q, S, I, G, d_0, \iota, B) \;:=\; \mathsf{Adm} \wedge \mathsf{O1}, \mathsf{O2}, \mathsf{O3}, \mathsf{O6}, \mathsf{O7}, \mathsf{O8}, \mathsf{O9}, \mathsf{O10}, \mathsf{KC}$$

for the kernel obligations other than the mode-local O4/O5 preservation checks, when the kernel proof is factored that way. Equivalently, the source-facing side condition "S admissible" in Section 11.2 names the full burden $\mathsf{Adm} \wedge$ O1-O10/KC. At the kernel level,

$$(\text{mode-local O4/O5 VCs}) \wedge \mathsf{SchedOK\_adeq} = \mathsf{Adm} \wedge \text{O1-O10/KC};$$

this is a proof-inventory equality, not a claim that the source-facing body triples are those VCs. The split is presentational, not a disjoint factorization: KC continues to require the produced trace to satisfy O1, O3, O4, O5, O8, O9, O10, so the mode-local O4/O5 content is also checked inside KC for the produced trace; no obligation is dropped, added, or restated.

11.7 Witnesses and the AWhile-Product Soundness Theorem

A *terminal accepted witness* for ρ_1, ρ_2 is a candidate finite coupled S-trace τ that ends in a joint terminal state $\neg b_1 \wedge \neg b_2$ and satisfies the per-trace obligations O1, O3, O4, O5, O8, O9, and O10. The terminal verification condition O7 is deliberately not part of the witness. A witness gives the concrete trace evidence; O7 is the separate certificate-level check that turns that evidence into Q and $c \leq B$.

The proof factors per concrete run pair. Lemma B produces the witness from a certificate that is non-stuck, productive, and covered by the kernel. Lemma A then converts that witness, plus the terminal verification condition, into the postcondition and the cost bound. Theorem 11.4 composes these two facts.

Lemma A: accepted-trace adequacy. ***(Proved.)*** For an admissible interpretation, suppose a scheduler certificate S yields a terminal accepted witness τ for a finite terminating run pair ρ_1, ρ_2. If the terminal verification condition O7 holds for the certificate, then the terminal states satisfy Q and the signed cost relation satisfies $c \leq B(C_1, C_2, n, d_0)$.

Lemma B: certificate-local witness production. ***(Proved.)*** For an admissible interpretation and a fixed scheduler certificate S, if the global obligations O2, O6, and KC hold for the finite terminating run pairs under consideration, then the coupled trace produced by S for each such pair is a terminal accepted witness for that same pair. This is not a semantic completeness statement for the logic or the encoding; it is the certificate-local existence part of the proof.

Composition. The Level 4 soundness direction, discharged by Theorem 11.4, is Lemma B followed by Lemma A: obligations satisfied by a concrete scheduler certificate imply the relational judgment for the finite terminating run pairs covered by that certificate. The direction is only

$$\{\text{concrete Level 4 certificates satisfying O1–O10/KC}\} \subseteq \{\text{terminating-run relational validity with } Q \wedge c \leq B\}.$$

No converse direction and no proof-search completeness claim is made.

The AWhile-Product Soundness Theorem. This subsection states the Level 4 kernel soundness theorem, now *proved* (Theorem 11.4). The full hand proof, over fully formal definitions of the coupled semantics, observation states, candidate and produced traces, witnesses, and obligations, is the companion artifact `popl/L4_AWHILE_PRODUCT_RULE_2026-06-09.md`; this subsection states the result and keeps the obligation inventory it certifies.

Relation to the awhile-async principle. The soundness principle in Section 11.2 is the readable, source-facing presentation: its local body triples describe the proof obligations one writes for the loop bodies, while the side condition "S admissible" names the certificate burden made precise in Section 11.6. The formal object actually proved is the theorem below, whose hypotheses are the kernel obligations $\mathsf{Adm} \wedge \mathsf{O1\text{-}O10/KC}$. Thus the principle is discharged by this unchanged kernel theorem, not re-proved as a separate syntactic rule.

Statement. Let $\mathsf{Adm}(P, Q, S, I, G, d_0, \iota, B)$ mean that the program pair, certificate (including its declared credit domain SA), invariant, ghost evidence interface, prophecy, scheduler index, and bound parameter satisfy the well-formedness conditions in Sections 11.5–11.7, with $0 \leq d_0 \leq |SA|$, B defined and nonnegative on that range, and P entailing d_0-realizability over SA (every P-pair has at least d_0 agreeing positions in SA). Because the validity judgment (Definition 11.1) quantifies over all well-formed interpretations while admissibility is a property of one interpretation, the rule reads Adm and the obligations as a *universal* premise: they hold at every well-formed interpretation under which P is satisfiable; interpretations falsifying P contribute vacuously to validity. Instances discharge this by P-entailment (P_{stride} entails the d_0 range and

realizability), which is why Theorems 11.1–11.3 hold unrestricted. The certified async-product soundness theorem is:

$$\frac{\begin{array}{c}\mathsf{Adm}(P, Q, S, I, G, d_0, \iota, B) \\ \mathsf{O1\text{-}O10/KC}(P, Q, S, I, G, d_0, \iota, B)\end{array}}{\models_\nu \{P\}\ \texttt{while}\, b_1\ \texttt{do}\ C_1 \sim\ \texttt{while}\, b_2\ \texttt{do}\ C_2\ \{Q \wedge c \leq B(C_1, C_2, n, d_0)\} \mid \iota}$$

Theorem 11.4 (AWhile-Product; general; ν reading)**.** *Let* `while` b_1 `do` $C_1 \sim$ `while` b_2 `do` C_2 *be an in-scope pair (single-pass top-level loops with loop-free bodies) under an iteration-additive cost model M (Definition 11.1). If, at every well-formed interpretation under which P is satisfiable,* $\mathsf{Adm}(P, Q, S, I, G, d_0, \iota, B)$ *holds and the certificate discharges the obligations* O1-O10/KC *of Section 11.6, then*

$$\models_\nu \{P\}\ \texttt{while}\, b_1\ \texttt{do}\ C_1 \sim\ \texttt{while}\, b_2\ \texttt{do}\ C_2\ \{Q \wedge c \leq B(C_1, C_2, n, d_0)\} \mid \iota,$$

with $c = m_1 - m_2$ the postcondition cost observable (Definition 11.1).

Proof (companion artifact, Sections 3–5). Fix a well-formed interpretation; if P is unsatisfiable under it, validity there is vacuous; otherwise the universal premise supplies admissibility and the obligations at it. Lemma B (kernel coverage): O2, O6, and KC make the trace that S produces against each finite terminating P-pair a terminal accepted witness for that same pair. Lemma A (accepted-trace adequacy): the witness packages the terminal invariant (O5), the ghost evidence $|G| \geq d_0$ (O9), and the index accounting (O10); the terminal verification condition O7 then yields Q and $c \leq B$ at the trace terminal, and exact projection (O1) transfers the terminal states and cost ledgers to the concrete runs. Composing the two lemmas per run pair and unfolding Definition 11.1 gives the judgment. The proof consumes only the loop-free-bodies half of the scope hypothesis; single-pass structure is stated for scope fidelity and never used. □

The source-facing AWHILE-ASYNC soundness principle of Section 11.2 is therefore discharged by this theorem. The premise O1-O10/KC abbreviates the obligations in Section 11.6: the per-trace obligations O1, O3, O4, O5, O8, O9, O10 are packaged inside KC for the produced traces; O2 and O6 are global; O7 is the certificate-level terminal verification condition. The conclusion is the partial-correctness judgment over finite terminating run pairs (Definition 11.1). No completeness or converse direction is claimed.

Part C: Bridge and Status

11.8 Encoding Bridge

The current automation corpus is pfwCSP/PCSAT. The Level 4 encodings synthesize candidate scheduler/evidence witnesses, including predicates such as `SchTT_search`, `SchTT_settled`, `SchTF`, and `SchFT`. These results are useful automation evidence, but a solver model does not automatically become a concrete Level 4 certificate.

The bridge is the admissibility/concretization theorem from encoded witnesses to concrete certificates. Its statement is:

$$\{\text{bridge-admissible cell-morphed pfwCSP/PCSAT witnesses}\}$$
$$\subseteq$$
$$\{\text{concrete Level 4 certificates satisfying O1–O10/KC}\}.$$

The bridge must show same-run extraction, exact projection, non-anticipation, realized focus-cell choices, HIT/MISS separation, distinct shared-credit evidence, invariant concretization, finite joint terminality, and terminal adequacy transfer (conditions A0–A12 of the companion bridge specification).

For the *stride-family* encodings this bridge is now *proved*, in the strongest per-model form: every satisfying solver interpretation of a schema-conforming encoding is bridge-admissible (Theorem 11.5). The schema is the following positive-form clause family. Let $g = \gcd(a, b)$, $E = lcm(a, b)$, $\alpha = b/g$, and $\beta = a/g$.

No.	**Shape**	**Content**
C1	Init	*Inv* at $i_1 = 0, i_2 = 0$ with $n > 0$, a focused $k < n$, HIT-equal or HIT-unequal focus data, $d = 0$, $d_0 \geq 0$, and $c = 0$.
C2	`TT_search` transition	Body has *Inv*, $i_1 < n$, $i_2 < n$, $i_1 = i_2$, and `SchTT_search`; head advances $i_1' = i_1 + a$, $i_2' = i_2 + b$ and sets either HIT-equal $c' = c, d' = d + 1$ or HIT-unequal $c' = c + 1, d' = d$.
C3	`TT_settled` transition	Body has *Inv*, $i_1 < n$, $i_2 < n$, and `SchTT_settled`; head keeps focus unchanged, advances both sides, and sets $c' = c + 1$.
C4	`TF` transition	Body has *Inv*, $i_1 < n$, and `SchTF`; head keeps focus unchanged, advances $i_1' = i_1 + a$, and sets $c' = c + 1$.
C5	`FT` transition	Body has *Inv*, $i_2 < n$, and `SchFT`; head keeps focus unchanged, advances $i_2' = i_2 + b$, and sets $c' = c$.
C6–C8	Fairness	Progress, run1, and run2 disjunctive clauses: from *Inv* and the activity guard, at least one corresponding scheduler head holds.
C9	`TT_search` forcing	`SchTT_search` follows from *Inv*, $i_1 < n$, $i_2 < n$, $i_1 = i_2$, and $d < d_0$.
C10	`TT_settled` forcing	`SchTT_settled` follows from *Inv* and $d \geq d_0$.
C11	`TF` realignment	`SchTF` follows from *Inv*, $i_1 < i_2$, $i_1 < n$, and $d < d_0$.
C11'	`FT` realignment	`SchFT` follows from *Inv*, $i_2 < i_1$, $i_2 < n$, and $d < d_0$; required iff $\beta \geq 2$, while for $\beta = 1$ the case is unreachable.
C12	`TF` tail forcing	`SchTF` follows from *Inv*, $i_1 < n$, and $n \leq i_2$; required for termination progress.
C12'	`FT` tail forcing	`SchFT` follows from *Inv*, $i_2 < n$, and $n \leq i_1$; required for termination progress.
C13	Guarded goal	$a(c + d_0) \leq n + a - 1$ follows from *Inv*, $n \leq i_1$, $n \leq i_2$, and $d \geq d_0$; equivalently $c \leq \lceil n/a \rceil - d_0$.

The covered set is the canonical `1v2` corpus encoding (which carries the termination-progress clauses) and the five bridge-grade variants (`popl/encodings/L4/bridge/`), each re-verified sat by PCSAT; the five frozen original variants lack the termination-progress clauses and remain corroboration only. Positive-form goals only: the violation-form reading is excluded.

This placement is the answer to Hiroshi's cell-morphing caveat, now formal for the stride family. Cell morphing is an over-approximation, while scheduler synthesis and focus-cell/evidence choices have an existential flavor. Therefore the caveat belongs at the encoding-to-logic boundary. It is not an extra O11 premise of the concrete kernel theorem, because the kernel theorem itself is stated over concrete coupled executions. A future ARWP or μCLP implementation can use the same bridge interface if it exposes the same extracted witness data, but the current corpus remains pfwCSP/PCSAT. The bridge for encoding classes beyond the stride-family schema remains open.

Theorem 11.5 (Encoded-certificate bridge; stride family; arbitrary models)**.** *Fix strides $1 \le a < b$ and a stride-family encoding $E_{a,b}$ conforming to the clause schema of the companion artifact: initialization, the four mode transitions, the disjunctive fairness clauses, the search/settled/realignment forcing clauses (the backward, FT-side realignment clause being required exactly when $\beta = a/\gcd(a,b) \ge 2$; for $\beta = 1$ that case is provably unreachable), the two termination-progress clauses, and the guarded positive-form goal in the integer form $a(c + d_0) \le n + a - 1$, equivalently $c \le \lceil n/a \rceil - d_0$. Let M be* any *interpretation of the unknowns (Inv, SchTT_search, SchTT_settled, SchTF, SchFT) satisfying every clause of $E_{a,b}$. Then the extraction Extract(M) is a concrete Level 4 scheduler certificate satisfying O1–O10/KC for every finite terminating P_{stride} pair, and by Theorem 11.4,*

$$\models_\nu \{P_{\text{stride}}\}\ \texttt{while}\ b_1\ \texttt{do}\ C_1 \sim \texttt{while}\ b_2\ \texttt{do}\ C_2\ \{Q \wedge m_1 - m_2 \le \lceil n/a \rceil - d_0\} \mid \iota.$$

Proof (companion artifact `popl/L4_ENCODED_BRIDGE_SOUNDNESS_2026-06-10.md`). The schedule is the model-independent merge policy $\sigma_{\alpha,\beta}$; the forcing and termination-progress clauses make each of its choices true in M along the run, and for $\beta = 1$ instances the backward-realignment case is unreachable (the artifact's Lemma B0, a divisibility argument on the pointer gap). The certificate phases are the cell-morphed abstract states, *next* is the shadow update reading focused cells from the concrete arrays, and the certificate invariant is the three-part conjunction: $M \models Inv$ at the phase, the configuration *realizes* the phase (scalars match; focused values are the concrete cells; the credit count equals the ghost-set size), and the concrete cost ledger is bounded by the encoded abstract charge. A joint induction (Lemma B1) establishes model membership, enabledness, and realization along the produced trace; a five-case domination lemma (B2) bounds the ledger by the charge; the concrete credit-fulfillment argument (B3: the exhaustive scan plus the P_{stride} agreement witness) discharges the goal clause's $d \ge d_0$ guard, so the solver is never asked to know where agreements lie. The terminal verification condition is proved for *arbitrary* terminal candidate traces satisfying the invariant (Lemma B4, applying the encoded goal only at

realized terminal shadow states), and the produced-trace bound is its corollary (B4c). The admissibility conditions A0–A12 are thereby established for every satisfying model. □

11.9 Status, Artifacts, and Open Items

The reader-facing status is:

Result	Status	Full proof pointer
AWhile-Product soundness theorem	Proved, Theorem 11.4	`popl/L4_AWHILE_PRODUCT_RULE_2026-06-09.md`
1v2 stride instance	Proved, Theorem 11.1	`popl/L4_1V2_V1_LOCAL_PROOFS_2026-06-09.md`; `popl/L4_BRIDGE_1V2_2026-06-09.md`
2v3 stride instance	Proved, Theorem 11.2	`popl/L4_2V3_SOUNDNESS_2026-06-09.md`; `popl/L4_2V3_DISTINCT_CREDIT_LEMMA_2026-06-09.md`
General stride corollary	Proved, Theorem 11.3	`popl/L4_GENERAL_STRIDE_SOUNDNESS_2026-06-09.md`
Stride-family encoded bridge	Proved, Theorem 11.5	`popl/L4_ENCODED_BRIDGE_SOUNDNESS_2026-06-10.md`

The kernel and bridge specifications remain useful audit references: `popl/L4_KERNEL_SPEC_2026-06-07.md` and `popl/L4_BRIDGE_THEOREM_2026-06-07.md`. Phase-0 mutation and reachability checks remain diagnostic evidence only; they do not discharge the kernel obligations. The verbose per-mode encoding schemas, lemma inventory, realized 1v2 walkthroughs, and proof checklist live in the companion artifacts and git history rather than in this section.

The present Level 4 result is semantic: concrete certificates satisfying the kernel obligations imply the relational validity judgment $\models_\nu$, and the stride instances above are branch-counting theorems for read-only single-pass stride scans. The stride-family bridge additionally shows that schema-conforming PCSAT witnesses can be concretized into such certificates.

The open items are deliberately narrower than those proved results. Raw execution-cost variants still require the structural-offset reconciliation before they should be claimed as theorems. A true syntactic Level 4 logic would need a derivability judgment $\vdash_{\mathrm{L4}}$ and a separate soundness theorem $\vdash_{\mathrm{L4}} \Rightarrow \models_\nu$; this section proves the semantic principle only. The encoding-to-logic bridge beyond the stride-family schema also remains open, and nested loops or arbitrary control-location pairs are left to a transition-system formulation. No completeness or converse theorem is claimed, no solver model is claimed sound outside the stated bridge theorem, and the current automation interface is pfwCSP/PCSAT rather than ARWP or μCLP.

References

[1] Ezgi Çiçek, Gilles Barthe, Marco Gaboardi, Deepak Garg, and Jan Hoffmann. Relational Cost Analysis. In *Principles of Programming Languages (POPL)*. ACM, 2017.

[2] Weihao Qu, Marco Gaboardi, and Deepak Garg. Relational Cost Analysis for Functional-Imperative Programs. *Proc. ACM Program. Lang.*, 3(ICFP), Article 92, pp. 1–29, 2019. DOI: 10.1145/3341696.